Suk-Hwan Suh
Seong-Kyoon Kang
Dae-Hyuk Chung
Ian Stroud

Springer Series in Advanced Manufacturing

Theory and Design of CNC Systems

Springer

Springer Series in Advanced Manufacturing

Series Editor

Professor D.T. Pham
Manufacturing Engineering Centre
Cardiff University
Queen's Building
Newport Road
Cardiff CF24 3AA
UK

Other titles in this series

Assembly Line Design
B. Rekiek and A. Delchambre

Advances in Design
H.A. ElMaraghy and W.H. ElMaraghy (Eds.)

Effective Resource Management in Manufacturing Systems: Optimization Algorithms in Production Planning
M. Caramia and P. Dell'Olmo

Condition Monitoring and Control for Intelligent Manufacturing
L. Wang and R.X. Gao (Eds.)

Optimal Production Planning for PCB Assembly
W. Ho and P. Ji

Trends in Supply Chain Design and Management: Technologies and Methodologies
H. Jung, F.F. Chen and B. Jeong (Eds.)

Process Planning and Scheduling for Distributed Manufacturing
L. Wang and W. Shen (Eds.)

Collaborative Product Design and Manufacturing Methodologies and Applications
W.D. Li, S.K. Ong, A.Y.C. Nee and C. McMahon (Eds.)

Decision Making in the Manufacturing Environment
R. Venkata Rao

Frontiers in Computing Technologies for Manufacturing Applications
Y. Shimizu, Z. Zhang and R. Batres

Reverse Engineering: An Industrial Perspective
V. Raja and K.J. Fernandes (Eds.)

Automated Nanohandling by Microrobots
S. Fatikow

A Distributed Coordination Approach to Reconfigurable Process Control
N.N. Chokshi and D.C. McFarlane

ERP Systems and Organisational Change
B. Grabot, A. Mayère and I. Bazet (Eds.)

Machining Dynamics
K. Cheng (Ed.)

ANEMONA
V. Botti and A. Giret

Suk-Hwan Suh • Seong-Kyoon Kang
Dae-Hyuk Chung • Ian Stroud

Theory and Design of CNC Systems

Springer

Suk-Hwan Suh, PhD
School of Mechanical & Industrial Engineering
POSTECH, San 31, Pohang, 790-784
Republic of Korea

Seong-Kyoon Kang, PhD
K&S International Patent and Law Firm
3F, Hanjin Bldg., 607-12 Yeoksam-dong,
Kangnam-gu, Seoul, 135-907
Republic of Korea

Dae-Hyuk Chung, PhD
Doosan Infracore Co., Ltd.
601-3, Namsan-dong, Changwon-Si,
Gyeongnam-Do
Republic of Korea

Ian Stroud, PhD
École Polytechnique Fédérale de Lausanne (EPFL)
STI-IGM-LICP, Station 9,
1015 Lausanne
Switzerland

ISBN 978-1-84800-335-4 e-ISBN 978-1-84800-336-1

DOI 10.1007/978-1-84800-336-1

Springer Series in Advanced Manufacturing ISSN 1860-5168

British Library Cataloguing in Publication Data
Theory and design of CNC systems. - (Springer series in advanced manufacturing)
1. Machine-tools - Numerical control 2. Machine-tools - Numerical control - Programming
I. Suh, Suk-Hwan
621.9'023'0285
ISBN-13: 9781848003354

Library of Congress Control Number: 2008928587

Cover design: eStudio Calamar S.L., Girona, Spain

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

This book is dedicated to:

Eun-Sook Choi,
Hyeon-Jeong Lee,
Hye-Jung Kim,
and Hildegarde Nagy-Stroud

and to the rest of our families
for their endurance of this headlong task.

Preface

CNC controllers, working as a brain for manufacturing automation, are high value-added products accounting for over 30% of the price of machine tools. CNC technology is generally considered as a measure of the level of manufacturing technology of a nation, and is currently led by major advanced countries such as USA, Japan, and Germany. CNC technology, which cannot be developed with one single technology but needs to integrate computer technology, hardware technology, machining technology, and so on, is often referred to as "The Flower of Industrial Technology", and requires a strategic long-term support, mostly on a governmental level.

Despite its significant role, textbooks on CNC controllers are quite rare worldwide, with a few published in the 1970s and some later. However, the earlier ones mostly deal with conventional technologies, while the later ones deal with fragmental contents, mostly focusing on part programming and machine operation. This book is written by several authors in collaboration who have long experience in CNC development, education, and research, and is designed as a highly focused textbook to provide knowledge on the principles and development technologies of CNC controllers. Therefore, this book can be used as a main textbook for courses related to CNC in such departments as mechanical engineering, precision engineering and control engineering, and as a guide for those working on CNC development in industry. If highly descriptive portions are taken out, it can also be used as lecture material in technical colleges.

The framework of industrial CNC controllers has been established by integrating the structure and element technologies of CNC controllers under research and development by the authors in their respective field of industry and academia over the years. Furthermore, this book intends to encourage the spirit of development by introducing actual realization cases.

This book is composed of two parts with a total of 11 chapters: Part I is composed of Chapters 1–6 on the principle and design of CNC, and Part II is composed of an open-architectural soft CNC system. Specifically, Chapter 1 provides general concepts and mechanisms of numerically controlled machines, while Chapters 2 through 5 cover the element technologies of NCK in charge of controlling the transfer axis, including interpreter, interpolator, control of acceleration and deceleration, and po-

sition control system. In Chapter 6, NCK development cases are described together with source code. Therefore, those who are interested in motion controllers can develop independent control devices by referring to the contents of Chapters 2 through 6.

Part II describes the open-architectural soft CNC system, including the principles of major modules of numerically controlled machines, except the NCK (dealt with in Part 1), and the system design process for the composition of the overall system from the perspective of open-architectural soft CNC systems. Specifically, Chapter 7 explains the PLC, controlling most mechanical motions except the transfer axis, while Chapter 8 presents the principles of the Man-Machine Interface (MMI) and the major modules for the development of conversational programming methods. Real-time operation concepts and methods necessary for designing real-time controllers are described in Chapter 9, Chapter 10 describes the architecture design of CNC systems based on personal computers. This is discussed from the perspective of soft CNC, including several approaches to the architecture of open-style CNC system with free external interfaces, and the design process of those approaches. The concept and primary elements of STEP-NC are introduced in Chapter 11, which has recently come under the spotlight as a method of realizing intelligent CNC machines. Therefore, those who are interested in designing and realizing open-style soft CNC devices can refer to the topics covered in Chapters 7 through 11 to materialize intelligent open-style NC devices.

As authors of this book, we recommend that instructors have their students actually code the NCK technologies (Chapters 2 through 5), which are the core elements, and finish a computer simulation system, one similar to the development case covered in Chapter 6, and verify the performance. One step further, if the interface board (encoder signal and PLC signal processing) and the *XY*-table can actually be connected by the students, the effect of learning can be doubled.

Those students who want to learn the general technologies related with CNC systems can achieve their goals by studying the PLC, conversational programming system, particularly actual cases of system programming methods to realize soft CNC, as covered in Part 2, Chapters 7 through 11.

To complete this book it took over three years to collect and organize all sorts of material accumulated over a period of many years, including technical papers and patent data materials. However, we feel there are many shortcomings. Some of the excuses we can offer could include the fact that CNC technology has been developed by industry itself and that each element technology derives from a completely different domain of knowledge. Therefore, for integrating them under the umbrella of CNC for academic purposes, many problems are posed such as un- or mis-defined technical terminologies and lack of systematic knowledge bases. However, despite this, the authors decided to publish this book in the hope that it will contribute to the advancement of CNC technology both at home and abroad, in consideration of the sheer reality that no proper textbooks are available for education or training in CNC technology. With lots of input from the readers, we hope this book can improve its contents in the future.

This book was originally published in Korean and has now been translated into English. We would like to take this opportunity to express our appreciation to Ms. Eunsook Choi, who encouraged preparation of the English version of the original Korean text book, Mr. Suho Jung and students of the Center for ubiquitous manufacturing at POSTECH for help in the editing, and Springer who willingly accepted publication of it.

We would also like to express our appreciation to Dae-Jung Seong of Doosan Infracore in charge of CNC development for providing contemporary industrial perspectives.

Postech,
March 2008

Suk-Hwan Suh, Seong-Kyoon Kang,
Dae-Hyuk Chung, Ian Stroud

Contents

Abbreviations xvii

Part I Principles and NCK Design of CNC Systems

1 Introduction to NC Systems 3
1.1 Introduction 3
1.2 The History of NC and NC Machine Tools 6
1.3 CNC Driving System Components 8
1.3.1 Driving Motor and Sensor 9
1.3.2 Linear Movement Guide 15
1.3.3 Coupling 16
1.4 CNC Control Loop 17
1.4.1 Semi-closed Loop 18
1.4.2 Closed Loop 18
1.4.3 Hybrid Loop 19
1.4.4 Open Loop 19
1.5 The Components of the CNC system 19
1.5.1 MMI Function 22
1.5.2 NCK Function 23
1.5.3 PLC Function 25
1.5.4 Real-time Control System 28
1.6 The Progress Direction of the CNC System 29
1.7 Summary 31

2 Interpreter 33
2.1 Introduction 33
2.2 Part Program 34
2.2.1 Program Structure 35
2.2.2 Main Programs and Subprograms 39
2.3 Main CNC System Functions 40
2.3.1 Coordinate Systems 40

2.3.2 Interpolation Functions 42
2.3.3 Feed Function 48
2.3.4 Tools and Tool Functions 50
2.3.5 Spindle Functions 53
2.3.6 Fixed-cycle Function 53
2.3.7 Skip Function 56
2.3.8 Program Verification 56
2.3.9 Advanced Functions 57
2.4 G&M-code Interpreter 62
2.5 Summary 66

3 Interpolator 69
3.1 Introduction 69
3.2 Hardware Interpolator 70
3.2.1 Hardware Interpolation DDA 71
3.2.2 DDA Interpolation 73
3.3 Software Interpolator 75
3.3.1 Software Interpolation Methods 78
3.3.2 Sampled-Data Interpolation 86
3.4 Fine Interpolation 96
3.5 NURBS Interpolation 98
3.5.1 NURBS Equation Form 99
3.5.2 NURBS Geometric Characteristics 100
3.5.3 NURBS Interpolation Algorithm 101
3.6 Summary 106

4 Acceleration and Deceleration 107
4.1 Introduction 107
4.2 Acc/Dec Control After Interpolation 108
4.2.1 Acc/Dec Control by Digital Filter 109
4.2.2 Acc/Dec Control by Digital Circuit 112
4.2.3 Acc/Dec Control Machining Errors 121
4.2.4 Block Overlap in ADCAI 126
4.3 Acc/Dec Control Before Interpolation 128
4.3.1 Speed-profile Generation 129
4.3.2 Block Overlap Control 132
4.3.3 Corner Speed of Two Blocks Connected by an Acute Angle 142
4.3.4 Corner Speed Considering Speed Difference of Each Axis . . 144
4.4 Look Ahead 145
4.4.1 Look-Ahead Algorithm 147
4.4.2 Simulation Results 152
4.5 Summary 155

5 PID Control System 157
5.1 Introduction 157
5.2 The Servo Controller 158
5.3 Servo Control for Positioning 160
5.4 Position Control 161
5.4.1 PID Controller 162
5.4.2 PID Gain Tuning 166
5.4.3 Feedforward Control 171
5.5 Analysis of the Following Error 179
5.5.1 The Following Error of the Feedback Controller 179
5.5.2 The Following Error of the Feedforward Controller 182
5.5.3 Comparison of Following Errors 183
5.6 Summary 185

6 Numerical Control Kernel 187
6.1 Introduction 187
6.2 Architecture of ACDAI-type NCK 187
6.2.1 Implementation of the Interpolator 188
6.2.2 Implementation of the Rough Interpolator 193
6.2.3 Implementation of an Acc/Dec Controller 199
6.2.4 Implementation of Fine Interpolator 203
6.2.5 Implementation of the Position Controller 208
6.3 Architecture of an ADCBI-type NCK 211
6.3.1 Implementation of the Look-Ahead Module 213
6.3.2 Implementation of an Acc/Dec Controller 215
6.3.3 Implementation of the Rough Interpolator 222
6.3.4 The Mapping Module 225
6.4 Summary 226

Part II Open-architectural Soft CNC Systems

7 Programmable Logic Control 229
7.1 Introduction 229
7.2 PLC Elements 230
7.3 PLC Programming 234
7.4 Machine Tool PLC Programming 235
7.5 PLC System Functions 240
7.5.1 Software Model and Communication Model 242
7.5.2 Programming Model 244
7.5.3 User Programming Languages 245
7.6 Soft PLC 247
7.7 PLC Configuration Elements 248
7.7.1 PLC System Functions 249
7.7.2 Executor Programming Sequence 253
7.7.3 Executor Implementation Example 254

7.8 Summary 268

8 Man–Machine Interface 271
8.1 MMI Function 271
8.1.1 Area for Status Display 271
8.1.2 Area for Data Input 273
8.1.3 Area for MPG Handling 273
8.1.4 Area for Machine Operation 273
8.2 Structure of the MMI System 275
8.3 CNC Programming 278
8.3.1 The Sequence of Part Programming 278
8.3.2 Manual Part Programming 279
8.3.3 Automatic Part Programming 280
8.4 Mazatrol Conversational System 289
8.4.1 Turning Conversational System 289
8.4.2 Programming Procedure 292
8.5 Conversational Programming System Design 294
8.5.1 Main Sequence for Design 294
8.5.2 Key Design Factors 296
8.6 Development of the Machining Cycle 305
8.6.1 Turning Fixed Cycle 305
8.6.2 Turning Cycle for Arbitrary Shape 306
8.6.3 Corner Machining Cycle 310
8.6.4 Drilling Sequence 312
8.7 Summary 314

9 CNC Architecture Design 315
9.1 Introduction 315
9.2 Operating Systems 317
9.3 Real-time Programming 319
9.4 Structure of a Real-time OS 321
9.5 Process Management 323
9.5.1 Process Creation and Termination 324
9.5.2 Process State Transition 324
9.5.3 Process Scheduling 325
9.6 Process Synchronization 330
9.6.1 Semaphores 330
9.6.2 Using Semaphores 331
9.6.3 Events and Signals 331
9.7 Resources 334
9.7.1 System Resources 334
9.7.2 Mutual Exclusion 335
9.7.3 Deadlock 336
9.8 Inter-process Communication 337
9.8.1 Shared Memory 337

9.8.2 Message System 338
9.9 Key Performance Indices 340
9.9.1 Task Switching Time 340
9.9.2 Context Switching Time 341
9.9.3 Semaphore Shuffling Time 341
9.9.4 Task Dispatch Latency Time 341
9.10 Hardware and Operating Systems 344
9.10.1 Architecture of Multi-processing Hardware 344
9.10.2 Operating System Configuration 347
9.10.3 CNC System Architecture 348
9.11 Summary 350

10 Design of PC-NC and Open CNC 353
10.1 Introduction 353
10.2 Design of Software Architecture 356
10.2.1 CNC System Modeling 356
10.3 Design of Soft-NC System 359
10.3.1 Design of Task Module 359
10.3.2 Design of the System Kernel 361
10.3.3 PLC Program Scanning and Scheduling 362
10.3.4 Task Synchronization Mechanism 365
10.3.5 Inter-Task Communication 369
10.4 Motion Control System Programming Example 376
10.4.1 Design of System Architecture 377
10.4.2 Creating Tasks 378
10.4.3 Task Synchronization 378
10.4.4 Task Priority 381
10.4.5 Inter-Task Communication 381
10.4.6 Create Event Service 384
10.5 Open-CNC Systems 387
10.5.1 Closed-type CNC Systems 387
10.5.2 Open CNC Systems 389
10.6 Summary 393

11 STEP-NC System 395
11.1 Introduction 395
11.2 Background of STEP-NC 397
11.2.1 Problems with G&M Codes 397
11.2.2 Historical Background 398
11.3 STEP-NC: A New CNC Interface Based on STEP 399
11.3.1 Contents 399
11.3.2 Relationship Between STEP and STEP-NC 399
11.3.3 Objectives and Impacts 401
11.4 STEP-NC Data Model 402
11.4.1 Part 1: Overview and Fundamental Principles 403

11.4.2 Part 10: General Process Data 405
11.4.3 Part 11: Process Data for Milling 407
11.4.4 Part 12: Process Data for Turning 407
11.4.5 Tools for Milling and Turning 408
11.5 Part Programming 410
11.5.1 Part Programming for the Milling Operation 411
11.5.2 Part Programming for the Turning Operation 414
11.6 STEP-CNC System 415
11.6.1 Types of STEP-CNC 417
11.6.2 Intelligent STEP-CNC Systems 418
11.7 Worldwide Research and Development 422
11.7.1 WZL-Aachen University (Germany) 422
11.7.2 ISW-University of Stuttgart (Germany) 424
11.7.3 POSTECH (South Korea) 425
11.7.4 Ecole Polytechnic Fédérale of Lausanne (Switzerland) 426
11.7.5 University of Bath (UK) 427
11.7.6 NIST (USA) 427
11.8 Future Prospects 428

A Turning and Milling G-code System 431
A.1 Turning 431
A.2 Milling 434
A.3 Classification of G-code Groups 437

Bibliography 439

Index 447

Abbreviations

AAM – Application Activity Model
AC – Alternating Current
Acc/Dec – Acceleration and Deceleration
ACS – Autonomous Control System
ADCAI – Acc/Dec Control After Interpolation
ADCBI – Acc/Dec Control Before Interpolation
AGV – Autonomous Guided Vehicle
AIM – Application Interpreted Model
AP – Application Protocol
API – Application Programming Interface
APT – Automatically Programmed Tool
ARM – Application Reference Model
ASCII – American Standard Code for Information Interchange

BCD – Binary Coded Decimal
BLU – Basic Length Unit

CAD – Computer-Aided Design
CAI – Computer-Aided Inspection
CAM – Computer-Aided Manufacturing
CAPP – Computer-Aided Process Planning
CAPS – Conversational Automatic Programming System
CCW – Counter Clock Wise
CD – Committee Draft
CES – Code Editing System
CGS – Code Generating System
CMM – Coordinate Measurement Machine
CNC – Computerized Numerical Control
CORBA – Common Object Request Broker Architecture
CPU – Central Processing Unit

CW – Clock Wise

D – Derivative, as in Derivative Control
D/A – Digital to Analog
DA-BA-SA – Design-Anywhere, Build-Anywhere, Support-Anywhere
DB – DataBase
DC – Direct Current
DDA – Digital Differential Analyzer
DNC – Direct Numerical Control
DPM – Dual Port Memory
DPR – Dual Port RAM
DRV – Drives
DRV – DRiVe
DSP – Digital Signal Processing

EDM – Electrical Discharge Machining
EH – chord Height Error
EIA – Electronic Industries Association
EISA – Extended Industry Standard Architecture
EOB – End Of Block
ER – Radial Error

FA – Flexible Automation
FBD – Function Block Diagram
FDIS – Final Draft International Standard
FIFO – First In, First Out
FIR – Finite Impulse Response
FMS – Flexible Manufacturing System
FPLC – Fast PLC
F/V – Frequency to Voltage

GPMC – General Purpose Motion Control
GUI – Graphical User Interface

HAL – Hardware Abstract Layer
HMI – Human Machine Interface
H/W – Hardware

I – Integral, as in Integral Control
ICS – Information Contents and Semantics
IEC – International Electrotechnical Commission
IKF – Inverse Compensation Filter
IL – Instruction List

IMS – Intelligent Manufacturing Systems
IO – Interrupt Overhead
IPC – Inter Process Communication
IPO – InterPOlation
IPR – InterPReter
IS – International Standard
ISA – Industry Standard Architecture
ISO – International Organization for Standardization
ISR – Interrupt Service Routine

LD – Ladder Diagram
LED – Light Emitting Diode
LM – Linear Movement
LSI – Large Scale Integrated Circuit

MDI – Multiple Document Interface
MES – Manufacturing Execution System
MMC – Man Machine Control
MMI – Man Machine Interface
MPG – Manual Pulse Generator
MRR – Material Removal Rate
MTB – Machine Tool Builder

NC – Numerical Control
NCK – Numerical Control Kernel
NPLC – Normal PLC
NURBS – Non Uniform Rational B-Spline
NWIP – New Work Item Proposal

OAC – Open Architecture Controller
OMM – On Machine Measurement
OS – Operating System
OSI – Open Standard Interface
OT – Over Travel

P – Proportional, as in Proportional Control
PC – Personal Computer
PCI – Peripheral Component Interconnect
PID – Proportional Integral Derivative
PLC – Programmable Logic Control
PMSMs – Permanent Magnet Synchronous Motors
POS – POSition

RAM – Random Access Memory

RM – Rate Monotonic
RMS – Rate Monotonic Scheduling
ROM – Read Only Memory
RPM – Revolutions Per Minute
RS – Recommended Standard
RTOS – Real Time Operating System
RTX – RealTime eXtension

SC – Sub Committee
SERCOS – SErial Realtime COmmunication System
SFC – Sequential Function Chart
SFP – Shop Floor Programming
SISO – Single Input Single Output
SOP – Shop floor Oriented Programming
ST – Structured Text
STEP – STandard for the Exchange of Product model data
S/W – Software

TC – Technical Committee
TPG – Tool Path Generation

VME – Virtual Machine Environment

WD – Working Draft
WOP – Workshop Oriented Programming

XML – eXtensible Markup Language

YACC – Yet Another Compiler Compiler

ZPETC – Zero Phase Error Tracking Control

Part I
Principles and NCK Design of CNC Systems

Chapter 1
Introduction to NC Systems

NC machines, being typical mechatronics products, comprise machine tools that have a mechanical component and a numerical control system that is an electrical component. In this chapter, the history, the constituent units, the functions, and future directions of NC systems, being the intelligence of NC machines, will be addressed. Through studying this chapter, you will obtain a comprehensive understanding and fundamental knowledge about NC systems.

1.1 Introduction

The machine tool is called "mother machine" in the sense that it is a machine that makes machines. In particular, as machine tools have advanced from manual machine tools to NC machines, these have become perfect in the role of mother machines with the improvement of accuracy and machining speed.

NC machine tools can be classified as "cutting machines" and "non-cutting machines". A cutting machine means a machine that performs a removal process to make a finished part; milling machines, turning machines and EDM machines being good examples. Non-cutting machine tools change the shape of the blank material by applying force and press machines are good examples of this. In addition, robot systems (Fig. 1.1a) for welding, cutting, and painting can be included in a broad sense.

When NC machines were developed, the purpose of the NC machine was to machine parts with complex shape in a precise manner. Therefore, the numerical controller was primarily applied to milling machines (Fig. 1.1b) and boring machines. However, recently it has become popular to apply NC for increased productivity and the kinds of machine with NC have been varied to include machines such as turning machines (Fig. 1.1c), machining centers (Fig. 1.1d), and drill/tapping machines. Particularly, the application of NC has extended to non-conventional machine tools such as wire electro-discharge machines (Fig. 1.1e) and laser cutting machines in addition to conventional metal-cutting machines.

Also, as factory automation has progressed, NC machine technology has also progressed to allow construction of Flexible Automation (FA) or Flexible Manufacturing Systems (FMS) (Fig. 1.1f) by connecting machines with production equipment such as robots, Autonomous Guided Vehicles (AGV), automated warehouses and computers. NC systems are used not only for machine tools but also all machines that need motion controlled by servo systems, such as cutting machines, drawing instruments, woodworking machines, Coordinate Measurement Machines (CMM) and embroidering machines and NC is the fundamental technology for factory automation.

The task flow that is needed for producing a part using an NC machine can be summarized as Fig. 1.2. The tasks can be classified as the following three types:

1. Offline tasks: CAD, CAPP, CAM
2. Online tasks: NC machining, monitoring and On-Machine Measurement.
3. Post-line tasks: Computer-Aided Inspection (CAI), post-operation

Offline tasks are the tasks that are needed to generate a part program for controlling an NC machine. In the offline stage, after the shape of a part has been decided, a geometry model of this part is created by 2D or 3D CAD. In general, CAD means Computer Aided Design but CAD in this book is regarded as a modeling stage in which both design and analysis are included because engineering analysis of a part cannot be carried out on the shopfloor.

After finishing geometric modeling, Computer Aided Process Planning, CAPP, is carried out where necessary information for machining is generated. In this stage, the selection of machine tools, tools, jig and fixture, decisions about cutting conditions, scheduling and machining sequences are created. Because process planning is very complicated and CAPP is immature with respect to technology, process planning generally depends on the know-how of a process planner.

CAM (Computer Aided Manufacturing) is executed in the final stage for generating a part program. In this stage, tool paths are generated based on geometry information from CAD and machining information from CAPP. During tool path generation, interferences between tool and workpiece, minimization of machining time and tool change, and machine performance are considered. In particular, CAM is an essential tool to generate 2.5D or 3D toolpaths for machine tools with more than three axes.

Online tasks are those that are needed to machine parts using NC machines. A part program, being the machine-understandable instructions, can be generated in the above-mentioned offline stage and part programs for a simple part can be directly edited in NC by the user. In this stage, the NC system reads and interprets part programs from memory and controls the movement of axes. The NC system generates instructions for position and velocity control based on the part program and servo motors are controlled based on the instructions generated. As the rotation of a servo motor is transformed into linear movement via ball-screw mechanisms, the workpiece or tool is moved and, finally, the part is machined by these movements.

To increase the machining accuracy, not only the accuracy of the servo motor, table guide, ball screw and spindle but also the rigidity of the machine construc-

Fig. 1.1 Types of NC machine (a) Robot, (b) Milling Machine (c) Turning machine (d) Machining Center (e) Wire EDM (f) FMS Line

tion should be high. The construction of the machine and the machine components should also be designed to be insensitive to vibration and temperature. In addition, the performance of the encoder and sensors that are included in the NC system and the control mechanism influences the machining accuracy. The control mechanism will be addressed in more detail in the following section.

In the online stage, the status of the machine and machining process may be monitored during machining. Actually, tool-breakage detection, compensation of thermal deformation, adaptive control, and compensation of tool deflection based on monitoring of cutting force, heat, and electric current are applied during machining. On-Machine Measurement is also used to calculate machining error by inspecting the finished part on the machine, returning machining errors to NC to carry out compensation.

The post-line task is to carry out CAI (Computer Aided Inspection), inspecting the finished part. In this stage, inspection using a CMM (Coordinate Measurement Machine) is used to make a comparison between the result and the geometry model in order to perform compensation. The compensation is executed by modifying tool compensation or by doing post-operations such as re-machining and grinding. Reverse engineering, meaning that the shape of the part is measured and a geometric model based on the measured data is generated, is included in this stage.

As mentioned above, through three stages, it is possible for machine tools not only to satisfy high accuracy and productivity but also to machine parts with complex shape as well as simple shapes. Because NC machines can machine a variety of parts by changing the part program and repetitively machine the same part shape by storing part programs, NC machines can be used for general purposes.

In this book, the functionalities and the components of NC in the online stage will mainly be addressed. However, considering that part of the CAM function has recently been included in the online stage, WOP (Workshop Oriented Programming) or SFP (Shop Floor Programming), which are types of online CAM systems, will be described in detail.

1.2 The History of NC and NC Machine Tools

As mentioned in the previous section, the NC is the system that enables machine tools to machine parts with various shapes rapidly and precisely. In NC, the servo motor is used for controlling the machine tool according to the operation of a user and a servo motor drive mechanism for activating the servo motor. That is, NC means a control device that machines a target part by activating the servo motor according to commands. The NC combined with computer technology is called computerized NC or CNC (Computer Numerical Control). An NC machine which consists of vacuum tubes, transistors, circuits, logic elements such as large-scale integrated circuits (LSI) is called “Hardwired NC”, and performs NC functions through connecting elements by electrical wiring. Instead of elements and circuits, NC functions are implemented based on software in CNC. That is, this change from hardwired NC to CNC was

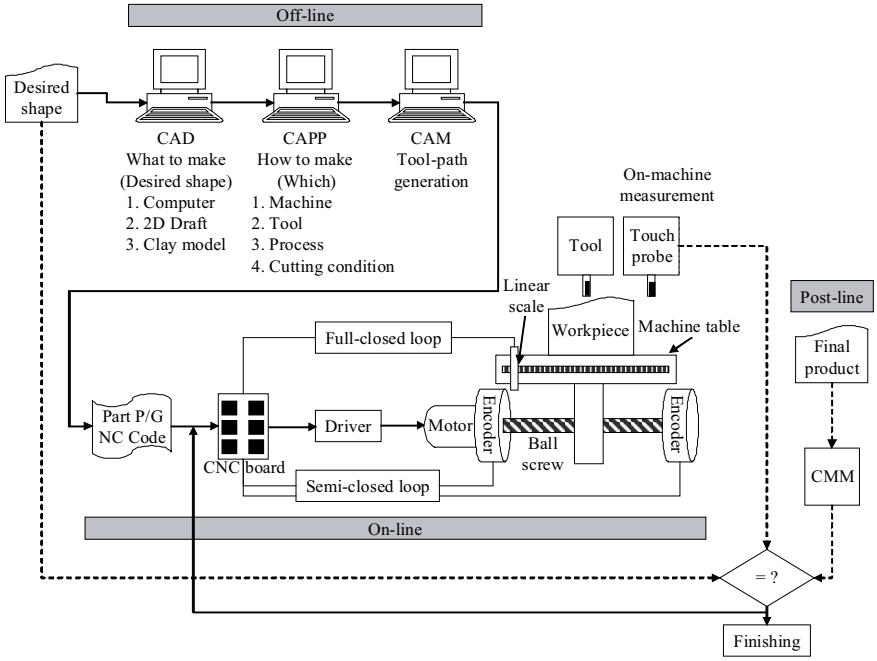


Fig. 1.2 The architecture of NC machine tools and machining operation flow

driven by the advance in capacity and availability of microprocessors and memory. Such CNC is called "Softwired NC".

Through observing the advancement of NC, the fact that NC has the same development history as its components can be seen. In the beginning, the pulse division circuit was made from the computer with tens of thousands of vacuum tubes and the machine tool was controlled by activating an oil-pressure motor and controlling a relay according to the result of logical processing. However, as semiconductors appeared and were applied to NC during the 1960s, electrical motors and power elements during the 1970s and PC components during the 1980s, so Hardwired NC evolved into a Softwired NC machine based on micro processors, electric power and electronic technology, and software technology.

Now, NC and CNC mean Numerical Controller and there is no difference between them. Therefore, NC machine means a machine tool with a CNC system.

It is known that the general-purpose manual machine tool was introduced after the steam engine was developed in the late 18th century. Thereafter, Jacquard invented the method of automatic control of the weaving of fabrics with a loom machine by using punch cards and this method was the beginning of the concept of NC. The concept of NC was actually applied to machine tools after World War II and in 1947, the United States Air force and the Parsons company developed the method almost simultaneously for moving two axes by using punch cards including coordinate data to machine aircraft parts. Since then, this technology was transferred to the servo

laboratory in MIT and in March 1952, a 3-axis milling machine, being the first NC machine tool, was developed. In this era, because there was not the circuitry such as transistors and ICs, a vacuum tube was used and the size of NC was bigger than that of the machine tools.

Since then, with the research effort, the use of NC became practical and in the USA, an NC milling machine was put on sale by Giddings & Lewis, Kearney & Tracker, and Pratt & Whitney. The concept of NC, which was introduced in scientific journals from the United States, was also introduced into Japan and, in 1957, an NC turning machine was developed.

1.3 CNC Driving System Components

The systems that transform the commands from NC to machine movements are shown in Fig. 1.3. Figure 1.3a depicts the servo driving mechanism that consists of a servo motor and power transmission device. The servo, the word originates from "servue" in Latin, is the device that carries out faithfully the given command. The command from NC makes the servo motor rotate, the rotation of the servo motor is transmitted to a ball screw via a coupling, the rotation of the ball screw is transformed into linear movement of a nut, and finally the table with the workpiece moves linearly. In summary, the servo driving mechanism controls the velocity and torque of the table via the servo driving device of each axis based on the velocity commands from NC. Recently, PMSMs (Permanent Magnet Synchronous Motors) have been used as servo motors in machine tools.

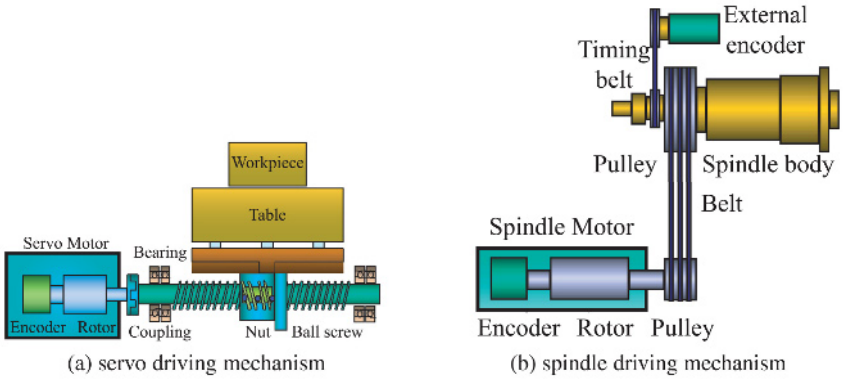


Fig. 1.3 Driving mechanisms of machine tools

Figure 1.3b depicts the spindle unit which consists of a spindle motor and power transmission device. The rotation of the spindle motor is transmitted to the spindle body via a belt and the velocity ratio is dependent on the ratio of pulley sizes between

the motor and spindle body. Recently, induction motors have come into general use as the spindle motors of machine tools because the induction motor, which has no brush, is better than DC motors with respect to size, weight, inertia, efficiency, maximum speed, and maintenance.

Some machine tools use gears to transmit power instead of a belt. However, power transmission by gears is not suitable for high-speed machining. Recently, a direct drive, in which the spindle motor and spindle body (headstock) are directly connected without a power transmission device, has been used for high-speed rotation beyond 10,000 rpm.

1.3.1 Driving Motor and Sensor

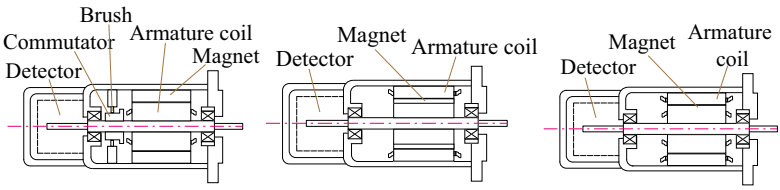
The term "driving motor" is used to mean both the servo motor, which moves the table, and the spindle motor, which rotates the spindle. The spindle is the device that generates adequate cutting speed and torque by rotating the tool or workpiece. Consequently, high torque and high speed are very important for spindle motors and an induction motor is generally used due to the characteristics of the spindle motor. Unlike 3-phase motors, the servo motor needs characteristics such as high torque, high acceleration, and fast response at low speed and can simultaneously control velocity and position. Machine tools, such as turning machines and machining centers, need high torque for heavy cutting in the low-speed range and high speed for rapid movement in the high-speed range. Also, motors with small inertia and high responsibility are needed for machines that frequently repeat tasks whose machining time is very short; for example, punch presses and high-speed tapping machines.

The fundamental characteristics required for servo motors of machine tools are the following:

1. To be able to get adequate output of power according to work load.
2. To be able to respond quickly to an instruction.
3. To have good acceleration and deceleration properties.
4. To have a broad velocity range.
5. To be able to control velocity safely in all velocity ranges.
6. To be able to be continuously operated for a long time
7. To be able to provide frequent acceleration and deceleration.
8. To have high resolution in order to generate adequate torque in the case of a small block.

9. To be easy to rotate and have high rotation accuracy.
10. To generate adequate torque for stopping.
11. To have high reliability and long length of life.
12. To be easy to maintain.

Servo motors are designed to satisfy the above-mentioned characteristics and the term comprises the DC Servo Motor, Synchronous Type AC Servo Motor, and Induction Type AC Servo Motor as shown in Fig. 1.4.



(a) DC Servo Motor (b) Synchronous-type AC Servo Motor (c) Induction-type AC Servo Motor

Fig. 1.4 Types of servo motor

1.3.1.1 DC Servo Motor

The DC Servo Motor is built as shown in Fig. 1.4a.

The stator consists of a cylindrical frame, which plays the role of the passage for magnetic flux and mechanical supporter, and the magnet, which is attached to the inside of the frame. The rotor consists of a shaft and brush. A commutator and a rotor metal supporting frame (rotor core) are attached to the outside of the shaft and an armature is coiled in the rotor metal supporting frame. A brush that supplies current through the commutator is built with the armature coil. At the back of the shaft, a detector for detecting rotation speed is built into the rotor. In general, an optical encoder or tacho-generator is used as a detector.

In the DC servo motor, a controller can be easily designed by using a simple circuit because the torque is directly proportional to the amount of current. The factor that limits the output of the power is the heat from the inside of the motor due to current. Therefore, efficient removal of the heat is essential to generate high torque. The velocity range of DC servo motors is very broad and the price is very low. However, friction with the brushes results in mechanical loss and noise and it is necessary to maintain the brushes.

1.3.1.2 Synchronous-type AC Servo Motor

The stator consists of a cylindrical frame and a stator core. The stator core is located in the frame and an armature coil is wound around the stator core. The end of the coil is connected with a lead wire and current is provided from the lead wire. The rotor consists of a shaft and a permanent magnet and the permanent magnet is attached to the outside of the shaft. In a synchronous-type AC servo motor, the magnet is attached to a rotor and an armature coil is wound around the stator unlike the DC servo motor. Therefore, the supply of current is possible from the outside without a stator and a synchronous-type AC servo motor is called a "brushless servo motor" because of this structural characteristic. Because this structure makes it possible to cool down a stator core directly from the outside, it is possible to resist an increase in temperature. Also, because a synchronous-type AC servo motor does not have the limitation of maximum velocity due to rectification spark, a good characteristic of torque in the high-speed range can be obtained. In addition, because this type of motor has no brush, it can be operated for a long time without maintenance.

Like a DC servo motor, this type of AC servo motor uses an optical encoder or a resolver as a detector of rotation velocity. Also, a ferrite magnet or a rare earth magnet is used for the magnet which is built into the rotor and plays the role of a field system.

In this type of AC Servo Motor, because an armature contribution is linearly proportional to torque, Stop is easy and a dynamic brake works during emergency stop. However, because a permanent magnet is used, the structure is very complex and the detection of position of the rotor is needed. The current from the armature includes high-frequency current and the high-frequency current is the source of torque ripple and vibration.

1.3.1.3 Induction-type AC Servo Motor

The structure of an induction-type AC servo motor is identical with that of a general induction motor. If multi-phase alternating current flows through the coil of a stator, a current is induced in the coil of rotor and the induction current generates torque. In this type of AC servo motor, the stator consists of a frame, a stator core, an armature coil, and lead wire. The rotor consists of a shaft and the rotor core that is built with a conductor.

An induction-type AC servo motor has a simple structure and does not need the detector of relative position between the rotor and stator. However, because the field current should flow continuously during stopping, a loss of heating occurs and dynamic braking is impossible, unlike the AC servo motor.

The strengths, weaknesses and characteristics of the servo motors mentioned above are summarized in Table 1.1.

Table 1.1 Servo-motor summary

	DC Servo Motor	Synchronous Type AC Servo Motor	Induction Type AC Servo Motor
Strengths	Low price Broad velocity range Easy control	Brushless Easy stop	Simple structure No detector needed
Weaknesses	Heat Brush wear Noise Position-detection needed	Complex structure Torque ripple Vibration Position-detection needed	Dynamic braking impossible Loss of heating
Capacity	Small	Small or medium	Medium or large
Sensor	Unnecessary	Encoder, resolver	Unnecessary
Life length	Depends on brush life	Depends on bearing life	Depends on bearing life
High speed	Inadequate	Applicable	Optimized
Resistance	Poor	Good	Good
Permanent magnet	Exists	Exists	None

1.3.1.4 Encoder

The device that detects the current position for position control is called an encoder and, generally, is built into the end of the power-transmission shaft. In order to control velocity, the velocity is detected by a sensor or is calculated by position control data detected from the encoder. The method for detecting velocity uses the encoder, a way of counting pulses generated in unit time and a means of detecting the interval between pulses together.

An encoder can be classified as an optical type or a magnetic type as depicted in Fig. 1.5.

The detection part of a magnetic-type encoder is different from that of an optical-type encoder but the two kinds of encoder generate an output signal in the same manner. Therefore, in this book, only the optical-type encoder will be addressed in detail.

An optical-type encoder can be classified as an incremental type or an absolute type with respect to function.

1. ***Incremental-type encoder***
   Figure 1.6 shows the structure of the incremental-type encoder with three kinds of slit - A, B, and Z; Slits A and B generate an output waveform, the Z slit generates the zero phase. The light emitted from an LED is detected by a photo-detector after passing one slit of the rotation disk and one of the slits A, B, or Z on a fixed slit panel. Slits A and B are arranged for a phase difference of 90 degrees and the electric signal of the output is generated as a square wave whose phase difference

is 90 degrees. The Z slit generates a square wave, indicating one revolution of the encoder.

An incremental-type encoder has a simple structure and is cheap. It is also easy to transmit a signal because the number of wires needed for sending output signals is small. The number of output pulses from the encoder does not indicate the absolute rotation position of a shaft but indicates the rotation angle of the shaft. If we want to know the absolute rotation angle, the number of output pulses should be summed and the rotation angle is calculated based on the number of accumulated pulses. Because the rotation angle is detected continuously, the noise that occurs during signal transmission can be accumulated in a counter. Therefore, some measure for preventing noise should exist as a basic requirement. If the power is off then this type of encoder cannot indicate a position. Because this type of encoder only generates pulses, the number of output pulses should be transformed into an analog signal that is proportional to the pulse frequency in an F/V converter in order to detect rotation velocity.

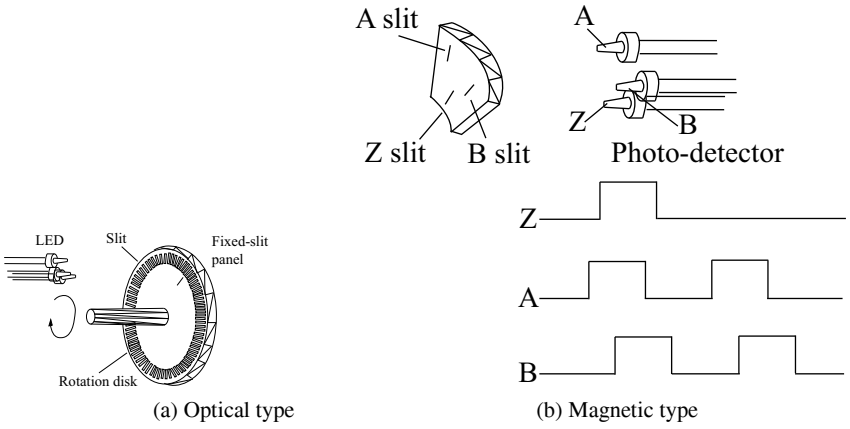


Fig. 1.5 The components and structure of an encoder

2. ***Absolute-type encoder***

The structure and the signal generation method of an absolute-type encoder are identical with those of an incremental-type encoder. However, the disk slit of an absolute-type encoder and the arrangement of photo-detectors are different from those of an absolute-type encoder as shown in Fig. 1.7. In this type of encoder, the slit on a disk slot provides a binary bit; so that, the outermost part of a disk is set to the lowest bit and as many slits and photo-detectors exist as the number of bits. The slits are arranged along concentric circles towards the interior of the disk. Based on these components, the rotation position data is output in binary or decimal form. In this way, the method where absolute position data is used is a graycode method.

Because an absolute-type encoder can detect the absolute position, the noise generated during sending the signal is not accumulated and the current position can be detected after the electric current has been cut off and then supplied again. However, because as many output signal wires are needed as the number of bits, it is difficult to minimize size and decrease price.

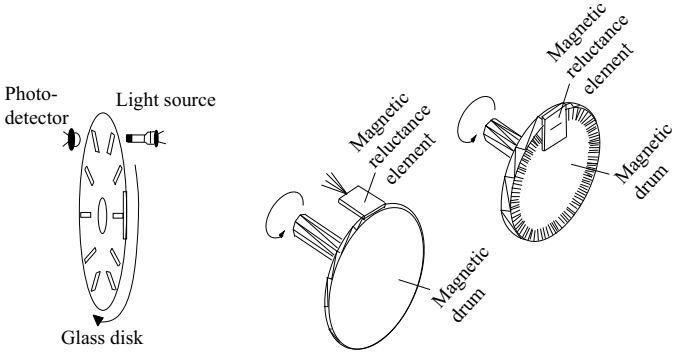
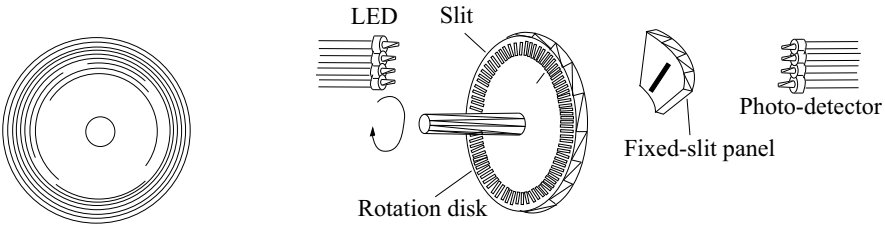


Fig. 1.6 Incremental-type encoder and output frequency



(a) Rotation disk (b) Encoder composition

Fig. 1.7 Absolute-type encoder

1.3.1.5 Resolver

A resolver is a detector of rotation angle and position and is used as the sensor of a motor. Unlike an encoder that generates an output signal in digital format, a resolver generates an output in analog format. A resolver consists of a stator, a rotor, and a rotation transformer. The coils of the stator and rotor are arranged to make the distribution of magnetic flux a sine wave with respect to the angle. A resolver has a similar structure to a motor and is insensitive to vibration and mechanical shock. In addition, because the output is an analog signal, the long-distance transmission

of signals and the miniaturization of the device are possible. However, the signal-processing circuit is complex and the device is more expensive than a rotary encoder.

1.3.1.6 Speed Sensor

Although an encoder and a resolver are typical position sensors, they can be used as speed sensors because speed can be calculated based on positional information from them. A tacho-generator is one of the typical speed sensors. In general, this is called a tacho-sensor and can be classified into brush-built-in types and brushless types. A brush-built-in type has a similar structure to a direct current dynamo. It comprises a stator, which is made from a permanent magnet, and a rotor, which is coiled. As a coil emits a magnetic flux with rotation of the rotor, a voltage is generated and is transmitted to the outside via the brush. The brushless-type comprises a rotor, being a permanent magnet, a coiled stator, and a single device that detects the position of the rotor. According to the rotational position of the rotor, the smoothed voltage induced from each coil is output sequentially. These two types generate a voltage that is proportional to the rotation speed. However, because the brush-built-in type has a limitation on life length, it is not used as the speed sensor of a servo motor.

1.3.2 Linear Movement Guide

A ball screw is used to move the tool post or table and plays the role of changing the rotation of a servo motor into linear movement. A Linear Movement (LM) guide is used to increase the accuracy and smoothness of the linear movement.

An LM guide consists of an M-shaped guide rail and a transferring part, Fig. 1.8. The bearing exists between the guide rail and the transferring part and lubricant is supplied to the surface of the LM guide rail to decrease friction while the transferring part is moving.

A ball screw is a lead screw that is operated by a ball bearing. A nut is designed to make the ball bearing rotate continuously and a ball bearing can come back by rotating from one end of the nut to the other end. As the ball is in contact with a screw and a nut, it plays the role of reducing the sliding friction of the lead screw that occurs due to rotation. Applying rolling contact to the contact surface between metals in contact minimizes the friction force when movement starts and prevents sticking when moving at low speed. In addition, reduction of backlash is possible by using an enlarged ball bearing or double nut.

The lead of a ball screw is related to the displacement unit of the machine tool table. In CNC, the displacement length per one pulse output from NC is defined as a BLU (Basic Length Unit). For example, if one pulse makes a servo motor rotate by one degree and the servo motor moves the table by 0.0001mm, one BLU will be 0.0001mm.

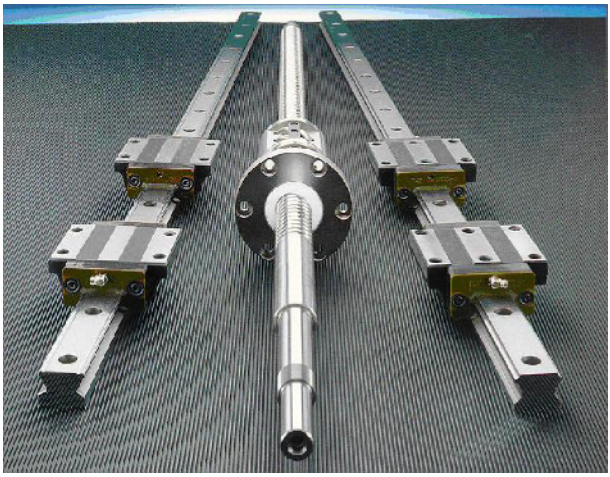
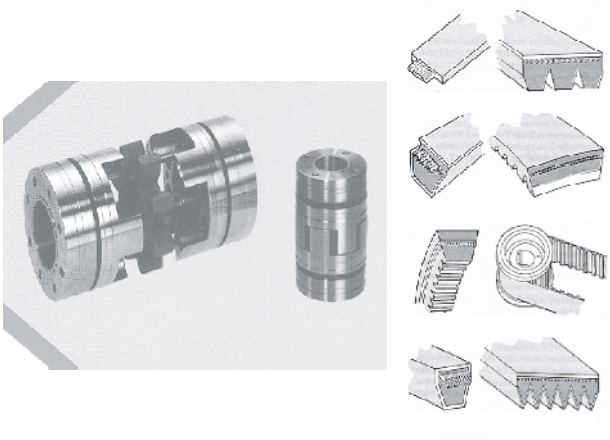


Fig. 1.8 Linear-movement guides

1.3.3 Coupling

A flexible coupling is mainly used as the machine component that connects a servo motor shaft with a ball screw. When a ball screw and servo motor are joined, the center of their shafts should be identical. However, in practice, this is very difficult. For this reason, a coupling should be designed to be insensitive to misaligned rotation centers. The flexible coupling shown in Fig. 1.9a meets the above-mentioned requirement and makes it easy to join the servo motor to the ball screw.



(a) (b)

Fig. 1.9 Flexible coupling and power transmission belt

In general, belts and gears are used as the mechanical components as the link between a spindle motor and a spindle body. In the case when a servo motor and a spindle body are separated, a belt is used for transmitting power. In the case when high torque is needed at low speed, gears are used as speed reducers. A way to use a gear or belt is called the "indirect driving method". Figure 1.9b shows a variety of belts. Using a belt is not suited to high-speed machining and has problems about noise and wear. To overcome these drawbacks, the direct driving method (direct drive) is used. In the direct driving method, the shaft of a spindle motor is directly connected with a spindle body or a spindle motor itself is built into a spindle body.

The advantages of direct drive are that backlash does not exist and the runout amount is very small. In addition, it is possible to suppress the variation of torque and rotation and it is easy to control. However, the price is very high.

1.4 CNC Control Loop

As the actual velocity and position detected from a sensor are fed back to a control circuit, the servo motor used in the CNC machine is continuously controlled to minimize the velocity error or the position error (Fig. 1.10). The feedback control system consists of three independent control loops for each axis of the machine tool; the outermost control loop is a position-control loop, the middle loop is a velocity-control loop, and the innermost loop is a current-control loop. In general, the position-control loop is located in the NC and the others are located in a servo driving device. However, there is no absolute standard about the location of control loops and the locations can be varied based on the intention of the designer.

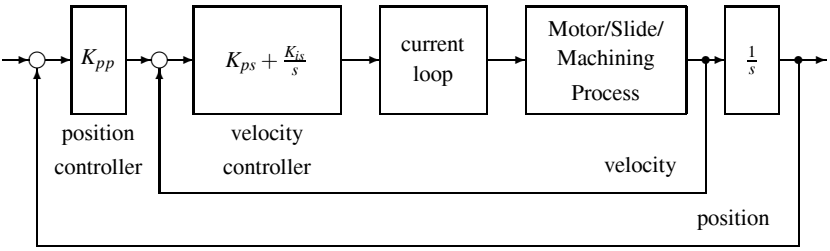


Fig. 1.10 Three kinds of control loop in CNC

In the spindle system of machine tools, feedback control of velocity is applied to maintain a regular rotation speed. The feedback signal is generally generated in two ways; a tacho-generator, which generates an induction voltage (analog signal) as a feedback signal, and an optical encoder, which generates pulses (digital signals). In recent times it is typical that feedback control is performed based on an optical encoder signal instead of a tachometer signal.

The detector can be attached to the shaft of a servo motor or the moving part and the control system can be categorized into four types according to the location at which the detector is attached.

1.4.1 Semi-closed Loop

The semi-closed loop is the most popular control mechanism and has the structure shown in Fig. 1.11a. In this type, a position detector is attached to the shaft of a servo motor and detects the rotation angle. The position accuracy of the axis has a great influence on the accuracy of the ball screw. For this reason, ball screws with high accuracy were developed and are widely used. Due to the precision ball screw, the problem with accuracy has practically been overcome.

If necessary, pitch-error compensation and backlash compensation can be used in NC in order to increase the positional accuracy. The pitch-error compensation method is that, at the specific pitch, the instructions to the servo driver system are modified in order to remove the accumulation of positional error. The backlash compensation method is that, whenever the moving direction is changed, additional pulses corresponding to the amount of backlash are sent to the servo driver system. Recently, the usage of the Hi-Lead-type ball screw with large pitch for high-speed machining has increased.

1.4.2 Closed Loop

The performance of the semi-closed loop depends on the accuracy of the ball screw and it is possible to increase the positional accuracy via pitch compensation and backlash compensation. However, generally speaking, the amount of backlash can be varied according to the weight of the workpiece and location and accumulation pitch error of the ball screw is varied according to the temperature. In addition, because the length of the ball screw is limited for practical reasons, a rack and pinion driving system is used in large-scale machine tools. However, the accuracy of the rack is limited. In this case, the closed loop shown in Fig. 1.11b is applied. In the closed loop, the position detector is attached to the machine table and the actual position error is fed back to the control system. Closed loop and semi-closed loop are very similar except in the location of the position detector, and the position accuracy of closed loop is very high. However, the resonance frequency of the machine body, stick slip, and lost motion have an influence on the servo characteristics because the machine body is included in the position control loop.

That is, a following error, the difference between the command position and the detected position, occurs and the servo is rotated at a speed proportional to this following error in order to decrease it. The decreasing speed of the following error is related to the gain of the position control loop. The gain is an important factor that

defines the property of the servo system. In general, as the gain increases, the response speed and dynamic accuracy increase. However, high gain makes the servo system unstable. Unstable means hunting, which is impossible to stop at the command position due to repetitive overshooting and returning. In the closed loop, if the resonance frequency of the machine driving system is not sufficiently higher than the gain, the control loop system becomes unstable. In addition, stick slip and lost motion are the main factors that give rise to hunting. Therefore, it is necessary to increase the resonance frequency of the machine driving system and, for this, it is necessary to increase the rigidity of the machine, decrease the friction coefficient of the perturbation surface, and remove the cause of lost motion.

1.4.3 Hybrid Loop

In closed loop, it is necessary to lower the gain in the case when it is difficult to increase the rigidity in proportion to the weight of the moving element or decrease lost motion as in a heavy machine. If the gain is very low, though, the performance becomes poor with respect to positioning time and accuracy. In this case, the hybrid loop shown in Fig. 1.11c is used. In the hybrid loop, there are two kinds of control loop; semi-closed loop, where the position is detected from the shaft of a motor, and closed loop, which is based on a linear scale. In the semi-closed loop, it is possible to control with high gain because the machine is not included in the control system. The closed loop increases accuracy by compensating the error that the semi-closed loop cannot control. Because the closed loop is used for compensating only positional error, it is well behaved in spite of low gain. By combining the closed loop and the semi-closed loop, it is possible to obtain high accuracy with high gain in an ill-conditioned machine.

1.4.4 Open Loop

Unlike the above-mentioned control loops, open loop has no feedback. Open loop can be applied in the case where the accuracy of control is not high and a stepping motor is used. Because open loop does not need a detector and a feedback circuit, the structure is very simple. Also, the accuracy of the driving system is directly influenced by the accuracy of the stepping motor, ball screw, and transmission.

1.5 The Components of the CNC system

The CNC system is composed of three units; the NC unit which offers the user interface and carries out position control, the motor unit, and the driver unit. In a narrow

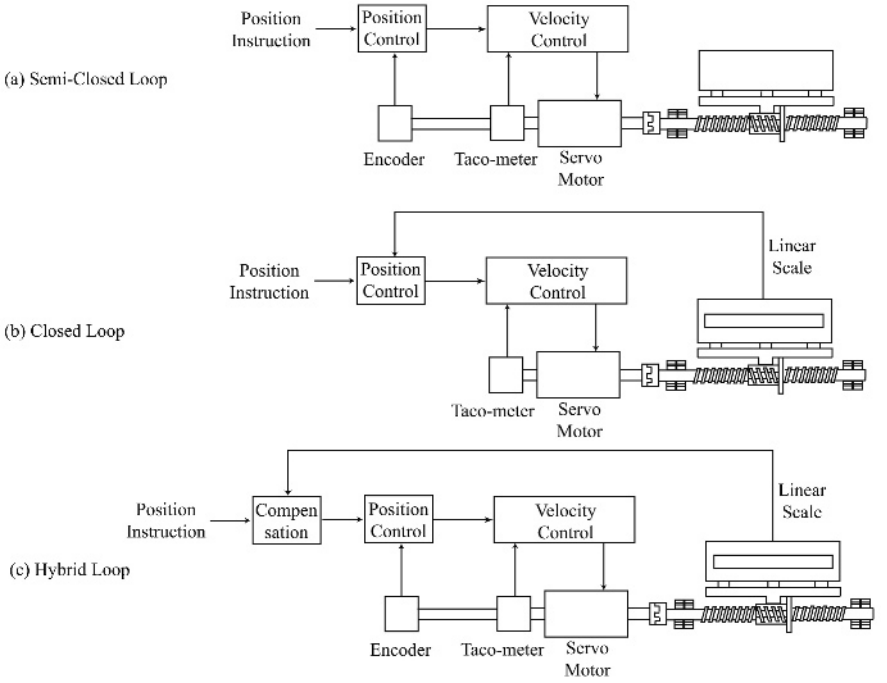


Fig. 1.11 Classification of control mechanism according to position data detection method

sense, only the NC unit is called a CNC system. The contents of this book focus on the architecture and function of NC and do not include the motor unit and the driver unit

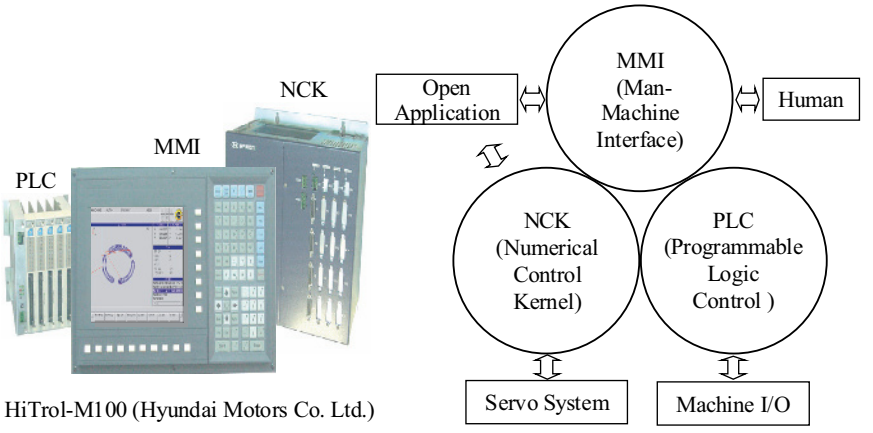


Fig. 1.12 The construction of CNC

From a functional point of view, the CNC system consists of the MMI unit, the NCK unit, and the PLC unit, Fig. 1.12. The MMI (Man Machine Interface) unit offers the interface between NC and the user, executes the machine operation command, displays machine status, and offers functions for editing the part program and communication. The NCK (Numerical Control Kernel) unit, being the core of the CNC system, interprets the part program and executes interpolation, position control, and error compensation based on the interpreted part program. Finally, this controls the servo system and causes the workpiece to be machined. The PLC (Programmable Logic Control) sequentially controls tool change, spindle speed, workpiece change, and in/out signal processing and plays the role of controlling the machine's behavior with the exception of servo control.

Figure 1.13 shows the conceptual architecture of CNC machine tools from the hardware and software points of view.

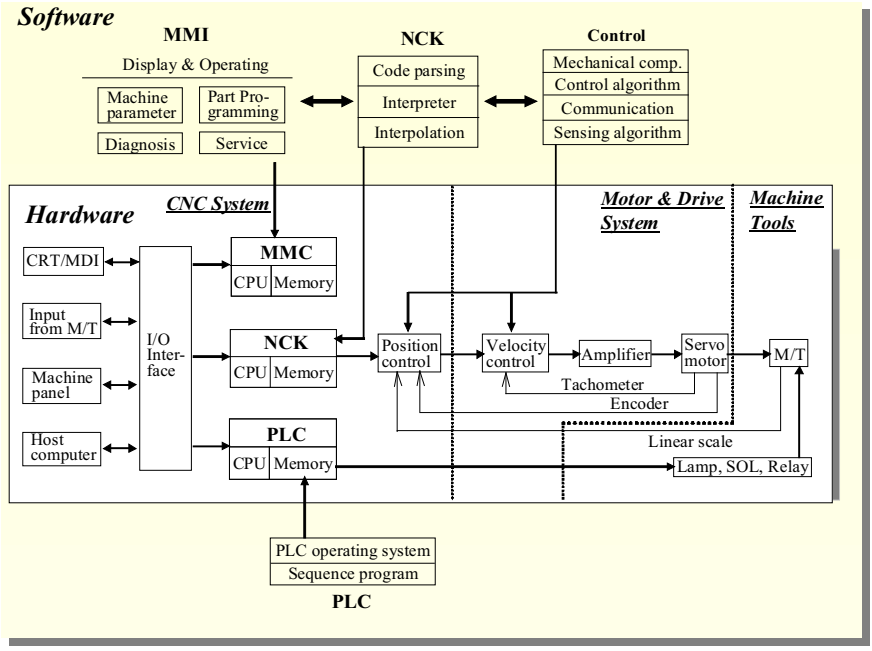


Fig. 1.13 The components of a CNC system

From the hardware point of view, CNC machine tools consist of CNC, motor drive system, and machine tools. The output of the position control, being the end function of the CNC system, is sent to the motor drive system, the motor drive system operates a servo motor by velocity control and torque control, and, finally, the servo motor makes the moving part move via the power-transmission device. In the CNC system, the processor modules that process the functions of the MMI unit, NCK unit, and the PLC unit consist of a main processor, a system ROM and a RAM that stores

user applications, part programs and PLC programs, respectively. The process module is connected with an interface that is equipped with key input, display control, external input and system bus. Therefore, the architecture of a CNC system is similar to that of a multi-process computer. The CNC system also has an Analog/Digital input/output device for direct communication with external machines and a communication interface for linking an external motor driving device with an input/output module.

In the CNC system, initially velocity commands in analog format were used for transmitting signals to the motor driving system. However, recently, because noise occurs while transmitting analog signals, not only are digital signals used for velocity command but also digital communication is used for communication between the CNC system and the motor driving system. SERCOS is the most popular digital communication mechanism and has come to be a de-facto standard. In digital communication there is an advantage that it is possible to exchange a variety of data and remove noise by using optical cables. Therefore, it is possible to set the parameters of the driving system in NC, monitor the status of the driving system, and increase accuracy by removing noise.

By expanding the concept of digital communication, the communication mechanism has been applied to input/output devices. That is, the connection between a CNC system and a variety of sensor and mechanical devices is done via only one communication line. For this communication mechanism, a standard communication protocol is essential and various protocols such as Profi-Bus, CAN Bus, and InterBus-S were introduced.

From the software point of view, the CNC system can be shown as in Fig. 1.13. The CNC system consists of MMI functions that support user operation and program editing and display machine status, NCK functions that execute interpretation, interpolation and control, PLC functions that carry out sequential logic programs. In the following sections, these will be addressed in detail.

1.5.1 MMI Function

The MMI unit offers the user interface that is needed when a user operates machine tools. Therefore there are many kinds of user interface based on the design concepts of the CNC maker. Functions of the user interface are generally classified into five groups.

1. ***Operation functions:*** These functions are used very frequently and support operation of the machine and the display that shows the machine status. Figure 1.14a depicts the status of the machine while it is running. In Fig. 1.14a, the position, distance-to-go, and feed of each axis, spindle speed, the block that is being executed, and override status are shown. In addition, functions to help machine operation such as jog, MDI, program search, program editor, and tool management are provided.

2. ***Parameter-setting functions:*** In the CNC system there are various parameters for internal use and these are categorized into three kinds: Machine parameters that are used for setting machine regulation, servo/spindle driving system, tool offset, work coordinate, and safety boundary; program parameters that should be set during editing of the part program; and customization parameters that are used to adapt the machine to user requirements. These functions provide the interface for setting, storing, and searching parameters. Figure 1.14b shows the display for searching for internal parameters and modifying them.
3. ***Program-editing functions:*** These functions are able to edit and modify the part program, which is G-code based on the EIA/ISO standard. Practically, it is necessary for the user to know G/M-codes and carry out mathematical calculations in order to generate the G-code part program. Because mathematical calculation makes it difficult to edit part programs, CNC has recently begun to employ conversational programming systems. Figure 1.14c shows the display that the conversational programming system provides in order to edit a part program for drilling. By interaction with the GUI a user can quickly generate a part program for drilling without memorizing the input attributes for G-code cycles. Figure 1.14d also shows the shape calculator to help a user define the geometric shape. Recently, the conversational programming system has come to be recognized as an essential function of CNC and therefore, in this book, the design and the example of a feature-based conventional programming system will be addressed in detail.
4. ***Monitoring and alarm functions:*** the CNC system always informs a user of the machine status by monitoring and, if need be, these functions execute the necessary tasks and inform the user of the result. These functions are essential when machine tools are executing at high speed. These functions play the role of providing monitoring information such as the alarm status, emergency recovery method, PLC status, and ladder diagram under execution.
5. ***Service/utility functions:*** Besides the other four essential functions, many useful functions are provided to assist users. The DNC function for transmitting the part program, which is edited externally, to the CNC, the file service for copying internal parameters to the outside, and the communication function for communicating with computers belong to these functions.

1.5.2 NCK Function

In general, the NC system interprets the input data, keeps them in memory, sends commands to the driving system, and detects feedback signals from the drive system. The NC system also performs logical decision making such as when coolant is provided and when the spindle starts rotating and mathematical calculations for acceleration control and interpolation of lines, circles and parabolae. Therefore, the NCK unit has the task of being in charge of the servo and driving control and the PLC unit has the task of being in charge of logic control, so the burden that occurs

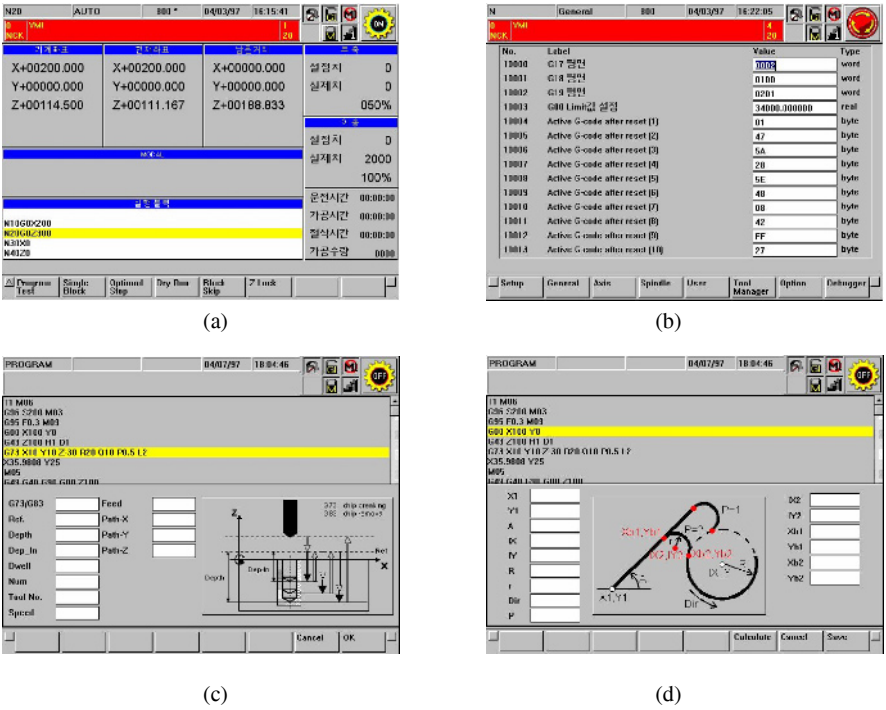


Fig. 1.14 Man-Machine Interface (HiTrol-M100) (a) Operation functions, (b) Parameter setting functions, (c) Drilling editing functions, (d) Geometric shape calculator

during control is adequately balanced. The functional blocks and the data flow of the NCK unit, being the key unit of the CNC system, are shown in Fig. 1.15. The interpreter, interpolator, acceleration/deceleration controller, and position controller are the main functions of the NCK unit.

1. An **interpreter** plays the role of reading a part program, interpreting the ASCII blocks in the part program, and storing interpreted data in internal memory for the interpolator. In general, NC issues the orders related to the interpreted data and the interpreter reads and interprets the next block while the command is being performed. However, if the time to interpret the block is longer than the time to finish the command, the machine should wait for the completion of interpretation of the next block so that a machine stop cannot be avoided. Therefore, in order to prevent machine tools from stopping, a buffer that temporarily stores the interpreted data is used. The buffer, called the internal data buffer, always keeps a sufficient number of interpreted data and all interpreted data are stored in the buffer. Details will be given in Chapter 2.
2. An **interpolator** plays the role of sequentially reading the data from the internal data buffer, calculating the position and velocity per unit time of each axis, and

storing the result in a FIFO buffer for the acceleration/deceleration controller. A linear interpolator and a circular interpolator are typically used in an NC system and a parabola interpolator and a spline interpolator are used for part of an NC system. The interpolator generates a pulse corresponding to the path data according to the type of path (*e.g.* line, circle, parabola, and spline) and sends the pulse to the FIFO buffer. The number of pulses is decided based on the length of path and the frequency of the pulses is based on the velocity. In an NC system, the displacement per pulse determines the accuracy; for example, if an axis can move 0.002 mm per pulse, the accuracy of the NC system is 0.002 mm. In addition, the NC system should generate 25000 pulses for the moving part to move as much as 50 mm and 8333 pulses per second to move at a speed of 1m per minute. In Fig. 1.15, the data in the FIFO buffer is transmitted to the next function via a fine interpolator, which interpolates precisely the interpolated data and, if not necessary, does not have to be implemented. Details will be given in Chapter 3.

3. If **position control** is executed by using the data generated from the interpolator, large mechanical vibration and shock occur whenever part movement starts and stops. In order to prevent mechanical vibration and shock, the filtering for **acceleration/deceleration control** is executed before interpolated data is sent to the position controller. This method is called the "acceleration/deceleration-after-interpolation" method. An "acceleration/deceleration-before-interpolation" method exists too, where acceleration/deceleration control is executed before interpolation. These two methods will be addressed in Chapter 4 in detail.
4. The data from an acceleration/deceleration controller is sent to a position controller and position control is carried out based on the transmitted data in a constant time interval. A position control typically means a PID controller and issues velocity commands to the motor driving system in order to minimize the position difference between the commanded position and the actual position found from the encoder.

   However, the problems of noise cannot be avoided by using an analog signal. Chapter 5 will address this subject in detail.

1.5.3 PLC Function

The logic controller is used to execute sequential control in a machine and an industry. In the past, logic control was executed by using hardware that consisted of relays, counters, timers, and circuits. Therefore, it was considered as a hardware-based logic controller. However, recent PLC systems consist of a few electrical devices including microprocessors and memory, able to carry out logical operations, a counter function, a timer function and arithmetic operations. Therefore, a PLC system can be defined as a software-based logic controller. The advantages of PLC systems are as follows:

Flexibility: The control logic can be changed by changing only a program.

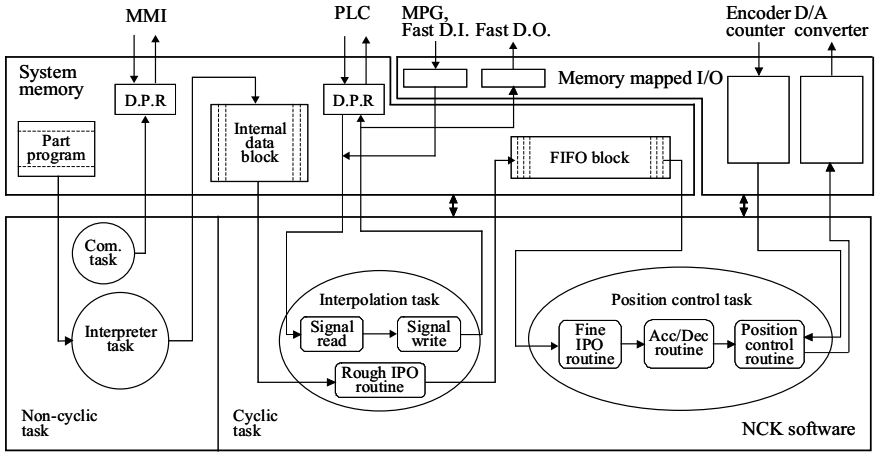


Fig. 1.15 NCK functional blocks

Scaleability: The expansion of a system is possible by adding modules and changing programs.
Economic efficiency: Reduction of cost is possible due to the decrease in design time, high reliability, and easy maintenance.
Miniaturization: The installation dimension is smaller compared to a relay control box.
Reliability: The probability of failure occurrence due to bad contact decreases because of using a semiconductor.
Performance: Advanced functions such as arithmetic operations and data editing are possible.

The hardware architecture of the PLC unit of an NC system comprises a microprocessor, a system memory, a program memory, and an input/output module as shown in Fig. 1.13. As soon as the power is turned on, the system memory sets the PLC hardware environment and the program memory, manages input/output, relay/timer/counter and stores a user program and the data to be interpreted by the microprocessor. The input/output module manages the interface with limit switch, relay, and ramp.

The function modules that are executed in a PLC unit can be defined as shown in Fig. 1.16 and are summarized below. Initially, a user creates the application program used in the PLC unit by using an external PLC program editor and inputs the application program to the PLC unit. At this stage, a specific device is used for helping the user to edit the program and is called a programmer or loader. The programmer consists of the editor that creates a program and the compiler that converts the program into the PLC-interpretable language. The reason why a compiler is used is that a compiled program is more efficient and hence the PLC can run the program quickly. The compiled PLC program is transmitted to the CPU module. In addition,

the status of the PLC that is being executed in the CPU module is sent to the PLC program for a user to monitor the activity status.

The module that reads the program edited by the Loader and executes sequential logic operations is the Executer, which is the core of a PLC kernel. The Executer is repeated successively, reading the input points, doing logic operations of the program, and sending the results to the output points via the output module.

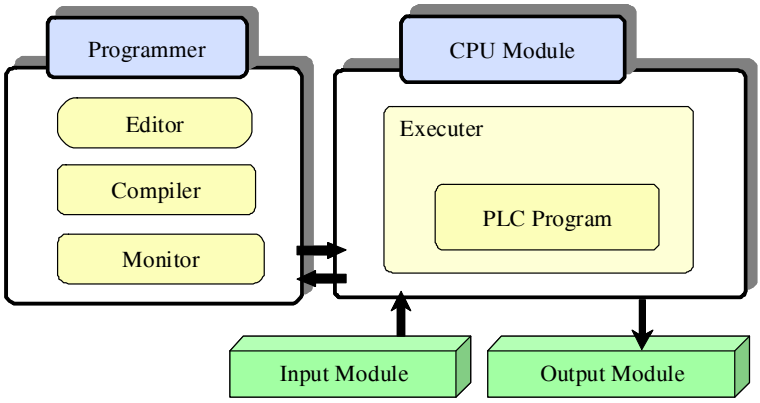


Fig. 1.16 The architecture and function of the PLC system

The PLC unit of a CNC system is similar to the general PLC system but there is an auxiliary controller that assists with part of the functions of the NCK unit. Therefore, the following functions are necessary:

- Circuit dedicated to communicating with NCK.
- Dual-port RAM for supporting high-speed communication.
- Memory for the exchanged data during high-speed communication with NCK.
- High-speed input module for high-speed control such as turret control.

In practice, according to the decisions of individual CNC and PLC makers, various PLC languages are used. Due to this, there is a problem with respect to maintainability and training of users. To overcome this problem, the standard PLC language (IEC1131-3) was established and usage has spread. The standard, IEC-1131-3, defines five kinds of language; 1) Structured Text (ST), 2) Function Block Diagram (FBD), 3) Sequential Function Charts (SFC), 4) Ladder Diagram (LD), and 5) Instruction List (IL 1). Now it is necessary for users to edit programs based on the standard language and it is required for developers to implement applications for interpreting and executing a PLC program.

1.5.4 Real-time Control System

In an NC system, the NCK unit, the PLC unit, and the MMI unit should be executed in constant time intervals. Because of this property, the NC system is a complex real-time system.

For example, assume that there is a system that has NCK functions such as interpretation, interpolation, and position control and the MMI function. In this system, fine management of the execution schedule of the modules that occupy the processor resource is required, as shown in Fig. 1.17. That is, the task scheduling function that manages the execution of the modules based on predefined time intervals and priority is necessary.

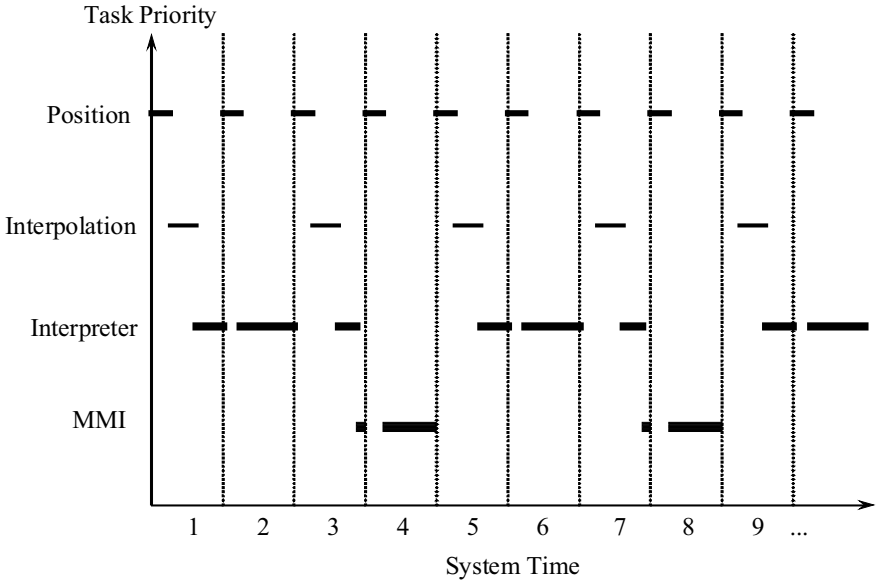


Fig. 1.17 Task scheduling in an NC system

In the above example, the design intention is that the position controller, the interpolator, the interpreter, and the MMI have, respectively, the highest, the second, the third, and the lowest priorities. Also in the design is the feature that the position control with the highest priority is activated every 1 msec and the interpolator with the second highest priority every 2 msec, and that the interpreter is executed once every 4 msec. The above-mentioned three tasks are designed to have constant cycle time (sampling time). The MMI with the lowest priority is designed to use the surplus processor resource after the cyclic tasks finish. That is, it is designed as a non-cyclic task.

For the NC system design, the modularization of tasks and the system design such as the allocation of priority for each task, the selection of scheduling method and the synchronization and communication mechanism among tasks are required.

In this book, the architecture and design of the CNC system and the architecture and functions of the NCK, the PLC, and the MMI unit will be addressed in detail. With this information it will be possible for a reader to implement a CNC system.

The contents that will be presented about the NCK unit in this book can be applied not only to the NCK unit of an NC system but also to the GPMC (General Purpose Motion Control) for the servo motor. In general, because MMI, NCK, and PLC comprise very complex and real-time functions, usage of individual microprocessors for each module is typical. However, the advancement of microprocessor technology makes it possible to execute all modules by using a single processor. That is, the implementation of a simple CNC system becomes possible by dividing the functions of the modules into cyclic tasks and non-cyclic tasks and using real-time OS. The design of the PLC and the MMI unit, real-time OS, operation of application modules, and system programming will also be addressed.

1.6 The Progress Direction of the CNC System

Over the last 50 years, the advance of NC and CNC can be summarized as shown in Fig. 1.18. NC systems developed in the 1950s were implemented based on hardware and research for replacing the hardware components with software components got under way. The advancement of electrical technology in the 1970s - 1980s, especially, meant that the NC system became a CNC system whose functions were executed by a micro processor. However, the CNC system is a closed system where a user cannot add customized functions to the CNC system.

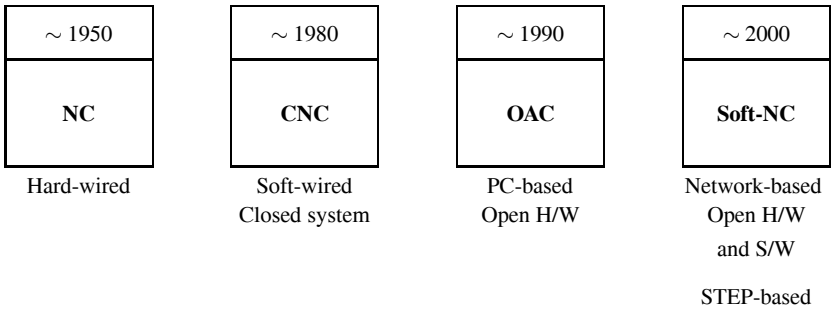


Fig. 1.18 The change of NC over time

Therefore, during the 1990s, an effort was made to change the closed CNC system to an Open Architecture Controller (OAC), which is a user-oriented CNC system,

and PC-based open CNC consisting of a PC and VME was introduced as an Open Architecture Controller. However, the PC-based open CNC did not achieve perfect openness to fully satisfy the various requirements of a user from the hardware and software points of view. Consequently, in order to overcome this drawback, it is expected that future CNC systems will advance to become soft-NC, being open CNC with an architecture where it is possible to achieve full openness in the software as well as in the hardware as modularized functions are systematically integrated via network technology.

Soft-NC is designed to use a PC platform as hardware and the architecture is designed based on object-oriented technology in order to fulfill openness from the software viewpoint. Component technology will be applied to the implementation of function modules and a common soft-bus that is able to support real-time distributed processing will be applied for integration of components. The soft-NC will have an architecture that can be applied to robots as well as to machine tools. The main characteristics of Open CNC are summarized below.

1. Support STEP as a standard data format for machining data for a seamless flow in the manufacturing process (from CAD to machining).
2. Use soft-bus (*e.g.* CORBA) to support a real-time distributed processing environment and to be free to add function modules to CNC.
3. Support component-based API to connect application software modules with the system platform.
4. Support standard interfaces to communicate with external systems.
5. Support many kinds of field bus to connect sensors with I/O.
6. Support a digital interface between the servo system and the CNC system.

Figure 1.19 shows an example of a CNC to which the concept of Soft-NC is applied. The CNC shown in Fig. 1.19 is designed based on the concept of modularization, standardization, and distributed processing and uses STEP-compatible part programs as input. This CNC generates toolpaths by using the code interpreter and the toolpath generator that are linked via soft-bus and API. After this, the toolpath is transferred to NCK via a standardized communication environment. During execution of the above-mentioned procedure, the operation monitoring module and the adaptive control module, which users develop and add to the CNC, are executed simultaneously.

In conclusion, the future NC system should be more intelligent and should be able to carry out complex tasks in a distributed environment with the advancement of micro processors and communication systems. That is, the future NC system will advance to be a distributed system that can execute real-time compensation based on machining status from sensors, communicate and share machining data with other CNCs, Cell controllers, and MES, and perform tasks by itself.

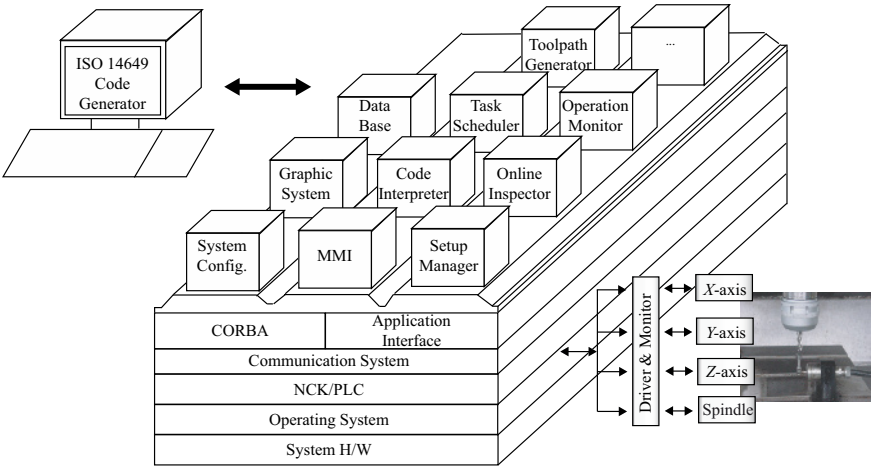


Fig. 1.19 Conceptual architecture of an intelligent STEP-compliant CNC

1.7 Summary

The MMI, NCK, PLC, and servo driver are components of which an NC system consists and, in particular, the NCK is a key component that has an influence upon the performance of CNC. NCK consists of the interpreter, the interpolator, Acceleration/Deceleration controller, and position controller; the interpreter interprets a part program and calculates the position to which the axes move. The interpolator calculates the displacement of axes for every sampling period. The Acceleration/ Deceleration controller plays the role of smoothing axis movement. The position controller controls the axis movement every sampling period for position control.

The PLC is the module that controls machine behavior, except for servo control, and is similar to the PLC system of an automated device with respect to sequentially carrying out the sequence program (the so-called ladder program) edited by the user. However, in respect of exchanging data with the NCK and sharing the system resource, the PLC in a CNC system is distinguished from the PLC system of an automated device.

The MMI plays the role of providing the various user interfaces that are needed for a user to use NC easily. In order to design the MMI, it is necessary to understand the cutting mechanism and machine operation mechanism unlike PLC and NCK. In order to execute the above modules, it is essential to guarantee a real-time environment. That is, the key for developing Soft-NC is the scheduling that satisfies the execution of the software modules in real time. Recently, NC systems have advanced from being closed systems to being open and distributed systems and the importance of software will grow.

Chapter 2
Interpreter

The Numerical Control Kernel (NCK) unit is the key component of a CNC system and consists of a variety of modules that are sequentially executed in a synchronized schedule. In this chapter the code interpreter will be addressed. This is responsible for converting the part program and machine instructions into internal commands for NC. In order to understand the code interpreter the first thing is to understand the part program that is the input to the interpreter. After this, the structure and the functions of the code interpreter will be addressed in detail.

2.1 Introduction

The code interpreter is a software module, which translates the part program into internal commands for moving tools and executing auxiliary functions in a CNC system.

Figure 2.1 depicts the internal behavior of the CNC system and shows the functions of the Man-Machine Control (MMC), Numerical Control Kernel (NCK), and Drives (DRV). The part program that a programmer generates based on the shape of the part, cutting conditions, and tools is entered into CNC via the MMC and the NCK subsequently generates the control commands for the drivers from the part program through various stages; calculating the movement path by interpreting the part program, generating velocity profile and displacement for each axis by interpolation, smoothing the movement by acceleration/deceleration (acc/dec) control, and generating position control command.

Among these stages, the interpreter could be considered as a simple task for the conversion of G/M codes to the CNC-understandable internal data structures. However, the design and implementation of the interpreter is a large and comprehensive task because programming rules or grammar described in a programming manual and an operating concept shown in an operation manual should be considered when developing the interpreter. Therefore, the interpreter is the representative indicator that shows the design concept and the functional aspect of a CNC and is a big part of

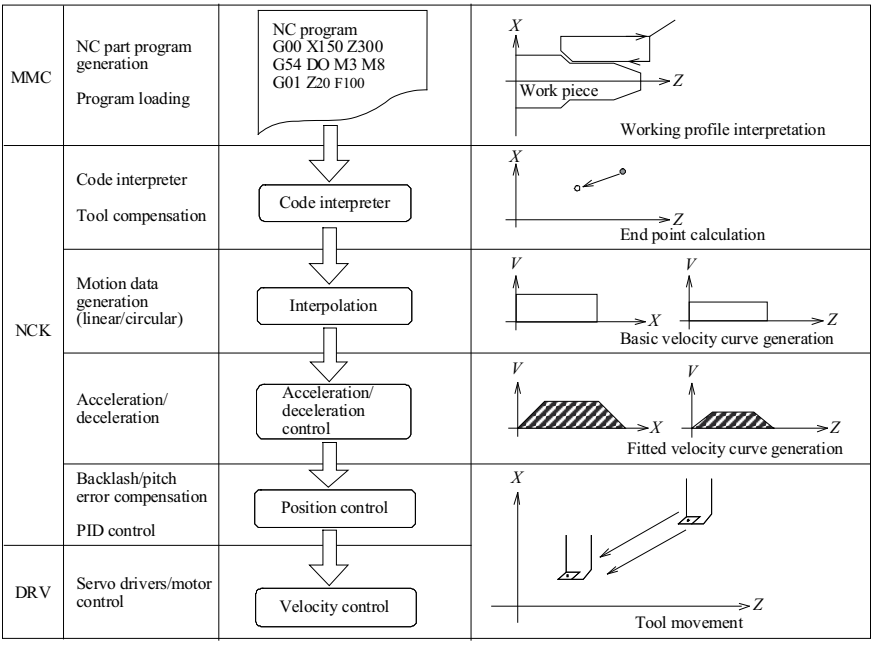


Fig. 2.1 Internal behavior of the CNC system (MMC, NCK, DRV functions)

CNC as it generally spends more than 50% of the total development time to develop the interpreter.

In this chapter, the format and function of CNC part programs will be briefly addressed and the architecture, such as the structure of an interpreter, execution procedure, and memory structure, will be addressed. However, because the detailed function of the CNC and the part program are slightly different for each CNC maker, the program manual should be referenced to find out the detailed functions of any particular CNC.

2.2 Part Program

Although the standard exists for generating a CNC part program, sequentially listing the commands for executing CNC, each CNC maker has, in practice, their own code system including their own commands. In this section, the common concept will be described based on the standard code.

2.2.1 Program Structure

A part program contains the commands, called blocks, for machining a part and each block can be defined using the following commands.

- NC commands such as G, M, S, T, H, D, F code and related address
- Call of sub program and displaying message
- Setting variable and conditional program calls

In a part program, the English alphabet, Arabic numbers, and symbols are used and Fig. 2.2 illustrates the format and elements of a part program.

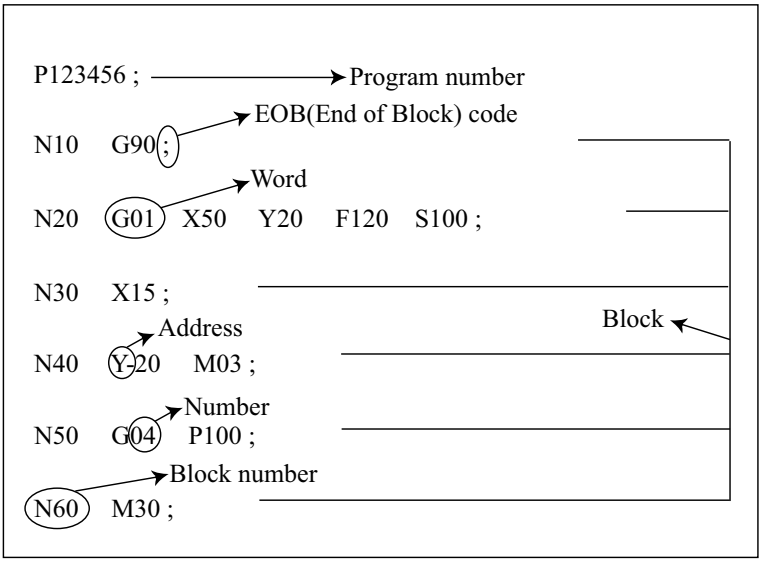
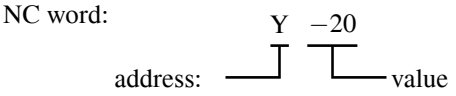


Fig. 2.2 Program format and construction elements

A part program consists of a sequence of NC blocks, each block consists of several words, and a word is composed of an address and number. The program number is a number for identifying the particular part program on CNC, where more than one part program is executed, and is written using a particular address and number in the heading of a part program. In this book, address P is used but O or # is also used by some specific CNC makers.

A block consists of one block number, at least one word, and the EOB, meaning the End Of Block. The word is the set of characters in a specific order. The word is the minimum unit for internal processing and commanding the machine tools to perform a particular behavior. The word consists of an address and a subsequent number, as below:



The address is constructed from one of the alphabetic characters (A Z) or a combination of alphabetic characters. The subsequent number provides the data that is required to execute the behavior related with the address. Table 2.1 summarizes the addresses that have typically been used and the function that is related with the address.

Table 2.1 Typical addresses and associated functions

Function	Address	Meaning (Example)	Unit
Program number	P	Program identity no. *e.g.* P123456	
Block number	N	NC seq.no. N100	
Preparatory function	G	Mode command G01	
Coordinate (command for translational axis)	X, Y, Z / U, V, W	Axis / dir. X100 W20	mm, inch
Coordinate (command for rotary axis)	A, B, C	Axis A30	deg
Feedrate	F	Feedrate per min. F200	mm/min, inch/min, deg/min
		Feedrate per rev. F1	mm/rev, inch/rev, deg/rev
Spindle speed	S	S3000	rpm
Tool	T	Tool number T12	
Auxiliary function	M	Machine command M06	
Offset Number	H, D	Offset register no. H10	
Number of repetitions of subprogram	L	Iteration no. L5	
Radius of circle or arc	R	Arc rad., corner rad. R3	mm, inch
Chamfer	C	Chamfer amount C2	mm, inch
Center position of circle	I, J, K	Circle center coords.	mm, inch

Among the addresses described in Table 2.1, the G address, for preparatory function, and the M address, for auxiliary function, are largely related to the performance of CNC system. G addresses denote commands for tool movement by moving the

translational axes or the rotary axes along the specified path. M addresses denote commands for controlling the on/off functions in machine tools. G-codes are classified into two types: one is a modal code and the other is non-modal code. A modal-type code is effective throughout the following blocks until the modal cancel command is used. On the other hand, a non-modal-type code is effective within the commanded block and automatically canceled by the next block. Modal-type codes are classified into several groups, called modal groups, with respect to the similarity of function. In one block, it is prohibited to use more than one G-code that is included in the same modal group.

The address groups based on the functions of CNC system are summarized in Table 2.2. As the standard for editing a part program based on these addresses, ISO 6983 has been widely used. However, each CNC maker has their own G&M code system where maker-specific functions have been added to ISO 6983. Accordingly, current G&M code systems for generating part programs depend on the CNC system. If the CNC system is changed, it is almost impossible to reuse the existing part program. Therefore, in order to create a part program manually, it is necessary to refer to the programming manual of the particular CNC maker.

According to the level of CNC system, the number of feasible addresses varies from several tens to several hundreds. Ths means that the more feasible addresses a CNC system has, so the more advanced the equipment category to which the CNC system belongs. Further, according to the machine type, the applicable addresses are defined in different ways. The G-code list and the modal group for a milling machine and a turning machine are summarized in Appendix A. In the following, the interpreter is the module that has the function of interpreting the various addresses, words, and grammar.

Table 2.2 Summary of address groups based on CNC system functions

Functions	Description
Preparatory function (G-code)	Codes are used for controlling axis and the CNC system prepares the execution of the particular function. (1) movement: command the relative movement between tool and workpiece. (2) Setting local coordinate system: specify the origin and orientation of local coordinate system including orthogonal coordinate system, the polar coordinate system, and rotational coordinate system. (3) Interpolation: command machining various profiles such as line, arc, helical, spline, etc. (4) Miscellaneous: commands for tool compensation, safety check, and skip.
Feed function (F-code)	Command the relative speed between tool and workpiece for interpolation command.
Spindle functions (S code)	Command the spindle speed (RPM).
Tool function (T-code)	Command tool change and specify the tool compensation.
Auxiliary function (M-code)	Function commands for the simple control of a machine tool including relay/switch on and off as well as axis control. (1) Coolant: command coolant on or off. (2) Start/End of a main and sub program: inform the beginning and end of a part program. (3) Spindle: command a spindle to rotate in CCW or CW direction and specify the spindle speed, limit speed and spindle gear change. (4) Miscellaneous: Command the change of tool, workpiece, pallet and rotation of the table/loader.
Utility functions	(1) Subprogram: To simplify the main program, subsidiary program (2) MACRO register a series of commands as a particular command and execute various functions by using the registered command. In a macro, the use of variables is possible and arithmetic operations between variables is possible. (3) Fixed cycle (i) Turning cycle Define a series of commands to perform the turning operations as one command. Outer turning, facing, grooving, and threading cycle are defined. (ii) Drilling cycle Define a series of commands to perform drilling, tapping, boring, reaming and peck drilling as drilling cycle.

Table 2.2 (continued)

	(iii) Milling cycle Pocket milling, Slot milling. Define a series of commands to machine profiles that are frequently machined during pocket milling and slotting as milling cycles. (iv) Touch Probe cycle This command enables the modification of a program via on-machine inspection before or after machining. This enables compensation of finishing with inspection of the machining accuracy.

It is possible to edit the description by inserting words in parentheses within a block, as below.

N20 G01X0Y0 (MOVE TO ZERO POINT);

comment

The description comment has no influence on the execution of a part program. Because the description can be shown on the display of the CNC system together with the block during editing or executing a part program, it is very useful for managing part programs.

The end of a part program is signalled by the command M02 or M03. By inserting M02 or M03 at the end of a part program, all modal values are initialized and reset. Since the commands M02 and M03 are executed last, they can be located anywhere within the last block.

2.2.2 Main Programs and Subprograms

2.2.2.1 Main program

A part program is classified into a main program and subprograms. Typically, the CNC system executes a main program. If a main program includes the command that is used for calling subprograms, the CNC system executes the subprogram indicated. If, during execution of the subprogram, the command for returning to the main program is called, the main program is then resumed at the block after the command that called the subprogram, as shown in Fig. 2.3.

2.2.2.2 Subprogram

In the case that there are fixed routine blocks or iterated operation patterns in a part program, part programming can be made easier if they are stored as a subprogram in the internal memory of CNC system. It is possible to call the subprogram from a

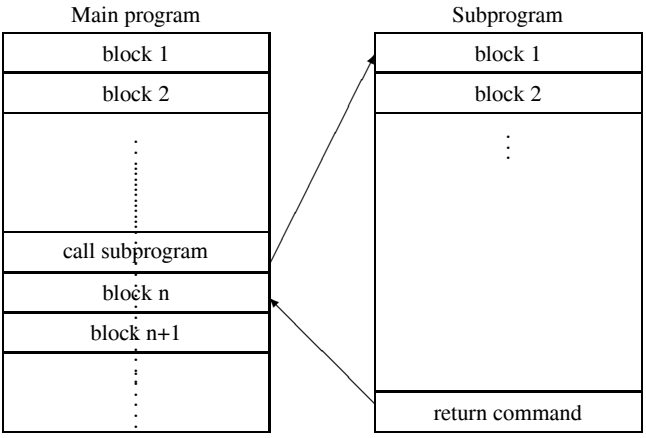


Fig. 2.3 Main program and subprogram execution

main program during auto mode of a CNC system. It is also possible to call another subprogram from within a subprogram.

2.3 Main CNC System Functions

The main functions of a CNC system can be classified into a variety of groups such as coordinating functions, interpolation functions, compensation functions, safety functions, and auxiliary utility functions. These will be described in the following sections.

2.3.1 Coordinate Systems

In CNC systems, a machine coordinate system, a workpiece coordinate system, and local coordinate systems are defined for convenience when editing a part program and handling machine tools.

A machine coordinate system is defined by setting a particular point of the machine tool as the origin of a coordinate system. A workpiece coordinate system is defined by setting a particular point on the workpiece as the origin so as to make editing a part program easier. That is, when editing a part program using one particular workpiece coordinate system, we can edit the part program by defining another coordinate system based on the workpiece coordinate system. We call this secondary coordinate system a “local coordinate system”. A workpiece coordinate system is set by commanding particular G-codes (G54 to G59) and a local coordinate system is defined by setting an offset (IP) that denotes the displacement of the local coordinate

system from the origin of the workpiece coordinate system. Based on the origin of the machine coordinate system, the relationship between each workpiece coordinate and local coordinate system is illustrated by Fig. 2.4.

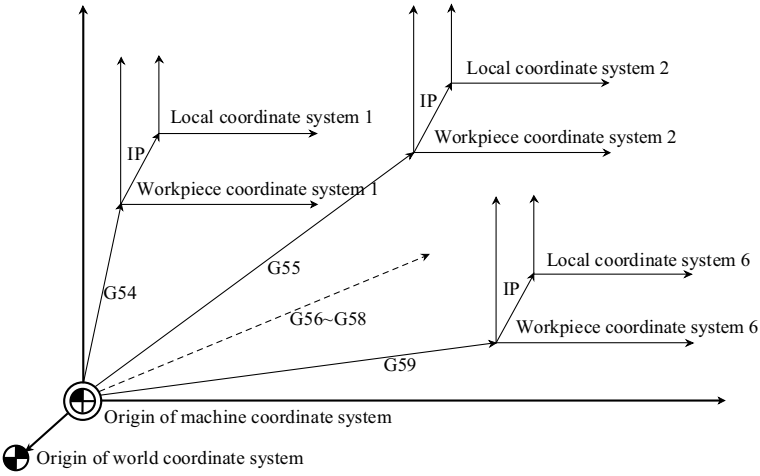


Fig. 2.4 Machine, workpiece and local coordinate systems

As methods to command displacements of each axis based on the specified coordinate system, there are two modes, absolute programming mode (G90) and incremental programming mode (G91). When absolute programming mode is used, the end position of each axis is programmed. When incremental programming mode is used, the relative displacement of each axis is programmed.

Besides orthogonal coordinate systems, it is also possible to use polar coordinate systems (G15) where a radius component and angle components are used. Figure 2.5 shows a part program using the polar coordinate system and the path that is commanded by the part program. To use the polar coordinate system, first a work plane is selected and then a polar coordinate system is invoked by issuing the command G15. Thereafter, when using address *X* and address *Y*, a radius and an angle, respectively, are commanded.

For part programming, it is possible to use the scaling function and the rotation function based on the specific coordinate system. The scaling function is used for scaling down or up the programmed workpiece shape. To command the scaling function you use the G51 X_ Y_ Z_ P_ format in a block, wherein the X, Y, Z address denotes the center position for scaling and is given as an absolute value. The P address is used for the magnitude of the scaling. As G51 is a modal G-code, the toolpaths in the following blocks are scaled P times up or down with respect to the point determined by the values above X, Y and Z.

To rotate the specific shape in a part program, the G68 α_ β_ R format is utilized wherein α and β denote the center position for rotation and R means a rotational angle (+R denotes CCW and –R denotes CW). Accordingly, after declaring G68 in

```
N0100   G17   G90   G15 ;   XY plane, absolute coordinate, polar coordinate start
N0200   G00   X100  Y30 ;   rad. 100mm, ang. 30deg
N0201         X100  Y150 ;  rad. 100mm, ang. 150deg
N0202         X100  Y270 ;  rad. 100mm, ang. 270deg
N0203   G16 ;               Polar coordinates cancel
```

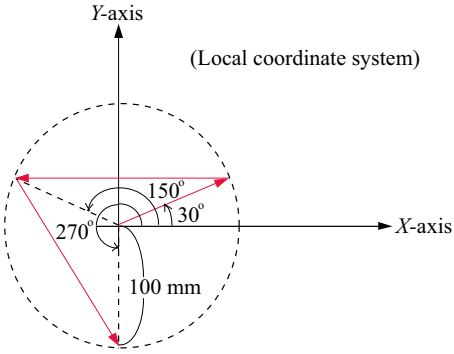


Fig. 2.5 Polar coordinate system programming

the block, the toolpaths in subsequent blocks are rotated by the angle R with respect to the point α, β.

If a workpiece is symmetric with respect to a specific axis, only part of the workpiece need be programmed, the other parts are created using the G51.1 address that utilizes a mirror image function. Figure 2.6 shows an example of usage of the mirror function. The subprogram below is for the path in the upper right side of Fig. 2.6 and the main program below commands the whole path with mirroring of the subprogram.

The subprogram makes the shape on the upper right. This is invoked in the original coordinate system in line N20 of the main program. The following command, on line N30 invokes the mirror function about the symmetry axis X=50. Line N40 then makes the symmetric shape on the top left. Following this, on line N50 the mirror function is again invoked to make the Y=50 symmetric axis. The next line, N60, then calls the subprogram to make the shape on the bottom left. On line N70 the X symmetry axis is revoked using the G50.1 command and the call of the subprogram on line N80 makes the shape at the bottom right. Finally, on line N90 the Y symmetry command is revoked.

2.3.2 Interpolation Functions

There are various interpolation functions that enable machine tools to move the axes along the specific path for multi-axis machine tools. A CNC system provides rapid movement, linear interpolation, circular interpolation, helical interpolation, and spline interpolation functions as interpolation functions.

Main Program
P000001;
N10 G00 G90;
N30 G51.1 X50; → sym. X=50
N40 M98 P100001;

N50 G51.1 Y50; → sym. X=50, Y=50
N60 M98 P100001;

N70 G50.1 X0; → reset X
N80 M98 P100001;
N90 G50.1 Y0; → reset Y
N100 M30;

Sub program
P100001;
N210 G00 G90 X60 Y60;
N230 Y100;
N240 X60 Y60;
N250 M99;

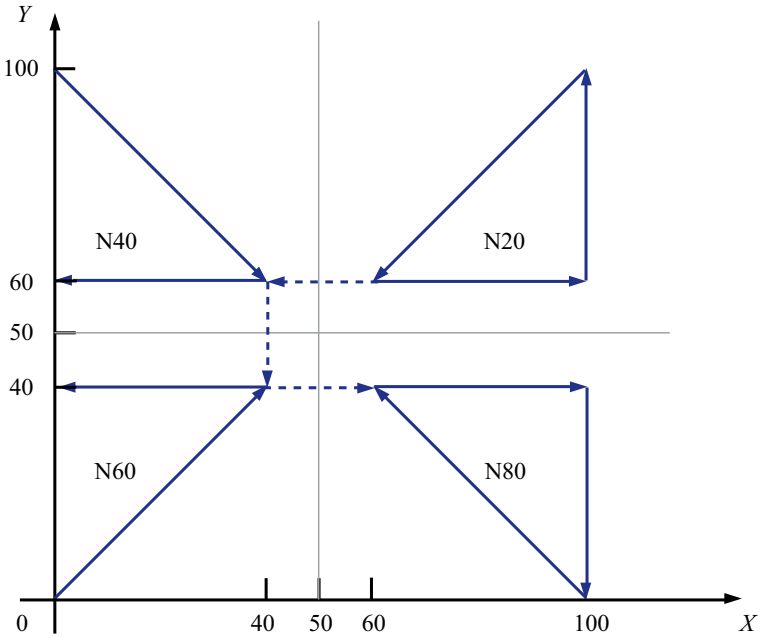


Fig. 2.6 Example of usage of mirror function

The rapid movement function (G00) is used for commanding the specific axes to move rapidly to the programmed position. In the case of an absolute programming mode (G90), this function makes the axes move to the commanded position from the current position. In the case of an incremental programming mode (G91), this function makes the axes move with the commanded incremental value and each axis moves with the specific feedrate defined in the CNC system. Therefore, it is not necessary to set an additional feedrate in G00.

The linear interpolation function (G01) is used for commanding the axes to move the tool along a line with the programmed feedrate, as shown in Fig. 2.7. G01 is a modal G-code and the commanded feedrate is effective until a new feedrate is commanded. Here, the feedrate means the joint speed of the axes.

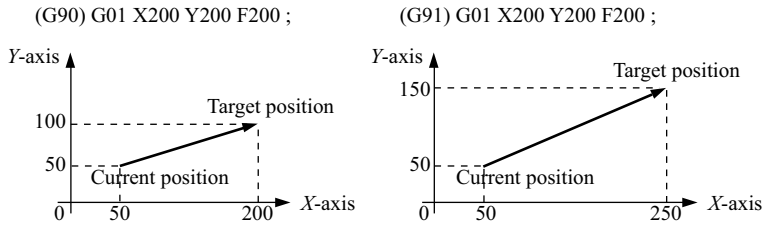


Fig. 2.7 Absolute (G90) and relative (G91) displacements

The circular interpolation function is used to command tool movement along a circle. G02 and G03 can be used for the circular interpolation function. G02 is for commanding circular interpolation in the clockwise direction and G03 is for commanding circular interpolation in a counter-clockwise direction. In order to command this function, the information summarized in Table 2.3 should be provided.

Table 2.3 Circular interpolation information summary

No.	Information		Command	Meaning
1	Plane		G17	Specification of arc on *XY* plane
			G18	Specification of arc on *ZX* plane
			G19	Specification of arc on *YZ* plane
2	Rotation direction		G02	Clockwise (CW) arc
			G03	Counterclockwise (CCW) arc
3	End pos.	G90 Mode	Two in *X*, *Y*, *Z* axes	End position in workpiece coord. sys.
		G91 Mode	Two in *X*, *Y*, *Z* axes	Distance from start to end
4	Distance between start point and the arc center		Two in *I*, *J*, *K* axes	Distance from start to arc center (Sign value)
	Arc radius		R	Radius of the arc
5	Feedrate along the arc		F	Feedrate along the arc

Normally, the rotation direction is defined based on the right-hand coordinate system. That is, if the programmed plane is the *XY* plane, then the CW or CCW directions are defined based on when the *XY* plane is viewed in the positive-to-negative direction of the *Z*-axis. Figure 2.8 shows the individual rotation directions in the cases where the programmed planes are the *XY*, *ZX*, and *YZ* planes.

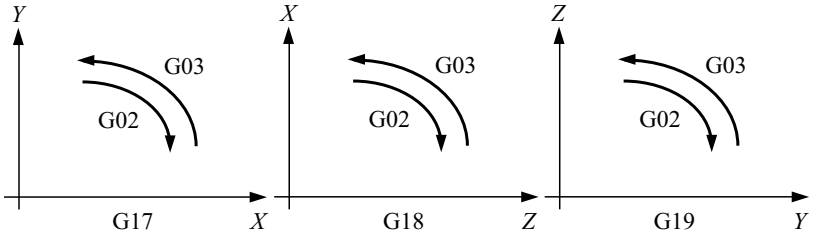


Fig. 2.8 CW and CCW directions for the *XY*, *ZX* and *YZ* planes

The end point of an arc is specified by the address *X*, *Y*, and *Z*, and is expressed as an absolute or incremental value according to whether G90 or G91 mode is current. For an incremental value, the distance of the end point that is viewed from the start point of the arc is specified by the sign value. The arc center is specified by addresses *I*, *J*, and *K* for the *X*, *Y*, and *Z* axes, respectively as shown in Fig. 2.9. The numerical value following *I*, *J*, or *K*, however, is a vector component in which the arc center is seen from the start point, and is always specified as an incremental value irrespective of G90 and G91 as shown below. *I*, *J*, and *K* must be signed according to the direction of the arc.

The arc center can also be specified by using radius *R* instead of addresses *I*, *J*, and *K*. In this case, there are two possibilities, where the arc is less than 180 degrees, or where it is more than 180 degrees. When an arc exceeding 180 degrees is commanded, the radius must be specified with a negative value. The feedrate in circular interpolation is equal to the feedrate specified by the F-code, and the feedrate along the arc (the tangential feedrate of the arc) is controlled by the specified feedrate.

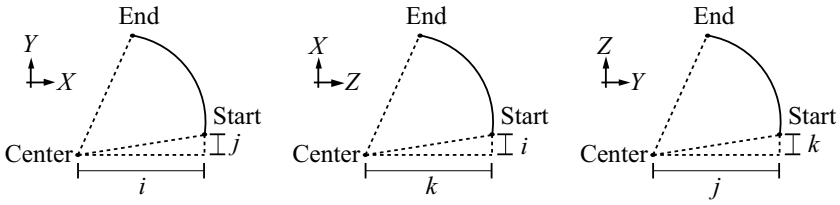
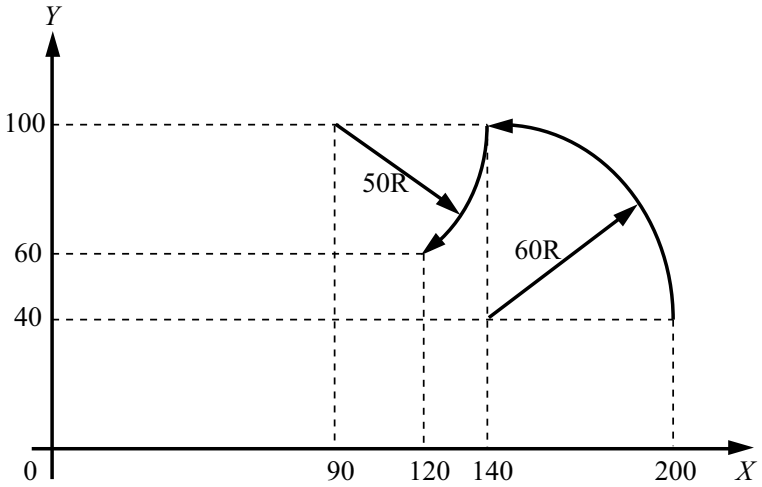


Fig. 2.9 Arc centers for the *XY*, *ZX*, and *YZ* planes

Figure 2.10 shows an actual programming example of circular interpolation in the case of G90 mode and G91 mode, respectively.

Helical interpolation is enabled by specifying up to two other axes that move synchronously with the circular interpolation by circular commands. The tangential feedrate of the arc is specified by an F-code and the feedrate of the linear axis to which circular interpolation is not applied is defined as follows:

Feedrate of linear axis = F * (Length of linear axis)/(Length of circular arc)



i) Absolute programming mode

```
G92   X200   Y40    Z0   ;
G90   G03    X140   Y100   I-60        F300;
      G02    X120   Y60    I-50   ;
or
G92          X200   Y40    Z0   ;
G90   G03    X140   Y100   R60         F300;
      G02    X120   Y60    R50   ;
```

ii) Incremental programming mode

```
G91   G03    X-60   Y60    I-60        F300;
      G02    X-20   Y-40   I-50   ;
or
G91   G03    X-60   Y60    R60         F300;
      G02    X-20   Y-40   R50   ;
```

Fig. 2.10 Absolute and incremental circular interpolation

Cylindrical interpolation, which is useful for slotting and CAM machining on a cylinder, is a function where the amount of movement of the rotary axis, specified by an angle, is converted to the amount of movement on the circumference to allow linear interpolation and circular interpolation with another axis. For cylindrical interpolation, the development surface of the cylinder is regarded as a 2D shape and is programmed as shown in Fig. 2.12.

When machining the specified shape on a cylindrical surface, the use of the 2D developed surface on the C-axis and the Z-axis and cylindrical interpolation makes part programming easy.

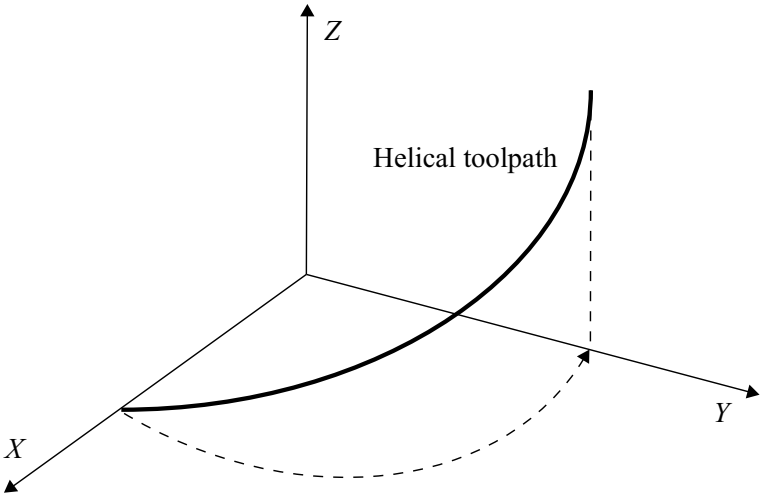


Fig. 2.11 Helical toolpath

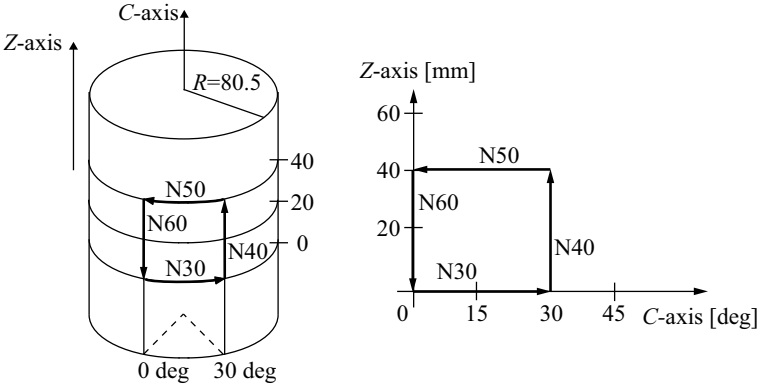
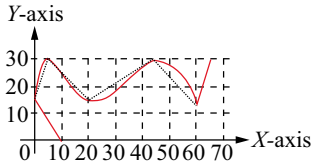


Fig. 2.12 Cylindrical interpolation

Spline interpolation (G06.1) is used for machining free-form curves or surfaces and enables the tool to be moved along the interpolated curve that passes through the specified points, as shown in Fig. 2.13. Spline interpolation is canceled by commanding another G-Code (*e.g.* G00, G01, G02, G03) that belongs to the same G-code group.

The typical type of spline interpolation is NURBS (Non Uniform Rational B-Spline) interpolation and the details of NURBS interpolation will be described in Section 3.5.




```
N10  G17 G01 X10 Y0 F200 ;
N20  X0 Y15 ;
N30  G06.1 X5 Y30 ;
N50  X20 Y15 ;
N50  X45 Y30 ;
N60  X60 Y15 ;
N70  G01 X65 Y30 ; → Spline interpolation is canceled
N80  M30 ;
```

Fig. 2.13 Spline interpolation

2.3.3 Feed Function

The feed function is used for controlling the feedrate of axes and rapid movement, machining movement, path control mode (*e.g.* exact stop mode and continuous mode), and dwell function belong to this function. The feedrate, specified by the F-code, can be programmed as feed per min (mm/min or inch/min) or feed per revolution (mm/rev or inch/rev).

The rapid traverse function is used for moving the tool quickly to the commanded position and the feedrate for rapid movement is specified in the CNC system. Machining feedrate means the feedrate specified for linear interpolation or circular interpolation.

To prevent a mechanical shock, acceleration/deceleration is automatically applied when the tool starts and ends its movement. Furthermore, when the movement direction is changed between a specified block and the next block during cutting feed, the toolpath may be curved due to the relationship between the time constant of a servo system and the commanded feedrate. In the CNC system, linear, exponential, and S-shape acceleration/deceleration profiles, shown in Fig. 2.14, have been typically used. Each profile provides its specific characteristics in its own way. In general, the linear acceleration/deceleration profile has been widely used and enables the axis to reach at the commanded feedrate rapidly, in a simple way. Note, though, that the S-shape profile makes the axis movement smooth and has been widely used for high-speed machining.

Automatic acceleration/deceleration is very useful for preventing mechanical shock. However, it results in a servo delay due to the shift of speed profile by the acceleration/deceleration time constant and, finally, causes machining error. In particular, due to the machining error caused by automatic acceleration/deceleration of circular interpolation, the radius of the machined circular path comes to be smaller than that of the programmed circular path. The machining error is in inverse proportion to the radius of the circle being interpolated and in proportion to the square of the commanded feedrate.

As the command method to control the speed at the corner between the specific block and the next block, Exact Stop (G09), Exact Stop Mode(G61), and Cutting Mode (G64) can be used.

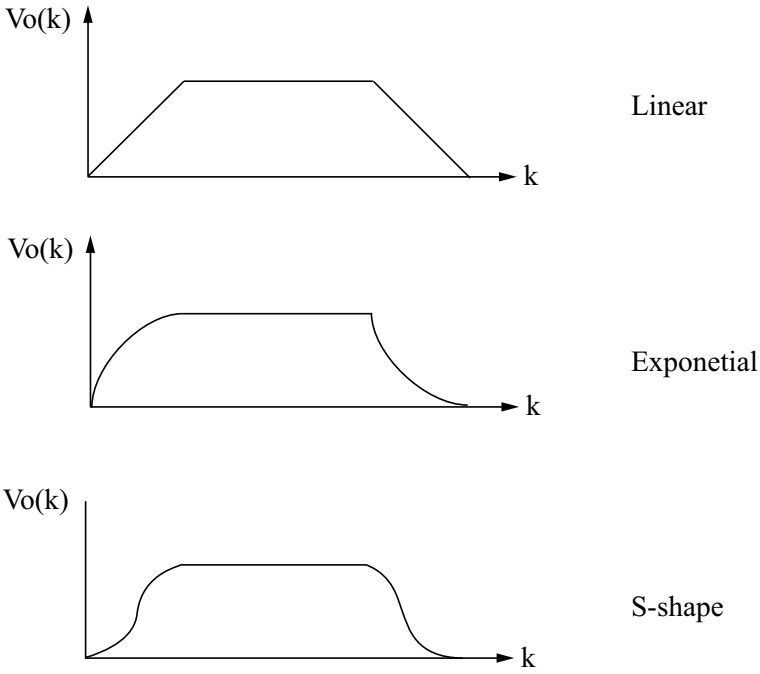


Fig. 2.14 Acceleration/deceleration profiles

In the block where Exact Stop(G09) is valid, the tool is decelerated at the end point of the block, then an in-position check is made. Under the rapid traverse movement, the tool is decelerated at the end point and an in-position check is made regardless of whether or not the command Exact Stop has been issued. When the Exact Stop Mode is specified, the tool is decelerated at the end point of a block, then an in-position check is made. This mode is valid until G62, G63, or G64 is specified and is used for making a right angle at the corner of a toolpath.

However, after Cutting Mode (G64) is specified, an in-position check is not made at the end point of the next blocks. In modern CNC systems, a Look-Ahead function is executed under Cutting Mode and this function is useful for increasing the actual machining feedrate during execution of the part program which consists of small line segments. The details of Look-Ahead function will be described in Chapter 3.

The dwell function (G04) is used for delaying the next execution block for the specified time interval. As this code is a one-shot G-code, it is valid only during the block where the function is commanded.

The threading function (G33) is used for the machining tapered threads and threads with a constant lead. When single screw threads are machined, the threading tool moves several times along the same path from roughing to the finishing process. For this thread cutting, the thread tool is started after detecting one revolution signal from the position coder attached to the spindle. Therefore, the start position

of threading is always identical in spite of repeating machining. In this way, it is possible to machine a single thread.

When multi-screw threads are machined, the start angle of threading is changed. If the angle is changed by 180 degrees, a double screw thread can be machined. If the change angle is 120 degrees, a triple screw thread can be machined. To machine multi-screw threads, the spindle speed is read from the position coder and the speed read is converted into the feed per minute value. The tool is moved based on the converted feedrate and the feedrate is identical during threading. However, if the feedrate calculated from the detected spindle speed exceeds the maximum allowable feedrate, the actual feedrate becomes smaller than the required feedrate and it becomes impossible to machine the thread with the required lead.

2.3.4 Tools and Tool Functions

The tool function (T-code) is used for selecting the machining tool with the specified tool number. The specified tool is effective until another tool is selected.

The tool life management function is used for managing the usage time and wear amount of each tool and the number of the part that is machined by each tool. This provides functions to replace the particular tool with a specified spare tool in the case when the usage time of the particular tool exceeds the pre-specified time or the number of parts machined by the particular tool exceeds the predefined number.

The tool radius compensation (G40, G41 and G42) functions are used for generating a path that is offset from the programmed path by the radius of tool. As shown in Fig. 2.15, the path followed by the tool center should be the path indicated by B, which is separated from A by the value R, in the case when a part, indicated by A, is machined by a tool with radius R.

Typically, the distance by which the tool is separated from the programmed path is called the "offset" and B in Fig. 2.15 is an offset path. The code G41 commands tool-radius compensation to the left of the tool movement direction, G42 commands tool-radius compensation to the right of the tool movement direction, and G40 commands cancelation of tool radius compensation. Tool compensation codes such as G41 and G42 are used with a D address that stores the tool offset value and the tool offset value is pre-specified by user.

Tool-radius compensation is applied differently according to the following modes.

1. *Cancel mode*: After power is turned on, the CNC system is reset, or M02/M30 is executed, the status of the CNC system turns into Cancel mode. In this mode, tool compensation mode is canceled and the path of the tool center point is the same as the programmed path.
2. *Start-Up mode*: If G41 or G42 is commanded in Cancel mode, the CNC system turns into Offset mode. (Figure 2.16).
3. *Offset mode*: This means the CNC operating period between the first block after declaring the tool radius compensation to the last block before canceling the tool-

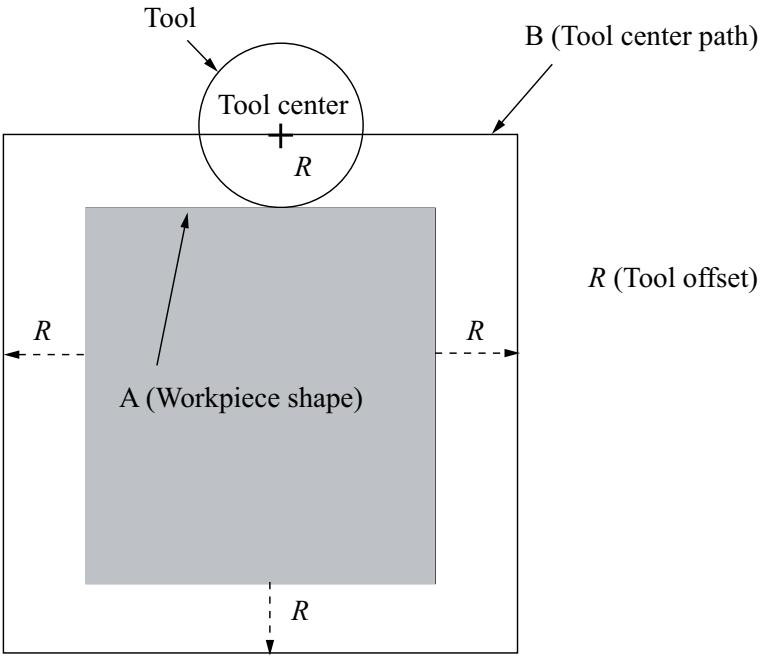


Fig. 2.15 Programmed path and offset path

radius compensation. During Offset mode, the offset path of the path programmed in each block is calculated and the real machining path is made by connecting these individual offset paths. (Figure 2.16)

4. *Offset Cancel mode*: In the case of commanding G40 during the Offset mode, the tool radius compensation function comes to be canceled. (Figure 2.17).

The tool-length compensation function is for compensating the difference between the pre-defined reference tool-length and the actual tool-length. This function is useful when the tool-length defined when editing a part program is different from the actual machining tool-length. Accordingly, it is possible to make the part program without knowing the actual machining tool-length. G43 and G44 are the codes for commanding tool-length compensation, G43 and G44 denote the tool-length increase and tool-length decrease, respectively. They use the value specified by the H address as the compensation amount, which is pre-specified by the user. The cancelation of tool-length compensation is specified by G49.

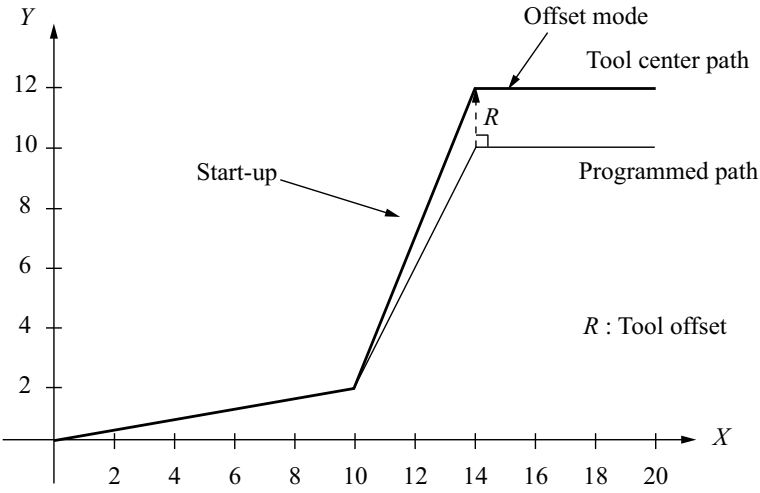


Fig. 2.16 Interpolation for start-up and offset modes

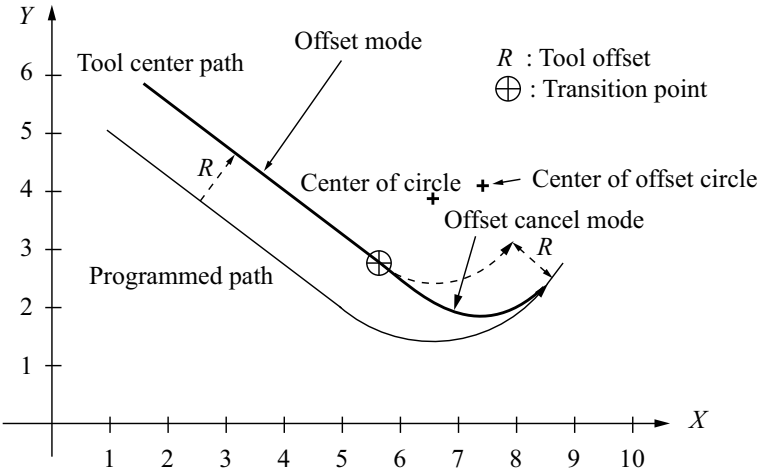


Fig. 2.17 Interpolation for offset and offset cancel modes

2.3.5 Spindle Functions

The spindle function (S-code) is for specifying the spindle speed and the spindle speed is restricted by the maximum spindle speed specified by user. The S-code is modal code and, therefore, the spindle speed specified by the S-code is effective until another spindle speed is specified. The spindle speed specified by an S-code is canceled after power on, or when the system is reset or when M30 is commanded. During execution of a part program, change of spindle speed is limited to being less than or equal to the specified maximum spindle speed.

The constant surface speed control function is used for rotating the spindle with constant surface speed regardless of the position of the tool. This function is applied for turning and the surface speed for this function is specified by the S-address. For this function the axis along which constant surface speed control is applied should be specified. To command constant surface speed control, the G96 command is used, and to cancel the constant surface speed control the G97 command is used.

Typically, the spindle connected to the spindle motor is rotated at a certain speed to rotate the workpiece mounted on the spindle. This spindle control status is referred to as spindle rotation mode. In addition to spindle rotation mode, the spindle position function, which turns the spindle through a certain angle, can be used to position the workpiece mounted on the spindle at a certain angle. Also, as the spindle orientation function is one of the spindle position functions, the spindle orientation function can be used to make the spindle stop at a pre-determined position. By specifying the particular angle using the S-code, it is possible to stop spindle at a particular angle. An example of the use of the spindle orientation function is given below.

N20 M03 S1000 ;
N30 M19 ; → spindle stops at 0.
N40 M19 S270 ; → spindle stops at 270
N50 M03 ; → spindle begins rotating in 1000 rpm in clockwise direction.
...

2.3.6 Fixed-cycle Function

The fixed-cycle function is used for executing specific machining for which more than one block is necessary using only one block. This is useful for simplifying a part program and the fixed-cycle code has been defined for a variety of machining in drilling, turning, and milling as shown in Table 2.4. The usage example of this function will be explained by using fine boring that is one of the cycle codes for drilling.

As shown in Fig. 2.18, the fine boring cycle command, G76, moves a tool to the reference position and stops. This command rotates the tool to a reference angle by commanding the spindle orientation function, moves the tool by a specified amount

Table 2.4 Operations and fixed cycle function codes

Operation		G-code	Operation		G-code
Drilling	Peck Drilling	G73	Turning	Roughing	G90
	Reverse Tapping	G74		Threading	G92
	Fine Boring	G76		Face roughing	G94
	Cycle Cancel	G80		Finishing	G70
	Drilling Cycle, Spot Boring	G81		Roughing	G71
	Drilling Cycle, Count Boring	G82		Face roughing	G72
	Peck Drilling	G83		Copying	G73
	Tapping	G84		Grooving	G74
	Rigid Tapping	G84.2		Face grooving	G75
	Reverse Rigid Tapping	G84.3		Multiple threading	G76
	Boring	G85	Milling	Circular Elongated Holes	
	Boring	G86		Circular	
	Back Boring	G87		Circumferential Slot	
	Boring	G88		Facing	
	Boring	G89		Circular Pocket	

to the opposite side of the tool cutter, and finally retracts the tool upwards to avoid damage to the machined surface.

The detailed procedure for the fine boring cycle function is given below.

1. The tool is moved to the cut start position.
2. The tool is moved rapidly to the R position.
3. With the tool movement to the Z position, boring is carried out.
4. If G76 is commanded with P address, the dwell function is executed.
5. The spindle orientation function (M19) is executed.
6. The tool is moved rapidly by the amount specified with the Q address along the direction specified by the parameter. (In this example, it is assumed that the *XY* plane is selected as the machining plane and, therefore, the tool can be moved along the *X*-axis or *Y*-axis.)
7. The tool is rapidly retracted to the specified position. If the G99 code is effective, tool movement position becomes the R position and if G98 code is effective, the tool movement position becomes the cut start position.
8. The tool is moved rapidly by the length specified with the Q address to the opposite direction pre-defined by parameter.
9. Tool rotation starts again.

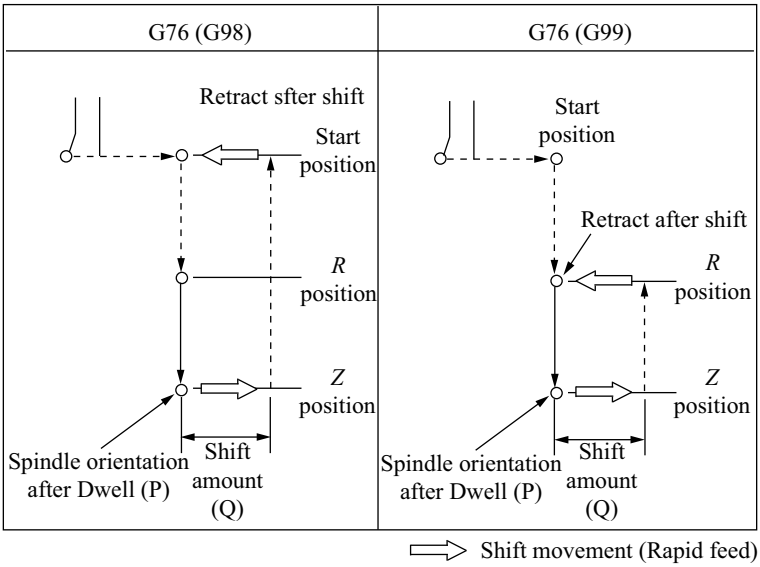


Fig. 2.18 Fine boring cycle movements

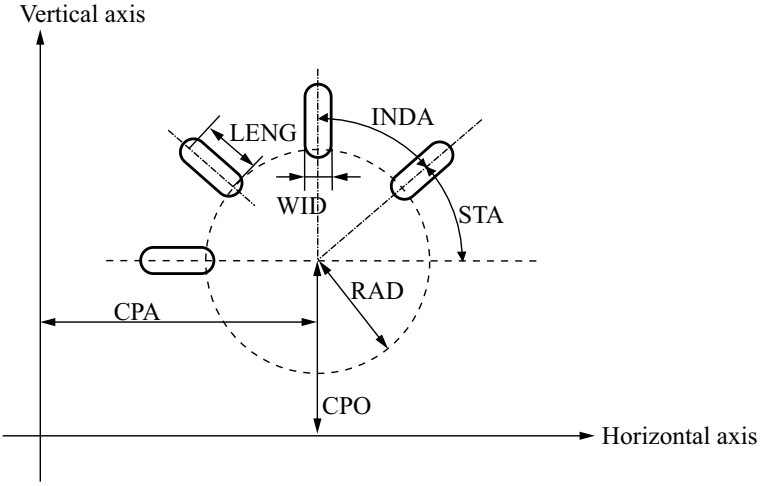


Fig. 2.19 Circular slot cycle - circular pattern of slots

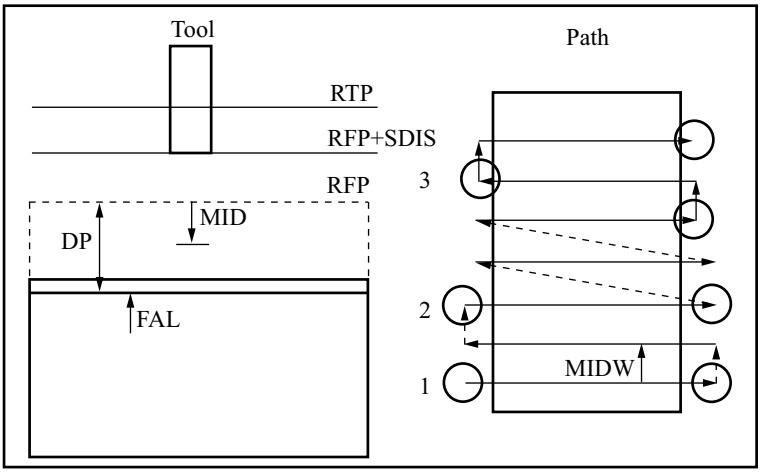


Fig. 2.20 Face milling pattern and parameters

2.3.7 Skip Function

During the execution of the skip function (G31), if an external skip signal is input, execution of the command is interrupted and the next block is executed. The skip function is commanded with linear interpolation such as G01. The skip function is used when the end of machining is not programmed but specified by a signal from the machine, for example, in grinding. It is used also for measuring the dimensions of a workpiece.

Figure 2.21 shows an example of the actual toolpath after the skip signal is detected in the case when absolute command mode is effective and the programmed path is on the *XZ* plane. As soon as a skip signal is detected, the tool (in this case a touch probe is generally used) is moved to the end point of the next block regardless of whether or not the tool reaches the end point of the current block. The feedrate of the linear path commanded by the skip function is specified by the F-address or a certain parameter and this feedrate is effective only on the linear path commanded by the skip function.

2.3.8 Program Verification

The part program edited by the machine tool operator is likely to include grammatical errors, logic errors, and numerical errors, such as incorrect computation of tool position, wrong tool-offset value, and invalid feedrate and spindle speed. Therefore, it is necessary to test the part program before executing it and the CNC system generally provides the functions listed below for immediate validation.

G90 G31 X200.0 F100 ;
X300.0 Z100.0 ; ⟶ The tool is moved to the point of the next block.

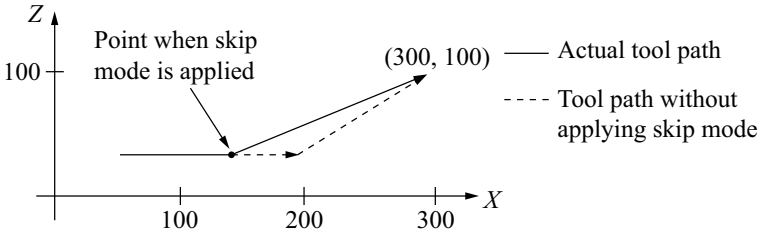


Fig. 2.21 Skip function action

1. Dry Run: During dry run mode, the tool is moved at the feedrate specified by a parameter regardless of the feedrate specified in the program. This function is used for checking the movement of the tool in the case where the workpiece is removed from the table. The tool moves at the feedrate specified by the parameter. The feed override switch can also be used for changing the feedrate during this mode but during automatic mode, dry run is not allowed to begin.
2. Pressing the single-block switch starts single-block mode. When the cycle start button is pressed in single-block mode, the tool stops after a single block in the program is executed. This function is used for checking the program block-by-block and can be used with the dry run function and machine lock function.
3. Machine lock is used to display the change in position without moving the tool and there are two types of machine lock: all-axis machine lock, which stops movement along all axes, and specified-axis machine lock, which stops movement along specified axes only.

2.3.9 Advanced Functions

Recently, CNC machine tools have become more accurate and faster and the functionality has become more complicated. To satisfy these requirements, advanced functions for high-speed and high-accuracy machining have been developed and applied in addition to the functions mentioned in the previous sections. The next sections describe typical advanced functions built into highly functional CNC systems.

2.3.9.1 Look Ahead

Generally, the part program for surface machining (die and mold is a typical example of surface machining) consists of a sequential linear path with short length and fast feedrate. In this case, if each block is executed line by line, the actual feedrate

becomes less than the programmed feedrate and the feedrate at the corners between one specific block and the next becomes discontinuous.

Therefore, the quality of the machined surface is degraded due to frequent acceleration/deceleration and the discontinuity of the feedrate and, after completing machining, grinding becomes essential. To solve this problem, the Look-Ahead function was developed. The look-ahead function looks ahead a hundred blocks and calculates an adequate feedrate for each axis within the maximum allowable feedrate and acceleration/deceleration.

With this function, it is possible to machine the free-form surfaces and contours of a complicated shape without stopping tool movement between successive blocks at high speed. The concept of the look-ahead function can be easily understood by comparing it with car driving. At night, it is difficult for the driver to see for long distances and, therefore, it is difficult to drive at the maximum allowable speed. However, during the day, a driver can see longer distances and, therefore, it is possible to examine the road status, predict maximum feasible driving speed, and, finally, to drive faster.

The look-ahead function calculates the maximum feasible feedrate of the specified block based on the interpreted result of the blocks that will be executed. This function requires much computing power. Recently, with the advance of CPU power, the number of the blocks that can be used for look ahead has grown to a thousand. Figure 2.22 shows the feedrate profiles when the look-ahead function is applied and when it is not and Fig. 2.22 also shows that the look-ahead function can increase the actual feedrate.

When the look-ahead function is applied, the feedrate at the end of the starting block (N1) is not decelerated and the programmed feedrate is kept to the programmed feedrate. To stop at the end position of the last block (N12), the deceleration of the feedrate starts in the preceding blocks. Therefore, the look-ahead function enables high-speed machining compared to exact stop mode where acceleration and deceleration is done at the start point and the end point of each block. Accordingly, with this function, reduction of machining time becomes possible.

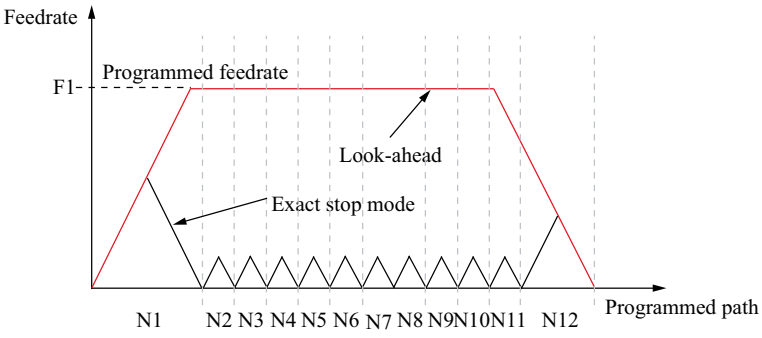


Fig. 2.22 Look-ahead mode and Exact stop profiles

2.3.9.2 Feedforward

The conventional position control method essentially has the following error and it is proportional to the square of the feedrate during high speed machining.

The cause of the control error is mainly based on the servo delay. In order to reduce the machining error, it is necessary to increase the position control loop gain. However, increasing the position control loop gain is likely to result in machine vibration and make the servo system and the machine unstable. Accordingly, as the feedforward control method plays the role of making the servo system stable and increasing the position control loop gain, it makes it possible to reduce the machining error and achieve high-speed and high-accuracy machining.

Figure 2.23 shows the actual feedrate profiles and path traces when the feedforward control method is applied and when it is not applied. From Fig. 2.23, we can see that when the feedforward control method is applied, the following error decreases and the machining error obviously also decreases.

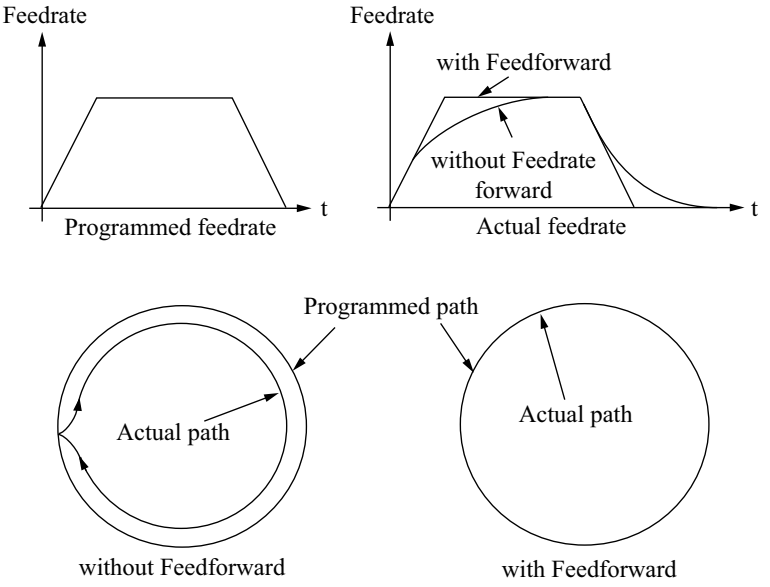


Fig. 2.23 Feedrate profiles and path traces with and without feedforward control

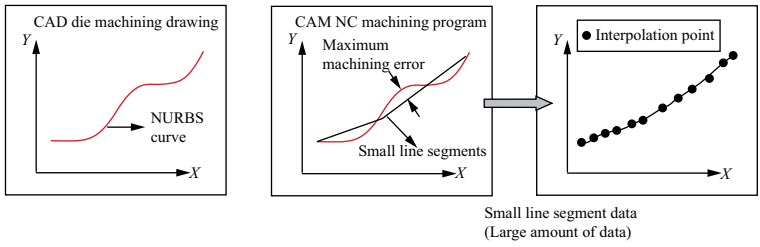
2.3.9.3 NURBS Interpolation

As high-speed machining and high-accuracy machining come to be generally used, the requirements for advanced functions to support them is growing. In particular, when conventional CNC systems (where free curve is defined by sequential small

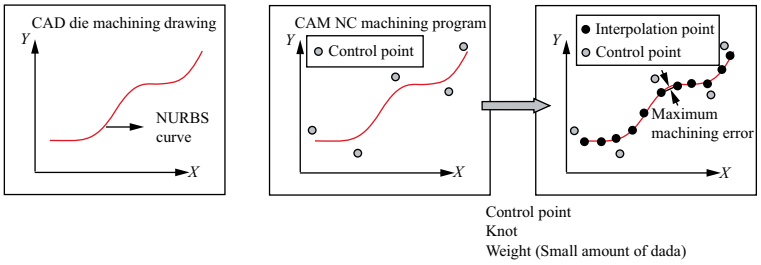
line segments or arcs) are used for machining free-form surfaces, the tool moves in a discontinuous manner and this makes the quality of the machined surface poor. Also in this case, because a lot of program blocks are required, the size of the part program is large. Because the size of the internal memory of CNC system is limited, DNC (Direct Numerical Control) mode has typically been used to machine free-form surfaces. Since the baud rate of DNC communication is restricted it becomes impossible to raise the machining speed over a restricted specific value when a conventional CNC system is used. To overcome this problem NURBS interpolation was developed. In this section, the necessity of NURBS interpolation will be described and the details will be given in Chapter 3.

In NURBS interpolation, NURBS curve data (*e.g.* control points, weights, and knot vector) are directly input to the CNC system instead of the small line segment data that are defined by the G01 command. As the CNC system generates interpolation points based on the NURBS curve data, the programmed feedrate and the tolerance, it makes it possible to perform high-speed and high-accuracy machining.

Figure 2.24 shows the difference between the interpolation methods based on line-segment approximation and a NURBS curve. When offline CAM software is used the free-form curve geometry is approximated to within a pre-defined tolerance by a set of line segments. These in turn are then subdivided into a set of shorter line segments to give the desired feedrate. With direct NURBS interpolation in the CNC the interpolation the feedrate and tolerance are used to determine the step length along the curve directly to give the required speed profile.



(a) Interoperation method based on small line segment approximation



(b) Interoperation method based on NURBS curve

Fig. 2.24 Indirect and direct NURBS interpolation

2.3.9.4 NURBS Surface Machining

For free-form surface machining, recent CAD/CAM systems include a function to transmit the free surface data into the CNC system using the NURBS surface form. The G-code format to specify the NURBS surface data is different according to the CNC maker. However, despite the differences in the G-code representation method, the data elements for representing a NURBS surface are the same. Table 2.5 summarizes the status of development of NURBS interpolation functions for different CNC makers.

Table 2.5 Controller NURBS development summary

	FANUC	SIEMENS	OKUMA	Mitsubishi	Toshiba Machine
CNC Model	15 Series, 16 Series, 18 Series, 30 Series	840D	OSP700M (spar H1 CNC)	M700	TOSNUC 888
CPU	64bit RISC	RISC	RISC		
G-code	G06.2	Language Type: BSPLINE	G132	G70.0 G70.1	

Figure 2.25 shows the G-code format for representing a NURBS curve and Fig. 2.26 shows an example of a part program to machine a NURBS curve profile.

```
G06.2        P_X_Y_Z_R_K_F ;
        G06.2 : NURBS interoperation
        P : NURBS curve order
        X, Y, Z : Control point
        R : Weight
        K : Knot
        F : Feedrate
```

Fig. 2.25 FANUC system NURBS G-code format

In Fig. 2.26, in the block whose line index is 110, the degree of the NURBS curve and the feedrate are specified. From the block whose line index is 110 to the block whose line index is 350, the control points and knot vector are specified. In the case of the NURBS curve defined in Fig. 2.26, the degree of the curve is 4, the feedrate is 10 mm/min, and the weight of all control points is 1.

2.4 G&M-code Interpreter

As mentioned in the previous section, the interpreter of the CNC system is the software module of the NCK unit that interprets the part program consisting of G&M-code commands and related addresses such as S, T, and F. The interpreter consists of a parser, an executor, a path generator, a macro executor, and an error handler. The parser consists of a lexical analyzer, a calculator, and a sentence interpreter (YACC). As for the software modules connected with the interpreter, there are the program access module, which reads a program file, and the interpolator module, which generates the interpolated points of the programmed path based on the interpreted data. Figure 2.27 shows these modules in graphic form.

```
N100    G05 P10000
N110    G06.2 P4 K0. X-1.6953
        Y-.75 Z-.2358 F10
N120    K0. X-1.6544 Z-.2313
N130    K0. X-1.5752 Z-.2225
N140    K0. X-1.4053 Z-.2067
N150    K.0313 X-1.3031 Z-.1982
N160    K.0781 X-1.1215 Z-.1847
                ...
N300    K.9063 X1.7085 Z-.2373
N310    K.9688 X1.75 Z-.2421
N320    K1.
N330    K1.
N340    K1.
N350    K1.
N360    G01 Y-.7188 Z-.238
```

Fig. 2.26 NURBS-profile G-code part program

The functionality of each module is given below.

1. *Parser*: this module interprets the part program block by block. The lexical interpreter of this module reads the block character by character and makes meaningful words from the characters. The calculator carries out numerical operations within the part program. The sentence interpreter retrieves the command and the related data such as G-code, M-code, S-code, T-code, conditional branching, and iteration loop based on the words from the lexical interpreter.
2. *Executor*: this module executes the functions related with the interpreted sentence and stores the execution result in the internal memory. In addition, this module generates the data required for executing the modal code.
3. *Path Generator*: This module generates the position data based on the programmed coordinates. In this module, the computation for mapping from work-

piece coordinates to machine coordinates, tool compensation, and the axis limit is carried out.

4. *Macro Executor*: this module interprets and executes macro commands included in an NC part program. As the macro is user-defined code, the user can make specific functions that are not provided by the CNC maker by using a macro language, which is similar to the BASIC language.
5. *Error Handler*: if there is an error in a part program, the error should be noticed and the user notified. This module is responsible for this.

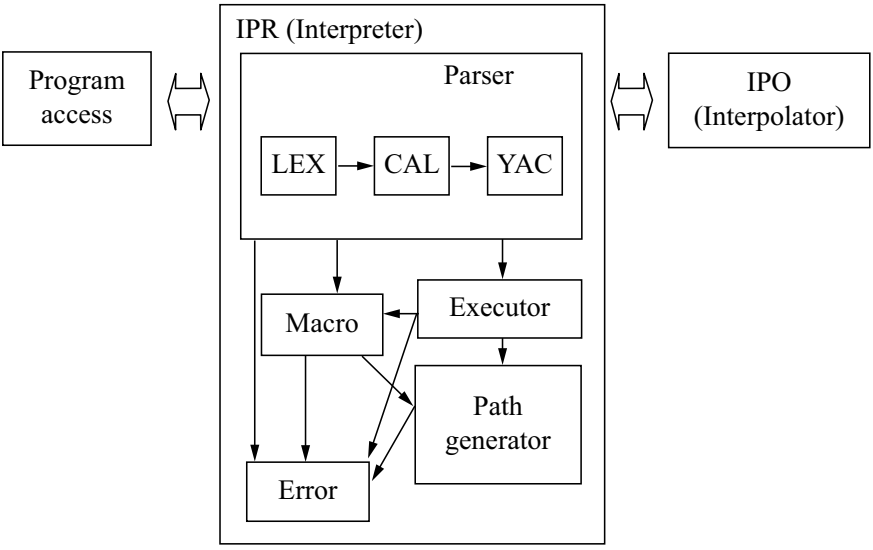


Fig. 2.27 Code interpreter modules

The workflow that should be executed by the interpreter with the various S/W modules from interpreting the part program to generating the tool position is shown in Fig. 2.28.

First, once the Cycle Start button on the MMI panel has been pushed, the interpreter starts pre-processing tasks such as reading the part program into the internal memory of the CNC system block by block, interpreting the block, and storing the interpreted data in the internal memory. During the pre-processing tasks, a cycle code is converted into blocks that consist of G01, G02, and G03. These converted blocks replace the cycle code and the interpreter reads and interprets the converted blocks. The internal block memory can be defined as in Table 2.6. The block memory shown in Table 2.6 shows the memory contents when the block whose line index is N300 has been read and where the block skip is specified and interpreted in subprogram P9000. The interpreter reads the addresses and the following numbers specified in the N300 block and stores the interpreted values in the related internal block mem-

ory. In addition, the SLASH variable that denotes whether a block skip command is on or not is set to true.

Next, the interpreter reads and performs the data on internal block memory. If the codes M02 or M30 are executed, the part program is 'rewound'. If the interpreter executes M98, the interpreter calls the subprogram whose number is the value following the P address. If M98 is commanded together with the L address, the interpreter executes the subprogram repeatedly as many times as the number following the L address.

Further, a macro call is monitored and, if it is called, the specified macro program is called. To make macro execution fast, it is more efficient to pre-interpret the program called by the macro, store the program as intermediate code, and execute this. As this method is one of the high-speed interpreting solutions, the definition of the virtual intermediate code is necessary and an additional execution module is required. How to call a macro program is similar to how to call a subprogram. In a macro program call, though, it is possible to specify arguments and use the global variables in contrast to execution of a subprogram call.

The next step is to execute the G-code. If the current block is the first block, the interpreter initializes the local variables and controls the optional block skip command. It also performs the G-code processing such as setting the G-code type, G-code group, modal data, and non-modal data.

The interpreter performs F-code processing where the interpreter reads the F-code data from the internal block memory and specifies the related variable. During F-code processing, the feedrate definition method such as feed-per-minute (mm/min, inch/min) or feed-per-revolution and the unit of feed-override and dwell time are set.

After completion of the above-mentioned stages, the end position of a block is computed as the final step of the interpretation. Generally, as stated earlier, various coordinate systems are used to make editing of the part program easy for machine tools with CNC systems.

There are three kinds of coordinate system used for machine tools; a machine coordinate system that is the reference coordinate system of machine tool itself and is defined with zero return, a workpiece coordinate system that is used for editing of a part program, and a local coordinate system that is used as a reference for machining and is specified based on the workpiece coordinate system. The machine coordinate system is specified by G53, a workpiece coordinate system is specified by G54 to G59, and a local coordinate system is specified by G52.

Accordingly, based on the above coordinate systems, for computation of the output position of the block, first, if necessary, values in inches are converted into values in millimeters. Incremental values are converted into absolute coordinate values. If rotation of the coordinate system is commanded, the programmed coordinate is rotated. If mirroring of the coordinate system is specified, then the mirror function is executed. If scaling of the coordinate system is specified, then the scaling function is performed.

The next stage is the path-modification stage where tool-length compensation and tool-radius compensation are executed. Finally, the data used for the check limit

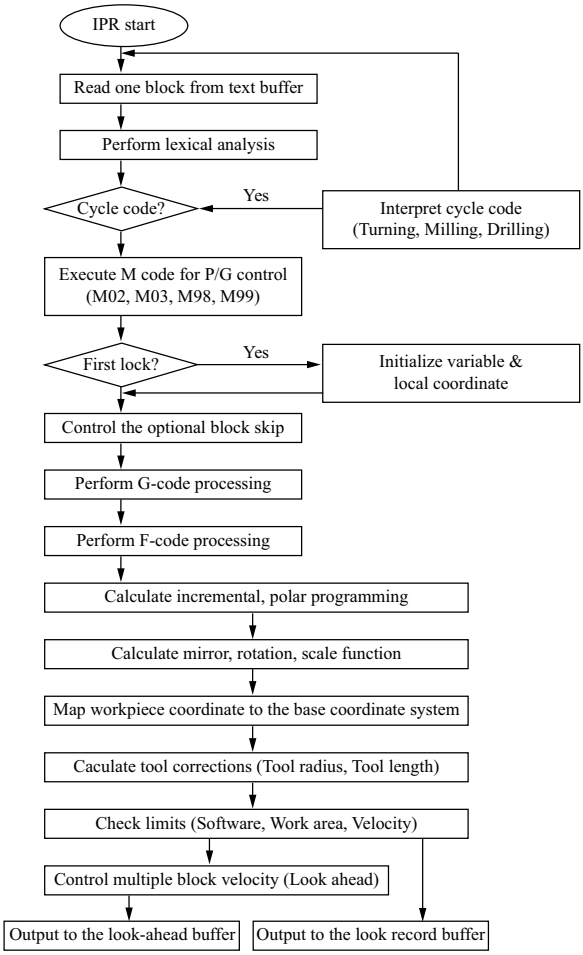


Fig. 2.28 Flowchart for interpreter function

value such as velocity, software, and work area are transmitted into the interpolator. Table 2.7 shows an example of the implementation of the block record memory.

The block record memory stores not only the interpreted data of a specific block but also the data specified in the CNC system. If the look-ahead function is used, the feedrate of the block is recalculated using the look-ahead function and the modified feedrate is stored in the additional memory for look-ahead function. The details of the look-ahead function that is used for preventing reduction of the feedrate for sequential small line segments will be described in Chapter 3.

Table 2.6 Memory map for the interpreter

Field	Type	Value	State	Type	Value
A	double		SLASH	unsigned	1
B	double		PERCENT	unsigned	
C	double		NUMBER	unsigned	
D	double		ID	unsigned	
E	double		CHANGE[M_ADD]	unsigned	
F	double	1000	GFLAG[MAX-G]	unsigned	
G	double	01	MFLAG[MAX-G]	unsigned	
H	double	G_MODAL	int		
I	double				
J	double				
K	double				
L	double				
M	double				
N	double	300			
O	double				
P	double				
Q	double				
R	double				
S	double	5000			
T	double				
U	double				
V	double				
W	double				
X	double	150			
Y	double	20			
Z	double	30			

P9000
N100 G92 G96;
N200 G00 X100;
N300 M03;
/N300 G01 X150 Y20 Z30 F1000 S5000;
N400 G1 Z120;

2.5 Summary

The interpreter plays the role of converting a user-edited part program into the internal data format for execution. In order to understand the structure and the internal behavior of the interpreter, it is necessary to understand the structure of a part program and the commands used therein.

In a CNC system, various coordinate systems, such as the machine coordinate system, workpiece coordinate system, and local coordinate system, are supported for the convenience of editing a part program and setting up the machine. Also, rotation, mirroring, and scaling of a coordinate system are provided and by using these functions it is possible to easily edit the part program.

Table 2.7 Block record memory

Entity	Type	Comments
End_point	Real Array[8]	Block end point
Start_point	Real Array[8]	Block start point
Prog_F_value	Real	Feedrate
Prog_S_value	Real	Spindle speed
F_limit	Real	Feedrate limit
S_limit	Real	Spindle speed limit
Scaling_factor	Real	Scale
G_code_group	Integer Array[20]	G-code group
Exact_stop	Integer	Exact stop mark
Block_number	Integer	Block number
Sub_PG_index	Integer	Subprogram index
Act_Number_PG_repeat	Integer	Number of program repetitions
Gear_box	Real	Parameter for gear box
Correction_active	Integer	Tool compensation start
Tool_correction	Real	Tool compensation
IPO_type	Integer	Interoperation type
G_Code	Integer Array[8]	G-code in block
Thread_flag	Integer	Thread start
Handwheel_flag	Integer	Handwheel start
Handwheel_direction	Integer	Handwheel rotation direction
Block_length	Real	Block length
Block_pointer	Pointer Array[4]	Block pointer

To control tool movement along a line, an arc, a helical, or a spline path, interpolation functions such as G01, G02, or G03 code, F-code for specifying the feedrate, and S-code for specifying the spindle speed are used. To carry out the part program where the tool shape and assembly are not considered, the tool-radius compensation function and tool-length compensation function are provided. Furthermore, the macro function, the so-called "cycle function" is provided for convenience of editing a part program and simplification of the part program. Recently, to fulfill requirements for high-speed machining and high-accuracy machining, various advanced functions such as the look-ahead function, feedforward control function, and NURBS interpolation function have been applied.

Finally, the interpreter, which performs the above-mentioned functions, consists of a parser, an executor, a generator, a macro executor, and an error handler. The interpreter converts the block data read from the text memory into the internal data structure. Based on the interpreted data, the position of a block is computed by executing various mathematical operations such as the coordinate rotation and tool compensation and is stored in the block record memory.

Chapter 3
Interpolator

The interpolator plays the role of generator of axis movement data from the block data generated by the interpreter and is one of the key components of CNC, reflecting its accuracy. In this chapter, the various software interpolators will be introduced and their strengths and weaknesses will be described. In addition, a NURBS interpolator, which is an advanced interpolation method, will be described and the implementation algorithm will be introduced.

3.1 Introduction

A CNC machine generally has more than two controlled axes to machine complex shapes. Two kinds of control can be carried out: The point-to-point control method is used to move the axis to the desired position; and the contour control method is used to move the axis along an arbitrary curve.

In order to execute these control methods successfully, tool movement should be divided into components corresponding to each axis; the locus of the tool is created through combining the individual displacements for each axis.

For example, if a tool should move from point $P1$ to $P2$ at feedrate V_f in the XY plane, as shown in Fig. 3.1, the interpolator divides the overall movement into individual displacements along the X- and Y-axes based on the pre-defined feedrate. Finally, the velocity command blocks for the two axes are generated as shown in Fig. 3.1.

Therefore, the interpolator requires the following characteristics so that it can generate the displacement and speed successfully for multiple axes from the part shape and the pre-defined feedrate.

1. The data from the interpolator should be close to the actual part shape.
2. The interpolator should consider the limitation of speed due to the machine structure and the servo specifications while calculating velocity.

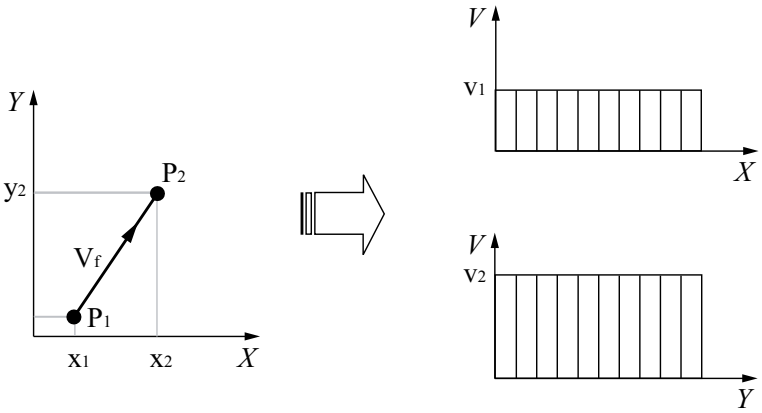


Fig. 3.1 The basic concept of an interpolator

3. The accumulation of interpolation error should be avoided in order that the final position should coincide as closely as possible to the position commanded.

The interpolator can be classified as either a hardware interpolator or a software interpolator by considering the implementation method. The hardware interpolator, which consists of various electric devices, was widely used until CNC was developed. However, today, interpolators implemented using software are used in modern CNC systems. The concept of the software interpolator originates from that of the hardware interpolator and the hardware interpolator is limited to control simple systems.

3.2 Hardware Interpolator

The hardware interpolator carries out the computation of interpolation and generates pulses by using an electric circuit. In the hardware interpolator, high-speed execution is possible, but it is difficult to adapt new algorithms or modify algorithms. In NC, the computation of interpolation and feedrate depends on hardware. However, the dependency on hardware has been gradually decreased because of the introduction of computer numerical control (CNC) systems.

The typical method for hardware interpolation uses a DDA(Digital Differential Analyzer) integrator. The method using the DDA integrator is transformed into a software version and can be applied to modern CNC. In this section, the DDA integrator will be introduced and the hardware interpolation method using a DDA integrator will be addressed.

3.2.1 Hardware Interpolation DDA

The hardware interpolator uses a DDA based on the principle of a numerical integration. A DDA is a digital circuit operated as a digital integrator that is similar to integration by an OP amplifier in an analog circuit.

Understanding of the concept of integration should be preceded by knowledge of the principle of interpolation. Given the velocity function $V(t)$, the displacement $S(t)$ can be approximated by summing the areas of the thin rectangles beneath the velocity curve as shown in Fig. 3.2.

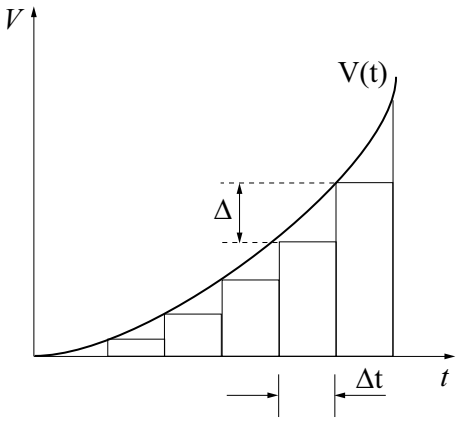


Fig. 3.2 Velocity curve and approximating rectangles

$$S(t) = \int_0^t V \cdot dt \cong \sum_{i=1}^{k} V_i \cdot \Delta t \tag{3.1}$$

where, Δt stands for an iteration time interval. If the displacement at time $t = k \cdot \Delta t$ is defined as S_k, Equation 3.1 can be rewritten as Eqs. 3.2 and 3.3.

$$S_k = \sum_{i=1}^{k-1} Vi \cdot \Delta t + V_k \cdot \Delta t \tag{3.2}$$

or

$$S_k = S_{k-1} + \Delta S_k \tag{3.3}$$

where ΔS_k is defined in Eq. 3.4.

$$\Delta S_k = V_k \cdot \Delta t \tag{3.4}$$

The following three processes are necessary for integration:

1. Calculate current velocity by velocity summing at the previous time unit and the velocity increment at the current time unit by using Eq. 3.5.

$$V_k = V_{k-1} + \Delta V_k \tag{3.5}$$

2. Calculate the distance increment at the current time unit using Eq. 3.4.
3. Calculate the total displacement by summing the displacement at the previous time unit and the distance increment at the current time unit using Eq. 3.3.

$$f = \frac{1}{\Delta t} \tag{3.6}$$

The above integration process is repeated for every constant time interval and the iteration frequency is given by Eq. 3.6.

The DDA integrator as mentioned above can be realized using hardware, and the hardware structure and representation symbol are shown in Fig. 3.3.

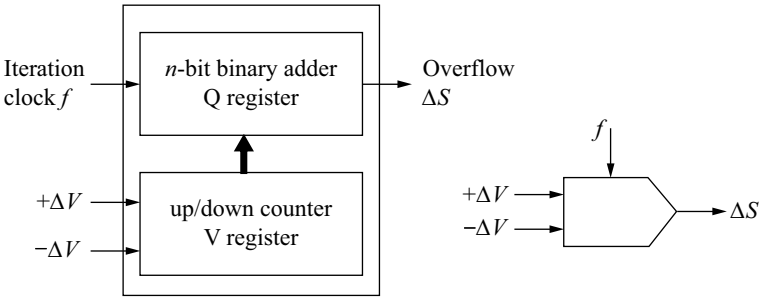


Fig. 3.3 DDA concept and representation symbol

The DDA integrator consists of two n-bit registers. The Q register is an n-bit binary adder and the V register is an n-bit up/down counter. The following is the integration process of a DDA integrator by using each component.

First, Eq. 3.5 is applied when the V register is updated by ΔV, being 0 or 1. This is added to the lowest bit of the V register whenever the DDA integrator is iterated. The value of the Q register and the value of the V register are summed up by binary addition. If the value of the Q register is greater than $(2^n - 1)$, which is the maximum value of an n-bit register, overflow occurs and this overflow becomes ΔS, which is the output of the DDA integrator.

Because the DDA integrator itself has no function for summing ΔS, an additional counter circuit is required in order to realize the second step of integration represented by Eq. 3.3. In mathematical form, the displacement ΔS is as in Eq. 3.7.

$$\Delta S_k = 2^{-n} \cdot V_k \tag{3.7}$$

Equation 3.7 can be written in the style of Eq.3.4 by utilizing Eq. 3.6, which gives Eq. 3.8.

$$\Delta S_k = 2^{-n} \cdot V_k \cdot (f \cdot \Delta t) = \frac{f}{2^n} \cdot V_k \cdot \Delta t \tag{3.8}$$

Accordingly, the average frequency for generating ΔS can be written as Eq. 3.9.

$$f_0 = \left(\frac{\Delta S}{\Delta t}\right)_k = \frac{f}{2^n} \cdot V_k \tag{3.9}$$

In the following, the bandwidth of the integration process is proportional to the frequency, f, and the velocity, V, while it is inversely proportional to 2^n. (where n is the length of a register and determines the resolution of the integration. Therefore, the larger the value of n, the higher the accuracy of the integration.)

3.2.2 DDA Interpolation

DDA hardware interpolation, which calculates the displacement and velocity of each axis based on part shape and command velocity, can be implemented using a DDA integrator. Figure 3.4 shows the circuit for a linear interpolator and a circular interpolator.

Linear interpolation means controlling the linear movement from a start position to an end position. In general, linear interpolation is implemented by simultaneously controlling two axes on a 2D plane or three axes in 3D space. However, in this book linear interpolation on a 2D plane will be used as an introduction in order to simplify the discussion.

When 2D linear interpolation is carried out, the most important thing is the synchronization of two axes with respect to the velocity and the displacement. For example, assume that the X-axis moves at maximum A BLU and Y-axis moves at maximum B BLU, as shown in Fig. 3.4a. In this case, the DDA hardware interpolator should generate 'A' pulses for the X-axis movement and 'B' pulses for the Y-axis movement. The frequency ratio of A to B should be maintained as constant.

A DDA hardware interpolator that is able to satisfy these conditions can be designed as shown in Fig. 3.4b. In this circuit, which consists of two DDA integrators, the X-axis and Y-axis are separated and can be executed simultaneously using identical clock signals. The total displacement of each axis is stored in the V register of the corresponding DDA integrator; The V register of the DDA integrator for the X-axis is set to the value 'A' and the V register of the DDA integrator for the Y-axis is set to the value 'B'. The overflow from each DDA integrator is generated as in Eq. 3.10 and Eq. 3.11. The overflow is fed to the input of the position control loop.

$$\Delta S_x = \left(\frac{f}{2^n}\right) \cdot A \cdot \Delta t \tag{3.10}$$

$$\Delta S_y = \left(\frac{f}{2^n}\right) \cdot B \cdot \Delta t \tag{3.11}$$

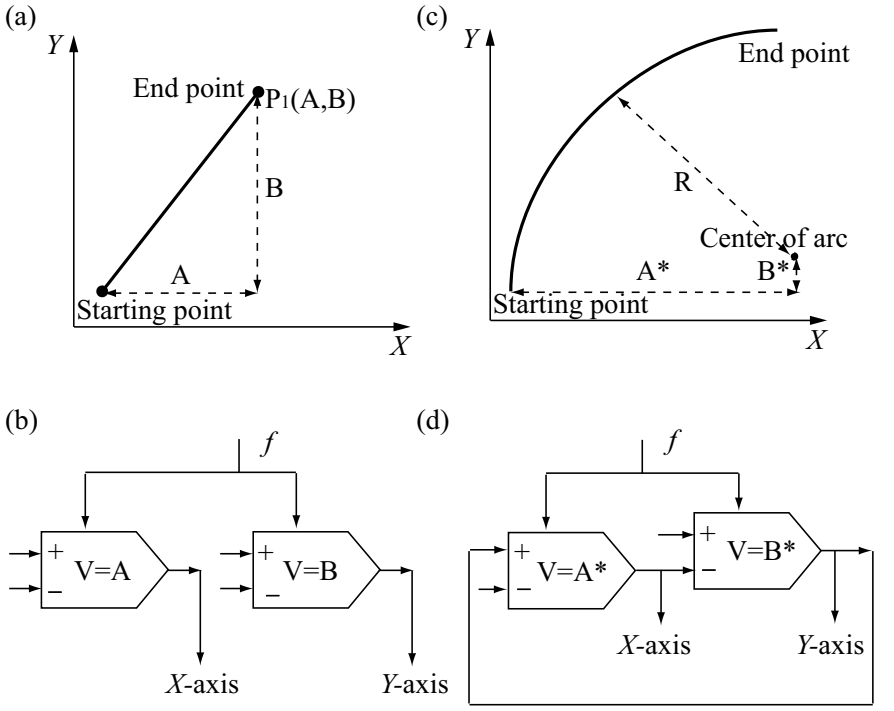


Fig. 3.4 DDA hardware interpolator

In order to interpolate the circle shown in Fig. 3.4c, a start position, an end position, a radius, and a vector from the start position to the center of the circle are needed. The circular interpolator should satisfy the following equations:

$$(X-R)^2+Y^2=R^2 \tag{3.12a}$$

$$X=R(1-\cos\omega t),\quad Y=R\sin\omega t \tag{3.12b}$$

where R is the radius of the circle and ω is the angular velocity.

By differentiating Eq. 3.12, the velocity for each axis can be calculated and are given by Eq. 3.13 or Eq. 3.14.

$$V_x=\frac{dX}{dt}=\omega R\cdot\sin\omega t \tag{3.13a}$$

$$V_y=\frac{dY}{dt}=\omega R\cdot\cos\omega t \tag{3.13b}$$

$$dX=\omega R\cdot\sin\omega t\cdot dt \tag{3.14a}$$

$$\omega R\cdot\sin\omega t\cdot dt=-d(R\cdot\cos\omega t) \tag{3.14b}$$

$$dY = \omega R \cdot \cos \omega t \cdot dt \tag{3.15a}$$

$$\omega R \cdot \cos \omega t \cdot dt = +d(R \cdot \sin \omega t) \tag{3.15b}$$

Based on Eq. 3.14 and Eq. 3.15, above, it is possible to design the DDA hardware interpolator. If $R \cdot \sin \omega t$ is assigned to the V register of the DDA integrator for the X-axis and $R \cdot \cos \omega t$ is assigned to the V register of the DDA integrator for the Y-axis, the output of each DDA integrator can be represented respectively by the right side of Eq. 3.14a and Eq. 3.15a. Considering Eq. 3.12b, the output of the DDA integrator for the X-axis can be used as the input of the summer of the DDA integrator of the X-axis. Based on the above concept, the circuit for a circular interpolation is designed by utilizing the two DDA integrators that have cross-connected inputs and outputs, as shown in Fig. 3.4d.

The behavior of the circuit for clockwise circular interpolation is shown in Fig. 3.4d and the description is as follows. Assume that the radius (R) of the circle is 15, the initial value of the vector (i, j) from the start position of the circle to the center is $(R, 0)$, and the length (n) of the register is 4. In this case, the value and the output of the registers can be summarized for the iteration period. Because the length of the registers is set to be n=4, the registers of the DDA for the X- and Y-axes can store 15, maximum. Furthermore, because the start position of a circle is (R, 0), the initial value of the V registers of the DDA for the X- and Y-axes are 15 and 0, respectively. In the first iteration, the initial value of the V register of the DDA for the X-axis is added to the Q register. Because overflow does not occur in the Q register, the DDA for the Y-axis is not changed.

In the second iteration, the value of the V register of the DDA for the X-axis is added to the Q register. Now, overflow occurs in the Q register and the value of the Q register becomes 14 $(30 - 16 = 14)$. Because of the overflow from the X-axis DDA, the V register of the Y-axis DDA comes to have value 1. By adding the value of the V register and the value of the Q register, the Q register comes to have the value 1.

As this process is repeated, due to the overflow from the DDA for the X-axis, the V register for the Y-axis is incremented. On the other hand, due to the overflows from the DDA for the Y-axis, the V-register of the DDA for the X-axis is decremented, resulting in circular interpolation. In this process, the overflow from the DDA for the X-axis is the movement signal along the X-axis at 1 BLU and the overflow from the DDA for the Y-axis is the movement signal along the Y-axis at 1 BLU. In Table 3.1, the total number of overflows occurring for the DDA for the X-axis is 15 and also for the DDA for the Y-axis, as indicated by the values for ΔS.

3.3 Software Interpolator

With the reduction of the price and size of PCs, a software interpolation method has appeared in which interpolation is carried out using a computer program instead of a logic arithmetic hardware device. Various algorithms have been introduced for soft-

Table 3.1 DDA integrator

Iteration step	DDA Integrator for X-axis			DDA Integrator for Y-axis		
	V	Q	ΔS	V	Q	ΔS
0	15	0		0	0	
1	15	15		0	0	
2	15	14	1	1	1	
3	15	13	1	2	3	
4	15	12	1	3	6	
5	15	11	1	4	10	
6	15	10	1	5	15	
7	15	9	1	6	5	1
8	14	7	1	7	12	
9	14	5	1	8	4	1
10	13	2	1	9	13	
11	13	15		9	6	1
12	12	11	1	10	0	1
13	11	6	1	11	11	
14	11	1	1	12	7	1
15	10	11		12	3	1
16	9	4	1	13	0	1
17	8	12		13	13	
18	8	4	1	14	11	1
19	7	11		14	9	1
20	6	1	1	15	8	1
21	5	6		15	7	1
22	4	10		15	6	1
23	3	13		15	5	1
24	2	15		15	4	1
25	1	0		15	3	1

ware interpolation and Table 3.2 summarizes the typical algorithms for the reference pulse interpolator and the reference word interpolator (Sampled-Data interpolator).

In the reference pulse method, a computer generates reference pulses as an external interrupt signal and the generated pulses are directly forwarded to the machine drive. Also in this method, 1 pulse denotes 1BLU of axis.

Table 3.2 The type of a software interpolator

Reference Pulse Method	Sampled Data Method
Software DDA Interpolator	Euler Interpolator
Stairs Approximation Interpolator	Improved Euler Interpolator
Direct Search Interpolator	Taylor Interpolator
	Tustin Interpolator
	Improved Tustin Interpolator

Figure 3.5 shows the structure and information flow for the reference pulse method. A CNC system based on the reference pulse method includes an Up/Down Counter. This counter compares the reference pulse from an interpolator with the

feedback pulse from an encoder, calculates the position error from the comparison result, and finally, the calculated position error is input to a drive. In this structure, the total of the generated pulses determines the position of an axis and the frequency of pulses determines the speed of an axis.

The interrupt frequency determines the speed of the axis in this method, so simple programs and a fast CPU are essential for high-speed operation. The DDA (Digital Differential Analyzer) algorithm, Stairs Approximation algorithm, and Direct Search algorithm are suitable for this interpolation method.

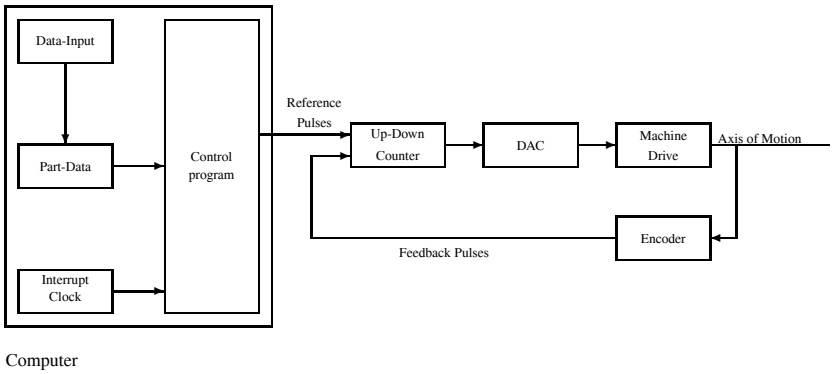


Fig. 3.5 Software interpolator based on a reference pulse method

In the Sampled-Data interpolation method, Fig. 3.6, the interpolation is executed in two stages unlike the reference pulse method. In the first stage, an input contour is segmented into straight line segments within an allowable tolerance and, in the second stage, the approximating line segments are interpolated and the interpolation result sent to the related axis. In general, the first stage is called rough interpolation and the second stage fine interpolation. When the performance of microprocessors was low, a software interpolator carried out the rough interpolation and a hardware interpolator was used for the fine interpolation. The Euler method, Taylor method, and Tustin method are typical algorithms for rough interpolation.

Table 3.3 summarizes the comparison between the above-mentioned software interpolators. In the case of the reference pulse interpolation method, the feedrate of an axis is subject to the CPU speed but, compared to the Sampled-Data interpolation method, control with higher accuracy is possible. The Sampled-Data interpolation method is suitable for a high-speed interpolator so that there is no limitation regarding the speed of an axis. However, because line approximation is used when circles are interpolated, calculation errors, rounding errors, and accumulation errors at the connection positions between lines occur and these errors result in large contour error. In addition, this method complicates the control loop and requires a larger internal memory compared with the reference pulse method to store the approximating line segments.

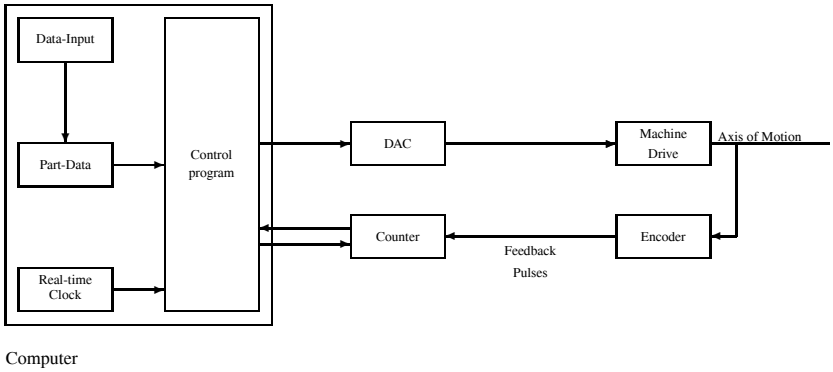


Fig. 3.6 Software interpolator based on a sampled-data method

Table 3.3 Reference pulse and Sampled-Data methods comparison

	Reference Pulse Method	Sampled-Data Method
Description	The distance to go is computed with external reference pulses and the calculated pules are fed to each axis.	The coordinate points to go per unit sampling time are computed and the calculated data is transmitted to each axis.
Strengths	Adequate for high-accuracy machining.	Adequate for high-speed machining.
Weakness	Inadequate for high-speed machining.	Inadequate for high-accuracy machining.
Remarks	Because the feedrate of each axis depends on the frequency of external interrupt signal, a high performance CPU is required.	

3.3.1 Software Interpolation Methods

In this section, the concepts and the flowcharts for the DDA algorithm, the Stairs Approximation algorithm, and the Direct Search algorithm will be addressed. These are typical algorithms for the reference-pulse method.

3.3.1.1 Software DDA Interpolator

Software DDA interpolation algorithms originate from hardware DDAs and their execution procedure is the same as the behavior of hardware DDAs. Figure 3.7 shows the flow chart for a software DDA interpolation algorithm and Fig. 3.7a and Fig. 3.7b respectively show linear interpolation and circular interpolation. In Fig. 3.7a, the variable L is a linear displacement and the variables A and B denote the displace-

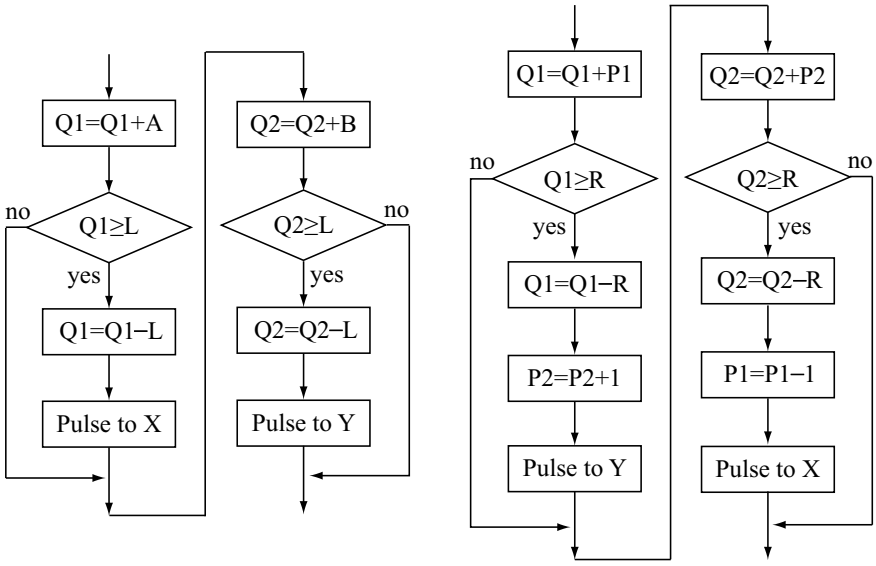


Fig. 3.7 Software DDA interpolator

ment of the *X*- and *Y*-axes, and the initial value of variables Q1 and Q2 is zero. In Fig. 3.7b, the initial values of variables are the same as those for linear interpolation, the variable *R* is the radius of the circle, and the variables P1 and P2 give the center position when the start point of the circle is the origin of the coordinate system.

The following is an example of a software DDA interpolation algorithm and an example part program is below. The length unit of the example part program is BLU and a speed unit is BLU per second.

G01 X0.Y10.F10
G02 G90 X10. Y0. I0. J-10. F10

The example part program denotes the circular movement in a clockwise direction in the first quadrant and Fig. 3.8 shows the result of the interpolation.

3.3.1.2 Stairs Approximation Interpolator

The Stairs Approximation algorithm, termed an incremental interpolator, determines the direction of the step every BLU interval and sends the pulse to the related axis. In this section, the Stairs Approximation interpolator for a circle will be addressed and the algorithm for a line can be easily determined from the algorithm for a circle. Figure 3.9 shows how the Stairs Approximation interpolator for a circle behaves in the case that the commanded circular movement is in the clockwise direction in the first quadrant with respect to the center of the circle.

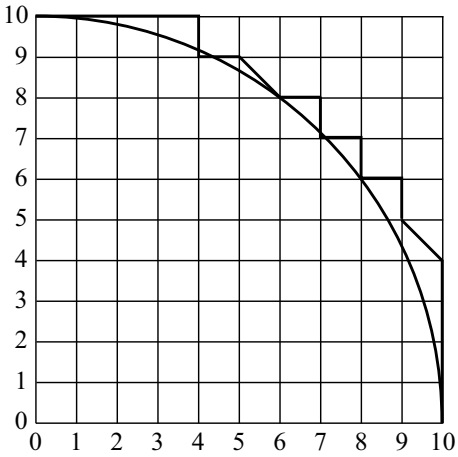


Fig. 3.8 Result of interpolation by using software DDA algorithm

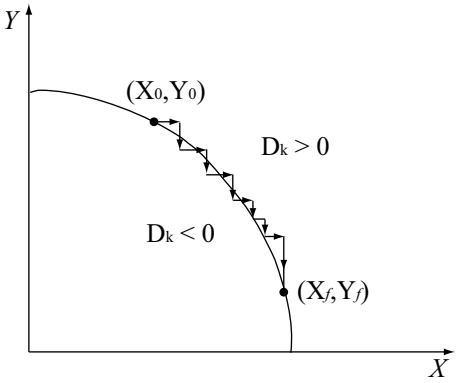


Fig. 3.9 The behavior of the Stairs Approximation interpolation algorithm

Assume that the tool reaches the position (X_k, Y_k) after the ith iteration. In this algorithm, the variable D_k is calculated by Eq. 3.16.

$$D_k = X_k^2 + Y_k^2 - R^2 \tag{3.16}$$

The direction of a step is determined based on D_k, the commanded circular direction, and the quadrant where movement is done. For example, if the circular movement is carried out in a clockwise direction in the first quadrant, the algorithm executed is as below.

1. $D_k < 0$: This case means that the position (X_k, Y_k) is located on the inside of a circle and, in this case, the step moves in the positive direction of X-axis.
2. $D_k > 0$: This case means that the position (X_k, Y_k) is located on the outside of a circle and, in this case, the step is moved in the negative direction of the Y-axis.

3. $D_k = 0$: One of the above rules can be arbitrarily selected and applied.

After one step is completed by applying the above rules, the position (X_{k+1}, Y_{k+1}) is updated and the procedure repeated until the tool reaches the position commanded, (X_f, Y_f).

Table 3.4, shows the displacement values for the eight cases of the Stairs Approximation algorithm, which can calculate a total of eight different cases according to the quadrant.

Table 3.4 Eight stairs for arc path

No.	Quadrant	Direction	$D < 0$	$D > 0$
1	1	CW	+X	–Y
2	1	CCW	+Y	–X
3	2	CW	+Y	+X
4	2	CCW	–X	–Y
5	3	CW	–X	+Y
6	3	CCW	–Y	+X
7	4	CW	–Y	–X
8	4	CCW	+X	+Y

This algorithm requires a small amount of computation and less memory space. However, numerous iterations are required and N, the number of iteration steps, can be calculated using Eq. 3.17.

$$N = |X_0 - X_f| + |Y_0 - Y_f|, \quad (X_0, Y_0) : \text{Start position} \tag{3.17}$$

For example, in the case of interpolating a quarter circle with radius R, the total number of iterations, N, is $2R$. In order to maintain the command velocity V_l (BLU/second), the interpolation should be repeated with frequency f_0 according to Eq. 3.18.

$$\frac{f_0}{V_l} = \frac{2R}{\pi R/2} = \frac{4}{\pi} \tag{3.18}$$

However, an improved algorithm is proposed because the above-mentioned Stairs Approximation algorithm has some problems. In the improved Stairs Approximation algorithm, Eq. 3.16 is changed to Eq. 3.19 by adding one more index.

$$D_{i,j} = X_i^2 + Y_j^2 - R^2 \tag{3.19}$$

In Eq. 3.19, i and j respectively denote the number of steps along the X-axis and the Y-axis. When one step is added along the X-axis, Eq. 3.19 is changed to Eq. 3.20.

$$\begin{aligned} D_{i+1,j} &= (X_i + 1)^2 + Y_j^2 - R^2 \\ &= D_{i,j} + 2X_i + 1 = D_{i,j} + \Delta X_i \\ \Delta X_{i+1} &= \Delta X_i + 2 \end{aligned} \tag{3.20}$$

Figure 3.10 shows the behavior of the Stairs Approximation algorithm.

G01 X0.Y10.F10
G02 G90 X10. Y0.(BLU) I0. J-10.(BLU) F10(BLU/sec)

The following is an example of the Stairs Approximation interpolation algorithm. The part program, below, is the same as that for the software DDA interpolation algorithm and denotes circular movement in a clockwise direction in the first quadrant. Figure 3.11 and Table 3.5 show the result of the algorithm for the example part program.

Table 3.5 Results of Stairs Approximation interpolation algorithm

step	D	Δx	Δy	ΔX_f	ΔY_f	X	Y
0	0	1	–19	10	10	0	10
1	1	3	–19	9	10	1	10
2	–18	3	–17	9	9	1	9
3	–15	5	–17	8	9	2	9
4	–10	7	–17	7	9	3	9
5	–3	9	–17	6	9	4	9
6	6	1	–17	5	9	5	9
7	–11	11	–15	5	8	5	8
8	0	13	–15	4	8	6	8
9	13	15	–15	3	8	7	8
10	–2	15	–13	3	7	7	7
11	13	17	–13	2	7	8	7
12	0	17	–11	2	6	8	6
13	17	19	–11	1	6	9	6
14	6	19	–9	1	5	9	5
15	-3	19	–7	1	4	9	4
16	16	21	–7	0	4	10	4
17	9	21	–5	0	3	10	3
18	4	21	–3	0	2	10	2
19	1	21	–1	0	1	10	1
20	0	21	1	0	0	10	0

The equation being approximated in this case is:

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, \quad (X_0, Y_0) = (0,10), (x_f, Y_f) = (10,0)$$

3.3.1.3 Direct Search

One of the weaknesses of the Stairs Approximation algorithm is that a lot of iterations are required because simultaneous movement of axes is not considered in the algorithm. Furthermore, possible error conditions are not considered when the interpolated position is decided. As an alternative, the Direct Search algorithm, which is introduced in this section, carries out optimal interpolation because the algorithm

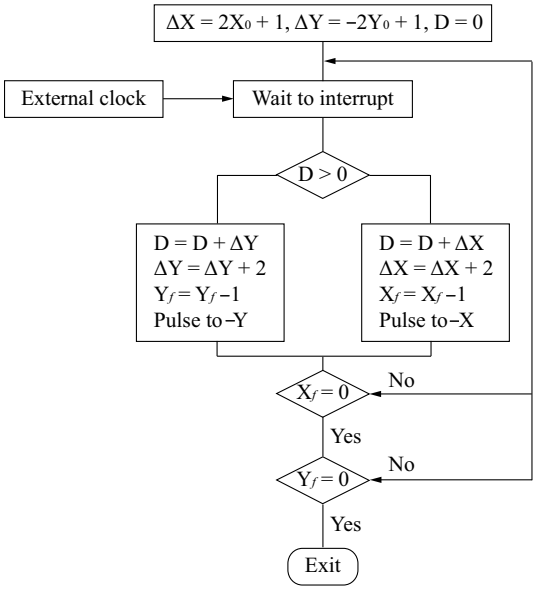


Fig. 3.10 Flowchart for the Stairs Approximation interpolation algorithm

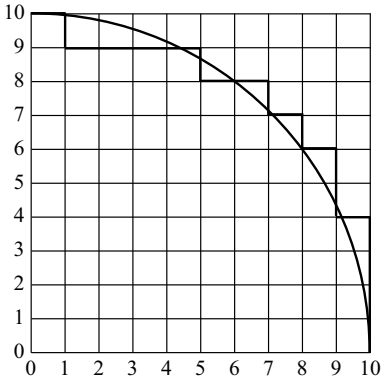


Fig. 3.11 Stairs Approximation

searches through all possible directions and finds a direction with the minimum path error. Basically, the Direct Search algorithm is very similar to the Stairs Approximation algorithm. However, the Direct Search algorithm considers the simultaneous movement of axes based on path error, in contrast to the Stairs Approximation algorithm.

In the Direct Search algorithm, it is known that the variable $D_{i,j}$ in Eq. 3.19 is proportional to the radial error $E_{i,j}$ as in Eq. 3.21. Evaluation of the radial error can be replaced by evaluation of $D_{i,j}$.

$$D_{i,j} \cong (2R)E_{i,j} \tag{3.21}$$

The Direct Search algorithm searches all possible points that can move from the present position, and finds a point having minimum error by estimating $D_{i,j}$ at possible points. The possible points are decided based on the commanded movement direction and a quadrant, as shown in Table 3.4. For example, when clockwise circular movement in the first quadrant is commanded, three cases should be considered: 1) increasing 1 BLU for the X-direction, 2) decreasing 1 BLU for the Y-direction, and 3) increasing 1 BLU for the X-direction and simultaneously decreasing 1 BLU for the Y-direction. The third case is different from the Stairs Approximation algorithm.

The $D_{i,j}$ for each case is evaluated and the case with smallest absolute value is selected for movement. Figure 3.12 shows the flow chart of the Direct Search algorithm for clockwise circular movement in the first quadrant.

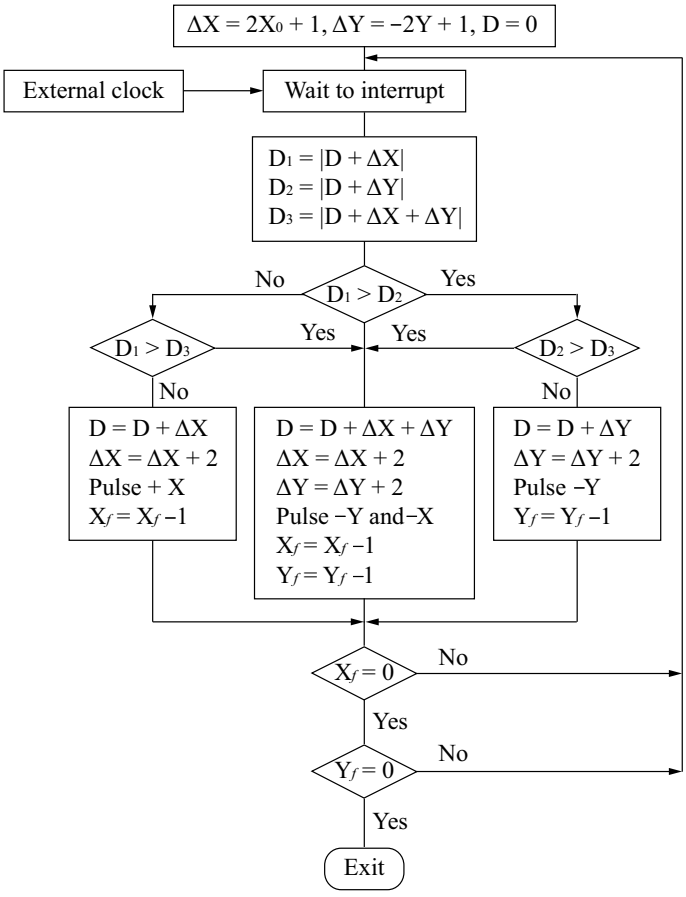


Fig. 3.12 Direct search flowchart

The maximum error of the Direct Search is 1/2 BLU and the accuracy of this algorithm is better than that of the Stairs Approximation algorithm. However, the Direct Search algorithm requires as many as $N = \sqrt{2}$ iterations in order to interpolate a circular arc with radius R than by using the Direct Search algorithm. The number of iterations is 30% smaller than that of the Stairs Approximation algorithm and about 20% smaller than that of the DDA software algorithm. The maximum allowable radius is the same as that for the Stairs Approximation algorithm. In order to hold the command velocity V_l (BLU/second), the algorithm should be repeated at frequency F given in Eq. 3.22.

$$F = \frac{2\sqrt{2}}{\pi} V \tag{3.22}$$

The following is an example of the Direct Search algorithm. The code sequence below is identical to that used to illustrate the DDA software interpolator and represents clockwise circular movement in the first quadrant. Figure 3.13 and Table 3.6 show the results of the Direct Search algorithm.

G01 X0.Y10.F10
G02 G90 X10. Y0.(BLU) I0. J-10.(BLU) F10(BLU/second)

The equation being interpolated is:

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2,\ (X_0, Y_0) = (0,10),\ (X_f, Y_f) = (10,0)$$

Table 3.6 Direct Search

step	D	D1	D2	D3	Δx	Δy	ΔXf	ΔYf	X	Y
0	0				1	–19	10	10	0	10
1	1	1	–19	–18	3	–19	9	10	1	10
2	4	4	–18	–15	5	–19	8	10	2	10
3	9	9	–15	–10	7	–19	7	10	3	10
4	3	16	–10	–3	9	–17	6	9	4	9
5	6	6	–20	–11	11	–17	5	9	5	9
6	0	17	–11	0	13	–15	4	8	6	8
7	–2	13	–15	–2	15	–13	3	7	7	7
8	0	13	–15	0	17	–11	2	6	8	6
9	6	17	–11	6	19	–9	1	5	9	5
10	–3	25	–3	16	19	–7	1	4	9	4
11	9	16	–10	9	21	–5	0	3	10	3
12	4	30	4	25	21	–3	0	2	10	2
13	1	25	1	22	21	–1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0

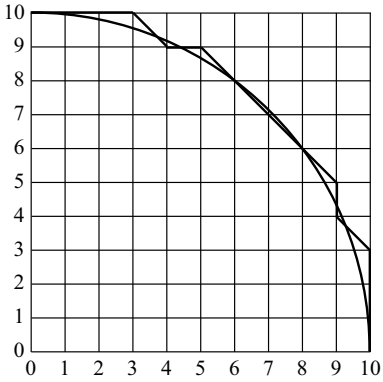


Fig. 3.13 Direct search results

3.3.1.4 Reference Pulse Interpolator Algorithms

Maximum allowable radius, consistency of feedrate, maximum allowable feedrate, and maximum error can be considered as performance indices for evaluating various reference-pulse interpolator algorithms. Table 3.7 summarizes the performance and characteristics of the above-mentioned algorithms considering these criteria. Many tests have been carried out for the various reference-pulse interpolation algorithms and Table 3.7 shows the result of these tests.

Table 3.7 Reference-pulse interpolation method comparison

Method	$R_{max}+1$	E_{max}	N	V	V_{max}	V_{min}
DDA	2^{n-2}	1	$(\pi/2)R$	F	F	F
Stairs	2^{n-2}	1	$2R$	$(\pi/4)F$	F	$(1/\sqrt{2})F$
DSM	2^{n-2}	1/2	$\sqrt{2}R$	$(\pi/2\sqrt{2})F$	$\sqrt{2}F$	F

Consequently, all algorithms hold the consistency of feedrate during linear interpolation but only the DDA software interpolator algorithm can hold consistency of feedrate for V_{max} and V_{min}. Note also that the Stairs Approximation algorithm allows the greatest maximum feedrate.

3.3.2 Sampled-Data Interpolation

As previously mentioned, the Sampled-Data interpolation method is typical for modern CNC. In this method, the algorithm is repeated every constant time interval.

3.3.2.1 Reference Word Interpolator for Lines

In the reference word algorithm, linear interpolation is very simple compared with circular interpolation. Figure 3.14 shows the concept and flowchart for linear interpolation.

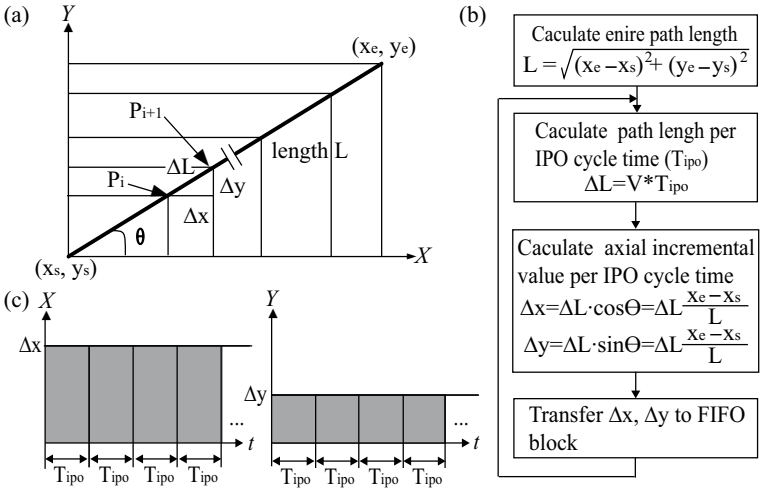


Fig. 3.14 Reference word algorithm – linear interpolation

The fundamental idea of this algorithm is segmentation of the path by the interpolation time, T_{ipo}. At each sampling time the interpolated point can be defined as in Eq. 3.23 wherein the displacement of each axis is given by Eq. 3.24.

$$x_{i+1} = x_i + \Delta x \qquad y_{i+1} = y_i + \Delta y \tag{3.23}$$

$$\Delta x = \Delta L \cdot \cos\theta = \Delta L \frac{x_e - x_s}{L} \tag{3.24}$$
$$\Delta y = \Delta L \cdot \sin\theta = \Delta L \frac{y_e - y_s}{L}$$

$$\Delta L = V \cdot T_{ipo} \tag{3.25}$$

$$L = \sqrt{(x_e - x_s)^2 + (y_e - y_s)^2} \tag{3.26}$$

$$V = V_0 \times feed\ override \tag{3.27}$$

As shown in Eq. 3.25, the line segment ΔL depends on velocity V. Velocity V is defined by Eq. 3.27 when the command velocity V_0 defined in a part program is compensated for by feed override.

Figure 3.14 represents the flow chart for the above-mentioned interpolation procedure. Figure 3.14a shows the total linear path to move, interpolation points to move to at each interpolation time, and axial increment at every interpolation time. Figure 3.14b shows the flow chart for the interpolation procedure and the equations for generating interpolated points and axial increments. Figure 3.14c shows axial increments for every interpolation time step as a 2D graph.

$$N = int\left(\frac{L}{\Delta L}\right) \tag{3.28}$$

Here, the total number of iterations for the interpolation is defined by Eq. 3.28. Axial increment is transferred to the ring buffer (FIFO buffer) every interpolation time step and axial increment is used as input for acceleration/deceleration control. In general, because constant feedrate is allocated to line blocks in the part program, Δx and Δy hold constant velocity. However, if feed override is applied, the axial increment can be changed every interpolation time.

At this moment, one question occurs. When the length of the line path, L, is not exactly N times the displacement per interpolation time ΔL, where N is an integer, how should the residual length be interpolated? The typical method for processing the residual length is to allocate the remainder evenly to every interpolation time. For example, if the residual length is 10, then add 1 to every interpolation value from the first to the tenth interpolation time. However, this method requires a heavy computational power although the velocity is maintained constant. Therefore, the best and real practical way of handling the residual length is simply to add all the residual length on the interpolation value at the last interpolation time. This does not affect the machining accuracy at all.

3.3.2.2 Reference Word Interpolator for Circles

During circular interpolation, tangential velocity, V should be held on the circular path. The individual velocity of the axes is defined by Eq. 3.29.

$$\begin{aligned} &V_x(t) = V\sin\theta(t) \qquad V_y(t) = V\cos\theta(t) \\ &\text{where} \quad \theta(t) = (Vt/R) \end{aligned} \tag{3.29}$$

The velocity of the axes, (V_x, V_y), is computed by a circular interpolator, and fed to the closed loop of position control as a reference input. In the reference word interpolator for a circle, a circular path is approximated by small line segments. The larger the number of line segments, the better the accuracy of interpolation. However, more computation power is required as the number of iterations increases. Therefore,

it is necessary to optimize the number of line segments so that the results of the path error comes to be within 1 BLU.

Figure 3.15 shows the relationship between two successive interpolated points for the reference word interpolator for a circle. In the reference word interpolator for a circle, the iteration step can be expressed using the angle α. The size of this angle α is a key factor for interpolation. Therefore, various algorithms have been proposed to determine angle α.

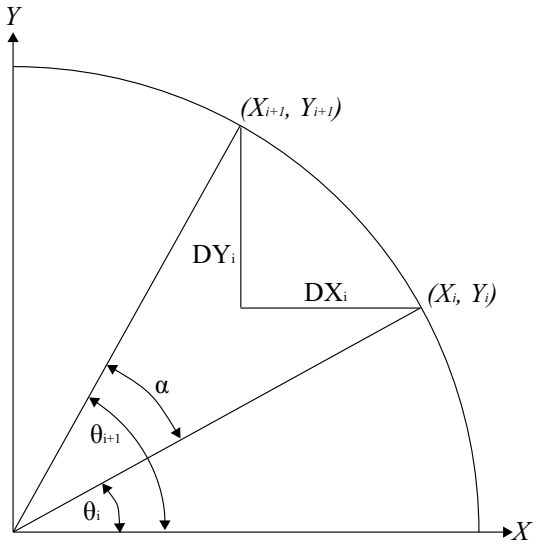


Fig. 3.15 Reference word – circular interpolation

Equation 3.30 is the mathematical relationship between two successive interpolated points. The conditions for reference word interpolation of a circle are given in Eq. 3.31.

$$\begin{aligned} \cos\theta(i+1) &= A\cos\theta(i) - B\sin\theta(i) \\ \sin\theta(i+1) &= A\sin\theta(i) + B\cos\theta(i) \end{aligned} \tag{3.30}$$

where,

$$\begin{aligned} A = \cos\alpha \qquad B &= \sin\alpha \\ \theta(i+1) &= \theta(i) + \alpha \end{aligned} \tag{3.31}$$

The final position of the segment $(X(i+1), Y(i+1))$ can be approximated as in Eq. 3.32.

$$X(i+1) = R(i)\cos\theta(i+1) \qquad Y(i+1) = R(i)\sin\theta(i+1) \tag{3.32}$$

From Eq. 3.30 and Eq. 3.32, Eq. 3.33 is derived. Equation 3.33 shows that it is possible to calculate successive interpolated points by using the current interpolated point,

$$X(i+1) = AX(i) - BY(i) \qquad Y(i+1) = AY(i) + BX(i) \tag{3.33}$$

Reference word interpolation algorithms that are introduced in this book depend on the above equations and the differences between algorithms are the methods used to determine angle α and how to approximate A and B from Eq. 3.33.

If angle α is determined regardless of the type of algorithm, the following interpolation routine is executed every iteration. The start position and the velocity are provided by a part program.

Using Eq. 3.33 and an initial interpolated point $(X(i), Y(i))$ the next interpolated point $(X(i+1), Y(i+1))$ can be computed. The length of the line segment is computed using Eq. 3.34 and Eq. 3.35 gives the velocities.

$$\begin{aligned} DX(i) &= X(i+1) - X(i) = (A-1)X(i) - BY(i) \\ DY(i) &= Y(i+1) - Y(i) = (A-1)Y(i) + BX(i) \end{aligned} \tag{3.34}$$

$$\begin{aligned} V_x(i) &= \frac{VDX(i)}{DS(i)} \\ V_y(i) &= \frac{VDY(i)}{DS(i)} \\ \text{where} \quad DS(i) &= \sqrt{DX^2(i) + DY^2(i)} \end{aligned} \tag{3.35}$$

$DX(i)$ and $DY(i)$ are, respectively, the increments along the X- and Y- axes. $V_x(i)$ and $V_y(i)$ are the velocities, called reference words, for the X- and Y-axes respectively. These values are transmitted to the acceleration/deceleration control routine via a ring buffer. Using these, the interpolation routine updates the current interpolated point $(X(i), Y(i))$.

3.3.2.3 Radial Error and Chord Height Error

Two kinds of an error occur when a circle is approximated by line segments, as shown in Fig. 3.16; radial error (ER) and chord height error (EH).

If the radius of a circle is R, the radial error, ER, is computed using Eq. 3.36.

$$ER(i) = R(i) - R = \sqrt{X^2(i) + Y^2(i)} - R \tag{3.36}$$

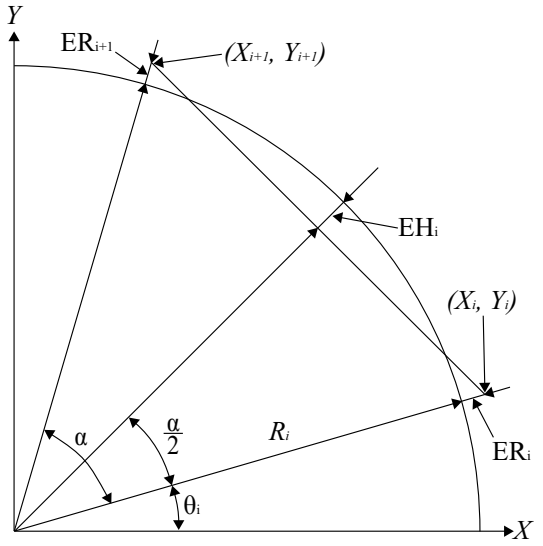


Fig. 3.16 Radial and chord height errors

ER is an error from a truncation effect. *ER* can be approximated by coefficients *A* and *B* and this error is accumulated with iteration. At the *i*th iteration, *ER* can be computed approximately using Eq. 3.37.

$$ER(i) = i(C-1)R \tag{3.37}$$
$$\text{where} \quad C = \sqrt{A^2 + B^2}$$

Unlike radial error *ER*, chord height error, *EH*, is not accumulated. *EH* is computed using Eq. 3.38. *EH* can be written as Eq. 3.40, which includes coefficient *A*, by using Eq. 3.39.

$$EH(i) = R - R(i)\cos\frac{\alpha}{2} \tag{3.38}$$

$$\cos\frac{\alpha}{2} = \sqrt{\frac{1+\cos\alpha}{2}} = \sqrt{\frac{1+A}{2}} \tag{3.39}$$

$$EH(i) = R - R(i)\sqrt{\frac{1+A}{2}} \tag{3.40}$$

Based on the above, α should be determined such that the value of *ER* or *EH* does not exceed a value equivalent to 1 BLU.

In the following sections, various reference word interpolation algorithms based on the above-mentioned basic idea will be addressed.

3.3.2.4 Euler Algorithm

In the Euler algorithm, $\cos\alpha$ and $\sin\alpha$ are approximated by first-order Taylor series expansion. A and B in Eq. 3.31 are written as in Eq. 3.41.

$$A = 1, B = \alpha \tag{3.41}$$

Since the series expansion is truncated, a radial error ER influences the accuracy of the algorithm. The maximum error of this algorithm is calculated using Eq. 3.42 and, if α is small, the maximum error can be approximated using Eq. 3.43.

$$ER_{max} = (\pi/2\alpha)(\sqrt{1+\alpha^2} - 1)R \tag{3.42}$$
$$ER_{max} \cong (\pi/4)\alpha R \quad \text{(for small } \alpha\text{)} \tag{3.43}$$

If a quarter circle is interpolated using this algorithm, angle α is computed from Eq. 3.44 and for interpolating a quarter circle, Eq. 3.45 gives the number of iteration steps.

$$\alpha = 4/(\pi R) \tag{3.44}$$
$$N = \pi^2 R/8 \tag{3.45}$$

3.3.2.5 Improved Euler Algorithm

The Improved Euler algorithm is similar to the Euler algorithm but differs in that $X(i+1)$ is used for calculating $Y(i+1)$ instead of $X(i)$. In the Improved Euler algorithm, Eq. 3.46 is used instead of Eq. 3.33. The average of coefficient A is approximated using Eq. 3.47 and is close to $\cos\alpha$.

$$\begin{aligned} X(i+1) &= AX(i) - BY(i) = X(i) - \alpha Y(i) \\ Y(i+1) &= AY(i) + BX(i+1) = (1-\alpha^2)Y(i) + \alpha X(i) \end{aligned} \tag{3.46}$$

$$A = \frac{1}{2}(1 + (1-\alpha^2)) = 1 - \frac{1}{2}\alpha^2 \tag{3.47}$$

The radial error ER of the Improved Euler algorithm is maximized at $\theta = \pi/4$ or the iteration where Eq. 3.48 is satisfied.

$$X(i) = (1+\alpha)Y(i) \tag{3.48}$$

When using the Improved Euler algorithm, angle α is computed as in Eq. 3.49 and for interpolating a quarter circle, Eq. 3.50 gives the number of iteration steps.

Equation 3.50 shows that the Improved Euler algorithm is more efficient than the Euler algorithm.

$$\alpha = 4/R \tag{3.49}$$

$$N = \pi R/8 \tag{3.50}$$

3.3.2.6 Taylor Algorithm

In the Taylor algorithm, the coefficients A and B are approximated as a truncated series as in Eq. 3.51. The coefficient A in Eq. 3.51 is equal to the average of coefficient A in the Improved Euler algorithm.

$$A = 1 - \frac{1}{2}\alpha^2,\ B = \alpha \tag{3.51}$$

By applying Eq. 3.51, a maximum radial error is derived as shown in Eq. 3.52 and the chord height error EH is derived as in Eq. 3.53.

$$ER_{max} = \frac{\pi}{2\alpha}\left(\sqrt{1+\frac{1}{4}\alpha^4} - 1\right)R \cong \pi R\alpha^3/16 \tag{3.52}$$

$$EH(i) = R - R(i)\sqrt{1-\frac{1}{4}\alpha^2} = R - R(i) + (\alpha^2/8)R(i) \tag{3.53}$$

In this algorithm, the chord height error EH is maximized at $R(i) = R$, the first iteration. The maximum chord height error is calculated using Eq. 3.54.

$$EH_{max} = R\alpha^2/8 \tag{3.54}$$

Equation 3.52 and Eq. 3.54 lead to Eq. 3.55.

$$ER_{max} = \frac{\pi\alpha}{2}EH_{max} \tag{3.55}$$

From Eq. 3.55 we know that if angle α is smaller than $2/\pi$, the chord height error (EH) is larger than the radial error ER. In general, because $\alpha < 2/\pi$ is satisfied if R is larger than 20 BLUs, this condition is practical.

Also, if the chord height error (EH) is set to 1 BLU, then α can be determined by Eq. 3.54.

In the case of using the Taylor algorithm, angle α is computed using Eq. 3.56 and for interpolating a quarter circle, Eq. 3.57 gives the number of iterations.

$$\alpha = \sqrt{8/R} \tag{3.56}$$

$$N = \pi/8\alpha \tag{3.57}$$

3.3.2.7 Tustin Algorithm

The Tustin algorithm is based on an approximation relationship between differential operators and a discrete variable z as shown in Eq. 3.58.

$$s = \frac{2}{T}\left(\frac{z-1}{z+1}\right) \tag{3.58}$$

Based on Eq. 3.58, A and B in Eq. 3.31 can be written as in Eq. 3.59.

$$A = \frac{1-(\alpha/2)^2}{1+(\alpha/2)^2} \qquad B = \frac{\alpha}{1+(\alpha/2)^2} \tag{3.59}$$

If coefficients A and B, as shown in Eq. 3.59, are applied to Eq. 3.34 and Eq. 3.35, then Eq. 3.60 and Eq. 3.61 are derived.

$$\begin{aligned} DX(i) &= -\frac{1}{1+(\alpha/2)^2}\left[\frac{\alpha^2}{2}X(i)+\alpha Y(i)\right] \\ DY(i) &= \frac{1}{1+(\alpha/2)^2}\left[\frac{-\alpha^2}{2}Y(i)+\alpha X(i)\right] \end{aligned} \tag{3.60}$$

$$\begin{aligned} V_x(i) &= -\frac{V}{R[1+(\alpha/2)^2]}\left[\frac{\alpha}{2}X(i)+Y(i)\right] \\ V_y(i) &= \frac{V}{R[1+(\alpha/2)^2]}\left[-\frac{\alpha}{2}Y(i)+X(i)\right] \end{aligned} \tag{3.61}$$

ER, EH, and angle α can be written down using the following equations.

$$ER(i) = 0 \tag{3.62}$$

Because the radial error ER is always zero, angle α is determined based on the chord height error EH. If $R(i) = R$, EH is determined using Eq. 3.63. If angle α is very small, EH is determined using Eq. 3.64. Angle α is calculated using Eq. 3.65.

$$EH(i) = R - \frac{R}{\sqrt{1+(\alpha/2)^2}} \tag{3.63}$$

$$EH(i) = \frac{\alpha^2}{\alpha^2+8}R \tag{3.64}$$

$$\alpha = \sqrt{\frac{8}{R-1}} \cong \sqrt{\frac{8}{R}} \tag{3.65}$$

When using angle α from Eq. 3.65 for interpolating a quarter circle, Eq. 3.66 gives the number of iteration steps.

$$N = \frac{\pi}{4}\sqrt{R/2} \tag{3.66}$$

3.3.2.8 Improved Tustin Algorithm

In the Tustin algorithm, the radial error ER is set to 0. However, from a practical point of view, ER can be set to 1 BLU. Doing this, it is possible to increase angle α and the efficiency of the algorithm increases. Based on this idea, the Improved Tustin algorithm was proposed. Figure 3.17 shows the radial error ER, the chord height error EH is set to 1 BLU from which a new algorithm including Eq. 3.67 through 3.70 can be derived.

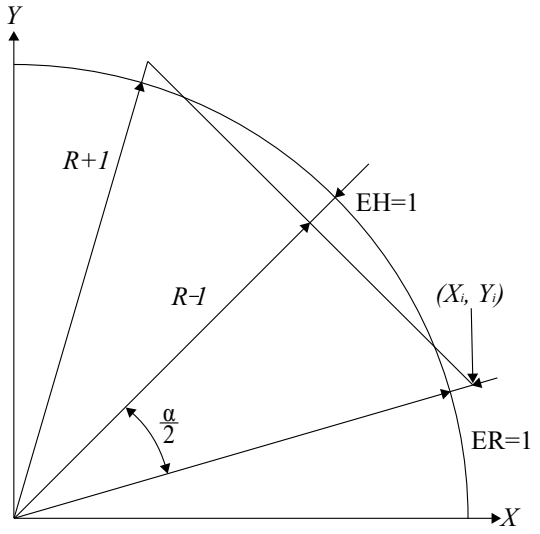


Fig. 3.17 Improved Tustin algorithm

$$\cos\frac{\alpha}{2} = \frac{R-1}{R+1} \tag{3.67}$$

$$\frac{R-1}{R+1} = \sqrt{\frac{2}{1+A}} = \sqrt{1+\left(\frac{\alpha}{2}\right)^2} \cong 1+\frac{\alpha^2}{8} \tag{3.68}$$

$$\alpha = \sqrt{\frac{16}{R-1}} \cong \frac{4}{\sqrt{R}} \tag{3.69}$$

$$N = \frac{\pi}{8}\sqrt{R} \tag{3.70}$$

As shown in Eq. 3.69, the angle α increases $\sqrt{2}$ times that in the Tustin algorithm. Also, the number of iteration steps for interpolating a quarter circle decreases, as indicated in Eq. 3.70. In the improved Tustin algorithm the radial error ER and the chord height error EH are not necessarily set to 1 BLU; *i.e.*, Eq. 3.71 can be derived for the general case when ER and EH are set to β BLU.

$$\begin{aligned} R_i &= R+\beta \\ \cos\frac{\alpha}{2} &= \frac{R-\beta}{R+\beta} \\ \alpha &= 2\cos^{-1}\left(\frac{R-\beta}{R+\beta}\right) \end{aligned} \tag{3.71}$$

3.3.2.9 Sampled-Data Interpolation

In the above sections, various algorithms for the Sampled-Data interpolation method were introduced. Their characteristics are summarized in Table 3.8, which shows that it is necessary to select the appropriate algorithm based on the application area. For example, if floating-point arithmetic is possible, the Improved Tustin may be a good selection because the number of iterations is relatively small and accuracy is guaranteed. If floating-point arithmetic is not possible, the Taylor algorithm may be adequate because the number of iterations is small and interpolating a circle with larger radius is possible.

Table 3.8 Sampled data interpolation method characteristics

Method	α	N
Euler	$4/\pi R$	$\pi^2 R/8$
IEM	$4/R$	$\frac{\pi}{4}\sqrt{R/2}$
Taylor	$\sqrt{8/R}$	$\frac{\pi}{4}\sqrt{R/2}$
Tustin	$\sqrt{8/R}$	$\frac{\pi}{4}\sqrt{R/2}$
ITM	$4/\sqrt{R}$	$\frac{\pi}{8}\sqrt{R}$

3.4 Fine Interpolation

When the sampling interval for rough interpolation and pulse train after acceleration/deceleration is larger than that of position control, fine interpolation is performed. For instance, if the sampling interval for rough interpolation and acceleration/deceleration is 4 ms, and the sampling interval for position control is 1 ms, then

the pulse train for every 4 ms is stored in the main CPU, which is fine-interpolated for every 1 ms by the CPU in charge of motion control.

There are methods for fine interpolation, the linear method where pulse train of 4 ms is divided into 1 ms, and the moving-average method where the moving average of pulse train is used for fine interpolation. Figure 3.18 shows a linear interpolation method, where pulse train of 4 ms is linearly divided into that of 1 ms.

Formally, Eq. 3.72 can be used for linear methods. In Eq. 3.72, $a(j)$ denotes the number of pulses from fine interpolation at arbitrary time j, and $p(i)$ is the number of the pulses from rough interpolation and Acc/Dec control at time i. t_{ipo} is the iteration time of rough interpolation and N is the ratio of the iteration time of rough interpolation and the iteration time of position control.

$$a(j) = \frac{p(i)}{N}, i \leq j < i + t_{ipo} \tag{3.72}$$

The second method is the moving average method. The equation used for the moving average can be represented by an iterative equation as shown in Eq. 3.73. In Eq. 3.72, $a(j)$ is from linear interpolation, and $b'(j)$ and $b''(j)$ are further interpolation for the moving average. Table 3.9 illustrates the computing procedure for the moving average.

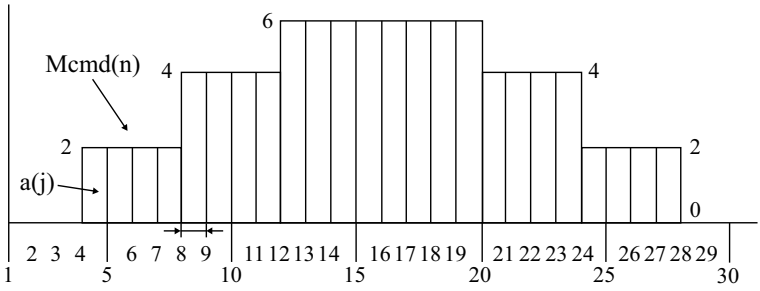


Fig. 3.18 Linear fine interpolation

Figure 3.19 shows the moving average of the pulse train shown in Fig. 3.18 and Table 3.9 gives the values from Fig. 3.19.

$$b(j) = \frac{\sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} a(j-k)}{N}, b'(j) = \frac{\sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} a(j-k)}{N}, b''(j) = \frac{b(j)+b'(j)}{2} \tag{3.73}$$

Table 3.9 Example of computing procedure for moving average

n	j	$a(j+4)$	$a(j+3)$	$a(j+2)$	$a(j+1)$	$a(j)$	$a(j-1)$	$a(j-2)$	$b(j)$	$b'(j)$	$b''(j)$
1	1	2	0	0	0	0	0	0	0	0	0
	2	2	2	0	0	0	0	0	0	0	0
	3	2	2	2	0	0	0	0	0	$\frac{1}{2}$	0.25
	4	2	2	2	2	0	0	0	$\frac{1}{2}$	1	0.75
2	5	4	2	2	2	2	0	0	1	$1\frac{1}{2}$	1.25
	6	4	4	2	2	2	2	0	$1\frac{1}{2}$	2	1.75
	7	4	4	4	2	2	2	2	2	$2\frac{1}{2}$	2.25
	8	4	4	4	4	2	2	2	$2\frac{1}{2}$	3	2.75
3	9	6	4	4	4	4	2	2	3	$3\frac{1}{2}$	3.25
	10	6	6	4	4	4	4	2	$3\frac{1}{2}$	4	3.75
	11	6	6	6	4	4	4	4	4	$4\frac{1}{2}$	4.25
	12	6	6	6	6	4	4	4	$4\frac{1}{2}$	5	4.75
4	13	6	6	6	6	6	4	4	5	$5\frac{1}{2}$	5.25
	14	6	6	6	6	6	6	4	$5\frac{1}{2}$	6	5.75
	15	6	6	6	6	6	6	6	6	6	6
	16	6	6	6	6	6	6	6	6	6	6
5	17	4	6	6	6	6	6	6	6	6	6
	18	4	4	6	6	6	6	6	6	6	6
	19	4	4	4	6	6	6	6	6	$5\frac{1}{2}$	5.75
	20	4	4	4	4	6	6	6	$5\frac{1}{2}$	5	5.25
6	21	2	4	4	4	4	6	6	5	$4\frac{1}{2}$	4.75
	22	2	2	4	4	4	4	6	$4\frac{1}{2}$	4	4.25
	23	2	2	2	4	4	4	4	4	$3\frac{1}{2}$	3.75
	24	2	2	2	2	4	4	4	$3\frac{1}{2}$	3	3.25
7	25	0	2	2	2	2	4	4	3	$2\frac{1}{2}$	2.75
	26	0	0	2	2	2	2	4	$2\frac{1}{2}$	2	2.25
	27	0	0	0	2	2	2	2	2	$1\frac{1}{2}$	1.75
	28	0	0	0	0	2	2	2	$1\frac{1}{2}$	1	1.25
8	29	0	0	0	0	0	2	2	1	$\frac{1}{2}$	0.75
	30	0	0	0	0	0	0	2	$\frac{1}{2}$	0	0.25
	31	0	0	0	0	0	0	0	0	0	0
	32	0	0	0	0	0	0	0	0	0	0

3.5 NURBS Interpolation

For high-speed and high-accuracy machining functions various interpolation functions such as splines, involute, and helical interpolation are used. In CNC free-form curves can be approximated by a set of line segments or circle arcs. However, to get an accurate approximation of the curve the approximating line or circle is generally very short. These short segments result in inconsistency of feedrate and this inconsistency of feedrate reduces the surface quality. In addition, many blocks are required to define these short paths and the size of the part program increases dramatically. To overcome this drawback, NURBS interpolation was developed. In NURBS interpo-

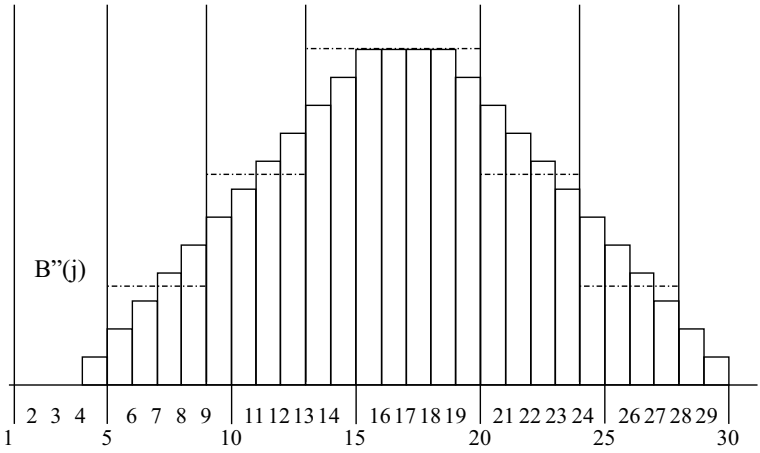


Fig. 3.19 Pulse train moving average

lation, the CNC itself directly converts NURBS curve data from the part program into small line segments, using positions calculated from the NURBS curve data. In this way it is possible to reduce the size of the part program and it is possible to increase the machining speed because the command feedrate depends on the interpolation.

3.5.1 NURBS Equation Form

There are various mathematical models such as cubic-spline, Bezier, B-spline, and NURBS to represent free-form curves. Among these, NURBS is the most general model covering others as special cases. With NURBS geometry it is possible to define free-form curves with complex shapes by using less data and to represent various geometric shapes by changing parameters. Nowadays, NURBS geometry is generally used in CAD/CAM systems.

The mathematical form of a NURBS curve is shown in Eq. 3.74.

$$P(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \qquad a \le u \le b \tag{3.74}$$

where $N_{i,p}(u)$ is a B-spline basis function and is defined as in Eq. 3.75

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u \le u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{3.75}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

In the above equation, the values u_i are termed "knots" and the NURBS curve has an associated 'knot vector', U. The knot vector U is defined as Eq. 3.76 and each value u_i in the knot vector is greater than or equal to the previous value, u_{i-1}.

$$U = u_0, \ldots, u_p, u_{p+1}, \ldots, u_{m-p-1}, u_{m-p}, \ldots, u_m \tag{3.76}$$

$$a = u_0 = \ldots = u_p (k \text{ multiplicity})$$
$$b = u_{m-p} = \ldots = u_m$$
$$m = n + p + 1$$

P denotes the degree of the B-spline basis function, P_i denotes control point i, and w_i stands for the 'weight' of P_i.

3.5.2 NURBS Geometric Characteristics

The characteristic of NURBS curves depends on that of the underlying B-spline basis function and can be summarized as follows:

- If $u \notin [u_i, u_{i+p+1})$, $N_{i,p}(u) = 0$
- If p and u are valid, $N_{i,p} \geq 0$
- $P(a) = P_0$, $P(b) = P_n$; the NURBS curve passes through the first control point and the last control point.
- $P(u)$ can be infinitely differentiated within defined parameter space and if u is u_i and multiplicity of knot u_i is k, $P(u)$ can be differentiated as many as $p-k$ times.
- The movement of control point P_i or the change of weight w_i affects the portion of the curve where the parameter $u \in [u_i, u_{i+p+1}]$.
- By adjusting the weight, the control point, and the knot vector of a NURBS curve it is possible to represent curves with various shapes.

In CAD systems, NURBS curves with degree 3 are mainly used. Also, multiplicity of the knot vector is usually 1 and the curve satisfies at least C2 continuity. These two characteristics result in good geometric properties. In modern CAD systems free-form shapes are represented using NURBS geometry.

The shape of a NURBS curve is defined based on control points, knots, and weights. Control points define the basic position of the curve. Weights decide the importance of individual control points. Knots decide the tangents of curves. Figure 3.20 shows the partial modification characteristics of a NURBS curve. Figure 3.20b shows the modified curve when control point V4 is moved. From Fig. 3.20b, the partial modification of curve is shown. Figure 3.20c shows the curve when the weight of control point V6 is changed.

Figure 3.21 shows the graphs that define a half-circle and a line by using a NURBS model. To represent the half circle shown in Fig. 3.21a, five control points (0,0), (0,5), (5,5), (10,5), (10,0) are used. There are five weights, one for each control

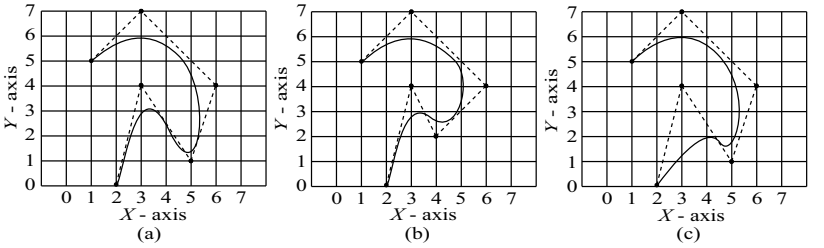


Fig. 3.20 NURBS curves

point $(1, \frac{1}{\sqrt{2}}, 1, \frac{1}{\sqrt{2}}, 1)$, and the knot vector (0,0,0,0.5,0.5,0.5,1,1,1) is used. The three 0s at the beginning and the three 1s at the one mean that the NURBS curve passes through the first and last control points. To represent a line shown in Fig. 3.21b, the following data are used:

Seven control points: (0,0), (1,1.5), (2,3), (3,4.5), (4,6), (5,7.5), (6,9)
weights for the control points: $(1,1,1,1,1,1,1)$
knot vector $(0,0,0,0,1,2,3,4,4,4,4)$

This shows that primitive curves such as lines and circles can be represented by NURBS geometry, as well as complex general curves.

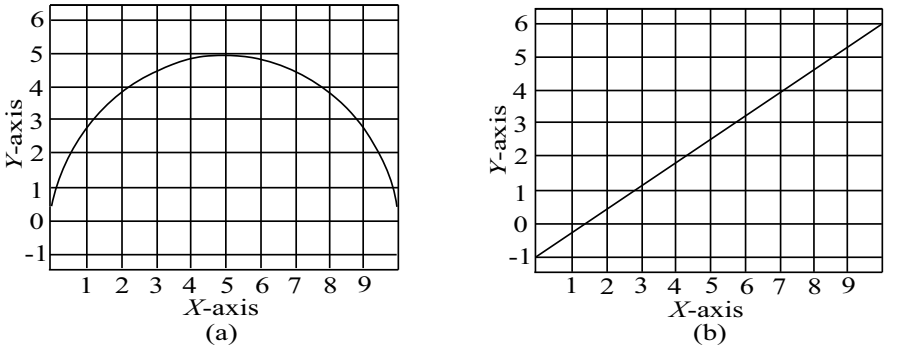


Fig. 3.21 NURBS line and circle

3.5.3 NURBS Interpolation Algorithm

The NURBS interpolation algorithm introduced in this section is suitable for the Sampled-Data interpolation method. This algorithm consists of 2 stages; In the first stage, successive interpolated points are obtained with a maximum allowable inter-

polation error. In the second stage, the interpolated point obtained from the first stage is checked to determine whether it exceeds the allowable acceleration. If necessary, a new interpolated point is calculated that satisfies the allowable acceleration. In the following sections, the detailed algorithms will be addressed.

3.5.3.1 NURBS Interpolation Errors

In the Sampled-Data interpolation method, the interpolation frequency is fixed and the speed is decided by the length of the interpolated line segment. The interpolation error varies according to the curvature of curve. The interpolation error h for a free-form curve is calculated as illustrated in Fig. 3.22. The center point of the line from the interpolated point (x_i, y_i) and the successive interpolated point (x_{i+1}, y_{i+1}) is compared with the midpoint of the curve between the points (x_i, y_i) and (x_{i+1}, y_{i+1}), denoted (x_c, y_c). If the interpolation error, h, is greater than the maximum allowable interpolation error (ε_{max}) this means that the curvature is too high to satisfy the maximum allowable interpolation error, the next interpolated point moves closer to the current interpolated point (x_i, y_i). The new interpolated point (x'_{i+1}, y'_{i+1}) is closer to the interpolated point (x_i, y_i).

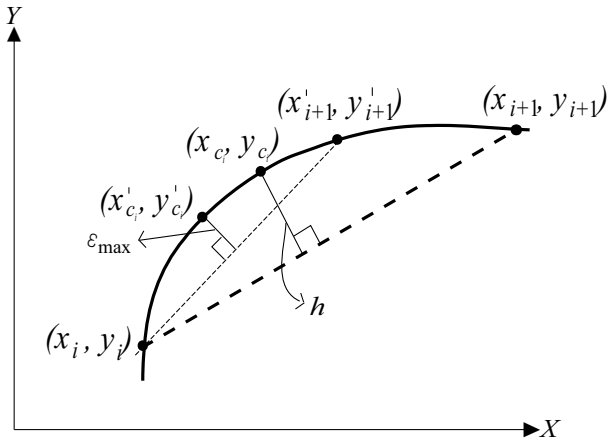


Fig. 3.22 Adapting step length to curvature

Figure 3.23 represents the definition of the curvature at a particular point on a free curve and Eq. 3.77 shows the curvature k and radius R.

$$\kappa \equiv \lim_{\Delta s \to 0} \frac{\Delta \phi}{\Delta s},$$
$$R \equiv \frac{1}{\kappa} \tag{3.77}$$

where κ : is the curvature
R : is the radius of curvature
$\Delta\phi$: angle at the circumference
Δs : the length of the partial curve

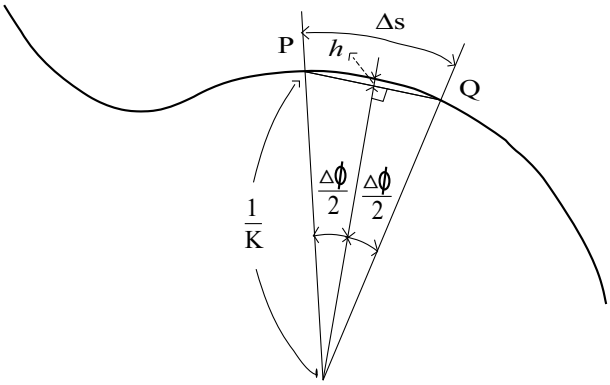


Fig. 3.23 Curvature definition

A curve PQ is regarded as the part of a circle with radius R and curvature κ. If we define P and Q as two successive interpolated points and the distance between the line and the circle as an interpolation error, the relationship between $\Delta\phi$, h, and k can be summarized as Eq. 3.78 based on Eq. 3.77.

$$h = \frac{1}{\kappa} - \frac{1}{\kappa} \times \cos\left(\frac{\Delta\phi}{2}\right) \tag{3.78}$$
$$\cong \frac{1}{\kappa} - \frac{1}{\kappa} \times \left[1 - \frac{1}{2} \times \left(\frac{\Delta\phi}{2}\right)^2\right] = \frac{\Delta\phi^2}{8\kappa}$$

In Eq. 3.78, the cosine function is approximated by a second-order Taylor series expansion. The angle at the circumference $(\Delta\phi)$ can be written as Eq. 3.79.

$$\Delta\phi = 2\sqrt{2\kappa h} \tag{3.79}$$

If the length of the partial curve Δs is approximated by the length of the line $\overline{PQ}$, the curvature is summarized as Eq. 3.80 based on Eq. 3.77 and Eq. 3.79.

$$\kappa \cong \frac{\Delta\phi}{\Delta s} \cong \frac{\Delta\phi}{\overline{PQ}} \cong \frac{\sqrt{8\kappa h}}{\overline{PQ}} \tag{3.80}$$

$$\kappa = \frac{8h}{\overline{PQ}^2} \tag{3.81}$$

From Eq. 3.81, the approximated relationship between the command feed-rate F, the iteration time for an interpolation (ΔT), curvature κ, and interpolation error h can be summarized as in Eq. 3.82.

$$h \cong \kappa \times \frac{\overline{PQ}^2}{8} = \kappa \times \frac{(F \times \Delta T)^2}{8} \tag{3.82}$$

where F : is the feedrate

ΔT : is the interpolation iteration time

$\overline{PQ} : F \times \Delta T$

Equation 3.82 says that the interpolation error (h) is proportional to the curvature κ. If interpolated points are calculated with constant feedrate, an interpolation error grows on curve parts with large curvature. Therefore, it is necessary to reduce the interpolation error with a reduction of feedrate on the highly curved path portions.

From Eq. 3.82, in order to compute the feedrate at which the interpolation error h falls within the maximum allowable interpolation error ε_{max}, the curvature k of the partial curve that connects a current interpolated point and a successive interpolated point should be computed.

Set the current interpolated point to $P(u_i)$ by Eq. 3.74 and the next interpolated point to $P(u_{i+1})$. To calculate the speed to $P(u_{i+1})$ from $P(u_i)$, it should be assumed that the previous interpolated points $P(u_{i-2})$, $P(u_{i-1})$, and $P(u_i)$ are located on the same circle.

It is assumed that the successive interpolated point $P(u_{i+1})$ is located on the same circle. Based on these assumptions, the curvature of the partial circle from $P(u_i)$ and $P(u_{i+1})$ is extrapolated from the curvature of the circle defined from $P(u_{i-2})$, $P(u_{i-1})$, and $P(u_i)$.

Using the maximum allowable interpolation error ε_{max}, the iteration time for an interpolation T, and the approximated curvature κ, the speed between $P(u_i)$ and $P(u_{i+1})$ can be computed.

From Eq. 3.82, the speed F_ε that satisfies the maximum allowable interpolation error ε_{max}, can be computed by Eq. 3.83.

$$h = \varepsilon_{max} \tag{3.83}$$

$$F_\varepsilon = \frac{2}{\Delta T}\sqrt{\frac{2\varepsilon_{max}}{\kappa}}$$

where ε_{max} : maximum allowable interpolation error

F_ε : allowable feedrate speed

It is necessary to define the relationship between the parameter variable u of the NURBS geometry and the feedrate $F_\varepsilon(t)$. The linear length between the current interpolated point and the next interpolated point, ΔL, can be computed using Eq. 3.84. We assume that the linear length from Eq. 3.84 is identical to the length of partial curve. Noting that ΔL in Eq. 3.84 is defined in a time domain, we need to find its equivalent value in the parametric domain to find the next interpolation point on the NURBS curve. Let $P(u)$ be the current point and we would like to find the next point $P(u+\Delta u)$. Then, using the property of ΔL in Eq. 3.84 and assuming the three points in Cartesian space are close enough, we can approximate ΔL as shown in Eq. 3.85.

$$\Delta L = F_\varepsilon(t) \times \Delta T = \sqrt{\frac{8 \times \varepsilon_{max}}{\kappa}} \tag{3.84}$$

$$\begin{aligned} P(u-\Delta u) &\cong P(u) \cong P(u+\Delta u) \\ \Delta L &= F_\varepsilon(T) \times \Delta T = P(u) \times \Delta u \end{aligned} \tag{3.85}$$

Thus, the parameter variable increment Δu can be computed by Eq. 3.86.

$$\begin{aligned} \Delta u = \frac{\Delta L}{P(u)} &\cong F_\varepsilon(t) \times \Delta T \times \frac{1}{P(u-\Delta u)} \\ &\sqrt{\frac{8 \times \varepsilon_{max}}{\kappa}} \times \frac{1}{P(u-\Delta u)} \end{aligned} \tag{3.86}$$

3.5.3.2 Acceleration Control keeping Axis-Velocity Limit

In the previous section we determined the next interpolation point based on the maximum error allowed for interpolation. In the case that the rate of velocity is changed sharply to beyond the acceleration for the joint abrupt motion will be caused, resulting in machine damage and machining operation failure. To avoid such a problem, the distance of the next point to be interpolated should be shortened based on Eq. 3.87. ΔL computed in this way should be applied to Eq. 3.84 and Eq. 3.85.

$$\begin{aligned} \text{if} \quad & \frac{|\frac{\Delta X}{\Delta T_i} - \frac{\Delta X}{\Delta T_{i+1}}|}{\Delta T} > A_{max} \\ \left(\frac{\Delta X}{\Delta T_{i+1}}\right) &= \frac{\Delta X}{\Delta T_i} + A_{max} \times \Delta T \quad (when \frac{\Delta X}{\Delta T_i} - \frac{\Delta X}{\Delta T_{i+1}} > 0) \\ \left(\frac{\Delta X}{\Delta T_{i+1}}\right) &= \frac{\Delta X}{\Delta T_i} - A_{max} \times \Delta T \quad (when \frac{\Delta X}{\Delta T_i} - \frac{\Delta X}{\Delta T_{i+1}} < 0) \\ \Delta L_{new} &= \sqrt{\left(\frac{\Delta X}{\Delta T_{i+1}}\right)^2 + \left(\frac{\Delta Y}{\Delta T_{i+1}}\right)^2} \end{aligned} \tag{3.87}$$

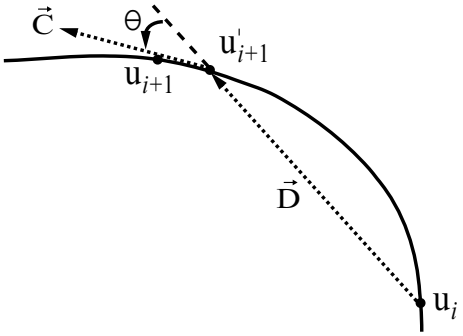


Fig. 3.24 Difference between $c(u+1)$ and $c(u)$

3.6 Summary

The type of interpolators is categorized into hardware interpolators or software interpolators, depending on the implementation method. Before CNC systems were developed, a hardware interpolator was widely used in NC systems, but in today's CNC systems, this is carried out by a software interpolator.

A DDA interpolator is a typical hardware interpolator and was used for a long time, but is not much used in today's CNC systems. In modern CNC systems, a software DDA interpolator is used, where the algorithm of the hardware DDA interpolator is implemented in software. A software interpolation method can be classified into a reference pulse method and a sampled-data interpolation method. A reference-pulse method is suitable for high-accuracy machining and a sampled-data interpolation method is suitable for high-speed machining. Due to the demand for high-speed machining, the sampled-data interpolation method is typically used in today's CNC system.

In this chapter, various algorithms for the reference-pulse interpolation method are explained, including, the software-DDA interpolation algorithm, Stairs Approximation Interpolation algorithm, Direct Search interpolation algorithm, Sampled-data interpolation method, Tustin interpolation algorithm, improved Tustin interpolation algorithm, Euler interpolation algorithm, improved Euler interpolation algorithm, and Taylor interpolation algorithm. These algorithms need to be developed as modules so that the appropriate algorithm can be selected based on the application of the CNC system.

Interpolation algorithms for free-form curves are important for solving problems existing in CNC systems where the free-form curve is approximated by a number of infinitesimal linear segments. Via the NURBS interpolation method, still under investigation, problems such as speed reduction, poor surface quality and poor machining accuracy have been solved.

Chapter 4
Acceleration and Deceleration

In order to smooth the movement of a machine, the acceleration and deceleration for the movement of the machine axes should be controlled. For CNC systems, two kinds of Acceleration and Deceleration (Acc/Dec) control methods have been developed; Acc/Dec Control Before Interpolation (ADCBI) and Acc/Dec Control After Interpolation (ADCAI). These are classified based on the order in which the Acc/Dec control is executed. In this chapter, first we will introduce the Acc/Dec control after interpolation that was originally used for NC systems. Following this, we will introduce the Acc/Dec control before interpolation which is suitable for high-speed and high-accuracy machining. In addition, a Look Ahead Algorithm will be addressed that is used with the Acc/Dec control before interpolation for Die and Mold machining.

4.1 Introduction

The Acc/Dec control method can be classified with an Acc/Dec control before interpolation and an Acc/Dec control after interpolation with respect to the processing order of acceleration and deceleration control.

The Acc/Dec control before interpolation (ADCBI) is constructed differently according to the interpolation type such as linear-, exponential- and S-curve-type interpolation. Because the ADCBI needs to hold a lot of information, related to all the interpolated points, a large amount of memory is required for executing this type of Acc/Dec control. However, because of the large amount of information the Acc/Dec control does not result in machining error because of the increased accuracy.

On the other hand, Acc/Dec control after interpolation is applied in an identical manner for all interpolation methods. Therefore, the implementation is simple but machining errors occur because each axis movement is determined separately. Since Acc/Dec control in ADCAI is individually applied for each axis, acceleration and deceleration for movements of each axis are carried out regardless of the interpolated position. Accordingly, the interpolated points deviate from the desired path. A typical

example of this deviation occurs during the corner machining process and the longer the Acc/Dec time, the larger the machining error.

In contrast, machining errors due to ADCBI do not occur because the command path is identical to the desired path. The key for executing Acc/Dec control before interpolation is finding the time of acceleration and deceleration timing based on the commanded feedrate, remaining displacement of the path, an allowable acc/dec value, and current velocity. Therefore, ADCBI requires more computing power and larger memory than ADCAI. From the point of view of real implementation, ADCBI is much more complex than ADCAI.

In Section 4.2, both software and hardware types of the Acc/Dec control after interpolation will be addressed. In addition, in Section 4.3, Acc/Dec control before interpolation will be explained, including the block overlap algorithm, the velocity control method at corners, and a Look Ahead algorithm.

4.2 Acc/Dec Control After Interpolation

In the case of ADCAI, firstly the NCK, Numerical Control Kernel, interprets a part program using the interpreter module and calculates the displacement distance for each axis, ΔX, ΔY, ΔZ for every interpolation time interval based on the interpreted result using the rough interpolation module. Next, independent Acc/Dec control of each axis is performed with respect to ΔX, ΔY, ΔZ and the fine interpolation then follows. Finally, the total remaining displacement of each axis for every position control time interval is calculated by the position control module.

The Acc/Dec control algorithm of Acc/Dec control after interpolation is different from that of Acc/Dec control before interpolation. Figure 4.1 shows the flowchart for implementing the NCK with Acc/Dec control after interpolation. The big difference with Acc/Dec control before interpolation is that the remaining displacement of each axis is calculated at each interpolation time by rough interpolation and Acc/Dec control of each axis is performed individually. Figure 4.2 shows the change of pulse profile after the Acc/Dec control. It can be seen that the individual pulse profile of each axis is generated by the rough interpolator and the individual acceleration and deceleration scheme is applied to each pulse profile.

The ADCAI method has been widely used for NC and Motion Control systems in both hardware and software interpolation. Regardless of the implementation manner, the algorithm itself is not different between hardware and software types. In this textbook, the theory of Acc/Dec control (software approach) and the hardware implementation method (hardware approach) will be addressed.

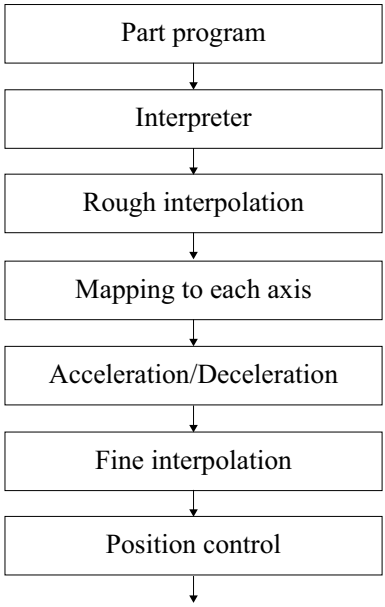


Fig. 4.1 NCK functional procedure with ADCAI

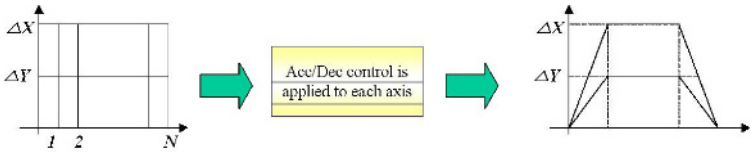


Fig. 4.2 Change to pulse profile after Acc/Dec control

4.2.1 Acc/Dec Control by Digital Filter

The Acc/Dec control algorithm for the ADCAI method is based on digital filter theory. According to the digital filter theory, if input signal $x[n]$ is entered into the filter with impulse response $h[n]$, the output signal $y[n]$ is represented by the convolution of h[n] and x[n]. Equation 4.1 shows the general convolution of $f_1[n]$ and $f_2[n]$ for a discrete time system.

$$\begin{aligned} f[n] &= f_1[n] * f_2[n] \\ &= f_1[0]f_2[n] + \ldots + f_1[k]f_2[n-k] + \ldots + f_1[n]f_2[0] \end{aligned} \tag{4.1}$$

Equation 4.1 can be written as Eq. 4.2.

$$f[n] = f_1[n] * f_2[n] = \sum_{k=1}^{n} f_1[k] * f_2[n-k] \tag{4.2}$$

As shown in Fig. 4.3, if we assume that $x[n]$ is defined as the output of a rough interpolator and $h[n]$ as the impulse response that has the normalized unit summation, as shown in Eq. 4.3 we can obtain the Acc/Dec pulse profile in which the summation of the input signal is the same as the summation of the output signal after convolution of $x[n]$ and $h[n]$.

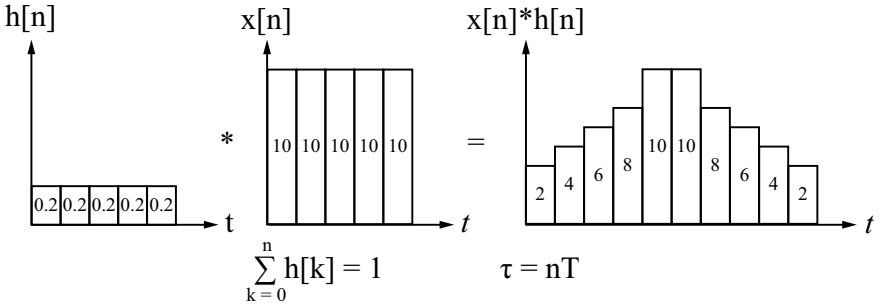


Fig. 4.3 Convolution of rough interpolation and impulse response

Where the Acc/Dec time τ is the multiplication of n and the sampling time T for continuous convolution.

$$\sum_{k=1}^{n} h[k] = 1 \tag{4.3}$$

Further, since $h[n]$ denotes the differentiation of velocity, *i.e.* acceleration, we can obtain the various Acc/Dec pulse profiles by changes of $h[n]$. The Linear-type, Exponential-type, and S-curve-type, as shown in Fig. 4.4, are used for the Acc/Dec filters of CNC systems. By using various digital filters different output profiles can be obtained even when identical input pulses are used.

Equation 4.4, Eq. 4.5, and Eq. 4.6 represent the Linear-type filter, the Exponential-type filter, and S-shape-type filter. Here, T means the sampling time and τ denotes the time constant for Acc/Dec control.

$$H_L(z) = \frac{1}{m}\frac{1-z^{-m}}{1-z^{-1}} \tag{4.4}$$

$$H_E(z) = \frac{1-\alpha}{1-\alpha z^{-1}} \quad \text{where } \alpha = exp^{-\frac{T}{\tau}} \tag{4.5}$$

$$H_S(z) = H_L(z) * H_L(z) = \frac{1}{m}\frac{1-z^{-m}}{1-z^{-1}} * \frac{1}{m}\frac{1-z^{-m}}{1-z^{-1}} \tag{4.6}$$

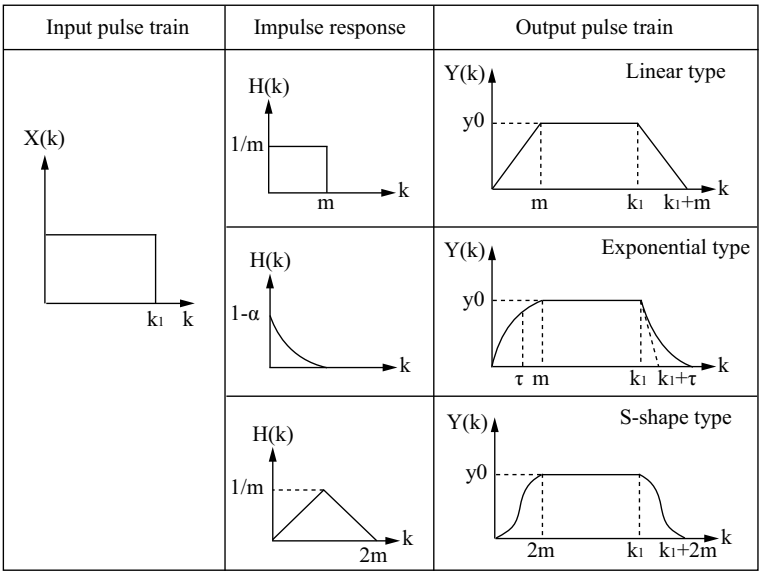


Fig. 4.4 Input and output pulse train profiles

Consequently, the Acc/Dec pulse profile generated by passing the input signal V_i through the above-mentioned filters can be represented by a recursive equation. Equation 4.7, Eq. 4.8, and Eq. 4.9 are recursive equations for obtaining the linear-type Acc/Dec pulse profile, the exponential-type Acc/Dec pulse profile, and the S-shape-type Acc/Dec pulse profile, respectively,

$$V_{LO}(k) = \frac{1}{m}(V_i(k) - V_i(k-m)) + V_0(k-1) \tag{4.7}$$

$$V_{EO}(k) = (1-\alpha)(V_i(k) - V_i(k-1)) + V_0(k-1) \tag{4.8}$$

$$V_{SO}(k) = \frac{1}{m}(V_i(k) - V_i(k-m)) + V_{Otemp}(k-1) \tag{4.9}$$

$$\text{where } V_{Otemp}(k) = \frac{1}{m}(V_{Otemp}(k) - V_{Otemp}(k-m)) + V_O(k-1)$$

Accordingly, the software Acc/Dec control algorithm is a relatively simple recursive equation and, therefore, has the merit of short calculation time.

4.2.2 Acc/Dec Control by Digital Circuit

Since the processing time of the Acc/Dec control method based on a digital circuit is very short, it has been used when the performance of CPUs was low. Hardware devices such as a shift register, a divider and an accumulator are used for implementing the Linear-type Acc/Dec control, the Exponential-type Acc/Dec control, and the S-shape Acc/Dec control. However, as CPU performance has improved, the hardware-type Acc/Dec control has been replaced by the software type Acc/Dec control that includes the same processing step as for the digital circuit.

In the ADCAI method, the pulse profile from rough interpolation is used as input of the Acc/Dec control circuit. The Acc/Dec control circuit plays the role of smoothing the change of pulse amount at the beginning and the end of a pulse profile.

In the following sections, three kinds of the Acc/Dec control algorithm will be addressed; a Linear-type Acc/Dec control, an Exponential-type Acc/Dec control and an S-shaped Acc/Dec control.

4.2.2.1 Linear-type Acc/Dec Control

The circuit for a Linear-type Acc/Dec control consists of n buffer registers (#1, #2, ..., #n), an Adder, an Accumulator, a SUM register, and a Divider. An Accumulator stores the output of an Adder, a SUM register stores the value of an Accumulator, and a Divider divides the value of an Accumulator by the number of buffer registers, n, where the buffer registers are serially connected. Each buffer register stores pulses from rough interpolation. The value of each buffer register shifts to the next buffer register every Acc/Dec control sampling time point (interpolation sampling time point).

As shown in Fig. 4.5 the value of buffer register #n is input to an Adder, the value of buffer register #$n-1$ is shifted to buffer register #n, the value of register #$n-2$ is shifted to buffer register #$n-1$ and so on. Finally, the most recent output of the rough interpolator ΔX is input to buffer register #1.

Based on the behavior of this circuit, at arbitrary sampling time k, the value of the Accumulator and the output of the Divider ΔX_o can be written as Eq. 4.10.

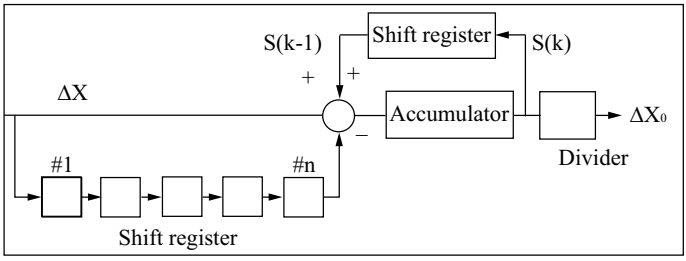


Fig. 4.5 Hardware Adder, Accumulator, SUM, Divider unit connections

$$S(k) = S(k-1) + \Delta X(k) - \Delta X(k-n) \tag{4.10}$$
$$\Delta X_o(k) = S(k)/n$$

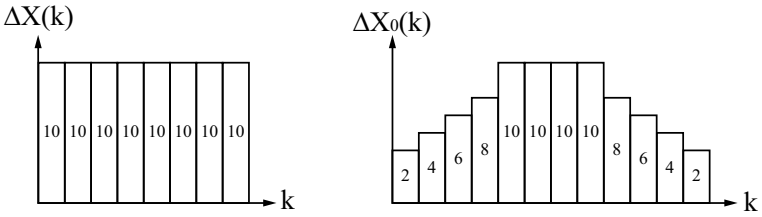


Fig. 4.6 Profiles $\Delta X = 10$, $\sum \Delta X = 80$

Let us explain the behavior of a Linear-type Acc/Dec control with an example. We assume that the sampling time T is 8 ms, the number of buffer registers, n, is 5, and the number of pulses ΔX that are entered into the circuit at every sampling time is ten, the output pulse profile of the circuit is shown in Table 4.1.

Table 4.1 Circuit output pulse profile

Sampling time: K	Input pulse: $\Delta X(k)$	Output of buffer register: $\Delta X(k-5)$	Output of Adder: $S(k)$	Output Pulse: $\Delta X_0(k)$
1	10	0	10	2
2	10	0	20	4
3	10	0	30	6
4	10	0	40	8
5	10	0	50	10
6	10	10	50	10
7	10	10	50	10
8	10	10	50	10
9	0	10	40	8
10	0	10	30	6
11	0	10	20	4
12	0	10	10	2
13	0	10	0	0
Σ	80			80

As shown in Fig. 4.3, initial output pulses increase or decrease by a constant number. The total number of input pulses (in this example, 80) is identical with the total number of output pulses from the circuit. After the number of input pulses becomes 0, we can see that the number of output pulses begins to decrease. This means that the acceleration mode continues until the buffer registers become full,

and then deceleration mode until the buffer registers are empty. Accordingly, we know that the number of buffer registers is proportional to the Acc/Dec time and the relationship between the Acc/Dec time constant and the number of buffer registers, n, can be written as Eq. 4.11.

$$\tau = nT \tag{4.11}$$

where, T denotes the sampling time and, in the case of the above example, the Acc/Dec time constant is 40 ms (5×8ms).

Alternatively, if the size of the input pulse train is smaller than the number of buffer registers, the maximum number of output pulses is different from the number of input pulses. For example, if the number of input pulses, ΔX, is 10 and the size of the input pulse train is 4, the output pulse train is obtained as Table 4.2.

Table 4.2 Output pulse train

Sampling time: K	Input Pulse: $\Delta X(k)$	Output of: Buffer Reg.: $\Delta X(k-5)$	Output of Adder: $S(k)$	Output Pulse: $\Delta Xo(k)$
1	10	0	10	2
2	10	0	20	4
3	10	0	30	6
4	10	0	40	8
5	0	0	40	8
6	0	10	30	6
7	0	10	20	4
8	0	10	10	2
9	0	10	0	0
Σ	40			48

As shown in Table 4.2, the maximum value of output pulses is 8 and this is less than the number of input pulses, 10. This means that a short block, having insufficient pulses, cannot reach the commanded (desired) speed. Accordingly, deceleration is started before full acceleration is developed because of the short length of the block.

4.2.2.2 S-shape-type Acc/Dec Control

Figure 4.7 shows an S-shape-type Acc/Dec control circuit for one axis. The circuit consists of n buffer registers, n Multipliers, an Adder, and a Divider.

In Fig. 4.7, S_1, S_2, ..., and S_n denote the buffer registers functioning as a shift register. ΔX denotes a recent output pulse from the rough interpolator that is input to the buffer registers. As shown in the Linear-type Acc/Dec control circuit, ΔX values stored in a buffer register are shifted at every sampling time ($S_1 \rightarrow S_2, S_2 \rightarrow S_3, \cdots, S_{n-1} \rightarrow S_n$). Further, multipliers (M_1, ..., M_n) have their own coefficients (K_1,

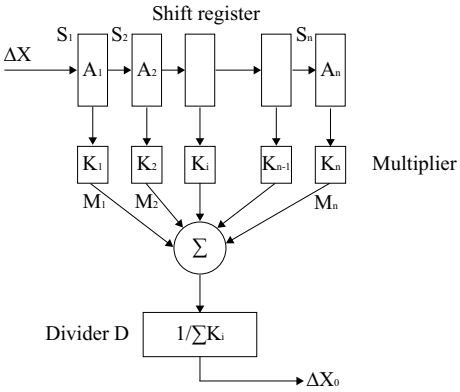


Fig. 4.7 S-shape-type Acc/Dec control

..., K_n) and the values of each buffer register are multiplied and addition and division follow for the circuit ΔX_o. Here, the dividing constant is defined as in Eq. 4.12.

$$\text{Dividing constant} = \sum_{i=1}^{n} K_i \tag{4.12}$$

Also, at arbitrary sampling time k, if the values of buffer registers $S_a \sim S_n$ are $A_1 \sim A_n$, then the value of an Adder can be written as Eq. 4.13.

$$\text{Adder} = \sum_{i=1}^{n} A_i \times K_i \tag{4.13}$$

Consequently, the output of the circuit is summarized as Eq. 4.14.

$$\Delta X_o = \frac{\text{Adder}}{\text{Divider}} = \frac{\sum_{i=1}^{n} A_i \times K_i}{\sum_{i=1}^{n} K_i} \tag{4.14}$$

An example of the behavior of an S-shape-type Acc/Dec control circuit is explained as follows. As shown in the example of the Linear-type Acc/Dec control circuit, we assume that the input to the circuit, *i.e.* the output of a rough interpolator, is the same as Table 4.3, the number of buffer registers is five, and all the coefficients of the Multipliers are one. Accordingly, the dividing constant is set to 5 ($1 + 1 + 1 + 1 + 1 = 5$). When the initial values of the shift registers are 0, the behavior of the circuit is simulated as shown in Table 4.4

Comparing Table 4.1 with Table 4.3, we know that the output of the S-shape-type Acc/Dec control circuit is identical with the output of the Linear-type Acc/Dec control circuit because the coefficients of all Multipliers are set to 1. If the coefficient of the Multipliers, K_i, are identical S-shape Acc/Dec control results in the same performance as Linear Type Acc/Dec control. Because of this fact we know that the coefficients of the multipliers, K_i, have a close relationship with the acceleration and deceleration. Therefore, it is possible to modify particular acceleration and deceler-

Table 4.3 S-shape output pulses

Sampling time: K	Input pulse: $\Delta X(k)$	Output of Adder:	Output Pulse: $\Delta X_o(k)$
1	10	10	2
2	10	20	4
3	10	30	6
4	10	40	8
5	10	50	10
6	10	50	10
7	10	50	10
8	10	50	10
9	0	40	8
10	0	30	6
11	0	20	4
12	0	10	2
Σ	80		80

ation curves by changing various coefficients of the multipliers, K_i. Table 4.4 shows the behavior of the S-shape Type Acc/Dec control circuit simulation with $K_1 = 0.5$, $K_2 = 1$, $K_3 = 2$, $K_4 = 1$, $K_5 = 0.5$. We can see that the summation of input pulses and the summation of output pulses are identical and the output of the circuit shows a non-uniform shape.

Figure 4.8 shows the result given in Table 4.4 in graphic form, from which we can conclude that the output of the circuit is depicted as an S-shape curve.

Table 4.4 S-shape pulse profile

Sampling time: K	Input Pulse: $\Delta X(k)$	Output of Adder:	Output Pulse: $\Delta X_o(k)$
1	10	5	1
2	10	15	3
3	10	35	7
4	10	45	9
5	10	50	10
6	10	50	10
7	10	50	10
8	10	50	10
9	0	45	9
10	0	35	7
11	0	15	3
12	0	5	1
Σ	80		80

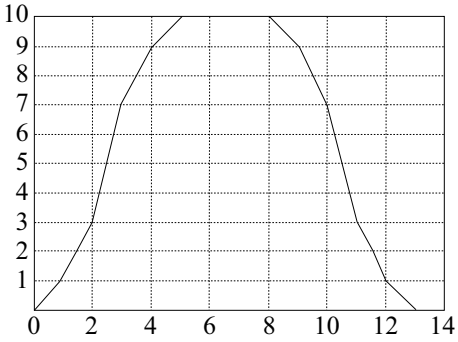


Fig. 4.8 Output of circuit for S-shape curve

4.2.2.3 Exponential-type Acc/Dec Control

Figure 4.9 shows the Exponential-type Acc/Dec control circuit that is constructed using DDAs unlike the Linear-type Acc/Dec control circuit and the S-shape-type Acc/Dec control circuit.

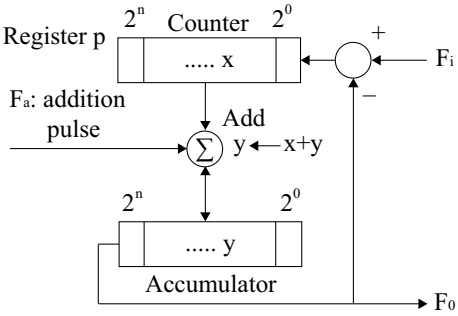


Fig. 4.9 Exponential-type Acc/Dec control circuit

The fundamental concept of the Exponential-type Acc/Dec control is as follows.

First, pulses with constant frequency $[F_i]$ are generated from a pulse generator as shown in Fig. 4.10. The number of generated pulses, N, determines the displacement of an axis and the frequency, F_i, determines the speed of that axis.

The value obtained by subtracting the output from the accumulator from the pulse of the pulse generator is stored in register p. At the same time, the value of register p (x) and the value of the accumulator (y) are added whenever Fa is generated, where Fa is generated at constant time intervals. Further, the sum of the value of the register, x, and the value of the accumulator, y, are stored in the accumulator ($y \leftarrow x + y$). In this circuit, an adder (ADD) and the register p can be regarded as an Up/Down

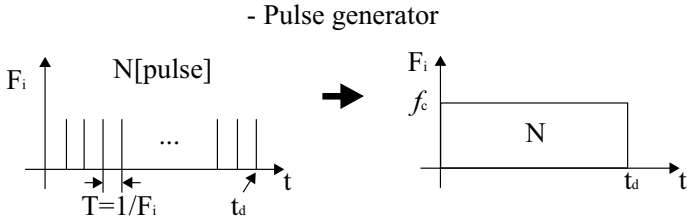


Fig. 4.10 Constant-frequency pulse generation

Counter. This can be simulated by connecting the output of the pulse generator as input to an Up port and the output from the accumulator as input to a Down port.

This Acc/Dec control circuit used a DDA (Digital Differential Analyzer) element when the performance of microprocessors was insufficient for fast calculation.

The principle of an Exponential-type Acc/Dec control is as follows:

If the output of the pulse generator is f_c[pulses/second], the input pulse of the Exponential-type Acc/Dec control circuit is written as Eq. 4.15.

$$F_i = \begin{cases} f_c \ (0 \le t \le t_d) \\ 0 \ \ t > t_d \end{cases} \tag{4.15}$$

During a short time segment Δt, the number of pulses, Δx, which is stored in register p, is written as Eq. 4.16.

$$\Delta X = (F_i - F_o)\Delta t \tag{4.16}$$

The number of pulses stacked at the accumulator during a short time segment Δt can be represented by the value of register p to which is added the number of addition command pulses ($F_a * \Delta t$) generated during Δt. Accordingly, ΔY can be summarized as in Eq. 4.17.

$$\Delta Y = XF_a\Delta t \tag{4.17}$$

The number of pulses that overflow from the accumulator for Δt is calculated as $F_o\Delta t$. If the number of bits in the accumulator is assumed to be n, the number of overflow pulses is $\Delta y/2^n$ so that Eq. 4.18 is derived as:

$$F_o\Delta t = \frac{\Delta Y}{2^n} \tag{4.18}$$

When t is 0, the values of the register p and the accumulator are zero, then this is represented by Eq. 4.19.

$$X(0) = Y(0) = 0 \tag{4.19}$$

If we approximate Δt, ΔX, and ΔY with dt, dx, and dy respectively, Eq. 4.16, Eq. 4.17, Eq. 4.18, and Eq. 4.19 can be expressed as Eq. 4.20.

$$\begin{aligned} dx &= (F_i - F_o)dt \\ dy &= xF_a dt \\ F_o dt &= \frac{dy}{2^n} \\ X(0) &= Y(0) = 0 \end{aligned} \tag{4.20}$$

By solving the differential equation in Eq. 4.20, we can derive the relationship between F_i and F_o as shown in Eq. 4.21.

$$\begin{aligned} &(i)\ 0 \le t \le t_d \\ &\quad F_o(t) = \ F_i(1 - e^{-\frac{F_a}{2^n}t}) = F_i(1 - e^{-\frac{t}{X}}), X = \frac{2^n}{F_a} \\ &(ii)\quad t > t_d \\ &\quad F_o(t) = \ F_i(1 - e^{-\frac{t_d}{X}})e^{-\frac{(t-t_d)}{X}} = F_o(t_d)e^{-\frac{(t-t_d)}{X}} \end{aligned} \tag{4.21}$$

From Eq. 4.21, it is evident that Acc/Dec control is executed in an exponential form. The number of bits in the accumulator in Eq. 4.21 determines the acceleration and deceleration time. When the number of Accumulator bits is larger, the number of the overflow pulses per unit time decreases and consequently the acceleration and deceleration time increases. Also, if the frequency of the addition pulse decreases, the time constant increases and the Acc/Dec time grows. This is due to the fact that the number of pulses that are added to the Accumulator per unit time decreases, which in turn means that the number of the overflowed pulses decreases.

If we represent the sampling time and output frequency of the pulse generator as T and F_i respectively, the number of pulses input to the Exponential-type Acc/Dec control circuit every sampling time is given by Eq. 4.22.

$$P_i = T \times F_i \tag{4.22}$$

For example, if the sampling time T is 8 ms and the frequency of the pulse generator is 1000 Hz, 8 pulses are input to the circuit every sampling time. At arbitrary moment k, the value of register p can be written as Eq. 4.23 when pulses overflowing from the Accumulator are not subtracted.

$$\hat{x}(k) = x(k-1) + P_i(k) \tag{4.23}$$

where, $x(k-1)$ is the value of register p at $k-1$.

At the arbitrary moment, k, the number of the pulses that have overflowed from the accumulator, $O(k)$, is given by Eq. 4.24.

$$O(k) = \frac{y(k)}{2^n} \tag{4.24}$$

where, n is the number of bits of the Accumulator.

Therefore, at the arbitrary moment, k, the number of overflow pulses, $(P_o(k))$, is given by Eq. 4.25.

$$P_o(k) = O(k) - O(k-1) \tag{4.25}$$

Furthermore, at the arbitrary moment, k, the value of register p, $(x(k))$, after subtracting the pulses that have overflowed from the accumulator can be written as Eq. 4.26.

$$x(k) = \hat{x}(k) = P_o(k) \tag{4.26}$$

At the arbitrary moment, k, the residual value of the accumulator, $(y(k))$ is given by Eq. 4.27.

$$y(k) = y(k-1) + \hat{x}(k) \tag{4.27}$$

By using the Z-transformation, the equation set Eq. 4.23, Eq. 4.24, Eq. 4.25, Eq. 4.26 and Eq. 4.27 is converted to Eq. 4.28.

$$\begin{aligned} \hat{X}(z) &= \frac{1}{z}X(k) + P_i(z) \\ Y(z) &= \frac{1}{z}Y(z) + \hat{X}(z) \\ O(z) &= \frac{Y(z)}{2^n} \\ P_o(z) &= O(z) - \frac{1}{z}O(z) \\ X(z) &= \hat{X}(z) - P_o(z) \end{aligned} \tag{4.28}$$

From Eq. 4.28, the relationship between the input P_i and the output P_o is derived as in Eq. 4.29.

$$P_o(z) = \frac{1}{z}P_o(z) + \frac{P_i(z) - \frac{1}{z}P_o(z)}{2^n} \tag{4.29}$$

Consequently, by executing the inverse Z-Transformation, Eq. 4.29 is transformed into Eq. 4.30.

$$P_o(k) = P_o(k-1) + \frac{P_i(k) - P_o(k-1)}{2^n} \tag{4.30}$$

With iterative calculation of Eq. 4.30 for every sampled datum, Exponential-type Acc/Dec control is performed wherein the Acc/Dec time constant τ is defined by Eq. 4.31.

$$\tau = T \times 2^n \tag{4.31}$$

4.2.3 Acc/Dec Control Machining Errors

As mentioned above, because Acc/Dec control with ADCAI is applied separately for each axis, the path after Acc/Dec deviates from the programmed path.

In the case of a linear path on the XY plane, because the speed ratio between the X- and Y-axes before and after applying Acc/Dec control is constant, machining error due to Acc/Dec control does not occur. However, in the case of a circular path on the XY plane, the speed of the X- and Y-axes input to the Acc/Dec control circuit is actually a sine wave or cosine wave form. After passing through an Acc/Dec control filter (or circuit), the speed profiles of the X- and Y-axes are changed to a sine wave or cosine wave whose beginning and end are distorted.

Due to the characteristics of the Acc/Dec control filter, the summation of input pulses and the summation of output pulses are identical and, therefore, after passing through the Acc/Dec filter the axes can reach the commanded position. However, with the distortion of the speed at the beginning and end of acceleration and deceleration, the speed ratio between the X-axis and Y-axis is changed. The consequence is that there is a deviation between the programmed path and the path after Acc/Dec control.

In this section, the machining error for Linear Type Acc/Dec control, S-shape-type Acc/Dec control, and Exponential-type Acc/Dec control are discussed with respect to a circular path. For convenience of explanation, we assume that the feedrate for a circular path is F(mm/min) and the radius of the circular path is R(mm). Then the speed of each axis is given by Eq. 4.32.

$$\begin{aligned} V_x(t) &= -Rw\sin wt \\ V_y(t) &= Rw\cos(wt) \\ w &= \frac{F}{R} \end{aligned} \tag{4.32}$$

By applying Laplace transformation, Eq. 4.32 can be converted into Eq. 4.33.

$$\begin{aligned} V_x(s) &= -Rw\frac{w}{s^2+w^2} \\ V_y(s) &= Rw\frac{w}{s^2+w^2} \end{aligned} \tag{4.33}$$

By using Eq. 4.32 and Eq. 4.33, the machining error occurring with the three Acc/Dec control methods previously described are addressed during circular machining.

4.2.3.1 Machining Error with Linear Type Acc/Dec Control

In the Linear-type Acc/Dec control shown in Fig. 4.4, the impulse response for the Linear Type Acc/Dec control whose time constant is τ is given by Eq. 4.34.

$$H_l(s) = \frac{1}{\tau}\frac{1}{s}(1 - e^{-\tau s}) \tag{4.34}$$

where τ represents the Acc/Dec time. Therefore, the output of the Acc/Dec control, $W_x(s)$, of the X-axis is given by Eq. 4.35.

$$\begin{aligned} W_x(s) &= H_l(s)V_x(s) \\ &= -Rw\frac{1}{\tau}\frac{1}{s}\frac{1}{s^2+w^2}[1 - e^{-\tau s}] \\ &= \frac{K}{w}[\frac{1}{2} - \frac{s}{s^2+w^2}][1 - e^{-\tau s}] \\ \text{where } K &= -Rw\frac{1}{\tau} \end{aligned} \tag{4.35}$$

Using the inverse Laplace transform, Eq. 4.35 is converted to Eq. 4.36.

$$W_x(t) = \frac{2}{w\tau}\sin\frac{w\tau}{2}(-Rw)\sin w(t - \frac{\tau}{2}) \tag{4.36}$$

$W_x(t)$ in Eq. 4.36 denotes the speed of the X-axis and, by integrating $W_x(t)$, we can obtain the radius of the circular path after applying the Linear-type Acc/Dec control as Eq. 4.37a. From Eq. 4.37a, we can find the radius of the circular path after Acc/Dec time ($t > \tau$), R', which is expressed by Eq. 4.37b.

$$r = R\frac{2}{w\tau}\sin\frac{w\tau}{2}\cos w\left(t - \frac{\tau}{2}\right) \tag{4.37a}$$

$$R = \frac{2}{w\tau}\sin\frac{w\tau}{2}R \tag{4.37b}$$

As the machining error is the difference between the radius of the commanded path and the distorted path due to the Linear-type Acc/Dec control, the error is simplified as Eq. 4.38.

$$\begin{aligned} \Delta R &= R - R' = R(1 - \frac{2}{w\tau}\sin\frac{w\tau}{2}) \\ \Delta R &= R\{1 - \frac{2}{w\tau}(\frac{w\tau}{2} - \frac{w^3\tau^3}{8\cdot 3!}\ldots)\} \\ \text{therefore } \Delta R &\approx \frac{R}{24}w^2\tau^2 \approx \frac{1}{24}\tau^2\frac{F^2}{R} \\ \text{where, } \sin z &= \sum_{n=0}^{\infty}(-1)^n\frac{z^{2n+1}}{(2n+1)!} \text{ from the Taylor series.} \end{aligned} \tag{4.38}$$

4.2.3.2 Machining Error with S-shape-type Acc/Dec Control

In the S-shape Acc/Dec control shown in Fig. 4.4, the impulse response for the S-shape Acc/Dec control whose time constant is τ is given by Eq. 4.39.

$$H_s(s) = \frac{4}{\tau^2}\frac{1}{s^2}(1 - 2e^{-\frac{\tau}{2}s} + e^{-\tau s}) \tag{4.39}$$

where τ represents the Acc/Dec time. For the X-axis, the output of the Acc/Dec control, $W_x(s)$, is represented by Eq. 4.40.

$$\begin{aligned} W_x(s) &= H_s(S)V_x(s) \\ &= -Rw\frac{4}{\tau^2}\frac{w}{s^2+w^2}[1 - 2e^{-\frac{\tau}{2}s} + e^{-\tau s}] \\ &= \frac{K}{w}[\frac{1}{s^2} - \frac{1}{s^2+w^2}][1 - 2e^{-\frac{\tau}{2}s} + e^{-\tau s}] \\ \text{where } K &= -Rw\frac{4}{\tau^2} \end{aligned} \tag{4.40}$$

As with Linear-type Acc/Dec control Eq. 4.40 is converted to Eq. 4.41 using the inverse Laplace transform.

$$\begin{aligned} W_x(t) = \frac{K}{w}[\{t - \frac{1}{w}\sin wt\} - 2\{t - \frac{\tau}{2}\sin w(t - \frac{\tau}{2}\} \\ +\{t - \tau - \frac{1}{w}\sin w(t-\tau)\}] \end{aligned} \tag{4.41}$$

Equation 4.41 is simplified as Eq. 4.42.

$$\begin{aligned} W_x(t) &= \frac{K}{w^2}\{-\sin wt + 2\sin w(t - \frac{\tau}{2}) - \sin w(t-\tau)\} \\ &= \frac{K}{w^2}\{2\sin w(t - \frac{\tau}{2}) - 2\sin wt\cos^2\frac{w\tau}{2} + 2\cos wt\sin\frac{w\tau}{2}\cos\frac{w\tau}{2}\} \\ &= \frac{K}{w^2}\{2\sin w(t - \frac{\tau}{2}) - 2\sin w(t - \frac{\tau}{2})\cos\frac{w\tau}{2}\} \\ &= \frac{K}{w^2}2(1 - \cos\frac{w\tau}{2}\sin t - \frac{\tau}{2} \\ &= \frac{8}{w^2\tau^2}(1 - \cos\frac{w\tau}{2})(-Rw)\sin(t - \frac{w\tau}{2}) \end{aligned} \tag{4.42}$$

$W_x(t)$ in Eq. 4.42 denotes the speed of the X-axis and, by integrating $W_x(t)$, we can obtain the path radius after S-shape-type Acc/Dec control is applied. Equation 4.43a shows the result of the integration of $W_x(t)$. From Eq. 4.43a we know that after the Acc/Dec time, the radius of the path from S-shape-type Acc/Dec control, R', is represented by Eq. 4.43b.

$$r = R\frac{8}{w^2\tau^2}(1-\cos\frac{w\tau}{2})\sin w\left(t-\frac{\tau}{2}\right) \tag{4.43a}$$

$$R' = \frac{8}{w^2\tau^2}(1-\cos\frac{w\tau}{2})R \tag{4.43b}$$

As the machining error is the difference between the radius of the command path and the distorted path due to the S-shape-type Acc/Dec control, the error is simplified as Eq. 4.44.

$$\begin{aligned}
\Delta R = R - R' &= R[1-\frac{8}{w^2\tau^2}(1-\cos\frac{w\tau}{2})] \\
&= R[1-\frac{8}{w^2\tau^2}\{1-(1-\frac{w^2\tau^2}{4\cdot 2!}+\frac{w^4\tau^4}{2^4\cdot 4!}\ldots)\}] \\
&= R\{1-\frac{8}{w^2\tau^2}(\frac{w^2\tau^2}{4\cdot 2!}+\frac{w^4\tau^4}{2^4\cdot 4!}\ldots)\} \\
\text{so, } \Delta R &\approx \frac{1}{48}w^2\tau^2 \approx \frac{1}{48}\tau^2\frac{F^2}{R} \\
\text{where, } \cos z &= \sum_{n=0}^{\infty}(-1)^n\frac{z^{2n}}{(2n)!} \text{ from the Taylor series.}
\end{aligned} \tag{4.44}$$

4.2.3.3 Machining Error with Exponential Type Acc/Dec Control

In the Exponential-type Acc/Dec control shown in Fig. 4.4, the impulse response for the Exponential-type Acc/Dec control whose time constant is τ is given by Eq. 4.45.

$$H_e(s) = \frac{\frac{1}{\tau}}{s+\frac{1}{\tau}} \tag{4.45}$$

For the X-axis, the output of the Acc/Dec control, $W_x(s)$, is given by Eq. 4.46.

$$\begin{aligned}
W_x(s) &= H_e(s)V_x(s) \\
&= -Rw\frac{1}{\tau}\frac{1}{s+\frac{1}{\tau}}\frac{w}{w^2+s^2} \\
&= \frac{K}{s+a}\frac{w}{s^2+w^2} \\
\text{where } K &= -Rw\frac{1}{\tau}, \quad a = \frac{1}{\tau}
\end{aligned} \tag{4.46}$$

Equation 4.46 is converted to Eq. 4.47 using the inverse Laplace transform.

$$W_x(t) = (Ae^{-at} + B\cos wt + \frac{C}{w}\sin wt) \tag{4.47}$$
$$\text{where } A = \frac{wK}{w^2+a^2}, B = \frac{-wK}{w^2+a^2}, C = \frac{awK}{w^2+a^2}$$

$W_x(t)$ in Eq. 4.47 denotes the speed of the X-axis and by integrating $W_x(t)$, we obtain the path radius after Exponential-type Acc/Dec control has been applied. Equation 4.48a shows the result of the integration of $W_x(t)$. From Eq. 4.48a we know that after Acc/Dec time the radius of the path from Exponential-type Acc/Dec control, R', is given by Eq. 4.48b.

$$r = -\frac{A}{a}e^{-at} + \frac{1}{w}\sqrt{B^2+\frac{C^2}{w^2}}\sin(wt-\theta) \tag{4.48a}$$
$$\text{where } \theta = cos^{-1}\left(\sqrt{B^2+\frac{C^2}{w^2}}\right)$$

$$R' = \sqrt{B^2+\frac{C^2}{w^2}} = B\sqrt{1+\frac{a^2}{w^2}} = \frac{-wK}{w^2+a^2}\sqrt{1+\frac{a^2}{w^2}} \tag{4.48b}$$
$$= \frac{-K}{a}\frac{1}{\sqrt{1+\frac{w^2}{a^2}}} = -R\frac{1}{\sqrt{1+\frac{w^2}{a^2}}}$$

As the machining error is the difference between the radius of the commanded path and the distorted path due to Exponential-type Acc/Dec control, the error is simplified as Eq. 4.49.

$$\Delta R = R - R' = R\left(1 - \frac{1}{\sqrt{1=\frac{w^2}{a^2}}}\right)$$
$$\Delta R = R[1 - \{1 - \frac{1}{2}\left(\frac{w}{a}\right)^2 - \frac{3}{8}\left(\frac{w}{a}\right)^4 \ldots\}] \tag{4.49}$$
$$\text{so } \Delta R \approx \frac{R}{2}\left(\frac{w}{a}\right)^2 \approx \frac{1}{2}\tau^2\frac{F^2}{R}$$
$$\text{where, } \frac{1}{(1+z)^m} = \sum_{n=0}^{\infty}\left(-m//n\right)z^n$$
$$= 1 - mz + \frac{m(m+1)}{2!}z^2 - \frac{m(m+1)(m+2)}{3!}z^3 + \ldots$$
from a binomial series.

4.2.3.4 Machining Error Summary

The machining error due to the Acc/Dec control depends on the type of Acc/Dec control filter. The machining errors are summarized in Table 4.5 with respect to each type of Acc/Dec control filter.

According to Table 4.5 the machining error is proportional to the square of the feedrate and the Acc/Dec time. It is also in inverse proportion to the radius of the circular path. Therefore, from this, we know that the higher the feedrate the longer the Acc/Dec time, the shorter the radius of the circular path and the larger the machining error. We also know that the accuracy of the S-shape-type Acc/Dec control is better than that of the alternatives.

Table 4.5 Machining error due to Acc/Dec filter

Control type	Machining Error	Remarks
Linear	$\Delta R = \frac{F^2\tau^2}{24R}$	F: Feedrate
Exponential	$\Delta R = \frac{F^2\tau^2}{2R}$	τ: Time constant
S-shape	$\Delta R = \frac{F^2\tau^2}{48R}$	R: Radius of circle

4.2.4 Block Overlap in ADCAI

As mentioned in Chapter 2, the G-code system provides various instructions for controlling axes. Setting the block control mode is one of the G-code functions. For example, in the G-code system of the FANUC controller, there are two kinds of path control mode; exact stop mode (G61) and continuous mode (G64).

In exact stop mode, the machine follows the programmed path as exactly as possible, stopping at sharp corners of the path. Alternatively, in continuous mode, sharp corners of the path may be rounded slightly so that the feedrate may be kept up. Figure 4.11 shows the actual toolpath when exact stop mode is applied and Fig. 4.12 shows the actual toolpath when continuous mode is applied.

Exact stop mode generally results in reduction of machined surface quality due to the stoppage of axis movement and increases machining time due to acceleration and deceleration for all blocks.

In continuous mode, the tool begins the movement to the successive block before the tool reaches the end of the block. Unlike exact stop mode, this mode does not result in reduction of the surface quality and increase in machining time. In contin-

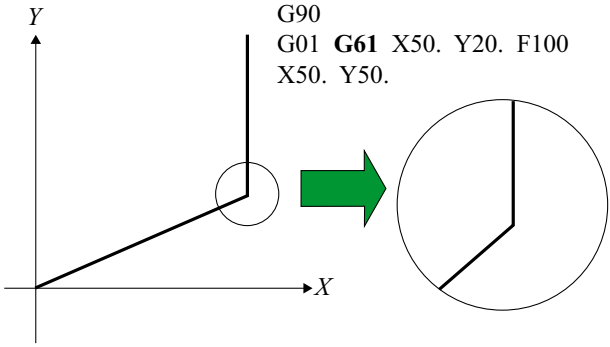


Fig. 4.11 Actual path in exact stop mode

uous mode, the toolpath does not pass through the programmed path as shown in Fig. 4.12. Therefore, machining error always occurs at sharp corners. The path near the corner depends on the Acc/Dec control type and, in general, the machining error is small enough so as not to reflect on machining accuracy.

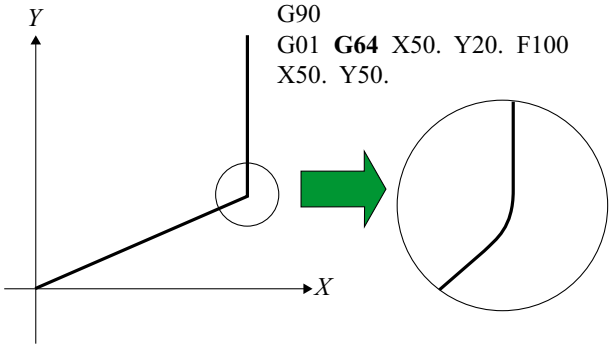


Fig. 4.12 Actual path in continuous mode

Figure 4.13 shows the result of X-axis interpolation and Acc/Dec control for two successive blocks. In Fig. 4.13, Block 1 and Block 2 are successive blocks and Fig. 4.13a and Fig. 4.13b show the interpolation result of Block 1 and Block 2 respectively. Figure 4.13c and Fig. 4.13d show the results of Linear Type Acc/Dec control for Block 1 and Block 2. If we combine the result of interpolation and Acc/Dec control for the two blocks with respect to time, we obtain the time–pulse graph shown in Fig. 4.14.

In continuous mode, the end result of Block 1 and the beginning of Block 2 are continuously connected. The connected interpolation pulse train is input continuously to the Acc/Dec controller and the Acc/Dec controller performs Acc/Dec control without considering blocks. Figure 4.14 shows the result of Linear-type Acc/Dec

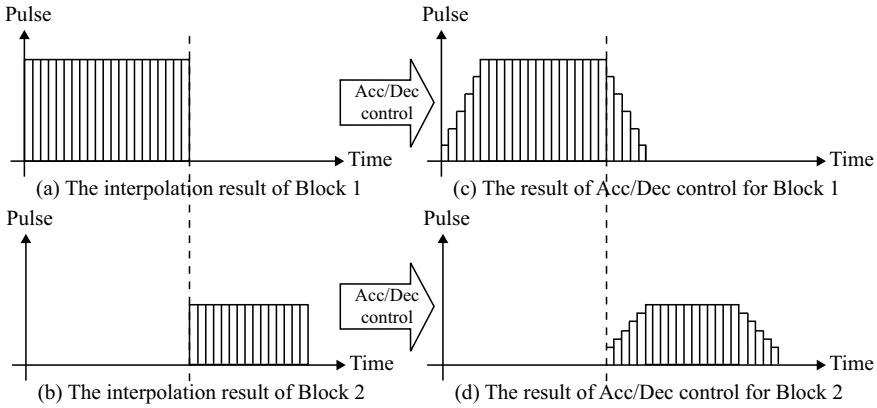


Fig. 4.13 *X*-axis interpolation and Acc/Dec control

control for two successive blocks. The time–pulse graph in Fig. 4.14 is identical with the summation of the two time–pulse graphs in Fig. 4.13b and Fig. 4.13d.

As shown in Fig. 4.14, in Continuous Mode, reduction of speed does not occur at the corner between two success blocks join. The speed is accelerated or decelerated considering the difference in the feedrate of the two blocks.

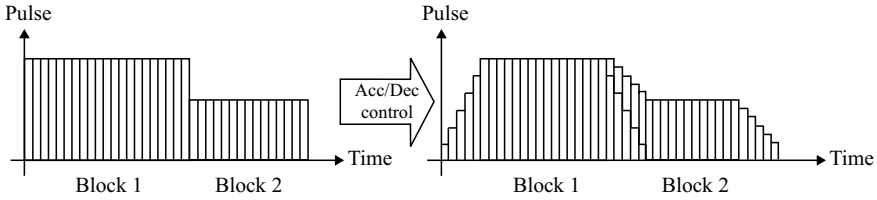


Fig. 4.14 Time–pulse graph for two successive blocks

4.3 Acc/Dec Control Before Interpolation

Unlike ADCAI-type NCK, ADCBI-type NCK generates the speed profile before executing rough interpolation. Also unlike ADCAI-type NCK, where Acc/Dec control is carried out separately for individual axes, ADCBI-type NCK carries out the Acc/Dec control for the programmed path itself. Therefore, theoretically, ADCBI-type NCK does not result in machining error.

As mentioned in Section 4.2.3, ADCAI generates machining error in proportion to the feedrate and this has become a serious problem considering that the machining speed of machine tools is getting faster. Therefore, ADCBI is essential to implement

the high-speed machining functions that have become a typical machine-tool function and consequently, the latest machine tools provide ADCAI as a basic function.

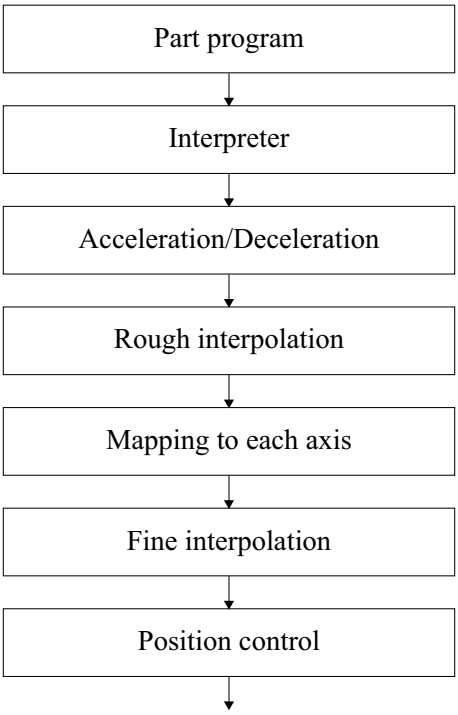


Fig. 4.15 ADCBI-type NCK flowchart

Figure 4.15 shows the flowchart for the overall procedure of the ADCBI-type NCK. Figure 4.16 shows the sequence of executing Acc/Dec control and rough interpolation and the output at each stage. The Acc/Dec Controller calculates the speed profile considering acceleration and deceleration. The rough interpolator then generates the interpolated points considering tool displacement and the remaining length of the programmed path for every iteration time instant based on the speed profile.

4.3.1 Speed-profile Generation

In ADCBI, the path length, the allowable acceleration and deceleration, the iteration time for rough interpolation, and the commanded feedrate are considered when generating a speed profile. For convenience, let us suppose that Acc/Dec control is applied to a linear path, the length of the linear path is L(mm), the allowable acceleration is A(mm/s^2), the allowable deceleration is D(mm/s^2), the iteration time

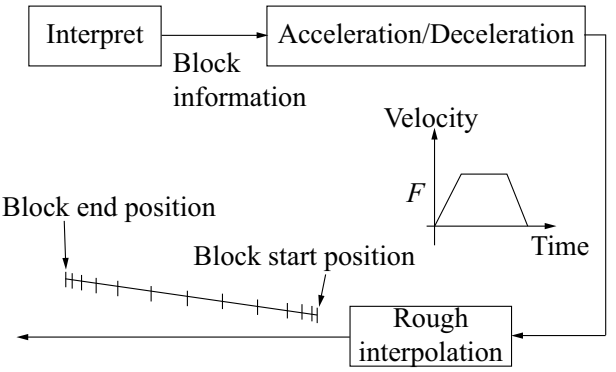


Fig. 4.16 Linear path Acc/Dec control

for rough interpolation is τ(s), and the commanded feedrate from a part program is F(mm/s^2).

In order to generate a speed profile, it is necessary to check if the linear path is a normal block or a short block. The normal block includes an acceleration zone, constant-speed zone, and deceleration zone, while the zone, or short block, does not include the constant-speed zone. Equation 4.50 is the condition that a normal block should satisfy. If Eq. 4.50 is not satisfied then the block is a short block.

$$\frac{F^2}{2A} + \frac{F^2}{2D} \geq L \tag{4.50}$$

In the case of a normal block, we can obtain a speed profile like that shown in Fig. 4.17a. In the case of a short block, we can obtain a speed profile like that shown in Fig. 4.17b. In the case of a short block, the length of the path is shorter than the length needed for the actual speed to reach the commanded feedrate F from zero speed and return back to zero speed. It is therefore impossible for the actual speed to reach the commanded feedrate, F.

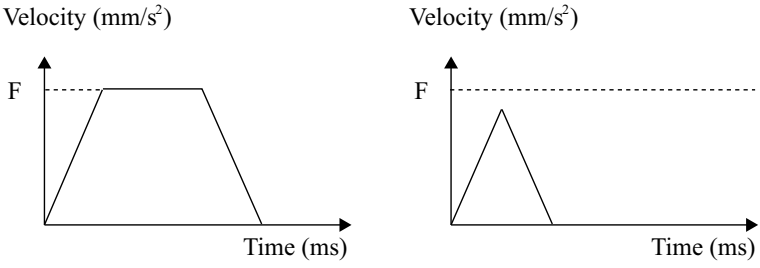


Fig. 4.17 Speed profiles

After checking whether the path is a normal block or a short block using Eq. 4.50, the speed profile is generated according to the path type. In the case of a normal block, the acceleration time T_A that is spent to reach the commanded feedrate F from 0(mm/sec), is computed by Eq. 4.51 and the deceleration time T_D, which is spent to reach 0(mm/s) from the commanded feedrate F is computed using Eq. 4.52. The constant speed time T_C is calculated by dividing the length of the path after subtracting the length needed for acceleration and deceleration by the commanded feedrate, as given by Eq. 4.53.

$$T_A = \frac{F}{A} \tag{4.51}$$

$$T_D = \frac{F}{D} \tag{4.52}$$

$$T_C = \frac{L - \frac{F^2}{2A} - \frac{F^2}{2D}}{F} \tag{4.53}$$

In the case of a short block, the length of the block is obtained by integrating the speed profile shown in Fig. 4.17b with respect to time. If the maximum reachable speed for the short block is F', acceleration time T_A, deceleration time T_D, and F' are calculated using Eq. 4.54.

$$\begin{aligned} T_A &= \frac{F'}{A} \\ T_D &= \frac{F'}{D} \\ L &= \frac{F' \times (T_A + T_D)}{2} \end{aligned} \tag{4.54}$$

From the above equations, it is possible to generate a speed profile for both normal blocks and short blocks. Also, based on the generated speed profile, the interpolation for a linear path can be carried out. In the ADCBI-type NCK, the rough interpolator calculates the interpolated point through which the tool should go for every constant iteration time for interpolation, τ. In the acceleration range, the length that the tool should move every iteration time for interpolation can be calculated using Eq. 4.55.

$$\begin{aligned} V_{i+1} &= V_i + \tau \cdot A, (i = 0, 1, 2, \ldots, N_A) \\ L_i &= \frac{V_{i+1}^2 - V_i^2}{2A} \end{aligned} \tag{4.55}$$

where, V_i is the velocity of the ith interval and $V_0 = 0$

L_i is the displacement for the ith sampling time.

$N_A = \frac{T_A}{\tau}$.

In the constant speed range the commanded feedrate is F and the tool moves $\tau \times F$ every iteration time for interpolation. In the deceleration interval, the length through which the tool moves every iteration time for interpolation can be calculated using Eq. 4.56.

$$V_{i+1} = V_i - \tau \cdot D, (i = 0, 1, 2, \ldots, N_D) \tag{4.56}$$
$$L_i = \frac{V_i^2 - V_{i+1}^2}{2D}$$

where, V_i is the velocity of the ith interval and $V_0 = F$

L_i is the displacement for the ith sampling time.

$N_D = \frac{T_D}{\tau}$.

It is possible to calculate the interpolated point by projecting the displacement through which the tool moves in every iteration time for interpolation onto the programmed path.

4.3.2 Block Overlap Control

Hardly ever is only one linear block or one circular block used for actual machining. In general, because an NC program consists of multiple linear blocks and circular blocks, it is true that direct usage of the above-mentioned equations for generating speed profile and interpolating is impossible. In ADCAI, interpolation and Acc/Dec control are applied to the individual block and it is not necessary to consider the connection of blocks. However, in ADCBI, because the speed at the beginning and the end of a block should be considered when generating a speed profile, the previous and the successive blocks should be considered when generating a speed profile and interpolating.

In the next sections, all possible cases for connection relationships that can occur between two successive blocks in actual machining will be addressed. The equations for generating a speed profile for each case will be described.

4.3.2.1 Classification of Continuous Blocks

In Section 4.3.1, we defined the block with constant speed interval as a normal block and the block without constant speed interval as a short block. From the way in which two blocks are connected it is possible to classify pairs of blocks into twelve types depending on the type of block (*e.g.* normal block and short block) and the difference of commanded feedrate between the two blocks. However, in the case when a short block and a normal block are successive, since the speed profile can be generated with an identical equation regardless of the commanded feedrate of the two blocks,

a method to calculate the speed profile when the commanded feedrate of the two blocks is identical will be described. Therefore, the way in which two blocks are connected can be classified into eight types, as shown in Fig. 4.18. For convenience, it is supposed that the direction of two successive blocks is identical.

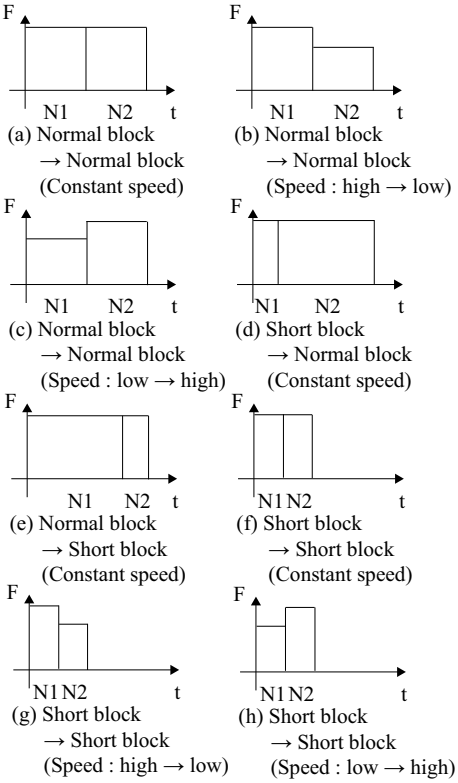


Fig. 4.18 Speed profiles for identical blocks

4.3.2.2 Normal Block/Normal Block, Identical Speed

As shown in Fig. 4.18a, if two blocks with an identical feedrate F are successive, it is possible to generate the successive speed profile by the methods mentioned in Section 4.3.1.

Because in Block N1, deceleration is not necessary, the acceleration time T_{A1} is computed by Eq. 4.51 and the constant-speed time T_{C1} is computed by Eq. 4.57.

$$T_{C1} = \frac{L_1 - \frac{F^2}{2A}}{F} \tag{4.57}$$

where, L_1 is the displacement of block $N1$

In Block $N2$, because at the beginning of the block the tool is moving with feedrate F, acceleration is not required and only deceleration is necessary. The deceleration time T_{D2} is computed by Eq. 4.52 and the constant-speed time T_{C2} is computed by Eq. 4.58.

$$T_{C2} = \frac{L_2 - \frac{F^2}{2D}}{F} \tag{4.58}$$

where, L_2 is the displacement of block $N2$

When two successive blocks have the same feedrate, the speed profile for the acceleration interval can be obtained based on Eq. 4.55. The speed profile for the deceleration interval can be obtained by Eq. 4.56. Based on the above-mentioned equations, it is possible to generate the speed profile for two successive normal blocks with the same feedrate as in Fig. 4.19.

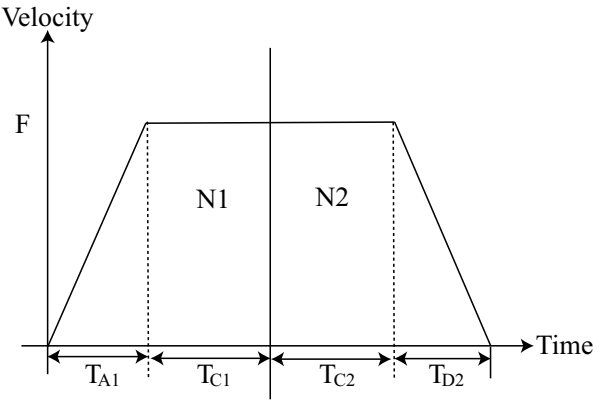


Fig. 4.19 Speed profiles for identical blocks

4.3.2.3 Normal Block (High Speed)/Normal Block (Low Speed)

In the case when two normal blocks with different feedrates are successive as shown in Fig. 4.18b, the lower of the two blocks' speeds is defined as the speed at the corner. For example, if the commanded feedrates of Block $N1$ and $N2$ are F_1 and F_2, respectively, and F_1 is higher than F_2, the speed at the corner is defined as F_2. This is done in order to avoid abnormal machining status such as tool breakage due

to the high speed. In Block $N1$, acceleration time T_{A1} is computed by Eq. 4.59 and deceleration time T_{D1} is computed by Eq. 4.60.

$$T_{A1} = \frac{F_1}{A} \tag{4.59}$$

$$T_{D1} = \frac{F_1 - F_2}{D} \tag{4.60}$$

In Block $N1$, the speed profile for the acceleration interval is obtained by Eq. 4.55 and the speed profile for the deceleration interval is obtained by Eq. 4.56 where the speed at the beginning of deceleration is F_1, and the speed at the end of deceleration is F_2. The constant speed time of Block $N1$ is calculated by Eq. 4.61.

$$T_{C1} = \frac{L_1 - \frac{F_1^2}{2A} - \frac{F_1^2 - F_2^2}{2D}}{F_1} \tag{4.61}$$

In Block $N2$, the acceleration at the beginning of the block is not necessary because the speed at the end of Block $N1$ is decelerated to the commanded feedrate F_2 of Block $N2$. The deceleration time T_{D2} is computed by Eq. 4.62 and the speed profile for the deceleration interval can be obtained by Eq. 4.56 where the speed at the beginning of deceleration is F_2 and the speed at the end of deceleration is 0. The constant speed time of Block $N2$ is calculated by Eq. 4.63.

$$T_{D2} = \frac{F_2}{D} \tag{4.62}$$

$$T_{C2} = \frac{L_2 - \frac{F_2^2}{2D}}{F_2} \tag{4.63}$$

Based on the above-mentioned equations, it is possible to generate the speed profile shown in Fig. 4.20.

4.3.2.4 Normal Block (Low Speed)/Normal Block (High Speed)

Figure 4.18c shows the case where two normal blocks with different feedrate are successive and the speed of the first block is smaller than that of the second block. In this case, the smaller speed between the two block speeds is defined as the speed at the corner as shown in Fig. 4.18b.

If the commanded feedrate of Block $N1$ is F_1 and the commanded feedrate of Block $N2$ is F_2, the speed at the corner is defined as F_1. In Block $N1$, acceleration time T_{A1} is computed by Eq. 4.64, but it is not necessary to calculate deceleration because the speed at the end position is F_1 and so it is not necessary to decelerate. The constant-speed time T_{C1} is computed by Eq. 4.65.

$$T_{A1} = \frac{F_1}{A} \tag{4.64}$$

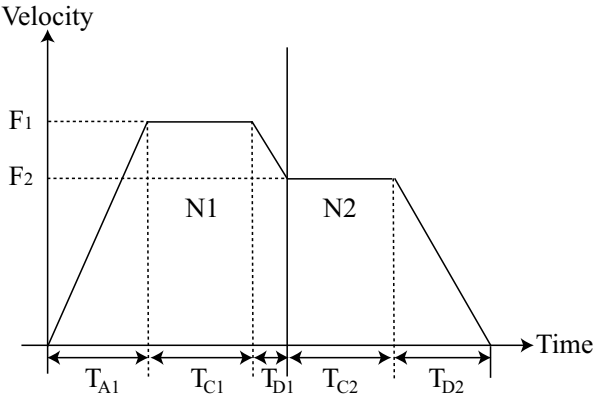


Fig. 4.20 Speed profiles for normal blocks with F_1 larger than F_2

$$T_{C1} = \frac{L_1 - \frac{F_1^2}{2A}}{F} \tag{4.65}$$

In Block $N1$, the speed profile of the acceleration interval can be obtained by Eq. 4.55 and the speed at constant speed interval is held at the commanded feedrate F_1.

In Block $N2$, because the feedrate is lower than the commanded feedrate of Block $N2$, F_2, at the end of Block $N1$, the speed at the beginning of the Block $N2$ is not changed and only deceleration is required at the end of the block. The deceleration time T_{D2} is calculated by Eq. 4.66 and the constant speed time is calculated by Eq. 4.67. The speed profile of the deceleration interval can be obtained by Eq. 4.56 where the speed at the beginning of deceleration is F_2 and the speed at the end of deceleration is 0.

$$T_{D2} = \frac{F_2}{D} \tag{4.66}$$

$$T_{C2} = \frac{L_2 - \frac{F_2^2}{2D}}{F_2} \tag{4.67}$$

Based on the above-mentioned equations, it is possible to generate the speed profile shown in Fig. 4.21.

4.3.2.5 Short Block/Normal Block with Identical Speed

Figure 4.18d shows the case where a short block precedes a normal block and the feedrate of the two blocks is identical. In order to generate a speed profile, firstly the speed at the connection point of the two blocks should be calculated. Unlike the

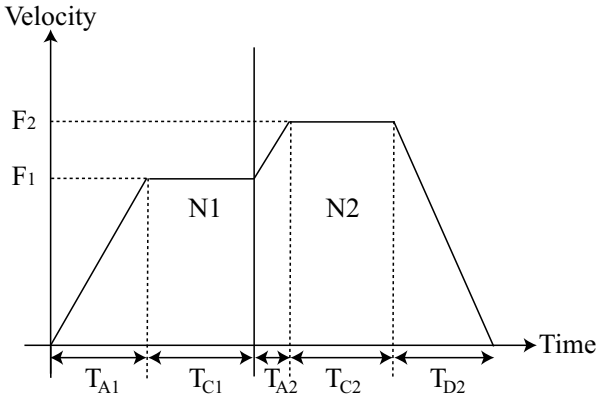


Fig. 4.21 Speed profiles for normal blocks with F_1 lower than F_2

case where two normal blocks are connected, because it is impossible to reach at the commanded feedrate on a short block, it is first necessary to consider the maximum reachable speed on the short block. Equation 4.64 is used for calculating this.

$$F' = \sqrt{2AL_1} \tag{4.68}$$

The speed F' from Eq. 4.68 is defined as the corner speed and the speed of the beginning of Block $N2$. The time spent to reach F' from 0 in a short block, T_{A1}, is computed by Eq. 4.69 and the time spent to reach the commanded feedrate of Block $N2$ from F', T_{A2}, is computed by Eq. 4.70.

Further, the speed profile of the acceleration interval in Block $N1$ can be obtained by Eq. 4.69 and Eq. 4.55 and the speed profile of acceleration interval in Block $N2$ can be obtained by Eq. 4.70 and Eq. 4.55 where the speed at the beginning of acceleration is F' and the speed at the end of acceleration is F_2.

$$T_{A1} = \frac{F'}{A} \tag{4.69}$$

$$T_{A2} = \frac{F_2 - F'}{A} \tag{4.70}$$

The deceleration time in Block $N2$, T_{D2}, is computed by Eq. 4.62 and the constant speed time is calculated by Eq. 4.71. The speed profile of the deceleration interval in Block $N2$ can be obtained by Eq. 4.56.

$$T_{C2} = \frac{L_2 - \frac{F_2^2 - F'^2}{2A} - \frac{F_2^2}{2D}}{F_2} \tag{4.71}$$

Based on the above-mentioned equations, it is possible to generate the speed profile shown in Fig. 4.22.

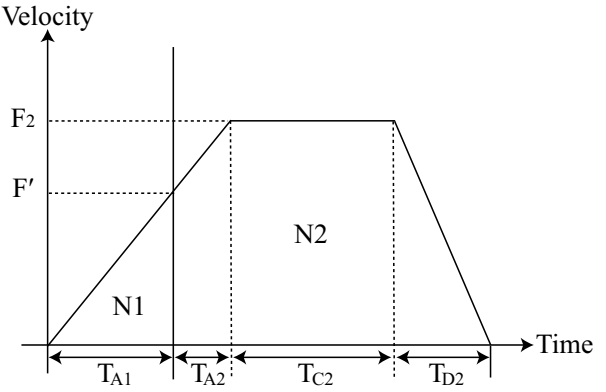


Fig. 4.22 Speed profiles for short block/normal block with identical feedrates

4.3.2.6 Short Block/Normal Block with Different Speed

In the case where a short block precedes a normal block are continued and the commanded feedrate of the two blocks are different from each other. It is possible to generate a speed profile by the same method as mentioned in Section 4.3.2.5. The reason is that it is impossible to reach the commanded feedrate in a short block and the corner speed is decided based only on the length of the short block, $L1$.

4.3.2.7 Normal Block/Short Block with Identical Speed

Figure 4.18e shows the case where a normal block precedes a short block and the feedrate of the two blocks is identical. As mentioned in Section 4.3.2.5, the speed at the connection point of the two blocks should be calculated based on the length of the short block in order to generate a speed profile. In this case, because a short block is executed after a normal block, the start speed of the short block that makes the speed at the end of the block zero should be calculated. Equation 4.72 is used for calculating the start speed of Block $N2$, F'.

$$F' = \sqrt{2DL_2} \tag{4.72}$$

The speed F' from Eq. 4.72 is defined as the corner speed and the speed at the end of Block N1. the acceleration time of Block $N1$, T_{A1}, is computed by Eq. 4.73 and the deceleration time of Block $B1$, T_{D1}, is computed by Eq. 4.74. Further, the constant speed time, T_{C1}, is calculated by Eq. 4.75.

$$T_{A1} = \frac{F_1}{A} \tag{4.73}$$

$$T_{D1} = \frac{F_1 - F'}{D} \tag{4.74}$$

$$T_{C1} = \frac{L_1 - \frac{F_1^2}{2A} - \frac{F_1^2 - F'^2}{2D}}{F_1} \tag{4.75}$$

The speed profile of the acceleration interval of Block N1 can be obtained by Eq. 4.73 and Eq. 4.55. The speed profile of the deceleration interval of Block N2 can be obtained by Eq. 4.74 and Eq. 4.56, where the initial speed at the deceleration interval, V_0, is F_1 and the end speed of the deceleration interval is F'.

$$T_{D2} = \frac{F'}{D} \tag{4.76}$$

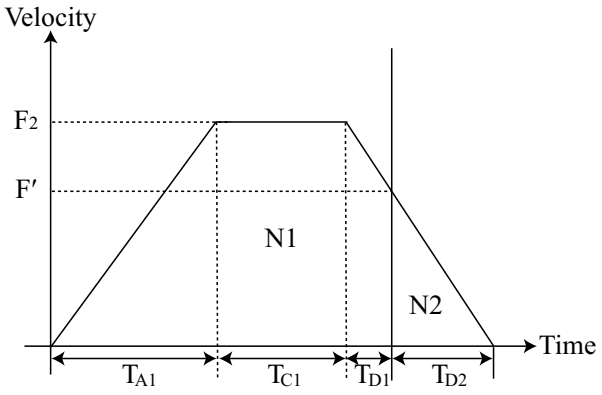


Fig. 4.23 Speed profile for Normal block/Short block with F_1 larger than F_2

There is only a deceleration interval in Block $N2$. The deceleration time, T_{D2}, can be obtained by Eq. 4.76. Figure 4.23 shows the speed profile generated from the above-mentioned equations.

4.3.2.8 Normal Block/Short Block with Different Speed

When a normal block precedes a short block, the commanded feedrate of two blocks can be different from each other. In this case, it is possible to generate a speed profile by a method similar to that of a normal block and short block with the same commanded feedrate, described in Section 4.3.2.7. This is because the corner speed is decided based on the length of the short block, $L2$, regardless of its commanded speed.

Therefore, for the case that is described in this section, the corner speed F' is computed by Eq. 4.72 and the speed profile is generated in the same way as the method mentioned in Section 4.3.2.7.

4.3.2.9 Short Block/Short Block with Identical Speed

Figure 4.18f shows the case where two short blocks with identical feedrate are connected. In this case, in order to generate a speed profile, it is first necessary to calculate the maximum feasible speed at the corner, F'. The end speed of Block $N1$, F_1', is computed by Eq. 4.77 and the start speed of Block $N2$, F_2', is computed by Eq. 4.78.

$$F_1' = \sqrt{2AL_1} \tag{4.77}$$

$$F_2' = \sqrt{2DL_2} \tag{4.78}$$

The smaller of the two speeds F_1' and F_2' is selected as the corner speed F' and it is possible to calculate the maximum speed, F_{max}, based on F'. If F' is the same as F_2', F_{max} is calculated by Eq. 4.79 and in the case when F' is F_1', F_{max} is calculated by Eq. 4.80.

$$\frac{F_{max}^2}{2A} + \frac{F_{max}^2 - F'^2}{2D} = L_1 \tag{4.79}$$

$$\frac{F_{max}^2 - F'^2}{2A} + \frac{F_{max}^2}{2D} = L_2 \tag{4.80}$$

If F' is F_2', the acceleration time of Block $N1$, T_{A1}, is calculated by Eq. 4.81 and the deceleration time, T_{D1}, is calculated by Eq. 4.82. In addition, the speed profile can be obtained by Eq. 4.55 and Eq. 4.56 where the initial speed of the deceleration interval, V_0, is F_{max} and the end speed of the deceleration interval is F'. Also, the deceleration time of Block $N2$, T_{D2}, is calculated by Eq. 4.76 and the speed profile of Block $N2$ can be obtained by Eq. 4.56 where the initial speed of deceleration, V_0, is F' and the end speed of deceleration is zero.

$$T_{A1} = \frac{F_{max}}{A} \tag{4.81}$$

$$T_{D1} = \frac{F_{max} - F'}{D} \tag{4.82}$$

Figure 4.24 shows the speed profile generated from the above-mentioned equations in the case that F' is F_2'. In the case that F' is F_1', it is possible to generate a speed profile in a similar way.

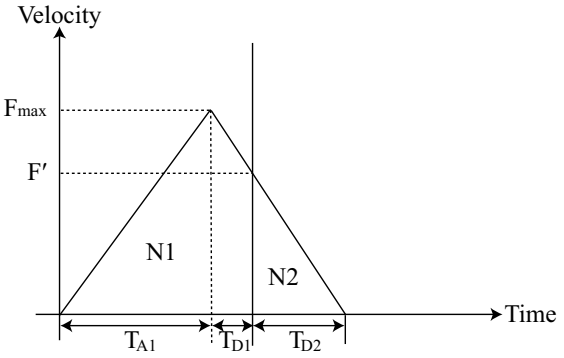


Fig. 4.24 Speed profile for two short blocks with F_1' larger than F_2'

4.3.2.10 Short Block (High Speed)/Short Block (Low Speed)

Figure 4.18g shows the case where two short blocks with different feedrates are connected. As mentioned in Section 4.3.2.6, the corner speed of two short blocks is decided by the length of the short blocks regardless of the commanded feedrate of the blocks. Therefore, the speed profile for the case mentioned in this section can be identically obtained by the method of the case in Section 4.3.2.9.

4.3.2.11 Short Block (Low Speed)/Short Block (High Speed)

Figure 4.18h shows the case where two short blocks are connected and the speed of the first block is smaller than that of the second block. Although the speed of the two blocks is different, the method to obtain the speed profile is identical with that of the case mentioned in Section 4.3.2.9 because the corner speed of the two short blocks is decided by the length of the short blocks regardless of the commanded feedrate of the blocks.

4.3.2.12 Overlap Between a Linear and a Circular Profile

During circular-path machining, the speed of each axis is continually changing. Therefore it is necessary to reduce the speed (feedrate) compared with the linear path. The change of the axis speed results in mechanical shock and, especially at the transition point from a circular path to a linear path, large mechanical shock occurs. The mechanical shock is proportional to the acceleration. The acceleration is proportional to the square of the feedrate and is inversely proportional to the radius of the circular path. Therefore, it is necessary to restrict the maximum allowable acceleration for a circular path. The allowable speed for a circular path is obtained as below.

During circular movement, the speed of each axis is computed by Eq. 4.83 and the acceleration of each axis is computed by Eq. 4.84.

$$V_x = F\cos\omega t \quad V_y = F\sin\omega t \tag{4.83}$$
$$\text{where, } \omega = \frac{F}{R}$$

$$A_x = -F\omega\sin\omega t \quad A_y = F\omega\cos\omega t \tag{4.84}$$

If the radius of the circular path is R_o, the allowable speed of each axis can be computed by Eq. 4.85. By using Eq. 4.84, the allowable feedrate for the circular path with radius R_c can be computed by Eq. 4.86.

$$F_x = \sqrt{A_x R_o} \quad (A_x, A_y : Jakam jaka) \tag{4.85}$$
$$F_y = \sqrt{A_y R_o} \quad F_o = \sqrt{F_x^2 + F_y^2}$$

$$\frac{F_1^2}{R_c} = \frac{F_o^2}{R_o}, \quad F_1 = F_o\sqrt{\frac{R_c}{R_o}} \tag{4.86}$$

The following is an example of an NC part program that sequentially commands machining of a linear path-circular path-linear path. Figure 4.25 shows the paths of the example NC part program. Figure 4.26 shows the allowable feedrate with respect to the radius of a circular path. Also, Fig. 4.27 shows the actual feedrate in the case where the commanded feedrate of the circular path is smaller than the allowable feedrate.

```
N1 G91 G01 X100. F10000;
N2 G02 X50. Y-50. R50;
N3 G01 Y-100;
N4 M06
```

4.3.3 Corner Speed of Two Blocks Connected by an Acute Angle

In Section 4.3.2, it is supposed that the direction of two successive blocks is the same. However, in practice, the direction of two successive blocks can be different from each other. The different direction of blocks results in acceleration or deceleration for each axis.

Figure 4.28 shows two successive blocks with different directions. The feedrate of the first block is F_1, the feedrate of the second block is F_2 and the angle between two successive blocks is θ. The acceleration at the corner is computed by Eq. 4.87.

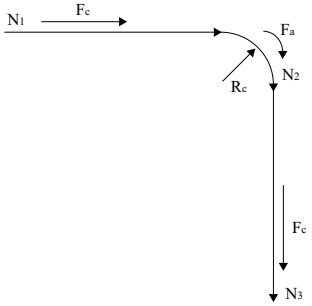


Fig. 4.25 Linear path – circular path – linear path

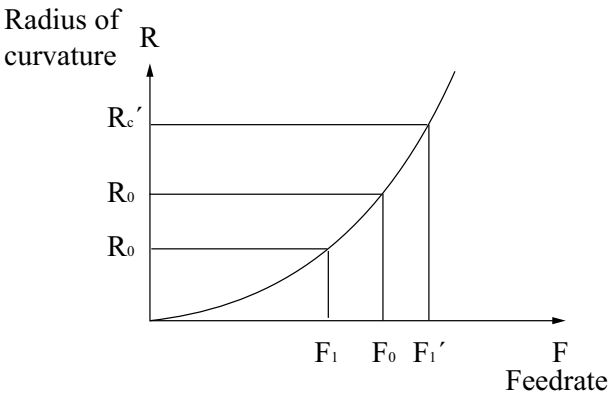


Fig. 4.26 Allowable feedrate with respect to radius of circular path

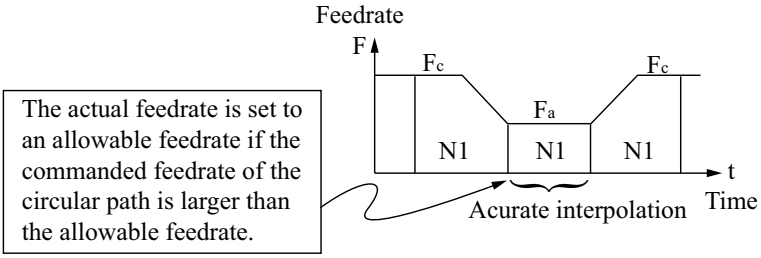


Fig. 4.27 Actual feedrate where commanded feedrate of circular path is smaller than allowable feedrate

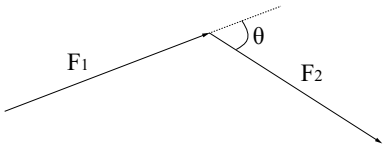


Fig. 4.28 Two successive blocks with different directions

$$A_C = \frac{F_1 - F_2 \cos\theta}{T_{pos}} \tag{4.87}$$

where, T_{pos} is the sampling time for position control

If the acceleration calculated from Eq. 4.87 is greater than the maximum allowable acceleration of the machine tool, a mechanical shock or vibration can occur, thereby a large machining error is produced. Therefore, a corner speed F_c, which does not exceed the maximum allowable acceleration, should be calculated using Eq. 4.88.

$$F_c = \frac{AT_{pos}}{1 - \cos\theta} \tag{4.88}$$

where, A is the maximum allowable acceleration

Based on the commanded feedrate, the length of blocks, the allowable acceleration and the corner speed from Eq. 4.88, a speed profile can be generated as described in Section 4.3.2. However, if the acceleration due to the radius of a circular path is greater than the allowable acceleration, the acceleration that is applied to generate the speed profile should be modified to be the acceleration calculated from Eq. 4.87.

4.3.4 Corner Speed Considering Speed Difference of Each Axis

As the corner speed control method mentioned in Section 4.3.3 is based on the allowable joint acceleration of the machine tool, it is mainly used for robot control. In general, machine tools have individual servo motors for each axis and each axis has an individual allowable acceleration value based on the performance of its servo motor. In this case, another method for deciding the corner speed is used instead of the method mentioned in Section 4.3.3.

For convenience of explanation, we define the first block as $N1$ and the next block as $N2$. We define that the start point and the end point of $N1$ are (X_{S1}, Y_{S1}, Z_{S1}) and (X_{E1}, Y_{E1}, Z_{E1}), respectively and the start point and the end point of $N2$ are (X_{S2}, Y_{S2}, Z_{S2}) and (X_{E2}, Y_{E2}, Z_{E2}), respectively. In this case, the speed of blocks $N1$ and $N2$ in the direction of each axis are given by Eq. 4.89.

$$V_{X1} = F_1 \cdot \frac{X_{E1} - X_{S1}}{L_1}, V_{Y1} = F_1 \cdot \frac{Y_{E1} - Y_{S1}}{L_1}, V_{Z1} = F_1 \cdot \frac{Z_{E1} - Z_{S1}}{L_1},$$
$$V_{X2} = F_2 \cdot \frac{X_{E2} - X_{S2}}{L_2}, V_{Y2} = F_2 \cdot \frac{Y_{E2} - Y_{S2}}{L_2}, V_{Z2} = F_2 \cdot \frac{Z_{E2} - Z_{S2}}{L_2}, \tag{4.89}$$

where V_{Ai} is the A-axis component of velocity of block Ni,
L_i is the length of block Ni

The difference in speed along the directions of each axis $(\Delta V_X, \Delta V_Y, \Delta V_Z)$ is given by Eq. 4.90 at the corner block where $N1$ and $N2$ are joined.

$$\Delta V_X = (V_{X2} - V_{X1}), \Delta V_Y = (V_{Y2} - V_{Y1}), \Delta V_Z = (V_{Z2} - V_{Z1}) \tag{4.90}$$

When we define the maximum allowable change of speed along each axis as ΔV_{mx}, ΔV_{my}, ΔV_{mz}, respectively, the smallest of the speed change ratios (Q) is given by Eq. 4.91.

$$Q = \min \left\{ \frac{\Delta V_{mx}}{\Delta V_x}, \frac{\Delta V_{my}}{\Delta V_y}, \frac{\Delta V_{mz}}{\Delta V_z} \right\} \tag{4.91}$$

Here, if Q is greater than 1, this means that the change of speed is smaller than the maximum allowable value. If Q is less than 1 it means that there is more than one axis whose speed change is greater than the maximum allowable value. Accordingly, if Q is greater than 1, it is necessary to decrease the feedrate of blocks. The end speed of $N1$, F_{E1}, and the start speed of $N2$, F_{S2} are calculated using Eq. 4.92.

$$F_{E1} = Q \cdot F_1, F_{S2} = Q \cdot F_2 \tag{4.92}$$

The corner speed from Eq. 4.92 is used to generate a speed profile. As mentioned in Section 4.3.2, the corner speed is calculated based on the commanded feedrate and the length of blocks. Next, the speed change of each axis at the corner is calculated by applying the computed corner speed to Eq. 4.90. Finally, the speed change ratio is computed using Eq. 4.91 and, if Q is smaller than 1, a new corner speed has to be calculated. It is possible that the end speed of $N1$ can be different from the start speed of $N2$. Although discontinuity of speed occurs, this does not result in any problem because the speed change is enough small for a servo motor to follow the changed speed.

4.4 Look Ahead

Machining speed and machining accuracy are the key factors for the performance of CNC machine tools. Machining accuracy depends on the ability to follow the trajectory of the controller. As mentioned in Chapter 3, the accuracy of the machining

trajectory is inversely proportional to the feedrate and sudden changes of feedrate result in reduction of accuracy of the CNC equipment.

In the ADCBI type of NCK, the accuracy of machining is very high (theoretically the error is zero) and sudden change of feedrate is a major factor of machining error. Therefore, in the ADCBI-type NCK, in order to minimize the machining error, it is necessary to smooth down change of feedrate and limit the axis speed to an allowable value. To smooth down the change of feedrate, axes should always be accelerated by an adequate acceleration value. Consequently, to maximize the performance of CNC systems it is necessary to maximize the acceleration ability of the CNC system.

As mentioned in the previous section, in the case when two short blocks are connected, the length of two blocks is too short to reach the commanded feedrate and the resulting speed profile shows a special shape similar to a saw tooth. The reduction of feedrate results in an increase of the machining time. To overcome this problem a method of minimizing the reduction of feedrate was introduced by considering the commanded feedrate and the length of the successive blocks.

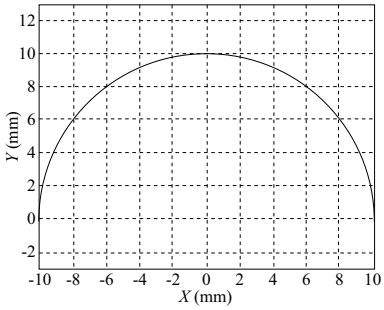


Fig. 4.29 Circular trajectory

For example, assume that the half-circle shown in Fig. 4.29 consists of 15 line segments, the radius of the half-circle is 10 mm, the commanded feedrate is 400 mm/min, and the allowable acceleration is 9600 mm/min. If the method described in Section 4.3 is used, the speed profile will be generated as shown in Fig. 4.30. The reason is that the length of the line segment is 2.094 mm and the length is too short to reach the commanded feedrate, 400mm/min. As shown in Fig. 4.30, the maximum reachable feedrate is 141.78mm/min and acceleration and deceleration were repeated.

To minimize the reduction of feedrate and decrease in machining time for short blocks, a Look Ahead algorithm has been widely used. The Look Ahead algorithm enables minimization of the decrease of feedrate by calculating the maximum allowable feedrate and the end feedrate for a current block investigating not only the current block but also successive blocks.

The latest FANUC controller is able to calculate the end speed of a current block by pre-interpreting about 1000 blocks. Therefore, it is not necessary to make the end

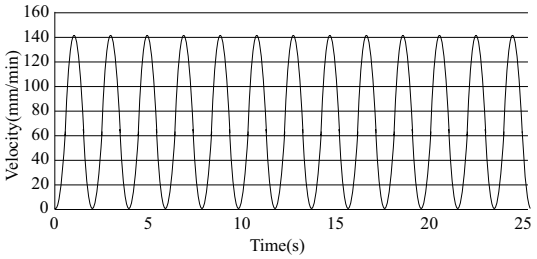


Fig. 4.30 Speed profile for circular profile

speed of the current block zero and it is possible to control the speed of successive blocks depending on the commanded feedrate and the length.

4.4.1 Look-Ahead Algorithm

A Look Ahead algorithm calculates the start speed and the end speed of each block based on the remaining length of the successive blocks and the maximum allowable acceleration.

4.4.1.1 Look Ahead with Respect to Length

The start speed of the current block should be a speed that enables deceleration to the end speed of the current block and is computed by Eq. 4.93

$$V_0 = \sqrt{V_f^2 + 2 \cdot A \cdot L} \tag{4.93}$$

where, V_f denotes a feasible entry feedrate to the next block, F is the actual feedrate of the current block, V_0 is the feasible entry feedrate of the current block, A is the maximum allowable acceleration of the machine tool, and L is the length of the current block.

$$V_f = \begin{cases} V, V < F \\ F, V > F \end{cases} \tag{4.94}$$

Since the entry feedrate to the next block from Eq. 4.93 cannot exceed the commanded feedrate, the feasible entry feedrate can be represented by Eq. 4.94.

For example, if the entry feedrate of the current block, V_0, is larger than the commanded feedrate F, Fig. 4.31a the feasible entry feedrate of the current block comes to be F and the end feedrate comes to be V_f. Also, the speed profile of the current block can be represented as shown in Fig. 4.31b.

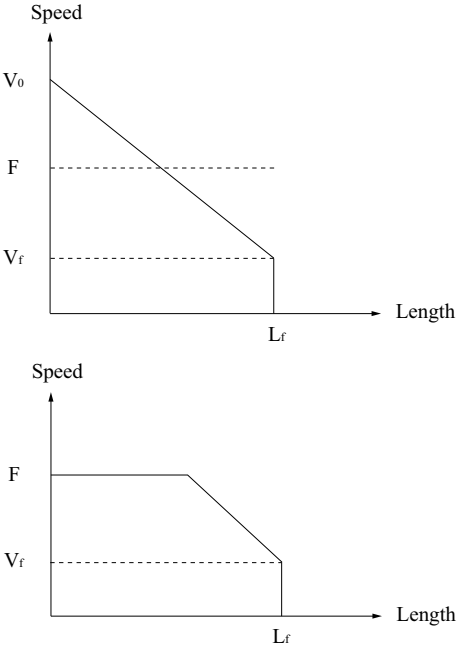


Fig. 4.31 Start conditions and speed profile with Look Ahead

Consequently, with sequential computing, the end speed and feasible entry speed from the look-ahead (pre-interpreted) block to the current block, and finally the start speed and the end speed of the current block are calculated. Here it is assumed that the end speed of the last block among the look-ahead blocks is zero.

4.4.1.2 Speed at a Corner

There are two methods for determining the speed between two blocks in a Look Ahead algorithm. The first is the method based on the angle between two blocks, as described in Section 4.3.3, and the second is the method based on the speed difference ratio of axes described in Section 4.3.4.

4.4.1.3 Look Ahead considering Length and Corner

As mentioned in the previous section, the corner speed between two successive blocks is decided by selecting the smaller value among the corner speeds based on maximum allowable acceleration and the feasible entry speed from the Look Ahead algorithm described in Section 4.4.1.1. Figure 4.33 shows the flow chart for deter-

mining the end speed of the current block considering the length of the block and the corner speed between two successive blocks.

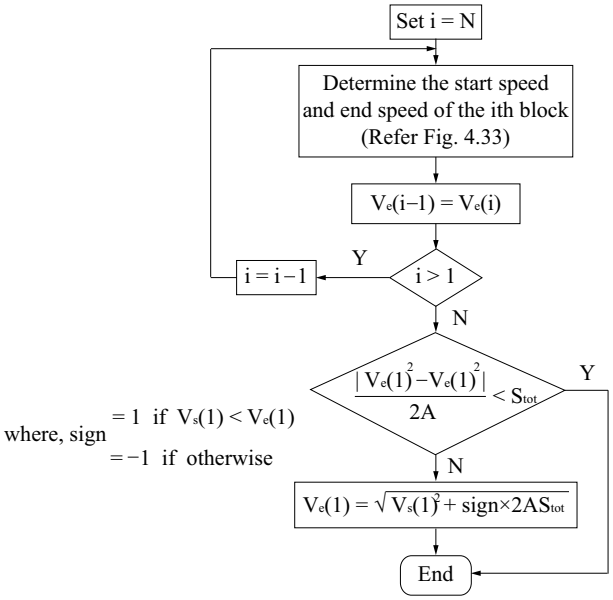


Fig. 4.32 Flowchart for determining end speed with Look Ahead

In Fig. 4.32, N denotes the number of Look Ahead buffers, $V_e(i)$ is the end speed of ith block, $V_s(i)$ is the start speed of ith block, A is the maximum allowable acceleration, and S_{tot} is the length of the current block. In addition, the start speed of the current block is equal to the end speed of the previous block and is used as input to the Look Ahead algorithm. In the Look Ahead algorithm, calculation of the start speed and end speed of each block is performed in reverse order from the look-ahead block to the current block. By comparing the end speed of the current block with the start speed of the current block, the availability of the end speed of the current block is checked as follows:

$$\frac{V_e(1)^2 - V_s(1)^2}{2A} < S_{tot} \tag{4.95}$$

If the distance that is required for acceleration or deceleration to the end speed from the start speed is smaller than the length of the current block, as given by Eq. 4.95, the end speed of the current block is available. Otherwise, the end speed of the current block should be calculated again using Eq. 4.96 based on the length and the start speed of the current block.

$$V_e(1) = \sqrt{V_s(1)^2 + sign \times 2AS_{tot}} \tag{4.96}$$

$$\text{where,} \quad sign \begin{matrix} = 1 & if\ V_s(1) < V_e(1) \\ = -1 & otherwise \end{matrix}$$

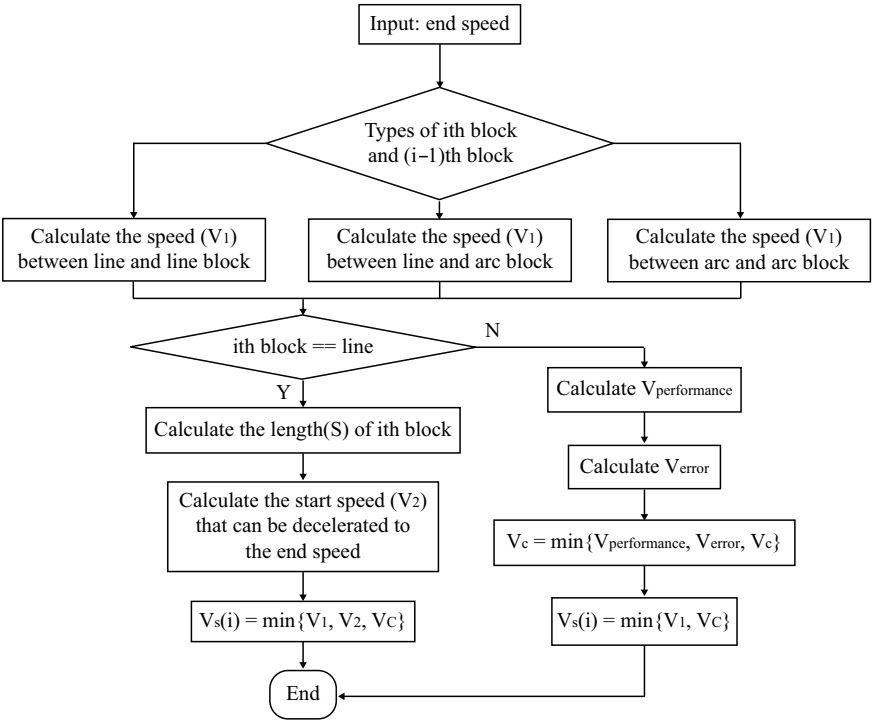


Fig. 4.33 Flowchart for calculating start speed for Look Ahead algorithm

Figure 4.33 shows the flow chart for calculating the start speed of the *i*th block for the Look Ahead algorithm. In the Look Ahead algorithm, it is assumed that the end speed of the last block used for look ahead is zero. The start speed of the last block is computed by carrying out the procedure shown in Fig. 4.33. The start speed calculated by the previous step is set to the end speed of the previous block to the last block. By repeating the procedure of Fig. 4.33, we can compute the start block of the previous block to the last block. Next, the procedure, to obtain the start speed of the current block by using the end speed is described below.

The start speed of the current block is determined based on the angle between the current block and the previous block. In Sections 4.3.3 and 4.3.4 the methods for determining the corner speed were addressed in detail and one of them can be used for calculating the corner speed $V1$. The start speed that is achievable for the end speed of the block, V_2, is computed based on the length of the block.

The smallest among V_1, V_2, and the commanded feedrate is determined as the start speed of the block. While, if the current block is a circular path, the feasible feedrate of the circular path as well as the above-mentioned speeds should be computed again. Because, axes are moving along a circular path, the speed of each axis is changed continuously and the change of speed is restricted to the acceleration performance of the machine tool. In addition, in order to decrease the chordal deviation error during circular interpolation, the circular path should be divided into smaller line segments and these small line segments make the actual feedrate on a circular path small. Therefore, for a circular path the feasible feedrate should be calculated based on the curvature (or radius) and acceleration performance of the machine tool. The feasible feedrate should be reflected on when computing the start speed.

Equation 4.97 shows the performance index of a machine tool and indicates the feasible acceleration of the machine tool during circular path machining.

$$\alpha = \frac{(V_{performance})^2}{R} \tag{4.97}$$

where, α is a performance index, $V_{performance}$ is the feasible feedrate on a circular path, and R denotes the radius of the circular path. We can obtain the maximum feasible feedrate, V_{error}, by considering the chordal error using Eq. 4.98.

$$V_{error} = \frac{2R \times \cos^{-1} \frac{R_E}{R+E}}{T_S} \tag{4.98}$$

where, E denotes the chordal error.

The minimum of $V_{performance}$, V_{error}, and the commanded feedrate is selected as the feasible feedrate on the circular path. Finally, the smaller value between the selected feedrate and the corner speed is used as the start speed of the block.

4.4.1.4 Speed within Block

Using the above-mentioned procedure, we can obtain the adequate start and end speeds of the current block. After calculating the speeds, it is necessary to calculate the speed at each iteration time of interpolation. Figure 4.34 shows the procedure for calculating the speed at each iteration time within the block.

In Fig. 4.34, F_i means the speed at the current iteration time, L_{rem} is the remaining length of the current block, F_c is the programmed feedrate of the block, and α_2 denotes the acceleration and deceleration within the block.

What is important during computation of the speed at each iteration time is whether it is possible to decelerate to the end speed when deceleration begins at the current iteration time. If the remaining length of the block is enough to decelerate to the end speed and the current speed is smaller than the programmed feedrate, it is necessary to increase the speed. If the remaining length of the block is enough to decelerate and the current speed is equal to the programmed feedrate, the speed is kept. If the remaining length of the block is not enough to decelerate, it is necessary

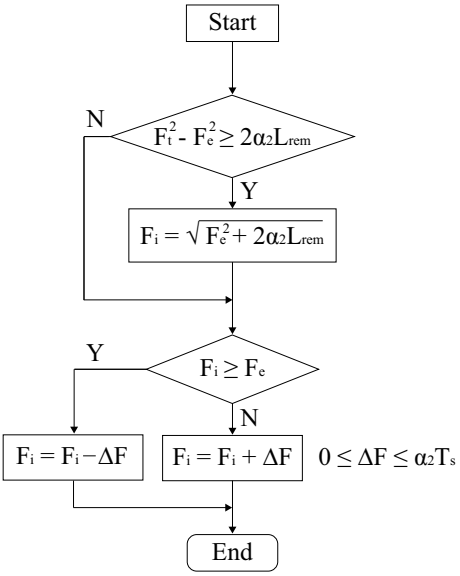


Fig. 4.34 Flowchart for calculating block speed

to decrease the speed. By repeating the above-mentioned process at each iteration time, we can obtain the speed at every iteration time of interpolation.

4.4.2 Simulation Results

If we apply the Look Ahead algorithm whose buffer size is 2 to the example shown in Fig. 4.29, we can obtain the speed profile shown in Fig. 4.35.

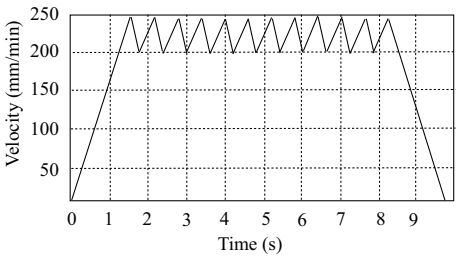


Fig. 4.35 Look-ahead buffer size 2

In the case that the look-ahead buffer size is 2, in order to compute the start and end speeds of the current block, only one next block is considered. Because we do

not know the information about the blocks after the next block, it is assumed that the end speed of the next block is zero. The feasible entry speed of the next block is computed as 200 mm/min ($= \sqrt{0^2 + 2 \times 2.094 \times 9600}$) by Eq. 4.99. When this feasible entry speed determines the end speed of the current block, the maximum speed of the current block can reach 245.17 mm/min.

$$V_s = \sqrt{V_e^2 + 2AS} \tag{4.99}$$

Comparing Fig. 4.30 with Fig. 4.35 shows that the Look Ahead algorithm increases the maximum feasible speed and decreases the machining time. Because the start speed of the first block is zero, the maximum reachable speed of the first block is 200 mm/min and the end speed of the first block is also 200 mm/min. However, in the case of the second block, because the start speed is 200 mm/min and the feasible end speed is 200 mm/min, it is possible to reach 245.17 mm/min.

If the size of the look-ahead buffer becomes 3, the maximum reachable speed becomes higher and the resulting machining time decreases. Figure 4.36 shows the speed profile that is generated when the size of the look-ahead buffer is 3. When we compute the end speeds of the blocks sequentially, the end speeds of the blocks are 0 mm/min, 200 mm/min and 283 mm/min ($= \sqrt{200^2 + 2 \times 2.094 \times 9600}$), respectively.

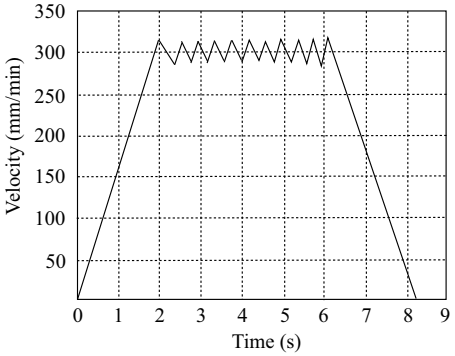


Fig. 4.36 Look-ahead buffer size 3

If the size of the look-ahead buffer increases to 6, a speed profile that has no speed fluctuations and looks like a normal block is generated, as shown in Fig. 4.37.

If linear paths and circular paths are combined, it is necessary to compute the corner speed based not only on the length of blocks but also the angle between blocks in order to compute the speed of the blocks to be looked at ahead. For example, assume that there is a part program that has paths shown in Fig. 4.38, and assume that the programmed feedrate of the paths is 2000 mm/min, the acceleration time is 200 msec, and the maximum allowable acceleration is 200000 mm/min^2.

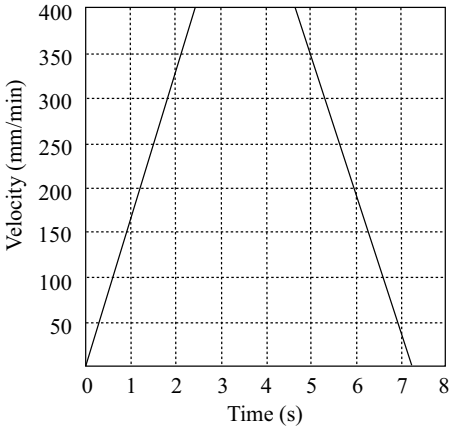


Fig. 4.37 Look-ahead buffer size 6

Figure 4.39 shows the speed profile considering only the length of the blocks, while Fig. 4.40 shows the speed profile when both the length of blocks and the angle between blocks are considered. From Fig. 4.40 we can see that, due to the Cartesian maximum allowable acceleration, the speed decreases at the corner where the arc and the line join.

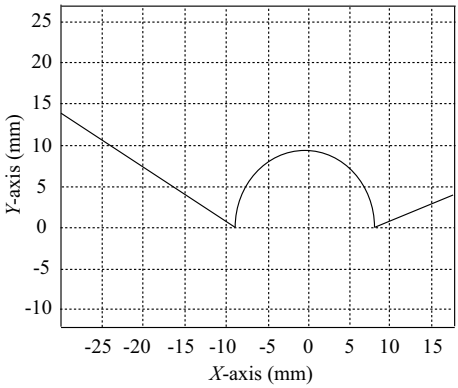


Fig. 4.38 Look-ahead path

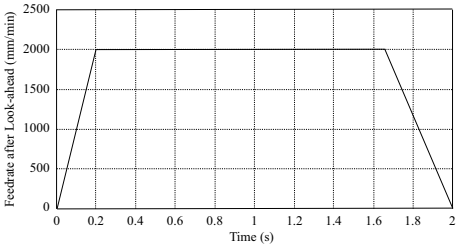


Fig. 4.39 Look-ahead speed profile 1

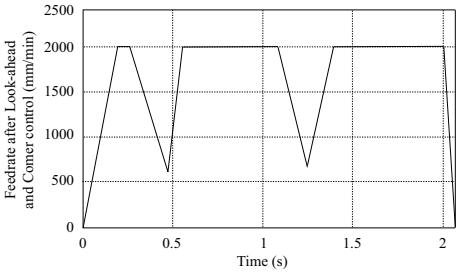


Fig. 4.40 Look ahead speed profile 2

4.5 Summary

In ADCBI, the speed profile for two successive blocks is generated by considering the type of the current block (*e.g.* normal block or short block), regardless of whether the commanded feedrate of the two blocks is the same or different and whether the next block is the last block.

Furthermore, unlike a linear path, an arc path generates acceleration and deceleration due to the change of velocity and the acceleration and deceleration generate a mechanical shock. In particular, the amount of the mechanical shock is proportional to the acceleration and the acceleration is inversely proportional to the radius of the circular path and proportional to the square of the feedrate. Therefore, the maximum allowable acceleration value should be restricted. When the speed profile for the circular path is calculated it is necessary to compute the acceleration due to the commanded feedrate and to compare the acceleration with the pre-specified maximum allowable acceleration value.

In ADCBI-type NCK, after the speed profile on path is computed the displacement of the tool at every iteration time of interpolation is calculated based on the speed profile. Therefore, machining error does not occur at corners where two blocks join. However, if the speed does not sufficiently decelerate when the tangents of two blocks are different from each other at the corner, mechanical shock occurs. Therefore, it is necessary to calculate the corner speed that the machine can stand when the speed profile is computed. In this book, as the method of determining the cor-

ner speed, two methods were introduced; the first is based on the joint angle of two successive blocks and the second is based on the maximum allowable ratio of speed change of each axis. In particular, the second method is typically used for machine tools.

In ADCAI-type NCK, Acc/Dec control is applied to each axis separately and this leads to machining error in the case of machining an arc. Because acceleration for the next block and deceleration for the previous block are done simultaneously near the corner, machining error is inevitable. However, in ADCAI-type NCK, the feasible speed at the end of the current block is always monitored by applying the look-ahead algorithm and monitoring the remaining length of the block. Therefore, it is possible to calculate the appropriate moment for acceleration or deceleration and to reduce the machining error within a specified amount. Because of these characteristics, ADCBI-type NCK is widely used for high-speed machining and ADCAI-type NCK is used for machining where high accuracy is not important, such as roughing machining.

Chapter 5
PID Control System

The information of a program block for moving the axis of a machine tool passes sequentially through the interpreter, the interpolator, and the Acc/Dec controller before being finally transmitted to the position controller. The position controller has the displacement at each interpolation interval from the interpolator as input and performs feedback control to minimize the position error. In this chapter, the Proportional Integral Derivative (PID) controller, which is widely used in industry, will be introduced and gain tuning for PID control will be addressed. In addition, the feedforward controller for high-speed applications will be introduced.

5.1 Introduction

In general, the axis control module for CNC systems of machine tools can be classified into a three-tier architecture as shown in Fig. 5.1. The control module consists of the adaptive control module and the error compensation module in the upper layer, the interpolation module in the middle layer, and the spindle control module and servo control module in the lower layer.

The adaptive control module in the upper layer generates optimized cutting conditions such as spindle speed and feedrate based on the programmed spindle speed, the programmed feedrate, measured actual cutting force, and the cutting capacity of machine tools. Accordingly, the adaptive control module plays the role of increasing Material Removal Rate (MRR) and decreasing machining time to increase productivity.

Another module in the upper layer, the error compensation module, carries out the compensation of error factors that cause deviations from the programmed path. This module handles various kinds of error including the heat from moving elements, the volumetric error of machine tools, tool wear, and tool deflection.

The optimized and compensated feedrate and position instructions from the upper layer are transmitted into the interpolator in the middle layer. Finally, the interpolated

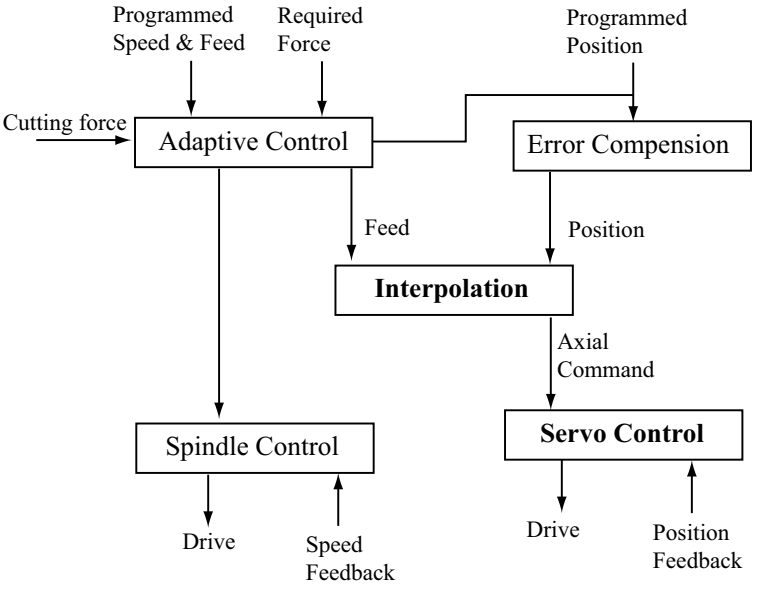


Fig. 5.1 Three-tier CNC system architecture

position instructions for each axis are fed to the servo control module in the lower layer.

The spindle speed optimized by the adaptive control module in the upper layer is transmitted to the servo control module in the lower layer. The servo control module and the spindle control module generate control instructions using appropriate control algorithms and the instructions generated are sent to the drivers.

As mentioned above, a fully functional CNC system consists of various control modules that provide many functions. Practically, however, the majority of CNC systems include control systems that consist of the interpolation module of the middle layer and the servo and spindle control modules of the lower layer.

In this chapter, a position control method in the servo control module for controlling the machine tool axes will be addressed. Except for the interpolation module, the other modules are addressed in other books and the interested reader should refer to these.

5.2 The Servo Controller

The servo controller, which enables the movement control of each axis based on position commands from the interpolator, is the last one in the NCK system. The servo controller should be able to control speed over a wide range, from fast speed for high-speed machining (m/min) to slow speed for high-accuracy machining (mm/min).

Furthermore, it should also guarantee practically reasonable accuracy and have robustness against external disturbances.

To fulfill the above-mentioned requirement, a closed-loop control, where the actual speed and position data are monitored and fed back to the servo controller, is commonly considered. In most systems actuated by servo motors, because the position and speed instructed to the servo motors are practically different from the actual values, the servo controller includes a feedback control loop where the actual position and speed data from the servo motors are fed back to the controller, and the controller generates commands to compensate for errors between commanded values and the feedback data. Therefore, the performance of the position controller depends on the control algorithm and the position and speed detectors as well as the control targets, such as motor, coupling, ball screw, slider, etc.

The closed loop for controlling the axis of the CNC system forms a cascade structure that is connected with a position, speed and current loop arranged in series, as shown in Fig. 5.2. The current loop is located at the innermost position of the loop, the speed control loop encompasses the current loop and the position control loop encompasses the speed control loop.

In the cascade-style control architecture it is easy to tune the characteristics of each control loop. However, it is necessary first to guarantee the stability of the inner control loop for the stability of the entire control loop and to minimize the dependency between the outer loop and the inner loop. To achieve the above-mentioned purpose, the inner loop should be set to perform with a faster system response than that of the outer loop by tuning the gains of each control loops.

In a CNC system, generally the NCK task is the outermost loop and performs only the position control with slowest response; while the speed control and current (torque) control are executed in a servo drive system. However, recently the conventional control configurational has no longer been preserved in the CNC system industry. All control loops can be implemented in the drive system, or the position and speed control loop are performed within the NCK task, while only the current loop for the fastest response is performed within the drive system. This can be chosen on the basis of the hardware capacity of the NCK and driver system, and the application purpose of the CNC system.

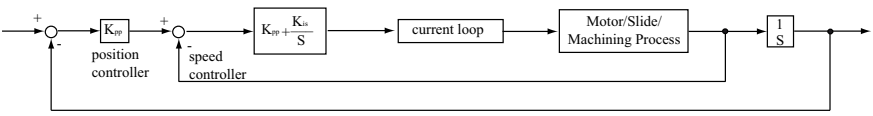


Fig. 5.2 Cascade structure for closed-loop control

Accordingly, the movement of the table or the tool attached to the machine tool depends on the response characteristics of the position loop, which is located at the outermost loop and has the slowest response. The slow response of the position loop creates a following error in the control system, thereby increasing the following error and decreasing the machining accuracy of corners or circular movements by ma-

chines having more than two axes. Therefore, the control parameter of the position loop should be set to achieve system response such as a high machining accuracy and short machining time. However, the cascade-style control architecture has a fundamental drawback that restricts the response of the position loop.

5.3 Servo Control for Positioning

A CNC system is widely used for the position control of different positioning machines including robots, chip mounters, semiconductor handlers as well as machine tools. The servo control system of the positioning machines is the core and most important part for the machine performance and quality, and the control strategy of each axis results in various positioning errors. Generally, the axis control strategy of a CNC system can be classified into point-to-point control, tracking control, and contour control. In point-to-point control, the most important factor is the elapsed time for moving from one point to another, ignoring the contour error during axis movement. In the case of tracking control, the most important factor is to minimize the following error which is the amount of deviation from the reference trajectory. While contour control involves minimizing the contour error occurring with simultaneous movement of more than two axes.

The trajectory and contour error appearing in the multi-axis machine tools are shown in Fig. 5.3. P is the current position, and R denotes the reference point to go to. The purpose of tracking control is to minimize the position errors of each axis, e_x, e_y, which are the deviation of the current position from the reference point. On the other hand, contour control is implemented to minimize the contour error, ε, which is the error of the current position to the desired contour. Therefore, the position error and contour error are used as the indices for evaluating the accuracy of the controlled path.

The position error is the linear distance between the actual tool position and the reference point, and is represented like Eq. 5.1 by using the position error of each axis.

$$e = \sqrt{(e_x)^2 + (e_y)^2} \tag{5.1}$$

where, e denotes the position error and e_x and e_y denote the position error of X-axis and Y-axis respectively. The contour error, ε, is the minimum distance between the actual tool position and the desired path, as shown in Fig. 5.3.

Minimising the position error of each axis does not always mean the reduction of the contour error. In the case of cutting machine tools, the accuracy of the final machined shape is the important factor, therefore reduction of the contour error should be considered more seriously than reduction of the position error.

In general, point-to-point control is used for controlling the movement to the commanded position within a short settling time, regardless of the intermediate path, by utilizing a P controller or PI controller. Compared with point-to-point control, how-

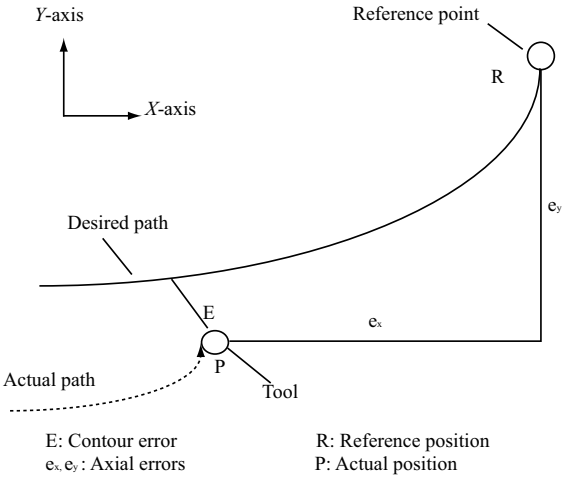


Fig. 5.3 Difference errors between real and reference points

ever, tracking control and contour control have many problems to be considered. Because of inherent system specific errors, including disturbance, friction force, backlash, distortion of table, and the characteristics of the servo system as well as dynamic errors due to high speed operation of the servo system, acceleration or deceleration operation, and the change of movement direction, some difference between the desired path and the actual controlled path cannot be avoided during contour machining by multi-axis machine tools.

Therefore, in the position controller of a CNC system there is a variety of control algorithms, including the P controller, PID controller, fuzzy controller, feed-forward controller, predictive controller, and cross-coupling controller that have been introduced to support point-to-point control, tracking control, and contour control.

Compared with the P controller, feedback controllers such as the PID controller and the Fuzzy controller decrease the position errors of each axis. The feed forward controller is useful for reducing the multi-axis error by reducing the axial error of each axis, and finally contributes to reduction of the contour error. The cross coupling controller performs accurate control in real time by generating control commands to reduce contour error based on a contour error model. In this textbook, the causes of various errors and algorithms for reducing the position error will be addressed.

5.4 Position Control

The last task of NCK is the position control task where the reference position from the interpolator and the actual position fed back from a servo motor (encoder) are

compared and control to decrease the difference between these two positions is carried out.

A PID controller and a feedforward controller are typical algorithms for carrying out the position control task and they will be described in the following sections.

5.4.1 PID Controller

Although various modern control theories have been developed, the PID controller has been widely used in industry due to its simple architecture and ease of implementation in hardware or software. Furthermore, the PID controller can easily be implemented, even when an accurate mathematical model of target system is unknown, and realizes excellent control performances, including target-value following or disturbance rejection.

However, the PID controller can only be used for time-invariant linear systems and it is necessary to tune gains accurately based on the dynamics of the process. When the process dynamics are changed due to change of the system weights, the gain tuning process should be redone. In addition, the PID controller has the limitation that it can be applied only for SISO (Single Input Single Output) systems.

Because CNC systems generally control more than two axes, they can be regarded as systems with more than one input and output. However, it is possible to use an individual PID controller for each axis because each axis of machine tools is actually independently controlled based on the interpolated data of every tiny interpolation time interval. Accordingly, each axis is controlled by a PID controller having a single input and output port.

The PID controller can be utilized as a P controller for only proportional (P) control actions, an I controller for only integral (I) control actions, and a D controller for only derivative (D) control actions. It can also be used as a combination of single controllers such as a PI controller, PD controller as well as PID controller. Actually, for position control of machine tool axes, a P controller or a PI controller having a small integral gain has been widely used.

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s \tag{5.2}$$

As shown in Fig. 5.4, a PID controller generates the output u as an input of the process, which makes the error (difference between the process output feedback and a reference input) zero. The transfer function of the PID controller including the proportional control action, integral control action, and derivative control action is represented by Eq. 5.2. Therefore, the design of a PID controller involves the determination of the proportional gain K_p, the integral gain K_i, and derivative gain K_d.

The proportional control action plays the role of handling the immediate error. Through the proportional control action, it is possible to decrease the rise time of a

system and the steady-state error. However this causes overshoot increase and steady-state error.

The derivative control action has the role of handling errors based on learning from the past. With the derivative control action it is possible to decrease overshoot and settling time. The integral control action has the role of handling future errors. Using the integral control action, it is possible to remove a steady-state error and decrease the rise time of a system. However, this causes an increase of overshoot and settling time.

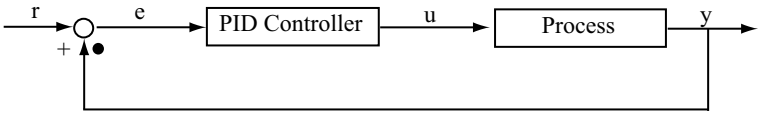


Fig. 5.4 Block diagram of position control in PID

$$G_c(s) = K_p(1 + \frac{1}{T_i s} + T_d s) \tag{5.3}$$

The transfer function in Eq. 5.2 can be rewritten as Eq. 5.3 where the integral time T_i and derivative time T_d are defined as in Eq. 5.4a and Eq. 5.4b, respectively.

$$T_i = K_p/K_i \tag{5.4a}$$

$$T_d = K_d/K_p \tag{5.4b}$$

The following gives more details about control actions of each component in a PID controller. First, in P control, meaning proportional control, the system error is multiplied by a constant P, called the *proportional gain*, in order to compensate for the system error. Therefore, larger proportional gain typically results in faster response of a system and decreases the time spent in going to the reference point. However, because large proportional gain makes it impossible to regulate accurately the system error, there is continuous vibration and abnormal noise. Therefore, the amount of the proportional gain is restricted due to this reason.

Second, I control, meaning *integral control*, is used in the case of not going to the reference point after transition to the steady state. In I control, the error is integrated over a period of time, multiplied by a constant I, called the integral gain, to reduce the integrated errors from the past. Larger integral gain results in faster response during transition states. Accordingly, it is necessary to use an integral gain within an adequate range because large integral gain results in excessive overshoot or undershoot.

Third, in D control, meaning *derivative control*, the first derivative over time (the slope of the error) is calculated, and this derivative is multiplied by a constant D, called the derivative gain, for damping the system and removing the vibration of a system during a steady state. Larger derivative gain results in a faster response. However, large derivative gain causes vibration of a system. Therefore, the amount

of the derivative gain should be restricted because the first derivative of position over time is sensitive to noise.

Above, the effect of different types of gain on the PID controller is explained. Therefore, in practice, a variety of combinations of P control, I control, and D control has been selectively used according to the purpose for which the control is used. Typically, the PI controller which has a relatively fast response has been widely used because D control has difficulty in gain tuning and easily results in vibration.

The transfer functions for the proportional controller, the integral controller, and the derivative controller defined in a continuous time domain can be approximated by the transfer functions for the discrete time domain for digital control as in Eqs. 5.5, 5.6 and 5.7.

$$G(s) = K_p \Leftrightarrow G(z) = K_p \tag{5.5}$$

$$G(s) = \frac{K_i}{s} \Leftrightarrow G(z) = \frac{K_i T}{1 - z^{-1}} \tag{5.6}$$

$$G(s) = K_d s \Leftrightarrow G(z) = \frac{K_d(1 - z^{-1})}{T} \tag{5.7}$$

By combining the above three approximated equations, the transfer function for the PID controller for discrete time domains can be approximated as in Eq. 5.8.

$$G_c(z) = \frac{k_0 + k_1 z^{-1} + k_2 z^{-2}}{1 - z^{-1}} \tag{5.8}$$

where, $k_0 = k_p + k_i T + \frac{k_d}{T}$, $k_1 = -k_p - \frac{2k_d}{T}$, and $k_2 = \frac{k_d}{T}$ and T denotes the iteration time for position control. Accordingly, the output of the digital PID controller with proportional, integral, and derivative control can be represented as the difference equation, as in Eq. 5.9.

$$\frac{u}{e} = \frac{k_0 + k_1 z^{-1} + k_2 z^{-2}}{1 - z^{-1}} \tag{5.9a}$$

$$\text{or, } u(n) - u(n-1) = k_0 e(n) + k_1 e(n-1) + k_2 e(n-2) \tag{5.9b}$$

Equation 5.9b can be rewritten as Eq. 5.10.

$$u(n) = u(n-1) + k_0 e(n) + k_1 e(n-1) + k_2 e(n-2) \tag{5.10}$$

Consequently, in the PID controller for the discrete time domain, the input of the controller at the current time is computed based on the controller's input at the previous iteration time, the error at the current time, the error at the previous iteration time (one step behind the current sampling time), and the error at the time before that (two steps before the current sampling time).

For example, when the PID controller is used for controlling a rotary table, the block diagram for position control of the complete system is as shown in Fig. 5.5.

Assume that the transition function for the particular axis of a rotary table is defined as in Eq. 5.11.

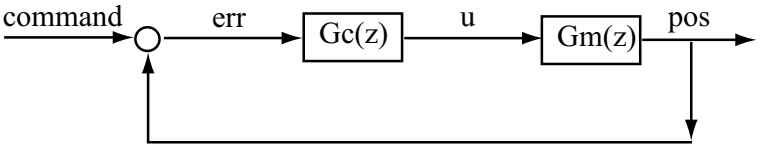


Fig. 5.5 Block diagram for controlling rotary table system

$$Gm(z) = \frac{a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}} \tag{5.11}$$

When Eq. 5.11 is expressed as the relationship between the controller output signal u and the actual position, pos, the discrete form with respect to the current iteration time, k, can be denoted as in Eq. 5.13.

$$\begin{aligned} pos(k) = b1 * pos(k-1) + b2 * pos(k-2) + b3 * pos(k-3) + \\ a1 * u(k-1) + a2 * u(k-2) + a3 * u(k-3) \end{aligned} \tag{5.12}$$

The actual position data detected from the servo is fed back as the input to the PID controller. The actual position data is compared with the command position data from the interpolator and the position error between the actual position and the command position data is input to the PID controller. Finally, the PID controller generates the output signal u for reducing the error, and signal u is fed to the controlled system Gm(z).

According to the above example, the program for realizing the PID controller for position control can be written as follows:

```
Procedure PID_Controller()
{
/ Setting process variables /
    a1=0.007001; a2=0.017284; a3=0.002475;
    b1=1.782; b2=-0.906; b3=0.124; /Process parameters/
    tpos=0.001; / PID gain setting /
    zkp=2.0; zki=0.0001; zkd=0.0;
    /Here, only PI schema is applied, I gain needs to be very low./
     zk0=zkp+zki*tpos+zkd/tpos;
    zk1=-zkp-2*zkd/tpos;
    zk2=zkd/tpos;
/ Initialisation of control variables /
    err1=0.0; err2=0.0; command=0.0; feedback=0.0;
```

```
    pos=0.0; pos1=0.0; pos2=0.0; pos3=0.0;
    u=0.0 u1=0.0; u2=0.0; u3=0.0;
    err=0.0; err1=0.0; err2=0.0;
begin (n=1,200)
    pos_command(n)=10.0-real(n)*0.02; /Generation of position/
                                          /command for simulation/
    pos_command (n+100)=8.0+real(n)*0.02;
end / Position control loop /
begin (n=1, 300)
    command= pos_command (n); /Command value for position control/
    feedback=pos ; /Position feedback/
    err=command - feedback; /Error/
    u=u1+zk0*err+zk1*err1+zk2*err2; /PID control algorithm, Eq. 5.10/
    /Transfer equation for rotary table, Gm, Real position simulation/
    /by Eq. 5.10/
    pos=b1*pos1+b2*pos2+b3*pos3+a1*u1+a2*u2+a3*u3;
    /Update for the next position loop/
    pos3=pos2; pos2=pos1; pos1=pos;
    u3=u2; u2=u1; u1=u;
    err2=err1; err1=err;
end
}
```

As shown above, it is easy to implement the PID controller for motion control. However, the performance of the PID controller depends highly on the P, I, and D gain according to the given environment. Whenever an operating point or the characteristic of the target process changes, it is necessary to tune the gains. The way to set the P, I, D gain is called the "gain tuning method" and an expert is typically required for gain tuning.

In the next section, a gain tuning method for the PID controller will be introduced.

5.4.2 PID Gain Tuning

If adequate gain for the PID controller is not set, the system response may become slow, vibration may occur, or the desired accuracy may not be achieved. Therefore, setting adequate gains is an important design factor.

As the gain tuning method for a PID controller, there are the Ziegler–Nichols method, where a mathematical model for the target process are not necessary, and the Relay method. For these methods, a user with a lot of experience should tune P, I, D gain by trial and error and, therefore, it takes a long time to complete gain tuning.

As gain tuning methods based on a target process model, there are the Frequency response method, the Pole placement method, and the Pole-zero cancelation method. However, because of the complication of the mathematical model for the target pro-

cess, it is difficult to derive the model of a target process in practice. Furthermore, because the experiment for obtaining frequency response is complicated and it is difficult to obtain an accurate result, the accuracy of the gains tends to decline in the case when the degree of a system is high. Also, because of the disturbance due to the load, the pole and zero in the system are not exactly canceled and thereby, the performance of the system declines.

Recently a controller with an auto tuning function has been developed that tunes the gains automatically to help users. The auto-tuning function in this controller performs internally advanced gain tuning methods including those mentioned above.

5.4.2.1 The Ziegler–Nichols Method

The Ziegler–Nichols method, which is one of the experiment-based gain tuning methods, is a method where the target process is tested as an open loop plant and the P, I, and D gains related to the characteristic of the transient response are calculated by simple formatted equations. This gain tuning method can be executed in two ways; the first is the step response method based on the response curve of the process and the second is the ultimate sensitivity method.

The step response method is a gain tuning method using the damping ratio of 0.25 from experiment and experience where the dominant transition diminution becomes 25% of the previous one after one cycle in the time domain. This method can be applied to a safe system where oscillation does not occur because the main polar-pole of the target process is not a complex conjugate, and typically a system having an S-shaped response curve satisfies this condition.

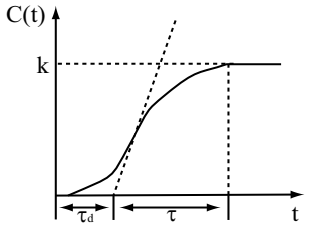
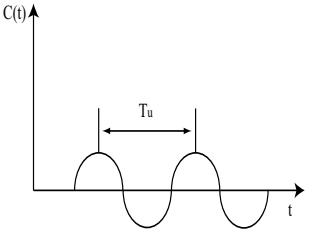
After the step pulse is applied to the system, the S-shape response curve is achieved. By analyzing the S-shape response curve, various characteristic parameters (response delay time, time constant) can be extracted, and, finally, the gains for the PID controller are calculated by the equations shown in Table 5.1.

The ultimate sensitivity method is useful for a target process that has poles at the origin or unstable poles resulting in system oscillation. For the the ultimate sensitivity method, first set $T_i = \infty$, $T_d = 0$ in Eq. 5.3, and increase the proportional gain step by step. When system oscillation is detected, a critical gain (K_u) and critical frequency (T_u) can be extracted. Finally, the PID controller's gains are obtained by the equations shown in Table 5.1. If system oscillation does not occur, even when the proportional gain is increased, this method cannot be applied.

The Ziegler–Nichols method is good and simple for gain tuning, but needs a fine tuning process by a tuning expert. Also, it cannot achieve satisfactory control performance in the case of a system having small damping characteristics.

As an automatic gain tuning method for a PID controller, a relay method is combined with the Ziegler–Nichols frequency response method to extract the critical gain and critical frequency. Besides these, various techniques, including adaptive control theory, optimization theory, and fuzzy control theory have previously been developed. Please refer to the bibliography for details of each technique. In this book, the

Table 5.1 Ziegler–Nichols Method

Control		Step Response	Ultimate Sensitivity Method
Condition		Step response has S-shape	As proportional gain increases, output oscillates
Procedure		1. By inputting step pulse to target, generate S-shaped response. 2. Draw tangent line on point of inflection of output. From intersection points where tangent line meets time axis and response K, calculate response delay time τ_d and time constant τ	1. By setting $T_i = \infty$ and $T_d = 0$ in closed loop system, activate only proportional control loop. 2. Until output c(t) keeps oscillating, regulate K_p 3. Proportional gain at moment of oscillation is kept to critical gain K_u. Set critical frequency T_u to frequency at this moment.
Output response			
Ctrl. eqn.	P	$K_p = \tau/\tau_d$	$K_p = 0.5K_u$
	PI	$K_p = 0.9\tau/\tau_d, T_i = 3.3\tau_d$	$K_p = 0.45K_u, T_i = 0.8T_u$
	PID	$K_p = 1.2\tau/\tau_d, T_i = 2\tau_d,$ $T_d = 0.5\tau_d$	$K_p = 0.6K_u, T_i = 0.5T_u$ $T_d = 0.125T_u$

relay method that is commercially applied in the industrial field, will be explained in detail.

5.4.2.2 Relay Gain Tuning

The Ziegler–Nichols method mentioned above has the limitation that it cannot be applied to a system that is not critically oscillated with only proportional gain control. To overcome the non-oscillating problem, a relay gain tuning method was introduced. In this method, a relay is utilized for system oscillation by force. When oscillation occurs, the limits of frequency and amplitude are extracted from the system response.

The circuit for the relay-based automatic gain tuning method consists of the conventional PID controller, a relay, and a switch for switching reference input. The relay and the switch are located in front of the PID controller, as shown in Fig. 5.6.

The difference between the reference input, r, and the process output, y, is input to the relay. The output of the relay is directly input to the PID controller. At the initial

stage of gain tuning, the switch is connected to contact a, thereby the reference input bypasses the relay and is directly input to the PID controller. Therefore, a non-tuned PID controller is operated.

At the next automatic tuning stage, the switch is connected to the contact b, and the difference between the reference input and the process output is fed to the relay. System oscillation due to the limit cycle phenomena is induced, and the ultimate frequency and amplitude are extracted for automatic tuning of the PID gains.

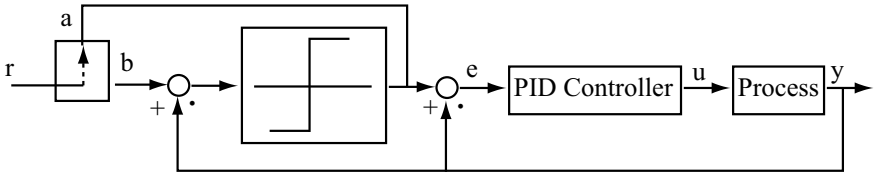


Fig. 5.6 Circuit for relay-based automatic gain tuning

The automatic gain tuning stage can be done as shown in Fig. 5.7a. The relay is modeled as a system with gain K_R. Furthermore, $G_C(s)$ and $G_P(s)$ mean the transfer transition function of the PID controller and the process, respectively. The error can be expressed as Eq. 5.13.

$$e = K_R r - y - K_R y = K_R r - (1 + K_R) y \tag{5.13}$$

If Fig. 5.7a is arranged as the form of the rightmost equation of Eq. 5.13, it can be denoted as Fig. 5.7b, and the loop transfer function is $(1 + K_R) G_C(s) G_P(s)$.

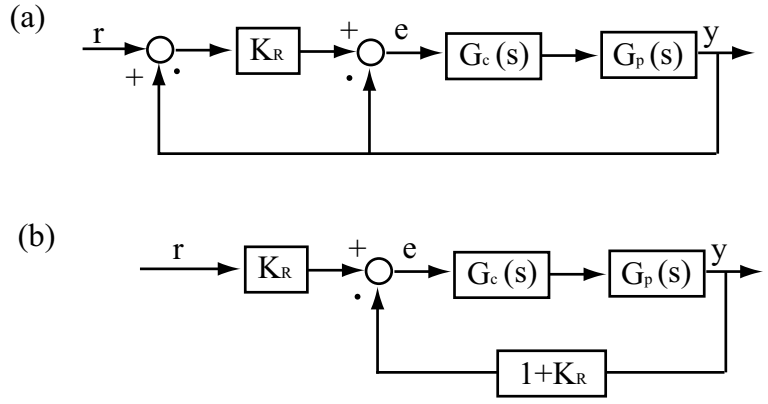


Fig. 5.7 Block diagrams of gain setting

While the gain of a relay is expressed using the describing function of non-linear system theory as Eq. 5.14.

$$K_R = \frac{4d}{\pi a} \tag{5.14}$$

where, a is the amplitude of the process output and d denotes the amplitude of the relay.

At this stage, the process transfer function is assumed to be as in Eq. 5.15 which is used in the Ziegler–Nichols method and briefly represents the characteristics of the process.

$$G_p(s) = \frac{e^{-\tau s}}{T_p s} \tag{5.15}$$

where, T_p is the time constant and τ stands for the dead time.

On the other hand, the transition function of the PID controller can be represented as in Eq. 5.16.

$$G_C(s) = K_C \left(1 + \frac{1}{T_i s}\right) \left(\frac{T_d s + 1}{\alpha T_d s + 1}\right) \tag{5.16}$$

where, T_i and T_d, respectively, denote the integration time and derivative time and α stands for derivative gain.

If oscillation is continuously induced at the moment that the relay is connected with the PID controller and the automatic gain tuning stage begins, the frequency and amplitude of the induced oscillation are derived as in the following equations,

$$\begin{aligned} &\arg\left[(1+K_R)G_P(j\omega_0)G_C(j\omega_0)\right] = \\ &\arg\left[(1+K_R)\frac{e^{-\tau s}}{T_p s}\left(\frac{K_C(T_i s+1)}{T_i s}\right)\left(\frac{T_d s+1}{\alpha T_d s+1}\right)\right]_{s=j\omega_0} = -\pi \end{aligned} \tag{5.17}$$

$$\begin{aligned} &\left|(1+K_R)G_P(j\omega_0)G_C(j\omega_0)\right| = \\ &\left|(1+K_R)\frac{e^{-\tau s}}{T_p s}\left(\frac{K_C(T_i s+1)}{T_i s}\right)\left(\frac{T_d s+1}{\alpha T_d s+1}\right)\right|_{s=j\omega_0} = 1 \end{aligned} \tag{5.18}$$

where ω_0 is the oscillation frequency. The dead time τ and time constant T_p which show the characteristic of the process can be found using Eqs. 5.19 and 5.20.

$$\tau = \frac{\tan^{-1}(T_i\omega_0) + \tan^{-1}(T_d\omega_0) - \tan^{-1}(\alpha T_d\omega_0)}{\omega_0} \tag{5.19}$$

$$T_p = \frac{(1+K_R)K_C\sqrt{T_i^2\omega_0^2+1}\sqrt{T_d^2\omega_0^2+1}}{T_i\omega_0^2\sqrt{\alpha^2T_d^2\omega_0^2+1}} \tag{5.20}$$

In order to compute PID gains by the estimation of the dead time and time constant from Eqs. 5.19 and 5.20, an ultimate gain and frequency are necessary to use the equation of the response curve of the Ziegler–Nichols method mentioned in Table 5.1. In the case that the Ziegler–Nichols gain tuning method is used, the relationships between those parameters, the frequency and the amplitude are given by Eqs. 5.21 and 5.22.

$$\arg\left[G_P(j\omega_u)G_C(j\omega_u)\right] = \arg\left[K_u\frac{e^{-\tau s}}{T_P s}\right]_{s=j\omega_u} = -\pi \tag{5.21}$$

$$\left|G_P(j\omega_u)G_C(j\omega_u)\right| = \left|K_u\frac{e^{-\tau s}}{T_P s}\right|_{s=j\omega_u} = 1 \tag{5.22}$$

where, ω_u denotes an ultimate frequency. From $T_u = 2\pi/\omega_u$, the ultimate frequency can be obtained from Eq. 5.23.

$$T_u = 4\tau \tag{5.23}$$

Also, the ultimate gain is given by Eq. 5.24.

$$K_u = \frac{2\pi T_P}{T_u} \tag{5.24}$$

Finally, by applying Eq. 5.23 and Eq. 5.24 to the ultimate sensitivity method, adequate PID gains can be obtained.

The automatic gain tuning method mentioned above is summarized in Fig. 5.8. It is assumed that the gains of the PID control loop are roughly tuned manually before executing automatic gain tuning, as depicted in the first and second steps of Fig. 5.8.

In the third step, the relay is connected at the set point input of the PID control loop. The fourth step means that the process output induces continuous oscillation as the response of the system when the relay is applied.

In the fifth step, the frequency and the amplitude of the induced oscillation are measured and stored. In the sixth step, based on measured parameters, the dead time and the time constant are computed using Eq. 5.19 and Eq. 5.20.

By substituting the dead time and the time constant for Eq. 5.23 and Eq. 5.24, the ultimate gain and amplitude are computed. Thereby, finally, the PID gains are computed by the equations of Ziegler–Nichols's ultimate sensitivity method.

In the seventh step, by decoupling the relay, the set point is coupled to the input of the PID control loop. In the eighth step, the new PID gains obtained from the sixth step are applied to the PID control loop.

5.4.3 Feedforward Control

The feedback controller, which is widely used as a servo controller, works based on the difference between input and output. The PID controller, being one type of

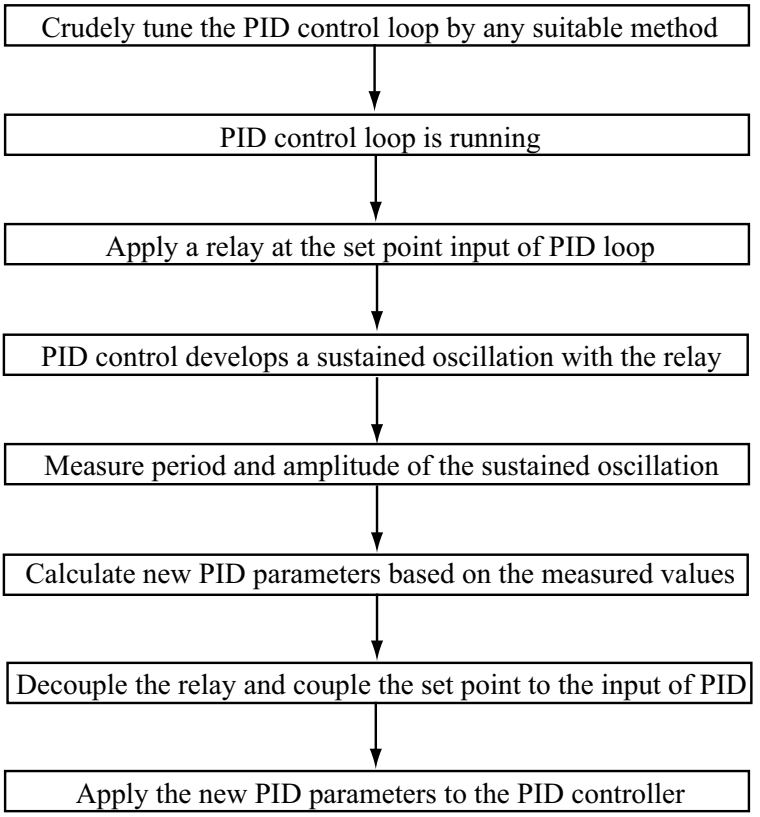


Fig. 5.8 Automatic gain tuning method

feedback controller, mentioned in the previous section, is very stable and robust even when there is disturbance. However, tracking error cannot be avoided when only a PID controller is used. This problem results in a poor cutting surface or inaccurate shape of the machined product of the CNC machine tool.

As an alternative, a feedforward controller is used together with a feedback controller in order to make up for the feedback controller's disadvantage and enable a system to track the desired reference path. The feed-forward controller does not work based on the difference between the commanded point and the actual point, as does the feedback controller, but works based on a pre-specified system model. As the feedforward controller is an open-loop type, it generates the output with calculations based on the pre-specified system model in order to increase the response characteristics of the complete system. Because of the error in the system model, perfect control of the complete system cannot be achieved. Therefore, it is typical to use feedback control and the feedforward control together.

The purpose of the feedforward control method is to overcome the response limitations of a position control loop and feedforward control belongs to tracking control

that enables the system to track the desired path by minimizing the tracking error. As shown in Fig. 5.9, the feedforward controller directly feeds the controller output to the inner loop by skipping the outer loop, which uses the fact that the response of the inner loop is faster than the outer loop in a cascade loop structure. Consequently, the feedforward scheme improves the response characteristics of the outer loop.

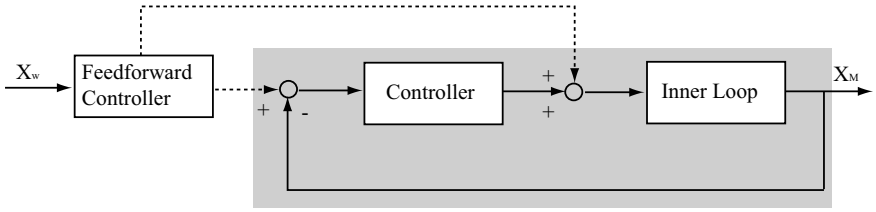


Fig. 5.9 Feedforward control placement

The feedforward control method can be classified into two types from the point of view of the loop structure.

1. After modifying the reference input, the modified input is fed to the feedback control loop, as shown in Fig. 5.10a.
2. The reference input is directly fed forward to the drive unit in the feedback control loop, as shown in Fig. 5.10b.

In the first type, $G_0^{-1}(z)$, the inverse of the transfer function of the complete system including the feedback loop, $G(z)$, is set as the transfer function of the feedforward control loop. Since the total transfer function of the complete system is set as $G_0^{-1}(z)G(z) = 1$, the actual controlled position and the desired position can become equal. In the second type, the transfer function of the feedforward controller is set to be the inverse of the transfer function of the drive unit, $D_0^{-1}(z)$. Therefore, the transfer function of the entire control loop is written as Eq. 5.25.

$$G_C(z) = \frac{D_0^{-1}(z)D(z) + H(z)D(z)}{1 + H(z)D(z)} \tag{5.25}$$

Consequently, if $D_0^{-1}(z)D(z) = 1$ is satisfied, the actual position and the desired position have come to be the same. In the first type, the inverse transfer function is very complex because the transfer functions of the feedback controller and the drive module are included in the inverse transfer function. However, when the inverse transfer function (G_0^{-1} or D_0^{-1}) has unstable poles, modification of the feedback controller is necessary. For modification of the feedback controller, the first type is more suitable than the second.

As typical techniques of the first type, ZPETC, IKF, causal FIR filter, and non-causal FIR filter will be described in the following subsections.

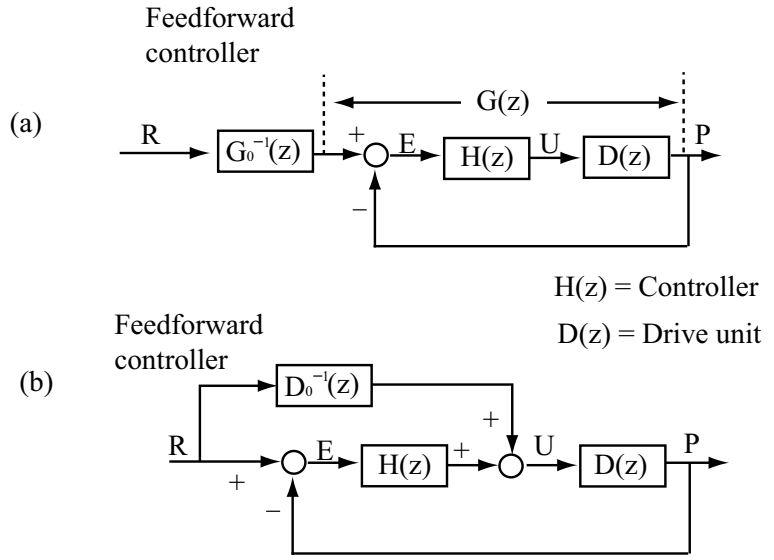


Fig. 5.10 Feedforward control structure

5.4.3.1 ZPETC, Zero Phase Error Tracking Control

Generally, the design of the feedforward controller is based on the transfer function of the entire system after designing the feedback controller. In considering Fig. 5.10a, 'R' denotes the desired path from the interpolator and 'E' denotes the error, the difference between the modified input from the feedforward controller and the output of process.

The purpose of the feedforward controller is to make $P(k)$ and $R(k)$ equal, or to minimize the difference between them if it is impossible to make $P(k) = R(k)$.

If the feedforward controller is designed as the inverse of $G(z)$ when $G(z)$ is a minimum-phase system where poles and zeroes are stable, the transfer function denoting the relationship between the input and the output results in 1, which means that the system traces the desired position correctly. However, the system is not always a minimum-phase system and some systems can have unstable zeroes. Furthermore, unstable zeroes can be produced when a continuous time system is transformed into a discrete time system, even if the system in the continuous time domain has no unstable zeroes.

The unstable zeroes are located in the left half of the Z-plane of the discrete time domain. In the case of a non-minimum phase system, if the feedforward controller is designed as the inverse of $G(z)$, it makes the system unstable because the feedforward controller itself can have an unstable pole and the output of the feedforward controller is unlimited. Therefore, for a non-minimum phase system, it is necessary to make the transfer function of the entire system close to 1 without making the system unstable. Since $G(z)$ is the transfer function including the feedback controller

and is designed to make the system stable, the poles of $G(z)$ are stable, in general. However the transfer function can include unstable zeros because the zero of $G(z)$ is not restricted. The unstable poles play a role in the poor performance of the system. Therefore, the feedforward controller is an effective way of removing the unstable zeroes. The typical algorithm for this is ZPETC (Zero Phase Error Tracking Control).

In ZPETC, the numerator term of the transfer function of a closed loop can be divided into terms including only the stable zeroes $B_c^s(z^{-1})$ and terms including only the unstable zeroes $B_c^u(z^{-1})$, as shown in Eq. 5.26.

$$B_c(z^{-1}) = B_c^s(z^{-1})B_c^u(z^{-1}) \tag{5.26}$$

Now, the ZPETC feedforward controller can be represented as follows by utilizing the special numerator term.

$$G^{-1}(z) = \frac{z^d A(z^{-1})B_c^u(z)}{B_c^s(z^{-1})(B_c^u(1))^2} \tag{5.27}$$

In consequence, the entire transfer function can be summarized as follows,

$$G_{zpetc}(z) = \frac{B_c^u(z)B_c^u(z^{-1})}{(B_c^u(1))^2} \tag{5.28}$$

We can understand that the phase difference of the transfer function from the desired path to output is zero because the transfer function results in the multiplication of two conjugating complex numbers. This means that the output traces the input with no time delay, the gain in the steady state is one and the error in the steady state becomes zero. When the transfer function of a closed loop has an the zeroes located on the left half of the plane, the gain increases. This means that ZPETC can have the gain error in the high-frequency range instead of making the phase difference zero. On the other hand, ZPETC is a good algorithm for building good characteristic by using small information in the low frequency range. The ZPETC requires an accurate system model, which is somewhat difficult, and shows poor tracing performance when the desired trajectory includes a fast transient shape such as at sharp corners. Furthermore, the inverse transfer function of the feedforward controller may require D/A converters or driving motors capable of handling high voltage, which is practically limited by the maximum output of D/A converter or the maximum voltage of the motor.

5.4.3.2 IKF, Inverse Compensation Filter

The IKF(Inverse Compensation Filter) was introduced by Weck to solve the corner-tracing problem of ZPETC. As shown in Fig. 5.11, a low-pass filter is added at the front of the feedforward controller to improve traceability. Consequently the smooth trace becomes possible by removing the high-frequency range from the input signal.

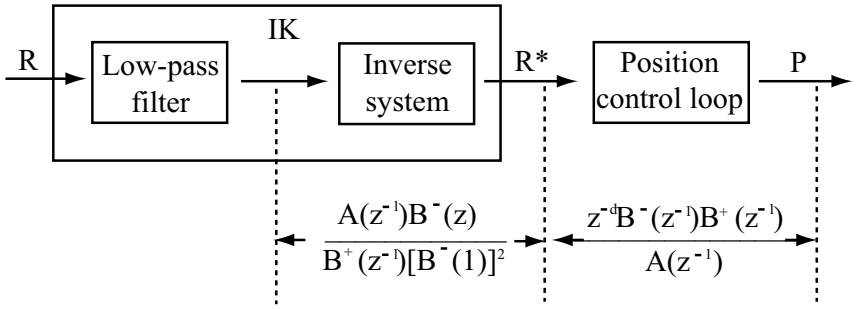


Fig. 5.11 IKF filter structure

5.4.3.3 Causal/Non-causal FIR

The controllers mentioned in the previous sections require an accurate system model and the transfer function is formulated using a complicated process. However, it is difficult to apply these controllers to a real CNC system. Therefore, a practical technique is typically used in industry, where the position command is forwarded to the feedback controller together with velocity and acceleration/deceleration commands, as shown in Fig. 5.12.

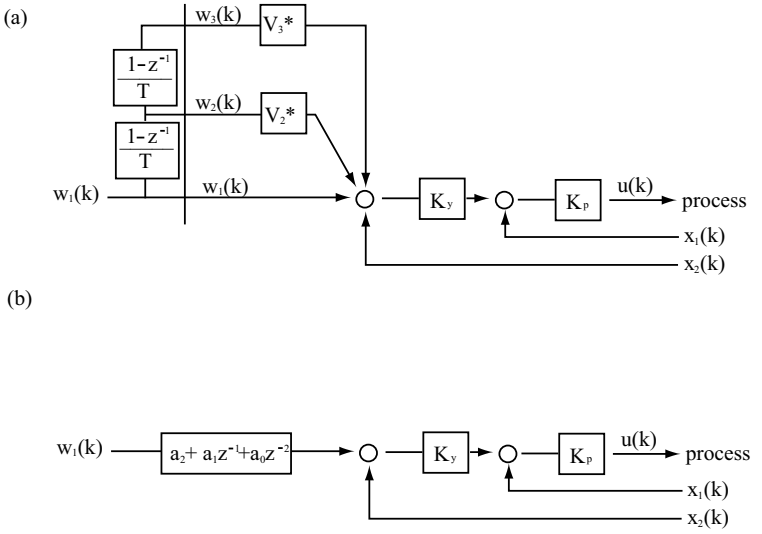


Fig. 5.12 Practical feedforward control

This type of controller can be represented mathematically in the two-step causal filter as Eq. 5.29.

$$G(z) = a_0 z^{-2} + a_1 z^{-1} + a_2 \tag{5.29}$$

$$\text{where, } a_0 = \frac{V_3}{T^2}, \quad a_1 = -\frac{V_2}{T} - \frac{2V_3}{T^2}, \quad a_2 = 1 + \frac{V_2}{T} + \frac{V_3}{T^2}$$

The above causal FIR filter uses the current position data and the position data from one and two steps before. It can also be represented in the non-causal FIR filter form that requests future position data.

$$G(z) = a_0 + a_1 z + a_2 z^2 \tag{5.30}$$

The architecture of the controller can be as shown in Fig. 5.13.

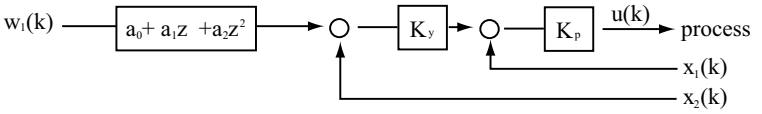


Fig. 5.13 Non-causal FIR

In a modern CNC system, the above non-causality, asking the future status value of CNC, can no longer be an obstacle. Since the CNC system typically stores the future position by NC block interpretation and position interpolation (*e.g.* linear/circular interpolation), the future position information at the current position can be obtained by reading the memory location. The time delay due to the degree of the filter and position sampling time may occur between the moment that an interpreter completes interpreting the NC block and the moment that actuation begins. However, this does not have any effect on the control performance and, actually, this control type is not worse than the other feedforward controller with respect to the contouring accuracy and the traceability at corners.

In the previous sections as well as in the current section, the feedforward controller that modifies the input signal and feeds the modified input to the feedback controller has been described. The following sections describe a controller where the input signal is fed directly to the feedback controller. A speed feedforward controller and a torque feedforward controller are typical of this kind of controller. This type of feedforward controller requires advanced control theories such as speed control and torque control of the servo system. In this book, only the basic architecture will be introduced. For other details, please read the reference on design of digital drive systems.

5.4.3.4 Speed Feedforward Controller

Figure 5.14 shows the architecture of a speed feedforward controller where the position command, α_w, is filtered and the filtered command is input directly to the speed control loop in a drive system in order to improve the response of the position control loop. K_v in Fig. 5.14 denotes the proportional gain of the position controller and only

proportional control is used for the position control of a CNC system. K_{pn} and K_{in} in Fig. 5.14 denote a propositional gain and an integral gain of the speed controller, respectively, and it is typical that a PI control is used for the speed control of a CNC system. α_w and α denote the desired position and the actual position respectively, and T is the sampling time of the position control loop.

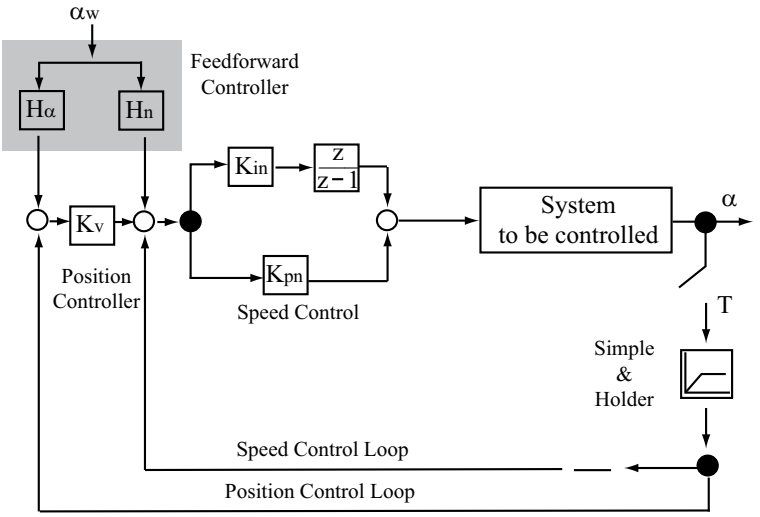


Fig. 5.14 Architecture of speed feedforward controller

5.4.3.5 Torque Feedforward Controller

Figure 5.15 shows the architecture of a torque feedforward controller. In the torque feedforward controller, a position command is directly input to the current loop (*e.g.* system to be controlled) that shows the fastest system response. Therefore, in the torque feedforward controller, the speed control loop including a current loop inside and position control loop should fulfill the input balance. Since the torque feedforward controller generates the position command that is directly input to the current loop, the speed command from an interpolator is used as an input to the feedforward controller for stable control. Because the torque feedforward controller actuates the motors by using the current loop with the fastest response, it generates the minimum following error under serial control architecture.

By using the feedforward controller for machine tools, the following error of the position control loop drastically decreases. The following error will be described in detail in the next section.

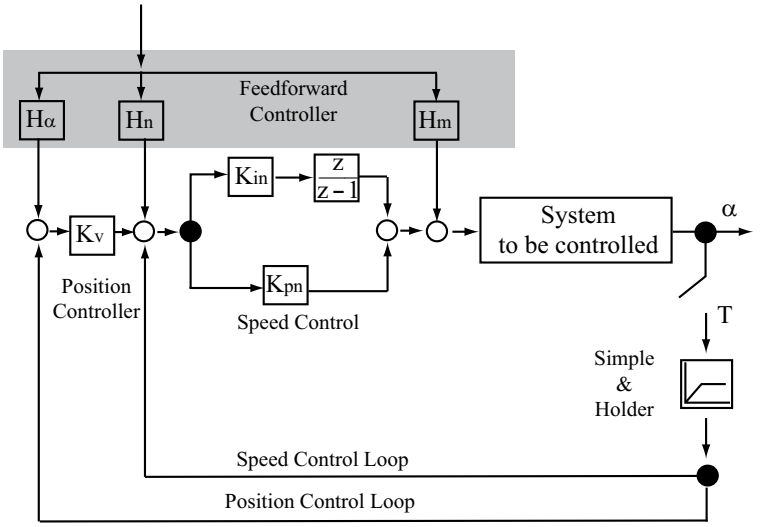


Fig. 5.15 Architecture of a torque feedforward controller

5.5 Analysis of the Following Error

The following error is defined as the difference between the desired position and the actual position of the CNC system. Since the position controller is typically a proportional controller, large proportional gain has an influence on the short setting time of machine tools. However, it has a limitation due to the steady-state error and overshoot.

5.5.1 The Following Error of the Feedback Controller

In a typical position control loop, a Unit Feedback System can be represented as in the block diagram shown in Fig. 5.16.

The transfer function of an open loop, $G(s)$, is generally represented as follows.

$$G(s) = \frac{K}{s^l} \cdot \frac{(1+sT_1')(1+sT_2')\ldots(1+sT_m')}{(1+sT_1)(1+sT_2)\ldots(1+sT_n)} \tag{5.31}$$

where, $l+n \geq m$ and K denotes the gain of the controller.

In this case, the position error is defined as the difference between the desired position and the output position.

$$E(s) = U(s) - Y(s) \tag{5.32}$$

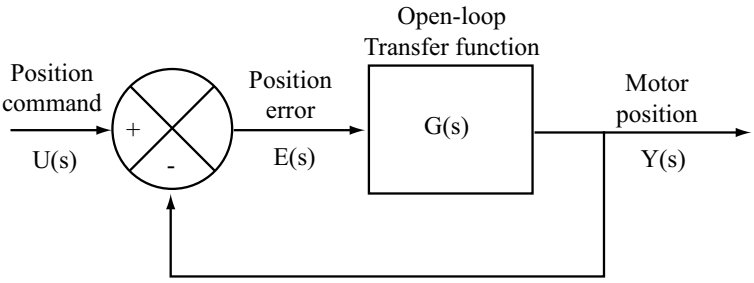


Fig. 5.16 Unit feedback controller

The transfer function of a closed loop, $W(s)$, can be written as follows,

$$W(s) = \frac{Y(s)}{U(s)} = \frac{G(s)}{1+G(s)} \tag{5.33}$$

From Eq. 5.32 and Eq. 5.33, $E(s)$ is summarized as follows,

$$E(s) + \frac{1}{1+G(s)}U(s) \tag{5.34}$$

where the steady state error is denoted by $e(\infty)$, being $e(t)$ as $t \to \infty$, and it can be calculated by using the Final Value Theorem for Laplace transformations,

$$e(\infty) = \lim_{t\to\infty} e(t) = \lim_{s\to 0} sE(s) \tag{5.35}$$

If we consider the steady-state error $e(\infty)$ when the input is a step input.

The steady state error e_p for the step input $u(t) = A$ or $U(s) = \frac{A}{s}$ is summarized as follows by using Eqs. 5.31, 5.34, and 5.35.

$$\begin{aligned} e_p &= \lim_{s\to 0} sE(s) \\ &= \lim_{s\to 0} s \cdot \frac{1}{1+\frac{K}{s^l}\cdot\frac{(1+sT_1')(1+sT_2')\ldots(1+sT_m')}{(1+sT_1)(1+sT_2)\ldots(1+sT_n)}} \cdot \frac{A}{s} \\ &= \lim_{s\to 0} s \cdot \frac{1}{1+\frac{K}{s^l}} \cdot \frac{A}{s} \\ &= \lim_{s\to 0} \frac{A}{1+\frac{K}{s^l}} \end{aligned} \tag{5.36}$$

Consequently, the steady state error is obtained from Eq. 5.36 as follows,

if $l = 0$ then $e_p = \frac{A}{1+K}$
if $l \geq 1$, then $e_p = 0$

wherein, $l = 1$ means that $G(s)$ or the controlled system includes the integral element $\frac{1}{s}$. If the number of the integral elements is more than one, e_p is equal to zero then the steady-state error does not occur.

Now find the normal error, $e(\infty)$, if the input is a Ramp input.

In the case of $u(t) = At$ or $U(s) = \frac{A}{s^2}$, the steady-state error e_p can be expressed as follows,

$$\begin{aligned} &= \lim_{s\to 0} s \cdot \frac{1}{1 + \frac{K}{s^l} \cdot \frac{(1+sT_1')(1+sT_2')\ldots(1+sT_m')}{(1+sT_1)(1+sT_2)\ldots(1+sT_n)}} \cdot \frac{A}{s^2} \\ &= \lim_{s\to 0} s \cdot \frac{1}{1 + \frac{K}{s^l}} \cdot \frac{A}{s^2} \\ &= \lim_{s\to 0} \frac{A}{1 + \frac{K}{s^{l-1}}} \end{aligned} \tag{5.37}$$

Finally,

if $l = 0$ then $e_p = \infty$.
if $l = 1$ then $e_p = \frac{A}{K}$.
if $l \geq 2$ then $e_p = 0$.

By utilizing the same method, if the steady state error e_p due to the acceleration input $u(t) = \frac{A}{2}t^2$ is considered the result is expressed as:

If $l = 0, 1$, then $e_p = \infty$.
If $l = 2$, then $e_p = \frac{A}{K}$.
If $l \geq 3$, then $e_p = 0$.

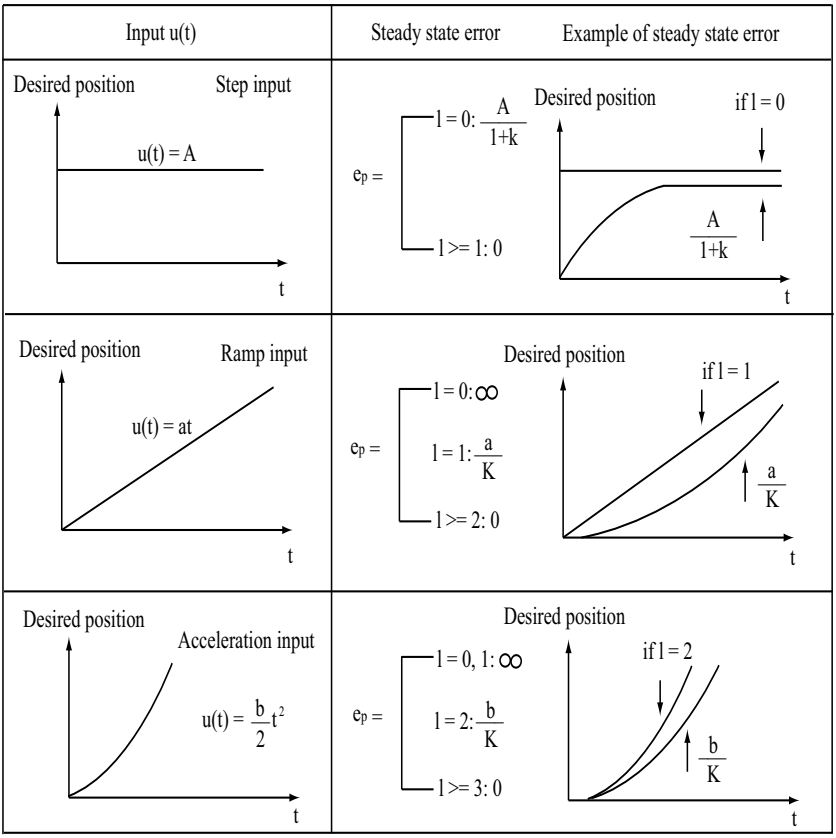
In this moment, the steady state errors of the feedback controller with respect to the inputs can be summarized as Table 5.2.

Consequently, if we assume that the input command is a function of time, $u(t) = Vt \times t$ (where, Vt is the commanded feedrate in the CNC system), the following error is typically as in Eq. 5.38.

$$e_{fbc} = \frac{V_t}{K_v} \tag{5.38}$$

Therefore, we conclude that the following error is inversely proportional to the gain of the position controller and is proportional to the feedrate when only feedback control is used. In consequence, it is necessary to increase the gain of the position controller or decrease the feedrate in order to decrease the following error.

Table 5.2 Normal error summary



5.5.2 The Following Error of the Feedforward Controller

If we summarize the following error of the speed feedforward without the complicated derivation process, it can be expressed in short form as Eq. 5.39.

$$e^{v}_{ffc} = V_t T_{en} \tag{5.39}$$

The following error of the torque feedforward controller can written as Eq. 5.40.

$$e^{t}_{ffc} = V_t(T_{ei} + T/2) \tag{5.40}$$

where V_t, T_{en}, T_{ei}, and T stand for the feedrate, the time constant of the speed control loop, the time constant of the current loop, and the sampling time, respectively.

Without the feedforward control we already know that the following error increases in proportion to the feedrate and is inversely proportional to the gain of the position controller, K_v. Now, when the speed feedforward control is applied, the fol-

lowing error is in proportion to the feedrate V_t and the time constant of the speed control loop, T_{en}. This means that K_v, the gain of the position controller does not have any effect on the following error and plays the role of regulator against external disturbance when the speed feedforward controller is used. In the case of the torque feedforward controller, the following error of the system is in proportion to the feedrate, time constant of the current loop and the sampling time of the position control loop.

5.5.3 Comparison of Following Errors

In the previous section, the main factors of the following errors of the feedback controller, the speed feedforward controller, and the torque feedforward controller were discussed. These main factors will be simulated by using actual machine tools, and the result of the simulation will be addressed in this section.

Figure 5.17a shows the commanded position data (the desired position) and actual position data after simulating without the feedforward controller. As shown in the figure, the actual position data follows the desired position with a time delay and the difference between them is called the following error. Figure 5.17b shows only the amount of the following error and we can investigate whether the following error converges to the value derived from Eq. 5.38. This following error is multiplied by K_v, the gain of the position controller, and the multiplied value is fed to the speed control loop.

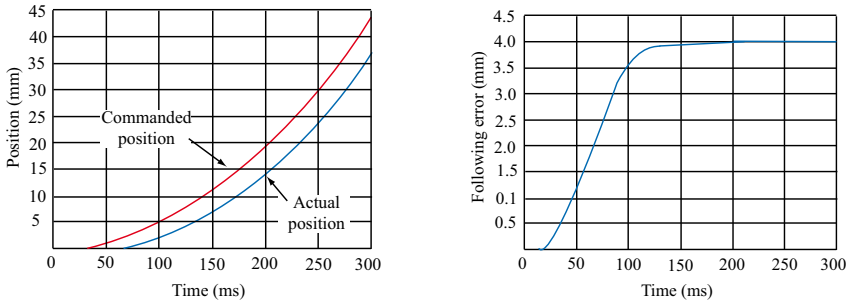


Fig. 5.17 Comparison of following errors

In the case of the speed feedforward controller, it is assumed that the gain of the position controller and feedrate are equal to the previous simulation, and the time constant of the speed control loop, T_{en}, is assumed to be 4 ms for the simulation.

When the speed feedforward controller is used, the divergence of the following error can be calculated from Eq. 5.39 as follows,

$$e^v_{ffc} = V_t T_{en} = 0.6667[\text{mm}]$$

Figure 5.18a shows the following errors when the speed feedforward controller is used and when the speed feedforward controller is not used. As shown in the figure, the following error decreases appreciably when the speed feedforward controller is used. We can also verify that the following error converges to the above calculated value when speed feedforward control is applied.

In the torque feedforward controller, the command data for actuating the axis is directly input to the current control loop. Because the current control loop has the fastest response, theoretically this architecture makes it possible to build a system with the smallest following error. When the torque feedforward controller is used, the convergence of the following error can be calculated from Eq. 5.40.

$$e^t_{ffc} = V_t(T_{ei} + T/2) = 0.771[\text{mm}]$$

Figure 5.18b shows the following errors when the speed feedforward controller is used and when the torque feedforward controller is used. For the simulation, the time constant of the speed control loop for the speed feedforward controller is specified as 4 ms and the time constant of the speed control loop for the torque feedforward controller is specified as 0.4 ms. As shown in the figure, it is possible to decrease the following error by using the torque feedforward controller.

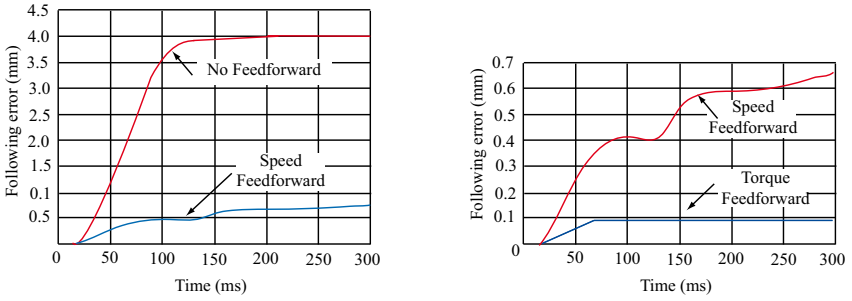


Fig. 5.18 Following errors when using speed feedforward

The following error of the position control system is not a critical problem in the case of a one-axis system. However, in the case of multi-axis systems, the following error is reflected in the accuracy of the machined shape. Therefore, it is very important to decrease the following error for machine accuracy.

As a summary, for the control of a CNC system, the PID controller is a typical feedback controller for decreasing the position error of each axis. The feedforward controller is used for decreasing the following error of the PID controller alone. The feedback controller and the feedforward controller are mutually combined to decrease the position error of each axis for multi-axis systems.

5.6 Summary

The quality of devices such as machine tools, robots, chip mounters, and semiconductor handlers depends highly on the performance of servo control. The position control, which actuates the moving unit (*e.g.*, slide, table, and servo) using the position data from the interpolator, is the last of the NCK tasks of a CNC system. The position controller not only controls accurately the position of the axes but also has to be robust against various disturbances. Motion control can be classified into three types with respect to the control purpose; the first is point-to-point control where fast and accurate arrival at the desired position is important regardless of the intermediate path. The second is following control where it is important for all axes to accurately follow the desired path. The third is contouring control where it is important to decrease the contouring error in the case of multi-axis systems. In the position controller, various algorithms have been used in order to meet the above conditions and the PID controller, feedforward controller, and cross-coupled controller are typical.

The PID controller is able to decrease the position error of an axis compared with a P controller. The feedforward controller is able to decrease the following error by decreasing the position error of each axis and consequently decreases the contouring error. The cross-coupled controller decreases the error by generating commands that can decrease the contouring error by using the contouring error model.

Chapter 6
Numerical Control Kernel

In this chapter, an NCK system is built by integrating the modules that were addressed in the previous chapters. Two kinds of NCK enabling the execution of Acceleration/Deceleration-After-Interpolation (ADCAI) and Acceleration/ Deceleration-Before-Interpolation (ADCBI) will be designed. The reader will acquire practical knowledge related to the implementation of the servo control system through investigation of the source code.

6.1 Introduction

The NCK (Numerical Control Kernel) is one of the units of which the CNC system is composed, the NCK is the unit for controlling the servo. The NCK, which consists of an interpreter, interpolator, acc/dec controller, and position controller, is the key unit not only of the CNC system for machine tools but is also a typical position controller where it is necessary to control servos. From Chapter 2 to Chapter 5, the detailed of the NCK components has been mentioned from the functional and structural viewpoints.

We will implement the NCK based on typical algorithms from among the various algorithms mentioned in the previous chapters. We will also mention execution algorithms and implementation by pseudo-code. In this chapter, only the design of NCK will be described and the overall design of the CNC system where the PLC and MMI units are included will be addressed in the following chapters in more detail.

6.2 Architecture of ACDAI-type NCK

ACDAI-type NCK consists of an Interpreter, Rough Interpolator, Acc/Dec Controller, Fine Interpolator, Position Controller, and ring buffers as shown in Fig. 6.1. The details of each of these modules has been mentioned in previous chapters.

Fig. 6.2 shows the data flow between the modules in ACDAI-type NCK. The data is transmitted between the modules via the ring buffer defined by global variables, and the ring buffers are located between Interpreter and Rough Interpolator, between Rough Interpolator and Acc/Dec Controller, and between Acc/Dec Controller and Fine Interpolator. Each ring buffer includes the data shown in Fig. 6.2. The Fine Interpolator and Position Controller use global variables to send the necessary data.

Details about implementation of the modules follows.

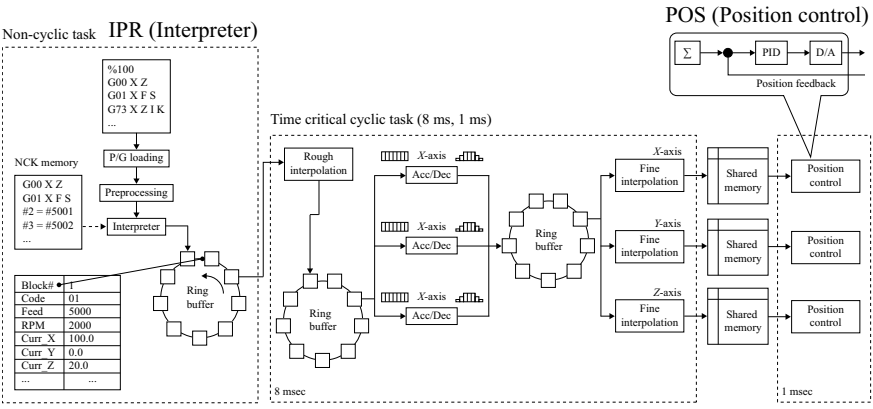


Fig. 6.1 Modules of the ACDAI-type NCK

6.2.1 Implementation of the Interpolator

As mentioned in Chapter 2, the input of the NCK is the part program. G-code, which is currently used for defining the part program, supports various functions such as tool compensation, coordinate transformation, cycle code, user-defined G-code, and sub program calls for convenience of editing part programs. To execute these functions and calculate the actual toolpath accurately, complicated computational tasks such as offsetting and coordinate transformations are required. The Interpreter has the task of computing the actual toolpath from the part program specified by G-codes or Macros.

6.2.1.1 The Structure of the Interpreter

Figure 6.3 shows the structure of an implemented interpreter. The Interpreter consists of a Compiler, G/M-code Interpreter, and Machine DB. The Compiler extracts the meaningful data from the part program based on the grammar of G/M-codes and

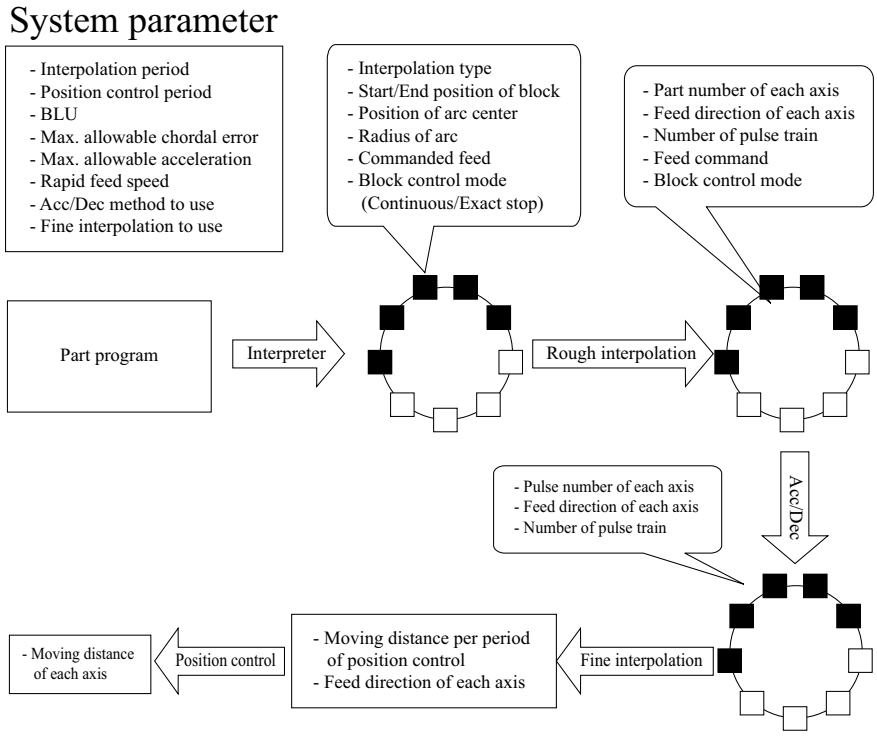


Fig. 6.2 Data flow between modules in the ACDAI-type NCK

calls the G/M-code Interpreter related to the extracted information. The G/M-code Interpreter called by the Compiler calculates the toolpath or transforms the coordinates according to the definition of the G-code and sends the computed result to the Compiler. The Machine DB stores the necessary data for compilation.

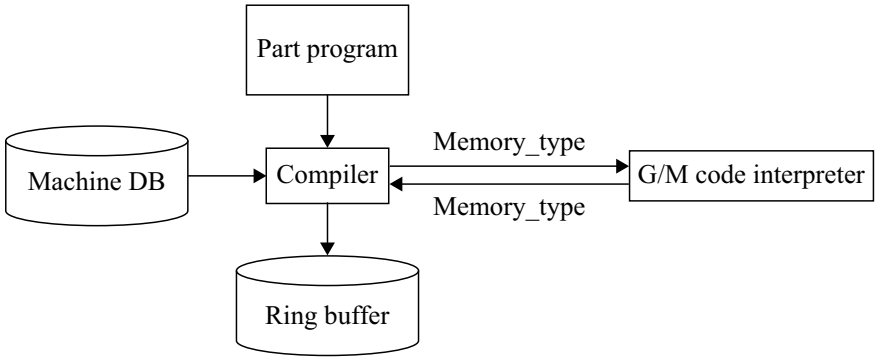


Fig. 6.3 Implemented interpreter structure

Figure 6.4 shows the execution procedure. The Interpreter carries out initialization, parsing the program blocks, G/M-code execution, and storing the interpretation result sequentially.

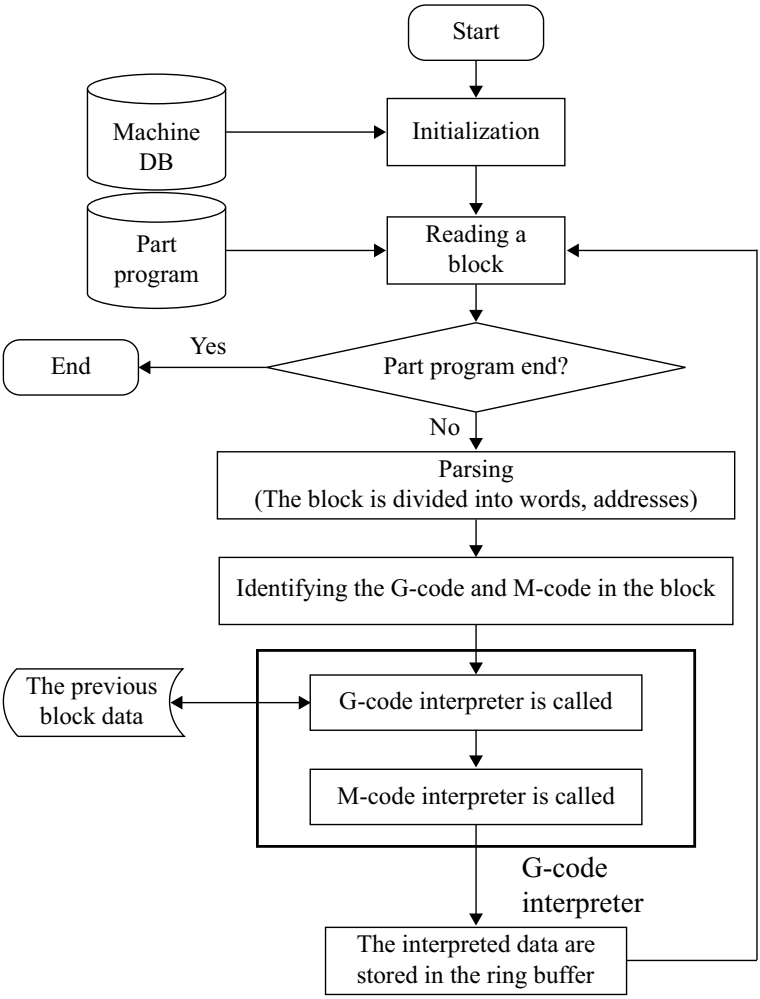


Fig. 6.4 Execution procedure for NCK

For the interpreter to execute correctly, tool offset data, workpiece coordinate data (G54, G55, G56, G57, G58, G59), and the programs connected with user-defined G-codes are essential and they are stored in the machine DB. During initialization, the interpreter sets the internal variables based on the information in the machine DB and receives the part program as input. Afterwards, to prepare the compilation, the interpreter replaces the blocks that call the sub program and user-defined G-code

with pre-defined programs. In addition, if there is an iteration loop, the interpreter repeatedly writes the loop blocks as many times as the specified value. Through these tasks, the part program is converted into a program where the program blocks are sequentially executed from beginning to end. The compiler parses the blocks sequentially based on the converted program. As the last task of initialization, the Interpreter checks the grammatical errors of the converted part program and decides whether the compiled program should be executed.

After initialization is complete, the compiler parses each block from the first block to the last sequentially. During parsing, the block is divided into words, addresses (see Table 2.1) and numerical data are extracted from the words and the extracted data are stored.

The following class "memory_type" is the implemented data structure to save the parsed data. The class saves not only the data that is needed to calculate the actual toolpath, such as G-code type, interpolation type, selected machining plane, tool-offset direction, tool-radius, coordinate system, rotation angle for rotational coordinate system, and tool-offset amount but also the spindle speed and feedrate that are applied while the block is being executed.

```
class memory_type {
  public :
  short code_type;                    // Interpolation(0),
                                      // Program num(1),
                                      // Subroutine call(2)
                                      // Custom macro(3),
                                      // Program end(4)
  short int_type;                     // Interpolation type
                                      // (01,02,03)
  short plane;                        // XY(17), ZX(18),YZ(19)
  short work_cood;                    // work_coordinate index
                                      // (54,55,56,57,58,59)
  short num_of_mid;                   // number of mid point when
                                      // offset is used
  short offset_end_type;              // Normal(0), Extended(1),
                                      // Shrink(2)
  short offset_dir                    // radius offset direction
                                      // Left(0), Right(1)
  BOOL rad_offset;                    // Offset applied(1), not (0)
  BOOL tool_moved                     // G00, G01, G01: 1,
                                      // Others: 0
  POINT3D pre;                        // real coordinate
  POINT3D cen;
  POINT3D mid[MAX_MID_POINT];
  POINT3D prev;                       // previous X,Y,Z code val.
  POINT3D curr;                       // current X,Y,Z code val.
  POINT3D prev_p;                     // previous I,J,K code val.
```

```
    POINT3D curr_p;                    // current I,J,K code val.
    double radius;                     // radius after radius offset;
    double original_radius;            // original radius
    double offset_radius;              // offset radius
    int feedrate;
    int spindle ;                      // spindle speed
    int tool_no;                       // Tool number used for
                                       // executing block
    int m_code;
    int g_gode[MAX_GROUP+1]            // Modal information
    int group_flag[MAX_GROUP_FLAG];
    int program_no;                    // program number
    int circular_mode;                 // in G02,G03 command
                                       // xyij(0) mode, xyR(1) mode
    POINT3D pair;                      // coordinate in G68
    double angle;                      // rotation angle in G68
    char content[80];
    int IsSubRoutine;
}
```

The block data that is saved in class memory_type is sent to the G/M-code interpreter and the G/M-code interpreter transforms the coordinate system (G15, G16, G54, G55, G56, G57, G58, G59,G68, G69, G92) and calculates the toolpath in the workpiece coordinate system (G00, G01, G02, G03, G40, G41, G42, G43, G44, G49) considering the previous block and the successor block data. Therefore, the G/M-code interpreter has computational functionality such as rotation, scaling or translation of the coordinate system and offsetting for tool radius compensation.

After transformation of the coordinate system and computation of the toolpath have been completed, the M-code interpreter is called. In general, M-codes are auxiliary functions for controlling machines and are used for commanding PLC-related tasks as mentioned in Table 2.1. So, the M-code interpreter has the task of filling out the tool number, rotation direction of the spindle, and spindle speed in class memory_type.

After interpretation of a particular block is finished, using the procedure mentioned above, the interpreted data, such as tool start position, tool end position, center position for circular interpolation, and compensation type are stored in the ring buffer and the stored data are used by the Rough Interpolator or Look-Ahead algorithm.

6.2.1.2 Interpreter Input and Output

In this section, the input and output data are addressed and the implemented data structure to store the output of the interpreter is as follows:

Input to interpreter
A part program is input to the Interpreter and the composition of the part program was mentioned in Section 2.2 in more detail.

Output from the interpreter
The interpreter finally generates and saves the actual start and end positions of the tool for each block in the workpiece coordinate system. If the block specifies an arc movement, the interpreter generates and saves the center position and radius of the arc. The interpolation type and control mode that are required for execution of the Interpolator and Acc/Dec Controller are stored. The output of the Interpolator is used as input to the Rough Interpolator in the case of ADCAI-type NCK and as input to the Look-Ahead algorithm. The following is an implementation example of the output of the Interpreter.

```
class CRingIR : public CObject { public :
    int nGCode;         // G-code type (0: G00, 1: G01, 2: G02)
    CVector Start;      // Start position of block (mm)
    CVector End;        // End position of block (mm)
    CVector Cen;        // Center point of arc (mm)
    double dRadius;     // Radius of arc (mm)
    double dFeed;       // Feedrate (mm/min)
    int nStatus;        // Block status (0: start, 1: end)
    int nStopMode;      // Path control mode. (1: Exact Stop,
                        // 0: otherwise)
    int nBlockNumber; // Block number.
};
```

6.2.2 Implementation of the Rough Interpolator

The Rough Interpolator carries out linear interpolation and circular interpolation depending on the G-code type from the interpolator. In Chapter 3, the algorithms for linear interpolation and circular interpolation were described in detail and we implemented the circular interpolator based on the Improved Tustin algorithm, which is better than other algorithms in terms of speed and accuracy.

6.2.2.1 Linear Interpolator

The linear interpolator interpolates tool movement specified by G00 and G01. The same algorithm is applied to G00 and G01 except that G00 uses the feedrate specified in a CNC system and G01 uses the programmed feedrate. The key function of the linear interpolator is calculating the displacement of the tool within the iteration time of interpolation and is executed as follows:

At first, the feedrate is selected depending on the G-code type and the length of the tool movement and the displacement of each axis is computed based on the start and end points of the block. Based on the selected feedrate, the actual time spent to move the axes is calculated. Based on the axis displacement and the movement time, the displacement of all axes is computed every iteration time of the interpolation. Therefore, the movement time must be a multiple of the iteration time of interpolation and the displacement of each axis must be represented by pulse units (one pulse means 1 BLU). Figure 6.5 shows the procedure for executing linear interpolation for the *X*-axis and the same procedure is applied for the other axes.

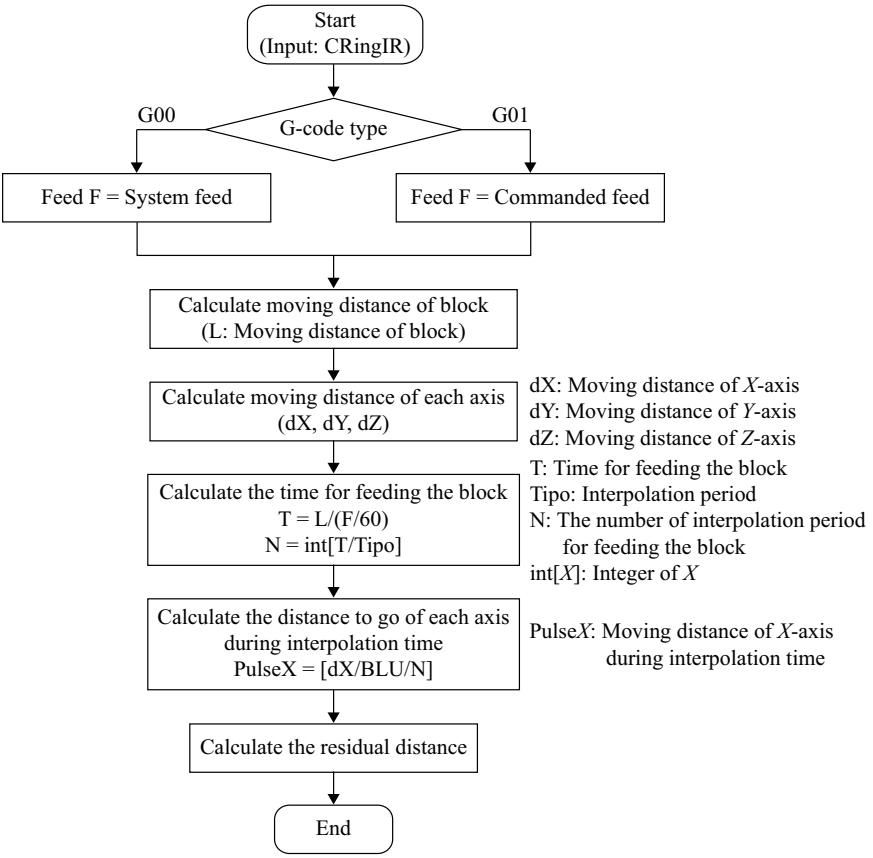


Fig. 6.5 Procedure for executing linear interpolation for *X*-axis

For practical implementation of the interpolator, however, there is something to be considered besides the above-mentioned procedure. In general, the moving length of an axis is not an exact multiple of pulses. In this case, numerical error can be accumulated because of the significant figures of numerical computation on a com-

puter and this accumulated numerical error causes reduction of the accuracy of the interpolator.

Therefore, it is essential to prevent numerical error from accumulating for actual implementation of an interpolator. For this, the length that each axis has to move within the interpolation time is necessarily a multiple of pulses. However, residual pulses still remain after assigning pulses evenly for each interpolation time. For example, if an axis should be moved with 745 pulses during 10 interpolation time periods, the axis has to move 74 pulses every iteration time of interpolation and 5 pulses still remain.

There are various methods to handle the remaining pulses. In this chapter, two practical methods will be introduced.

The first method is to move the axis with the remaining pulses during an additional iteration interpolation time. For example, if this method is applied to the above example, 745 pulses are distributed in 11 iteration interpolation times. The remaining 5 pulses are added to the last (eleventh) iteration after ten iteration interpolation times, each of 74 pulses. In this method, the difference of the number of pulses between the eleventh iteration time of interpolation and other iteration interpolation times can be large and, consequently, results in a drastic feedrate change.

The second method is to allocate the remaining pulses to each iteration interpolation time. If the second method is applied to the first method's example, the five remaining pulses are equally distributed to the first five iteration interpolation times. This means that the axis is moved with 75 pulses every iteration interpolation time from the first to the fifth. From the fifth to the tenth, the axis is moved with 74 pulses every iteration interpolation time. Because one pulse is practically a very small length, the change of feedrate is slight between the iteration interpolation times. Therefore, we used the second method for implementation.

There is no one correct method to handle the remaining pulses and the developer has to decide on an adequate method depending on the application environment.

6.2.2.2 Circular Interpolator

A circular interpolator approximates an arc by a set of line segments within the specified chordal error. As mentioned in Chapter 3, the greater the number of line segments, the better the accuracy of the interpolation. However, the time spent interpolating increases as the number of line segment increases. Actually, the performance of a circular interpolator depends on the minimization of the number of approximating line segments while fulfilling the chordal error criterion. The Improved Tustin algorithm, which was introduced in Section 3.3.2.8, satisfies the above condition and, therefore, the Improved Tustin algorithm was used to implement the circular interpolator.

Figure 6.6 shows the flowchart of the implemented circular interpolator. The implemented circular interpolator was divided into two parts. In the first part an arc is approximated into the line segments and, in the second part, the divided line segment is itself subdivided into the micro line segments to go along during each interpola-

tion period. The details of each part are as follows. In the first part, α the length of the approximating line segment and δ, the angle between the start point and the end point of arc, are computed by the Improved Tustin algorithm. Strictly speaking, α is the angle between the start point and the end point of the approximated line segment, as shown in Fig. 3.24.

Furthermore, the number of line segments is calculated by dividing δ from the Improved Tustin algorithm by α, where δ may not be an exact multiple of α. In this case, modification of α is needed in order for δ to be a multiple of α. This task is necessary to make the length of the line segments equal and, in consequence, to make a uniform feed. To perform this task, we implemented the following method.

Because α is proportional to the chordal error, α has to be decreased so that the distance between any point on the line segment and the circle does not exceed the specified chordal error. Consequently, the best way is to decrease the number of line segments by one and to compute a new α by dividing δ by the decreased number of line segments. Finally, with the newly computed α, the start point of the arc and the center point of the arc, it is possible to obtain the final start and end points of the line segments.

In the second part, the line segments from the first part are divided into small line segments based on the feed and the iteration time of interpolation. Dividing the principle line segments into small line segments is done in the same way as interpolating the linear profile (line block) in the linear interpolator. The line segments into which an arc is approximated are equal to the linear profile (line block) and the small line segments determine the displacement of the axis movement within each interpolation iteration time.

6.2.2.3 Input and Output of the Interpolator

In this section, the implemented data structure to store the input and output of Rough Interpolator is addressed.

Rough Input

As input to the Rough Interpolator, the G-Code type, the start point and end point of block, feedrate, path control mode (e.g, Exact Stop Mode and Continuous Mode), and the center point and radius of an arc have been implemented. The following is an implementation example of the input to the Rough Interpolator.

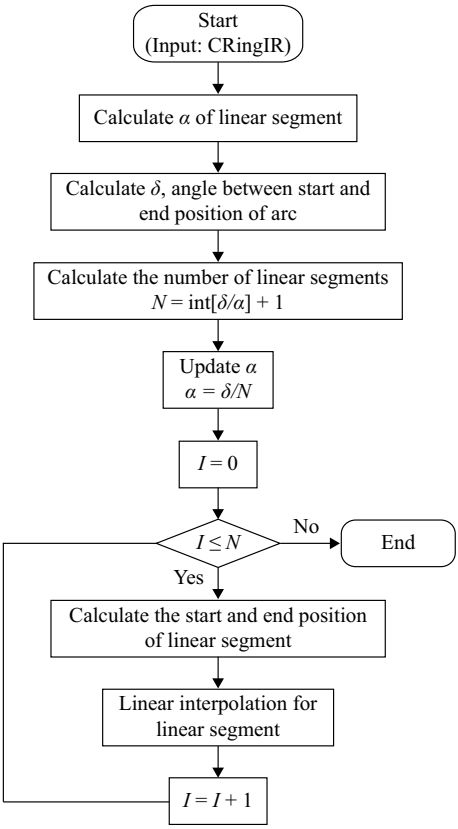


Fig. 6.6 Flowchart of the implemented circular interpolator

```
class CRingIR : public CObject { public :
    int nGCode;          // G-code type (0: G00, 1: G01, 2: G02)
    CVector Start;       // Start position of block (mm)
    CVector End;         // End position of block (mm)
    CVector Cen;         // Center point of arc (mm)
    double dRadius;      // Radius of arc (mm)
    double dFeed;        // Feedrate (mm/min)
    int nStatus;         // Block status (0: start, 1: end)
    int nStopMode;       // Path control mode. (1: Exact Stop,
                         // 0: otherwise)
    int nBlockNumber; // Block number.
};
```

Rough Output

The output from the interpolator consists of the displacement of each axis within the interpolation iteration time (in pulse units) and the number of interpolation sampling times that are required to carry out the block. In addition, the feedrate that is applied for Acc/Dec control and the path control mode are stored as the output. The following is the implemented data structure to store the output from the Rough Interpolator.

```
class CRingRA : public CObject { public :
    double dFeed;   // Feedrate (mm/min)
    int nStopMode; // Path control mode (1: exact stop,
                   // 0: otherwise)
    CVector* P;    // Distance to move per interpolation
                   // cycle in terms of number of pulses.
    int nStatus;   // Block status (0: start, 1: end)
    int N;         // Number of repetitions for interpolation
                   // in executing block.
};
```

6.2.2.4 Functions for the Rough Interpolator

ARoughInterpolation()

This is the main function of the Rough Interpolator. The behavior of this function follows the flowchart shown in Fig. 6.7. This function gets the block data from the ring buffer that saves the result from the Interpreter. It also carries out linear interpolation and circular interpolation depending on the G-code type specified in the block.

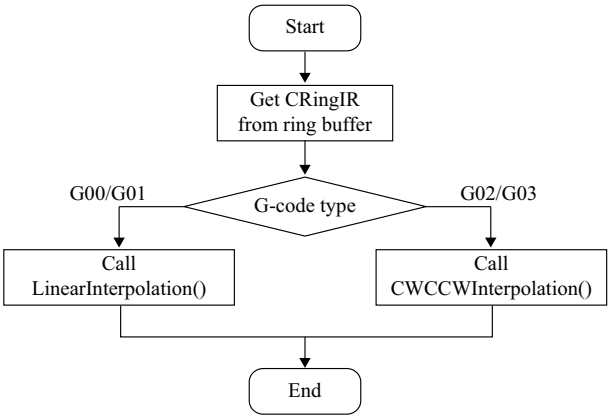


Fig. 6.7 Rough interpolation

LinearInterpolation()
This function executes the linear interpolation and realizes the flowchart shown in Fig. 6.4. Based on the input mentioned in Section 6.2.2.3, this function computes the displacement of each axis every interpolation iteration time (series of pulses) and stores the result in the ring buffer.

CWCCWInterpolation()
This function executes circular interpolation and realizes the flowchart shown in Fig. 6.6. Based on the input mentioned in Section 6.2.2.3, this function computes the displacement (pulses) of each axis every interpolation iteration time and stores the result in the ring buffer.

6.2.3 Implementation of an Acc/Dec Controller

If the acceleration performance of a servo motor is not enough when an axis just starts or when the difference of the velocity between two successive blocks is large, it is difficult to reach the programmed feedrate within the interpolation iteration time when the feedrate is changed and, therefore, it is impossible for an axis to move with the programmed displacement.

Actually, the interpolator does not consider the acceleration performance of the servo motor when calculating the displacement of each axis every interpolation iteration time (meaning the pulse profile), only the geometry of a block is considered.

The Acc/Dec Controller smoothes the change of velocity, meaning the number of pulses, at the beginning and the end of a block. The details of acceleration and deceleration algorithms were addressed in Chapter 4. Linear Acc/Dec, Exponential Acc/Dec, and S-Shape Acc/Dec algorithms, mentioned in Section 4.2.1, were implemented by transforming Eq.s 4.7, 4.8, and 4.9 into software functions.

Moreover, the CNC system provides Exact Stop Mode and Continuous Mode as path control modes. The algorithms for the two path control modes were described in Section 4.2.4. Exact Stop Mode was implemented by applying Acc/Dec control to individual blocks and executing the next block after completion of execution of the current block motion. As mentioned in Section 4.2.4, Continuous Mode implements Acc/Dec control continuously to each of the following blocks. The concept of Exact Stop Mode and Continuous Mode are shown in Fig. 6.8.

6.2.3.1 Input and Output of the Acc/Dec Controller

Input
The Acc/Dec Controller receives the output from the interpolator as input via the ring buffer. The displacement of each axis every interpolation iteration time (pulse profile), an iteration number of interpolation steps for the block, the feedrate, and the path control mode are input to the Acc/Dec Controller.

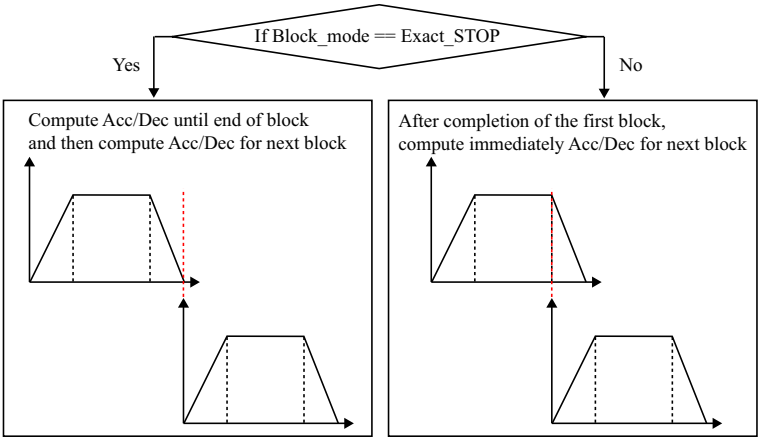


Fig. 6.8 Exact Stop Mode and Continuous Mode

```
class CRingRA : public CObject { public :
    double dFeed;   // Feedrate (mm/min)
    int nStopMode; // Path control mode (1: exact stop,
                   // 0: otherwise)
    CVector* P;    // Distance to move per interpolation
                   // cycle in terms of number of pulses.
    int N;         // Number of repetitions for interpolation
                   // in executing block.
};
```

Output

The Acc/Dec Controller outputs the displacement of each axis every interpolation iteration time (pulse profile) and the interpolation iteration time for each block.

```
class CRingAF : public CObject { public:
    CVector* P; // Distance to move per interpolation
                // cycle in terms of number of pulses.
    int N;      // Number of repetitions for interpolation
                // in executing block.
}
```

6.2.3.2 Functions for Acc/Dec Controller ACC_DEC()

This is the main function of the Acc/Dec Controller and calls particular functions depending the pre-specified algorithms to generate the Acc/Dec profile. Figure 6.9 shows the flowchart of this function.

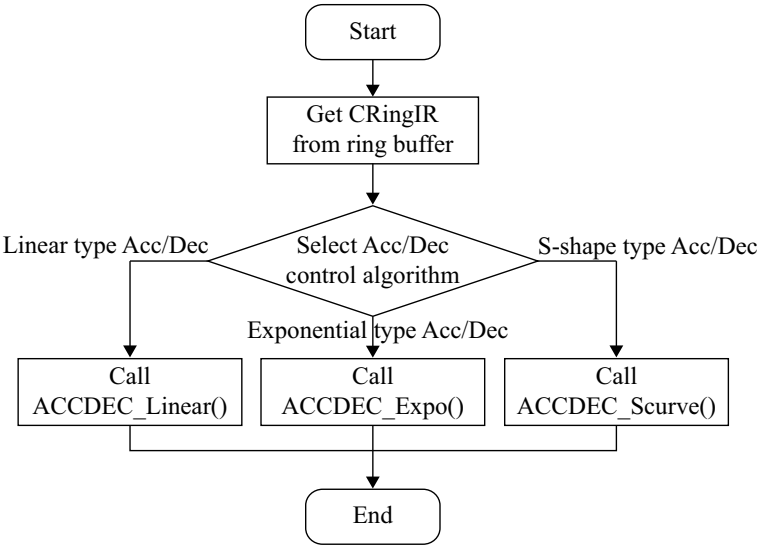


Fig. 6.9 Flowchart for generating pre-specified functions

ACCDEC_Linear()
This is the implementation of the Linear-type Acc/Dec Control algorithm. This function calls lcal() to generate the Acc/Dec pulse profile for each block and generates the final pulse profile by using the following two functions depending on the pre-specified path control mode.

ACCDEC_Linear_ES()
This function carries out Linear-type Acc/Dec control in Exact Stop mode. The implementation method of Exact Stop mode was described in detail in Section 4.2.4.

ACCDEC_Linear_BO()
This function carries out Linear-type Acc/Dec control in Continuous mode. In detail, the implementation method of Continuous mode was mentioned in Section 4.2.4.

ACCDEC_Scurve()
This is the implementation of the S-Shape Type Acc/Dec Control algorithm. This function calls scal() to generate an Acc/Dec pulse profile of each block and generates the final pulse profile by using the following two functions depending on the pre-specified path control mode.

ACCDEC_Scurve_ES()
This function carries out S-Shape-type Acc/Dec control in Exact Stop mode. The implementation method of the Exact Stop mode was described in detail in Section 4.2.4.

ACCDEC_Scurve_BO()
This function carries out S-Shape-type Acc/Dec control in Continuous mode. The implementation method of Continuous mode was described in detail in Section 4.2.4.

ACCDEC_Expo()
This is the implementation of the Exponential-type Acc/Dec Control algorithm. This function calls ecal() to generate the Acc/Dec pulse profile of each block and generates the final pulse profile by using the following two functions depending on the pre-specified path control mode.

ACCDEC_Expo_ES()
This function carries out Exponential-type Acc/Dec control in Exact Stop mode. The implementation method of Exact Stop mode was described in detail in Section 4.2.4

ACCDEC_Expo_BO()
This function carries out Exponential Type Acc/Dec control in Continuous mode. The implementation method of Continuous mode was described in detail in Section 4.2.4.

lcal()
This function generates the pulse profile of a block by applying Linear-type Acc/Dec control algorithm. This function is the realization of Eq. 4.7 in Section 4.2.1.

ecal()
This function generates the pulse profile of a block by applying the Exponential-type Acc/Dec control algorithm. This function is the realization of Eq. 4.8 in Section 4.2.1.

scal()
This function generates the pulse profile of a block by applying the S-Shape-type Acc/Dec control algorithm. This function is the realization of Eq. 4.9 in Section 4.2.1.

6.2.3.3 Implementation of the Acc/Dec Controller

To verify the implemented Acc/Dec Controller, we applied the part program in Fig. 6.10 to the implemented Acc/Dec Controller. Figure 6.10 shows the acceleration and deceleration results for the X-axis during continuous cutting mode and Fig. 6.11 shows the results for the X-axis during Exact Stop mode. According to Fig. 6.10 and Fig. 6.11, we can see that the drastic change in the number of pulses shown in the output from the Rough Interpolator is smoothed through the Acc/Dec Controller.

Moreover, we can see the different acceleration and deceleration profiles depending on the path control mode from Fig. 6.10 and Fig. 6.11. Strictly speaking, in Exact Stop Mode, the speed at the corner point where two successive blocks meet becomes 0 and in continuous cutting mode the speed at the corner is not reduced to zero. From

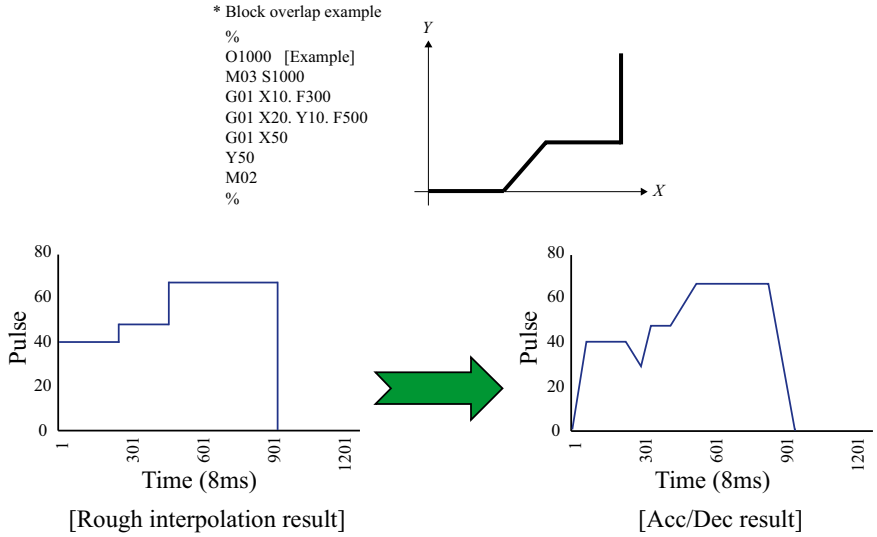


Fig. 6.10 Continuous cutting mode

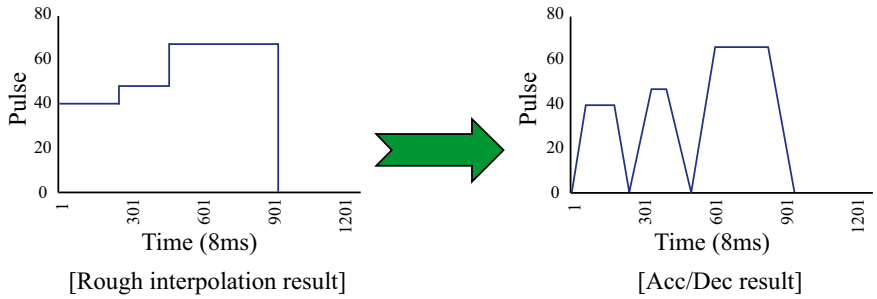


Fig. 6.11 Exact Stop mode

the above test, we can verify that the implemented Acc/Dec Controller reflects the two path control modes well.

6.2.4 Implementation of Fine Interpolator

As mentioned in Section 6.2, due to limitations in control accuracy and hardware performance, the interpolation iteration time and position control sampling time were implemented differently. Therefore, the displacement, which axis moves within the interpolation iteration time, from the Acc/Dec Controller should be divided into the displacements through which an axis moves within the position control sampling time.

Consequently, the Fine Interpolator divides the displacement of an axis within the interpolation iteration time into the displacement of the axes within the position control sampling time. In this book, we introduce the moving-average method and the linear method and source code for the two implemented methods. The details of the two algorithms have been given in Section 3.4.

6.2.4.1 Input and Output of the Fine Interpolator

Fine Input
The Fine Interpolator receives the output of the Acc/Dec controller as input via the ring buffer. The input for the Fine Interpolator is the length through which an axis moves every interpolation iteration time. The implemented data structure of the ring buffer is as follows.

```
class CRingAF : public CObject { public:
   CVector* P; // Distance to move per interpolation
               // cycle in terms of number of pulses)
   int N;      // Number of repetitions for interpolation
               // in executing block.
}
```

Output
The output of the Fine Interpolator is the length through which each axis moves every position control sampling time. Instead of a ring buffer, a global variable, shown below, is used to transmit the data between the Fine Interpolator and the Position Controller. The transmitted data consists of the displacement and direction of the axis movement.

```
CVector P[16]; // Distance to move per interpolation cycle in terms
               // of number of pulses.
```

6.2.4.2 Functions for the Fine Interpolator

FIPO()
This is the main function of the Fine Interpolator and carries out uniform fine interpolation by calling FIPO_Linear() and moving average fine interpolation by calling MovingAverage(). The type of fine interpolation should be specified by the user.

FIPO_Linear()
This function carries out linear fine interpolation. The linear method means distributing the displacement of each axis uniformly within the interpolation iteration time over the iteration times for position control. Figure 6.13 shows the flowchart for lin-

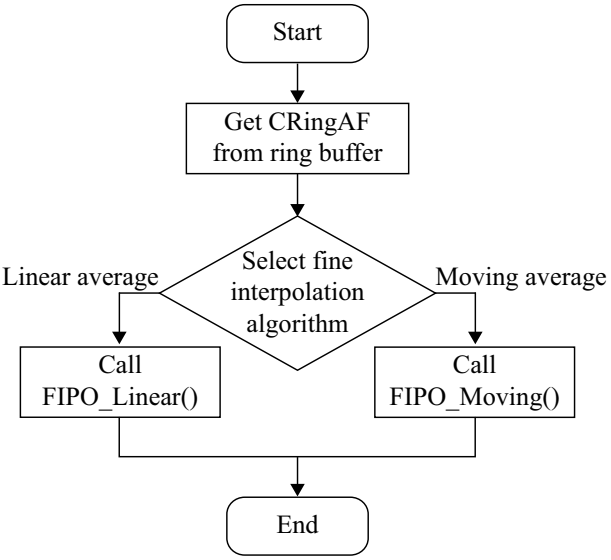


Fig. 6.12 Fine interpolation flowchart

ear fine interpolation and can be applied identically for other axes. This function is the realization of the procedure of Fig. 6.13.

FIPO_Moving()
This function carries out moving-average fine interpolation. In the moving-average method, firstly the number of pulses for every iteration time of position control is calculated by applying the linear fine interpolation to a block. Secondly, the moving average is applied to the pulse profile. Figure 6.14 shows the flowchart for moving-average interpolation and can be applied identically to other axes. This function is the realization of the procedure of Fig. 6.14.

6.2.4.3 Verification of the Fine Interpolator

To verify the implemented Fine Interpolator, we applied the part program in Fig. 6.15 to the implemented Fine Interpolator. The example part program consists of three linear paths that include a diagonal path, a perpendicular path, and a horizontal path.

Figure 6.16 shows the results of fine interpolation for the above example. The overall profiles of moving-average fine interpolation and linear fine interpolation appear to be equal. However, if the box-shaped areas are enlarged, we can see the difference between the two fine interpolation methods.

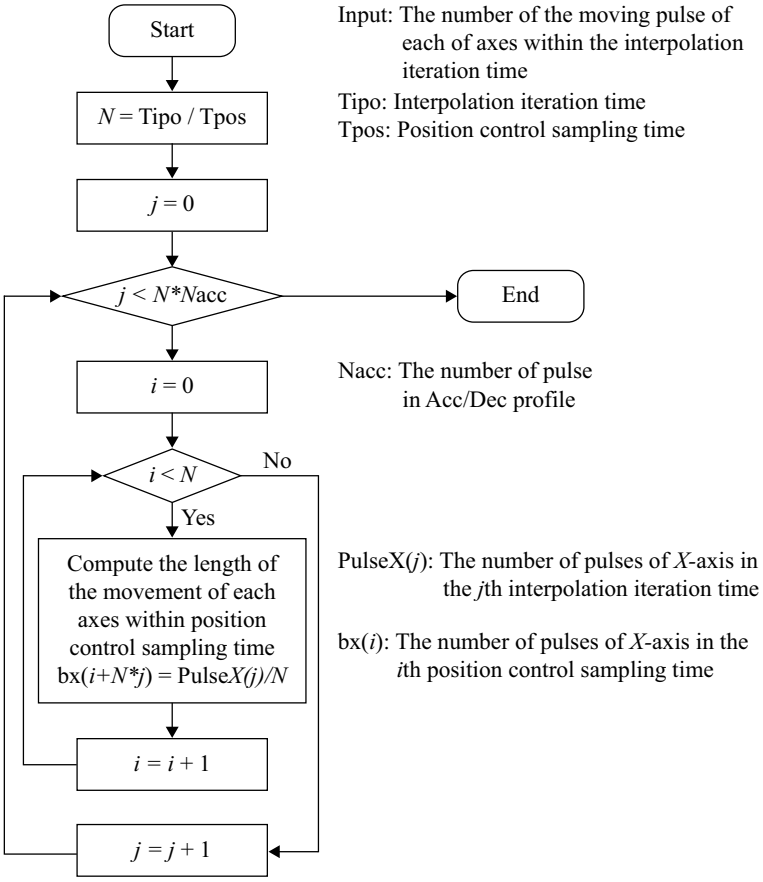


Fig. 6.13 Flowchart for linear fine interpolation

Figure 6.17 shows the enlargement of box A in Fig. 6.16. The implemented interpolation iteration time is 8 ms and the iteration time of position control is 1 ms. Figure 6.17a shows the input (pulse profile) to the Fine Interpolator. The implemented moving-average fine interpolation method generates the output (pulse profile) shown in Fig. 6.17b and the implemented linear fine interpolation method generates the output (pulse profile) shown in Fig. 6.17c.

Figure 6.18 shows the enlargement of box B in Fig. 6.16. Figure 6.18a shows the input to the Fine Interpolator, which was generated by the Acc/Dec Controller. Figures 6.18b and 6.18c show the results of the moving-average method and the linear method, respectively.

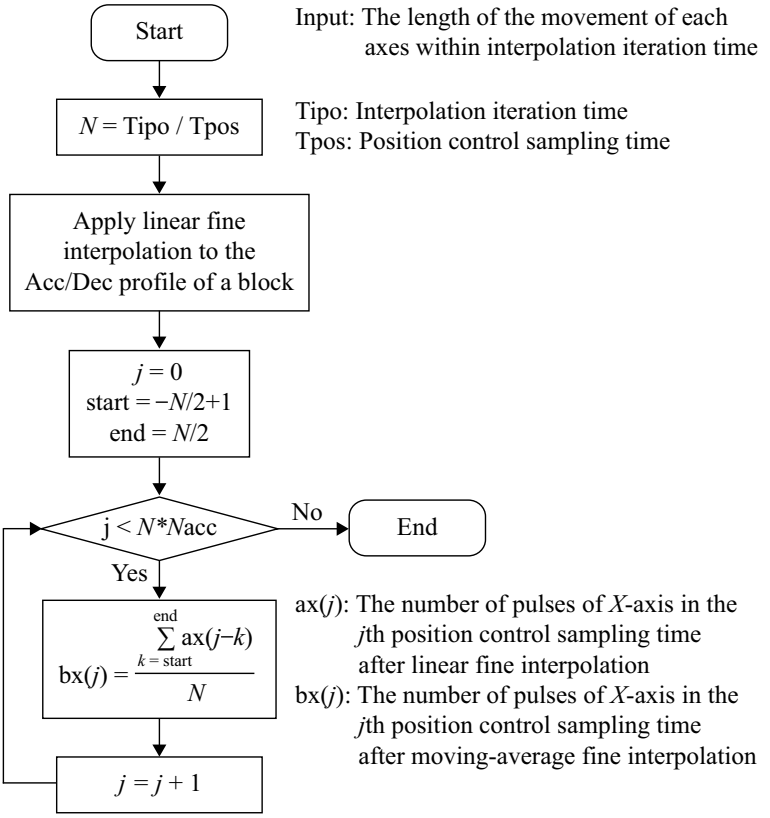


Fig. 6.14 Flowchart for moving-average interpolation

```
%
O1000 [Example]
M03 S1000
G01 X20. Y10. F500
G01 X30.
G01 Y20.
M02
```

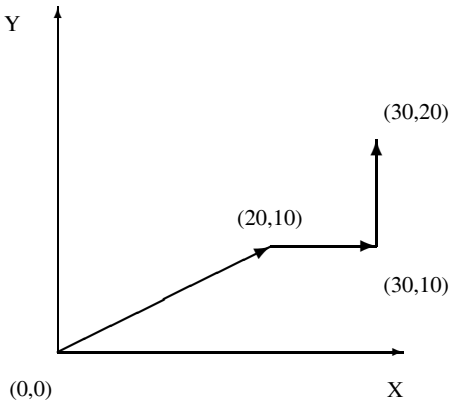


Fig. 6.15 Part program for fine interpolation verification

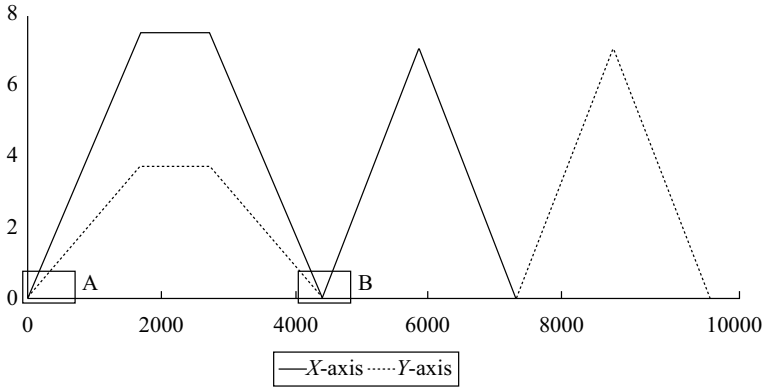


Fig. 6.16 Fine interpolation

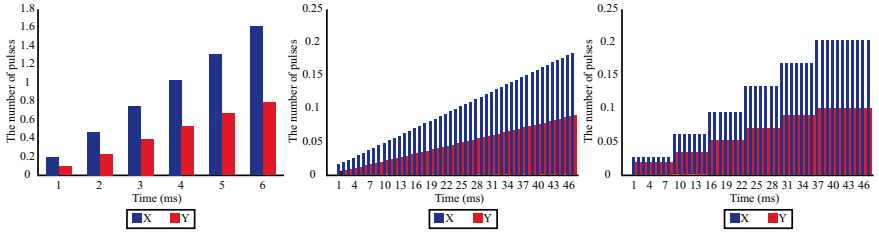


Fig. 6.17 Enlargement of box A from Fig. 6.16

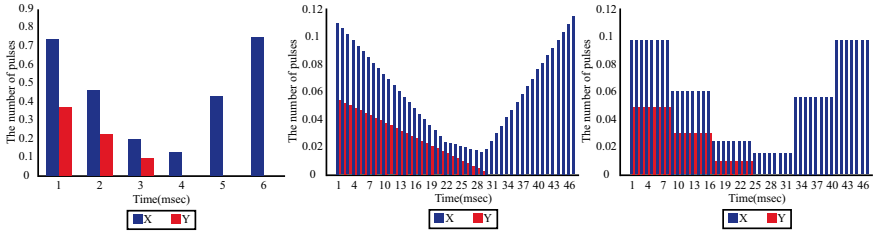


Fig. 6.18 Enlargement of box B from Fig. 6.16

6.2.5 Implementation of the Position Controller

The Position Controller is the module that compares the actual position of each axis obtained from an encoder with the commanded position and generates an analog signal (voltage signal) to be sent to each servo drive based on the comparison result. It is iteratively executed every interpolation time of position control. In Section 5.4, the PID control algorithm, the feedforward algorithm, and the cross-coupling algorithm

were addressed. From these, the PID control algorithm described in Section 5.4.1 was implemented for realizing the Position Controller.

6.2.5.1 Input and Output of the Position Controller

Input
The displacement (pulses) through which an axis should move within the interpolation time of position control is input to the Position Controller. The actual position data of the axes, another input to the Position Controller, are obtained from an encoder.

```
CVector P;      // Distance to move per interpolation
                  // cycle in terms of number of pulses.
CDirection Dir; // Moving direction of each axis.
```

Output
The Position Controller outputs the voltage signal that is sent to the servo drive system. The voltage signal is converted to a current signal by a servo drive system and, finally, the rotation speed of the servo motor is decided by the current signal.

6.2.5.2 Functions for the Position Controller

POS()
This is the function that realizes the pseudo-code of the PID_Controller described in Section 5.4.1. In this function, first the actual position data is taken from encoders and the position error is computed by comparing the actual position from the commanded position. The computed position error is used as the input to the PID control algorithm together with PID gains. The implemented PID algorithm generates the control signal, meaning the velocity of the axes during the iteration time of the position control and this control signal is then converted into a voltage signal that is transmitted to the servo drive.

6.2.5.3 Verification of the Position Controller

To verify the implemented Position Controller, we assume a virtual servo motor and apply two PID gains for this virtual servo motor. In Test 1, the P gain, I gain, and D gain were set to 2.0, 0.0001, and 0.0 respectively. In Test 2, the P gain, I gain, and D gain were set to 1.0, 0.0001, 0.0, respectively. For the simulation, the part program shown in Fig. 6.19 was used.

```
%
O1000
M03 S1000
G01 X10. Y10. F200
G01 Y20.
X20.
M02
%
```

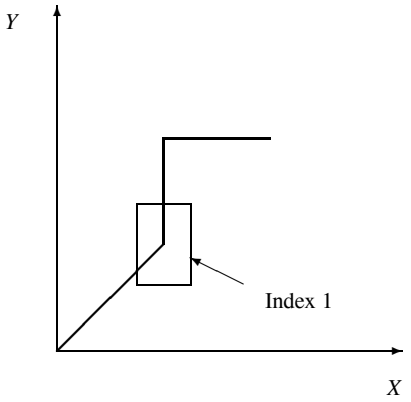


Fig. 6.19 Part program for simulation

The tool trace shown in Fig. 6.19 shows the two simulation results. Because of the scale of the graph, the difference between the two simulations is not obvious. However, if we enlarge box A, it is possible to see the difference between the two simulations. Figures 6.20 and 6.21 show the tool traces of Test 1 and Test 2 respectively. In Test 1, the position error of each axis falls within 0.005 mm and in Test 2, the position error of each axis falls within 0.03 mm.

According to the two simulations, we can verify that the implemented Position Controller and the performance of the Position Controller depends on the PID gains. In practical terms, finding suitable PID gains is very important for implementation of a Position Controller.

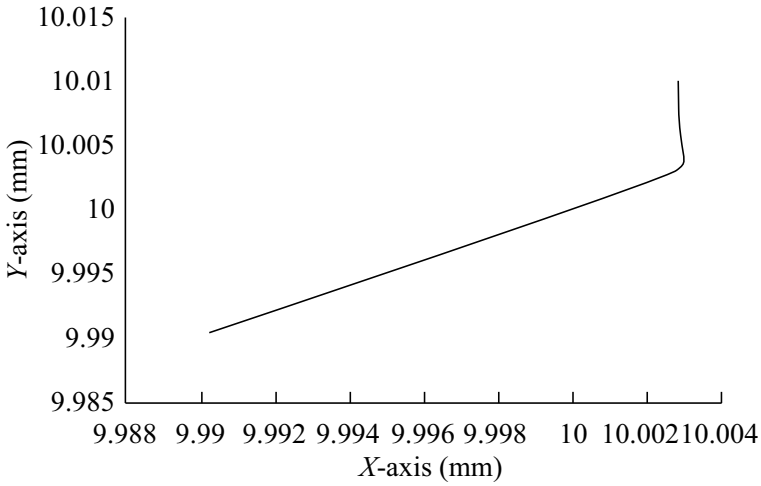


Fig. 6.20 Test 1 index 1

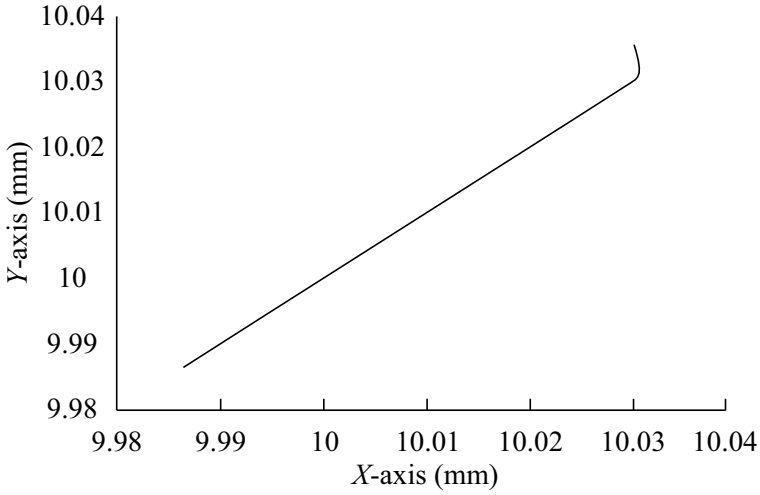


Fig. 6.21 Test 2 index 1

6.3 Architecture of an ADCBI-type NCK

As shown in Fig. 6.23, an ADCBI-type NCK consists of an Interpreter, a Look-Ahead module, a Rough Interpolator, a Fine Interpolator, and a Position Controller. The difference between an ADCAI-type NCK and an ADCBI-type NCK is the execution sequence of Acc/Dec control and rough interpolation. In an ADCBI-type NCK, the velocity profile based on acceleration and deceleration capability and programmed feedrate is generated and rough interpolation is executed from the velocity profile. Therefore, the rough interpolation algorithm and the Acc/Dec control algorithm of an ADCBI-type NCK are different from those of an ADCAI-type NCK. The details of the algorithms were given in Chapter 4.

Most of the algorithms for an ADCBI-type NCK are the same as those for an ADCAI-type NCK, except for the Look-Ahead module, Acc/Dec Controller, and Rough Interpolator. Consequently, the modules of an ADCAI-type NCK are identical to those used for an ADCBI-type NCK.

Figure 6.23 shows the data flow between the modules in an ADCBI-type NCK. In an ADCBI-type NCK, the ring buffers between the Interpreter and the Look-ahead module, between the Look-ahead module and the Acc/Dec Controller, between the Acc/Dec Controller and the Rough Interpolator, between the Rough Interpolator and the Mapping module, and between the Fine Interpolator and the Position Controller, are used for synchronization of data flow between the modules.

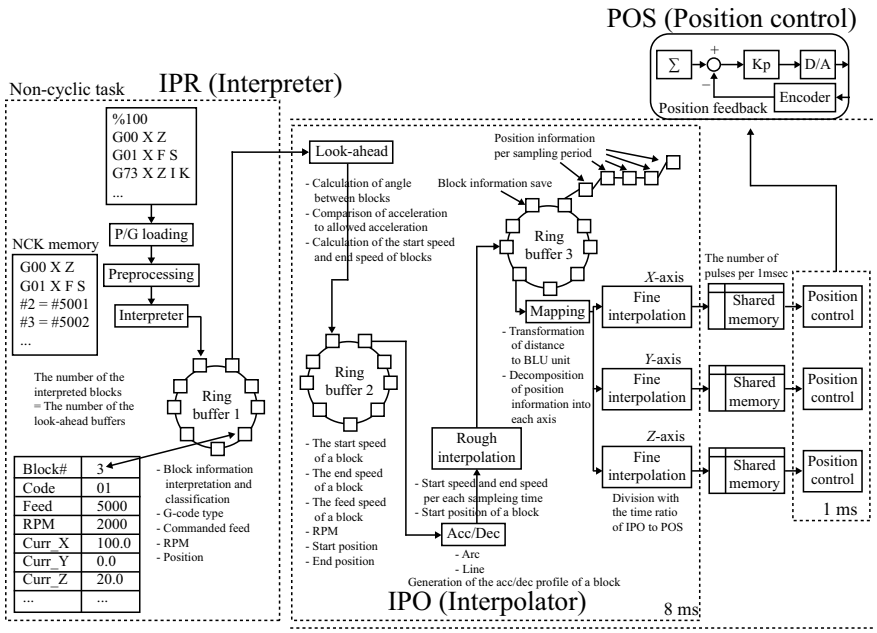


Fig. 6.22 ADCAI-type NCK

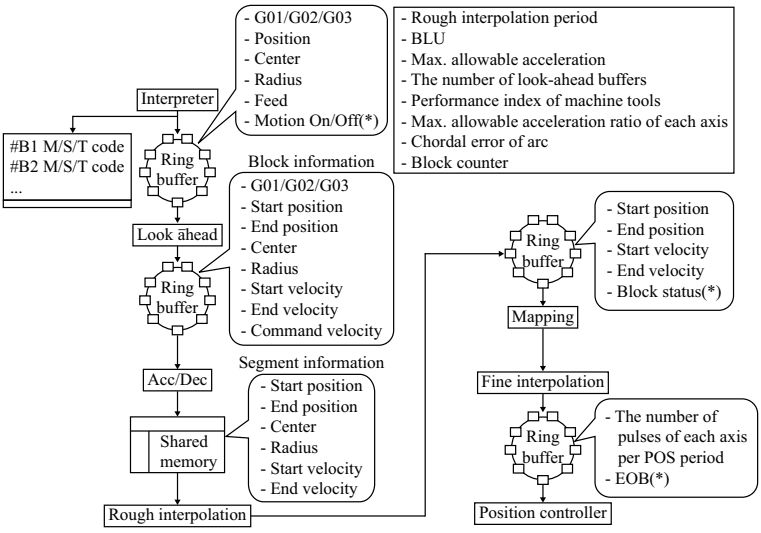


Fig. 6.23 ADCBI-type NCK

6.3.1 Implementation of the Look-Ahead Module

The Look-ahead module calculates the feasible speeds at the beginning and end of the current block by looking ahead at successive blocks. In Section 4.4, the details of the Look-ahead function were described and, in particular, the detailed algorithm was introduced from Fig.s 4.32 and 4.33. To realize the Look-ahead module, we implemented the procedures shown in Fig. 4.32 and Fig. 4.33.

6.3.1.1 Input and Output of the Look-Ahead Module

Input

The output of the Interpreter is input to the Look-ahead module. The following is the data structure that is implemented to store the input of the Look-ahead module. It is used as an element of the ring buffer between the Interpreter and Look-ahead modules.

```
class CRingIR : public CObject { public :
    int nGCode;          // G-code type, (0: G00, 1: G01, 2: G02)
    CVector Start;       // Start position of a block (mm)
    CVector End;         // End position of a block (mm)
    CVector Cen;         // Center point of arc (mm)
    double dRadius;      // Radius of arc (mm)
    double dFeed;        // Feedrate (mm/min)
    int nStatus;         // Block status (0: start, 1: end)
    int nStopMode;       // Path control mode. (1: Exact Stop,
                         // 0: otherwise)
    int nBlockNumber; // Block number.
};
```

Output

The Look-ahead module generates the speeds at the beginning and the end of a block as an output. The following is the data structure used to store the output of the Look-ahead module and is used as the element of the ring buffer between the Look-ahead module and the Acc/Dec Controller.

```
class CRBLookaheadBlock : public CObject { public:
    int nGCode;          // G-code type (0: G00, 1: G01, 2: G02)
    Cvector vPStart;     // Start position of a block (mm)
    Cvector vPEnd;       // End position of a block (mm)
    double dVStart;      // Speed at the start position (mm/sec)
    double dVEnd;        // Speed at the end position (mm/sec)
    double dVComm;       // Speed modified by Look-ahead
                         // algorithm (mm/sec)
    CVector vPCenter; // Center position of arc (mm)
    double dRadius;      // Radius of arc (mm)
};
```

6.3.1.2 Functions for the Look-Ahead Module

LookAhead()
This is the main function of the Look-ahead module and the implementation of the flowchart shown in Fig. 4.32. In this function, based on the look-ahead buffer storing the information about the following blocks, the allowable speed at the end of a block is computed and the feasibility of the end speed is verified based on the length of the block and the start speed. If a longer length is required than the length of the block for acceleration or deceleration from the speed at the beginning of the block to the speed at the end of the block, a new end speed is calculated by using Eq. 4.96. Otherwise, the computed end speed is specified as the end speed of the block.

DetermineIBlockVelocity()
As the function realizes the flowchart shown in Fig. 4.33, it computes the speed at the beginning of a block considering the end speed of that block. Strictly speaking, this function is used to compute the speeds at the beginning and the end of the *i*th block in LookAhead(). In this function, two kinds of start speed are computed; the first reflects the length of the block as mentioned in Section 4.4.1.1 and the second reflects the feasible corner speed as mentioned in Section 4.4.1.2. Finally, comparing the two kinds of speed and the programmed feedrate, this function selects the smaller speed as the start speed of the block.

DetermineVelocityBetweenLL()
This function calculates the speed at the corner where two successive linear blocks meet and realizes the algorithm mentioned in Section 4.4.1.2. In Section 4.4.1.2, two algorithms for determining the corner speed were introduced; one is based on the joint maximum allowable acceleration and deceleration and the other is based on the cartesian maximum allowable acceleration and deceleration. This function realizes two methods and is used in DetermineIBlockVelocity().

DetermineVelocityBetweenLC()
This function calculates the speed at the corner where a preceding linear block and a following arc block meet and realizes the algorithm described in Section 4.4.1.2. In

this function, the angle θ between the two successive blocks, depicted in Fig. 4.28, means the angle between a linear block and the tangent vector of an arc block at the corner. This angle θ is input to Eqs. 4.87 and 4.88 to determine the speed at the corner.

DetermineVelocityBetweenCL()
This function calculates the speed at the corner where a preceding arc block and a following linear block meet and realizes the algorithm described in Section 4.4.1.2. The way of determining the angle θ between two successive blocks is the same as that in DetermineVelocityBetweenLC().

DetermineVelocityBetweenCC()
This function calculates the speed at the corner where a preceding arc block and a following arc block meet and realizes the algorithm described in Section 4.4.1.2. How to determine the angle θ between two successive blocks is the same as that in DetermineVelocityBetweenLC().

6.3.2 Implementation of an Acc/Dec Controller

An Acc/Dec Controller of ADCBI-type NCK generates the speed profile of a block. As mentioned in Section 4.3.1, a block can be classified as either a normal block or a small block. Depending on the type of the block, the way of generating the speed profile is different. In an Acc/Dec Controller, whether a block is a normal block or a small block and whether a block is a linear block or an arc block is checked first of all. According to the type of block, a block can be divided into four kinds; a linear normal block, a linear small block, an arc normal block, and an arc small block. The Acc/Dec Controller then calls the appropriate sub-function based on the block type.

The implemented Acc/Dec Controller includes a Look-ahead function. Because a Look-ahead function practically includes speed profile generation of all cases mentioned in Section 4.3.2, it is not necessary to implement the algorithms for speed profile generation of the twelve cases mentioned in Section 4.3.2 and, therefore, the following four functions for generating speed profile were implemented.

6.3.2.1 Input and Output of the Acc/Dec Controller

Input
The Acc/Dec Controller gets the start position, the end position, the allowable start speed, and the allowable end speed of a block as input. In addition, if the block is an arc block, the center position and the radius of an arc are input to the Acc/Dec Controller. The following is the implemented data structure to store the input of the Acc/Dec Controller.

```
class CRBLookaheadBlock : public CObject { public:
    int nGCode;          // G-code type (0: G00, 1: G01, 2: G02)
    CVector vPStart;     // Start position of a block (mm)
    CVector vPEnd;       // End position of a block (mm)
    double dVStart;      // Speed at the start position (mm/sec)
    double dVEnd;        // Speed at the end position (mm/sec)
    double dVComm;       // Speed modified by Look-ahead
                         // algorithm (mm/sec)
    CVector vPCenter;    // Center position of arc (mm)
    double dRadius;      // Radius of arc (mm)
};
```

Output

The Acc/Dec Controller generates the speed for every interpolation iteration time in the case of a linear block and generates the angular speed every interpolation iteration time in the case of an arc block. In addition, it outputs the G-code type, the start position, the end position, and the center position of a block for Rough Interpolation.

```
class CRBAccDecBlock : public CObject { public:
    int nGCode;          // G-Code type (0: G00, 1: G01, 2: G02)
    CVector vPStart;     // Start position of a block (mm)
    CVector vPEnd;       // End position of a block (mm)
    double* dVi;         // Speed during iteration time of inter-
                         // polation in the case of a linear block.
    int nBlockN;         // Iteration number of interpolation.
    double* dAi;         // Angular speed during iteration time of
                         // interpolation in the case of an arc
                         // block.
    double dError;       // Remaining distance.
    double dRadius;      // Radius of an arc (mm).
    double dTheta;       // Remaining angle of arc.
    CVector vPCenter;    // Center position of an arc (mm).
};
```

6.3.2.2 Functions of the Acc/Dec Controller

DetermineVelocityProfile( )

This is the main function of the Acc/Dec Controller. This function determines whether the input block is a normal block or a small block and calls the appropriate function depending on the type of the block. Figure 6.24 shows the flowchart of this function.

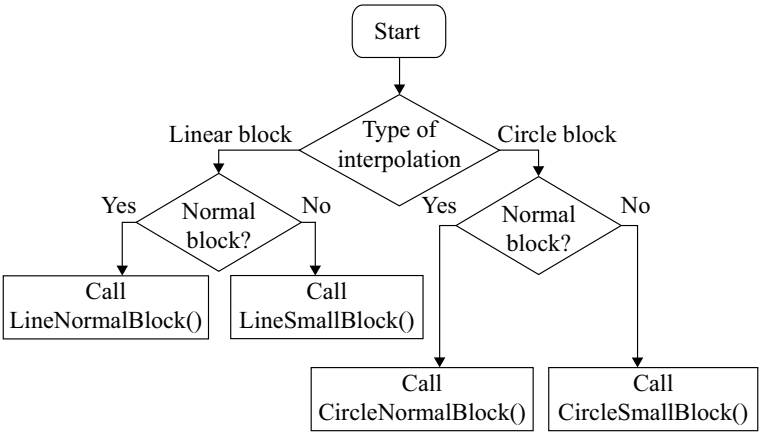


Fig. 6.24 Flowchart of DetermineVelocityProfile function

LineNormalBlock( )

This function generates the speed profile every interpolation iteration time for a linear normal block. T_{acc}, T_{dec}, T_{const} in Fig. 6.25 denote the time spent to reach the commanded speed from the start speed by the maximum allowable acceleration, the time spent to reach the end speed from the commanded speed by the maximum allowable deceleration, and the time period of constant speed, respectively. a_1 and a_3 denote the acceleration and the deceleration respectively. In addition, V_{start}, V_{end}, and V_{max} denote the start speed, the end speed, and the commanded speed of a block and T_{ipo} denotes the interpolation iteration time. V_i is the speed of the ith interpolation iteration.

In the case of a linear normal block, the speed profile consists of an acceleration period, a deceleration period, and a constant speed period. To generate the speed profile during the acceleration period, the acceleration value and the acceleration time need to be determined. Although the maximum allowable acceleration is specified in the NCK, the new acceleration value, which makes the acceleration time a multiple of the interpolation iteration time and is the closest to the maximum allowable acceleration value, is calculated for the simplicity of the algorithm. If the acceleration time is not a multiple of the interpolation iteration time, the change of acceleration during the acceleration period occurs and this makes it difficult to compute the displacement along the axis during the iteration time of rough interpolation.

In the same way, the deceleration time and the deceleration value are determined. Like the acceleration value, the deceleration value that makes the deceleration time a multiple of the interpolation iteration time and is closest to the maximum allowable deceleration, is determined.

After the acceleration time and the deceleration time have been determined, the displacement during the constant period can be computed by subtracting the displacement during the acceleration and the deceleration period from the length of the

block. The constant time should now be a multiple of the interpolation iteration time, like the acceleration time.

Finally, it is possible to compute the speed at the beginning and end of each iteration time from the above-mentioned values. However, because the acceleration time, the deceleration time, and the constant time should be multiples of the interpolation iteration time, a tool may not reach the programmed position. The remaining length is distributed by a residual pulse-handling algorithm in the Rough Interpolator.

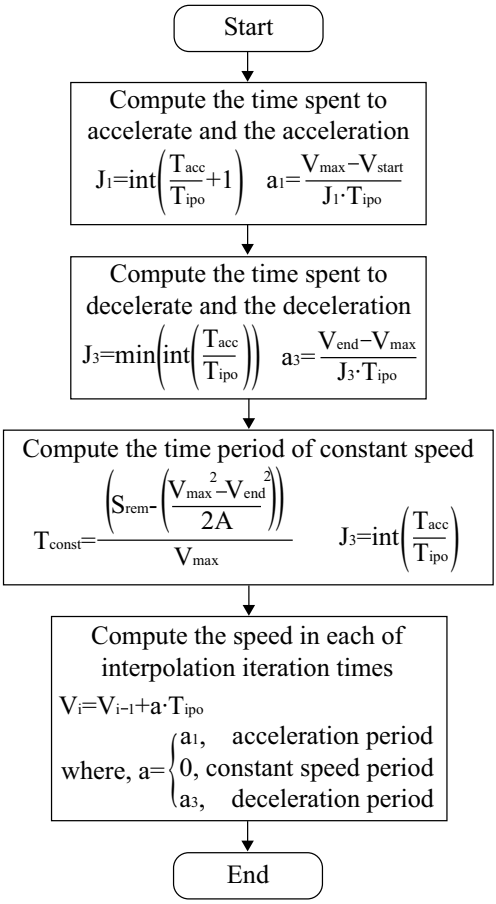


Fig. 6.25 Flowchart for LineNormalBlock function

LineSmallBlock( )

This function generates the speed profile for a linear normal block. Figure 6.26 shows the flowchart of this function. In Fig. 6.26, S_{tot} denotes the length of a block and V_{max} denotes the maximum reachable speed in the block. The other notations are the same as those in Fig. 6.25.

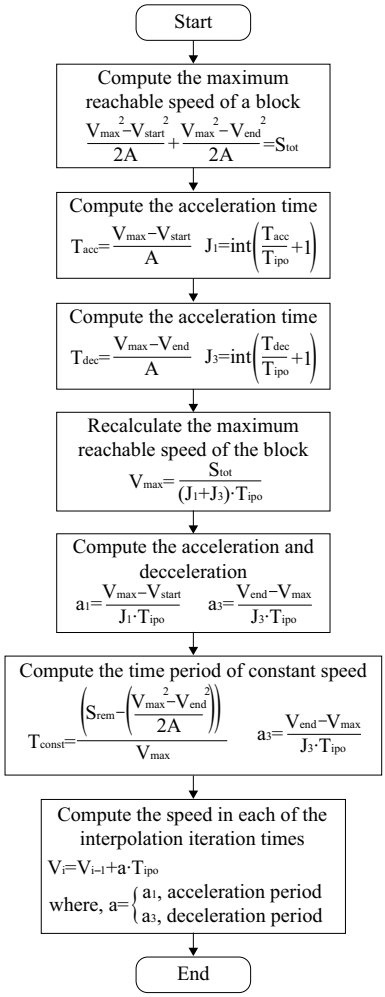


Fig. 6.26 Flowchart for LineSmallBlock function

In the case of a linear small block, for simplicity of the algorithm it is assumed that the acceleration and the deceleration are equal. From this assumption, the maximum reachable speed is computed based on the maximum allowable acceleration and deceleration within a block. Here, like the algorithm for a linear normal block, the acceleration or deceleration time is made a multiple of the interpolation iteration time. Next, by using the computed acceleration time or deceleration time, the maximum reachable speed within a block is recalculated and the final acceleration or deceleration is computed.

From the acceleration/deceleration and acceleration/deceleration time, it it possible to compute the speed at the beginning and end of each iteration time.

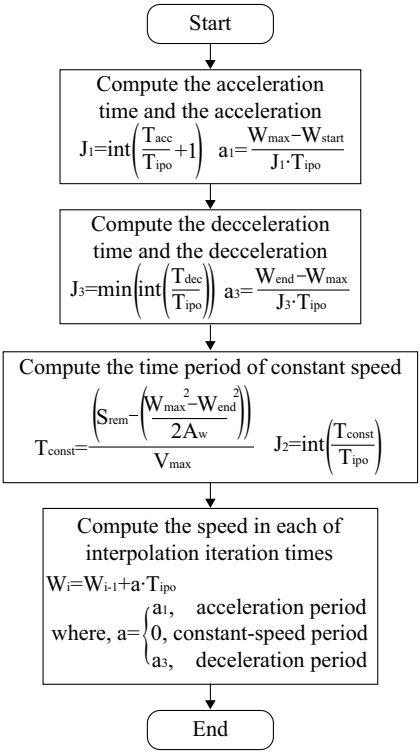


Fig. 6.27 Flowchart for CircleNormalBlock function

CircleNormalBlock( )

This function generates the speed profile for an arc normal block. It is implemented by the same algorithm as that of LineNormalBlock(). In the case of a linear normal block, a speed means the feedrate in a cartesian coordinate system and in the case of an arc normal block, a speed means an angular speed. Figure 6.27 shows the flowchart for generating the speed profile for an arc normal block and W and A_W in Fig. 6.27 are an angular speed and an angular acceleration, respectively.

CircleSmallBlock( )

This function generates the velocity profile of a small arc block and it is implemented with the same algorithm as that of LineSmallBlock(). Figure 6.28 shows the flowchart of the Acc/Dec control algorithm for a small arc block. In Fig. 6.28, W and A_W denote an angular speed and an angular acceleration, respectively. θ_{tot} denotes the angle between the start position and the end position of an arc.

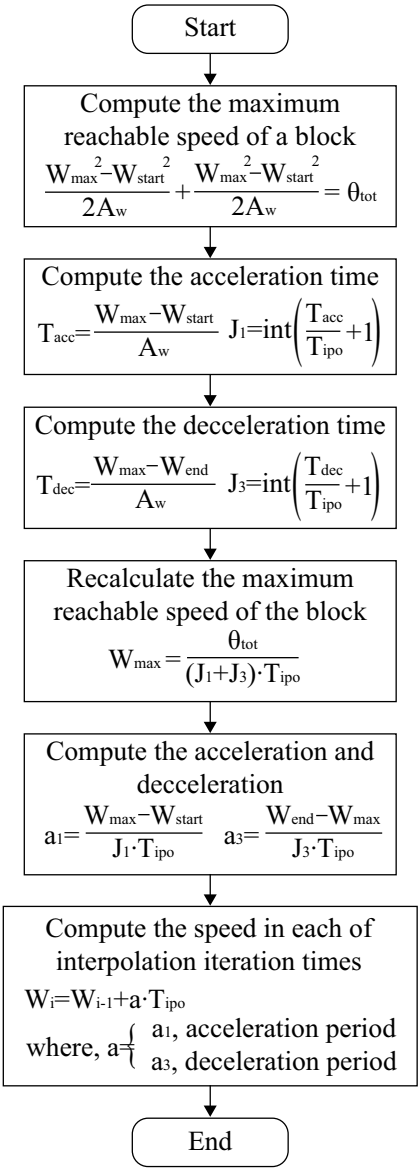


Fig. 6.28 Flowchart for CircleSmallBlock function

6.3.3 Implementation of the Rough Interpolator

The Rough Interpolator of an ADCBI-type NCK calculates the position at which the tool should arrive for every interpolation iteration time based on the velocity profile from the Acc/Dec Controller.

6.3.3.1 Rough Interpolator Input and Output

Input

The output of the Acc/Dec Controller is input to the Rough Interpolator. The following is the implemented data structure of the Rough Interpolator. The inputs consist of the speed within the interpolation iteration time (an angular speed in the case of an arc), the remaining displacement, etc.

```
class CRBAccDecBlock : public CObject { public:
    int nGCode;         // G code type (0: G00, 1: G01, 2: G02)
    CVector vPStart;    // Start position of a block (mm)
    CVector vPEnd;      // End position of a block (mm)
    double* dVi;        // Speed during iteration time of inter-
                        // polation in the case of a linear block.
    int nBlockN;        // Iteration number of interpolation.
    double* dAi;        // Angular speed during interation time
                        // of interpolation in the case of an
                        // arc block.
    double dError;      // Remaining distance.
    double dRadius;     // Radius of arc (mm).
    double dTheta;      // Remaining angle of arc.
    CVector vPCenter; // Center position of arc (mm).
};
```

Output

The Rough Interpolator generates and stores the position at which a tool should arrive for every interpolation iteration time. The following is the implemented data structure for storing the output from the Rough Interpolator.

```
class CRBRoughIPOBlock : public CObject { public:
    CVector* vPi; // Position at which a tool should arrive
                  // each iteration time of interpolation.
    int nBlockN;  // Iteration number of interpolation.
};
```

6.3.3.2 Functions of the Rough Interpolator

RoughInterpolation()
This is the main function of the Rough Interpolator and calls one of the following two functions depending on the type of interpolation; linear interpolation or circular interpolation. The flowchart is shown in Fig. 6.29.

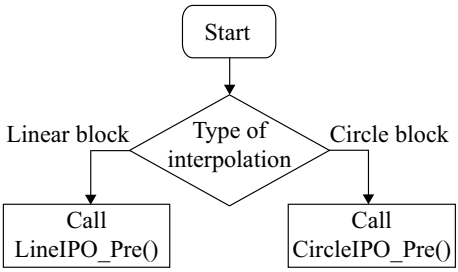


Fig. 6.29 Flowchart for the RoughInterpolation function

LinearIPO_Pre()
This function carries out rough interpolation for a linear block. Figure 6.30 shows the flow chart for this function. In Fig. 6.30, P_{start} and P_{end} mean the start position and the end position of a block respectively. P_i denotes the position where the tool should arrive for the ith iteration time and S_{rem} is the remaining length, being the length between the programmed position and the position at which the tool should arrive when the tool moves at the speed specified by the speed profile. N is the number of iteration times over a block. The other notations are the same as those in Fig. 6.25.

For rough interpolation of a linear block, the displacements through which the tool should move every interpolation iteration time, based on the speed profile, are computed. If length remains afterwards, the remaining length is divided by the number of iteration times and the computed value is added to the displacement for each interpolation iteration time.

Finally, Rough Interpolator calculates the position at which the tool should arrive every interpolation iteration time by using the above computation result and the start position of the block.

CircularIPO_Pre()
This function carries out the rough interpolation for an arc block. The procedure and the algorithm for this function is similar to that of LinearIPO_Pre() except in the meaning of the speed. In the case of an arc block, the speed means an angular speed. Therefore, by replacing the speed by angular speed in Fig. 6.28, the flowchart for carrying out circular interpolation can be obtained as shown in Fig. 6.31. In Fig. 6.31, X_{start} and X_{center} mean the start position and the center position of an arc block with respect to the X-axis coordinate.

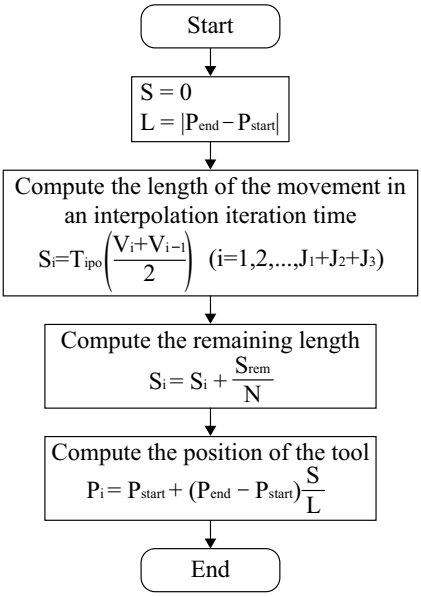


Fig. 6.30 Flowchart for LinearIPO_Pre function

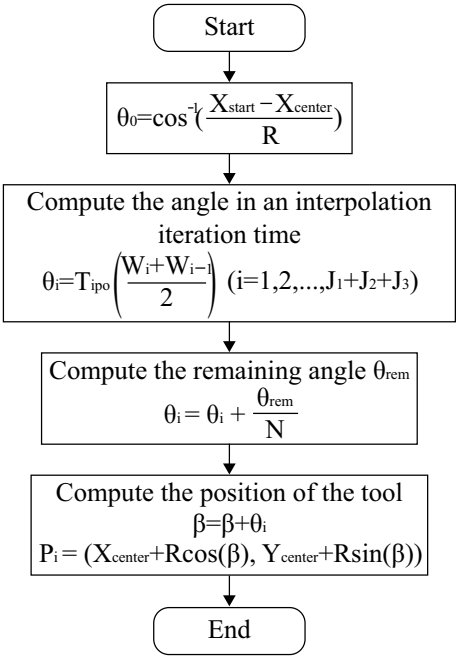


Fig. 6.31 Flowchart for CircularIPO_Pre function

6.3.4 The Mapping Module

The Mapping module divides the tool movement over the interpolation iteration time into displacements of each axis, being the input to the Fine Interpolator. In the case of ADCBI-type NCK, because Acc/Dec control and rough interpolation are applied to the tangential direction of tool movement, unlike ADCAI-type NCK, it is necessary to divide the interpolated point into displacements for each axis. In general, the Mapping function can be included in the Rough Interpolator. However, in this textbook, the Mapping function is implemented as a separate module in order to show the execution procedure of NCK.

6.3.4.1 Input and Output of Mapping Module

Mapping input
The position at which the tool should arrive for every interpolation iteration time is input to the Mapping module. The following is the implemented data structure to store the input to the Mapping module.

```
class CRBRoughIPOBlock : public CObject { public:
   CVector* vPi; // Position at which tool should arrive
                 // each iteration time of interpolation.
   int nBlockN;  // Iteration number of interpolation.
};
```

Mapping output
The Mapping module generates and stores the displacement of each axis every interpolation iteration time. The stored output is used as input to the Fine Interpolator. The following is the implemented data structure to save the output of the Mapping module.

```
class CRingAF : public CObject { public:
   Cvector* P; // Distance to move per interpolation
               cycle in terms of number of pulses.
   int N;      // Number of repetitions for interpolation
               // in executing block.
};
```

6.3.4.2 Mapping Functions

Mapping()
This function implements the mapping function, it computes and stores the difference between the start position and the end position of each axis for every interpolation iteration time.

6.4 Summary

An ACDAI-type NCK consists of an Interpreter that converts a programmed block into the actual toolpath, a Rough Interpolator that divides the toolpath into linear segments, an Acc/Dec Controller that smoothes drastic changes in axis speed, a Fine Interpolator that divides the linear segments over one interpolation iteration time into linear segments over iteration times for position control, and a Position Controller that computes the displacement of each axis based on the position error. These modules are executed sequentially and the data between the modules is transferred via ring buffers.

An ADCBI-type NCK consists of an Interpreter, a Look-ahead module, an Acc/Dec Controller, a Rough Interpolator, a Fine Interpolator, and a Position Controller, which are executed sequentially. The difference between an ADCBI-type NCK and an ADCAI-type NCK is the execution sequence of the Acc/Dec Controller and the Rough Interpolator. In ADCBI-type NCK, the speed profile along the tangential direction of tool movement is generated before rough interpolation. Therefore, the functions and algorithms for rough interpolation and Acc/Dec control are different from those of an ADCAI-type NCK. In addition, a Look-ahead function can be used in ADCBI-type NCK. Besides these algorithms, others are commonly used.

Part II
Open-architectural Soft CNC Systems

Chapter 7
Programmable Logic Control

The Programmable Logic Control, PLC, is the automation component for controlling the execution sequence of machines or production processes. In the CNC system, the PLC has the task of controlling the mechanical behavior of the machine with the exception of axis control. In this chapter, the general architecture, function, and behavior of a PLC will be addressed and the role and characteristics of PLCs in the CNC system will be described. Also, IEC-1311-3, the international standard PLC programming language, will be introduced and Soft-PLC will be introduced. Finally, with respect to the design of a PLC system, the architecture and execution conditions will be described and an executor of a PLC program will be implemented.

7.1 Introduction

In general, a PLC system contains a logical operator, relay, counter, timer, and arithmetic calculation functions to control various machines and processes. A PLC system can be defined as the controller that consists of a CPU and memories to edit and execute a PLC program. A PLC system is a software-based control system that enables easy editing, execution, and modification of a PLC program instead of using hardware sequence control that controls processes by wiring together the hardware elements such as relay, timer, and counter. Therefore, a PLC system has the following advantages compared with a hardware-based sequence control system.

- *Flexibility*: the modification of control logic is possible by changing a PLC program.
- *Scalability*: the extension of a system is easily possible by adding logic and then changing PLC programs.
- *Economic efficiency*: the cost decreases through reduction of design, high reliability, and convenience of maintenance.
- *Miniaturization*: the physical volume required for installation is small compared with hardware control systems.

- *Reliability*: the probability on breaking down due to poor contact decreases because the system consists of a semiconductor having a non-contact switch.
- *Performance*: advanced control functions such as data handling and arithmetic calculation are enabled.

The comparison between hardware-based sequence control and PLC are summarized in Table 7.1.

Table 7.1 Comparison between hardwired sequence controller and PLC

Type	Sequence control	PLC
Logic type	Hard Logic	Soft Logic
Supported functions	Relay, Timer, Preset Counter	Relay (AND, OR, NOT), Up/down Counter, Shift register Arithmetic calculation, logic calculation
Control type	Contact type (limited life, slow control)	Non-contact type (long life, fast, high reliability)
How to change the logic	By changing wiring between hardware elements	By changing PLC program
Installation time	Building, inspection and test run take a long time.	The time for inspection and test run decreased
System characteristics	Stand-alone control equipment	It is easy to extend system. It is possible to connect to a computer.
Maintainability	For maintenance, long time is needed.	Due to high reliability and long life, the need for maintenance is small.
Volume	Miniaturization is difficult.	Miniaturization is possible.

7.2 PLC Elements

As depicted in Fig. 7.1, the PLC system consists of a Programming Tool, Input Unit, Output Unit, Program Memory, DATA Memory and CPU Unit. The details of each unit are described below.

1. *Programming Tool* – used for editing the PLC program and loading the program to the CPU unit,
2. *Input Unit* – receives on/off signals from a variety of switches, sensors, andtimer and converts the received signals into CPU-interpretable signals,
3. *Output Unit* – outputs on/off signal to motor, relay, and display,

4. *Program Memory* – stores the user program,
5. *DATA Memory* – stores executable program such as OS, and
6. *CPU Unit* – interfaces various auxiliary equipment and executes the logic calculation.

The Input Unit consists of:

1. The input signal terminal that connects the outer elements with the PLC system,
2. The input signal converter that converts the high voltage of external equipment into a low voltage signal to meet the CPU Unit's voltage requirements,
3. The display circuit that shows the functional status of the Input Unit, and
4. The interface circuit that transmits the status of the Input Unit to the CPU Unit.

From the hardware point of view, the Input Unit receives various voltage signals including DC or AC and finally transmits 5V DC to the CPU Unit. The Input Unit prevents instant noise whose duration is below 5 ms by using a noise filter that includes a 5 ms delay circuit. In order to protect the CPU Unit from excessive voltage and current, the Input Unit has a protection circuit, that is a photo-coupler which converts the electric signal into a photo-signal.

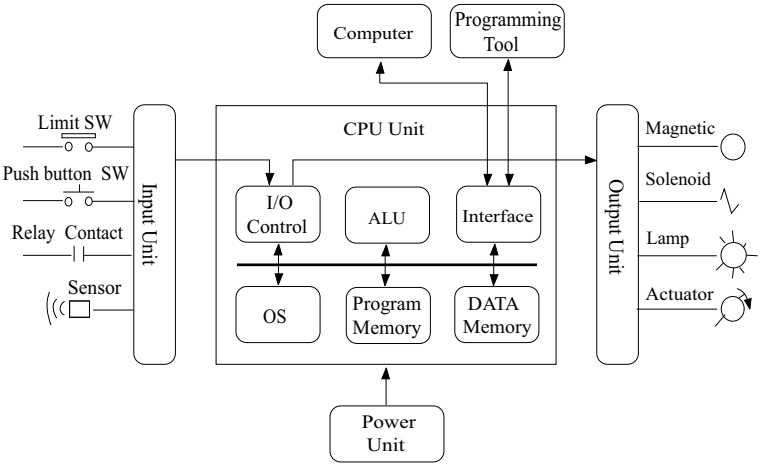


Fig. 7.1 Elements of the PLC system

The Output Unit converts the calculation result from the PLC to a signal for the outer actuator and outputs this signal to the outer actuator, *e.g.* a switch or a relay. It consists of an interface/multiplex circuit, latch circuit, electrically insulated circuit, module status display circuit, output signal converter, output signal terminal, and external output device. The interface circuit is used for transmitting the output of the CPU and the latch circuit is used for temporary storage of the output from the CPU. The insulation circuit is used for electrically insulating the signals from external devices. The module status display circuit is used for displaying the status of the

Output Unit, and the output signal converter amplifies the output signal of the CPU to activate the external device. Further, the output signal terminal is used for connecting the PLC system and external devices. In addition, the Output Unit includes a circuit to prevent excessive current due to short circuits in the output signal terminal.

The CPU Unit is the core module of the PLC system, executing logic calculation and arithmetic calculation by interpreting the PLC program and the final results are sent to the Output Unit. In other words, the CPU Unit handles the serial or parallel sequence logic operations (*e.g.* AND, OR, and NOT), timer or counter operations that are used for controlling the elapsed time based on an internal pulse counter, the four arithmetical operations, the comparison operation, junction operations (*e.g.* JUMP and CALL), mathematical function operations (*e.g.* COS, SIN, TAN, and Square root), data transmission, and code conversion.

Program Memory, which stores the user programs, and System Memory, which stores system data, OS, and application S/W, can be rewritable and can keep the data using an internal battery even in the case of power failure.

In addition, auxiliary units such as the programming tool and interface units for RS-232C serial communication and ethernet communication are included in the PLC system.

Therefore, the maximum input/output contact points, the speed of the CPU Unit (in FANUC, this is defined as the time consumption per step), the size of the Program Memory (in FANUC, this is defined as the maximum number of steps.), the kinds of commands, and the kinds of allowable external communications are specified to represent the performance of the PLC system.

The method of executing the command specified in a program is as follows.

A PLC programmer creates, edits and saves the PLC program by utilizing a programming language such as the ladder diagram, instruction list, etc.

After saving the PLC program, the PLC system scans and executes the steps from first to last. Accordingly, the PLC system generates the output signal by execution of the program sequence every specified time period.

As shown in Fig. 7.2, an internal interpreter takes one command from the part program in Program Memory and interprets the command. The interpreted command is executed by calling the appropriate built-in function. In the process of interpretation, bits of an address in the PLC program are read, and the corresponding address bits are set to ON or OFF.

The above-mentioned method is an interpretative method whereby the interpretation and execution of steps is repeated one by one while logic control is performed. Because an interpreter-type PLC system reads and interprets the individual native code of a program sequence and performs the pre-specified macro routine related with the native code, reduction of execution speed cannot be avoided. Furthermore, because various subroutines for handling internal commands are included in an interpreter-type PLC system, calling subroutines and returning results occur frequently during the execution of a sequence program.

In general, the elapsed time for one scan is very important for the performance of PLC system. Therefore, in order to overcome the slow speed and inefficiency of the interpretative method, a more efficient and faster method is required.

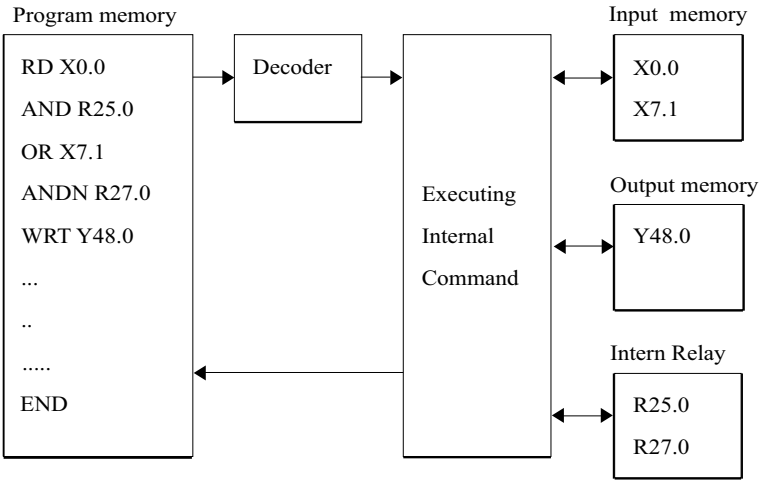


Fig. 7.2 Operation of internal interpreter

The compiling method was introduced to overcome the disadvantages of the interpretative method and the behavior of the compiling method is shown in Fig. 7.3. In the compiling method, a program sequence is interpreted in advance and the internal commands are replaced with pre-specified routines. Jumps and returns are omitted during logic control and the execution speed can be increased.

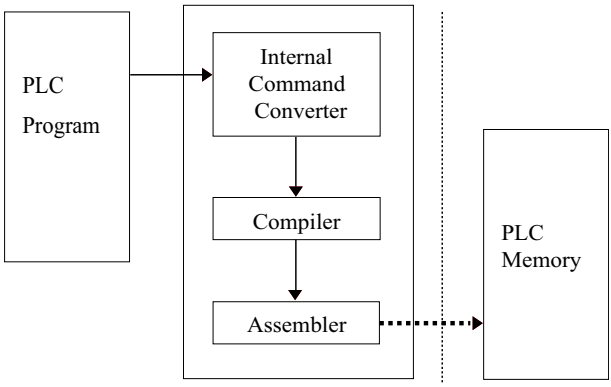


Fig. 7.3 Behavior of compiling method

The workflow of the compiling method can be summarized as follows:

1. The PLC programmer edits a sequence program using a programming tool, which can exist outside or inside the PLC system.
2. The sequence program is converted to a native program with internal commands by the Internal Command Converter.

3. The compiler replaces the internal commands with the appropriate pre-defined assembly codes.
4. The assembler converts the assembly code into the machine language (binary code) which can be executed by the CPU. Because the majority of commands in a PLC program are independent from each other it is possible to save memory and increase interpretation speed if commands are handled one by one in the assembler after the labels and variables relevant to multiple steps (blocks) have been handled.
5. Finally, the native code is transmitted to the internal memory when the PLC is idle. It is then executed sequentially.

In consequence, the PLC program edited by a programmer is converted to an executable binary code by a compiler and is sent to the PLC memory. Fast scanning becomes possible compared with the interpretation method.

7.3 PLC Programming

There are a variety of the programming languages to represent logic sequences and the IEC (International Electrical Committee) classifies the programming language into the statement list representation and the graphical representation.

As the graphical language, there is the ladder logic that is a method of drawing electrical logic schematics. As the statement list (textual) language, there are mnemonic language, Boolean language, and machine language. In practice, the ladder logic, which can be easily mapped with a sequence logic drawing, has been widely used. Figure 7.4 shows a ladder diagram and a mnemonic program that has same meaning as the ladder diagram.

Because the symbols and commands used in ladder logic and mnemonic language are slightly different, depending on the makers, it is essential to edit new programs when the PLC system is changed.

Table 7.2 shows the mnemonic symbols of the basic and advanced command sets (*e.g.* timer/counter function, control function and register manipulation function) for Yasnac's PLC programming.

In a typical PLC program, basic commands such as LD, LD-NOT, AND, AND-NOT, OR, OR-NOT, and OUT are widely used. The Timer function, which sets the output port as ON or OFF after a pre-determined time, is widely used. Basically, the Timer measures the specified time by counting the time-based pulses generated every constant time interval and multiplying the number of pulses counted by the sampling time for pulse generation. The Timer sets the output port as ON of OFF after the specified time. According to the PLC system, the sampling time of pulse generation can be set as 10 ms, 100 ms, and 1 s. The Timer can be classified as one of two types: an UP-timer, which counts the incremental time to the specified time, or a DOWN-timer, which counts downwards with decremental time from the specified time.

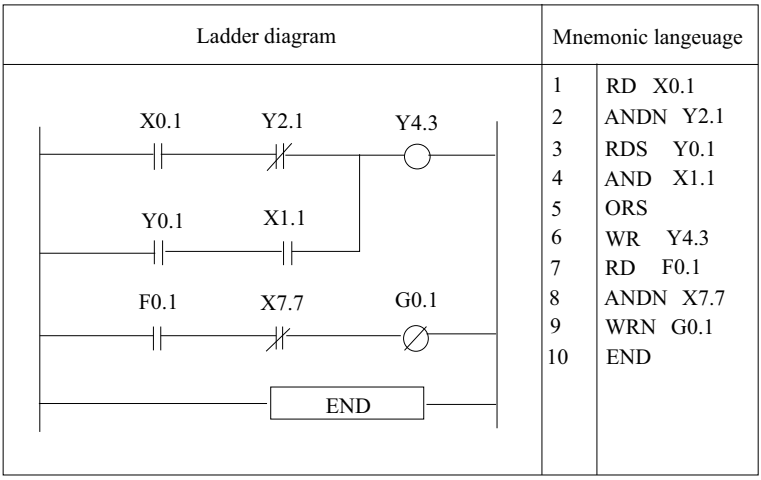


Fig. 7.4 Ladder diagram and mnemonic program

A Counter is used for counting time like the Timer. However, unlike the Timer, the Counter uses an external input signal, whereas the Timer uses the internal time base pulse for counting time.

Unlike the initial PLC systems, which enabled the fundamental logical operations, a modern PLC system can perform the four arithmetical operations of BCD values, conversion between decimal and hexadecimal values, branching operations (*e.g.* JUMP and CALL), and trigonometrical functions and special functions for advanced control.

Editing PLC programs is outside the scope of this book. If you want more information about PLCs, refer to the related books on PLC.

7.4 Machine Tool PLC Programming

The PLC system of a CNC machine tool executes not only M-, T- and S-codes specified in a part program but also activates or inactivates external switches, executing the PLC program together with input signals from the sensors in machine tools.

Therefore, when we create a PLC program for a CNC machine tool, special considerations are necessary compared with the general PLC system. However, from a functional point of view, the two types of PLC system are not different.

The role and characteristics of a PLC program in a machine tool are summarized as follows:

1. The PLC program sends the status of the operation panel to the NCK and shows the status of the NCK to the operator via the operation panel.

Table 7.2 Basic commands and functional commands for PLC programming

Type	Instruction	Meaning
Basic command	LD	Regular contact used in beginning of step.
	LD-NOT	'Not' contact used in beginning of step.
	AND	Regular contact that represents the serial connection in Ladder diagram.
	AND-NOT	'Not' contact that represents the serial connection.
	OR	Regular contact that represents the parallel connection.
	OR-NOT	'Not' contact that represents the parallel connection.
	XOR	Exclusive OR.
	XNR	Exclusive AND.
	STR	After storing the operation result on stack, perform LD command.
	STR-NOT	After storing the operation result on stack, perform LD-NOT command.
	AND-STR	Operation result AND the value on stack.
	OR-STR	Operation result OR the value on stack.
	OUT	Ouput the operation result.
Timer	TIM	Fixed timer.
	TMR	Variable timer.
Control command	NOP	No action.
	MCR	If input condition is ON, the program is performed until END.
	END	Represents the end of MCR command.
	RET	Represents the end of PLC program.
	RTI	If input condition is ON, perform RET command.
	SET	Set ON.
	RTH	Rep. end of high-speed PLC program.
	JMP	Jump to the number specified by ADR.
	ADR	Specify the number to which is jumped by the JMP command.
Register command	INR	Increase the value in register by one.
	DCR	Decrease the value in register by one.
	CLR	Reset the register.
	CMR	Reverse the register
	ADI	Add value of register to the specified value.

i. The operator changes the machine operation mode (*e.g.* Auto mode, MDI mode, and Zero Return mode) by turning on or off switches on the operation panel. The change of machine operation mode is sent to the NCK by a PLC program.
ii. The operator controls the axis' movement, such as JOG, cycle start, or emergency stop by turning on or off switches on the operation panel.
iii. By turning on or off the LEDs and lamps on the operation panel, the PLC program displays the execution status of a part program.

2. Through interaction with the NCK, a PLC program helps the execution of a part program.
   i. A PLC program prevents execution of the next block until the execution of an M-code is completed.
   ii. PLC program prevents execution of the next block until the spindle speed reaches the value specified by an S-code.
   iii. The PLC program prevents execution of the next block until the tool specified by a T-code has been attached to the spindle.
3. A PLC program provides various interlock functions to prevent the operator and the workpiece being damaged.
   i. It prohibits rotation of the spindle in the case of the chuck being unclamped.
   ii. It stops axis movement as soon as the spindle is stopped.
   iii. It changes operation mode to single block mode when the coolant's motor is overheated.

The general procedure for editing a PLC program is follows:

1. Assign addresses to the input and output ports,
2. Assign addresses to the internal relays and counters,
3. Design the sequence circuit to enable the intended logical operation based on the assigned addresses,
4. Select the appropriate programming language and edit the PLC program in the selected language,
5. Load the PLC program to the CPU module and carry out debugging.

The first step for editing a PLC program for a machine tool is to assign addresses to the input and output ports. The address means the connection point for transmitting the signal from/to the machine tool, CNC, relay, timer, counter, and data table. The type, transmitting direction, and reserved address for PLC programming are shown in Fig. 7.5. In the PLC shown in Fig. 7.5, the X address denotes the input signal transmitted from the machine to the PLC and the Y address denotes the output signal transmitted from the PLC to the machine. It is assumed that the number of input signals and output signals are both 64. 'G', 'F', and 'R' are used to represent the signal output from the PLC to the CNC, the input from the CNC to the PLC, and the internal relay. In addition, 16 32-bit timers and counters are defined and the 2048 bytes are allocated to the internal memory.

According to the addresses assigned in Fig. 7.5, the address definition for PLC programming is partially shown in Fig. 7.6.

After assigning the addresses, the sequence flow is designed and programming is performed as described in the sequence flow. Figure 7.7 shows an example of a typical sequence flow for a 3-axis machining center. The following addresses how to design the sequence flow of a PLC program.

1. The first thing at the beginning of a program is to check the condition of the emergency stop.

Add.	Signal Direction	Example Notation
X	Input from M/C to PLC	64 points (X0.0 ~ X7.7)
Y	Output from PLC to M/C	64 points (Y0.0 ~ Y7.7))
G	Output from PLC to CNC	256 points (G0.0 ~ G31.7)
F	Input from CNC to PLC	256 points (F0.0 ~ F31.7)
R	Internal Relay	512 points (R0.0 ~ R73.7)
T	Internal Timer	512 byte (32bit timer * 16)
C	Internal Counter	512 byte (32bit counter * 16)
D	Internal Memory	2048 byte (32bit data * 64)

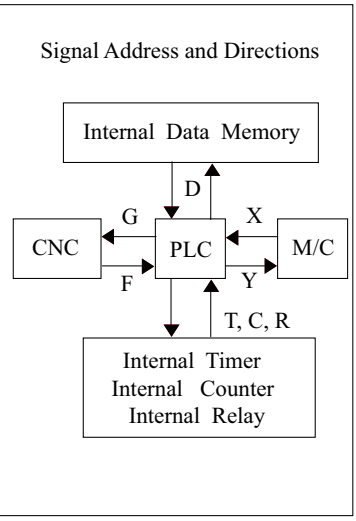


Fig. 7.5 Type, transmitting direction, and reserved address for PLC programming

2. Next is to design a sequence flow to handle an axis operation and transformation mode input from the user interface panel.
3. After step 2, the processes for handling T, S, and M codes are designed. For the T-code operation, the control flow related to the tool magazine rotation, tool change mechanism, spare tool management, turret rotation of a lathe, etc. should be considered. Several subroutines are essential for magazine operation of the machining center, as examples are 1) a rotational direction decision for the shortest distance based on the current tool position, 2) an ACC/DEC control for smooth rotation of the magazine, and 3) interrupt handling for high-speed rotation of the magazine, etc.
4. In the case of an S-code, the essential subroutines for spindle operation are relatively simple. Examples of necessary subroutines are 1) generation of a spindle-enable signal, 2) generation of a rotation direction (CCW or CW) signal, and 3) checking whether rotational speed is as commanded by communication with the NCK system, etc.
5. For handling of M-codes, the machine-specific sequence flow should be designed including M03 (Spindle CW), M04 (Spindle CCW), M05 (Spindle stop), M08 (Supply the cutting fluid), M09 (Stop supplying the cutting fluid). However, it is not necessary to design the conventional M-codes commonly interpreted by all types of machine, such as M00 (Program stop temporarily), M01 (Program stop if an optional stop button is pressed), M02 (Program end), and M30 (Program end and repeat) because the NCK system is handled in the normal CNC system.
6. Finally, the processes for turning on the ramps and displaying the messages for the user interface are designed.

X00.0	MPG Selection	Y00.0	Spindle CW	F05.0	CNC Ready
X00.1	+X axis Jog	Y00.1	Spindle CCW	F05.1	CNC Mode Auto
X00.2	- X axis Jog	Y00.2	Chuck Clamp	F05.2	CNC Mode Manual
X00.3	+Y axis Jog	Y00.3	Chuck Unlamp	F05.3	CNC Mode MDI
X00.4	- Y axis Jog	Y00.4	Servo(on/off)	F05.4	Reset
X00.5	+Z axis Jog	Y00.5	Hybraulic Motor	F05.5	CNC Alarm
X00.6	- Z axis Jog	Y00.6	Lubrication Motor	F05.6	Ref retrun-X axis
X00.7	Rapid	Y00.7	Work Light	F05.7	Ref return-Y axis
X01.0	Emergency Stop	Y01.0	Program End	F07.0	Ref return-Z axis
X01.1	+X axis Limit	Y01.1	+X axis Lamp	F07.1	Dry Run
X01.2	- X axis Limit	Y01.2	- X axis Lamp	F07.2	Machine Lock
X01.3	+Y axis Limit	Y01.3	+Y axis Lamp	F07.3	Optional Stop
X01.4	- Y axis Limit	Y01.4	- Y axis Lamp	F07.4	Optional Block Skip
X01.5	+Z axis Jog	Y01.5	+Z axis Lamp	F07.5v	Spindle Orient
X01.6	- Z axis Jog	Y01.6	- Z axis Lamp	G03.0	PLC Ready
X01.7	Over-travel Cancel	Y01.7	Cycle Start Lamp	G03.1	Emergency Stop
X02.0	Edit Lock	Y02.0	Feed Hold Lamp	G03.2	MPG Selection
X02.1	Cycle Start	Y02.1	Machine Lock	G03.3	PLC Alarm
X02.2	Feed Hold	Y02.2	Coolant Auto Lamp	G03.4	Rapid
X02.3	Chuck Switch	Y02.3	Coolant Run Lamp	G03.5	Overtravel Cancel
X02.4	Auto Door	Y02.4	Coolnat Stop Lamp	G03.6	Cycle Start
X02.5	Bar Feeder FW	Y02.5	Run Lamp	G03.7	Free Hold
X02.6	Bar Feeder BW	Y02.6	Alarm Lamp	G04.0	Spindle CW
X02.7	Door Interlock	Y02.7	Spindle CW Lamp	G04.1	Spindle CW
* * *					

Fig. 7.6 PLC programming signal definition (partial)

As can be seen from the above programming procedure, third parties have difficulty in understanding the PLC program without detailed descriptions and sequence charts. Also, re-programming is needed to add and modify other functions. Due to the absence of a standardized programming language, a programmer must know a variety of languages depending on different PLC systems and makers. This makes the training of a programmer and the maintenance of the PLC system difficult.

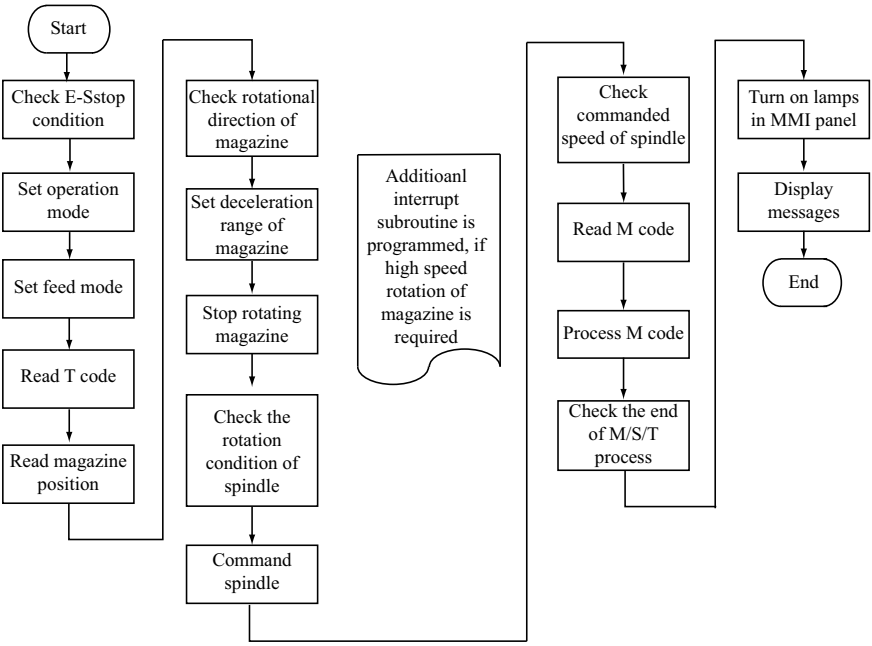


Fig. 7.7 Typical 3-axis machining center sequence flow

7.5 PLC System Functions

In order to establish Factory Automation, enabling cost reduction, unmanned operation, and quality improvement, it is essential to build a network system to connect various automation units (*e.g.* CNC machine tools, FA robots, PLCs, sensors and actuators) and the production management system (*e.g.*, MRP system and POP system).

Among these, the importance of the PLC, which is applied to various areas, has been emphasized, not only as the logic controller but also as the core technology for building FA systems. Therefore, as functions of the PLC, advanced control functions, a user-friendly interface, and network interface functions for communication with sensors and management systems are required.

However, the PLC system is generally a closed system and depends highly on the maker's own technology. This means that the user can use only the functions provided by the maker and the user's own technology and functions cannot be applied. Because of this, whenever the PLC system is changed, the user should be re-trained and the PLC program should be re-programmed. To solve the above-mentioned problem, compatibility and openness of PLC system are necessary. For this, the PLC

system has advanced to become an open PLC system that can meet the following requirements:

1. Portability: The PLC program can be operated and is reusable regardless of PLC system and maker.
2. Connectivity: Communication (data transmission) between PLC systems whose makers are different should be guaranteed.
3. Standardization: The user interface and programming language are unified regardless of system and maker.

For example, the PLC systems and programming languages mentioned in previous sections are not compatible with other systems and languages that other makers provide. Therefore, users should learn the maker's own programming languages. It is also very difficult for third parties to understand and modify PLC programs. In addition, when a new function is added, it is almost impossible to guarantee successful execution within a specified time.

To overcome these problems, the activity for standardizing programming environments for industrial automation equipment was started and the IEC, (International Electrotechnical Commission), established IEC1131-3 in 1993. The standard IEC1131, is the international standard for PLC, consisting of five parts and IEC1131-3 is one of the parts of which IEC1131 is composed.

1. IEC1131-1: PLC General information.
2. IEC1131-2: Equipment and test requirements
3. IEC1131-3: PLC programming language
4. IEC1131-4: User guidelines
5. IEC1131-5: Communications

IEC1131-3 is the international standard for programmable controller programming languages. It specifies the syntax, semantics and display for the following suite of PLC programming languages: 1) Ladder diagram (LD), 2) Sequential Function Charts (SFC), 3) Function Block Diagram (FBD), 4) Structured Text (ST), and 5) Instruction List (IL).

If we use IEC1131-3 to edit a PLC program, it is possible to obtain the following advantages:

1. Because syntax and semantics are unified, it is possible to generate a program that can be operated on all makers' systems and the program can be executed regardless of maker.
2. It is easy to maintain the program.
3. Because the standard supports the structured programming method, any complex program can be edited in easily understandable and structured format and can easily be maintained.
4. Due to the rigorous syntax and semantics it is possible to reduce program error.
5. The standard makes modularization of a program easy and it is possible to increase the efficiency of programming using program modules.

However, the following disadvantages have been identified:

1. Compared with the sequence programming method, the programming procedure is complex due to computer programming.
2. Much effort is needed to understand and know the grammar of the programming language.
3. Due to the rigorous grammar, the flexibility of programming is restricted.
4. Because IEC1131-3 is heavy, it is not appropriate for application in small-sized PLC systems.

IEC1131-3 consists of the configuration model of a PLC system, five programming languages, and the common generality of programming languages.

The software model and communication model address the name and definition of the parts from which a PLC system is composed and the data transmission mechanism between running programs. In the Programming model, not only basic elements such as identifiers, keywords, data types, and variables but also program elements such as functions, function blocks, programs, resources, and tasks are described as the common factors of the programming language.

To understand IEC1131-3, it is necessary to undertake a study of the configuration model, which represents the design concept of a PLC system and includes a software model, communication model, and programming model.

7.5.1 Software Model and Communication Model

In the introductory part of IEC1131-3 the software model is described and represents the PLC system as a controller with multitasking-enabled architecture, as shown in Fig. 7.8.

In the software model,

1. Configuration is the top-most concept that represents the PLC system and includes all the software that is contained in one PLC system.
2. Resource is the element that makes up the configuration and means the functions that a processor board provides. It consists of the software that is needed to execute a PLC program.
3. Program means the logical management unit of a user program and is edited by one of the languages specified by IEC1131-3.
4. Function block is a key concept of IEC1131-3 and makes a program structured and modularized. It is the logical management unit for data transmission and consists of the data for defining input/output parameters and the algorithms for performing specific functions.
5. Task represents how a program or function block works. It begins iteratively or by a specific trigger.

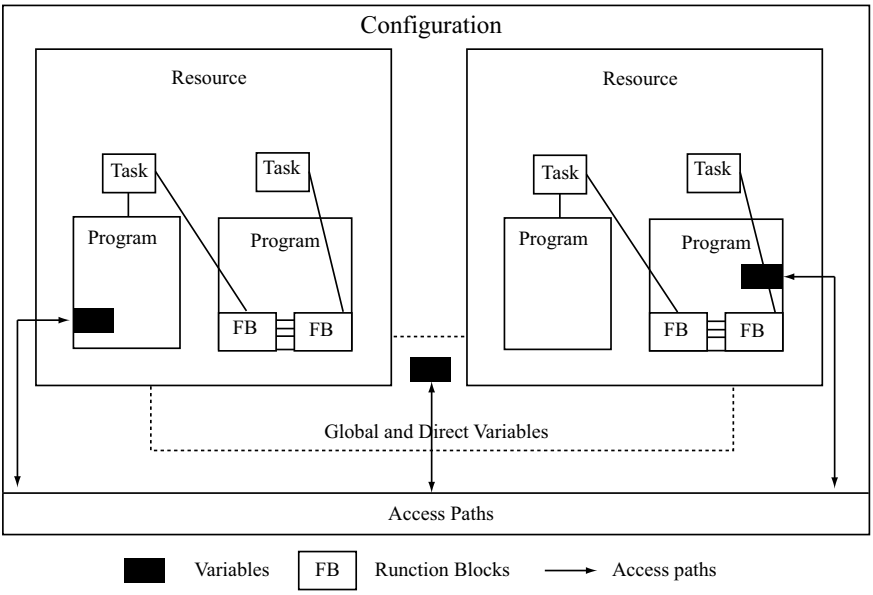


Fig. 7.8 IEC1131-3 Software Model

6. Function is one of the elements of which a program is composed. It is different from the function block, and it denotes the software that generates a single output from a specific input.
7. Access path does not exist in a single resource system. In multiple resource systems it manages the data of elements and the communication between elements. There are various ways to transmit data in a PLC system. In a program, internal variables are used. To input and output the data to the program, function, and function block, input variables and output variables are used. To share resource or configuration information between programs, external variables that specify them are used. In addition, data transmission between configurations is done by the communication object defined by an access path. Using the access path, it is possible to exchange data between the functions and the programs that are located in different resources or configurations.

Comparing PLC systems with the software model in IEC-1131-3, we can regard the controlled system as configuration. Configuration exchanges data or information with other configurations via Access paths (only specific variables can be transmitted via access paths and extended communication functions, as specified in IEC1131-5).

This configuration consists of one or multiple resources and each resource consists of one or multiple tasks. Because of the high functionality of PLC systems, multiple processing is required and the CPU board can be regarded as a resource. Each task executes a program or function block based on regular interrupts or irregular triggers. Consequently, this systematic structure makes it possible to execute a

lot of individual programs synchronously. In addition, resource has a function for supporting the interface between I/O channel and a program.

Furthermore, small-sized PLC systems consist of one processor and one piece of software that controls single operations. A single configuration is formed, even in the case of large-sized PLC systems with multiple processors, a variety of resources (multiple processors) are controlled in real-time. In the case of complicated distributed systems, a system is composed of more than one configuration connected by a network and each configuration can include any resource or program on the network.

As is known from the software model, the key concept of IEC1131-3 is to support the structured programming concept. Actually, by using the task, function, and function block mentioned in the software model, it is possible to change the programming style of the user. Therefore, by using IEC1131-3 it is possible to design a PLC system by distinguishing an iterative task and an interrupt-driven task (or event-driven task). Furthermore, it is possible to divide the iterative tasks into tasks with the same iteration time. In addition, by implementing common tasks or programs as functions or function blocks, it is possible to decrease the program size. Consequently, this structured programming method enables the modularization of programs and this modularization increases the productivity and maintainability of large-sized PLC systems.

7.5.2 Programming Model

The programming model describes the relationship between the common elements of the programming language specified in IEC1131-3. The programming model is based on the concept of derivation and reuse. In other words, the programmer can define new data types from basic data types, new functions or function blocks from basic functions (or standard functions) and can make libraries using them. These libraries can be used not only in an identical system but also in other non-identical systems. The reusability of programs enables advanced programming techniques such as libraries, functions, and function blocks and provides ease and reliability of programming.

As mentioned above, as IEC1131 is the international standard for PLC, it consists of a configuration model and a programming model. To build the standard and open system, it is necessary for the PLC system developer to design the system and the functions based on IEC1131. It is also necessary to design the interpreter for the standard programming language. As well as programmers or system designers, it is also necessary for system users (operators) to understand the key concepts and functions.

7.5.3 User Programming Languages

In IEC1131-3, five programming languages (actually, four languages and one common element) are specified; LD (Ladder Diagram), IL (Instruction List), ST (Structured Text), FBD (Function Block Diagram), and SFC (Sequential Function Chart). The user can select the appropriate language depending on the characteristics of the program.

Actually, SFC is not a programming language for implementing the control program, but the representation tool to depict all sequences of a control program. As shown in Fig. 7.9, SFC classifies continuous tasks into Steps as well as Transitions, which are the conditions for shifting between steps, and Actions, being the job to be performed at a step.

In other words, SFC is composed of multiple steps, the module of a program, an action block associated with a particular step, and a transition to represent the condition for shifting between steps. This graphical representation method is based on Petri-nets or IEC848. SFC can be used not only by itself but also with other program languages specified in IEC1131-3. Therefore, SFC is used as a common element in IEC1131-3. SFC supports not only conditional sequencing but also parallel sequencing where one sequence monitors or executes a background task simultaneously with another performing the main control.

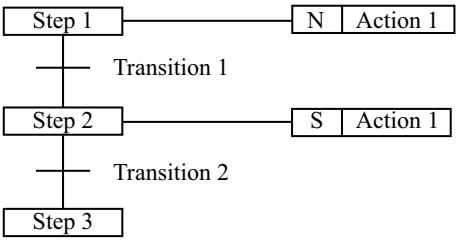


Fig. 7.9 SFC continuous task classification

For effective programming, it is necessary to reuse functions or partial programs. For this, functions, function blocks, and programs are reused in application programs. A function is composed of basic functions such as ADD, ABS, SQRT, SIN, and COS and user-defined functions. A function block contains data and algorithms, as semiconductor hardware, which enables the specific function. It can be reusable in other application programs. In addition, Functions, function blocks, and programs are common elements that can be used in all programming languages specified in IEC1131-3.

Besides SFC, the possibility of powerful data addressing is another common element. It is used to prevent a programmer from substituting (allocating) a different data type to a variable. Boolean type, integer type, byte type, word type, date type, and date-time type are defined as the data types of IEC1131-3. Local variables and global variables can be used as variables in IEC1131-3. A local variable is a variable

that can be used only within software elements where it is defined. Global variables mean variables that can be used over whole software elements. It is also possible to define direct pointer variables that refer directly to memory locations.

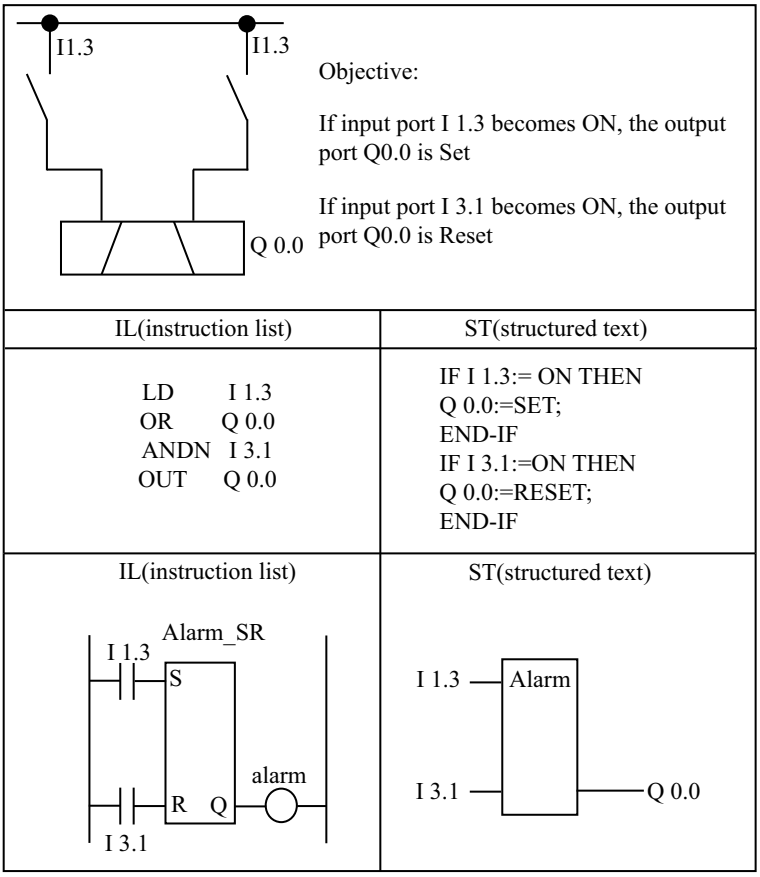


Fig. 7.10 The PLC languages specified in IEC1131-3

The key characteristics of the four languages specified in IEC1131-3 are summarized in Fig. 7.10 with the above-mentioned common elements. For convenience of understanding, the program that sets or resets the output port depending on the type of input is edited by four languages and the basic format of each language will be described.

Among the four languages, IL (Instruction List) and ST (Structured Text) are textual languages, while the LD (Ladder Diagram) and FBD (Function Block Diagram) are graphical languages.

1. IL, which is widely used in Europe, is a low-level language like assembler. The advantage of IL is that it is adequate for small-sized programs thanks to simple se-

quences and logic. It can be used for describing transitions in SFC with function, function block, and program sequences.
2. As ST is a high-level language, it is related to Ada, Pascal, and C. By using ST, it is possible to allocate values to variables and describe tasks by wiring functions, function blocks, and program blocks. Conditional branching and iteration loops are possible too. It can also be used to describe steps, action blocks, and transitions. In addition, it is adequate for numerical calculation and defining complex function blocks.
3. LD is a well-known language. A program is composed of a combination of input and output contacts. LD is used not only for editing the control program but also monitoring the output/input contacts while a program is being executed. In addition, with functions, function blocks, and programs, it can be used for describing the results of transitions in SFC.
4. FBD is a graphic-based language. It is largely used for programming signal flow between control blocks because of its easy understanding of the flow of a program. FBD is similar to electric circuits including the signal flow of process control. It can be used for describing the behavior by wiring functions, function blocks, and program blocks and describing the step, action block, and transition in SFC.

7.6 Soft PLC

In the late 1960s, after GE powertrain introduced the concept of the PLC system that replaced the relay board, PLC systems that control a variety of processes by using simple sequence programs has been widely used for 40 years.

However, as the controlled systems have become more complex, faster, and larger, the demand for openness and standardization of PLC systems has increased. To meet this requirement, the PLC system has been changed from a hardware-based system to a software-based system. Consequently, Soft PLC systems, which operate from personal computers and enable logical sequence control functions in real time, were introduced. With the advancement of the PCs performance, Soft PLC systems have come to provide not only conventional sequence control functions but also easy user interfaces, network communication functions, and advanced functions for factory automation. In Soft PLC systems, the basic and advanced functions of PLC and communication functions are executed by one processor module, except for input and output modules. It is possible to make a standardized PLC system based on the software model and programming languages specified in IEC1131-3.

Figure 7.11 shows an example of a Soft PLC that has been applied to the transfer line in Ford Motors. In this system, the interface board that can be connected to various I/O devices (*e.g.*, AB1771, Seriplex, OPTO 22) is built into a PC that contains the Soft PLC system. This system is a good example of a Soft PLC system that satisfies the openness requirement by using PC hardware.

To develop a soft PLC system, real-time operation and reliability of response, which are key requirements for industrial PLC systems, should be guaranteed. Ba-

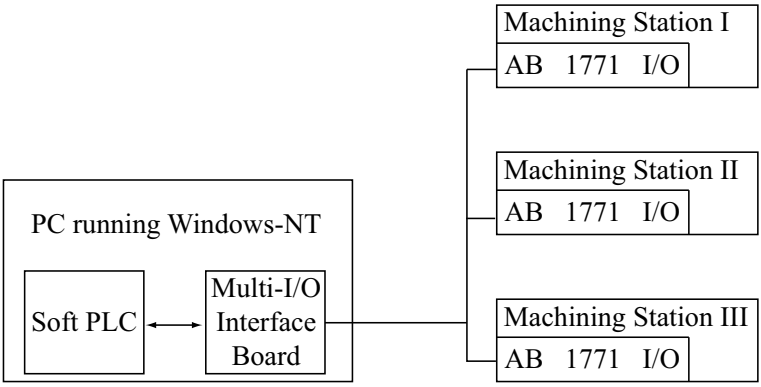


Fig. 7.11 Soft PLC automotive transfer line

sically, though, PC operating systems cannot satisfy these. However, the non real-time property of DOS or Windows OS can be overcome by various methods and the method of designing a soft PLC system will be described together with design of Soft-NC in the later in this textbook. In Soft-NC, a PC is used as the hardware platform and all CNC functions including PLC functions are implemented in software. In this point, Soft PLC is very similar to Soft-NC. Furthermore, Soft NC includes more functions than Soft PLC, used for NCK control and MMI. Therefore, if NCK functions and MMI functions are omitted or simplified from Soft NC, Soft NC and Soft PLC can be regarded as the same system. Soft PLC which is made by a user interface and the PLC kernel based on the IEC1131-3 can be regarded as partial systems of Soft-NC.

Figure 7.12 shows the open CNC system of MDSI. The figure shows that the CNC system consists of Interpreter (NC Code Parser), Servo controller (Servo Algorithm), Interpolator (Path/trajectory Planning) for NCK, user interface for MMI, Soft PLC for PLC, and APIs for external users. This model shows that Soft PLC is one of the software modules from which a CNC system is composed. If you want to know more details about the hardware configuration and software of a Soft PLC, please read other references about Soft-NC and Open CNC systems.

7.7 PLC Configuration Elements

In this section, the configuration and execution structure of a PLC system will be addressed from the system designer's point of view. Also, implementation of the PLC program executor will be shown.

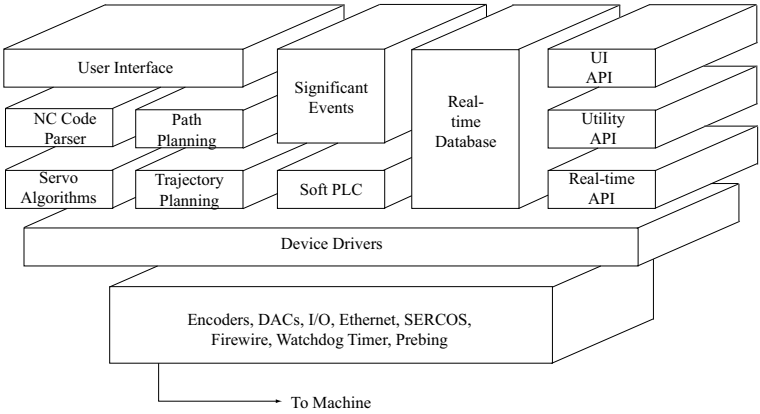


Fig. 7.12 Open CNC system of MDSI

7.7.1 PLC System Functions

The execution environment and main functions of a PLC system can be summarized as in Fig. 7.13. The PLC program, which is interpreted and executed by the CPU module, is edited by an external PLC programmer. The PLC programmer consists of the editor module, compiler module, and communication module. The editor module is used for editing the program, the compiler module is used for translating the edited program into a CPU-understandable language, and the communication module is used for transmitting the PLC program into the CPU module. In addition, the monitoring module is used for sending the PLC status to the PLC programmer and displaying the PLC status.

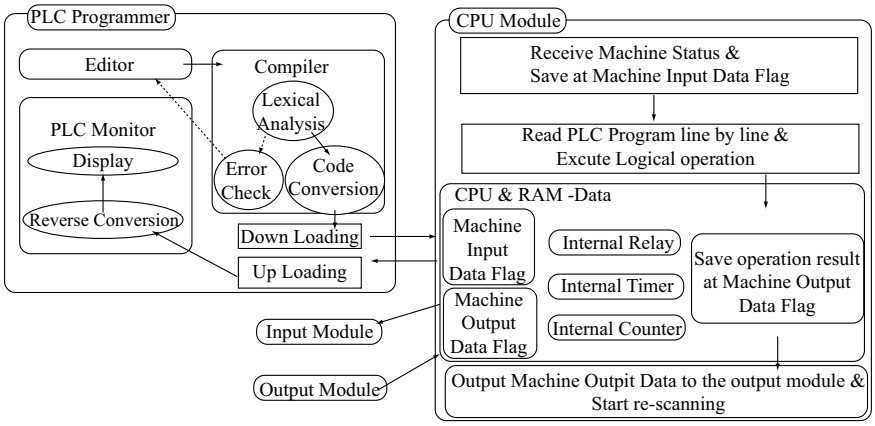


Fig. 7.13 Execution environment and main functions of a PLC system

The PLC programmer should have a program editor that provides an easy user-interface and supports PLC programming languages such as Ladder, Mnemonic, and Function block. The specification of the editor is given in IEC1131-3.

The PLC programmer can be implemented as one of two types; one is the 'interpreter type' and the other is the 'compiler type'. In the interpreter type, a native program, which a user edits using some particular procedure, is interpreted line by line and executed whenever it is needed. In the compiler type, a native program is converted once to a CPU-understandable binary or hexadecimal file and, whenever necessary, the compiled file is sent to the CPU module and executed.

Typically, a PLC programmer is implemented by the compiler type in order to realize high-speed execution. Let's see the procedure of a compiler. A compiler reads one block of a native program line by line and divides the lines into elements from which the block is composed.

As a sentence of a natural language consists of multiple words, a block of a program is composed of tokens (*e.g.*, constant values, variables, operators, and keywords).

The compiler recognizes the tokens and finds the meaning of the words by analyzing the tokens, syntax, and schema. The compiler searches for relationships between tokens. The compiler generates intermediate code based on the previous result. The compiler generates optimized code from the intermediate code for effective execution. Finally, it generates the instruction code.

A compiler is the software that converts a native program into CPU-understandable code. It provides the functions that transmit compiled code to the CPU module and monitors the execution status of the PLC system.

The main task of the CPU module is to read and execute a binary program from the PLC programmer. We call this module the 'executor' and how the CPU module works is as follows. Firstly, the CPU module receives the machine status from the input port and stores it to the machine input data flag. In the next step, it reads the PLC program (binary code) line by line and carries out the logical operations. At this moment, the internal relay, timer, counter, and input data flag are referenced. The result from one line execution is saved to the machine output data flag and, finally, the values from the machine output data flag are sent to the output port for actuating the relay, solenoid, and user interface.

If one scan like that mentioned above has been completed, the executor repeats the scan by reading the input ports (or address). Therefore, for designing the PLC system, it is necessary to understand the behavior of this executor.

In Table 7.3, the set of PLC programming instruction is summarized together with a description of the instruction set, mnemonic format, and function description. This instruction set consists of the basic instruction set for the editing sequence and the functional instruction set for describing tasks that are difficult to describe using only basic instruction sets. The functional instruction set includes sequence control, timer/counter, arithmetic operations.

Table 7.3 PLC programming commands and functions

Command	Mnemonic	Function
Read	RD X1.0	Read the specific address.
Read Not	RND X2.1	Read the specific address and reverse the value.
Write	WR Y1.0	Write the operation result to the specified address.
Write Not	WRN Y2.0	Reverse the operation result and write the reversed result to the specified address.
And	AND X3.2	Perform logical product between the value of the specified address and the value on the stack register and store the result on the stack.
And Not	ANDN Y0.2	Reverse the value of the specific address, perform logical product between the reversed value and the value on the stack register, and store the result on the stack.
Or	OR Y2.7	Perform logical sum between the value of the specified address and the value on stack register and store the result in stack.
Or Not	ORN G2.1	Reverse the value of the specified address, perform logical sum between the reversed value and the value on the stack register, and store the result on the stack.
Read Stack	RDS Y3.1	Shift the values of the stack to the left and store the value of the specified address in bit 0 of stack.
Read Not Stack	RDNS R1.7	Shift the values of the stack to the left and store the reversed value of the specific address in bit 0 of stack.
And Stack	ANDS	Perform logical product between the values of the lower two bits, save the result in the first bit of stack, SR1, and shift the values of stack to the right.
Or Stack	ORS	Perform logical sum between the values of lower two bits, save the result in the first bit of stack, SR1, and shift the values of stack to the right.

Table 7.3 (continued)

End of P/G	END	Terminate the program.
Timer	TMR #5, 4000	When the input signal is ON, the timer with specified number is executed during the specified time.
Counter	CTR #1, 100	Whenever the input signal is changed to ON from OFF, the counter with the specified number is actuated. When the count reaches the specified number, the counter output maintains ON status before receiving the reset signal.
Move	MOV 7, X1.2	Copy the left operand to the right operand.
And Move Y2.3	ANDM 1, 3,	When the input signal is ON, performs logical product between the values of left operand and middle operand and finally saves the result in the right operand.
Or Move X1.1	ORM 3, 7,	When the input signal is ON, performs a logical sum between the values of the left operand and the middle operand and finally saves the result in the right operand.
Call	CALL Rosa	When the input signal is ON, call the subroutine with the specified name.
Subroutine	SBRT Stephy	Start the sub routine.
Return	RET	Terminate the sub routine.
Jump	JMP Khang	When the input signal is ON, jump to the program part starting with the specified label.
Equal	EQU 5, X1.0	When the input signal is ON, if the value of the left operand and the value of the right operand are equal, set the specified bit as ON. Otherwise, set the specified bit as OFF.
Greater Than	GT Y1.1, 10	When the input signal is ON, if the value of the left operand is greater than that of the right operand, set the specified bit as ON. Otherwise, set the specified bit as OFF.
Less Than	LT 10, X4.0	When the input signal is ON, if the value of the left operand is less than that of the right operand, set the specified bit as ON. Otherwise, set the specified bit as OFF.

Table 7.3 (continued)

Shift Right	SFTR X1.0,5	When the input signal is ON, shift the value of the left operand to the right as many bits as given by the right operand.
Shift Left	SFTL Y1.1, 3	When the input signal is ON, shift the value of the left operand to the left as many bits as given by the right operand.
Addition	ADD 1, 2, X1.7	When the input signal is ON, add the values of the left operand and the middle operand and store the result to the right operand.
Subtraction	SUB 3, 1, Y3.1	When the input signal is ON, subtract the values of the left operand and the middle operand and store the result to the right operand.
Multiply	MUL 2, 5, X1.3	When the input signal is ON, multiply the values of the left operand and the middle operand and store the result to the right operand.
Division	DIV 4, 2, Y2.2	When the input signal is ON, divide the value of the left operand by the value of the middle operand and store the result to the right operand.
Inverse	INV X1.1, 5	When the input signal is ON, reverse the bit of the address specified by the left operand as specified by the value of the right operand.

Since the above functions denote how a sequence program (logic program) is interpreted and executed in the program executor, an in-depth understanding of them is needed to design a PLC program executor.

7.7.2 Executor Programming Sequence

To describe the sequence execution function of a PLC executor, let's use the program that was edited by the ladder diagram or mnemonic shown in Table 7.4.

The program that the user edited in a ladder diagram is converted into mnemonic form to be executed by an interpreter-type executor. Alternatively, the user can edit the program directly in mnemonic form. The executor that is designed in this textbook handles the operations via a stack register. During execution of the mnemonic program, the operation results are temporarily stored in a stack register and the operations continue by using the values from the stack register. For example, the result

of the current operation is stored in SR0, the zero bit of the stack register. Through continuous operations, the data in SR0 shifts to SR1 and a new value is stored on SR0. Therefore, the previously stored value is shifted to the right and is saved by the RDS command. To recall the value saved by the RDS, the ANDS command is utilized.

As a practical example, when the executor performs the tasks from line number 10 to line number 30, the behavior of the executor is as follows.

1. The executor reads address X0.1 using the Read command and carries out the logical operation and saves the the result of the operation in SR0.
2. It reverses the logic status of address Y2.1 and executes the AND operation (logical product) between the serial-connected previous operation result and the reversed status of Y2.1. The operation result is saved in SR0.
3. To execute parallel-connected instructions, the value of SR0 is shifted to SR1 using the RDS command and the value of address Y0.1 is stored in SR0.
4. It executes the AND operation (logical product) between the value of address X1.1 and the value of SR0 and the result is saved in SR0.
5. OR operation (logical sum) between SRB0 and SRB1 is executed and the result is stored in SR0.
6. Finally, the value of SR0 is output to address Y4.3.

Subsequently, reading the status of the input port, executing the logical operation by using the status via stack register and storing the result to the output port are repeated.

7.7.3 Executor Implementation Example

In this section, a real implementation example of the executor capable of handling the commands shown in Table 7.3 will be described. As an example, the programs for the basic instruction sets are described. The programs for the functional instruction set including comparison function of the values of two registers, and the four fundamental arithmetic operations can be achieved by extending the method shown in the basic instruction cases.

7.7.3.1 Stack Register

The stack register saves the temporary operation result. It is assumed to be composed of 16 bits as follows:

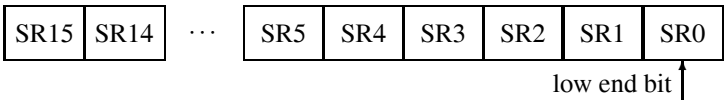
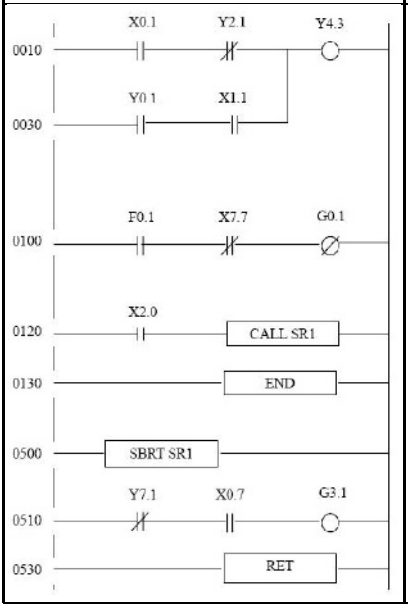


Table 7.4 Comparison between Ladder diagram and mnemonic programming

Ladder diagram	Mnemonic
	1 RD X0.1 2 ANDN Y2.1 3 RDS Y0.1 4 AND X1.1 5 ORS 6 WR Y4.3 7 8 RD F0.1 9 ANDN X7.7 10 WRN G0.1 11 12 RD X2.0 13 CALL SR1 14 END 15 SR1 16 RDN Y7.1 17 AND X0.7 18 WR G3.1 19 RET

In order to save the result of previous operation, the values of whole bits of the stack register are shifted to the left. When the recall command is invoked for recalling the stored value, the values of whole bits of the stack register are shifted to the right, and the value shifted last comes first.

```
class CPLCStack
{
public:
    CPLCStack();
    virtual ~CPLCStack();
 private:
    bool m_stack[STACKSIZE];
 public:
    void RD(char simbol, int upperadd, int loweradd);
    void RDN(char symbol, int AddNum, int BitNum);
    void WR(char symbol, int AddNum, int BitNum);
    void WRN(char symbol, int AddNum, int BitNum);
    void AND(char symbol, int AddNum, int BitNum);
    void ANDN(char symbol, int AddNum, int BitNum);
    void OR(char symbol, int AddNum, int BitNum);
    void ORN(char symbol, int AddNum, int BitNum);
    void RDS(char symbol, int AddNum, int BitNum);
```

```
    void RDNS(char symbol, int AddNum, int BitNum);
    void ANDS(char symbol, int AddNum, int BitNum);
    void ORS(char symbol, int AddNum, int BitNum);
public:
    void LShift(int i);
    void RShift(int i);
};
void CPLCStack  LShift(int i)
{
    for(int j = STACKSIZE − 1;j >= i;j − −)
        m_stack[j] = m_stack[j-i];
}
void CPLCStack::RShift(int i)
{
    for(int j = 0;j < STACKSIZE − i;j + +)
    m_stack[j] = m_stack[j+i];
}
```

7.7.3.2 Basic Command

The basic commands are mainly used for building the sequence program. They carry out one-bit operations and consist of twelve commands. To order basic commands they are represented as follows:
(Command [Address name][Address number].[bit number]]
(Note that the ADNS and ORS commands are used without any arguments).

a) RD (READ)

- This command is to read the value of the specified address and save the value read in SR0.
- It is used for A-type switch.
- Program structure

- Ladder Diagram

- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X01.0						①
2	WR	Y00.0	W1 Output					①

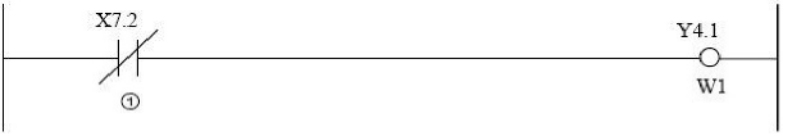
void CPLCStack::RD(char symbol, int AddNum, int BitNum)

```
{
    CSoftPLCDoc* pDoc = GetDoc();
        int index;
        bool value;
        switch (symbol) {
            case 'X':
                // Get the logical status of the particular address.
                index = AddNum*8 + BitNum;
                value = pDoc− >XAddress[index];
                break;
            case 'Y':
                // Get the logical status of the particular address.
                index = AddNum*8 + BitNum;
                value = pDoc− >YAddress[index];
                break;
            default:
                break;
              }
        m_stack[0] = value; // Save the status of SR0
}
```

b) RDN (READ NOT)

- This command is to read the value of the specified address, reverse the value, and save the reversed value to SR0.
- This is used for B-type switch.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X7.2						①
2	WR	Y4.1	W1 Output					①

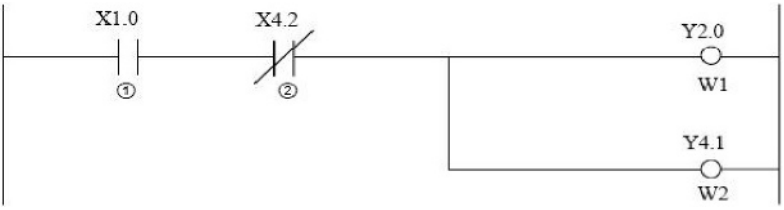
```
void CPLCStack::RDN(char symbol, int AddNum, int BitNum)
{
        CSoftPLCDoc* pDoc = GetDoc();
        int index;
        bool value;
        switch (symbol) {
            case 'X':
```

```
                // Get the logical status of the particular address.
                index = AddNum*8 + BitNum;
                value = pDoc− >XAddress[index];
                break;
            case 'Y':
                // Get the logical status of the particular address.
                index = AddNum*8 + BitNum;
                value = pDoc− >YAddress[index];
                break;
            default:
                break;
        }
        m_stack[0] = !value; // Save the reversed status of SR0
}
```

c) WR (WRITE)

- After completing logic operation, it commands to output the value of SR0 to the specific address.
- The logic operation result can be output to more than one address.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X1.0						①
2	ANDN	Y4.2						①*②
3	WR	Y2.0	W1 Output					①*②
4	WR	Y4.1	W2 Output					①*②

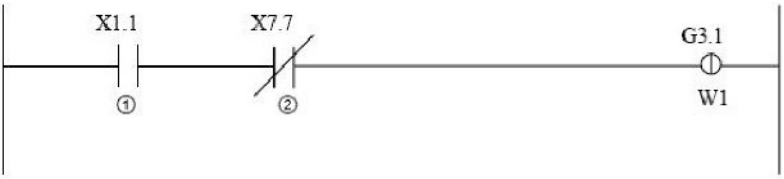
```
void CPLCStack::WR(char symbol, int AddNum, int BitNum)
{
        CSoftPLCDoc* pDoc = GetDoc();
        int index;
        switch (symbol) {
            case 'X':
                // Output the logical status to the specific address.
```

```
                index = AddNum*8 + BitNum;
                pDoc− >XAddress[index] = m_stack[0];
                break;
            case 'Y':
                // Output the logical status to the specific address.
                index = AddNum*8 + BitNum;
                pDoc− >YAddress[index] = m_stack[0];
                break;
            default:
                break;
        }
}
```

d) WRN (WRITE NOT)

- After completing logical operation, this commands reversal of the value of SR0 and output of the reversed value to the specified address.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X1.1						①
2	ANDN	X7.7						①*②
3	WRN	G3.1	W1 Output					!(①*②)

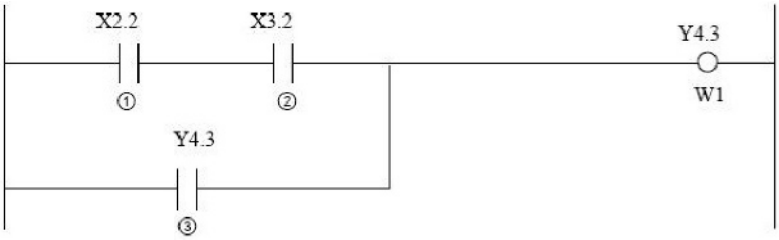
```
void CPLCStack::WRN(char symbol, int AddNum, int BitNum)
{
        CSoftPLCDoc* pDoc = GetDoc();
         int index;
         switch (symbol) {
            case 'X':
                // Output the reversed logical status to the specific address.
                index = AddNum*8 + BitNum;
                pDoc− >XAddress[index] = !m_stack[0];
                break;
            case 'Y':
                // Output the reversed logical status to the specific address.
```

```
                index = AddNum*8 + BitNum;
                pDoc− >YAddress[index] = !m_stack[0];
                break;
            default:
                break;
        }
}
```

e) AND (AND)

- It commands to perform the AND operation (logical product) between the values of the specified address and SR0 and save the operation result to SR0.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X2.2						①
2	AND	X3.2						①*②
3	OR	Y4.3						①*②+③
4	WR	Y4.3	W1 Output					①*②+③

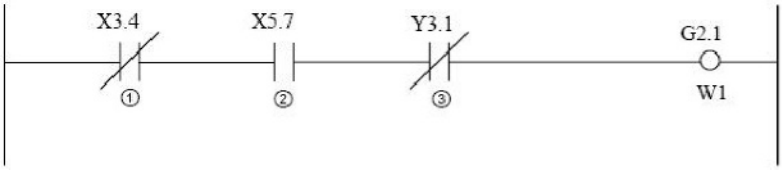
```
void CPLCStack::AND(char symbol, int AddNum, int BitNum)
{
        CSoftPLCDoc* pDoc = GetDoc();
         bool state;
        int index = AddNum*8 + BitNum;
         switch (symbol) {
            case 'X':
                // Get the logical status of the particular address.
                state = pDoc− >XAddress[index];
                break;
            case 'Y':
                // Get the logical status of the particular address.
                state = pDoc− >YAddress[index];
                break;
```

```
            default:
                break;
        }
        if(m_stack[0] && state)
            m_stack[0] = true;
        else
            m_stack[0] = false;
}
```

f) ANDN (AND NOT)

- This commands reversal of the value of the specific address, execute the logic product with the value of SR0, and save the result on SR0.
- Program structure.

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X3.4						①
2	AND	X5.7						①*②
3	ANDN	Y3.1						①*②*③
4	WR	G2.1	W1 Output					①*②*③

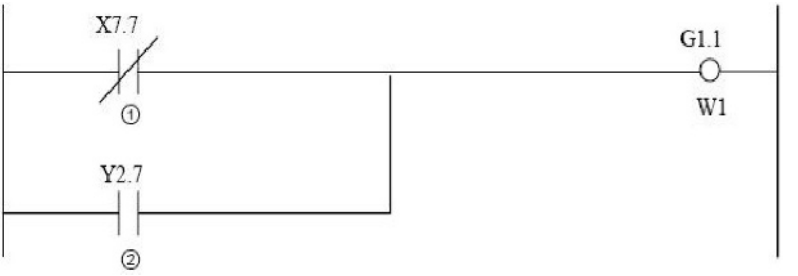
```
void CPLCStack::ANDN(char symbol, int AddNum, int BitNum) {
        CSoftPLCDoc* pDoc = GetDoc();
        bool state;
        int index = AddNum*8 + BitNum;
        switch (symbol) {
            case 'X':
                // Get the logical status of the particular address.
                state = pDoc->XAddress[index];
                break;
            case 'Y':
                // Get the logical status of the particular address.
                index = AddNum*8 + BitNum;
                state = pDoc->YAddress[index];
                break;
```

```
            default:
                break;
        }
        if(m_stack[0] && !state)
            m_stack[0] = true;
        else
            m_stack[0] = false;
}
```

g) OR (OR)

- It commands to execute OR operation (logical sum) between the values of the specific address and SR0 and save the result on SR0.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X7.7						①
2	OR	Y2.7						①+②
3	WR	G1.1	W1 Output					①+②

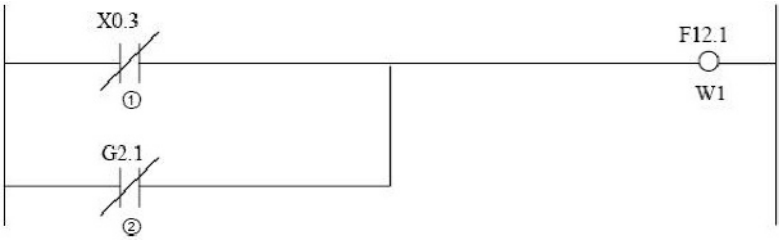
```
void CPLCStack::OR(char symbol, int AddNum, int BitNum)
{
        CSoftPLCDoc* pDoc = GetDoc();
         bool state;
        int index = AddNum*8 + BitNum;
        switch (symbol) {
                // Get the logical status of the address.
            case 'X':
                state = pDoc->XAddress[index];
                break;
            case 'Y':
                state = pDoc->YAddress[index];
```

```
            break;
        default:
            break;
    }
     if(m_stack[0] || state)
        m_stack[0] = true;
    else
        m_stack[0] = false;
}
```

h) ORN (OR NOT)

- This command is to reverse the value of the specific address, perform OR operation (logical sum) with the value of SR0, and save the result on SR0.
- Program structure.

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X0.3						①
2	ORN	G2.1						①|②
3	WR	F12.1	W1 Output					①+②

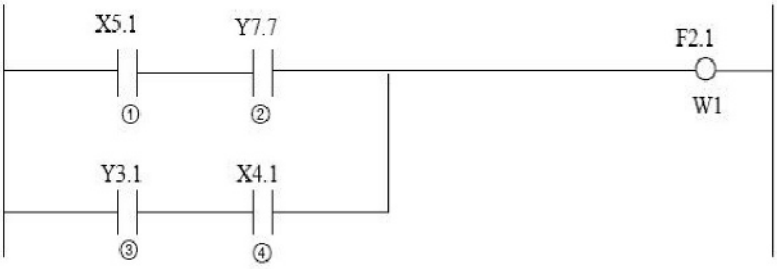
```
void CPLCStack::ORN(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
     bool state;
    int index = AddNum*8 + BitNum;
     switch (symbol) {
            // Get the logical status of the address.
        case 'X':
            state = pDoc->XAddress[index];
            break;
        case 'Y':
            state = pDoc->YAddress[index];
```

```
            break;
        default:
            break;
    }
     if(m_stack[0] && !state)
        m_stack[0] = true;
    else
        m_stack[0] = false;
}
```

i) RDS (READ STACK)

- This command is to shift the values of stack register to left bit and set the value of the specific address in SR0.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR2	SR1	SR0
1	RD	X51					①
2	AND	Y7.7					①*②
3	RDS	Y3.1				①*②	③
4	AND	X4.1				①*②	③*④
5	ORS						①*②+③*④
7	WR	F2.1	W1 Output				①*②+③*④

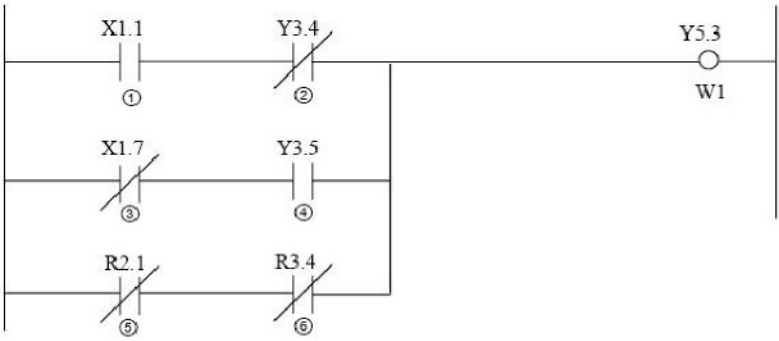
```
void CPLCStack::RDS(char symbol, int AddNum, int BitNum)
{
        CSoftPLCDoc* pDoc = GetDoc();
        bool state;
        int index = AddNum*8 + BitNum;
         switch (symbol) {
                // Get the logical status of the address.
            case 'X':
                state = pDoc− >XAddress[index];
```

```
            break;
        case 'Y':
            state = pDoc->YAddress[index];
            break;
        default:
            break;
        }
        LShift(1);
        m_stack[0] = state;
}
```

j) RDNS (READ NOT STACK)

- This command is to shift the values of stack register to the left bit, reverse the value of the specific address, and set the result on SR0.
- Program structure

- Ladder diagram



- Coding sheet and operation result

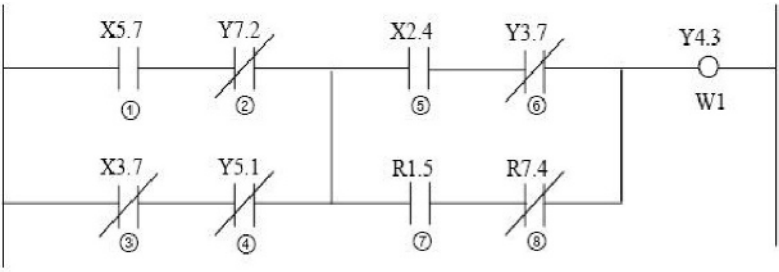
Number	Command	Address	Comment		SR1	SR0
1	RD	X1.1				①
2	ANDN	Y3.4				①*②
3	RDNS	X1.7			①*②	③
4	AND	Y3.5			①*②	③*④
5	ORS					①*②+③*④
7	RDNS	R2.1			①*②+③*④	⑤
7	ANDS	R3.4			①*②+③*④	⑤*⑥
8	ORS					①*②+③*④+⑤*⑥
9	WR	Y5.3	W1 Output			①*②+③*④+⑤*⑥

```
void CPLCStack::RDNS(char symbol, int AddNum, int BitNum)
{
        CSoftPLCDoc* pDoc = GetDoc();
        bool state;
        int index = AddNum*8 + BitNum;
         switch (symbol) {
                // Get the logical status of the address.
            case 'X':
                state = pDoc− >XAddress[index];
                break;
            case 'Y':
                state = pDoc− >YAddress[index];
                break;
            default:
                break;
        }
         LShift(1);
        m_stack[0] = !state;
}
```

k) ANDS (AND STACK)

- This command is to execute the AND operation (logical product) between the values of SR0 and SR1 and shift all the values of the stack register to the right.
- Program structure

- Ladder diagram



- Coding sheet and operation result

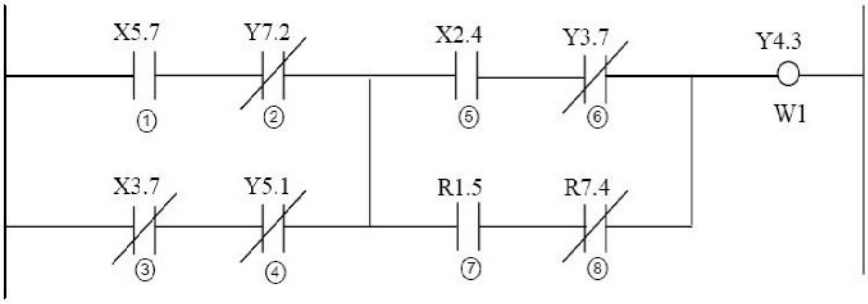
Number	Command	Address	Comment		SR2	SR1	SR0
1	RD	X5.7					①
2	ANDN	Y7.2					①*②
3	RDNS	X3.7				①*②	③
4	AND	Y5.1				③*④	③*④
5	ORS						①*②+③*④
7	RDS	X2.4				①*②+③*④	⑤
7	AND	Y3.7				①*②+③*④	⑤*⑥
8	RDS	R1.5			①*②+③*④	⑤*⑥	⑦
9	ANDN	R7.4			①*②+③*④	⑤*⑥	⑦*⑧
10	ORS					①*②+③*④	⑤*⑥+⑦*⑧
11	ANDS						(①*②+③*④)* (⑤*⑥+⑦*⑧)
12	WR	Y4.3	W1 Output				

```
void CPLCStack::ANDS(char symbol, int AddNum, int BitNum)
{
        if(m_stack[1] && m_stack[0])
            m_stack[1] = true;
        else
        m_stack[1] = false;
         RShift(1);
}
```

l) ORS (OR STACK)

- This command carries out the logical summation of SR0 and SR1 and sets the result to SR1. It also shifts the value stack register one place to the right.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR2	SR1	SR0
1	RD	X5.7					①
2	ANDN	Y7.2					①*②
3	RDNS	X3.7				①*②	③
4	AND	Y5.1				①*②	③*④
5	ORS						①*②+③*④
7	RDS	X2.4				①*②+③*④	⑤
7	AND	Y3.7				①*②+③*④	⑤*⑥
8	RDS	R1.5			①*②+③*④	⑤*⑥	⑦
9	ANDN	R7.4			①*②+③*④	⑤*⑥	⑦*⑧
10	ORS					①*②+③*④	⑤*⑥+⑦*⑧
11	ANDS						(①*②+③*④)*(⑤*⑥+⑦*⑧)
12	WR	Y4.3	W1 Output				

```
void CPLCStack::ORS(char symbol, int AddNum, int BitNum)
{
        if(m_stack[1] || m_stack[0])
            m_stack[1] = true;
        else
        m_stack[1] = false;
         RShift(1);
}
```

As mentioned above, the PLC executor performs the bit operations by using stack registers and, therefore, the execution time is very short. In general, it takes several tens of milliseconds to execute a PLC program with hundreds of lines. Depending on the performance of the PLC processor, the time for execution can be even shorter.

7.8 Summary

A PLC system consists of a programming tool that is used for editing and loading a PLC program, Input unit, Output unit, processor unit, memories, and auxiliary units. AC and DC can be used for the input signal and output signal of a PLC system. Various inputs and outputs, such as On/Off signals and timers/counters, can be used.

Textual language such as mnemonic and graphical languages such as the ladder diagram are used as PLC programming languages. Each programming language has a different structure and command list depending on PLC makers. This makes it impossible to exchange PLC programs between different systems. In order to overcome

this problem, IEC1131, the international standard for PLC systems, was established. The programming languages specified in IEC1131-3 have come to be widely used.

To satisfy the openness and compatibility of PLC systems, hardware-based PLC systems have come to be replaced by software-based PLC, the so-called Soft PLC system. A Soft PLC system is regarded as a software-oriented PLC system that is based on PC hardware.

The behavior of the executor, being the key module of PLC system, is as follows. First, the PLC processor reads the input contacts and saves the values in the appropriate input memory. Next, the PLC processor executes the operation and stores the operation result in the output memory. Finally, the PLC processor sends the values from the output memory to the output module. Consequently, the PLC executor plays the role of performing bit operations based on the data in input memory according to the PLC program and saving the result in the output memory.

Chapter 8
Man–Machine Interface

The Man–Machine Interface (MMI) provides the interface that enables a user to operate a machine tool, edit a part program, perform the part program, set the parameters, and transmit data. In this chapter, the function and components of the MMI will be addressed, and programming methods such as CAPS (Conversational Automatic Programming System) will be described. In addition, for designing CAPS, the main functions and components of CAPS will be described.

8.1 MMI Function

In order for a user to operate a machine effectively and to use the function of the machine optimally, it is necessary to design the operation panel for usability according to the machine–tool characteristics. In other words, an operation panel should be designed from the point of view of ergonomics, operation error prevention, key grouping and key allocation for specific machine tools with regard to user convenience. Figure 8.1 shows a typical operation panel and, in general, the operation panel can be divided into four areas.

8.1.1 Area for Status Display

This area displays the machine status and NC parameters. It provides the graphical user interface (GUI) for interaction between the CNC and the user. Figure 8.2 shows a typical display of this area and the functions related to the numbers shown in Fig. 8.2 are as follows.

1. *Machining information*: Displaying information related to the current machine status including the coordinates of machine tools, current part program, cutting tools and machine parameters.

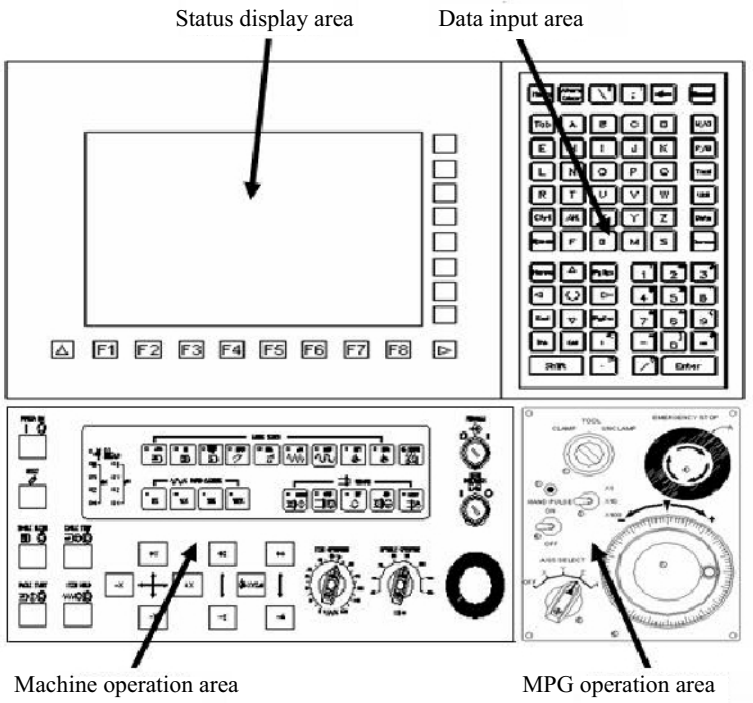


Fig. 8.1 Typical operation panel

2. *Operation Mode*: Displaying the operation modes of machine tools, such as zero position return mode, JOG mode, Automatic mode and MDI mode.
3. *Program name*: Displaying the name of the program that is currently loaded in the memory for machining.
4. *Alarm window*: Displaying the warning and alarm messages.
5. *Key input window*: Displaying the strings that are typed by a user.
6. *Window for displaying user interface relevant to operation mode and function*:

   - *Machining status (POS)*: operation status such as axis position, spindle speed, feedrate, modal G-codes, and tool number is displayed by this function.
   - *Program (PROG)*: the GUI for editing a part program, managing the program folders, graphical simulation, and CAPS is provided by this function.
   - *Tool management*: the GUI for managing tool compensation, tool life, and tool offset is provided by this function.
   - *Parameter and system*: the GUI for managing the NC parameters, system parameters for servo and spindle is provided.
   - *Auxiliary application*: the GUI for monitoring PLC, displaying alarms, performing DNC, and compensating pitch error is provided.

7. *Function keys*: these keys are horizontally placed in the bottom or vertically on the right-hand side of the display and are mapped to the particular functions. Therefore, to effectively design the menu structure, it is important to classify the functions into the appropriate group and enable the necessary keys to be displayed in one display. It is necessary to consider that the number of hierarchical layers increases if CNC functions are grouped and are designed as a hierarchical structure. Therefore, if the user wants to select a particular menu at the bottom of the hierarchical structure, the user has to select a sequence of menus from the top menu to the bottom menu. Also, the user has to remember the hierarchical structure and the menus located in each layer. This problem makes the user interface inefficient.

   To overcome this problem, it is necessary to design a ring menu structure of menu trees where, by selecting the displayed menu tree, the user can carry out the desired task from the function keys displayed on one screen as much as possible and each function keys is connected with the various modes. In this type of menu structure it is not necessary to remember the menu structure. However, the menu structure may be inconsistent and many function keys may be required.

8.1.2 Area for Data Input

As this area is the keyboard for inputting user data to the CNC system, it consists of alphanumerical input buttons and hot keys for executing the functions of CNC.

8.1.3 Area for MPG Handling

This area consists of the MPG (Manual Pulse Generator), the MPG handle ON/OFF switch and the feed ratio selection key that are used for the user to move each servo axis manually. In addition, the Chuck CLAMP/UNCLAMP key for manually loading and unloading tools to the spindle and the emergency stop button are located in this area.

8.1.4 Area for Machine Operation

This area consists of many kinds of switch and lamp that provide various functions as follows.

1. *Mode selection switch*: for selecting Auto mode, MDI mode, Teach-In mode, Return mode, JOG mode, Handle mode, Incremental Moving mode, and Rapid Moving mode.

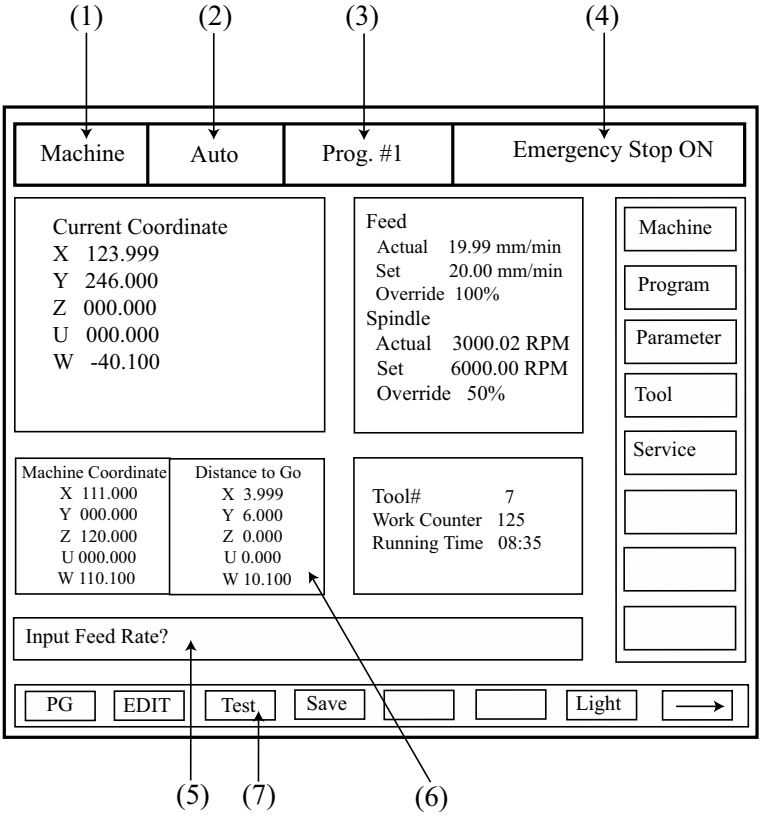


Fig. 8.2 Typical machine status and NC parameters display area

2. *Rapid Override button*: by using this button, rapid feed can be adjusted in scale to 10%, 50%, and 100%.
3. *Feed override switch*: by using this switch, the commanded feedrate can be adjusted from 10% to 150%.
4. *Spindle speed override switch*: using this switch, the commanded spindle speed can be adjusted from 50% to 150%.
5. *Spindle handling buttons*: these buttons consist of the spindle start button, the spindle stop button, rotation direction selection button, and the spindle orientation button, inverse. These buttons are used in MDI mode.
6. *Cycle Start button*: This button is used for starting the auto-execution or resuming the execution of a part program during feed hold state.
7. *Feed Hold button*: This button is used for temporarily stopping the axis movement in automatic machining. When the button is pushed, the spindle continues to rotate. If any axis of the machine tool is moving, that axis is stopped after deceleration.

8. *Single Block Button*: Single block execution means that in auto mode or MDI mode, the execution of a part program is stopped after the execution of one block has been completed and the next block begins only after the Cycle Start button has been pushed. The single block button turns on or off single block execution mode. If this button is ON during the execution of a part program, the CNC system goes into the idle state after completing the executed block. If this button is OFF, the remaining blocks are executed.
9. *Zero return button*: This button is used for making each axis return to the zero position. All axes can be returned to the zero position simultaneously. Feed override is validated during zero return.
10. *Emergency Stop button*: This button is used for stopping the machine in an abnormal state as soon as possible.
11. *Part program modification Lock/Unlock key*: This key is used for preventing an unauthorized user from modifying, editing, or deleting part programs or preventing unintended modification of a part program due to incorrect operation by a user.
12. *Door Interlock key*: In the case that this key is ON, if a door is opened while the spindle is rotating, the emergency stop is invoked.
13. In addition, there is an OT (Over Travel) cancel button that temporarily cancels safety mode when an axis moves beyond its set limit, a power switch, and a reset button that initializes the CNC system.

8.2 Structure of the MMI System

The ultimate design goal for the MMI system is to provide ease of operation and various functions for users. Following this trend, MMI has advanced to become PC-based MMI that is operated by an individual processor and allows various functions and advanced functions to be invoked from a single panel whereas traditional MMI only allows simple operations.

PC-based MMI allows the usage of a graphical user interface that replaces the earlier simple textual user interface. It also allows a CAM system to be used on the CNC system itself and enables the CNC system to communicate with external equipment. Furthermore, the user can use the various functions normally found on a PC. In recent times, the majority of PC-based MMIs use Windows OS from the Microsoft Corporation as an operating system, which makes third-party development and deployment of MMI applications relatively easy. Accordingly, the MMI system of PC-based systems are developed continuously to meet various user requirements. The details of PC-based systems will be addressed in Chapter 9.

As shown in Fig. 8.3, the structure of the MMI software can be divided into three layers; Application layer, Kernel layer, and OS layer.

The application layer is composed of the applications with which the user interacts. The following MMI functions belong in this layer and each application is made in stand-alone executable file format.

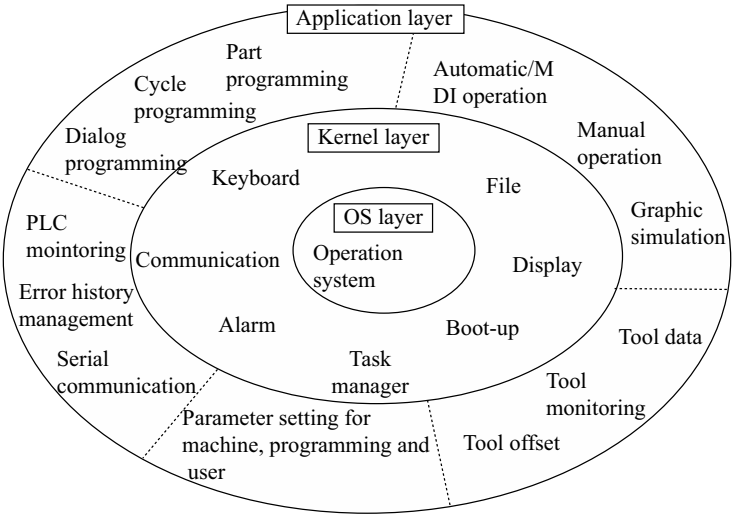


Fig. 8.3 MMI software structure

1. *Machine Manager*: This program monitors the machine status and displays the real-time tool path during machining in Auto mode or MDI operation mode.
2. *Parameter Manager*: The user can edit NC parameters and system parameters using this program.
3. *Program Manager*: This program provides the functions for editing G-code programs and managing part programs such as saving and deleting.
4. *Tool Manager*: This program is used for editing and managing the tool information, such as tool offset, tool life, and tool geometry.
5. *Utility*: Service functions of the CNC system such as alarm history management, PLC monitoring, DNC, and communication with external systems are provided.

The functions provided in the application layer may be added, deleted, or replaced according to the user's needs. Therefore, in order to make this possible, openness should be considered when the kernel layer is designed.

As the kernel layer is the core of the MMI software, it plays the role of linking the applications and the NCK. It sets environment variables during system boot-up, links application modules with key input and alarm/help file, and transfers files and parameters. The binary modules for executing the following functions are placed in the kernel layer. The modules are automatically linked with the applications while the CNC system is running.

1. *System boot-up*: This function initializes the variables of the operating system and system boot manager for setting the language type of MS Windows, machine parameters, etc.

2. *Communications interface*: This carries out communication and data exchange with the NCK and PLC. It manages the services for sending the data required by the user to the MMI for display.
3. *File management*: This provides the services for managing folders and files, such as copying, saving, deleting, and changing part programs and PLC programs.
4. *Alarm*: This displays alarm and error messages from the machine, PLC, and MMC in the alarm window. It manages the history and displays the help information.
5. *Key input*: This transmits the key input from soft keys, keyboard, and dialog boxes to the applications and the CNC system.
6. *Screen Display*: This handles the horizontal or vertical function key window that is shared by all applications and connects the function keys with particular applications. In addition, it provides the interface for handling MMI soft keys.
7. *Task manager*: This executes the programs registered in the application layer and provides the function for calling and switching them. It registers the applications as a program list in a text file format and executes the applications sequentially when the task manager begins. When the task manager is terminated, it terminates the applications in reverse order. The basic functions can be summarized as follows.

   - Registering/terminating applications
   - Defining the execution sequence for applications and initializing them while booting up.
   - Switching applications while they are executing.
   - Monitoring system resources.

An MMI system based on PC hardware typically uses a PC operating system as OS. MS Windows or Linux have both been used (recently, Windows embedded XP and Windows CE have become widely used) However, these operating systems cannot provide the real-time capabilities required by a CNC system. Generally, an MMI system requires a non-real-time environment, whereas an NCK system needs a real-time environment. Therefore, when the overall architecture of the CNC system is designed, techniques to overcome the non-real-time capabilities of the PC operating system must be considered. One simple solution is to use two operating systems, using a PC operating system (non-real-time OS) and a hard real time OS for the MMI and NCK systems, respectively. In this case, it is very important to regard the execution of the MMI system as one specific task in the NCK system.

In the MMI, various applications are executed based on the kernel and the user interface for editing a part program, which is one of the key applications in MMI. In general, the machine tool operator spends a lot of time learning how to generate a part program. So, from the MMI designer's point of view, the MMI should be designed for the MMI to be able to provide the most efficient method for generating a part program. In the following sections, the advantages and disadvantages of various programming methods will be discussed. The design of an efficient programming system will also be addressed.

8.3 CNC Programming

In order to machine the part in a drawing by using CNC machine tools, it is necessary to generate a series of instructions for activating those CNC machine tools. This task is called CNC programming.

8.3.1 The Sequence of Part Programming

Roughly, CNC programming is composed of the generation of a process plan from a part drawing and the generation of the part program. The detailed processes are as follows.

1. To analyze the part drawing.
2. To decide on the removal volume and to select the machine.
3. To decide on the jig and chuck.
4. To decide on the setups, machining sequences, cut start points, cut depths for roughing and finishing allowance.
5. To select tools and tool holders and to decide on the tool position.
6. To decide on the technology data such as spindle speed, feedrate, and coolant on/off.
7. To generate the part program (including post-processing).
8. To verify the part program.
9. To machine.

The tasks from stage 1 to stage 6 are included in the preparation stage where the part drawing is analyzed and the machining strategy is decided for creating a part program. These tasks are called “process planning”. Process planning is done by a programmer or a machine operator. Extensive knowledge about the machine tools, CNC equipment, tools, and cutting theory is required to generate fine process planning. However, in practice it is very difficult to find experts for these. Therefore, many studies on CAPP (Computer Aided Process Planning) for automatically executing process planning have been carried out.

After process planning, a part program (stage 7) for controlling CNC machine tools is generated. The generation of this part program can be done by the manual programming method or the automatic programming method. In the manual programming method, a programmer directly edits the part program in CNC-readable EIA/ISO code. In the Automatic programming method, a programmer edits the program in terms of graphical symbols or a high-level language via a computer. The CNC system then converts this program into machine-readable instructions and executes those instructions.

The automatic programming method can be classified into two types in terms of the editing method; the first is the language-type programming method where a high-level language is used for programming. The second type is the conversational

programming method where a programmer creates the program as he/she converses with the CNC system using graphical symbols. The various programming methods are depicted in Fig. 8.4. The key characteristics of each programming method will be described in detail in the following sections.

After completing the part program, the part program is verified by using simulation (stage 8). Through the simulation, errors can be detected and corrected. Also, if necessary, test cutting is carried out before real machining begins.

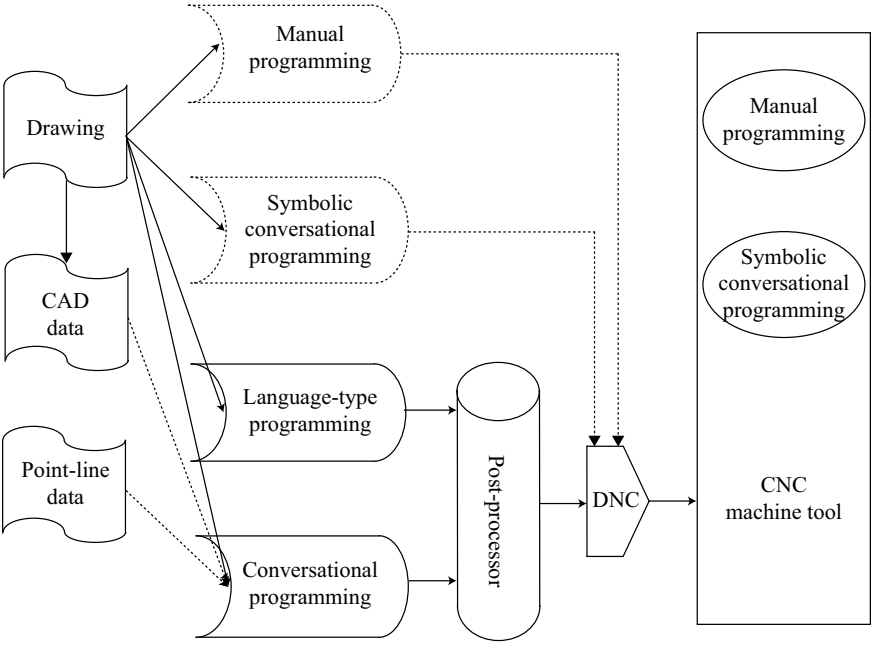


Fig. 8.4 Programming methods

8.3.2 Manual Part Programming

CNC equipment provides various instructions for the preparation functions, feed functions, spindle functions, tool functions, auxiliary functions, and other functions to meet the EIA/ISO standards. Direct editing of the program with the instructions (codes) provided by the CNC equipment is called manual programming. The part program generated by manual programming method can be executed not only within CNC equipment but also outside the CNC equipment.

Due to the differences in terms of function and design concept between CNC makers, each CNC system has a slightly different programming instruction set compared

with other CNC systems, although the EIA/ISO standard for programming instructions exists. This makes it difficult for a programmer to use a variety of CNC systems. Also, for the manual programming method, the efficiency and productivity of the part program depends on the programmer's ability. Therefore, knowledge about process planning, machining theory, G-code, and complex computations for toolpath generation are necessary for a good programmer and a long training time and much effort are also required. Further, because of the lack of compatibility between programming instructions (G-code), a programmer has to learn new programming instructions if the CNC system is changed. In addition, it is almost impossible to create a part program for machining 2.5D or 3D shape using the manual programming method. However, in the case of simple machining and repeated machining tasks, the manual programming method makes quick programming possible. It also makes it possible to generate a part program quickly by modifying an existing program and using macro programming. Moreover, depending on the programmer's ability, it is possible to generate a part program for unusual and specific shapes.

The automatic programming method, where a computer is used, was developed to overcome the above-mentioned problems with the manual programming method. The automatic programming method makes it easy to machine parts with complicated or 3D shapes. It also makes it possible to generate the large part programs in a short time. In addition, with computer simulation, it makes it possible to detect and modify machining errors before actual machining begins.

8.3.3 Automatic Part Programming

The automatic programming method can be classified into the language-type programming method and the conversational programming method. In the language-type programming method, the machining sequence, part shape, and tools are defined in a language that can be understood by humans. The human-understandable language is then converted into a series of CNC-understandable instructions. In the conversational programming method, the programmer inputs the data for the part shape interactively using a GUI (Graphical User Interface), selects machining sequences, and inputs the technology data for the machining operation. Finally, the CNC system generates the part program based on the programmer's input. Typically conversational programming can be carried out by an external CAM system and a symbolic conversational system that is located either inside the CNC system or in the external computer. In this book, the implementation of symbolic conversational programming systems embedded in the CNC will be addressed in detail.

8.3.3.1 Language-type Programming

Language-type programming is the method in which a programmer edits a part program using language-type instructions that the user can easily understand. As the

manual programming method is similar to assembly language programming, so the language-type programming is similar to programming in BASIC or FORTRAN. For language-type programming, APT, EXAPT, FAPT, KAPT, and COMPACT II have been widely used.

- *APT (Automatically Programmed Tool)*
  APT, which was developed in the USA in the 1960s, is the most famous system for the language-type programming tool and has the greatest number of functions. APT allows representation of various geometries, such as line, circle, ellipse, sphere, cylinder, cone, tabulated cylinder, and general two-dimensional surfaces. By using APT, it is possible to generate programs for 3-axis, 4-axis, and 5-axis machining, including rotation control for spindles and machining tables.
  Figure 8.5 shows the structure of a part program in APT. The part program consists basically of four parts; 1) the shape definition part where the shape for the machined part is specified, 2) the motion definition part where the tool paths are specified, 3) the post processor part where cutting conditions and the characteristics of the CNC system are specified, and 4) the Auxiliary part where auxiliary data such as tool size, workpiece number, and so on is specified.
- *EXAPT*
  EXAPT was developed in Germany. There are three kinds of EXAPT; EXAPT I for position control and linear machining, EXAPT II for turning, and EXAPT III for milling such as two-dimensional contour machining and one-Dimensional linear machining. EXAPT is very similar to APT but without workshop technology. EXAPT decides automatically how many tools are needed by considering the material of the workpiece, required surface roughness, and the shape of the hole specified by the programmer. It calculates automatically the spindle speed and feedrate. In EXAPT II, with user specification of the shape of the blank material and machined part, all machining operations including the machining allowances are generated automatically. On the other hand, it is necessary to register the prespecified data because appropriate spindle speed, feedrate, and cutting depth can be varied according to the machine and tools. Because EXAPT generates automatically not only the tool path but also machining operations and cutting conditions, it is easier to use than APT. However, the kinds of machineable part shape that can be handled are more limited than with APT.
- *FAPT*
  FAPT was developed by FANUC and is similar to APT. FAPT can be used in carry-on exclusive programming equipment. By using particular programming software such as FAPT Turn, FAPT Mill, and FAPT DIE-II, part programs for turning, milling, and die and mold machining can be generated easily. The FAPT Turn/Mill system has the following characteristics.
  FAPT turn is a software library for turning. For part programming, the coordinate system of the rotation axis of the workpiece is defined as the Z-axis and the radius direction of the workpiece is defined as the X-axis. It is possible to program based on both diameter and radius values of the X-axis. FAPT turn provides 1) roughing, 2) finishing, 3) grooving, and 4) threading as metal-removal operations. The

```
CLPRNT
LI82    =LINE/6.25,-1.0,2.0,0.25,-1.0,2.0
LI83    =LINE/0.25,-1.0,2.0,2.0,3.5,2.0
LI84    =LINE/2.0,3.5,2.0,6.7525,1.1319,2.0
CI58    =CIRCLE/6.2507,0.125,2.0,1.125
LI85    =LINE/6.2507,-1.0,2.0,6.25,-1.0,2.0

      CUTTER/0.25,0.05,0.075,0.05,0.0,0.0,4.0
      COOLNT/ON
      SPINDL/1200
      FEDRAT/1.0
      OUTTOL/0.005
      TLAXIS/0.0,0.0,1.0
          FROM/0.0,0.0,5.0
          RAPID
          GOTO/-0.1228,-1.255,3.0
      THICK/0.0,0.13
           DNTCUT
           GOTO/-0.1228,-1.255,1.0
           GO/ON,LI82,TO,(PLANE/0.0,0.0,1.0,1.0),TO,LI83
            CUT
            INDIRV/0.3624454,0.932005,0.0
            TLLFT, GOFWD/LI83, PAST, LI84
            GORGT/LI84, TANTO,CI58
            GOFWD/CI58,TANTO,LI85
       THICK/0.0,0.13,0.0
            GOFWD/LI85,ON,(LINE/(POINT/6.25,-1.255,1.0),PERPTO,(LINE/$
             6.2507,-1.255,1.0,6.25,-1.255,1.0))
       THICK/0.0,0.13
            GOFWD/LI82,PAST,LI83
            GORGT/LI83,PAST,LI84
            GORGT/LI84,TANTO,CI58
            GOFWD/CI58,TANTO,LI85
       THICK/0.0,0.13,0.0
            GOFWD/LI85,ON,(LINE/(POINT/6.25,-1.255,1.0),PERPTO,(LINE/$
             6.2507,-1.255,1.0,6.25,-1.255,1.0))
       THICK/0.0,0.13
            GOFWD/LI82,PAST,LI83
            TLON,GORGT/(LINE/-0.1228,-1.255,1.0,2.0,1.0,1.0),ON,(LINE/$
              POINT/2.0,1.0,1.0),PERPTO,(LINE/-0.1228,-1.255,1.0,2.0,1.0,1.0))
       FINI
```

Fig. 8.5 APT program structure

tool nose compensation such as leaving finish allowance based on the machining tolerance and tool radius is possible. In addition, the tool path can be displayed graphically.
FAPT Mill is the automatic programming system for generating a part program for milling. It supports drilling, 2.5D machining of shapes made from lines and arcs, 3D machining of shapes made from spheres, cylinders, cones, and slanted planes. Free-form curves made using discrete points and pattern drilling, which is a repetition along a pattern element such as a line, arc, or grid, are possible. During simulation, the tool path can be displayed on the *XY* plane, *YZ* plane, *ZX* plane, or on an arbitrary plane projected from an arbitrary direction. In FAPT Mill:

1. it is possible to define a variety of geometries based on point, line, arc, slant plane, cylinder, cone, and sphere.
2. it is not necessary to define extra geometries for generating desired shapes.
3. it is possible to specify tool movement with a descriptive geometry name.
4. Tool radius compensation (left/right) and subroutine calls are possible.
5. variables and a variety of mathematical functions, such as the four arithmetical operations and trigonometric functions, can be used.

Apart from these, other programming languages, such as COMPACT-II, have been developed. However, the basic concept of these is similar to that of APT.

8.3.3.2 Conversational Programming

In order to carry out manual programming or language-type programming, a programmer must know the program instructions, and this makes the generation of part programs difficult. To overcome this problem, creation of part programs without knowledge of detailed program instructions needs to be possible. Due to this requirement, conversational automatic programming systems were developed that enable a programmer to generate tool paths by selecting machining features and operations as well as inputting data and following the system's instructions. In general, the conversational programming system category includes systems executed outside the CNC system in order to generate part programs for two-dimensional contours and three-dimensional free-form surfaces, so-called CAM (Computer-Aided Manufacturing). There are various examples of this type of system, such as CATIA, MasterCAM, EdgeCAM, so on.

As the above-mentioned conversational programming system is an offline system, a part program is generated on an external computer rather than on the CNC system. Because of this, the part program has to be transferred to the CNC system via a DNC system. Therefore, the creator of the part program and the operator of the part program can be different and so, in practice, it can be difficult to apply data optimization to the part program. In addition, in the case of simple machining, the usage of a CAM system reduces productivity. Accordingly, with the improvement in CPU and graphic performance, the symbolic conversational programming method,

which enables programmers (including novices) to generate part programs quickly and accurately on the shop floor in order to overcome the disadvantages of CAM systems, has been widely used.

In general, the symbolic conversational programming method is called WOP (Workshop Oriented Programming) or SOP (Shopfloor Oriented Programming). As shown in Table 8.1, this has different characteristics compared with other programming methods. It has been widely used on the shop floor and has a good effect on productivity. In this text book, the design and development of Shopfloor Oriented Programming systems embedded in CNC systems and used on the shopfloor will be addressed in detail.

Table 8.1 Comparison between programming methods

	Advantage	Disadvantage
EIA/ ISO	Easy to apply to simple operations such as tapping, drilling Basic function of CNC equipment.	Full knowledge of G-code required. Knowledge of geometry/ mathematics needed for calculating toolpaths.
CAM	Possible to specify complicated shape. Possible to generate programs for various machines with one package.	Very expensive and requires expert. Impossible to feed back programs optimized on shopfloor.
Symbolic	Experienced person can use easily. Easy to create part program. Possible to feed back program optimized on shopfloor.	Program can be used only on a particular machine. In order to apply program to different machine, re-programming required. Programming for complicated parts is restricted.

The shopfloor programming system in CNC can be widely used for generating a part program on a variety of machine tools. In particular, when this programming system is applied to machines that produce parts with simple 2D, 2.5D, and primitive 3D shapes, it is possible to improve productivity and flexibility.

Considering that an operator edits the part program at the front of a machine, offline CAM systems are more appropriate than shopfloor programming system in the case of the milling, for which it takes a long time to specify the part shape. However, shopfloor programming systems can be applied to wire-EDM or turning where the part shape is simple. In particular, the usefulness of the shopfloor programming system can be maximized when it is applied to turning machines with milling functions. Figure 8.6a shows how a turning machine with milling function can machine a milling feature on the end of cylinder. Figure 8.6b shows how a turning machine can generate a groove on the surface of a cylinder. To carry out the machining shown in Fig. 8.6 it is necessary to make a part program whereby the rotation of the spindle and the movement of the turret or tool post are controlled simultaneously. In practice,

even experts have difficulty in creating part programs for turn-mill machining manually. However, if a programmer uses the shop floor programming system, he/she can generate a part program by merely entering the feature geometry data and cutting depth for the machining shown in Fig. 8.6a and by merely entering the data of the groove shape and cutting depth for the machining shown in Fig. 8.6b. Thereby, the productivity of novice operators can be drastically increased by using shopfloor programming systems.

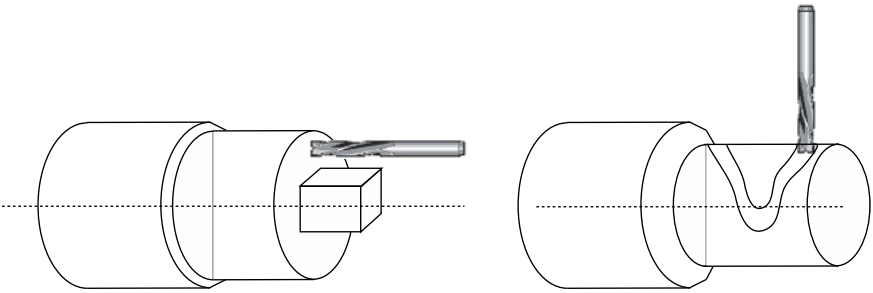


Fig. 8.6 Turning with milling

8.3.3.3 CAM Systems and Shopfloor Programming

Recently, with the use of PCs as MMI hardware, attempts have been made to embed PC-based CAM systems in the MMI and to replace shopfloor programming systems with online CAM systems. Because the ultimate goal is to edit the part program easily, they play similar roles. Each system consists of a graphical user interface, initialization module, contour module for specifying part profiles, machining cycle module for specifying machining operations and generating toolpaths, tool module for managing tools, simulation module for verifying the toolpath, and utility module for managing the part programs, as shown in Fig. 8.7.

The CAM system and Shopfloor programming system have slight differences in terms of function. The target machine of a shopfloor programming system is restricted to one machine or to machines of a similar type, but CAM systems can be applied to a variety of machines by providing a post-processing function. Therefore, in the case of a CAM system, a variety of machining conditions have to be considered. However, because only machine-specific functions are considered in the case of the shopfloor programming system, the function and architecture of the shopfloor programming system can be simpler than those of the CAM system.

However, there are too many problems caused by the difference between the design concepts to use CAM systems designed for offline usage on a CNC system. For example, a pointing device such as a mouse can be used for specifying the part profile and inputting the data to the CAM system. However, in the shopfloor pro-

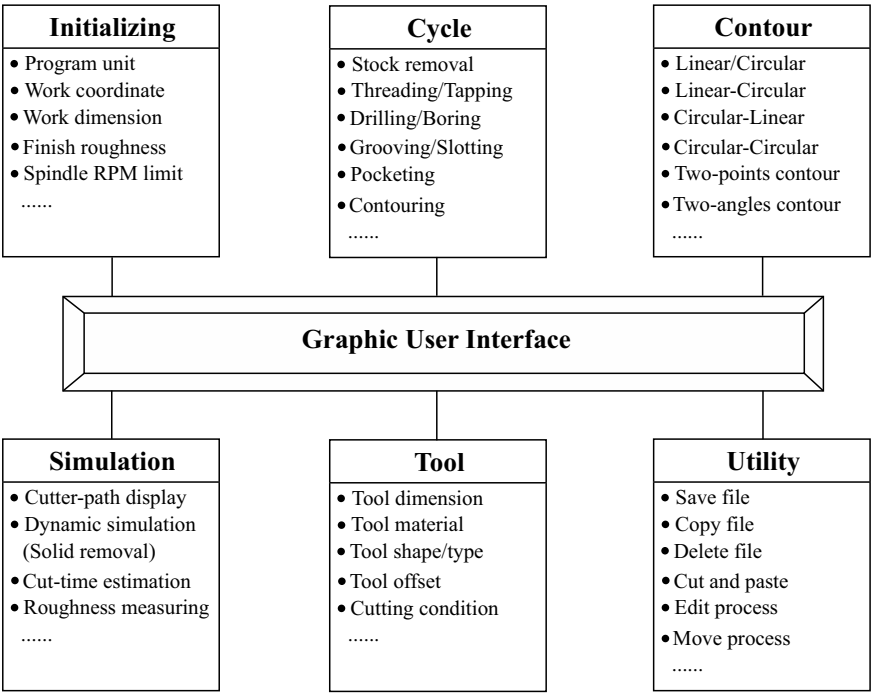


Fig. 8.7 CAM system structure

gramming system, a pointing device cannot really be used and only limited buttons are available. Also, because it is impossible for a user to edit a program at the front panel of a machine for a long time, quick and easy programming methods to specify part design and machining operations and enter key inputs are needed. In addition, it is necessary that the data modified is, after simulation, directly incorporated into the part program and saved.

Further, it is necessary to generate a machining cycle reflecting the parameters specified in the CNC system and it is also necessary to prevent programming that is outside the machine's performance. Moreover, a way of helping a novice operator to decide input data (operation sequences, removal volumes (features), tools, and cutting conditions) or recommending input data values, is required.

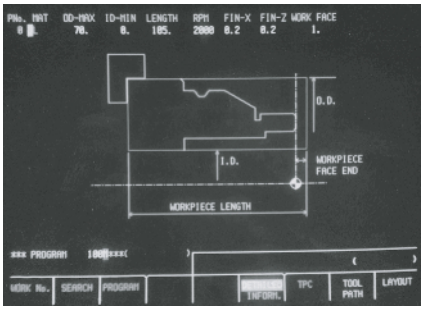
Therefore, the shopfloor programming system must provide a variety of methods to increase the program's productivity by supporting the graphic user interface. The basic modules of the shopfloor programming system have the following properties and examples of modules are shown in Fig. 8.8.

1. Initialization module: In this module, the global variables, coordinate system (absolute/incremental), programming unit (inch/metric or diameter/radius), spindle data, feed unit (mm/rev or m/min), tool retract position, tool-retraction method, workpiece material, and machine data are specified, (see Fig. 8.8a).

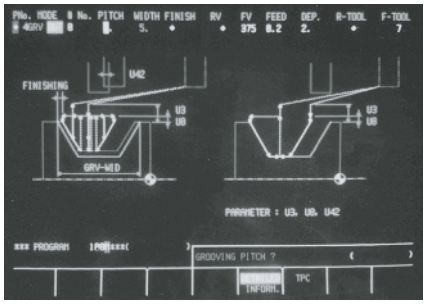
2. Machining Cycle Module: The specific machining scheme for milling, turning, and drilling is defined as one block. This block makes programming simple and efficient. The block is called a module. The machining operations such as roughing, finishing, drilling, slotting, and pocketing provide a variety of machining strategies. It is important to minimize the data that a programmer should input and select via the GUI during programming. This module is the core module of the conversational programming system (see Fig. 8.8b).
3. Module for defining the part profile: This module is used for defining the part shape. For this module, a different GUI is provided compared to that of a CAD system. This module provides the conversational contour programming GUI that consists of various graphic menus including line and arc geometries. In particular, in the case of finishing, individual surface finishes can be specified for each profile and feedrate can be computed automatically for each profile based on the surface finish. Of course, in the case of threading, slotting, and drilling, except for contour machining (profile machining), feature definition is carried out together with specification of machining cycles. The important thing for contour programming is that the dimensional data can be input easily without additional calculations during specification of the part profile. In addition, chamfer and round should be easily specified (see Fig. 8.8c).
4. Tool module: The tool module actually consists of two modules; the first is used for attaching the tool to the turret or tool magazine and the second is used for selecting the tool from the turret or tool magazine. One provides the GUI for specifying tool position, tool type, and tool geometry and the other provides the GUI for selecting the appropriate tool from the turret or tool magazine. Cutting conditions and spindle speed are automatically recommended by the system based on the tool, workpiece material and tool geometry. When the tool has been selected, a variety of data required for machining are automatically set using predefined values. If modification is needed, the programmer can modify these individually (see Fig. 8.8d).
5. Toolpath verification module: This module provides the functions for graphically simulating the toolpath of the program that was generated based on the programmer's input. By using this module, a programmer can verify the process from blank material to the final shape. Moreover, because this module displays the machining time (cutting time and non-cutting time) it can be used for optimization of the toolpath, (see Fig. 8.8e).
6. Utility module: this module provides the functions for copying, deleting, saving, and moving part programs, tool data files, and tool path files. It provides a text editor for modifying the file and moving, deleting, and editing operations for the generated programs, (see Fig. 8.8f)

The above-mentioned system can be summarized as a system that enables an operator to execute sequentially the steps of setting the program environment, setting the tool, selecting the machining cycle, and verifying the toolpath. The system provides a variety of graphical user interfaces for easily specifying the machining cycle

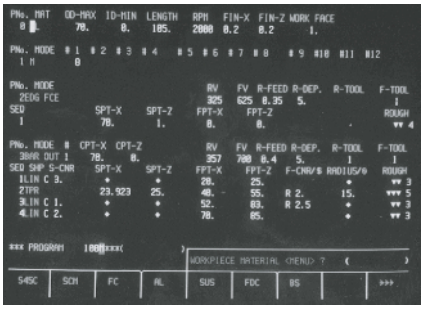
(a)



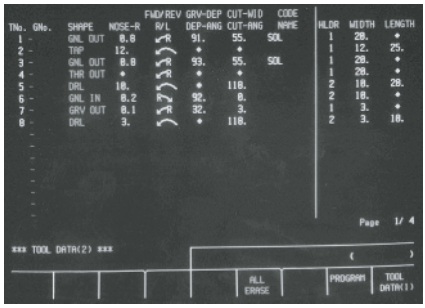
(b)



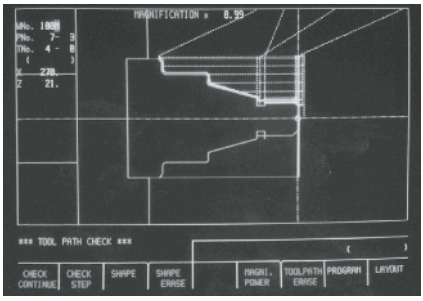
(c)



(d)



(e)



(f)

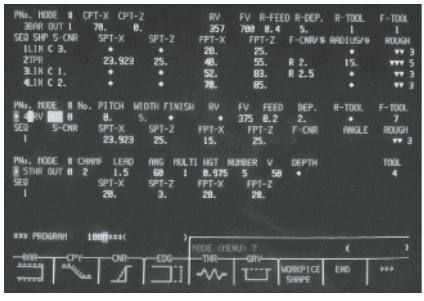


Fig. 8.8 CAM function displays (Courtesy of Mazatrol)

and machining feature, and generating a part program by interaction with the system without needing to memorize the programming method.

In order to help an operator generate, verify, and modify a part program quickly, CNC makers have developed and provided various shopfloor programming systems that can be operated only on their own CNC systems. For example, Siemens provided the Blue print programming system, the Support cycle programming system, and the WOP system. FANUC, Mazak, and Yasnac have provided the EZ-guide, the Mazatrol conversational programming system, and the Compact programming system, respectively.

These support various programming levels from low-level programming to high-level programming including complicated part machining. In the following section, using the Mazatrol system as an example, the characteristics of a shopfloor programming system will be addressed.

8.4 Mazatrol Conversational System

The Mazatrol Conversational Programming System is designed to enable a programmer to generate a part program quickly and verify it without needing either a manual or an assistant. It has been widely used in industry and provides various machining cycles that include the machinist's know-how. In addition, it provides a graphic interface (Fig. 8.8b) to enable programming without detailed programming knowledge.

8.4.1 Turning Conversational System

The machining cycles in terms of the machining mode and cutting mode, the key characteristic of the Mazatrol Turning Conversational Programming System, are summarized as follows.

1. *Feature Mode*: This denotes the machining cycles that are provided in conversational programming system. In this system, twelve machining cycles are provided as machining cycles, as shown in Fig. 8.9. As can be seen from the figure, the twelve cycles are as follows:

   - BAR: This denotes the operation for machining a cylindrical part by turning. This cycle is used for rough machining of an arbitrary part.
   - CPY: This is used for finish machining of a specified part with finishing allowance.
   - CNR: After finish and rough machining, an undercut area can be left due to the tool's shape. This cycle is used for machining the undercut area.
   - EDG: This cycle is used for machining the end face of the cylindrical part.
   - THR: This cycle is used for threading.
   - GRV: This cycle is used for machining a groove with arbitrary shape.

- MTR: This cycle is used for cutting in the part.
- DRL: This cycle is used for drilling a hole.
- TAP: This cycle is used for tapping.
- MNP: This is used for generating a part program in manual mode in order to machine special features that are not included specifically in this list.
- MES: This is used for measuring the machined part on the machine after machining has been completed.
- M: This is used for setting M-codes for controlling the machine behavior other than the servo motors.

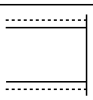
1. BAR	2. CPY	3. CNR	4. EDG	5. THR	6. CRV	7. MTR	8. DRV
							
9. TAP	10. MNP	11. MES	12. M				
	(Manual)	(Measurement)	(Auxiliary)				

Fig. 8.9 Machining cycles

2. *Cutting Feature*: After selecting the feature mode, the cutting method should be decided. For example, the rough machining mode feature (*i.e.* BAR) should be followed by inner contouring, outer contouring, facing, and back facing depending on the machined region. Therefore, the Cutting Feature is restricted by the type of Feature Mode. The relationship between Feature Mode and Cutting Feature is shown in Fig. 8.10. When BAR, CPY, CNR, EDG, THR, or GRV are selected, eight kinds of Mode Feature can be selected. In the case of MTR, only OUT (outer contouring) and IN (inner contouring) can be selected. In addition, because DRL and TAP can be applied in the face, selection of Mode Feature is not needed.
3. *Machining Strategy*: In order to execute the operation selected from Mode Feature, it is necessary to decide on the machining strategy. The machining strategies that can be applied according to the Feature Mode are shown in Fig. 8.11. For BAR and CNR, the tool retraction method has to be selected. When THR is selected, six kinds of machining strategy can be selected. In the case of GRV, various groove shapes can be selected. Since the conversational programming system guides the choice of appropriate strategies depending on the feature and operation, even non-expert programmers can select the appropriate machining strategy.

4. *Tool and cutting condition*: After Feature Mode, Cutting Feature, and machining strategy have been specified, it is necessary to select the appropriate tool and decide on the cutting conditions. The tool is selected from the tool database that

[1]. Mode Feature: BAR, CPY, CNR, EDG, THR, GRV

1. OUT	2. [OUT]	3. IN	4. [IN]	5. FCE	6. [FCE]	7. BAK	8. [BAK]
							

[3]. Mode Featuring: DRL, TAP-Set as “FCE”

[2]. Mode Feature: MTR

1. OUT	2. IN
	

Fig. 8.10 Relationship between Feature Mode and Cutting Feature

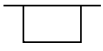
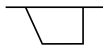
[1] Mode: BAR, CNR

1. #0	2. #1	3. #2
		

[2] Mode: THR

1. #0	2. #1	3. #2	4. [#0]	5. [#1]	6. [#2]
Standard	Constant depth	Constant width	Standard	Constant length	Constant area

[3] Mode: GVR

1. #0	2. #1	3. #2	4. #3	5. #4	6. #5
					

[4] Mode: DRL

1. #0	2. #1	3. #2	4. #3	5. #0	2. #1	3. #2
Stationary hole drilling	Deep hole drilling	High-speed drilling	Stationary hole reaming	Through hole drilling	Deep hole drilling	High-speed drilling

[5] Mode: TAP

1. #0	2. #1	3. #2	4. #3	5. #4	2. #5
M	UM	PT	PF	PS	Special

Fig. 8.11 Machining strategies to be applied according to Feature Mode

is pre-specified based on the tools loaded onto the machine. The cutting conditions can be recommended automatically by the system according to the tool and workpiece material or can be input directly by the programmer.

5. *Machining Geometry*: The last step to input the feature data is to specify the machining geometry. The geometry of the feature can be made up from lines, slanted lines, convex arcs, concave arcs, and circle centers, see Fig. 8.12. The programmer selects the geometric elements that compose the feature and inputs their positions to define them fully. For each segment, surface roughness can be specified. If the programmer does not specify this, a default value, defined by a global variable, is set.

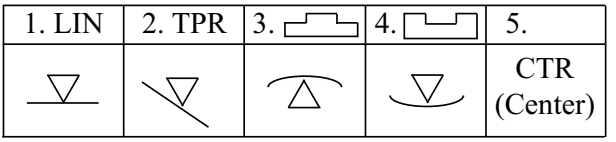


Fig. 8.12 Feature geometric elements

8.4.2 Programming Procedure

The programming procedure in the Mazatrol system is as shown in Fig. 8.13. The procedure is composed of three parts; the first is the header part where the part program number is specified and global data in the initialization module are defined. The second is the body part where a variety of information for machining, such as feature data, machining operation data, and cutting condition data, are defined. The last is the end part where the data for the task to be carried out for completing the program are specified.

In the header part, the material, diameter, and length of the workpiece, maximum spindle speed, finish allowance, and surface finish are specified as global data.

In the body part, the data for defining machining features are specified. First, the machining mode (*e.g.*, BAR, CPY, DRL, and TAP), called "Mode Feature" in the Mazatrol system, is selected and the machining part (in Mazatrol called "cutting feature") relevant to the selected Mode Feature (*e.g.*, internal, external, and face) is selected. After that, the machining strategic data are specified and the tool and cutting conditions are selected. The cutting conditions can be selected automatically from a pre-specified database or selected manually by the operator. Finally, if it is necessary to specify the part profile depending on the selected Mode Feature (for example, bar machining, copy machining, and grooving machining), the shapes of the blank material and finished part are specified by inputting segment features.

The end part can be used optionally. In this part, the tasks that must be executed before completing the part program are specified. For example, it is possible to spec-

ify whether the number of a finished part is counted or the M-code for activating automation equipment is called. In addition, it can be optionally specified how often the part program is repeated or what program will be invoked for subsequent execution.

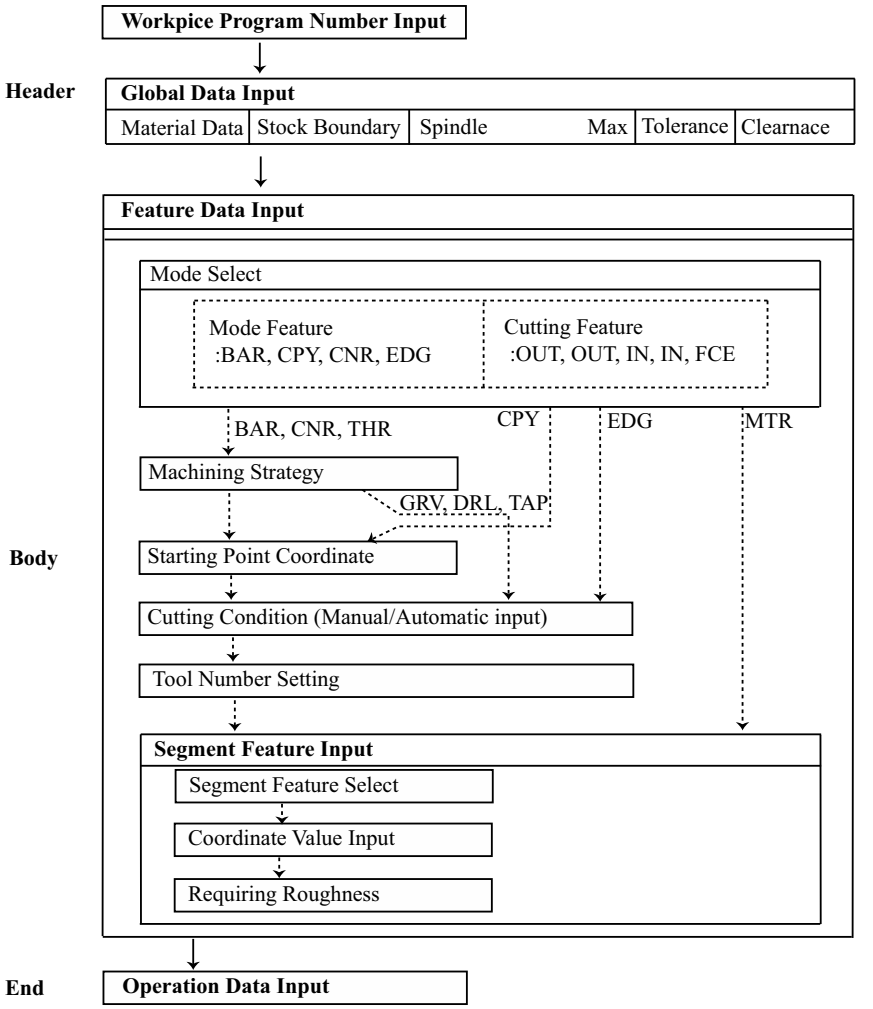


Fig. 8.13 Mazatrol programming procedure

8.5 Conversational Programming System Design

As the shapes of parts have become more complex and their accuracy has increased, so has part programming become more difficult. At the same time, on the shopfloor, the number of expert programmers has decreased. Because of this, a conversational programming system has become an essential function of an advanced CNC system. This conversational programming system must provide not only basic functions provided by the CNC user interface but also intelligent machining cycles based on a graphical user interface, strategy and cutting condition database based on expert know-how, tool path verification functions, and a variety of utilities.

A conversational programming system is designed as a system that:

1. can be used even by an inexperienced operator,
2. can generate a part program quickly with minimal key input,
3. can verify the generated part program in a short time,
4. can introduce the modified information easily into the generated part program, and
5. can be operated on the CNC system on the shopfloor.

8.5.1 Main Sequence for Design

The procedure for creating a part program in a conversational programming system can be summarized as shown in Fig. 8.14.

1. Start the conversational programming system by selecting the programming key in the MMI.
2. Set the initial data (global data) following the screen indications generated by the Conversational Programming System.
3. Select the particular operation and input the data via the GUI (graphical user interface) relevant to the selected operation. The shape of a part, machining strategy, tool, and cutting condition are given as input data.
4. After specifying all operations, generate the part program in standard G-Code or the manufacturer's own code by selecting the program generation key in the MMI.
5. Check the tool path or the finished part via the simulator.
6. If the verification result is not satisfactory, select the modification key and modify the data of the unsatisfactory operation.

Through the above-mentioned programming procedure, an operator inputs the data in steps 2 and 3 (these are represented by the gray boxes in Fig. 8.14) and the others are executed by the conversational program system. Editing the data in a parameter database is possible during programming and before programming. This database has the default values for the parameters for machining strategy, tool, and cutting conditions. In particular, the important thing is that the tool offset, which is

measured on a machine, is managed by the database that is connected to the conversational programming system. The value in the parameter database is used during selection of the machining feature. Figure 8.14 depicts the main components of the conversational programming system for turning. In the case of a milling system, the major components are the same and only the details about machining features, tool databases, and machining strategies are different.

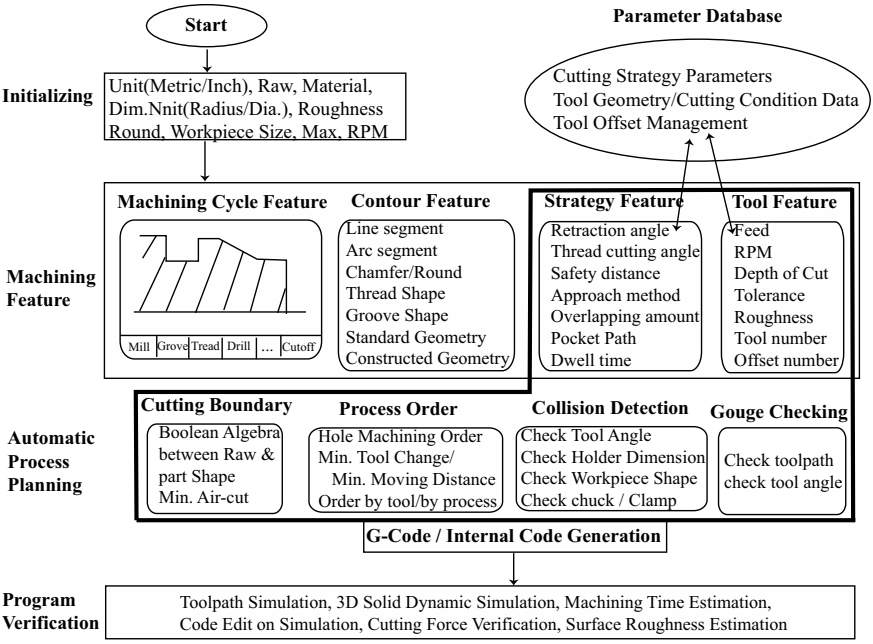


Fig. 8.14 Typical operation steps

Therefore, as mentioned above, the following are key points for designing a conversational programming system with an easy-to-use user interface.

1. For adequate design of the machining cycle feature that meets the machining characteristics of the machine, key functions are needed for minimizing user input by automatic recommendation of machining operations, helping non-experts to edit a program by automatic recommendation of cutting conditions, generation of the toolpath without tool interference, overcut, undercut, and aircut, and determining the operation sequence that minimizes the cutting time and tool change.
2. A unique method for specifying the part shape is needed. In order for an operator to generate a part program quickly at the machine, a simple and easy way of specifying the part shape is needed instead of an offline CAD system.

3. In addition, for realistic simulation, 3D graphics functions are needed. However, because this subject is outside the scope of this book, details of this are omitted.

In this book, the key design factors for a shopfloor conversational programming system will be summarized in terms of a milling system. The implementation of the machining cycle, which generates the toolpath from the machining feature, will also be addressed.

8.5.2 Key Design Factors

The initial setup, machining operation cycle, part profile drawing, tool management, utility, and operation management should be defined as key functions for a conversational programming system.

8.5.2.1 Initial Setup

In the initial setup module, not only global data that is used by the CNC system but also the coordinate system (absolute/incremental), programming units (inch/metric, diameter/radius), spindle data, feed unit (mm/rev, m/min), tool retract point, tool retract strategy, workpiece material and machine specification data are defined in this module. As shown in Fig. 8.15, workpiece geometry, start Z-point, Z safety plane, work coordinates, and clamp design are defined as well. The workpiece material is used together with a tool database for calculating the cutting conditions automatically. The workpiece geometry is used for displaying the initial material on the solid simulator. The working coordinates are used as the reference coordinates during real machining. Clamp design is used by the simulator for detecting tool collisions. Therefore, in order to use the advanced functions of the conversational programming system efficiently, it is essential to carry out an initial setup before specifying the machining operation cycle.

8.5.2.2 Machining Operation Cycle

Initially, how a tool approaches a workpiece, how it is retracted from the workpiece, and how an operation is terminated are specified regardless of the types of machining operation (*e.g.*, milling, turning, and drilling). The region to be machined can be arbitrarily divided and machined.

The machining operation cycle is the code block to command roughing, finishing, drilling, or grooving. It is used for making programming simple and efficient. The machining operation cycle supports various machining strategies and is designed for minimizing user input via interactive user interface. The majority of CAM systems have CAD functions as methods to specify the part shape and features. However,

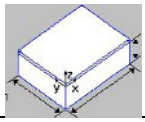
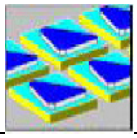
Categories	Contents
File type	Main Program / subprogram
Work material	Workpiece material is classified in terms of the hardness of material.
Work Size	Unit, Length, Width, Height, Face-off 
Initial point - Z	The Z position where tool and workpiece do not collide in the case of rapid traverse (G00).
Multi-mode	Pitch-X Pitch-Y or arbitrary points Xn, Yn  Number : N × M
Work-Zero	Work Zero offset (G54, G55, G56, ...)

Fig. 8.15 Fundamental data for machining

with regard to shopfloor programming systems, CAD functions are inconvenient for an operator. So, for a shopfloor programming system, a different part specification method is used where machining features and operations are specified simultaneously is used.

The machining operations for milling can be classified into three groups, each of which is composed of various machining operation cycles, as shown in Fig. 8.16.

Drilling cycle: As frequently used drilling operation cycles, drilling, boring, fine boring, back boring, tapping, reaming, step boring, and circular milling cycles are supported. In these cycles, it is possible for an operator to input the center of a hole and it is possible to generate automatically the pre-machining/post-machining sequence for a tool that has been selected by the automatic sequencing-tool module. Moreover, since a variety of patterns to specify the center of hole are provided for each cycle, patterned drilling cycles are possible with only one setting.

Profile machining cycle: This cycle is used for machining the specified profile by making a tool move along the specified profile. Linear-profile machining on a plane (top or side), chamfering of the specified profile, and engraving of characters on a plane are provided as profile machining cycles.

Milling cycle: As this cycle is most frequently-used in milling operation, facing, boss machining, pocketing and slotting are supported. As facing cycles, standard shapes

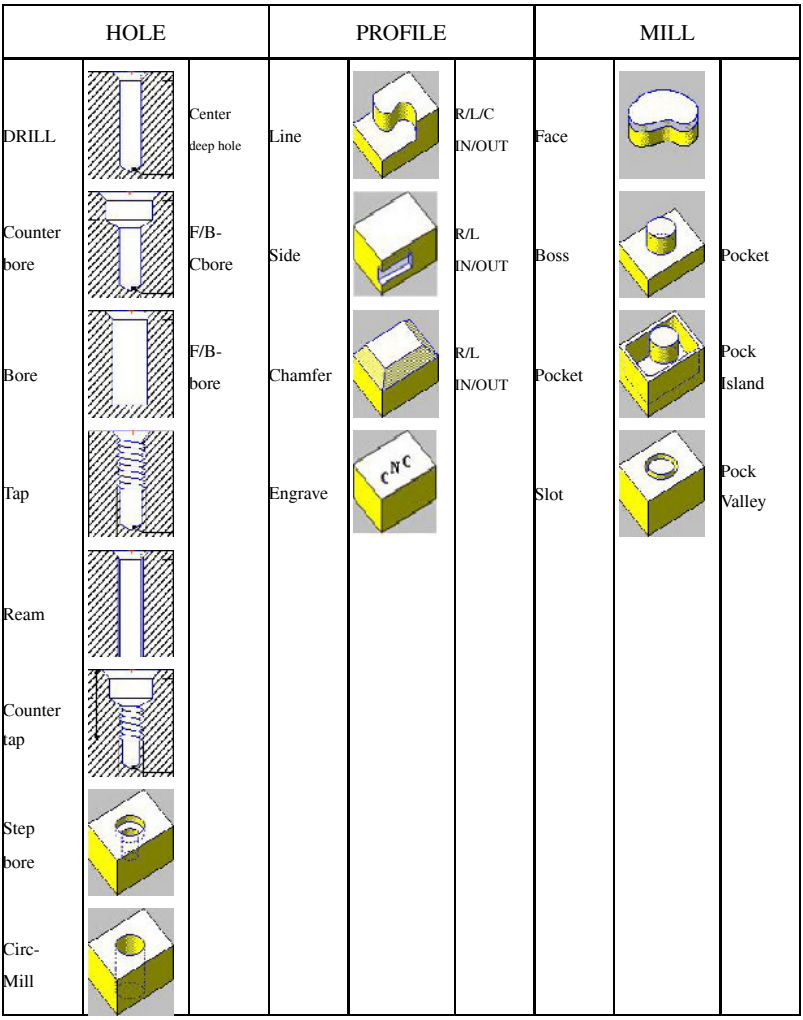


Fig. 8.16 Classification of milling operation cycles

such as rectangular unidirectional, rectangular bidirectional, circular unidirectional, circular bidirectional, circular and rectangular island facing cycles are supported. The pocketing cycle is used for removing the interior of a particular profile. The pocketing cycles for standard shapes such as rectangular, circular, track, and slot profiles are provided. Furthermore, free pocketing cycles for complex profiles composed of linear profiles are also implemented. Contour (unidirectional) type or helical type (bidirectional) can be selected as toolpath generation strategies for optimal toolpaths.

.

8.5.2.3 Input Data List for Machining Operation Cycles

The user input needed to specify the cycles mentioned in the previous section is summarized in Tables 8.2, 8.3 and 8.4.

Table 8.2 Comparison between programming methods

1. Hole machining

Operation	type	Z-height	Diameter 1	Depth 1	Chamfer 1	Diameter (d2)	Depth (d2)	Chamfer 2	Pitch	Cuting Condition
drill	center drill dp.-hl. break	O	hole dia.	hole dep.	hole cham.	x	x	x	x	Ts,Td, Tch: Speed, Feed
count. bore	face back	O	c-bore Dia.	c-bore Dep.	c-bore Cham.	Drill Dia.	Drill Dep.	Bott. Cham.	x	Ts, Td Tch,Tm: S,F
Bore	face back	O	bore Dia.	bore Dep.	bore Cham.	Drill Dia.	Drill Dep.	x	x	Ts, Td, Tch ,Tb: S,F
Tap		O	Hoj.-ky.	Tap Dep.	Cham.	Drill Dia.	Drill Dep.	x	Pit.	Ts, Td, Tch ,Tt: S,F
Ream.		O	ream. Dia.	ream. Dep.	Cham.	Drill Dia.	Drill Dep.	x	x	Ts, Td, Tch ,Tr: S,F
Count. tap		O	Hoj.-ky.	Tap Dep.	Cham.	Cbore Dia.	Cbore Dep.	Bott. Cham.	Pit.	Ts, Td, Tch,Tm, Tt: S,F
Step bore		O	Bore Dia.	Bore Dep.	Cham.	Bore Dia.	Bore Dep.	Bott. Cham.	x	Ts, Td, Tch,Tb: S,F
Cir-Mill		O	Circ. Dia.	Circ. Dep.	Cham.	Drill Dia.	Drill Dep.	Bott. Cham.	x	Ts, Td, Tch ,Tm: S,F

* Ts: spot drill, Td: drill, Tch: chamfer, Tm: endmill, Tb: bore, Tt: thread

8.5.2.4 Machining Geometry Definition

In order to specify the shape and profile of a part, another user interface, different from and easier than that of a conventional CAD system, must be provided. For this, three kinds of method for part shape specification are provided, as shown in Fig. 8.17. The first is a method to design primitive geometric elements, rectangles, polygons,

Table 8.3 Comparison between programming methods

2. PROFILE

Operation	M-type	Start-Z	Cut-D	Cut-R	Chamfer	Fin-D	Fin-R	Cutting Condition
Line	R/L/C IN/OUT	O	O	O	O	O	O	Tr, Tf, Tch: S,F
Chamf.	R/L IN/OUT	O	kans. -D	kans. -R	x	x	x	Tch: S, F
Side Grv	R/L IN/OUT	O	O	O	O	x	O	Tr, Tf: S, F
Engr.	Font	O	Width	Height	x	Col. Span	Row Span	Tf: S, F

Table 8.4 Comparison between programming methods

3. Mill

Operation	M-type	Start-Z	Cut-D	Cut-R	Chamfer	Fin-D	Fin-R	Cutting Condition
FACE		O	O	x	O	O	x	Tr, Tf, Tch: S,F,Dd,Dr
Boss		O	O	O	O	O	O	Tr, Tf, Tch: S,F,Dd,Dr
Pocket	island valley	O	O	x	O	O	O	Tr, Tf, Tch: S,F,Dd,Dr
Slot		O	O	Slot width	O	O	O	Tr, Tf, Tch: S,F,Dd,

and ellipses, by specifying basic parameters. This is called the “standard geometry method”. The second is a method to design the contour profile by adding lines and/or arc profiles sequentially via a graphic menu (called the oriented geometry method). The third method, which is similar to that of a 2D CAD system, is a method to design profiles by cutting, connecting and copying points, lines and arcs (called the constructed geometry method).

In addition to the graphic-based conventional programming system, the software design should allow that connection elements, such as chamfers and rounds, to be specified when the profile (line or arc segment) is specified. To assign individual surface finish to each segment, it has to be possible to assign different feed rates to individual segments while profiles of a part are designed.

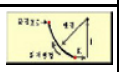
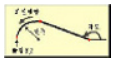
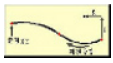
Method	Geometry
Standard Geometry	Rectangle, Circle, Ellipse, Polygon
Oriented Geometry	Line  Arc  Line-Arc  Arc-Line  Arc-Arc  2 Points  2 Angles 
Constructed Geometry	Point (Cartesian/polar, circular, matrix, line) Line (parallel, perpendicular, tangent...) Circle (center + R, 3 line,...)

Fig. 8.17 Part-shape specification methods

8.5.2.5 Tool/Technology Data

The Tool/Technology database recommends the appropriate cutting conditions in terms of the tool and workpiece material and the tool shape. Four kinds of database are supported for the tool used and tool sequence for hole machining.

1. *Tool database*: As shown in Fig. 8.18, this manages the data about tool shape and material. It provides the data for generating toolpaths and deciding cutting conditions.
2. *Cutting condition database*: this manages the data about cutting speed and feed according to tool type, material, and workpiece material.
3. *Tool sequence database*: This manages the machining sequence for efficient hole machining.
4. *Tool offset database*: This manages the data about tool offset.

8.5.2.6 Machining Strategy Data

To generate toolpaths using an operation cycle, it is necessary to decide the machining strategy once the machining features and tools have been decided. In Table 8.5, the data about the machining strategy that should be considered for milling operations are summarized. The usage of these for each operation cycle (hole/profile/milling or 'H', 'P', 'M' in Table 8.5) is checked.

Table 8.5 Machining strategy data

Parameters	Description	H	P	M	Remark
Return type	Tool retraction method (XY/Z, XYZ)	O	O	O	
CLD	Allowance amount when tool approaches	O	O	O	
RTD	Tool retraction distance	O	O	O	
Chamfer all. -R	Chamfer allowance along radial direction	O	O	O	
Chamfer all. -D	Chamfer allowance along depth direction	O	O	O	
Spot Depth	Depth of Spot Drill	O	x	x	
Through Hole	Selection of stop hole or through hole	O	x	x	
Through all.	Backside allowance of through hole	O	x	x	
Dwell	Dwell Time at bottom of hole	O	x	x	
Relief	Return relief amount in case of drilling and boring	O	x	x	Drill/Bore
Feed factor	Retract feed factor	O	x	x	Bore/Reamer
Mill type	Exact arc processing method	O	x	x	C bore/Cir-Mill
Tool end allowance	Tool end allowance in Boring	O	x	x	Bore
Tool end allowance 2	Tool end allowance in Back-Boring	O	x	x	Bore
Finishing allowance	Allowance of bottom in boring	O	x	x	Back C-Bore/ Bore
Stop Hole allowance	Drilling depth allowance for threading, boring or reaming	O	x	x	Bore
Incomplete thread num.	Number of incomplete threads in tapping	O	x	x	
Interference	Checking the interference before machining (the number of blocks ahead that are examined)	x	O	O	
Corner	Machining method at edge. (Round/sharp/square/ trang/fanuc)	x	O	O	
Cycle path	Straight/Zigzag with or without retract	x	x	O	Face
Pock path	para-contour/para-axial	x	x	O	
Cut type	Down/upward machining	x	x	x	
Direction	machining direction: any/ CW/CCW	x	x	x	
Run-in	Approach: no/Arc/ Parallel/Perpendicular	x	O	x	

Table 8.5 (continued)

Run-out	Approach: no/Arc / Parallel/Perpendicular	x	O	x	
Face R feed factor	Feed factor in radius direction.	x	x	O	Face
Face R allowance	Facing allowance in radius direction.	x	x	O	Face
Face allowance	Removal rate in radius direction	x	x	O	Face
Pock R feed factor	Feed factor in radius direction	x	x	O	Pocket
Boss outer Ffac	Feed factor for machining outside of Boss	x	x	O	Boss
Axial D-Ffac	Feed factor for machining in axial direction	x	x	O	Boss, Pocket
Pock R-Fac	Feed factor for full slot cutting in the case of pocketing.	x	x	O	Pocket
Overlap amount for closed shape	Overlap amount when approaching and retracting in closed shape	x	O	O	

8.5.2.7 Graphic Simulation for Verification

Graphic simulation is carried out by a path simulator for verifying tool paths and a solid simulator for verifying the machined shape. A path simulator displays toolpaths as a sequence of lines or arcs and is used for visual verification of the toolpath of a part program (see Fig. 8.19a). It provides the functions for checking for collisions between tools and clamps and editing a part program for correcting incorrect tool paths.

A solid simulator shows the change of part shape of a 3D solid model during machining. Also by using a solid simulator, it is possible to verify tool paths and analyze realistically the machined part (see Fig. 8.19b).

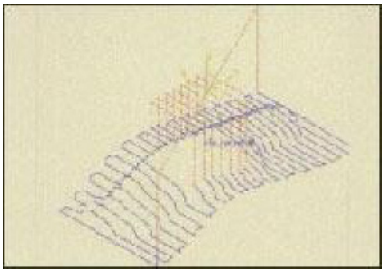
During the programming sequence, the screen displays complete operations. Therefore, if a verification result is different from the operator's expectations, it is possible to modify the operation and quickly correct the program during simulation.

The part shape is also displayed on the screen and regions that cannot be cut due to the tool geometry are checked and displayed. In particular, blank material and removal volumes are displayed simultaneously on the screen and whenever a particular operation is specified, the volume remaining after completion of the specified operation is displayed.

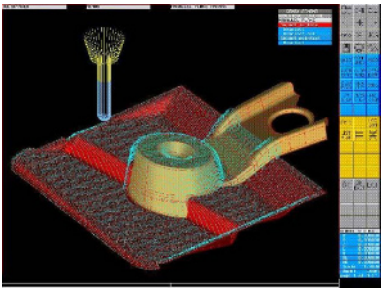
Tool interference due to the diameter of the specified tool is checked automatically. Because machining time (including cutting time and non-cutting time) is always displayed during simulation, the simulation function can be used as the tool for optimizing toolpaths.

Tool	Feature	Tool data	Operation to which tool is applied
Center		Material, diameter, length, point, angle	Hole
Chamfer		Material, diameter, length, point, angle	Profile-chamfer
Drill		Material, diameter, length, point, angle	Hole
Bore		Material, diameter, length	Hole
Tap		Material, diameter, length	Hole
Reamer		Material, diameter, length	Hole
Face mill		Material, diameter, length, cutting teeth number	Mill-face
End mill		Material, diameter, len. flute num., tool type (flat, ball), ball radius	Hole, Profile, Mill
Side mill		Material, diameter, len. cutting teeth num., cutter length	Profile-side

Fig. 8.18 Milling tool database



(a)



(b)

Fig. 8.19 Graphical simulation

8.5.2.8 Operation Sequence Control

This module shows the specified operation cycles and enables an operator to modify and delete them while editing the operation cycle. It also enables addition of new operation cycles and operation sequence changes. It enables operators who are unfamiliar with process planning to generate consistent and efficient programs. Moreover, it is possible to store the generated program on memory or disk and use it whenever it is needed.

8.6 Development of the Machining Cycle

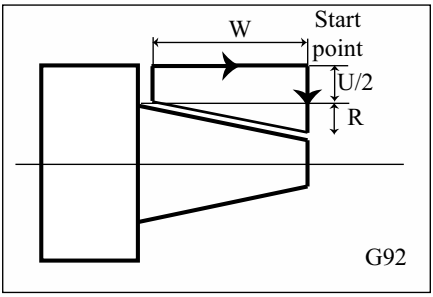
In this section, the implementation of turning manual G-code cycles and various machining cycles for conversational programming system will be described.

8.6.1 Turning Fixed Cycle

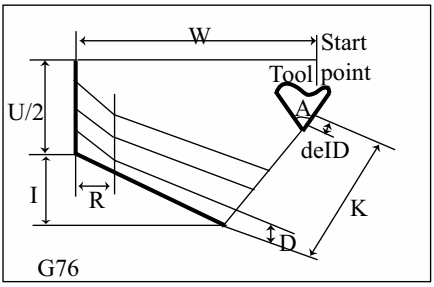
From the programmer's point of view, it is necessary that frequently used machining operations are defined in fixed format and used like subprograms when a part program is edited. A series of machining operations that are used repeatedly in NC machining are defined as one block that is called a "fixed cycle". The fixed cycle for turning can be classified into two types as shown in Table 8.6. Figure 8.20 shows G92, which is the simple fixed G-code for threading, and G76, which is the complex fixed G-code for threading. Compared with the tool path of the simple fixed cycle, that of the complex fixed cycle is complicated. However, it is relatively simple to generate the toolpath from the input data.

Table 8.6 Machining strategy data

Type	Code	Description	type	Code	Description
Single Fixed Cycle	G90	Turning (Cutting Cycle A)	Complex Fixed Cycle	G70	Finishing
				G71	Outer turning
	G92	Thread cutting		G72	Facing rough
	G94	Facing (Cutting Cycle B)		G73	Pattern repet.
				G74	Peck drilling in Z-axis
				G75	Grooving in X-axis
				G76	Thread cutting



G92 U_ _ W_ _ R_ _ F_ _
U: Incremental value of coordinate system X
W: Incremental value of coordinate system Z
R: Taper depth (incremental value, sign required)



G76 U_ _ W_ _ I _ _ K _ _ deID_ _ E_ _A
U: *X*-axis distance from start to end point
W: *Z*-axis distance from start to end point
I: Radial difference at start and end of thread
K: Thread height (radial data)
deID: First cut depth (radial data)
E: Lead of thread
D: Finish allowance

Fig. 8.20 Simple and complex G-codes for threading

8.6.2 Turning Cycle for Arbitrary Shape

8.6.2.1 Characteristics of Machining Cycles for Arbitrary Shapes

The G-code cycles mentioned in the above section are used for generating tool paths for a cylindrical part. In order to apply them successfully it should be assumed that the radius of a part increases or decreases consistently and that tool interference does not occur. However, a forged part or cast part typically has arbitrary shape and, in this section, the roughing cycle for these will be addressed. The roughing cycle generates the toolpath without tool interference by considering the geometry of the tool. It does not generate toolpaths for regions where material is absent in order to prevent cutting air. If there is a region where tool interference cannot be avoided, the region is not cut and remains to be cut in a subsequent operation.

For example, as shown in Fig. 8.21, the dotted line that represents the toolpath without air-cut is an appropriate tool path for obtaining the finished part from a cast workpiece.

8.6.2.2 Toolpath Algorithm

The cycle algorithm for generating an optimal toolpath is executed using eight steps. The steps are as follows.

In the first step, the workpiece shape and desired shape are specified (Fig. 8.22).

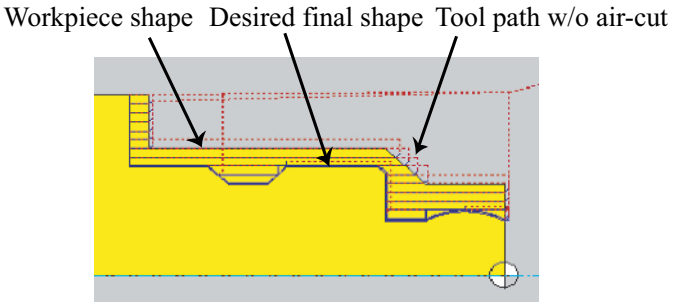


Fig. 8.21 Appropriate toolpath

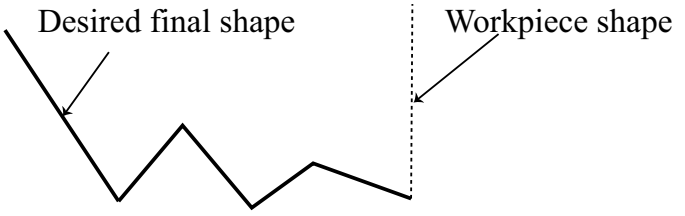


Fig. 8.22 Workpiece and desired shapes

In the second step, the collision-free region (machineable region) is calculated based on the cutting edge angle (side cutting edge angle and end cutting edge angle) of the tool, cutting angle, tool imaginary nose, tool type, tool holder's shape and workpiece shape, see Fig. 8.23.

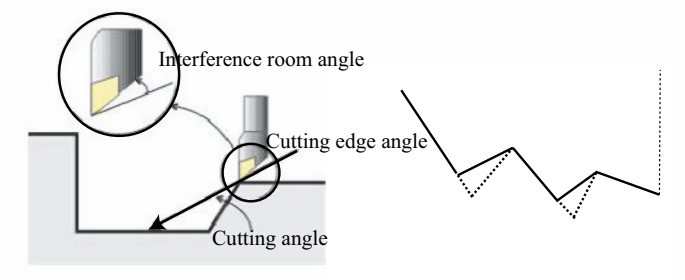


Fig. 8.23 Collision-free region calculation

The cutting angle is calculated as follows, based on the cutting edge angle and interference room angle.

Cutting angle = cutting edge angle - interference space angle (in general, 3 or 5 degrees)

Based on the computed cutting angle, the machineable region that prevents collisions between tool and workpiece and over-cut is calculated.

In the third step, the offset profile is generated by offsetting the machineable region from the second step within the finish allowance and with the tool nose radius.

In the fourth step, by combining the offset profile with the original profile of the part, a new profile is generated. In the case where the blank material is a cylinder, a new profile is generated by adding the linear profile of the cylinder, SA[], to the offset profile, S[], as shown in Fig. 8.24a. In the case where the blank material has an arbitrary shape, as with a cast part, the profile to be machined is created by combining the profile of the part, SA[], with the offset profile, S[], as shown in Fig. 8.24b.

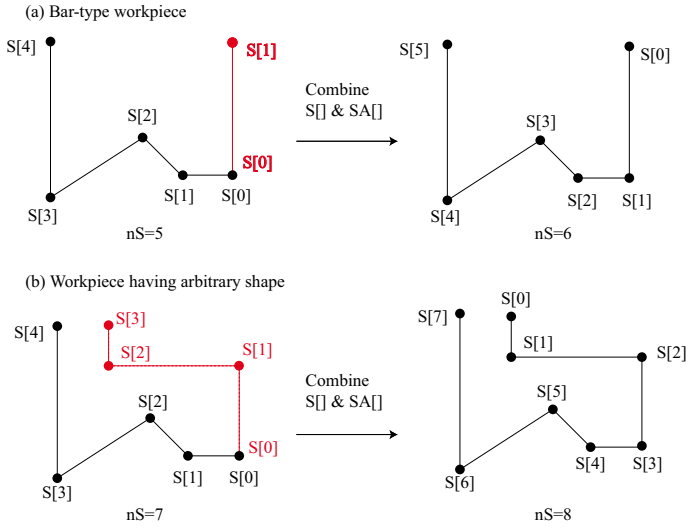


Fig. 8.24 Profile combination

In the fifth step, peak points and valley points are sought from the S[] obtained from the fourth step, and the total number of peak points is counted.

Peak point (P_i): $\{ P_i | X_i \geq X_{i-1}$ and $X_i > X_{i+1}, \forall i\}$
where, i is the index of a point on the profile.
If $X_i = X_{i-1}$ and $X_i < X_{i-2}$, P_i is not peak point.
Valley point (V_i): $\{V_i | X_i \leq X_{i-1} \&\& X_i < X_i, \forall i\}$
where, i is the index of a point of the profile.

In Fig. 8.24a, S[3] is a peak point and S[1] is a valley point. In Fig. 8.24b, S[5] is a peak point and S[4] is a valley point.

In the sixth step, the profile from the fourth step is divided into multiple profiles at the valley points and the divided profiles are stored in a buffer. In the case of the profile shown in Fig. 8.24a, the profile is divided at valley point S[1] as shown in Fig. 8.26.

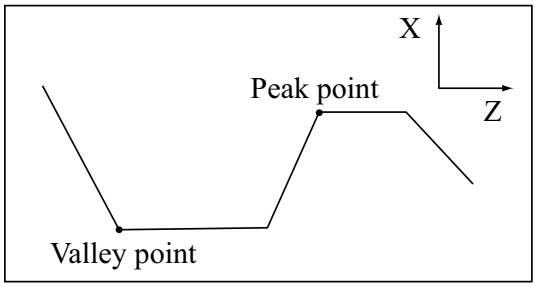


Fig. 8.25 Peak and valley points

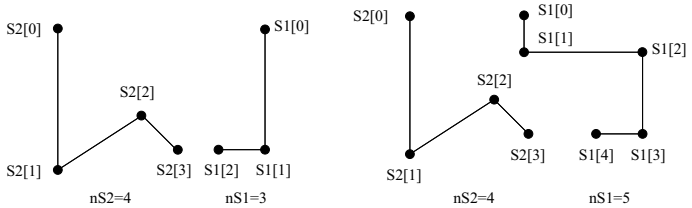


Fig. 8.26 Divided profiles

In the seventh step, the toolpath is generated based on the divided profiles and the specified cutting depth, as shown in Fig. 8.27a.

Stock_removal_path ( ) {

- The peak points are sorted in terms of the X position and stored in peak_array[].
- The number of cutting layers is calculated ($num = (Xe - Xs)/feed + 2$).
- From S1[] (first layer) and cutting depth are calculated the intersection point, cross_S1[] and cross[].
  Where, cross[] is the path whose X position is constant and cross_S1[] is the intersection point between cross[] and S1[].
- From S2[] and cross[], cross_S2[] is computed.
- The tool path is generated based on cross_S1[], cross_S2[], S1[], and S2[].

}

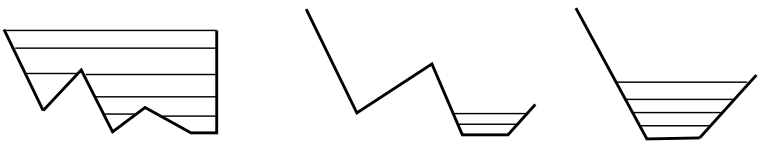


Fig. 8.27 Cutting toolpaths for turning

The eighth step, after the above steps have been completed, checks whether the current machined profile is the last peak profile. If yes, the cycle is terminated and, if not, the region to be machined is recalculated as shown in Figs. 8.27b and 8.27c.

Consequently, as the turning cycle for machining the part with arbitrary shape generates the toolpath automatically using the data on the workpiece shape, finished shape, and tools from the operator, it allows the unskilled operator to create the part program quickly.

8.6.3 Corner Machining Cycle

As mentioned in the previous section, an uncut area due to the tool shape can remain after completing the machining by the specified tool in the case of turning. This is why a toolpath is not generated in a region in which tool interference occurs. Therefore, in order to machine the uncut region after roughing, partial machining is executed after selecting a different tool. In the case of turning this is called "corner machining".

The toolpath for corner machining is generated based on the intersection points (A and B) between the uncut area and the finished shape, cutting depth d, and finishing allowance k. The details of the algorithm can be summarized, with Fig. 8.28, as follows:

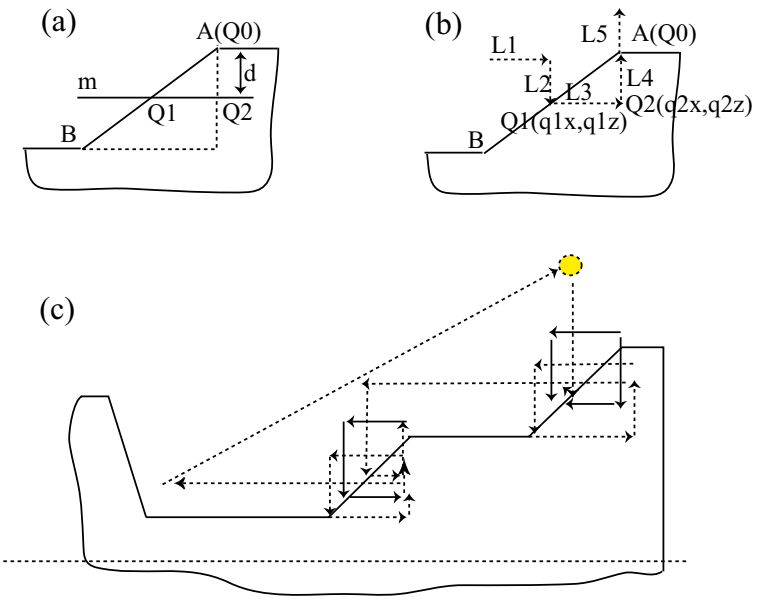


Fig. 8.28 Corner machining geometry

STEP 1. Cutting depth d and finish allowance k are input by the user and intersection points A and B are retrieved from the previous roughing cycle, (Fig. 8.28a).

STEP 2. The line m, which is at cutting depth d below the highest Z position of the corner, is defined. The intersection points $Q1$ and $Q2$ between the line m and the uncut area at the corner are calculated, (Fig. 8.28a).

STEP 3. The approach path for approaching the uncut area and the net-cut path for machining are generated. As shown in Fig. 8.28b, $L1$ and $L2$ are generated as the approach path for approaching and $L3$ and $L4$ for machining from $Q1$ to $Q0$ through $Q2$ are generated. In addition, $L5$ as a rapid path for retracting to the safety plane is generated.

STEP 4. The line m moves in steps of cutting depth d along the negative Z-axis. STEP 2 and STEP 3 are repeated until the Z position of the line m is smaller than the lowest Z position of the uncut area, (Fig. 8.28c).

STEP 5. If there is more than one uncut area, STEP 2, STEP 3, and STEP 4 are repeated for each uncut area. The join paths connecting the paths obtained from STEP 3 are generated and inserted. Finally, the rapid path for moving to the tool retract position is generated and inserted, (Fig. 8.28c).

This algorithm can be summarized as the procedure chart in Fig. 8.29.

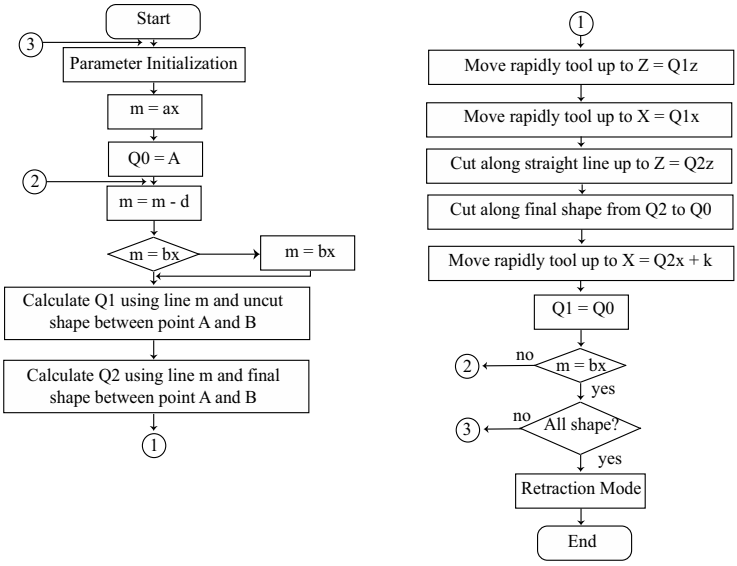


Fig. 8.29 Procedure chart for corner machining

8.6.4 Drilling Sequence

Typically, when the NC program for drilling multiple holes is generated, the programmer first classifies the holes into the groups depending on the hole shape and generates a program where the holes belonging to the same group are machined in a row. After completing machining of the holes in the group, holes belonging to another group are machined. In order to machine a hole, it is typical to use more than one tool. For example, in the case of tapping, center drilling, drilling, boring, and tapping should be executed one after the other. Therefore, if we consider the usage of tools related to drilling, the sequence of tools is considered as follows:

Group 1: $T_{11}, T_{12}, \ldots\ldots\ldots T_{1a}$
Group 2: $T_{21}, T_{22}, \ldots\ldots\ldots T_{2b}$
Group M: $T_{m1}, T_{m2}, \ldots\ldots\ldots T_{mm}$

where tools used in one particular group may be used in another group. Therefore, if the usage sequence of tools is well determined, it is possible to decrease the number of tool changes and hence the machining time.

However, if machining is executed group by group, the same tool may be used several times, which increases the tool change time and hence the total machining time. Therefore, it is necessary to make an NC program that reduces the number of tool changes.

Supposing that more than one hole can be classified into several groups and a particular tool can be used for holes belonging to different groups. In this case, if the tool is used for all the holes in a row, the number of tool changes is decreased as many as $M - 1$ times, where M is the number of groups where the tool is used.

The generation procedure of an NC program for drilling can be divided into three steps;

1. In the first step, the individual part program for each hole with different shape is generated.
2. In the second step, the usage sequence of the tools used in several groups is determined.
3. In the third step, the NC program for complete drilling is generated depending on the usage sequence of the tools.

In detail, the above steps are described as follows (Fig. 8.30). First, the usage sequence of tools is determined according to the shape of the holes. In the example, tools A, B, C, D, and E are used sequentially in the first part program P1. For the second part program P2, tools A, F, C, G, and E are used sequentially. For the third part program P3, tools H, I, C, J, and K are used sequentially.

After completing the first part program, the second, and the third, a check is made as to whether tools used for common operations exist. If these do exist, the tool that is used in the most operations is selected. In Fig. 8.30, this is tool C, which is used in three part programs.

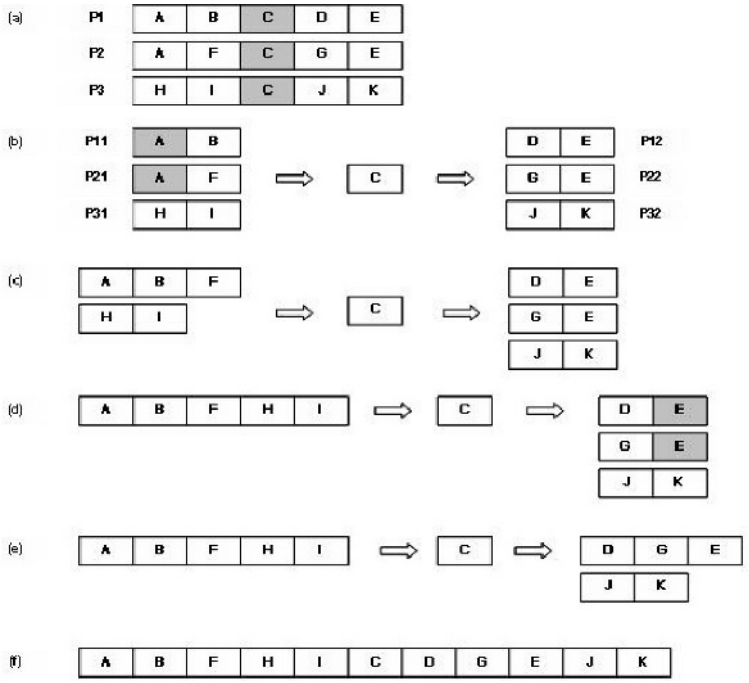


Fig. 8.30 Part programs to be executed

After selecting the common tool, each part program is divided into three parts; the part which should be executed before using the common tool, the part that is executed by the common tool, and the part that is executed after using the common tool. In addition, it is necessary to divide the tools in each group into 1) the tools used before the common tool, 2) the tools used after the common tool, and 3) the common tool.

In Fig. 8.30, P11, P21, and P31 are the part programs that should be executed before using the common tool. The tools used in P11, P21, and P31 are as follows.

P11: A, B
P21: A, F
P31: H, I

In Fig. 8.30, P12, P22, and P32 are the part programs that should be executed after using the common tool and the tools used in P12, P22, and P32 are as follows.

P12: D, E
P22: G, E
P32: J, K

After that, the above steps are repeated until no shared tool is found in P11, P21, P31, P12, P22, and P32,

The tool sequence in P11, P21, and P31 is determined as A'B'F'H'I.

The tool sequence in P12, P22, and P32 is determined as D'G'E'J'K.

Finally, the usage sequence of all the tools is determined as A' B' F' H' I' C' D' G' E' J' K. After determining the usage sequence of tools, the part programs are reassembled depending on the determined tool usage sequence.

8.7 Summary

The part that provides the user interface in the CNC system is the MMI unit. For the design of an MMI that allows a user to edit a part program, operate the machine, and monitor the machine status, it is necessary to consider ergonomics, design, and the user's aptitude as well as technological aspects.

In order to develop an MMI system, not only the user applications but also the design of the kernel layer for connecting to the NCK and the applications for monitoring the machine status, operating the NC, editing the program, and managing parameters are required.

The program-editing function, which is one of the key MMI functions, has advanced in a variety of versions developed by many CNC makers. Recently, conversational programming systems that can be used on the shopfloor have become a basic programming tool. The conversational programming system makes it possible for an unskilled user to generate a part program by allowing users to input data in an interactive way.

The core of the conversational programming system provides functions for generating interference-free toolpaths whose machining time is minimized and in which the number of tool changes is minimized, verifying tool paths and the finished part, and estimating machining time based on the minimum user input.

Chapter 9
CNC Architecture Design

It is necessary to design the architecture of hardware and software modules in order to implement a CNC system consisting of a variety of modules such as NCK, MMI, and PLC. System programming for operating these modules in real time is also required. In this chapter, the main functionalities and commands of real time operating systems (OS) for real-time programming systems will be described. Through investigation of multi-processing hardware architecture, the user will learn the basic approach for designing the architecture of a CNC system that requires the guarantee of real-time operation.

9.1 Introduction

A CNC system consists of the NCK unit that is composed of the interpreter module that interprets the part program, the interpolator module that creates the moving path of the tool, the acceleration/deceleration module that smoothes the axis movement, and the position control unit that controls the servo motor based on the feedback signal and interpolation result. In addition to the NCK unit, a CNC system contains an MMI unit that enables a user to operate the CNC, monitor the operation status and messages, and make a part program. Finally there is the PLC unit that logically controls the machine except for the servo motors.

In the CNC system, various tasks from the NCK, MMI, and PLC units are executed simultaneously and each task requires real-time operation. Therefore, it is essential to use a real-time OS in order to operate various real-time tasks in a multi-processing environment. In addition, the design of the architecture of hardware and software tasks to realize CNC system is required. In conclusion, the following must be considered in order to design a CNC system:

1. Real-time operating system (the kind of system call and usable hardware)
2. The architecture of hardware (the number of CPUs and the way of connecting between modules)

3. The architecture of the software (the design of task modules and the system kernel)

A real-time OS provides various functions such as task scheduling, inter-task communication resource sharing, and task synchronization in order to use the hardware resources effectively. Accordingly, the CNC designer should consider simultaneously the hardware resource to be managed and the functionality (or performance index) of the OS.

Recently, desktop PCs have been used as the basic hardware for industrial control systems. However, a novel idea is required to build a CNC system utilizing a desktop PC and PC operating system because the DOS or Windows used widely as the OS of a desktop PC cannot support the real-time requirements of a CNC system. One idea to meet the real-time criteria of PC-based CNC systems is to utilize a real-time OS and non-real-time PC OS simultaneously.

When the hardware architecture of a CNC system is designed, the number of processors needed to satisfy the real-time processing requirements of NCK, PLC, and MMI should be considered. If an architecture based on multi-processors is designed, the communication method between processors should be considered.

In terms of software, it is necessary to divide all the functions of the CNC system into appropriate modules from a functional point of view. Furthermore, the design of the system kernel, which plays the role of booting up the CNC system, terminating the CNC system, communicating data and events to the tasks, and switching tasks is also required.

While the CNC system is being implemented, an adequate development language should be selected and how to debug should also be considered. Because typical debugging tools for traditional procedural systems are not efficient and appropriate as debugging tools for real-time systems, a compiler in which an appropriate debugging tool is supported should be selected.

In addition, it is necessary to consider how to increase the reliability of the system and how to program for the real time nature of the task. For example, how to handle the exceptions that occur during system execution should be considered and the use of assembly code for the hard real-time tasks that must be carried out within extremely short times may also be necessary.

In this chapter, real-time OS and hardware architecture for designing CNC systems will be described. The system programming method under real-time OS, the basic structure of real-time OS, and the concept of resource handling will also be addressed. Moreover, with programming examples using the functionalities of real-time OS, the reader will learn the system programming method for implementing real-time CNC systems. By introducing the advantages and disadvantages of various hardware architectures which can be considered as the basis hardware for CNC system to the reader, the reader will gain the knowledge to design the hardware architecture that is appropriate for the development goal.

9.2 Operating Systems

An operating system is a software program for the purpose of using the hardware efficiently. As the purpose of an OS is to make it efficient to use hardware resources (processor, memory, input/output devices), the OS manages the usage of hardware resources between applications and enables applications to share data and hardware resources. In addition, an OS controls various input/output devices and user programs. In terms of the hardware architecture, user requirements, and environment to be used, an OS can be divided into several systems.

A multi-programming system increases the throughput of the processor by loading several user applications simultaneously into main memory, executing different user applications in turn using one processor. For example, when an application does not use a processor while using an input/output device because of the difference between the input/output device's speed and the processor's speed, the system increases the throughput of the processor by passing the right of processor occupation to another task in the queue waiting to be executed. Furthermore, skilled management of main memory is required in order to allow several programs to be in a ready-state simultaneously and processor scheduling is essential in order to manage the execution of several applications or tasks.

As a time-sharing system is the extension of the multi-programming concept, several tasks are carried out by one processor in turn by giving them specified time slices. This means that a time sharing system divides the processor usage time based on processor scheduling and multi-programming and passes the right of processor's usage to each user in turn in order to enable several users to use one system simultaneously. Therefore, a time-sharing system is appropriate when many users share one computer simultaneously. For this kind of system, techniques for switching tasks, managing memory, scheduling the processes, managing disk devices and managing files are required.

A multi-processing system is used to overcome the drawback of a single processor system, it is a system where multiple processors carry out tasks collaboratively through communication and memory sharing in order to improve the system performance and reliability. Multi-processing systems have been widely used as fundamental systems for modern CNCs. The key problem for the design of a multi-processing system is how to connect multiple processors and input/output processors with memory. There are differences between the OS of a multi-processing system and that of a single processor system and the differences will be described in another section.

The term "distributed system" denotes a system that carries out a task by using simultaneously multiple executors such as node, site, and computer in order to share the resources of multiple computers, increase the computing power by means of parallel and distributed execution, and increase the reliability of a system. In this type of system, data communication between computers is done via external communication cables. Therefore, each computer has its own memory and communicates with other computers via a high-speed bus or communication cable.

Because the term "real-time system" denotes a system that can begin the desired tasks within a specific time and complete it within a given time period, it is necessary that the response time (operational deadline) from event to system response is smaller than a specific time period. To be more precise, a real-time system does not mean the fastest system in a narrow sense, a real-time system means a system that does not work correctly in the case that the response time exceeds the specific time period. The ultimate goal of a real-time system is handling the data regularly occurring in real time or promptly handling occasional events. For example, in the case of a robot system that picks and moves objects on a conveyer belt, it is necessary to recognize the object and reach the object in a limited time. If picking up the object fails because of the delay of reaching the object, the robot system malfunctions. Therefore, the ultimate goal of a real-time OS for a real-time system is to provide an environment capable of completing applications in real time.

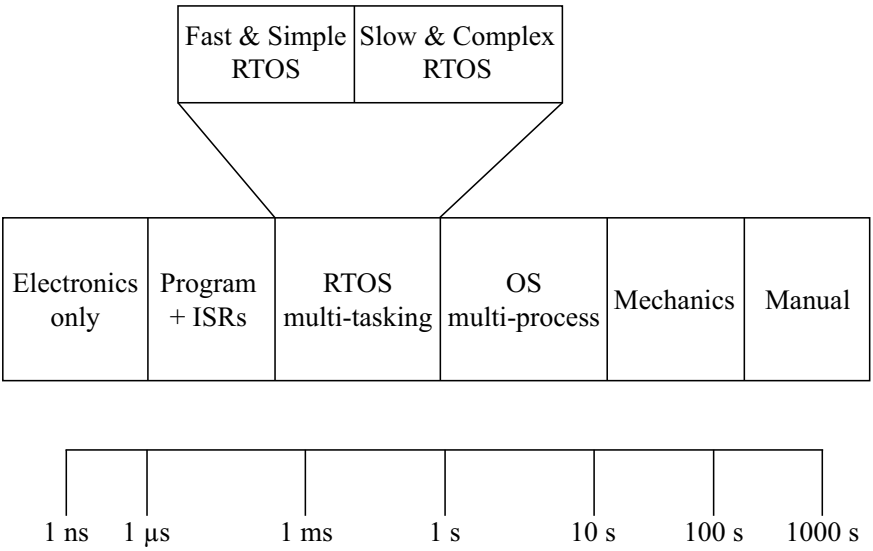


Fig. 9.1 Time spectrum of a real-time system

Figure 9.1 compares the time spectrum required by a real-time OS with the time spectrum required by a general system. In Fig. 9.1, the time on the *X*-axis indicates the deadline of the system (the amount of time is not absolute and can be varied depending on the processor's performance). Therefore, a real-time OS can be defined as a system that can carry out promptly a particular task every few to several tens of milliseconds, which is longer than the runtime of the interrupt service routine (ISR) but shorter than the OS runtime.

In addition, a real-time system means a system that can generate accurate output within a limited time in any surroundings as well as one that can respond quickly to external events or signals.

Accordingly, firstly in a real-time system, the exactness of time by a highly accurate scheduling function should be considered in contrast to a non-real-time system. In other words, allocating adequate resources to the tasks, initiating tasks, and finishing tasks have to be completed within the deadline. In order to achieve the exactness of time, it is necessary to get the property and priority of a task before executing it. The exactness of time can be achieved by a real-time scheduler that allocates resources and executes tasks based on the information of the tasks including time limits.

Secondly, the reliability of the OS also has to be considered. Because real-time systems are used in surroundings where only one error results in unrecoverable disaster, it is essential to guarantee robustness from errors and predictability of trouble. Therefore, all policies and techniques for a real-time system should be designed for predicting problems occurring due to time limits and guaranteeing reliability of the running tasks by preparing for occurrence of problems and removing them.

Real-time systems can be divided into two types: hard real-time systems and soft real-time systems. In general, a hard real-time system executes iteratively within several milliseconds or several tens of milliseconds. In the case of hard real-time systems, it is not permitted that completion of a task exceeds the deadline. If completion of the task exceeds the deadline, the completion result has no meaning and, in the worst case, the system may be destroyed. Therefore, a hard real-time system can be defined as a system in which damage is large in the case of missing a deadline.

On the other hand, in a soft real-time system, tasks execute iteratively within several tens or several hundreds of milliseconds, in general, and missing a deadline results in slight damage or loss of system efficiency. Accordingly, a hard real-time system means a system that should never miss a deadline and a soft real-time system means a system that does not miss a deadline.

From the point of view of a real-time system, a CNC system can be defined as a hard real-time system. Generally, position control of a CNC system is repeated every few milliseconds or every few hundreds of nanoseconds and delay in position control stops the machine and results in a difference between the programmed path and the actual path. Therefore, the OS for a CNC system must be a real-time OS that guarantees the hard real-time property on a single processor or multiple processors.

9.3 Real-time Programming

A program is a set of instructions that lists various operations and data objects such as constants and variables. The procedure of creating a program is called “programming”. In general, how to make a program can be classified into two types; sequential programming and parallel (real-time) programming. Sequential programming means that the instructions of a program are executed sequentially in a fixed sequence (see Fig. 9.2a). The purpose of sequential programming is to convert an input into a particular output via an algorithm or a procedure. Therefore, in a sequential program, the execution of a program is not restricted by time. In terms of efficiency, the execution

time is considered and the output depends on the algorithm. Accordingly, it can be generated without detailed knowledge of the OS, hardware devices, and resources, since a sequential program can be executed sequentially in a fixed order depending on the input data.

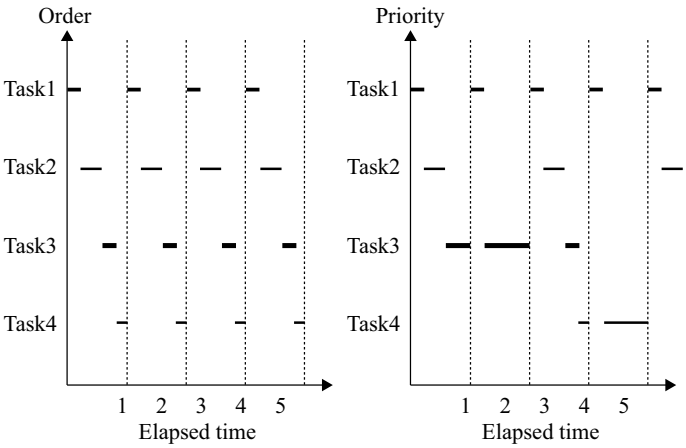


Fig. 9.2 Comparison of task operation sequence between sequential (a) and real-time programming (b)

On the other hand, since the task operation sequence in real-time programming is different from that in sequential programming, a programmer has to consider the environment where the program is to be carried out and cannot code the program indepently of hardware or system resources. Unlike the sequential program, where processes or tasks are carried out sequentially, multiple processes and tasks in a real-time program are executed in parallel. As shown in Fig. 9.2b, since each task passes the right to processor usage to the task with next priority after completing execution and the task with the lowest priority should be performed during the idle time, it is necessary to consider the execution sequence of tasks and feasible resources in real-time programming. For example, a CNC system carries out a variety of tasks simultaneously, such as the NCK tasks that interpret a part program and calculate the displacement of each axis during the sampling time of position control, the monitoring task that checks the abnormality of a machine every specified time, and the MMI task that displays the status of machine.

Therefore, for real-time programming, the programmer must make an effort to consider how to respond to the input signal interrupt every constant time unit, how to calculate task execution time, how to pass the right to processor usage to another task, how to switch the context and how to allocate resources (*e.g.*, memory) to tasks or processes.

What should be considered specifically for real-time programming and what should be needed to respond to external requests cannot be represented by the ordinary functions for a sequential program. Since a variety of programs or tasks are

executed simultaneously and cooperation between processes or tasks has to be considered in a real-time system, special functions for real-time programming are required. They can be summarized as follows:

1. Multi-tasking function that enables execution of more than one task by a single processor or multiple processors.
2. Synchronization function that enables adjustment of the execution of tasks.
3. Inter-communication function that enables exchange of data between tasks.
4. Preemption function that enables stopping the task being executed in order to carry out a high-priority task and scheduling functions that enable management of the execution order of tasks.
5. Timer function that enables beginning a task at the right time.
6. Interrupt function and exception handling function that enables transmission of an asynchronous event to a processor.

The ultimate goal of a real-time OS is to provide an environment where it is possible to use the above-mentioned functions required for real-time programming. The CNC system designer must be able to do real-time programming by selecting a real-time OS that meets the requirements for system performance, studying the functions of the selected real-time OS, and using them efficiently. Therefore, in the following sections, the kernel structure of a real-time OS, system calls, and real-time programming will be addressed.

9.4 Structure of a Real-time OS

The operating system is the software that enables efficient and easy use of a hardware resource, as mentioned in the previous section. In general, the core of an OS is the kernel that handles a variety of system calls, being the interface between an application and the OS, based on a hardware system such as a processor, memory, and disk, as shown in Fig. 9.3. A real-time OS (RTOS) makes it easy to build real-time systems and its kernel provides the following functions:

1. Multi-tasking
2. Priority-based, pre-emptive scheduling
3. Synchronization and inter-process communication
4. Interrupt service

The functions of the modules that compose the kernel of a real-time OS is as follows:

1. *Process Manager*: This module is the key function of a real-time OS and makes multi-tasking possible. It makes it possible to switch context depending on the priority and condition of a process or task. Context switching is the particular procedure that occurs when a process makes a system call, a timer interrupt is issued, or an external interrupt signal is issued. During context switching, the

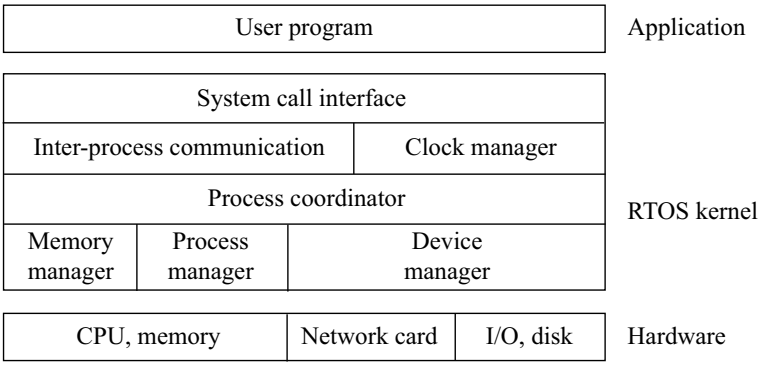


Fig. 9.3 Structure of a real-time OS

register values of the system are stored on a stack before stopping the task and the stored register values are restored when the stopped task is resumed.
Complex operations are required for the exchange of context information between the task stack and the system registers. Because the time spent switching context takes up the majority of the whole response time it is necessary to increase the chance of switching context and decrease the time spent in switching context by increasing the frequency of the system timer. The context switching time has a large influence on the performance of a real-time OS and is generally implemented in assembly code.
In order to manage tasks, an identification number and priority are assigned to each task. The priority can be specified by the programmer. The process manager is able to stop a running task to activate a task with higher priority and, for this, there is a priority-based preemptive scheduler.

2. *Memory Manager*: As this module plays the role of managing the memory among hardware resources, it provides the functions for allocating and freeing memory. In the case of C programming, an automatic memory allocation method using the malloc function is used for allocating memory but, in the case of real-time programming, a memory-handling method to specify directly the size and address of the allocated memory is used. This is for preventing the memory of each task from coming into conflict. Because particular tasks are always executed in real-time programs, this module provides a simple and fast memory management function by using a flat model instead of a paging model.
3. *Process coordinator*: In a real-time OS, a process exists in one of a number of states, such as executing state, suspended state, and ready state. A process coordinator performs state transition according to the priority and schedule of a task. It plays the role of controlling the status of a task, such as creating a task, deleting, suspending until a specified condition is satisfied, and resuming if the specified condition is satisfied.
4. *Inter-process Communication (IPC)*: a process must be able to exchange data with other processes or the interrupt service routine (ISR) in order to be selectively

synchronized with other processes. In addition, it is possible to occupy exclusively protected hardware resources in order to prevent conflict between processes. To meet these requirements, a semaphore mechanism is used for synchronization of processes and controlling critical regions and a message queue and mail box are used for exchanging data between processes.

5. *Clock Manager*: Basically, this module plays the role of real-time timer for multiprocessing. It is used for invoking the sleep function for delaying the execution of a task and the wake-up function for synchronizing the execution of tasks.
6. *Device Manager*: This module plays the role of managing input/output devices (*e.g.*, RS232C, Ethernet, I/O, Printer, and Servo, etc.) connected to the CNC system via the device driver. It provides the input/output management functions to enable consistent interfacing for all kinds of input/output devices regardless of the kind of device. By providing common input/output instructions (*e.g.*, open, close, getc, putc, read, and write), it enables a programmer to transmit data to a communication port or store data on disk with the same instructions, only using different device identification numbers.

Besides the above-mentioned modules, a real-time OS consists of a file manager, which carries out file handling such as creation, deletion, copying, and renaming; and an auxiliary memory manager that handles large-sized auxiliary memory devices such as a hard disk. In this chapter, the core functions of the real-time OS kernel such as management of a process, protection of a resource, and communication and cooperation between processes that is needed to implement embedded systems such as a CNC system will be described in detail. Also, using real-time programs based on kernel functions, how to do system programming for a CNC system will be described.

9.5 Process Management

The process management method, being a basic element defining processor activity in a real-time OS, will be addressed. The process is a basic element defining processor activity in a real-time OS and, in other words, a process is a running program. A process is composed of a code region where program instructions are stored, a data region where process variables are stored, the heap area, where dynamic memory is allocated, and the stack area, where arguments of subroutines, return addresses, and temporary variables are stored. Although a program is edited and compiled in the same high-level language, a program operating on a different hardware system is executed as a different process, with the different code regions, heaps, and stacks. Therefore, a process is an instance that has CPU register values, the addresses of code/data/stack, and a pointer that refers to the next instruction to be executed. The basic data about the execution of the process is defined as the ‘context’. A process control block is the region where all data changed by a process are stored, it includes

process status, program counter (the pointer of the instruction to be executed next), schedule data, memory management data, and input/output status data.

9.5.1 Process Creation and Termination

A mechanism to create and terminate processes is required in order to execute multitasking, executing multiple processes by single processor.

1. *Creation*: Several processes can be created with a system call for process creation. The process that creates a process is called the 'parent process' and the generated process is called the 'child process'. The parent process can share resources with the child processes.
2. *Termination*: After completing the last instruction, it is possible to request the OS to terminate a process or delete a particular process by a system call from another process. Typically, the system does not permit the existence of a child process after destruction of its parent process. Therefore, all child processes should be terminated when a parent process is terminated.

9.5.2 Process State Transition

In order to execute multiple processes efficiently, it is necessary that the status of processes can be changed to a variety of states. According to the process activity, a process in a real-time OS can be classified as being in one of six kinds of state.

1. *Current state*: This means the case that a process occupies a processor and is being executed (running). In the case of a single processor, only one process can be in the current state.
2. *Ready state*: This means the case that a process does not currently occupy a processor but can be executed at any time.
3. *Receiving state*: This means the case that a process is awaiting a message or a mail from another process.
4. *Sleeping state*: This means the case that a process in 'sleeping' during a specified time.
5. *Suspended state*: This means the case that a process has stopped execution. When a process is created it is always in this state.
6. *Waiting state*: This means the case that a process is waiting for an external event or semaphore.

The terms mentioned above are not definitive and different names are used depending on the operating system. However, the names in different operating systems can be matched with the above-mentioned six states.

Figure 9.4 depicts transitions between the above-mentioned six states. Initially, a process that is generated by a "create" instruction is in Suspended state. The state of the process passes into the Ready state through a "Resume" instruction. When a resource is allocated to the process by the scheduler, the process moves to the Current state. A process in Current state moves to another state (*e.g.*, Waiting state, Ready state, and Suspended state) through "wait", "resched", "suspend" instructions. The transition of the process state continues until the "delete" instruction is called.

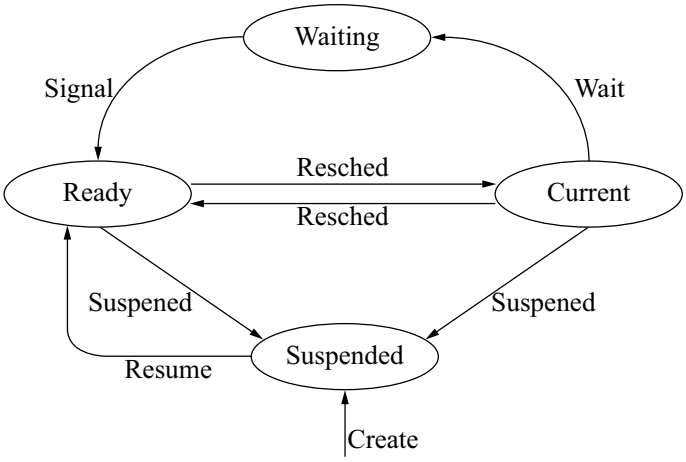


Fig. 9.4 Diagram of Process State Transitions

Figure 9.5 shows a program example to implement process management. Task 1 is created by rt_create(), it is suspended by rt_suspend(), it transfers to Ready state by rt_resume(), and, finally, it is deleted by rt_delete(). The bold elements in the example code show the instructions of the real-time OS.

9.5.3 Process Scheduling

A special strategy to select the next task from among the tasks waiting for execution is necessary in order to maximize the utilization of a processor. The task that should be carried out in the specified time is selected by a scheduler. The scheduler is a service module that is called whenever a task in the Current state releases possession of a processor.

The majority of real-time operating systems use a scheduling algorithm for managing several real-time tasks that require real-time execution in a preemption multitasking environment. If pre-emption scheduling is not done, the hard real-time property cannot be achieved and, in conclusion, it is impossible to guarantee correct be-

```
/* RTOS task management                    */
/*   - Create/Suspend/Resume/Delete a task   */

#include<conf.h>
#include<kernel.h>
#include<errno.h>

void main()
{
    int     err, tid;
    void  task1();
    struct timespec time;

    printf("[1. Create a task by the name of task1.]\n");
    tid = rt_create(task1, 1, INITPRIO+1, &err);
    if (err != RET_OK)
            printf("*** Error: can't create a task.\n");
    time.seconds = 5;
    rt_delay(time, &err);

    printf("\n[2. Suspend task1.]\n");
    rt_suspend(tid, 0, &err);
    time.seconds = 2;
    rt_delay(time, &err);

    printf("\n[3. Resume task1.]\n");
    rt_resume(tid, 0, &err);
    time.seconds = 3;
    rt_delay(time, &err);

    printf("\n[4. Delete task1.]\n");
    rt_delete(tid, 0, &err);

    printf("\n[------- End of test -----]\n");
}
```

Fig. 9.5 Process management program example

havior of the system. The opposite type of scheduler to the pre-emption scheduler is the non-pre-emption scheduler.

Using a non-pre-emption scheduler means that the operating system cannot stop the execution of a task during execution. In this case, since stopping a task can only be done by an interrupt, the design of the OS kernel is simple. However, because the OS kernel cannot control the execution rights of a task, a programmer has to plan the execution sequence of tasks in order to prevent a high-priority task from

waiting for completion of a low-priority task. Therefore, in general, a real-time OS does not use a non-preemption scheduler and the scheduling algorithms mentioned in the following sections includes the pre-emptive property.

9.5.3.1 First-Come, First-Served Scheduling

As First-Come, First-Served scheduling is the simplest scheduling algorithm, it allocates a resource according to the queue of requests. When the task is inserted into a ready queue the control block of the task is connected to the end of the queue. When the current task ends, the resource is allocated to the task at the head of the queue and the allocated task is deleted from the queue. Consequently, system resources are allocated by the sequence of the queue.

9.5.3.2 Time Slice

In the time-slice scheduling algorithm, time is split into intervals of the same length and a task is allowed to operate during a certain amount of a time slice. The execution sequence of tasks is typically determined by a round-robin method. After priority has been assigned to each task according to the task characteristics, round-robin scheduling is applied depending on the priority. Here, round-robin scheduling means that the execution sequence of a task follows a pre-specified order and the task is carried out only during the constant time interval. So, if the task finishes within its time interval, the task is deleted from the queue. However, if the task does not finish within the allocated time interval the task is added to the tail of the queue. The simplicity of the time-slice scheduling algorithm is a merit. However, when tasks with different characteristics are assigned to the same CPU, serious problems can occur. Therefore, this scheduling algorithm is generally used for soft real-time systems and is appropriate for background scheduling of regular tasks having long response times.

9.5.3.3 Priority

As a more complicated scheduling method, a method based on task priority can be used. Priority is allocated to each task and the scheduler allocates the processor to the highest priority task. If tasks have the same priority, they are executed by a First-Come First-Served scheduling method. The priority specified by a programmer can be changed while the task is being carried out.

In the case of pre-emptive scheduling, as soon as a task is inserted into the queue, the priority of the inserted task is compared with the priority of the task being executed. If the priority of the inserted task is higher than that of the currently execut-

ing task, the inserted task pre-empts the processor. In non-pre-emptive scheduling, the task is inserted at the head of the queue. A scheduling method where the priority can be changed during task execution is necessary, since forced pre-emption of tasks being executed is not desirable. Therefore, unlike the fixed-priority/static-priority scheduling where priority change is not permitted, dynamic-priority scheduling, where priority change is possible during system execution has been introduced.

The fixed-priority scheduling minimizes the execution burden of a real-time system and the Rate Monotonic (RM) algorithm is the most typical fixed-priority scheduling algorithm. In this algorithm, the priority is static and tasks with shorter periods are given higher priorities. The task with the highest priority that can be run immediately pre-empts all other tasks. In the RM algorithm, each task has its own static priority and the instance of each task is not given a new priority. Because the static-priority scheduling consumes less computing power and implementation is easier compared to dynamic-priority scheduling, it is widely used in real-time systems that require a deterministic guarantee with regard to response time.

In static-priority scheduling, only the task with highest priority may be executed. To overcome this problem, the priority of the task being executed is linearly decreased by the scheduler when its current time slice is gone. Therefore, the executing task comes to have lower priority than a waiting task. By using this method, it is certain that all tasks come to be executed. In consequence, dynamic priority assignment is done at the end of each time slice.

As another dynamic-priority assignment method, the aging method is used. In the aging method, the priority of a task becomes higher after each time slice. This method prevents the task with low priority from waiting endlessly and allows the task with lowest priority to be executed. In conclusion, because of the different initial priority, a task with high priority is executed more frequently than a task with low priority. Therefore, a task which has to be called frequently or promptly has high priority and a task in which long response time is permitted has low priority.

9.5.3.4 Fixed Sample Time

In fixed-sample-time scheduling, time is not divided into fixed slices but is sliced depending on the property of a task. This means that, in the case that the same time slice is assigned to all the tasks, a task that has not completed in the fixed time may be terminated without any result. To solve this problem, at the stage of defining a task, an adequate time period is specified for each task and the task is scheduled by using the individual software timer corresponding to the sample time.

9.5.3.5 Event-driven

The majority of scheduling methods assume periodic task service. However, event-driven scheduling is used for irregular tasks. This method is appropriate in the case when some task is fired by an event or data from a sensor.

Figure 9.6 shows an example of the task scheduling function and dynamic-priority assignment function. Task 1 and task 2 with the same priority are created by rt_create() and the scheduling function is stopped by rt_lock() for some specific time. After this time has passed, the scheduling function is resumed by rt_unlock() and the priority of task 1 is changed by rt_priority().

```
/* RTOS task scheduling              */
/*   - Lock/unlock scheduling        */
/*   - Change priority of a task     */

#include<conf.h>
#include<kernel.h>
#include<errno.h>

void main()
{
    int     err, tid1, tid2;
    int     flag;
    void  task1(), task2();
    struct timespec time;

    printf("[1. Create two tasks(task1, task2) which have the same priority.]\n");
    tid1 = rt_create(task1, 1, INITPRIO+2, &err);
    if (err != RET_OK)
            printf("*** Error: can't create a task\n");
    tid2 = rt_create(task2, 2, INITPRIO+2, &err);
    if (err != RET_OK)
            printf("*** Error: can't create a task\n");

    time.seconds = 5;
    rt_delay(time, &err);

    printf("\n[2. Lock scheduling.]\n");
    flag = rt_lock();
    time.seconds = 3;
    rt_delay(time, &err);

    printf("\n[3. Unlock scheduling.]\n");
    rt_unlock(flag);
    time.seconds = 2;
    rt_delay(time, &err);

    printf("\n[4. Let task1 have higher priority than task2.]\n");
    rt_priority(tid1, INITPRIO+1, &err);

    printf("\n[5. Delete task1.]\n");
    rt_delete(tid1, 0, &err);

    printf("\n[6. Delete task2.]\n");
    rt_delete(tid2, 0, &err);

    printf("\n[------- End of test -----]\n");
}
```

Fig. 9.6 Programming example of task scheduling

9.6 Process Synchronization

In a system based on multi-processing OS, all tasks can possibly be carried out simultaneously. Therefore, in order to guarantee the right execution sequence of tasks it is necessary for the OS to provide a synchronization mechanism between tasks.

The semaphore, which was proposed as a task synchronization and mutual exclusion method by Edsger Dijkstra in the 1960s, has been used in the majority of multi-tasking OS. Mutual exclusion, which enables access to a shared resource when a specific condition is met, will be described in the next section and in this section details of semaphores for task synchronization will be addressed.

9.6.1 Semaphores

Originally, the term 'semaphore' meant a railroad signal to indicate the change of a railroad line. Determining the usage of a shared resource according to the status of a semaphore is similar to determining whether a train goes or waits according to a railroad signal.

A semaphore can be changed only by P and V actions and is a variable with only integer values. Each process has a semaphore variable and whenever a process wants to access the shared resource, the value of the semaphore variable has to be checked. If the semaphore variable is equal to one, a process can access the shared resource. If the semaphore variable is zero, access to the shared resource is prohibited. In other words, a semaphore is a special variable that indicates whether a process can access the shared resource. If a semaphore variable is greater than zero this indicates that access to the shared resource is possible. Before access to the resource, a process records the usage of the resource via a P action. After using the resource, a process increases the value of the semaphore connected to the next process by one via a V action and passes access rights to the next process.

In conclusion, the behavior of the semaphore can be summarized as follows.

The P action decreases the semaphore variable by one and is performed by calling WAIT(semaphore variable). By a P action, whether a particular process can access a shared resource is checked. If the semaphore variable is greater than one, the process that connects to the semaphore variable can access the resource and the P action decreases the semaphore variable by one before access.

The V action increases the semaphore variable by one and is performed by calling SIGNAL(semaphore variable). It gives the access right to the next process. After increasing the semaphore value by one, a process connecting to the semaphore can access the resource during its scheduled execution.

A semaphore whose value is either 0 or 1 is called a "binary semaphore" and a semaphore whose value can be greater than one is called a "counting semaphore".

9.6.2 Using Semaphores

In order to use semaphores, a semaphore variable has to be created for each task and assigned to its individual task. In this section, in order to show synchronization by semaphore variables, two examples where three tasks exist, named 'A', 'B', 'C' respectively, are shown.

Figure 9.7 shows the first example. The state of three tasks is automatically moved to the execution state and ready state by the OS scheduler. In this example, synchronization between the tasks is not activated and the execution result is arbitrarily generated.

In the second example, the synchronization between the three tasks works using semaphore variables and, in conclusion, 'A', 'B', and 'C' are displayed in turn, as shown in Fig. 9.8. The semaphore variables are generated for each task by screate(), and "printa = screate(1)" is declared first in order to start the task corresponding to the "printa", semaphore variable. After this, tasks are created and moved to the ready state. Task 1 has the right to run because the semaphore variable "printa" is equal to one and the process displays 'A'. After displaying 'A', Task 1 signals the semaphore variable "printb" corresponding to the next execution task. Because "printb" is signaled by Task 1, Task 2 which is in wait state moves to execution state and displays 'B'. Next, Task 2 signals the semaphore variable "printc" and execution right is passed to Task 3. In conclusion, each task is executed one after the other using the semaphore mechanism and the execution result is as shown in Fig. 9.8.

9.6.3 Events and Signals

The synchronization mechanism using a semaphore is typical. However, it is not true that this method can be applied for all cases. In addition to a semaphore, an event or signal is widely used for implementing the synchronization mechanism.

The event method uses an event flag and is an appropriate mechanism for realizing synchronization when multiple events occur. The event flag that corresponds to a particular event is located in the event memory. So, if a particular event occurs the corresponding event flag is turned on. As soon as the event flag is turned on, the task that is waiting for that event moves into the ready state. The event flag plays the role of passing control and causes the OS to activate the appropriate event handler.

The signal method is slightly different from the event method and works like an interrupt. If a particular signal is fired, the task currently running is stopped and the task corresponding to the signal is called. This is very similar to the way that the interrupt service routine (ISR) is activated by an interrupt.

```
/* Coordinated by scheduler for displaying 'A', 'B', 'C' */

#include<conf.h>
#include<kernel.h>

void main()
{
    int     proc1(), proc2(), proc3();

    printf("\n Display 'A', 'B', 'C'\n");
    printf("    Output....\n\n");
    rt_resume(rt_create(proc1, INITSTK, INITPRIO, "proc1", 0, 0) );
    rt_resume(rt_create(proc2, INITSTK, INITPRIO, "proc2", 0, 0) );
    rt_resume(rt_create(proc3, INITSTK, INITPRIO, "proc3", 0, 0) );
}

proc1()
{
    int i;
    for (i = 0; i < 1000; i++) {
            printf("A");
    }
}

proc2()
{
    int i;
    for (i = 0; i < 1000; i++) {
            putc(CONSOLE, 'B');
    }
}

proc3()
{
    int i;
    for (i = 0; i < 1000; i++) {
            putc(CONSOLE, 'C');
    }
}
```

Output result

```
Display 'A', 'B', 'C'
Output....
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBB
BBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCAAAAAAAAAA
AAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCC
```

Fig. 9.7 Programming example without the synchronization mechanism

```
/* Coordinated by semaphore for displaying 'A', 'B', 'C' */

#include<conf.h>
#include<kernel.h>

void main(int argc, int argv)
{
    int     proc1(), proc2(), proc3();
    int     printa, printb, printc;
    printa = screate(1);
    printb = screate(0);
    printc = screate(0);
    printf("\n Display 'A', 'B', 'C' by turns using semaphore and three process \n");
    printf("    Output....\n\n");
    resume(create(proc1, INITSTK, INITPRIO, "proc1", 2, printa, printb) );
    resume(create(proc2, INITSTK, INITPRIO, "proc2", 2, printb, printc) );
    resume(create(proc3, INITSTK, INITPRIO, "proc3", 2, printc, printa) );
}

proc1(printa, printb)
{
    int i;
    for (i = 0; i < 10; i++) {
            wait(printa);
            printf("A");
            signal(printb);
    }
}

proc2(printb, printc)
{
    int i;
    for (i = 0; i < 10; i++) {
            wait(printb);
            putc(CONSOLE, 'B');
            signal(printc);
    }
}

proc3(printc, printa)
{
    int i;
    for (i = 0; i < 10; i++) {
            wait(printc);
            putc(CONSOLE, 'C');
            signal(printa);
    }
}
```

Output result

```
Display 'A', 'B', 'C'
Output....
ABCABCABCABCABCABCABCABCABCABCABCABCABCABC
ABCABCABCABCABCABCABCABCABCABCABC.....
```

Fig. 9.8 Programming example of task synchronization by using semaphores

9.7 Resources

9.7.1 System Resources

A resource means not only hardware such as a printer or a disk but also things that the task being executed accesses such as variables in main memory. In multi-programming, competition between tasks for the use of resources sometimes occurs. If this competition is not effectively managed, a system may work abnormally or it may be terminated. Therefore, concern about resource protection is one of the key issues in multi-programming theory.

As traditional examples of resource protection, airplane ticket reservation systems and bank accounting systems are often given. Before a flight, the seats of an airplane are located in the memory of the ticket reservation system. In order not to allocate the same seat to more than one customer when tickets are issued at the same time, the ticket reservation system must protect the seats resource. If different tasks use the same variables and modify them without the pre-specified sequence, unexpected problems can result.

For example, suppose that two tasks read and modify the same variable. If an interrupt is fired as soon as one task reads the variable, the other task can modify the variable while the task is in wait state. The former task cannot know that the variable has been changed and resumes execution based on the changed variable. In a multi-processing environment, a task can be pre-empted at any time and resume at any other time. In this case, more than one task can access the same resource without any restriction. Therefore, the variable to which access by multiple tasks is allowed has to be regarded as a resource whose protection is necessary and an adequate protection mechanism is needed to protect this variable. Accordingly, to avoid competition, resource allocation should follow a pre-specified mechanism.

The fundamental theme for resource protection is that the resource that is occupied by some task should not be changed by another task. The most difficult thing for resource protection in a multi-processing environment is that any task can interrupt any other task. The programmer cannot control and detect the time of the interrupt. Therefore, the first method to guarantee resource protection is to prohibit interrupts while the resource is occupied by another task. This is a way to prohibit the processor's response due to interrupt by force. This can be accomplished by implementing a "critical section", a series of instructions or blocks that cannot be stopped by another task. Resource protection can be guaranteed by disabling interrupts before the task enters the critical section and enabling interrupts after the task leaves the critical section.

9.7.2 Mutual Exclusion

It is possible to prevent system failure by allowing only one task at a time to have access to a common variable. While only one task is using the common variable, the other tasks that want to access the same variable wait for the completion of the task. After the task finishes the usage of the variable, one of the waiting tasks is allowed access to the variable. Allowing only one task to have access to a common variable among the tasks that want to access it is called the "mutual exclusion mechanism".

When some task has access to particular common data, the task is said to be in its critical section. Each task has a code segment called the "critical section". In the critical section, the task can change common variables, update tables, and read and write files. Therefore, when one task is in its critical section, the mutual exclusive mechanism is required not to allow other tasks to execute their critical sections. A progress mechanism and bounded waiting condition for managing the tasks that need to have access to their critical sections is required. If a task stops in its critical section, it is necessary for the OS to allow another task access to its critical section by freeing the mutual exclusive condition.

Figure 9.9 shows a mutual exclusive mechanism using a semaphore when three processes share one resource.

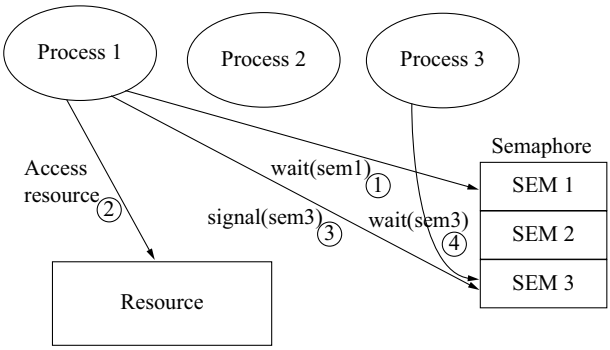


Fig. 9.9 Operating order of mutual exclusive mechanism using a semaphore

If Process 1, Process 3, and Process 2 are executed one after the other, the behavior is as follows:

1. When the OS scheduler puts Process 1 in execution state, Process 1 checks the semaphore variable SEM1.
2. If the value of SEM1 is more than 1, Process 1 accesses the resource.
3. After finishing usage of the common resource, SEM3 is signalled for Process 3 that will use the common resource.
4. If Process 3 is started by the scheduler, Process 3 checks the semaphore variable SEM3 and has access to the variable.

The method of realizing mutual exclusivity based on semaphores as described above is very similar to the semaphore-based synchronization method shown in Fig. 9.8. The mutual exclusive method can be realized by Wait() for waiting for the semaphore to access the resource and signal() for signalling the semaphore to allow another process access to the resource after completion of the usage of the resource.

9.7.3 Deadlock

In a multi-tasking programming environment, multiple tasks compete to use limited resources. If it is impossible to use a resource when a task requests that resource, the state of the task becomes the waiting state. The case when the task state does not change because the resource requested by the task in waiting state is occupied by other tasks in waiting state can occur. For example, let us suppose that the system has one printer and one tape drive, task 1 occupies the tape drive, and task 2 occupies the printer. If task 1 requests the printer and task 2 requests the tape drive, the execution of the two tasks is stopped until one or other frees occupation of the printer or the tape drive. The case when a system cannot continue execution because of this sort of occurrence is called deadlock. However, since the majority of operating systems do not provide a function to prevent deadlock, it is necessary for a programmer to exercise caution. Practically, it is possible to prevent deadlock by finding the occurrence condition of the deadlock and avoiding this condition. Theoretically, the necessary and sufficient condition of the deadlock occurrence can be summarized as follows:

Mutual exclusion: at least one resource is governed by a non-sharing method. This means that only one process can use a resource at one time. If another process requests the resource, the execution of the requesting process is delayed until the resource is freed. In conclusion, only one process can use a resource at any specific moment.

Hold and Wait: One process should occupy at least one resource and this process must wait to occupy additional resources held by another process.

Non-Pre-empted allocation: It is impossible to pre-empt the resource. The occupied resource cannot be freed by force and can only be freed after the process holding the resource is terminated. Therefore, the process to which the resource is allocated is the only one able to free the resource for other processes.

Circular wait: In the case when a set of processes, $P_0, P_1, \ldots P_n$, are in waiting status, P_0 requests the resource that is occupied by P_1, P_1 requests the resource occupied by P_2, P_{n-1} requests the resource held by P_n, and P_n requests the resource occupied by P_0.

Since deadlock occurs when all the above four conditions are met, the deadlock condition can be prevented by not allowing at least one condition among the four conditions. Accordingly, as a practical method to prevent the deadlock, the first thing

is to ensure that all necessary resources are available before the start of a process. The second is that the process frees all resources and waits if the requested resource cannot be promptly allocated when the process occupies the same resources and requests another resource. The third is that a linear sequence number is assigned to all resources and each process can only request resources having sequence numbers in ascending order. Therefore, a process that has to use multiple resources simultaneously asks for a high-priority resource and, thereafter, a low-priority resource.

9.8 Inter-process Communication

A communication mechanism is necessary for each process to access particular data during parallel execution or to send data to another process. The communication mechanism should not have an influence on the transmitted data. Data and communication protocols have to be defined in each process and have to be independent of the specific communication method. In a broad sense, the synchronization problem mentioned in earlier sections can be defined as a problem of inter-process communication.

As methods to realize the inter-process communication, shared memory and message passing can be used. These complement each other and they can be simultaneously used in one OS.

9.8.1 Shared Memory

For inter-process communication via shared memory, global variables where processes can read and write can be considered. However, because usage only of global variables may cause data clashes when more than one process accesses the global variables simultaneously, it is essential to use critical sections with this method. A critical section can be realized by using a synchronization mechanism such as a semaphore. This method is simple and fast. However, when a high-priority task pre-empts global data from a low-priority task, the global data can be distorted. In order to prevent this problem a data buffer is used. The buffer between the task that generates data and the task that uses data is called the “damper”. In this case, the buffer can be specified by various data structures such as a stack and unstructured data.

The shared memory has to be located at an area whose address in the memory map is known. This is not difficult for assembly languages. However, in the case of high-level languages that cannot access memory directly, additional techniques are required for implementing this.

9.8.2 Message System

A message system is a method that enables process synchronization and data exchange without shared variables. Basically, the communication function between processes provides two operations; the first is SEND for sending a message and the other is RECEIVE for receiving a message. A method is needed to refer to each other for communication between processes. There are two methods for this.

Direct Communication: The process that wants to send or receive a message specifies the name of the receiver or sender. For this, it is necessary to know the name of the corresponding process.

SEND (P, message): send message to P.
RECEIVE(Q, message): receive message from Q.

Indirect Communication: the message is transmitted via a mail box. The mail box has a unique identification number and when two processes share the same mail box communication is possible.

SEND (A, message): Send message to mail box A.
RECEIVE(A, message): Receive message from mail box A.

Sending a message to a mail box is a simple task. The message is copied into the specified mail box and then the first in-message is copied into the receiver's message data structure when the receive function is called. After reading the message, it is deleted from the mail box.

A mail box does not have an individual structure and is located in memory or on disk. It exists when the system is on and activated. If a mail box exists on disk, it is regarded as a temporary file and is deleted when the system is turned off. A mail box does not have a unique identifier or name. When it is created, it is distinguished by a logical identifier. All processes that use a mail box use the logical identifier to distinguish it.

Figure 9.10 shows an example of a message-passing program. If proc1 and proc2, the processes created by the main program, receive a message that is not zero, they write out "A" and "B" respectively and send a message that is not zero. As a result, since synchronization of the two processes is realized, texts "A" and "B" are written out successively.

In another method, which is similar to using a mail box, a queue is used. A queue can store more than one message and can be implemented as an array of mail boxes. As a practical implementation method, a ring buffer is used in order to receive service requests for a device and queues at the head and tail of the ring buffer are used in order to control access to the ring buffer.

As other communication methods, the rendezvous method and monitor method are used. The rendezvous method is a method for synchronization and communication between tasks in the ADA programming language. One task requests a rendezvous and the other task declares that it is ready to receive. The task that requests the rendezvous has to know the name of the task called. However, it is not necessary

```
/* Display 'A' and 'B' one by one utilizing message passing method */

#include<conf.h>
#include<kernel.h>

void main()
{
    int     proc1(), proc2();
    printf("\n Display A and B one by one by utilizing message passing method \n");
    printf("    Output....\n\n");
    pid1 = create(proc1, INITSTK, INITPRIO, "proc1", 0, 0); /* create process 1 */
    resume(pid1); /* make process 1 READY */
    pid2 = create(proc2, INITSTK, INITPRIO, "proc2", 0, 0);
    resume(pid2);
}

proc1()
{
    int i;
    int msg1, msg2;

    for (i = 0; i < 10; i++) {
            /* receive message, if message is not received, process1 turns to wait state */
            msg1 = receive();
            if (msg1 != 0) printf("A");
            msg2 = 1; /* assign nonzero value for sending to process2 */
            send(pid2, msg2);
    }
}

proc2()
{
    int i;
    int msg2 = 0, msg1 = 5;
    for (i = 0; i < 10; i++) {
            msg2 = receive(); /* receive message from process1 */
            if (msg2 != 0) printf("B");
            send(pid1, msg1);
    }
}
```

Fig. 9.10 Programming example utilizing a message for inter-task communication

for the task called to know the name of the caller. The concept of the rendezvous method is the same as that for a subroutine call.

The classical method for resource protection and communication between tasks is the monitor. The monitor consists of a reserved data region (monitored region) and a process that has the exclusive right to manage the reserved data region. Other processes do not have direct access to the monitored region and have to call the monitor process. The monitor provides the service to only one process at a time. Using this mechanism, the monitor can guarantee that the execution of a process completes before another process has access to the same data region.

9.9 Key Performance Indices

A real-time OS supports pre-emption multi-tasking. The multi-tasking ability enables effective resource management as well as parallel execution of processes or tasks. For efficient multi-tasking, it is essential to increase the response characteristics of the OS by reducing the context switching time. Furthermore, a real-time OS can predict the required time for running tasks in order to realize the real-time scheduling and synchronization mechanism. In addition, it is necessary to know the characteristics of the interrupter, such as interrupt latency, which is the time spent to resume a task after an interrupt has been fired, the maximum elapsed time of the system call, etc.

Knowledge of the above performance indices makes it possible to predict the user application's execution and makes it easy to design the process schedule.

In the following sections, the key performance indices of a real-time OS will be addressed. In general, the terminology and definition of these indices are slightly different depending on the kind of OS. In this book, the performance indices that are typically used will be described.

9.9.1 Task Switching Time

Task switching time means the average time spent to switch between two tasks with the same priority. As shown in Fig. 9.11a, it is supposed that all tasks have the same priority. Task switching is done when real-time software uses the time-sharing algorithm to carry out tasks with the same priority. Task switching time is used for storing and restoring context. And it depends on the efficiency of control data structure, processor architecture, and instruction set.

9.9.2 Context Switching Time

The context switching time denotes the time spent to start task B when task A with low priority is being executed, as shown in Fig. 9.11b. For the context switching time, the context of the pre-empted task is stored, the context of the new task is loaded, and the new task is scheduled. Note that the task switching time denotes the time spent to switch tasks with the same priority, and that the context switching time is different from the task switching time.

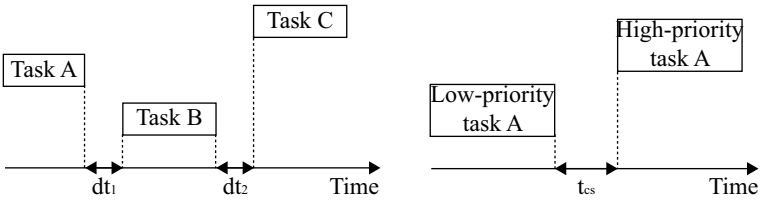


Fig. 9.11 Definition of Task switching time (a) and context switching time (b)

9.9.3 Semaphore Shuffling Time

The semaphore shuffling time denotes the time delay from when some task frees the semaphore to when the task waiting for the semaphore is activated. Because the semaphore shuffling time itself represents a computing burden related with the mutual exclusion, the semaphore shuffling time is one of the key performance indices of real-time systems.

Figure 9.12 shows the mechanism to pass a semaphore when more than one task is competing for the same resource. Task A is being executed and requests the semaphore corresponding to the resource in order to access the resource at t1. At t2, Task A is stopped and Task B starts. At t3, Task B requests the semaphore to access the resource that is occupied by Task A. Because Task B cannot be continued, Task B is stopped and Task A is activated again. At the end of execution, t4, Task A frees the semaphore. As soon as the semaphore is freed, the scheduler resumes Task B and Task B receives the semaphore. Mutual exclusivity based on a semaphore is the most effective method of allowing only one task to access a particular resource.

9.9.4 Task Dispatch Latency Time

The task dispatch latency time is a frequently used performance index for evaluating real-time systems. In a real-time system, a real-time task waits for a particular event

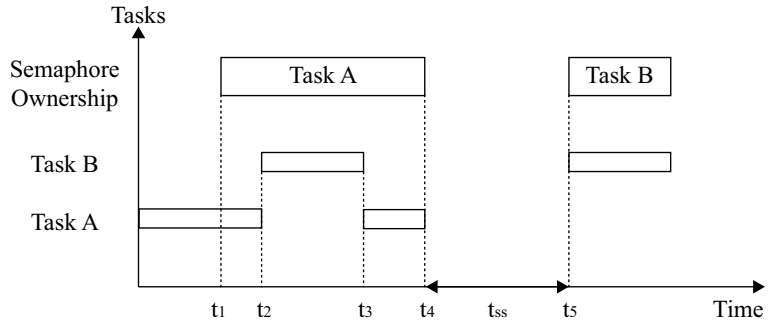


Fig. 9.12 Semaphore passing mechanism

to happen. When the interrupt occurs, the task being executed with a low priority should be stopped quickly and the real-time task activated. The task dispatch latency time denotes the time spent to start a task from the interrupt service request. The task dispatch latency time is highly related to the interrupt latency time and the context switching time.

Figure 9.13 shows the interrupt mechanism to activate an interrupt service routine (ISR) from the point of interrupt and to come back to the task status before the interrupt. The interrupt response time consists of the delay time related to the hardware and software. The hardware latency (delay time) is defined as the time span from detection of an interrupt to acknowledgement of it by a processor. If a processor receives the interrupt signal, the OS has to wait for completion of the instruction currently being executed. This delay time is defined as an interrupt overhead (IO). After an IO, the system needs the time for interrupt latency, which an interrupt dispatcher manages, for the whole interrupt to be worked out.

In conclusion, during the interrupt response time, which is composed of the above three steps, the system is ready to execute an interrupt service routine by acknowledging an interrupt to a processor and storing the parameters or context of the system. After the interrupt response time, the interrupt service routine (ISR) is invoked to handle the requirement of the interrupt. When the ISR has completed its work, a scheduling latency is spent for the OS to reschedule and switch the context to the task before the interrupt. Therefore, the scheduling latency time is defined as the time span from completion of the ISR to activation of the first instruction of a scheduled task.

Figure 9.14 shows the worst case example of the task dispatch latency time executing in LynxOS (one of the widely used RTOS). It includes several behaviors and delay times consumed, such as issuing an interrupt, activating multiple ISRs, and resuming the task after the interrupt.

Table 9.1 shows the comparison of the performance index of three kinds of a real-time OS; Hyperkernel 4.3 (Imagination), INTime 1.2 (Radisys), and RTX4.1 (VentureCom). They were tested on a PC with a Pentium 200MHz CPU. (Note:

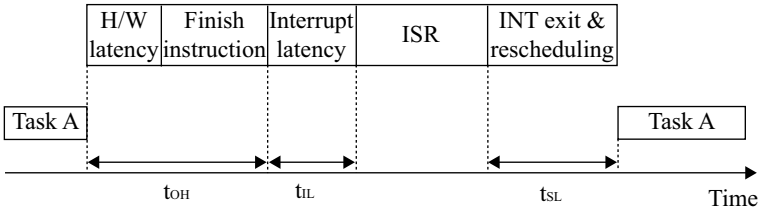


Fig. 9.13 Definition of task dispatch latency time

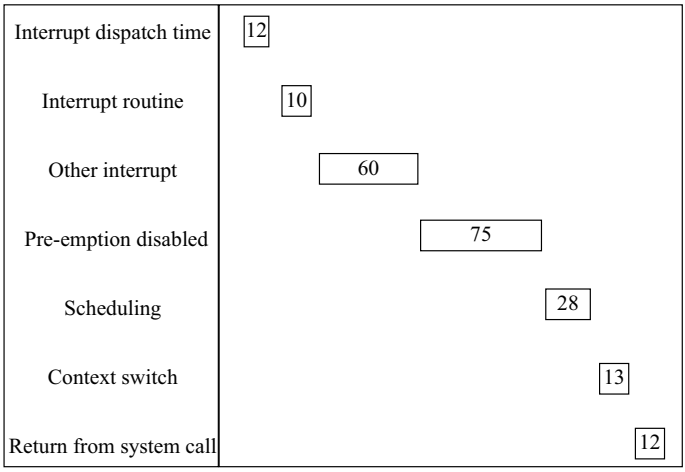


Fig. 9.14 Example of a task dispatch latency time

Because this table only intends to show examples of the performance index, the actual system is not identified.)

The performance index shown in the above table is described by an average value and a maximum value obtained from thousands of tests. When we select the right real-time OS, the maximum value, meaning worst case, should be considered rather than the average value. In order to design the reliable and deterministic system that is required to meet the hard real-time property, the OS with lower maximum value is preferred. From this point of view, system B is a good operating system. Of course, it is not always to be concluded that the OS with the lowest performance index is the best. According to the characteristics of the system, various performance indices should be considered.

According to the above-mentioned standards and the requirements for a real-time OS, Windows NT (Microsoft), widely used as a PC OS, is adequate as a general-purpose OS but is not adequate for real-time OS. Firstly, Windows NT is able to support multi-threading but is not suitable for real-time scheduling because it does not provide enough priorities and it is impossible to define clearly a tiny time slice. For example, Windows NT provides a good hardware interface. However, since Pentium power management interrupts the system for an unpredictable time, time analysis of

an application is very difficult and development of a reliable system is impossible. Also, it is impossible to use a small real-time clock pulse because Windows NT does not provide a tiny programmable timer. Besides, Windows CE 2.0, which is widely used as an embedded system OS, is not suitable for medium-sized or large-sized systems. It is only useful for small systems having a long performance index latency, and few allocable priorities.

Table 9.1 Comparison of performance indices of three RTOS (times: μs)

Performance Index		System A	System B	System C
Task Switching Latency	Avg.	5.47	4.68	2.64
	Max.	23.13	10.68	5.73
ISR Latency	Avg.	11.26	8.78	7.66
	Max.	19.23	14.52	25.68
Scheduling Latency	Avg.	25.95	4.73	6.36
	Max.	39.0	10.14	32.25

In addition to the above-mentioned operating systems, various real-time OSs, such as CHORUS/OS, IRIX, LynxOS, OS-9, p-SOS, QNX, RT-mach, SORIX 386/486, VRTX, and VxWorks have been used. However, the source code of the application developed for one particular OS cannot be reused on a different OS, since each OS has its own code format, such as API and system calls. To overcome this compatibility problem, POSIX 1003.4, which is based on POXIS, was established, but it can be expected that a lot of time will be needed for perfect standardization.

9.10 Hardware and Operating Systems

In the case of the design of real-time systems, one of the key considerations is how to integrate the hardware components for equally distributing task load and how to operate the real-time OS to manage hardware resources. Therefore, it is important to define a multi-processing hardware architecture allowing combination of multiple processors for handling particular tasks, input/output processors and the operating system of a multi-processing system.

9.10.1 Architecture of Multi-processing Hardware

As the hardware architecture for multi-processing systems, a bus structure has been widely used. Bus structures are classified into two types; one is the common bus type, where one bus is shared by multiple processors, and the other is the standard bus type, where a computer unit with a heterogeneous local bus is connected to a standard bus (*e.g.*, Multi-bus and VME bus).

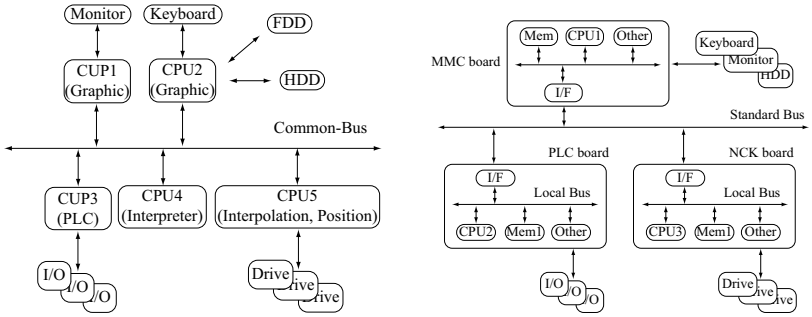


Fig. 9.15 Architecture of multi-processing hardware having a bus structure

For the sake of explanation, let us suppose that each bus type is applied to the CNC system. The common bus type shown in Fig. 9.15a is a structure where multiple processors, carrying out tasks such as PLC, MMI, interpreter, and interpolator, are connected via one bus, using the common memory and having their own individual input/output interfaces. It can be called a "closed" architecture, where system scaling is very difficult.

The architecture shown in Fig. 9.15b implements several processor units executing NCK, PLC, and MMC and all units are connected via a standard bus. Accordingly, expansion of such a system is easily possible only by adding process units for particular tasks to the standard bus. In this architecture, each processor unit communicates with the other units via a common memory module.

As mentioned above, in the bus-type architecture only one communications channel is provided to processors, memory modules, and input/output devices. In this simple architecture, the common bus works basically as a passive device and communication between devices is controlled by the devices own bus interface. First, the processor that wants to transmit data or input/output to the processor has to check whether the bus and the counter device can be used. It must then transmit the data after informing the device how to handle the transmitted data. The device that receives the data has to know that the message from the bus is its own. It must also be able to recognize the message and perform particular actions according to the message.

However, the bus-type architecture has serious drawbacks due to having only one communications channel. If the bus does not work the entire system does not work. Also, the communication bandwidth of a system is restricted by the bus bandwidth. In addition, as the system becomes busy, the competition for the bus grows and the efficiency of the bus decreases drastically. Therefore, bus-type architecture is economical, simple, and flexible, but the application of this architecture is restricted to small-sized multi-processing systems due to the limitations of the bus.

As hardware architecture for multi-processing, a distributed architecture can be considered instead of the bus-type architecture. The distributed architecture is regarded as a loosely coupled type of architecture where more than one individual distributed computer system is integrated by a communication line, as shown in Fig. 9.16. Each system has its own operating system and memory and is operated

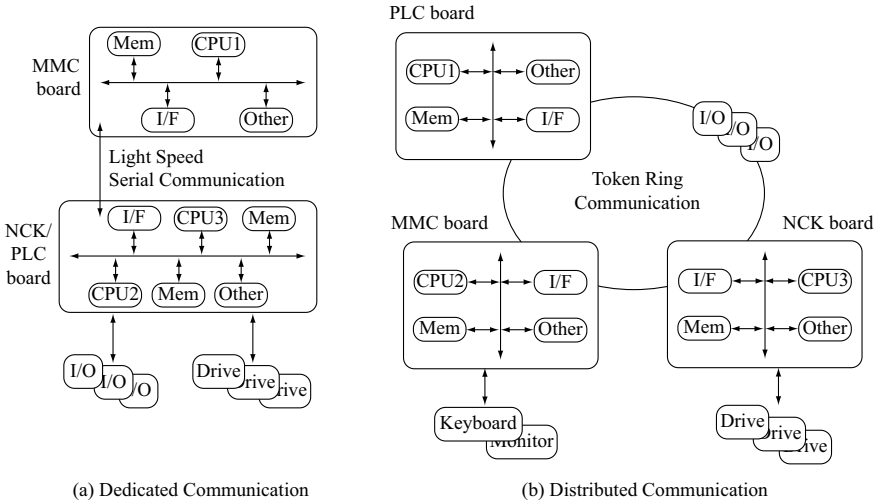


Fig. 9.16 Architecture of multi-processing hardware having a distributed structure

individually. Only if necessary, does each system communicate with another system. The distributed system can access files located in another system via a communication line and a busy system can pass tasks to another system that is less loaded. In this architecture, since the communication speed has a large influence on the performance of the entire system, the design should allow that tasks that need high-speed communication are always executed on the same processor taking into consideration the balanced distribution of tasks among processors.

As shown in Fig. 9.16a, NCK and PLC which need high-speed communications are executed on the same unit. In contrast, the MMI unit, whose communication data is relatively small, communicates with the NCK/PLC unit via a high-speed serial communication line. Figure 9.16b shows another distributed architecture where the NCK, PLC, and MMI tasks are executed on each unit and combined using a ring-type high-speed communication line.

In recent times, the sustained development of microprocessors makes it possible for only one processor to carry out many tasks that could only be performed by multiple processors in the past. Accordingly, the old multi-processing system architecture was replaced by an architecture where one processor performs multiple tasks, or threads, which denote the processes that were executed individually by multiple processors. As a system with one processor is a system that maximizes usage of the processor resource, this has an advantage in terms of cost. However, it has the disadvantage that some trouble results in the malfunction of the whole system because one processor performs all the tasks. Also, because all the tasks of the CNC system are performed by one processor and memory, it is necessary to use a highly reliable, hard real-time OS that can guarantee regular execution of tasks within the allowable time and manage perfectly shared resources.

9.10.2 Operating System Configuration

Not only integration of hardware for multi-processing but also configuration of operating systems must be considered. The configuration method for allocating and managing resources, protecting resources, preventing deadlocks, terminating abnormal execution, balancing input/output load, and balancing process load should be defined together with the configuration method of the hardware. As configuration methods for multi-processing operating systems, there are the master/slave method, separate executive method, and symmetrical method.

In the master/slave architecture, the main processor only executes the OS and slave processors perform user applications, as shown in Fig. 9.17a. In this method, when it is necessary for the OS to handle a process that was executed by a slave processor, the OS generates an interrupt and waits for the processor's interrupt handling. Depending on the number of slave processors and how often a slave generates an interrupt, the size of the waiting queue may be different. Although the slave processor has no task to run, it must wait for the master processor's interrupt. If the slave processor performs only short and simple tasks, the master processor will have a large burden. If the main processor cannot respond to the requests of slave processors quickly, the capability of the slave processor is wasted. As a main processor plays the role of general-purpose processor, it performs not only arithmetic operations but also input/output operations while the slave processor only performs arithmetic operations. Therefore, slave processors can effectively perform arithmetic operations but because input/output operations are done only through the master, a slave processor cannot perform these.

In terms of reliability, if one slave processor does not work, the computing power is decreased by some amount but the whole system continues to work. However, if the main processor fails, the system can do no input nor output. Therefore, the main drawback of the master/slave architecture is that processors are not equal and only the main processor manages input/output operations. In conclusion, since malfunction of the main processor makes execution of the system impossible, the reliability of this architecture is low but it is easy to implement this architecture. Therefore, this architecture is suitable for systems where the computing burden is well known and the main processor can manage task scheduling accurately. It is appropriate for asymmetric systems where the performance of the main processor is superior to that of the slave processor.

As shown in Fig. 9.17b, in the case of separate executive architecture, each processor has an individual OS and interrupts from each processor are handled by the processor responsible. The data about the whole system is stored in a table and access to the table should be controlled based on the mutual-exclusivity mechanism. A task that is allocated to a particular processor is executed on that processor until the task is finished.

Because this architecture is more reliable than the master/slave architecture, the malfunction of one processor does not result in a halt of the entire system. However, it is not easy to restart the malfunctioning processor and synchronize it with the whole

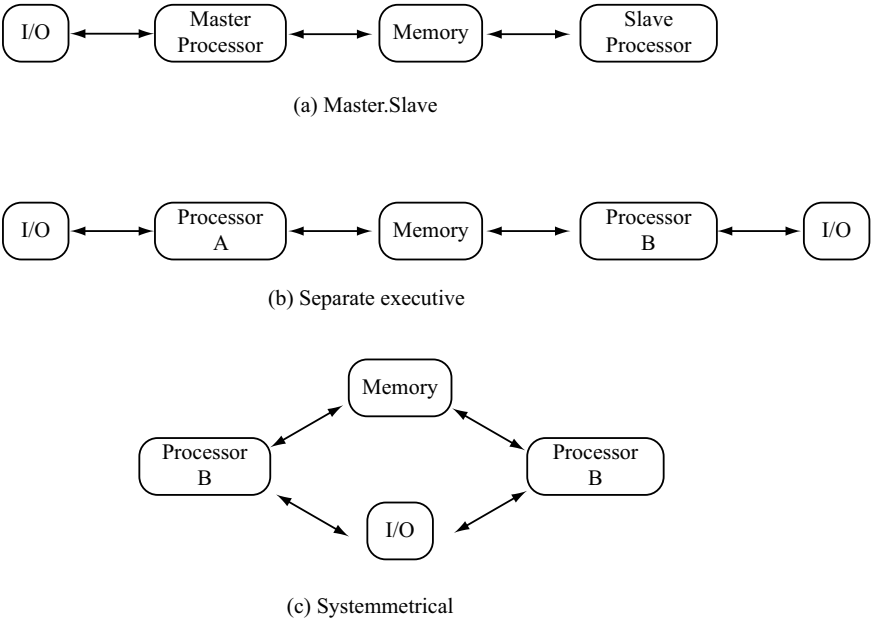


Fig. 9.17 Architecture of multi-processing software

system. Since it is not necessary for processors to help each other for execution of processes, other processors can be idle even when another processor is busy.

Although the symmetrical architecture shown in Fig. 9.17c is the most complex structure, all processors are at the same level. One operating system can utilize all memory and I/O devices of all processors and can distribute the workload more effectively. This type is the most reliable. A task working on one unit can be transferred to another unit without modification, and all processors of each unit can cooperate to execute a special task. Further, it can use the resources effectively because a reasonable load can be assigned according to the workload of the processors.

9.10.3 CNC System Architecture

As mentioned in the previous section, the configuration of multi-processing systems varies and there is a variety of advantages and disadvantages according to the architecture. Some commercial CNC systems are configured based on the above-mentioned architectures, while other systems use modifications of those architectures or combinations of them. Figure 9.18 shows one example of the architectures of commercial CNC systems using a standard bus.

The architecture shown in Fig. 9.18 is a typical platform that was developed for CNC systems by universities and research centers. The architecture in Fig. 9.18 is

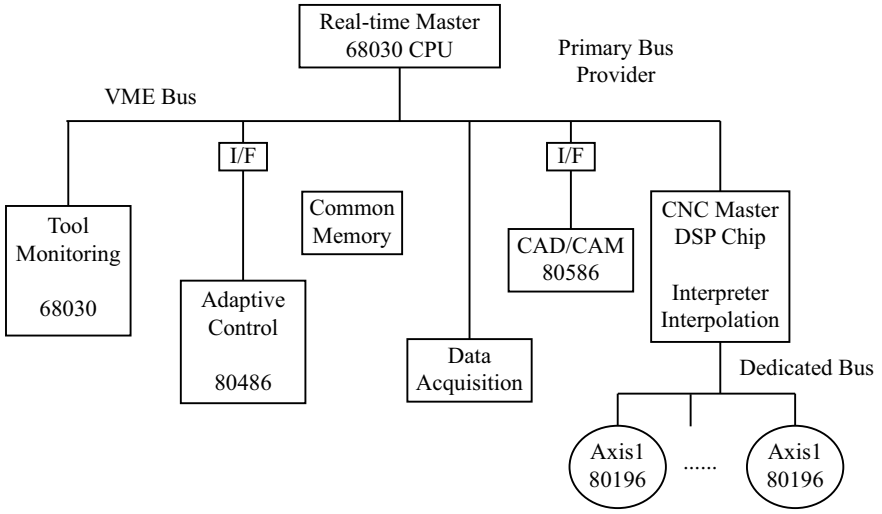


Fig. 9.18 Example of CNC systems having a standard bus

based on the common bus, being the standard VME bus, and is a master/slave type that consists of a main processor, controlling the whole system, and slave processors and it is possible to add various kinds of slave processor. In particular, a real-time master that manages the system entirely is connected with slave processors via a VME bus. As slave processors, there are the DSP units for executing interpretation and interpolation, the tool monitoring unit for detecting tool status, the adaptive control unit for performing adaptive algorithms, the data acquisition unit for obtaining sensor data, and a common memory unit. It was designed so that, if necessary, a CAD/CAM processor unit for generating toolpaths can be added.

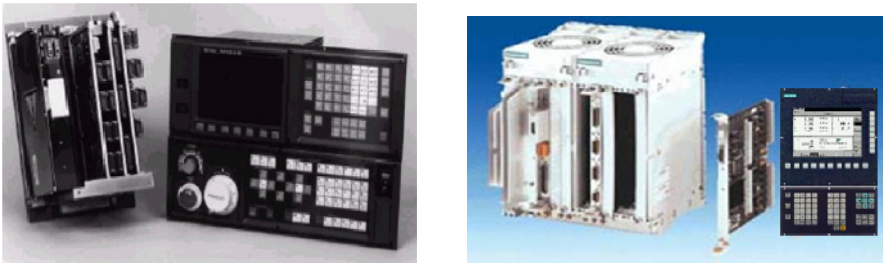


Fig. 9.19 Commercial CNC system having a bus system

Figure 9.19 shows CNC systems using a bus system; the FANUC 0 series and Siemens 840C control units. The FANUC 0 series is famous for CNC systems, with simple functions and was developed based on the common bus. The processors that are dedicated to PLC, NCK, and MMI are individually connected via FANUC's own

bus and are managed by only one OS. However, this architecture is of closed type because it does not allow the user to add new functions and hardware. Therefore, only the functions implemented by the CNC maker are usable. It also makes it impossible to implement machine tools with advanced functions. To overcome this problem, recently open CNC system architectures have been developed in terms of hardware and software architecture.

The Siemens 840C has an architecture where process boards for NCK, MMI, and PLC are connected via Siemens' own bus in a backplane. This is similar to the architecture of the system based on the VME bus. The 840C is a PC system where NCK and PLC boards use Siemens' own bus as a local bus and the MMI board uses an ISA bus. In this way, it is easy to add software that operates on a PC but it is impossible to add new functions to the NCK and PLC. As an operating system for the 840C, both DOS, a non real-time OS, and FlexOS, a real-time OS, are used.

Fig. 9.20 Commercial CNC system having a loosely coupled system

In the case of the Siemens 840D, the processors dedicated to PLC and NCK are connected via a local bus and carry out real-time control based on a unique OS. The processor that is dedicated to MMI uses an operating system, Windows 3.1, being a non-real-time OS. The NCK and PLC boards are connected through high-speed communication. The FANUC 150i is similar to the 840D but the main difference between them is the communication method. In the case of the 840D, all hardware components are connected via a ring-type communication line, as shown in Fig. 9.16b. In the case of the FANUC 150i, the NCK/PLC board and the MMI board are connected by high-speed serial communication. Since a non-standard communication method is applied to them, they cannot be connected with third-party systems.

9.11 Summary

A CNC system is a real-time system where NCK, MMI, and PLC should be executed within a specified time. In order to design a CNC system that guarantees hard real-

Fig. 9.21 Coupled commercial CNC systems

time property, a harmonized relationship between the functionality of real-time OS and the architecture of hardware and software has to be considered.

The kernel, being the core component of a real-time OS, provides the process manager for priority management and context switching, memory manager, process coordinator for controlling processes, a synchronization mechanism for avoiding clashes between processes, and communication management functions for controlling communication between processes. Accordingly, the system designer has to implement a method for operating the real-time system using the process management, scheduling functions, system resource protection mechanisms, synchronization mechanisms between processes and communication mechanisms provided by system calls or the API of a real-time OS.

The performance index of a real-time OS can be defined in various ways. However, the task switching time between two tasks having the same priority and the context switching time spent to switch between the execution of two tasks are regarded as key performance indices of a real-time OS. The semaphore shuffling time, being the time delay from when some task frees the semaphore to when the task waiting for the semaphore is activated and the task dispatch latency time, being the time spent to start a task from the interrupt service request, are also regarded as key performance measures. Though, as the delayed times of various performance indices decrease with advancements in micro processor technology, the importance of taking the performance indices into consideration has decreased.

In terms of the design of a real-time system, a necessary consideration is how the hardware elements are configured, how they are connected, and how OSs are configured. As the configuration structure of hardware, there are two types; one is based on the bus system such as common bus and standard bus. The other is based on the communication interface such as serial communication and ring network. Moreover, as the configuration of OSs, there are master/slave configuration, separate executive configuration, and symmetric configuration.

In conclusion, in order to implement a real-time system, an OS that can provide multi-tasking, synchronization mechanism, priority-based scheduling, and preemption function should be selected. Also, according to the type, the execution time, and the unique characteristic of a task, the architecture of the software should be designed. In addition to designing the architecture of the software, the architecture of hardware should be designed in parallel.

Chapter 10
Design of PC-NC and Open CNC

Recently, industrial controllers based on PC hardware seem to have replaced conventional controllers based on a closed hardware structure. We will discuss the design issue of PC-NC running the NC functions described in other chapters on a personal computer (PC). The hardware architecture, software model, and communication mechanism for building PC-NC will be addressed. In particular, Soft-NC, where a single processor is used and all functions of PC-NC are implemented as software tasks, will be described. In addition, an open CNC architecture supporting openness of H/W and S/W of CNC systems will be discussed.

10.1 Introduction

The CNC system in the 1970s and 1980s was a multi-processor system with several 8-bit CPUs or 16-bit CPUs and an individual processor and memory were assigned for each function. Therefore, the former CNC systems were closed systems, in which the design of the CNC system depends on the CNC maker's hardware and firmware. In the late 1980s, PCs built on Intel 80386 and 80486 processors having high computing power and based on 32-bit CPUs were introduced and so, naturally, PC-NC was developed using a PC system as base hardware for the CNC system.

Unlike the closed CNC system, where NC functions depend on the hardware, PC-NC makes it possible to implement CNC functions in software modules by supporting a real-time operating system (RTOS).

The architecture of PC-NC can be divided into three kinds:

1. Embedded motion controller, which carries out the NCK/PLC function with its own processor, is attached to the extended slot of PC. The MMI is operated on the PC.
2. Two PCs are used and are connected via high-speed communication. One PC is used for MMI and the other is used for NCK/PLC.

3. One PC with single CPU executes MMI, NCK and PLC in a multi-threading environment with real-time OS.

Type 1 and Type 2 use more than one CPU and use a PC for the user interface (MMI). Type 1 uses an embedded board for executing NCK/PLC functions while Type 2 uses the PC hardware. Therefore, configurations using more than one CPU can be classified into two types; one is based on the standard PC bus, such as ISA, EISA, and PCI, while the other is based on high-speed communication such as Ethernet and high-speed serial communication. In the case of Type 3, where a single CPU is used, the hardware components for communication are removed and the MMI function is carried out as one software task on a single CPU. In Type 3 architecture, the NCK, PLC, and MMI functions are regarded as individual tasks and are performed on a single CPU by a real-time scheduler. Therefore, this configuration makes it possible to reduce the size and cost of the CNC system.

The following describes type 3 in more detail. The CPU of PCs has advanced from 16 bits to 64 bits. The computing power of the Intel Pentium CPU has more than doubled compared to the 32-bit CPU. Furthermore, it is possible to provide a Graphic User Interface using Windows OS. The advancement of CPUs has made it possible to carry out MMI and user applications after performing NCK. According to some experiments, in 8-axis control systems based on a Pentium 133 CPU, 40% of the computing power is used for executing all tasks except for the MMI task. Therefore, the conclusion is that 60% of the computing power can be used for MMI and various user applications. For example, in a PC-NC using a single CPU, advanced functions such as remote monitoring or diagnosis can be realized via the network established between controllers in a shopfloor.

In conclusion, with modern CPUs it is possible to execute all tasks that were previously performed on two CPUs using one modern CPU. Type 3 is also called Soft-NC, and the functions of NC and PLC are designed as functional modules or software tasks in the multi-processing environment of a real-time OS. Soft-NC has the aim of realizing the PC-NC architecture using software. Therefore, the NC and PLC functions in Soft-NC are executed together in cooperation with user applications via the various internal services of the OS. This type of system can easily be advanced to become an open system where the user can add user-specific functions and modify existing functions.

For various configuration types, operating systems are applied differently. Since Type 1 and Type 2 use more than one CPU, it is possible to interchange data between them via the bus interface or asynchronous high-speed communication even when different operating systems are used. Therefore, in general, a PC operating system is used for the MMI system and a real-time OS is used for the NCK/PLC system which requires the real-time property. Because of the separated OS environment, the above-mentioned systems are easy to design from the point of view of the system programmer.

Two methods to implement the OS can be considered for a Type 3 system.

The first idea is that two kinds of operating system run on a single PC. Accordingly, in order to perform MMI operations on a non-real-time OS (*e.g.*, DOS and

Windows OS) and NCK/PLC operations on a real-time OS on a single CPU, a real-time OS that regards real-time tasks as high-priority tasks and non-real-time tasks as low-priority tasks is required. As examples of this type of OS, there are FlexOS, operating together with DOS, iRMX, operating with Windows 3.1, and InTime, operating with Windows NT.

Figure 10.1 shows Soft-NC based on iRMX. The NCK/PLC functions are assigned as tasks having high priorities and are performed preferentially by high-priority scheduling. MMI functions executed on Windows 3.1 are assigned as tasks having low priorities and are carried out by VM86 mode, iRMX's virtual mode, after all tasks relating to motion control are finished. InTime is used with the same concept as with iRMX, but Windows NT is used as the basic OS.

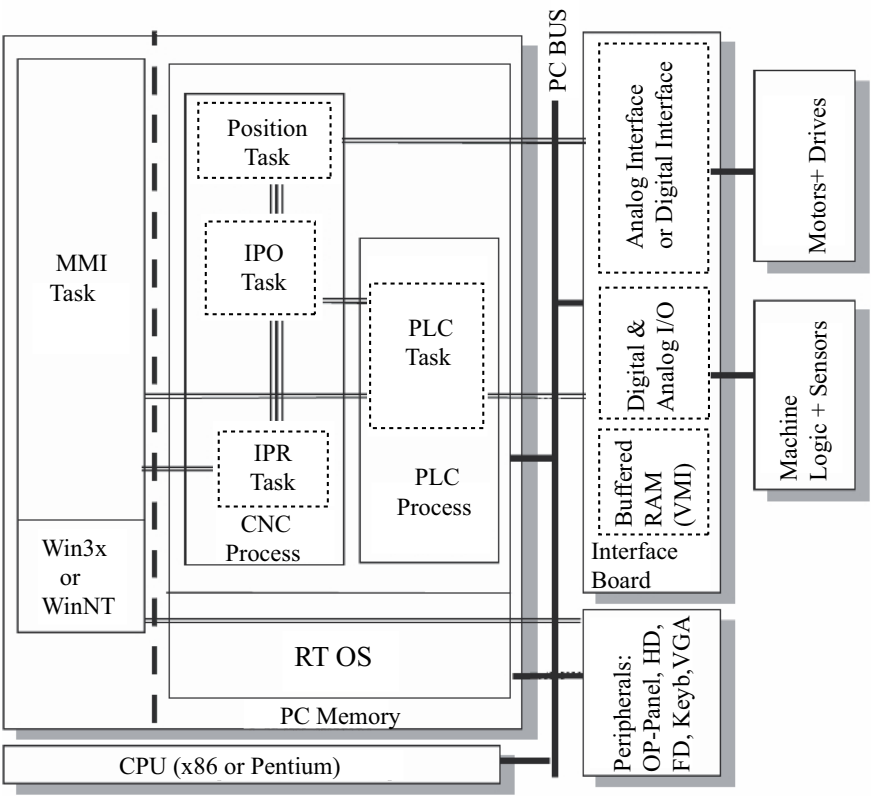


Fig. 10.1 Example of Soft-NC architecture using both RTOS and Windows

The second method for implementing the OS is to utilize only one operating system capable of real-time operation. Windows NT was announced as an OS having real-time capabilities. It is possible to set the minimum time slice to 1 ms (default

10 ms). However, it cannot be used for a hard real-time system that needs a high resolution timer capable of several tens or 100 μs setting.

RTX (Real-Time eXtension) developed by Venture Com is one solution for resolving the hard real-time requirements of the Windows NT system. This package provides a variety of RTX call functions similar to the system calls of Windows NT for the system programmer's convenience. RTX is a kind of supplementary code to access the HAL (Hardware Abstract Layer) of Windows NT. Nowadays, Windows NT-RT extension is preferred as the development environment for the Soft-NC system.

The reason that Windows NT is preferred by Soft-NC developers is summarized as follows:

1. Operates all application programs developed under previous operating systems, such as DOS, Windows 3.X, and Windows 95.
2. Supports three-dimensional graphic tools such as OpenGL, and
3. Provides network capabilities as a basic function.

Recently, the Linux system for PC has also been used as a OS for the Soft-NC. Because the Linux used is an OS of Type 3 it is not suitable for a hard real time system, it implements RT-Linux in a similar manner to the RTX of Windows NT.

10.2 Design of Software Architecture

As previously mentioned, the software design of a CNC system must be done together with the design of the hardware and the operating system. As the design of the software architecture belongs to the stage of developing MMC, NCK, PLC, and the system kernel, it is difficult to apply formal methodologies and regular rules and depends on the system designer's know-how. Therefore, in this book, the fundamental design of software architecture for CNC system modeling, kernel design, and communication among modules will be described in the case of Soft-NC design.

10.2.1 CNC System Modeling

To design the software architecture, system modeling of the processes and tasks to be performed on CNC system is needed.

The software modules of the CNC system consist of the OS and system kernel based on PC hardware and three key application modules, *i.e.* NCK, MMC, and PLC. Figure 10.2 shows an example of a software model with three application layers and the tasks belonging to each application layer. For each task, the classification of the application layer is done from a functional point of view. However, this is just an example, and from a different point of view they could be classified differently.

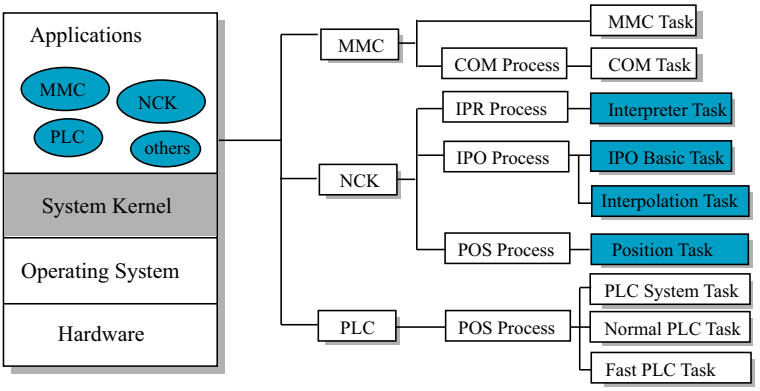


Fig. 10.2 Software model and application tasks

If we classify the variety of tasks in terms of function in order to make the model, they can be divided into two types; one is the non-cyclic task and the other is the cyclic task. A non-cyclic task means a task in which the real-time property is not required and the MMC, interpreter and communication function are typical examples of the non-cyclic task. A cyclic task is a task for which the real-time property is required and the interpolator and position control tasks are typical examples of cyclic tasks. Details of these will be given in the following sections.

10.2.1.1 Non-cyclic Task

The interpreter, intermediate handler and external communication tasks belong to the non-cyclic task category.

The interpreter reads ASCII blocks from a part program and interprets them. Then, for the next task, it saves the interpreted data in internal memory. The interpreter steps performed are as follows:

1. Read ASCII block and check the schema and grammar.
2. Transform the G-code block read into an internal data structure.
3. Save modal or one-shot data in an internal data structure.
4. Perform operations related with variables in part program.
5. Perform control flow operations such as jumps and subroutine calls in the part program.
6. Interpret macros in the part program.
7. Synchronize with the interpolator.

The intermediate handler generates data concerning tools, spindle, and coordination data based on the information in the data from the interpreter and stores it in an internal buffer. The details of the intermediate handler are as follows:

1. Set spindle function.
2. Transform program coordination system into the local coordinate system specified in the CNC system.
3. Do tool compensation.
4. Check speed limits and prohibited machining areas.
5. Get the data ready for interpolation.
6. Perform look-ahead functions

The external communication task is to provide an interface between NCK and external components and provides the following functions:

1. Send a part program to NCK.
2. Read and write the parameters and user-specified variables such as tool compensation to and from NCK.

10.2.1.2 Cyclic Task – Low Priority

The interpolator belongs to the category of cyclic tasks with low priority. The interpolator reads the interpreted data from the internal buffer and then interpolates the movement of all axes. The interpolated data is then sent to the position control loop. The interpolator is executed as follows:

1. Read the data from the internal buffer.
2. Read the actual position from the position controller.
3. Interpolate spindle rotation in position control mode
4. Interpolate axis movement along the path.
5. Transform the coordinate system and check prohibited machining areas.
6. Send interpolated data to the position controller and check software limits of axes.

10.2.1.3 Cyclic Task – High Priority

The position control task belongs to the category of cyclic tasks with high priority. In the position control task, interpolated data is transformed to motor rotation speed and, if necessary, the torque of the drive is computed. The limits of the control loop and drive are checked.

1. Actual position is fed back from a drive.
2. Perform position control algorithm and compute commands for drives.
3. Communicate with the interpolator.
4. Prepare the instruction at the next position control sampling time.

In conclusion, the key to the design of a Soft-NC is:

1. to realize the system kernel programming to make the tasks execute perfectly in the kernel, and

2. to develop each task in software form as modeled, under the determined hardware architecture and operating system.

10.3 Design of Soft-NC System

10.3.1 Design of Task Module

The task modules of the CNC system can be designed in unlimited combinations according to the specification of the CNC and the designer's concept. Typically, however, the functions of commercial systems can be classified as shown in Fig. 10.3 and the major function of each module is as shown in Table 10.1.

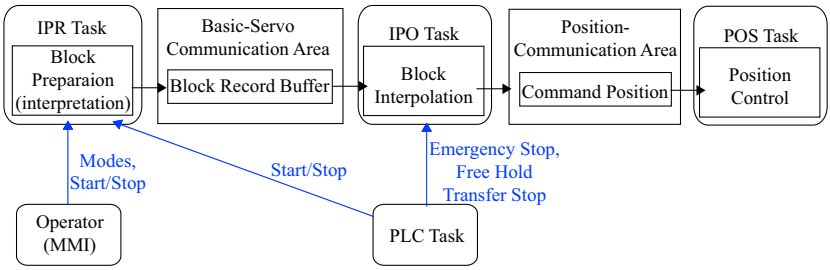


Fig. 10.3 Design example of the task modules

After the IPR task reads a part program, including the user's instructions, it interprets the ASCII blocks and transforms them into an internal data structure, generates data about tools, spindle, and coordination system, and stores them in the internal buffer in order for the interpolator to use them. In addition, the IPR task interprets the control mode and user input from MMI tasks and input signals from the PLC task.

In the IPO task, various interpolation methods (linear, circular, and curve interpolation) are performed based on the interpreted data in the internal buffer and then acceleration and deceleration control is carried out to smooth the movement of the machine. Finally, the displacement and the velocity of each axis are computed and written in dual-port memory (DPM) in order to send them to the position control task. The final instruction generated from the IPO task is input as reference data to the PID control algorithm. The position control task is performed in order to minimize the difference between the actual position from the motor encoders and the reference data.

Table 10.1 Functions of each task module

Module	Task	Main function
NCK	Interpreter Task (IPR)	Control the operating mode of NC. Send position data and NC status to MMI. Communicate with the Interpolation Task. Communicate with the PLC task. Interpret NC blocks. Store the interpreted data in the block record buffer. Compensate for tool offset and tool length. Check software limits.
	Interpolation Task (IPO)	Read the block record buffer. Communicate with the Position Task. Read/write CNC-PLC interface. Perform Linear/Circular/ Spline/Polar interpolation. Perform real-time transformation. Compensate for backlash and pitch error. Carry out spindle processing.
	Position Task (POS)	Read the encoder. Calculate position error. Perform the position control algorithm. Monitor servo faults. Output velocity instructions to each drive.

Module	Task	Main function
PLC	Fast PLC Task (FPLC)	Handle interrupt signals. Handle inspection signals. Handle Fast I/O signals.
	Normal PLC Task (NPLC)	Handle Normal I/O signals. Handle M, S, and T codes.

Module	Task	Main function
MMI	Machine	Display auto-mode, MDI mode, and tool path.
	Program	Execute G-code editor, folder manager, and conversational programming system.
	Parameter	Manage parameters related to system, programming, and tool parameters.
	Tools	Manage tool offset, tool life, and tool shape.
	Utility	Manage DNC, PLC monitoring, alarm, and data communication
	Kernel	Manage screen display, key input, file management, application module handling, system boot up and external communication.

10.3.2 Design of the System Kernel

In order to perform the independent tasks of the CNC system, such as the MMI, NCK, and PLC tasks in Fig. 10.4, the system kernel should be designed based on a pre-emptive multi-tasking OS in which each task should be regularly executed within a specified time. In particular, the NCK/PLC system that must guarantee hard real-time property is executed regularly and the MMI system with soft real-time property is executed non-periodically.

Therefore, a variety of tasks are executed sequentially every specified sampling time. The position control task having the highest priority is executed every shortest sampling time, the interpolation task having the next highest priority is performed every multiple times of the sampling time of the position control task. The interpreter task having low priority is regularly performed during the spare time of the processor. Finally, the MMI system for user interface process, having the lowest priority, is designed as a non-cyclic process. Therefore, the MMI task is executed using the remaining computing power of the processor after all other tasks are finished. The scheduling method, above, is somewhat simple and is called "monotonic scheduling". It is suitable for a CNC system because the functions, or tasks, of a CNC system can easily be modularized and the behavior of each task working within the module depends on the sequential result of preceding tasks.

As an example of systems using the above-mentioned scheduling, two hypothetical systems having different CPU models are compared, as shown in Table 10.2.

In the case of the first system, using a Pentium-100, the sampling times of the position control task is set as 1 ms, the sampling time of the interpolation task is set as 2 ms, twice the sampling time of the position control task, and the sampling time of the interpreter task is set as 4 ms, four times the sampling time of the position control task. As shown in Table 10.2, it takes 210 μs to complete the position control task, it takes 470 μs to complete the interpolation task, and it takes 1800 μs to complete the interpretation task. Even with the system using a Pentium-75, the total time for the position, interpolation and interpretation tasks is less than 4000 μs, which means that the computing capacity of a Pentium processor is enough for the CPU of Soft-NC.

Table 10.2 Sampling time and execution time of each task

CPU	Position Time (ms)	Position Transit Time (μs)	IPO Time (ms)	IPO Transit Time (μs)	IPR Time (ms)	IPR Transit Time (μs)
Pentium-75	2	350	4	700	8	2600
Pentium-100	1	210	2	470	4	1800

The time occupation of the processes or tasks can be depicted as in Fig. 10.5. The loop cycle time of each task can be arbitrarily specified by the developer or user. In particular, because the cycle time of the interpolator and position control tasks has a direct influence on the performance of a machine tool, substantial caution is needed to harmonize the performance of machine tools.

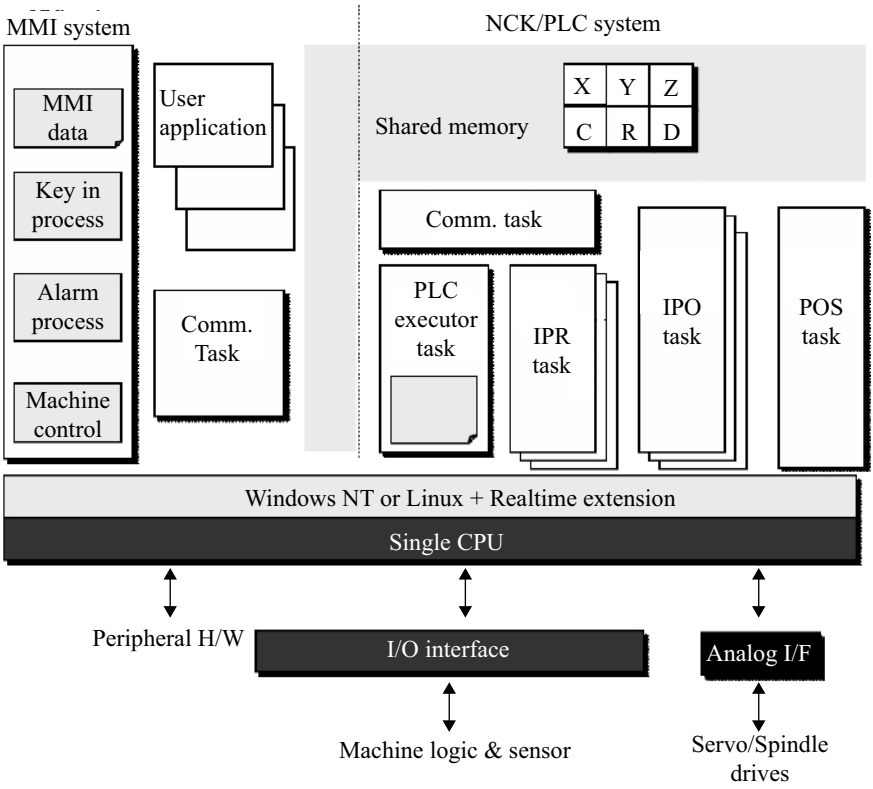


Fig. 10.4 Modular structure of Soft-NC

In the above example, the MMI uses the remains of the CPU power. Although the extra CPU power can be used for PLC it is omitted in Fig. 10.5 for explanatory convenience. It is supposed that the cycle time of the position control task is 1ms in Fig. 10.5. If a faster processor is used in order to shorten execution time, more computing power of the processor can be used for MMI and PLC. Therefore, we can see that as the processor's clock rate or capacity is increased, advanced tasks of Soft-NC that spend more time can be performed adequately.

10.3.3 PLC Program Scanning and Scheduling

After the code sent from a programming device is stored in the memory of the CPU module, it is executed sequentially by a PLC program executor. The PLC program executor plays the role of continuously scanning the user program and performing

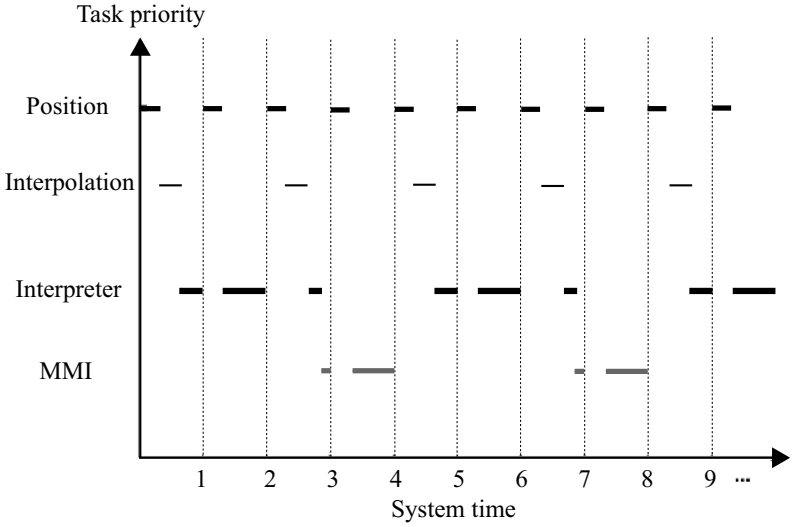


Fig. 10.5 Task scheduling of Soft-NC

the specified instructions. The execution flow of the PLC program executor is to scan iteratively the input/output ports and PLC program as shown in Fig. 10.6. The scan cycle consists mainly of three steps:

Step 1: Checking input ports: first, PLC checks whether an input port is on or off status and then it stores these statuses in memory.

Step 2: Executing a program: PLC executes a program sequentially code by code. It calculates the status of output ports based on the status of the input ports obtained in Step 1. The calculated results are stored for use at the next step.

Step 3: Changing output: Finally, PLC changes the output port status based on the results from Step 1 and Step 2. After finishing Step 3, PLC returns to, and repeats Step 1.

Accordingly, one scan time is defined as the total time spent to perform the three steps and, in general, some tens of milliseconds to some hundreds of milliseconds are spent. Since the PLC task consists of programs with various priorities, the scheduler of the OS manages them in order to execute them in real time.

The functions executed by PLC can be summarized with respect to the response time and periodic behavior, as shown in Table 10.3. PLC tasks can be variously classified, from tasks that require high-speed response, such as an interrupt to a program, to tasks that require low-speed response, like user utility functions. Therefore, if they are programmed in one program, tasks that require high-speed response may not be completed in the specified time because all tasks are executed once during one scan time.

Therefore, it is necessary to assign a priority to each task and complete tasks sequentially depending on the priority of the task in order to overcome the above problem. Of course, high priority is assigned to tasks having short scanning time and

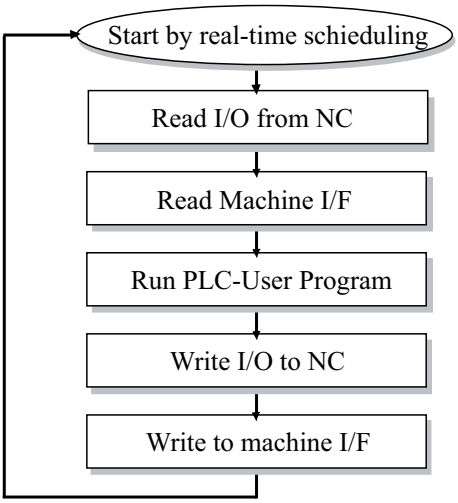


Fig. 10.6 Scan cycle of PLC program executor

requiring prompt action. It can be designed that an emergency stop, for example, is performed immediately by using an interrupt signal if the emergency button is pushed. As shown in Table 10.3, a PLC program is classified as four kinds of tasks. The start of each classified program is controlled by a system scheduler. Eventually, the priorities of all tasks in the CNC system can be defined, as with the example in Table 10.4.

Table 10.3 Priorities of PLC tasks

Interrupt Program	1. Irregularly executed from the input signal of a sensor. 2. The program with the highest priority among PLC tasks. 3. Used for handling signals requiring quick response. 4. Processing within hundreds of microseconds. 5. Feedback control routine performed by PLC system.
Fast Processing Program (Fast Task)	1. Repeated within every several milliseconds. 2. Highest priority among cyclic tasks. 3. Used for handling signals requiring quick response. 4. Processing under 1 ms. 5. Routine for counting position of turret and ATC.
Main Processing Program (Normal Task)	1. Repeated every tenth of a millisecond. 2. Program edited by user. 3. Program length should be controlled for executing every 10–20 ms. 4. PLC program edited in a variety of languages such as Ladder diagram.
Custom Program	1. Executing custom program after completing main program. 2. Program having the lowest priority. 3. Programs such as screen display and serial interface.

10.3.4 Task Synchronization Mechanism

In order to perform the tasks successfully according to the schedule, a synchronization mechanism for the tasks is needed. The following synchronization mechanism can be used for Soft-NC.

To activate the components of the CNC system in a multi-processing environment, first, a priority is assigned to each tasks and then a schedule made according to the assigned priority. Table 10.4 shows an example of assigning priority to tasks. The position control task has the highest priority and the MMI task has the lowest priority. The PLC task is divided into the Fast PLC task and the Normal PLC task. The Fast PLC task has lower priority than the Interpolation task, while the Normal PLC task has lower priority than the Interpreter task.

Table 10.4 Priority assignment for various tasks

Task	Priority
CNC Position task	2
CNC Interpolation task	5
Fast PLC task	12
CNC Interpreter task	17
Normal PLC task	22
Comm task	36
MMI task	64

To synchronize all tasks of the CNC system an OS provides various mechanisms, such as global variables, semaphores, and mailboxes. In this section, we use the semaphore to design Soft-NC, as shown in Fig. 10.7. The system kernel procedure for synchronization is as follows:

1. Create the semaphores for the position control task and the interpretation task.
2. Create the position control task and the interpretation task and then set them into idle status.
3. Set an interrupt enabled by the timer for the position control task.
4. On receiving the semaphore from the position control task, the interpretation task continues to execute its own routine.

As shown above, all tasks, except for the position control task, are put into sleep status after they have been created, to wait for the semaphore from the position control task. The position control task checks the status of all tasks before releasing the semaphore to pass execution right to another task. Only when a particular task is in idle state does the position control task release the semaphore. However, if the particular task is in busy state, the semaphore is not released and the task in busy state is rescheduled based on the priority-based scheduling according to the priorities shown in Table 10.4.

```
.....
while(True)
{
    IPR_status = idle;
    /* wait for semaphore form POS task */
    receive_unit (IPR_semaphore, .....);
    IPR_statue = busy;

    /* execute IPR routines */
       for (i=0; i<number of routine; i++)
          { routine table [i];   }

    IPR_Act_CNT++;
    }
```

Fig. 10.7 Task synchronization by semaphore

The design of scheduling and synchronization to activate the tasks of the CNC system by the system kernel software mentioned above was described. Now we consider the routines included in particular tasks. How these routines are executed based on the scheduling of the system kernel will be addressed.

PLC is a large process having a series of continuous subtasks. If we investigate the working flow of NCK, NCK has an iterative loop so that the data calculated by the routine of the interpreter task are stored in the block buffer, the interpolator performs interpolation, compensation, and spindle control after reading the data from the block buffer, and, finally, the interpolated data is sent to the position controller.

In the case of the PLC process, the work flow is similar to the NCK process. The PLC process scans the input/output ports iteratively every specified sampling time, performs the user program, and outputs the operation results. Although the MMC process is executed in various ways depending on user's input, it does not result in any problem for real-time control when MMC is handled as conventional application software. Therefore, when the NCK and PLC processes use processors according to scheduling, their own routine is repeated until an interrupt stops the current routine.

If a process is interrupted during execution, it stops execution promptly and passes the right to use the processor to the task with next priority. When it gets its own turn, it starts from the new routine, which was the next routine due before interruption at the previous turn. If the interrupt does not occur during execution, the right to processor use is passed to the task having the next priority after the current task has completed. In conclusion, the routine for each task is performed as a closed-loop form and the routine is executed iteratively during the specified time. This mechanism is shown as the FOR-loop of the program code in Fig. 10.7.

It is necessary to load the functional modules into memory during system boot-up in order to perform the functional modules sequentially after a particular task takes the right of processor usage via a semaphore. The functional modules are loaded into

the memory segment assigned to each task regardless of the execution order when the system is booted up. If the same functional module exists on an auxiliary storage device, the system software loads the updated module by using the module header of the functional module loaded in the memory segment. To execute the functional modules sequentially in a memory segment according to the assigned priority, we can use the routine table defined in the data segment (CS), as shown in Fig. 10.8. Accordingly, the routine table is created according to the routine's priority during system boot-up and the corresponding routines are executed continuously after system boot-up has been successfully completed.

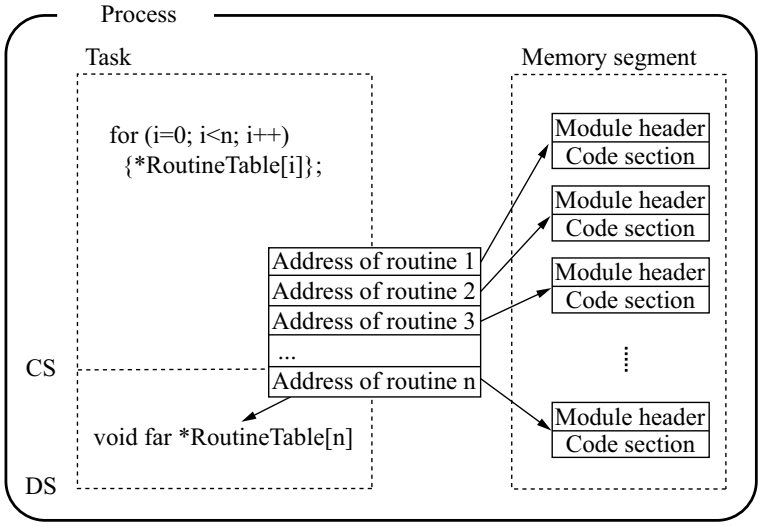


Fig. 10.8 Execution method of routines in a task

Let us investigate how other tasks take the semaphore from the position control task and how they are activated at the specified time. Figure 10.9 shows that the interrupt from the timer wakes up the position control task every sampling time, performs a series of processes related to position control, and then passes the right of processor usage to another task using the semaphore.

In the above example, the sampling time of other tasks is designed to be a multiple of the sampling time of the position control task. So, it is possible to control the frequency of other tasks. The variables defined in the above are defined as follows.

- *Task_status*: this is the flag variable to denote the status of a task. If the task has completed before the interrupt, it has 'idle' value. If the task was interrupted during execution it has 'busy' value.
- *Task_Set_CNT*: This is computed by dividing Task_Loop_Time by Position_Sampling_Time. It defines the execution frequency of the task.

```
PositionTask()
{
    ..............................
    IPO_Act_CNT = IPO_Act_CNT % IPO_Set_CNT;
    If(IPO_Act_CNT == 0)
    {
        if (IPO_statur = busy)
          return busy;
        else
          /* Send semahpore to activate IPO task */
          send_unit(IPO_semaphore);
    }

    IPR_Act_CNT = IPR_Act_CNT % IPO_Set_CNT;
    If (IPR_Act_CNT ==0)
    {
        if (IPR_status = busy)
          return busy;
        else
          /* Send semaphore to activate IPR task */
             send_unit (IPR_semaphore);
    }
    ..............................
}
```

Fig. 10.9 Semaphore in the position control loop

- *Task_Act_CNT*: means how many times the tasks are executed. So, whenever the task completes its own routine, this variable is increased by one.

Suppose that the sampling time of the position control task is 1 ms. If the user sets Task_Loop_time to 4 ms, Task_Set_CNT becomes 4. This means that whenever the position control task is executed four times, the task with 4 ms Task_Loop_Time is performed once.

Let us consider how to control the sampling time of the position control task accurately. First of all, it is possible to assign the interrupt by an external timer to one from among the PC interrupts IRQ10, 11, 12, 13, 14, and 15. Whenever the interrupt is signalled in a constant period, the interrupt service routine performs the position control task and then rate monotonic scheduling (RMS) operates, as in the above example. As described above, the non-maskable interrupt based on the hardware that is capable of generating predictable time ticks is typically used for hard real-time systems.

Another method is to use a high-resolution timer that a real-time OS has unlike a general-purpose OS. As an example, RTX, the real-time extension of Windows NT, has its own timer whose resolution is 100 ns and various system ticks can be generated by using it according to the needs of the application software. In most cases of a real-time OS, it is possible to set the highest priority to a self-waking thread. Therefore, synchronization by software timer is possible without any external

timers. For example, the self-waking thread in QNX can be implemented easily, as shown in Fig. 10.10.

```
while (True)                //infinite loop
{
    RTSleepFT(&period);
// It is difficult to get a deterministic response by using
    Sleep() of Windows NT.

// Execute some control routines here
// example, Position control

}
```

Fig. 10.10 Synchronization using a software timer

As shown in the above code, if the self-waking thread is set to have the highest priority, another process takes the right for usage of the CPU at every specified time period. When, the specified time of the timer has elapsed, the process takes the right for usage of the CPU and executes, for example, position control. Using this mechanism, a particular process can be repeated every specified time in a deterministic manner.

10.3.5 Inter-Task Communication

Up to now, the design concept for the basic modules and system kernel of Soft-NC has been investigated, and widely uses multiple tasks activated by semaphores and a loop drive method using a routine table as an implementation example. In this section, data flow and the communication mechanism between modules will be addressed.

For the data handled in the CNC system, various data formats are used, from bit data for status flags to ASCII files for part programs. According to the direction, the frequency, and the method of data transmission between modules, the data may be classified as shown in Table 10.5.

Therefore, from the point of view of each module, the data transmitted between modules can be summarized as in Table 10.6, according to the direction, data type, and transmission frequency.

As a method for sending a variety of data, shared memory has typically been used. For accessing shared memory from different modules, there are the Direct Access method (reading the data stored in memory) and the Request/Answer method (reading the data produced by requests), as shown in Fig. 10.11.

Table 10.5 Communication data classification of CNC system

Classification	Items	Comment
Data	Files	NC program, PLC program, NC parameter
	Command	Request service
	Variable	Various data set
Direction	NCK/PLC → MMI	Data for display
	MMI → NCK/PLC	Transfer information for op.
	PLC ↔ NCK	Bidirectional data
Frequency	Event	Data transfer if needed
	Cyclic	Data transfer periodically
Method	Shared memory	Dual port memory
	Comm task	Communication Task

In the Direct Access method, various data such as screen display, the status variables of processes, fast input signals and error messages, are stored and updated in the shared memory in a periodic manner or through events occurring at irregular time intervals. In this method, the transmission mechanism is controlled by turning a special flag ON or OFF. For example, in the case of sending some data from NCK to MMI, the NCK sets the flag to zero and sets the flag to one after updating the shared memory. At this moment, MMI reads the data in shared memory when the flag becomes one and changes the flag to zero after completing reading the data. Using to the above simple flag management, the reading and writing actions of two processes do not compete. The same mechanism can be used for sending data from MMI to NCK. The Direct Access method is appropriate for a job to send bit data that is periodically updated and event-driven.

Table 10.6 Inter-module communication data classification

Direction	Classification	Item	comments
NCK/PLC → MMI	CNC Status	Actual Position/Velocity Distance to go Actual Lag, Velocity % Final Position for NC block Active G code Active Block number Selected Axis Tool information/ Correction	Display purpose Cyclic
	Part Program	Program number/name Subprogram number/ name Program repeat/ Program active	
	Parameter	Control Gain PLC Tool Programming	Display purpose Request/Answer
	System State	Emergency stop Cycle stop	Cyclic
	Spindle/Feed	Actual RPM/feed Programmed RPM/feed Maximum RPM/feed Minimum RPM/feed Override % Active Spindle number	Display purpose Cyclic
	PLC Information	Input/Output Byte information Timer information	Display purpose Request/Answer
	Error Handling	Error Level - Key/command delete - Control reset - Cold/warm start Error Type - Editor - Block processing - Servo - Hardware - PLC - Communication	Event
	Message Handling	Send Message	Event

Figure 10.6 (continued)

PLC ↔ NCK	Interface	Signal Input / Output A/D, D/A information Spindle Value/ Axis Position Handwheel values NC parameter NC correction	Cyclic
MMI → NCK/PLC	Operation Mode	Cycle Start/stop Auto/Manual/MDI Single block/Homing Warm start Jog+/Jog- Time(sec/min/hour)	Cyclic
MMI ↔ NCK	File Management	Program file Transfer System parameter PLC program loading Copy/Delete file	Request/Answer

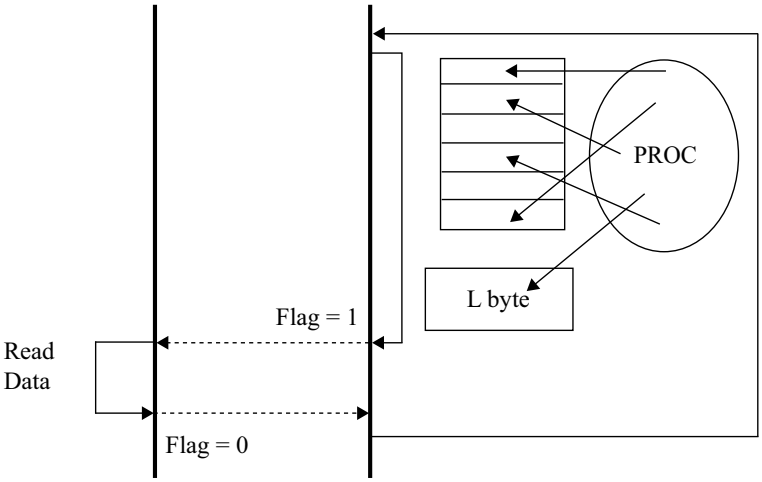


Fig. 10.11 Direct access method

The Request/Answer method shown in Fig. 10.12 does not automatically update the shared memory. When a service is requested by the module (requester) intending to access the shared memory, the module (receiver) asked of a service passes the right to access the shared memory after finishing the action corresponding to the requested service.

When a service is requested to the receiver, the requested service is interpreted, and recognition of the request is informed to the requester. After interpreting the requested service, the receiver writes the response with respect to the requested service to the shared memory, and allows the requester to access the shared memory. When

the requester retrieves the data generated by the requested service, an acknowledgement signal is transmitted to the receiver, and the process of accessing the shared memory is terminated.

In general, this method is used for safe sending of long data such as program files, system parameters, and PLC programs to the NCK/PLC modules from the MMI module. For example, in the case of file transfer between MMI and NCK, NCK sends MMI a request asking to send a part program to MMI and then MMI transmits the program file to NCK via the shared memory.

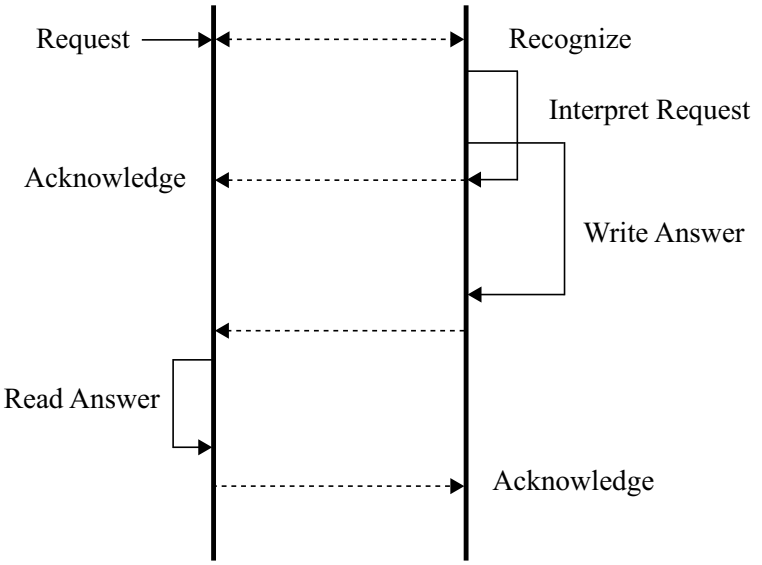


Fig. 10.12 Request/Answer method

The typical data structure of shared memory for communicating data in Operation Mode can be realized as shown in Fig. 10.13.

10.3.5.1 Inter-task Communication in NCK

Data exchange between IPR, IPO, and POS tasks in the NCK module can be designed by using shared memory, as shown in Fig. 10.14.

A General Communication Area is the area in shared memory where states, positions, and control flags are exchanged between the IRP task and the IPO task. Emergency Stop, Axis Position, Error Handling Diagnosis, Debug Information, and Watchdog Information are transmitted via the General Communication Area.

A Block Record Buffer is also defined, and is the area used for transmitting data about NC blocks to the IPO task, the IPR can only write the data to it, the IPO can only read data from it. It is defined as an area of shared memory and has a ring buffer

SMDB11	NCK → MMI (Operation status)							
Byte (or Offset)	Bit 7	6	5	4	3	2	1	0
0				Free hold		Stop	Operation	
1	Active G-Code No. (Group 1)							
2	Active G-Code No. (Group 2)							
⋮					⋮			
10 - 11	Active Block No.							
⋮					⋮			

Fig. 10.13 Typical data structure of shared memory

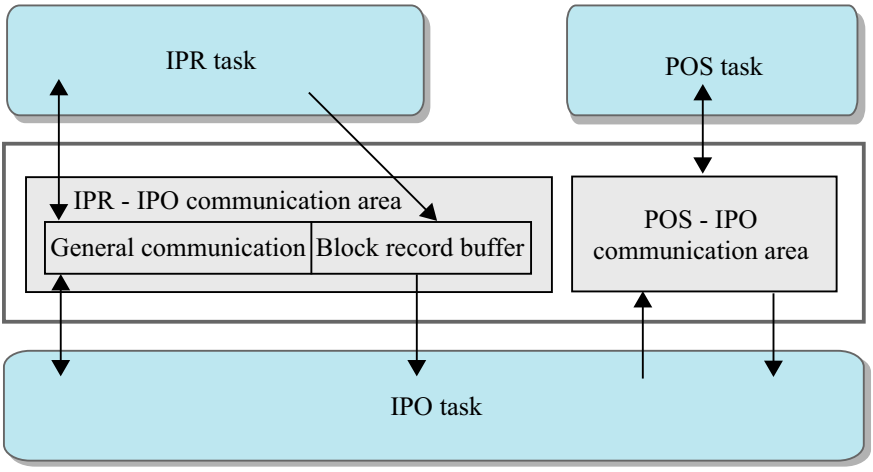


Fig. 10.14 Data exchange in NCK module

structure. Interpolation Type, Start and End Position, Corrections, Axis components, Path Parameters, Path Velocity, Dwell time, and Spindle Information are stored in the Block Record Buffer.

The POS-IPO communication area is the area for exchanging data between the IPO task and the POS task. It is defined in shared memory and Command Axis Position, Spindle Output, Machine Position, Emergency Stop, Watchdog Information, and Error Message are exchanged via it.

10.3.5.2 Communication Between NCK and MMI

Where the MMI and NCK/PLC modules operate on a single OS and single CPU, communication between MMI and NCK can be designed easily. However, in the case

of a CNC system where the MMI and NCK modules operate on individual CPUs, a bus interface and high-speed serial interface can be used for communication between modules. This configuration has a more complex communication mechanism than Soft-NC. Not only should the communication speed necessary for meeting the real-time requirements be considered but also reliability, communication distance, cost, extensibility, and robustness.

The Open System Interface (OSI) seven layers shown in Fig. 10.15 has been typically used as the layer architecture for communication software. However, the OSI seven-layer method is not appropriate in industry because it may not meet real-time requirements. So, to realize the high-speed communication between MMI and NCK, simplified layers are used based on three layers from among the seven layers of OSI. Field buses such as BACNET and PROFIBUS have been developed based on the simplified layers.

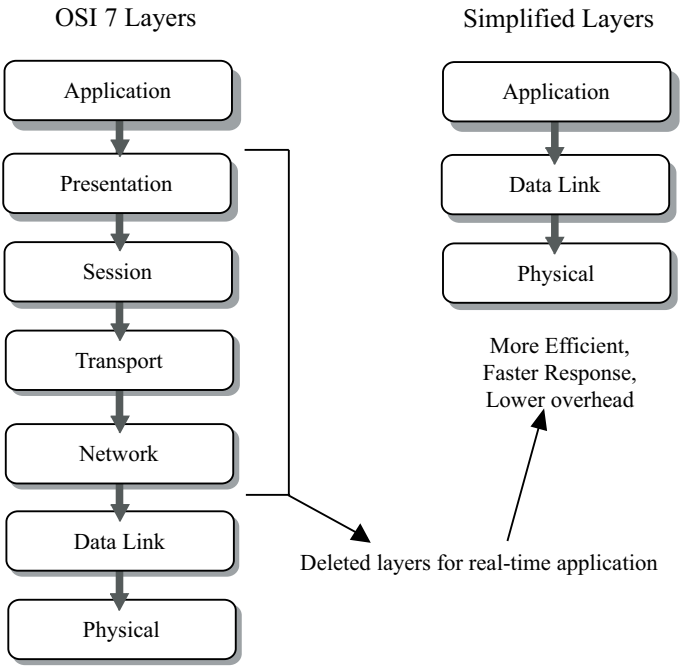


Fig. 10.15 Comparison between OSI 7 and Simplified Layers architectures

The communication software layer architecture to which the above 3-layer architecture is applied is shown in Fig. 10.16. It consists of a Hardware Driver for activating the Ethernet Controller and an Interface Level Driver for providing network services and interface with application software. On the side of NCK, the communication task is designed as application software based on real-time OS. The communication task of MMI is designed as application software based on Windows 95. Therefore, on the side of MMI, the communication task can be designed easily by

using an Ethernet driver for PC and Windows OS. However, in the case of NCK, the communication task can be developed only when a real-time OS supports Ethernet.

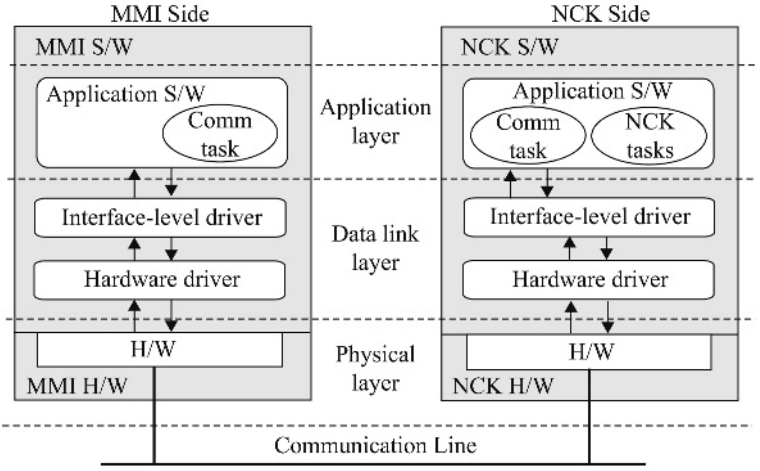


Fig. 10.16 Software layer for communication between MMI and NCK

To manage the communication data efficiently, it is necessary to define the data block for assembling the related data in shared memory. Having defined the block for communication data of MMI, as for the block for NCK, it is possible to develop easily application software for each module.

10.4 Motion Control System Programming Example

In this section, the example of system programming for implementing a motion control system in a real-time environment based on the NCK module detailed in Chapter 6 will be described. As the development environment of the motion control system, PC hardware with INTEL processor, Windows NT, and RTX (VentureCom) were considered.

The Acc/Dec-Control-Before-Interpolation-type NCK, which was implemented in this section, consists of the Rough Interpolation task, the Acc/Dec control task, the Fine Interpolation task and Position Control tasks. As mentioned above, it is necessary not only to implement all tasks but also to execute them successfully for realization of NCK. For this, various API functions and techniques provided by RTX were used for system programming. This section shows only one example of designing NCK based on RTX and the system programming of NCK will be described based on the implemented code.

10.4.1 Design of System Architecture

The implemented ADCAI-type NCK, as shown in Fig. 10.17, is composed of main function, timer handlers for the tasks of NCK, event server and handler for handling the instruction from MMI, and ring buffers for transmitting the data between tasks.

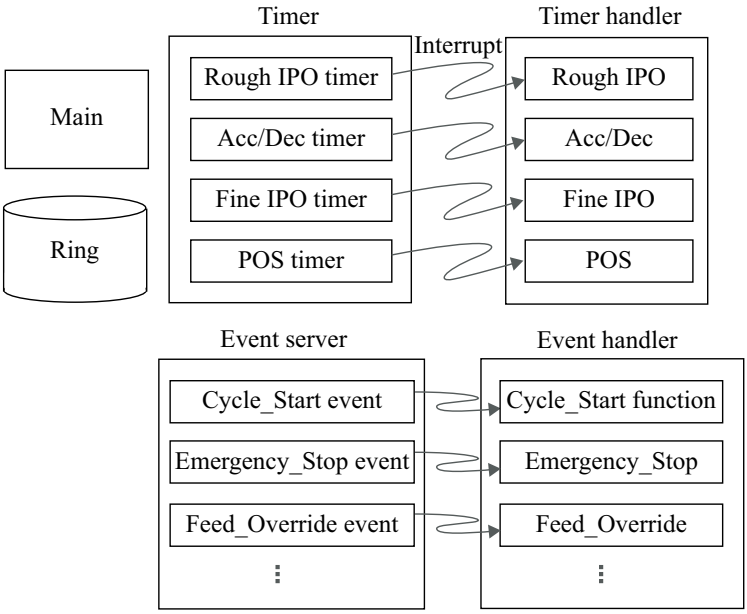


Fig. 10.17 Sample architecture of ADCAI-type NCK

The main function is the function that is called first when the NCK boots up and performs the following steps:

1. To initialize internal variables required for execution of NCK.
2. To create the timer and timer's handler for each task.
3. To create the event server for handling the instruction from MMI.
4. To create ring buffers for transmitting data between tasks.

The timer handler is used for tasks that are iteratively activated every specified time. The event handler is used for handling aperiodic events.

Because the rough and fine interpolation tasks, Acc/Dec control task, and position control task are performed periodically, as shown in Fig. 10.17, they were realized using a timer. MMI instructions, such as cycle start, emergency stop, and feed override, were realized by an event handler. (Note, because the emergency stop is a very urgent instruction, it is realized by unique hardware interrupt. This example is only to show how to implement motion control.)

10.4.2 Creating Tasks

An individual timer for each task is created. This timer activates iteratively particular handler functions every specified time. The following shows how to use the timer for creating the Acc/Dec control task using RTX API.

```
// ********************************************************** //
// Description: Create Timer, set start time and period //
// ********************************************************** //
hAccDec = RtCreateTimer(
        NULL,              // Security - NULL is none
        0,                 // stack size - 0 is use default.
        DoAccDec,          // Timer handler
        NULL,              // NULL context
        P_ACCDEC,          // priority
        CLOCK_2);          // RTX HAL Timer
    RtSetTimer(hAccDec, &Start_time4, &RTPeriod);
```

As the function RtCreateTimer() shown in the above code is for generating a timer, it receives data about security option, the size of stack, the name of the timer handler, priority, and the reference time as arguments. “DoAccDec”, the third argument, is the name of the timer handler function. And P_ACCDEC, the fifth argument, gives the priority of timer handler.

It was mentioned that the latency time is the key performance index of RTOS in Chapter 9. The average latency time of RTX is within some tens of μs (RTX version 4.3 and P-III 866MHz). The maximum latency time does not exceed several μs. This means that the timer provided by RTX can be used to implement a task whose sampling time is 1 ms. Actually, regardless of the assigned priority, there is the non-interrupted job that hinders real-time tasks. In the case of RTX, access (reading/writing) to the hard-disk driver belongs in this non-interrupted job category.

Therefore, to guarantee the real-time property, it is essential to find a non-interrupted job and avoid executing it together with real-time tasks.

10.4.3 Task Synchronization

In order to synchronize the execution of tasks, the start time and the sampling time of each task must be properly decided. The result of the rough interpolation task is used as the input to the acc/dec control task. Further, the result of the acc/dec control task is used as input to the fine interpolation task. The result of the fine interpolation task is used as the input to the position control task. Therefore, the rough interpolation task, acc/dec control task, fine interpolation task, and position control task must be started in that order. This can be represented by Eq. 10.1. In Eq. 10.1, TS_{RIPO}, TS_{ACCDEC}, TS_{FIPO}, and TS_{POS} mean the start times of the rough

interpolation task, acc/dec control task, fine interpolation task, and position control task, respectively. Moreover, A ¡ B denotes that task A starts before task B.

$$TS_{RIPO} < TS_{ACCDEC} < TS_{FIPO} < TS_{POS} \tag{10.1}$$

The implemented NCK can perform continuous mode machining. In order to implement continuous mode machining, the Acc/Dec control task requires the interpolation result from two previous blocks. Therefore, the start time of the rough interpolation task and the Acc/Dec control task can be represented as in Eq. 10.2:

$$TS_{RIPO} + 2 * P_{RIPO} < TS_{ACCDEC} \tag{10.2}$$

where, P_{RIPO} means the sampling time (cyclic time) of the rough interpolation task.

The fine interpolation task and the position control task were separately implemented. Once the fine interpolation task has been performed, the position control task is executed eight times. (The sampling times of the fine interpolation task and the position control task are set to 16 ms and 2 ms, respectively.) According to the description, above, the relationship between the sampling time and the start times of tasks is represented by Eq. 10.3:

$$TS_{FIPO} < TS_{POS} < TS_{FIPO} + P_{POS} \tag{10.3}$$

The relationship shown in Eq. 10.3 is because data transmission from the fine interpolation task to the position control task is done by shared memory instead of a ring buffer. In general, the ring buffer is used when the difference between the speed of producing data and the speed of consuming it is not constant. In the NCK module, the data transition between the Acc/Dec control task and the fine interpolation task is a good example of the use of the ring buffer. Whenever the Acc/Dec control task is carried out once, the Acc/Dec profile for one block is generated. However, the fine interpolation task consumes one profile segment corresponding to one sampling time. A single execution of the Acc/Dec control task generates data that the fine interpolation task can consume during several tens or several hundred time periods. Because the data consumption speed of the position control task and the data production speed of the fine interpolation task are equal and fast data access speed is needed, shared memory was used. Because the usage of shared memory instead of the ring buffer makes the execution speed of the task predictable, the usage of shared memory has a positive influence on the reliability of the implemented NCK module.

After the position control task consumes the whole result from the fine interpolation task, the fine interpolation task interpolates the path segment (precisely speaking, the velocity profile) corresponding to the next sampling time. According to Eqs. 10.1, 10.2, and 10.3, the start time of tasks is specified as follows:

```
// ********************************************************* //
// Description: Set start time and period for each timer //
// ********************************************************* //
// Set reference time.
    RtGetClockTime(CLOCK_2, &Start_time);
```

```
// Start rough IPO after 20 milliseconds.
    Start_time1.QuadPart = Start_time.QuadPart + 200000;
     // Start ACC/DEC after 54 milliseconds.
    Start_time4.QuadPart = Start_time.QuadPart + 200000 + 160000 +
                                                        160000 + 20000;
     // Start fine IPO after 62 milliseconds.
    Start_time3.QuadPart = Start_time.QuadPart + 540000 + 80000;
     // Start POS after 79 milliseconds.
    Start_time2.QuadPart = Start_time.QuadPart + 630000 + 160000;
     // Set repeat for each timer.
    RTPeriod.QuadPart = 160000;       // 16 msec
    POSPeriod.QuadPart = 20000;       // 2 msec
     // Create rough timer and start.
    hRIPO = RtCreateTimer(
        NULL, // Security - NULL is none
        0, // stack size - 0 is use default.
        DoRIPO, // Timer handler
        NULL, // NULL context
        P_RIPO, // priority
        CLOCK_2); // RTX HAL Timer
     RtSetTimer(hRIPO, &Start_time1, &RTPeriod);
```

In the above example, RtGetClockTime() is the function to get the current time and the time from RtGetClockTime() is used as the reference time (the reference time is saved in the variable Start_time.). The start time for each task is decided on by adding a particular time to the reference time. For example, the rough interpolation task starts after 20 ms from the reference time. The position control task starts after 79 ms from the reference time. 20 ms is the allowance time spent to set variables and create the timer for the rough interpolation task. The Acc/Dec control task starts after the rough interpolation task has been executed twice and the allowance time, 2 ms, has passed.

As mentioned above, in RTX the timer is generated by RtCreateTimer() and the timer is set by RtSetTimer(). RtSetTimer() gets the timer's handler, the start time of the timer and the frequency of the timer are given as arguments. hRIPO, the first argument, denotes the handler of the rough interpolation task's timer and Start_time1, the second argument, means the start time of the timer. RTPeriod, the third argument, gives the sampling time of the timer.

The above was applied to the NCK implemented in this book. Depending on the programming environment and algorithms, the description above varies.

10.4.4 Task Priority

Priority is applied not only to a timer's handler but also to an event handler. Equation 10.4 denotes the priorities of tasks. In Eq. 10.4, PR_{RIPO}, PR_{ACCDEC}, PR_{FIPO}, and PR_{POS} denote the priorities of the rough interpolation task, the Acc/Dec control task, the fine interpolation task, and the position control task, respectively. PR_{ES} means the priority of the event for the emergency stop instruction. PR_{MMI} denotes the priority of the event for other instructions from the MMI except for the emergency stop instruction.

In Eq. 10.4, $A < B$ denotes that the priority of task A is lower than that of task B. And $A <= B$ denotes that the priority of task A is less than or equal to that of task B.

$$PR_{MMI} < PR_{RIPO} <= PR_{ACCDEC} < PR_{FIPO} < PR_{POS} < PR_{ES} \tag{10.4}$$

10.4.5 Inter-Task Communication

A ring buffer and a shared memory are used for communication between tasks in the NCK module.

The following is the data structure for the ring buffer between the interpretation task and the rough interpolation task and the function to access the buffer. In general, when the memory size is not enough, a ring buffer is used and, by using the ring buffer, the usage of memory can be restricted. In particular, because of the difference between the speed of producing data from the Acc/Dec control task and the speed of consuming data by the fine interpolation task, memory may be used excessively. Excessive usage of the memory may reduce the performance of system. To overcome this problem, each task checks the number of items stored in the buffer. If as many items exist as the specified number, the task does not work. Using this method, the number of items does not exceed the specified size and the following code works like a ring buffer.

```
// ********************************************************** //
// Description: Ring buffer structure and handling function //
// ********************************************************** //
 // Ring buffer structure for communication between IPR and rough IPO
typedef struct CRingIRTag {
    int nGCode;                 // IPO type 0 : G00, 1 : G01, 2 : G02
    Vector Start;               // Start position (mm)
    Vector End;                 // End position (mm)
    Vector Cen;                 // Center of circular IPO (mm)
    float dRadius;              // Radius of circular IPO (mm)
    float dFeed;                // Feedrate (mm/min)
    float dSpindle;             // Spindle speed (RPM)
    short int nSpindleDir;     // CW: 1, CCW: 2, Stop: 0
```

```
    int nStatus;                    // block status 0: start, 1: end
    int nControlMode;               // EXACTSTOPMODE
                                    // /BLOCKOVERRAPMODE
    int nBlockNumber;               // Index number of block
    int nWorkingstepID;             // Working step ID
    BOOL IsProgramEnd;              // TRUE: end of block, FALSE: under machining
    struct CRingIRTag* next;
} CRingIR;
 // Head and tail of buffer
typedef struct CIRListTag
{
    CRingIR* head;
    CRingIR* tail;
}CIRList;
 // ********************************************************** //
// Description: Add item at tail of buffer //
// ********************************************************** //
void CIRList_AddTail(CIRList* list, CRingIR* item)
{
    item− >next = NULL;
     if(list− >tail == NULL) {
        list− >head = item;
        list− >tail = item;
    }
    else {
        list− >tail− >next = item;
        list− >tail = item;
    }
}
 // ********************************************************** //
// Description: Delete item structure at head of ring buffer //
// ********************************************************** //
BOOL CIRList_RemoveHead(CIRList* list)
{
    CRingIR* temp;
     if(list− >head == NULL)
        return TRUE;
     if(list− >head == list− >tail) {
        free(list-¿head);
        list− >head = list− >tail = NULL;
        return TRUE;
    }
    else {
        temp = list− >head;
        list− >head = list− >head− >next;
```

```
        free(temp);
        return FALSE;
    }
}
// ********************************************************** //
// Description: Delete all item structures in ring buffer. //
// ********************************************************** //
void CIRList_DeleteAll(CIRList* list)
{
    BOOL bRtn;
     while(1) {
        bRtn = CIRList_RemoveHead(list);
        if(bRtn)
            break;
    }
}
```

Let us investigate communication between the NCK module and the MMI module. The NCK module sends the MMI module data about the status of NCK and MMI displays the data. Thus, the NCK module must do real-time communication with an external system and shared memory is used for the real-time communication. The following code is an example of creating shared memory in RTX. RtCreateSharedMemory() is the function for generating shared memory and it has the reason for access to the memory, the name of the memory, and the variable for storing the memory's virtual address as arguments. PAGE_READWRITE, the first argument, denotes that the created shared memory is to be used for reading and writing some data. The second and third arguments denote the size of the shared memory. "StatusDBName", the fourth argument, means the name of the shared memory. Other processes can access the shared memory through "StatusDBName". "locate", the fifth argument, is the name of the variable for storing the virtual address of the shared memory.

```
// ********************************************************** //
// Description: Create a shared memory. //
// ********************************************************** //
HANDLE hShm;
DWORD dwMaximumSizeHigh = 0;
PVOID locate;
 hShm = RtCreateSharedMemory(PAGE_READWRITE,
                             dwMaximumSizeHigh, sizeof(struct NC_Status_DB),
                             "StatusDBName", &locate);
 RtUnmapSharedMemory(locate); // Unmapping of created memory
RtCloseHandle(hShm); // Close handle of shared memory
```

When other processes or threads want to use the shared memory generated by the above code, access to the shared memory is possible by using RtOpenSharedMemory(). The function RtOpenSharedMemory() has the reason for the access to the memory, the name of the memory, and the variable for storing the memory's virtual address as arguments. In the example below, the first argument, SHM_MAP_WRITE, denotes that the shared memory is opened for writing some data. The third argument "StatusDBName" is the name of the shared memory. The fourth argument, locate, is the name of the variable for storing the memory address. The second argument has no meaning here and is ignored in the function. The code below is to show that the position and velocity of an axis are written in the shared memory. After using the shared memory, it is necessary to free the virtual address and the handler of the shared memory by using RtUnmapSharedMemory() and RtCloseHandle(). In the code below, the data written in the shared memory is periodically read by MMI every specified time interval and is displayed.

```
// ********************************************************** //
// Description: Access the previous shared memory. //
// ********************************************************** //
hShm = RtOpenSharedMemory(SHM_MAP_WRITE, FALSE,
                                    "StatusDBName", &locate);
 pStatus = (NC_Status_DB*)locate;
pStatus− >CommandX = CurPos_X.dCommand;
pStatus− >CommandY = CurPos_Y.dCommand;
pStatus− >CommandZ = CurPos_Z.dCommand;
pStatus− >CurrentX = CurPos_X.dGmOut;
pStatus− >CurrentY = CurPos_Y.dGmOut;
pStatus− >CurrentZ = CurPos_Z.dGmOut;
pStatus− >CurrentFeed = ActualFeed;
 RtUnmapSharedMemory(locate);
RtCloseHandle(hShm);
```

10.4.6 Create Event Service

In general, shared memory is used for data transition between processes and an event is used for sending the on/off signal. The instructions for cycle start and emergency stop from MMI to NCK are examples of the usage of an event. (Note that, in general, the on/off signal of switches is sent to PLC via a memory map. The on/off signals of switches are written in a memory map and PLC checks the memory map iteratively every specified time period. If PLC notes the change of a particular value in the memory map, PLC performs the corresponding task. The memory map is used for sending/receiving massive data amounts between hardware components (*e.g.*, MMI operation panel, NCK board, and PLC board). However, when the size of the data

is small and all components are implemented in software, an event may replace the memory map.)

Communication between the MMI operation switch and NCK/PLC was implemented via an event. The usage of an event makes it possible for NCK/PLC to respond quickly and handle MMI operation switch without delay. It can reduce the burden for PLC or NCK to monitor a memory map iteratively. The following code shows an example for generating an event variable and a handler for handling the cycle start instruction.

```
// ********************************************************** //
// Description: Create Event //
// ********************************************************** //
// Create event object.
hStartEvent = RtCreateEvent(
                NULL, //security attribute
                FALSE, //manual reset
                FALSE, //initial state
                "NCKSTARTEVENT" //the event name
                );
// Create thread to handle the event.
hStartHandler = RtCreateThread(0, 0, StartHandler, NULL,
                                    CREATE_SUSPENDED, 0 );
// Priority assignment of thread.
if (RtSetThreadPriority(hStartHandler, STARTPRIORITY) == FALSE)
{
    RtPrintf("RtSetThreadPriority error = %d
n",GetLastError());
    ExitProcess(1);
}
// Start thread.
dwSuspendCount = RtResumeThread(hStartHandler);
```

The usage of an event follows four steps. The first is to create the event variable (object). The second is to create a thread for handling the event. The third is to assign a priority to the generated thread. The fourth is to activate the thread. RtCreateEvent() is the RTX API function for creating an event variable (object) and receives the identification of the event object and initial status of the object as arguments. "NCKSTRTEVENT", the fourth argument, denotes the identification of the event object and FALSE, the third argument, means that when the event object is created, it is in non-signal status.

RtCreateThread() is the function for creating a thread and it receives the thread handler function and the initial status of the thread as arguments. StartHandler, the third argument, denotes the name of the handler function and the fifth argument, CREATE_SUSPENDED, means that as soon as the thread is created, the handler

function is suspended. The priority of the generated thread is specified by RtSetThreadPriority(). As arguments, RtSetThreadPriority() receives the handler of the function to which the priority is assigned and the priority that will be assigned. In the example, hStartHandler denotes the handler of the function to which the priority is assigned and STARTPRIORITY means the priority. After assigning the priority to the handler function, the thread begins.

The following is the thread handler for handling the cycle start instruction. The following function is defined by RtCreateThread(). As RtWaitForSingleObject() in the code is the function to wait for that particular event object to be turned true (or on), it plays the role of temporarily stopping execution of the function. RtWaitForSingleObject() receives the handler of the awaited event and the awaited time as arguments. In the code, hStartEvent denotes the handler of the awaited event and INFINITE means that RtWatForSingleObject() waits indefinitely for the event until the signal of the event is changed to true (or on).

```
// ********************************************************** //
// Description: Handling "Cycle start" thread. //
// ********************************************************** //
// Function to handle 'cycle start' event.
ULONG RTFCNDCL StartHandler(void * nContext)
{
    DWORD dwEventReturn;
     while(1) {
        // Wait until Cycle start button is pushed.
        dwEventReturn = RtWaitForSingleObject(hStartEvent, INFINITE);
         Sim_total_count = 0;
         // Start NCK.
        StartNCK();
    }
     return(0);
}
```

The following shows how other processes call the event function, which is included in the MMI module. For calling the event handler, first, the event handler should be taken and the status of the event object should be changed. RtOpenEvent() plays the role of opening the handler of a particular event and receives as arguments the access method to the event object and whether the returned handler can be inherited. In addition, it also receives the name of an event object as argument. After taking and using the event object handler, the handler should be closed by RtCloseHandle(). After RtSetEvent() is called, the above thread function is resumed.

```
// ********************************************************** //
// Description: Open the created Event object. //
// ********************************************************** //
hStartEvent = RtOpenEvent(EVENT_MODIFY_STATE, TRUE,
```

```
                                    "NCKSTARTEVENT");
RtSetEvent(hStartEvent);
RtCloseHandle(hStartEvent);
```

10.5 Open-CNC Systems

After development by MIT in the early 1950s, CNC systems have advanced with the appearance and advancement of the microprocessor. With the introduction of automation systems in the 1970s, the function of CNC systems has made rapid progress. However, due to the complexity of NC technology, which requires not only fundamental control function but also various auxiliary technologies such as machining technology, process planning technology, and manufacturing technology, the market for NC systems has been dominated by a few market leaders in Japan and Germany. The advanced manufacturers evolved CNC systems into closed systems in order to prevent their own technology from leaking out and keeping their market share.

However, after the middle of the 1980s, a new manufacturing paradigm, where computer network and optimization techniques were applied to manufacturing systems with the progress of computer technology, has appeared together with the requirement for advanced control functions for high-speed and high-accuracy machining. Closed CNC systems were not adequate for realizing the new manufacturing paradigm. The architecture of closed CNC systems could not meet the user's requirements and improvement of CNC systems was possible, not by MTB (Machine Tool Builders) but CNC makers. The limited resources of CNC makers made it impossible to meet the new paradigm.

Therefore, various efforts to develop open CNC systems have been tried. As a typical result of these attempts, PC-NC that was introduced in the early 1990s. Like IBM PC technology, which appeared in the early 1980s, has progressed by third-party developments based on openness, CNC systems have progressed to PC-NC based on the openness of PC technology. However, now, despite low price, openness, and many developers of PC-NC, the lack of reliability and openness to application S/W has made it impossible to implement perfectly open systems.

10.5.1 Closed-type CNC Systems

For better understanding of Open-CNC systems, a conventional CNC system, the Closed CNC system contrasted with the Open-CNC system will be addressed.

In terms of functionality, a CNC system consists of the NCK function, which executes the interpretation of a part program, interpolation, acceleration and deceleration control, position control, and compensation algorithm, the MMI function,

which provides the interface to enable a user to operate a machine, edit a part program, communicate with external systems, and watch the status of the machine, and the PLC function, which controls auxiliary functions such as tool change, spindle control, and input/output signal control. Details of these functions were introduced in previous chapters. The architecture and functionality of a CNC system that is typically used on the shop floor are designed as shown in Fig. 10.18.

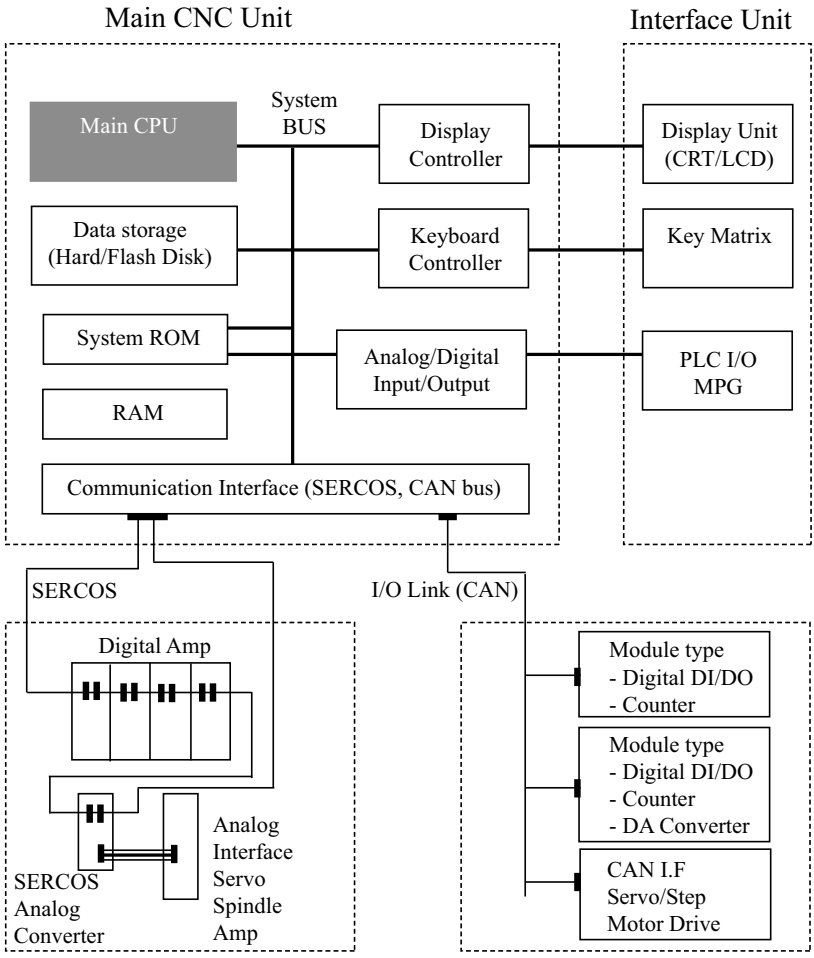


Fig. 10.18 Architecture and functionality of a typical CNC system

A CNC system to execute MMI, NCK, PLC modules has a closed architecture that has no way to communicate to a third party system. Like the architecture of a typical PC system the main CPU unit that carries out each module consists of a main processor, ROM for storing system programs, RAM for storing applications,

and keyboard and display unit for user interface. These are connected using a system bus.

However, unlike a PC, a CNC system has analog/digital input/output devices for communication with the machine and communication interface. In the past, in CNC systems, communication between NC equipment and motors and drives was done by analog signals and the communication interface was very simple. Because of the problem of noise, digital communication has now typically come to be used and SERCOS is a typical digital communication method. With the usage of fiber optical cable, digital communication makes the exchange of various data and the removal of noise possible. Therefore, it is becoming possible that the CNC system adjusts the parameters of servo drives and motors directly and that the CNC system monitors the status of the servo system in real time. Improvement of machining accuracy has become possible due to the removal of noise. In addition to communication with the servo system, digital communication has been applied to communication with input/output devices. In order to enable communication with a variety of sensors and machine components via a single communication line, a standardized communication method is required. For this, various kinds of field bus, such as Profi-bus, CAN Bus and InterBus-S, have been introduced, but one standard method has not yet been established.

Because of the closed nature of the architecture, users and MTBs (Machine Tool Builders) cannot add some functions to the CNC system mentioned above or would have to pay a lot of money to the CNC maker to add them. From the position of the MTBs, though, this closed nature is a weapon for dominating the market. The problems that users raised are summarized in Table 10.7. In order to meet the requirements of users, it is necessary to develop open systems where the functions can be reconfigured and extended and standardized communication protocols, hardware, and interfaces are applied.

10.5.2 Open CNC Systems

To overcome the drawbacks of the closed CNC system, an open CNC system has been developed. In this section, the concept, definition, and architecture of an open CNC system will be addressed.

10.5.2.1 Target of Open-CNC Systems

As mentioned in the previous section, the closed architecture makes it difficult to add sensors for process monitoring or machine control. Nor does it not provide flexibility to adapt the CNC system to a variety of machines and a variety of purposes. In addition, since in the closed architecture the scalability of the function and the standard interface for exchanging the data with other systems are not provided, only one kind

Table 10.7 Requirements for closed CNC systems

Item	Requirement
Reconfigurability	In the case of machining an engine cylinder block of a car, about 80% of machining does not require high precision machining and not and only hole drilling and plane milling are performed. The CNC system used in this machining does not require a variety of functions in the user interface but functions about automation. Therefore, the functionality of the CNC system can be added to or subtracted from according to the user's requirements.
Extensibility	The hardware and software, such as the number of controllable axes, cycle program, and program storage are independent in terms of functionality. They can be reconfigured if needed.
Program	Part programming and macro programming based on EIA are very complicated and each CNC maker provides their own special functions. In the case of using CAD/CAM, there are many problems with exchanging data between software programs and devices. To solve these problems, a new CNC programming language is required.
Advanced function	In the case of milling machining for mold and die, surface interpolation functions for machining free-form surfaces are needed in order to avoid grinding operations for post-milling operations. Sensor-based feedback control for high-precision machining is also needed. Therefore, when it is necessary to apply new technology, the addition of new functions should be possible.
Intelligence	The cutting conditions for machining should be determined depending on the diameter of the tool and the materials of the workpiece and tool. As the selection of cutting conditions requires much know-how, automation of the selection in the CNC system is insufficient and an intelligent CNC system is required for optimal process planning and optimal toolpath generation.
Standardization	Despite the fact that a variety of machines are used together in the field, they cannot communicate with each other unless their CNC equipment is the same. A CNC system has limitations on communication with PC and FA controllers. In addition, the options provided by CNC makers depend on the CNC makers and are expensive. To solve these problems, standardization and openness are required.

of CNC system exists for one kind of machine and it cannot exchange hardware, software, and data with other CNC systems.

Since monitoring of processes and the control tasks based on the monitoring are executed simultaneously, in the CNC system that enables it has been necessary to add new sensor-based algorithms in the control/monitoring module in order to expand the functionality of CNC systems. It is also necessary to create a CNC system where real-time change of algorithms and control architecture is possible, sensed data are shared, and communication with other systems is possible.

The need for open systems has not been raised by CNC makers but by MTBs and users. Together with the requirement of MTBs and users, CNC makers accepted this trend to adopt quickly new technology with low cost. For example, suppose that an MTB/user applies a sensor based on a particular Field Bus to a CNC system or wants to add particular software. In this case, the CNC maker provides an environment to enable a third party to create the functions that are better than those developed by the CNC maker.

An open system is defined as a system that satisfies the following:

1. Interoperability: This means the ability that the components that compose the system cooperate to perform the specified task. For this ability, the standard specifications of the data representation language, behavior model, physical interface, communication mechanism, and interaction mechanism are needed. A bus-based system design is most important.
2. Portability: This means the ability for a component to be executed on the CNC system with different hardware or different software. Portability is very important from the commercial point of view. Since this means that a hardware device or a software module can be used on various platforms, it contributes to increasing the efficiency of a platform.
3. Scalability: This means the ability to make extensions to or reductions of the system's functionality possible without large cost. Adding memory or a board to a PC is a typical example.
4. Interchangeability: This means the ability to replace the existing component with a new component. Instead of replacing the whole system, replacing an existing motion board with a motion board with a new algorithm is a typical example.

As the definition of an open system, modularity, extensibility, reusability, and compatibility can be considered. However, these can seem to belong to the above properties. From another point of view, an open system can be defined as a system with flexibility and standardization. Flexibility, though, has a similar meaning to interoperability and scalability and standardization are similar to portability and interchangeability.

10.5.2.2 Classification of an Open System

The openness of a CNC system should include the openness not only of stand-alone units but also FA systems. Therefore, besides the development of special-purpose

systems by modularization of system components and standardization of platforms and interfaces between modules, it is necessary to make integration between the devices made by various vendors easy with optimized integration of existing components. The development of an open system has progressed as shown in Fig. 10.19. As the first step, a system with open environment controller, with each piece of equipment connected to another via an open communication network. As the second step, a system with open environment common interface controller, portability, interchangeability, and scalability can be realized. As the last step, a system with open modular architecture controller, a variety of applications based on distributed network technology and component technology can be implemented.

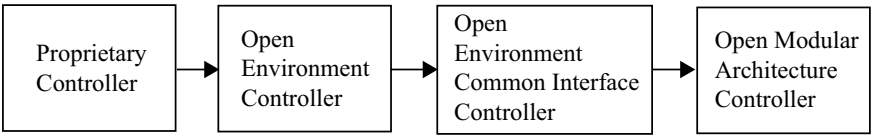


Fig. 10.19 Progress of Open System development

Until the late 1990s much effort for establishing the open environment had been made. From the early 2000s the development of an open CNC system has progressed.

In the terms of the system architecture, an open CNC system can be classified into three types, as shown in Fig. 10.20. MMI and NCK including the PLC unit exchange data via a particular communication module. As Type 1, shown on the left, is Open MMI, NCK is closed and only allow data exchange with MMI. In Type 1, MMI is divided into a basic area and a special area. In the basic area, the fundamental functions of MMI are located. In the special area, the special functions (*e.g.*, conversational programming system, CAD/CAM, production management system, and tool database) are located. This type makes it possible for the MTB and user to develop and use the user-specified MMI. The majority of commercial CNC systems have been developed based on this architecture.

Type 2 denotes the architecture where new functions can be added to NCK as well as MMI. By modularizing the core functions of NCK, adaptation of new algorithms is possible. However, since the openness is not reflected in the design of the whole system architecture, the problem of the interface between modules and their compatibility results when this architecture is applied in practice.

Type 3, the rightmost in Fig. 10.20, denotes the architecture where system software and application software in MMI and NCK are modularized and the communication interface between them is standardized. This makes extensibility and compatibility possible.

As Type 1 and Type 2 are semi-open architectures compared with Type 3, they are practical considering that they make it possible to add new functionality only with partial modification of a conventional closed architecture. However, CNC systems of Type 1 or Type 2 cannot change or modify a part of hardware or software without knowledge of the interface and dependency of each module. In contrast, Type 3

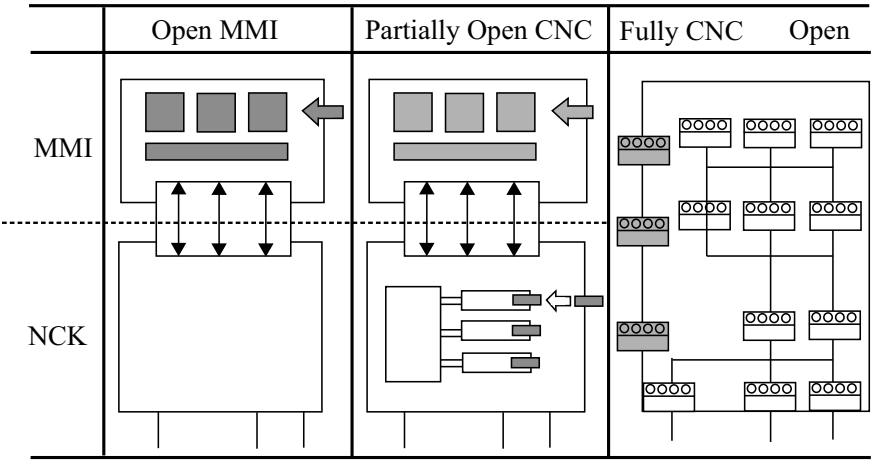


Fig. 10.20 Open CNC System classification

CNC systems can modify or replace modules independently owing to the full open architecture.

10.6 Summary

PC-NC, being the alternative for replacing a closed and expensive hardware-based NC, is distinguished from the traditional hardware-based NC in terms of hardware architecture, software model, and communication mechanism. The configuration of PC-NC is classified into the following three types:

1. PC is used for MMI and the closed-type motion control boards for NCK and PLC are inserted into the PC.
2. Two PCs are used for MMI and NCK/PLC, respectively. The two PCs are connected via high-speed communication.
3. MMI, NCK, and PLC are implemented as software tasks operating in a multiprocessing environment using a single CPU.

In particular, the third configuration, where a single CPU is used and CNC functions are implemented in software, is called Soft-NC. In Soft-NC, NCK, PLC, and MMI are regarded as individual tasks and are executed by the scheduler of a real-time OS. In this configuration, the volume of hardware is reduced. Unlike the closed-type NC and other types of PC-NC, it is easy to add user requirements, modify functions, and link to other applications by utilizing various functions from real-time OS.

In terms of Soft-NC design, a variety of modules are regarded as tasks and they can be divided into non-cyclic tasks and cyclic tasks. A non-cyclic task is a task that

does not require tight response time, such as MMC, interpreter, and external communication manager. A cyclic task is a task that requires the hard real-time property. For developing Soft-NC, it is necessary to design a scheduler, synchronization between tasks, and communication between tasks. Moreover, this design is realized by real-time programming techniques.

Chapter 11
STEP-NC System

With the rapid advancement of information technology associated with NC technology, the manufacturing environment has changed significantly since the last decade. However, the low-level standard, G&M codes, have for over 50 years been used as the interface between CAM and CNC, and are now considered as an obstacle for global, collaborative and intelligent manufacturing. A new model of data transfer between CAD/CAM systems and CNC machines, known as STEP-NC, is being developed worldwide to replace G&M codes. In this chapter, we will give an overview of STEP-NC and its related technology, including data models for STEP-NC, CNC systems based on STEP-NC, namely STEP-compliant CNC systems, together with worldwide research status and future prospects.

11.1 Introduction

A designer, "A" makes a 3D design by ANYCAD system in Korea. A CAM specialist "B" in the USA generates a process plan by ANYCAM system for manufacturing the design transmitted through the internet. Operator "C" in South Africa downloads the design and its process plan and executes the ANYCNCFRONT simulator, followed by machining with ANYMACHINE controlled by INTELLIGENT CNC as shown in Fig. 11.1. When the operator clicks on "Cycle Start" button, machining is started. During the machining operation, the INTELLIGENT CNC controls the machining operation precisely in an optimized fashion (feedrate optimization) with adaptive capability in handing unexpected situations such as tool wear/breakage/unavailability. After machining, INTELLIGENT CNC reports to the stakeholders in the worldwide chain the machined results including the accuracy of the machine part compared with the geometry and tolerance information of ANYCAD measured by the on-machine inspection module of INTELLIGENT CNC. Except for the machining time, the whole information transaction period is in the order of a few minutes.

This scenario is not fiction, but can be achievable by using STEP and STEP-NC. STEP-NC aims at providing an information bus for manufacturing running in the

world via the internet and that the various stakeholders of CAD, CAM, CNC require for realizing seamless DA-BA-SA (Design-Anywhere, Build-Anywhere, Support-Anywhere), which has become the catch phrase of e-Manufacturing. Also, STEP-NC aims at realizing intelligent CNC having optimization and adaptability capability. Toward these aims, efforts for standardization, research and development are in progress worldwide.

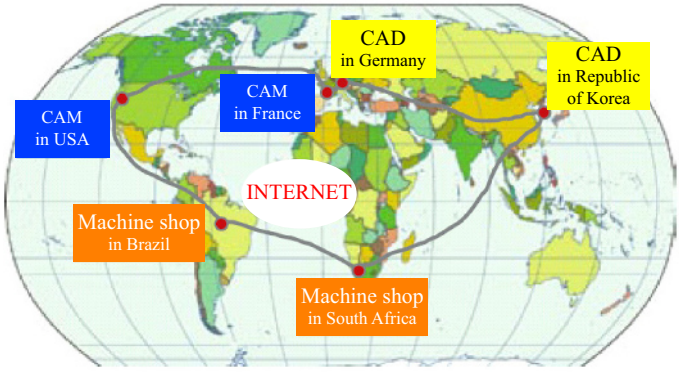


Fig. 11.1 e-Manufacturing for DA-BA-SA

As STEP-NC has been approached by many affiliations, institutions, companies from a variety of perspectives, there are several jargon expressions, or terminology related to STEP-NC. Thus, before getting into detail, we clarify the following terminologies. Note that the definitions below are from the best knowledge of the authors working in this area for a long time, but other people might have different meanings for these.

- **STEP-NC:** Two meanings are used in a narrow and broad sense. In a narrow sense, STEP-NC means the new interface language between CAM and CNC. In a broad sense, STEP-NC includes not only the new interface language but also technologies to implement CAD/CAM and CNC software or products based on the new interface. In this book, STEP-NC is used to mean the new interface language; *i.e.*, the narrow meaning of STEP-NC.
- **ISO 14649:** ISO 14649 is an international standard specification defining the data model for STEP-NC. It specifies information contents and semantics (ICS) for various CNC manufacturing processes and resources including cutting tools and machine tools. The contents will be explained in Section 11.4.
- **STEP-NC data model:** STEP-NC data model means the contents of ISO 14649. In other words, it is the same as the narrow meaning of STEP-NC.
- **STEP compliant CNC:** STEP-compliant CNC means a kind of new CNC controller implementing STEP-NC, *i.e.* taking STEP-NC as input and controlling the machine tool motion. Depending on how the STEP-NC is interfaced and used,

STEP-compliant CNC is classified into 3 types: 1) Type 1; Conventional, 2) Type 2; Basic, and 3) Type 3; Intelligent. The details will be discussed in Section 11.5.
- **STEP-CNC:** STEP-CNC is an abbreviation of STEP-compliant CNC.
- **STEP-NC technology:** STEP-NC technology means various technologies required for implementing software and products based on the STEP-NC interface.

The remainder of this chapter is organized as follows. In Section 11.2, we will introduce problems of current G&M codes and the historical background to STEP-NC. In Section 11.3 we will give an overview of STEP-NC, including information contents, structures, objectives and impacts. In Section 11.4 we will explain details of the STEP-NC data model, followed by interpretation and part programming in Section 11.5. In Section 11.6, technologies for implementing STEP-compliant CNC are illustrated and the world wide research and development status together with future prospects of STEP-NC and STEP-NC technologies are given in Section 11.7.

11.2 Background of STEP-NC

11.2.1 Problems with G&M Codes

The manufacturing environment has been changing, with more collaboration and intelligence since the 1990s. High-speed machining, high-precision machining and multi-axis complex machining have extensively enhanced the productivity and quality of manufacturing. Furthermore, advanced internet technology has introduced a new paradigm of e-Manufacturing so that DA-BA-SA can be realized via the collaborative scheme of a distributed manufacturing system. Toward this goal, however, the machine language, the so-called G&M codes (formalized as ISO 6983 by ISO or similarly RS274 in USA, DIN 66025 in Germany) used for CNC since the invention of NC technology in the early 1950s is the major bottleneck in information-based manufacturing systems. Because ISO 6983 was developed at a time when computer power was limited and machines were controlled offline the needs and possibilities, then, were very different from those of today. These machines used simple instructions to move tools through the air and for cutting metal. In particular, the problems of G&M codes have been reported as follows:

- **Information loss**
  A G&M-code part program is defined by simple alphabetical or numerical codes such as G, T, M, F, S indicating the movement of a machine and an axis to the controller. Since this delivers only limited information to the CNC (excluding valuable information such as part geometry and the process plan used to generate the NC code), it makes the CNC simply an executing mechanism, completely unaware of the motions being executed.

- **Difficult traceability**
  As a G&M-code part program is made up of a coded set of numbers for axis movements, it is not easy for machine operators to understand the operational flow, machining condition and specification of tools only by reading the low-level part program. In particular, it makes it more difficult, not only in finding which part happens to cause problems, but also modifying the program for solving these problems.
- **Lack of interoperability**
  The G&M code schema is dependent on the machine tool builder or controller maker. For example, G70s and G80s mean cycle codes, G98 orders feed per minute in the FANUC 0 series. On the other hand, G60s and G80s mean cycle codes, G94 instructs feed per minute in the FAGOR 8055T series. The part program for a certain targeted controller cannot be applied to another heterogeneous controller. For this reason, it is necessary to apply a post-processing process in which the program is adapted to a specific CNC machine configuration. This post-processing is one of the main interferences with the seamless data flow in the CAD-CAM-CNC chain.
- **Non-compatibility with higher level systems**
  In higher-level manufacturing systems, such as office level CAD/ CAPP/ CAE/ CAM/ PDM/ MRP, compatible information exchange is being increased gradually by the introduction of STEP. In contrast, the rich information environment has almost perished at the CNC on the shop floor level. Also, there is little information feedback from the CNC, which makes the shop floor status obscure to the upper systems. Inevitably, the CNC on the shop floor remains an isolated island in the CAD-CAM-CNC chain.

11.2.2 Historical Background

The initial effort on the new CNC data model was made by WZL of Aachen University between 1994 and 1996 in the European Project OPTIMAL (ESPRIT III 8643). In this project, the data model for 3D milling was investigated based on the STEP paradigm, in which STEP data was first used as the basis of the interface scheme between CAM and CNC. This STEP-based interface scheme was extended to 2.5D milling and other operations like turning and EDM in the subsequent European Project ESPRIT IV 29708 - STEP-NC (STEP-compliant data interface for Numerical Controls) between 1999 and 2001. The USA agreed with this feasible research from Europe and actively joined this project from 1999. The STEP-NC project has gained worldwide consensus, and was promoted to international IMS (Intelligent Manufacturing Systems) status project 97006, composed of Europe, USA, Korea and Switzerland from 2002. Recently, research and development for commercialization have been actively and collaboratively contributed by a large number of nations including Brazil, Canada, China, France, Germany, Japan, New Zealand, Pakistan,

South Korea, Switzerland, UK, USA and so on (alphabetical order). More details of their research results will be discussed in Section 11.7.

11.3 STEP-NC: A New CNC Interface Based on STEP

STEP-NC is expected to encompass the whole scope of e-Manufacturing. The new STEP-NC data model has been developed for the replacement of the old standard G&M codes for milling, turning and EDM, and development on implementations is under way. Now that the new data model has been established, development and implementation of STEP-compliant CAD/CAM/ CNC systems based on the new data model is drawing worldwide attention.

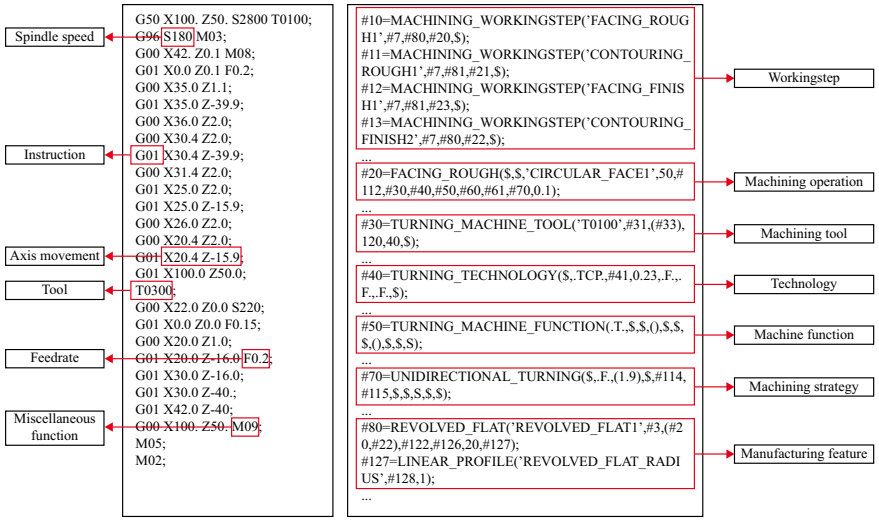
11.3.1 Contents

STEP-NC is a new model of data transfer between CAD/CAM systems and CNC machines. As shown in Fig. 11.2a, G-code contains just axis movement, spindle speed, feedrate, tool position in the tool changer and coolant. With this information, it is very difficult for machine operators to understand the operational flow, machining conditions and specification of tools only by reading a part program. Also, it is impossible for the CNC controller to execute an autonomous and intelligent control and to cope with emergency cases with this limited information. In contrast, STEP-NC contains the required functional information, as shown in Fig. 11.2b, such as workingstep, machining feature, machining operation, machining tool, machining strategy, machine function and workpiece. In other words, STEP-NC includes a much richer information set including 'what-to-make' (geometry) and 'how-to-make' (process plan).

11.3.2 Relationship Between STEP and STEP-NC

STEP-NC adopts the definition of STEP or modifies it according to CNC. Figure 11.3 depicts the relationship between STEP (ISO 10303) and STEP-NC (ISO 14649). 2.5D machining features in ISO 14649 use those from ISO 10303 AP 224 and the free-form surface information references from the 3D design information of ISO 10303 AP 203. The expressive language of data model schema applies ISO 10303 Part 11 (EXPRESS) and the implementation method for the file format complies with ISO 10303 Part 21 (clear text encoding rule) and ISO 10303 Part 28 for XML.

STEP-NC is under development by ISO TC 184 (Technical Committee in International Standardization Organization) SC1 and SC4 (Sub-Committees 1 and 4). Strictly speaking, there are two versions of the STEP-NC data model. The first is the



(a) G-code part program (b) STEP-NC part program

Fig. 11.2 Comparison of G-code part program and STEP-NC part program

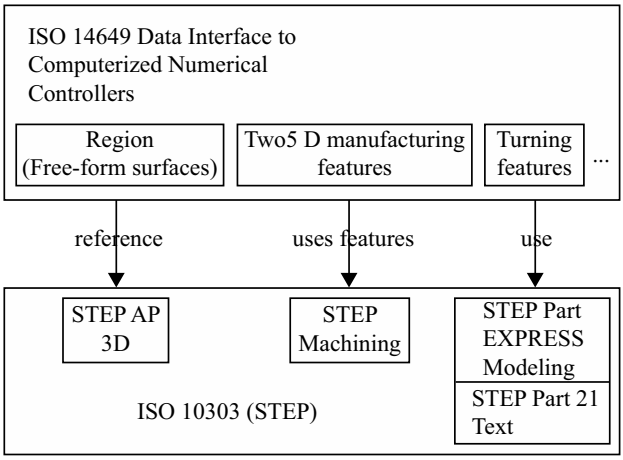


Fig. 11.3 The relationship between ISO 14649 and ISO 10303

Application Reference Model (ARM), *i.e.* ISO 14649, which describes machining processes in terms of domain terminology. The second is the Application Interpreted Model (AIM), *i.e.* ISO 10303 AP (Application Protocol) 238, which maps the ARM to the existing integrated resources of the STEP (STandards for Exchange of Product data model formalized as ISO 10303). In this text, we refer to ISO 14649 as the data model for STEP-NC.

Recently, STEP-NC has been harmonized with the STEP set of standards as ISO 10303 AP238, which offers the possibility of seamless integration of applica-

tion throughout design and manufacturing. This effort is for realizing information-based manufacturing, namely the suite of STEP-Manufacturing recommended by ISO TC184 SC4 and SC1. The suite shows the functions and associated information standards in terms of STEP APs, as shown in Fig. 11.4. The input to the STEP-NC system is AP203 and/or AP224, and the output is STEP-NC codes. Functions and information within the STEP-CNC system are from macro-process-planning based on AP240, followed by micro-process-planning (ISO 14649).

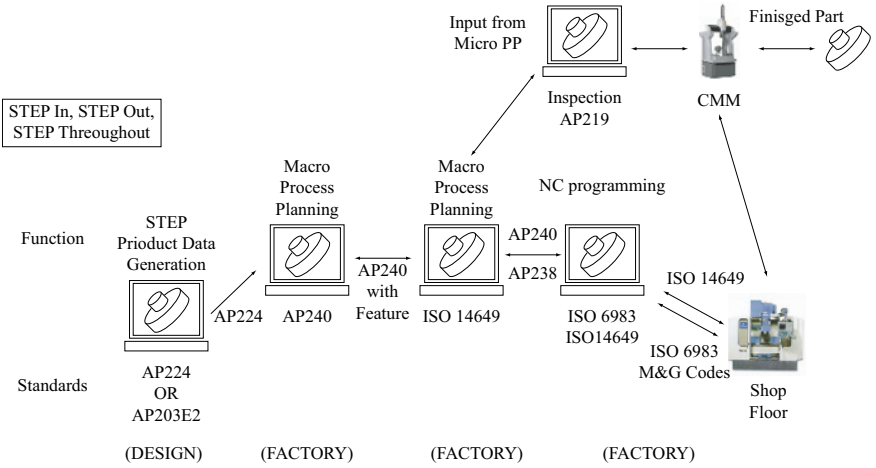


Fig. 11.4 Suite of STEP-Manufacturing

11.3.3 Objectives and Impacts

According to Part 1 of ISO 14649, the objectives of the STEP-NC data model are as follows:

- To cover the current and expected future needs for data exchange
- To support the direct use of computer-generated product data from ISO 10303
- To create an exchangeable, workpiece-oriented data model for CNC machine tools
- To use standard, modern languages and libraries for the implementation of the data model
- To ensure compatibility of CNC input data

The impact of the new interface scheme is most felt in the CAD-CAM-CNC chain as follows:

- In the sense that the STEP-NC data model is an extended product data model including process plan information, it can be used as an information highway encompassing CAD, CAPP, CAM, and down to CNC.
- At the CAD-to-CAM level between design and manufacturing, the NC extension parallels STEP's overall ability to facilitate seamless data flow in B2B cases.
- Because the 3D model can be sent directly to manufacturing, a time saving of 75% can be achieved, according to an analysis by Hardwick, as shown in Fig. 11.5.
- Further, because the new data model defines all the information for process planning, the process planning step can be greatly simplified, saving from 35% to 60% from the time normally required for the step (Fig. 11.5).
- From the perspective of CNC, the new data model is very significant, providing CNC with all the information about what to make and how to make it. According to a survey, a time saving of 50% is reported (Fig. 11.5).
- Currently, for machining with G-code, a post-processor is required to convert the process plan from product space to machine space, as shown in Fig. 11.6a. However, with the wide adoption of STEP-NC the requirement for these post-processors will be eliminated, as illustrated in Fig. 11.6b.
- Depending on the implementation, there can be many other tangible benefits that cannot be measured by time, such as machining accuracy, quality improvement, automatic part setup, on-machine inspection, automatic collision avoidance, among others.

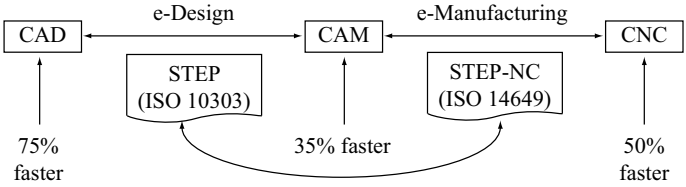
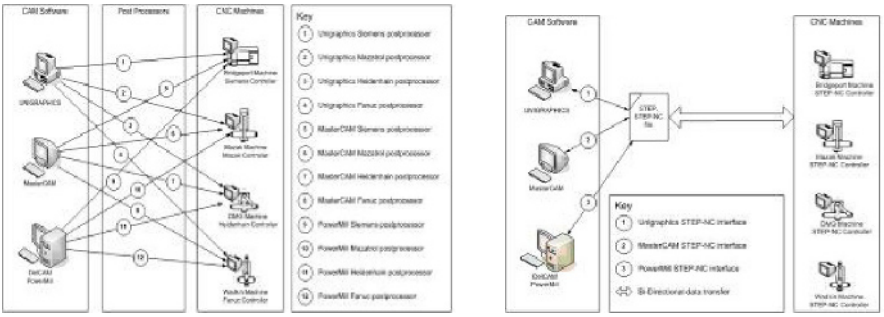


Fig. 11.5 Time reduction effects in the CAD-CAM-CNC chain of STEP-NC

11.4 STEP-NC Data Model

Since ISO 14649 was published as a CD (Committee Draft) version in 1997, it is now almost in IS (International Standard) version except for Part 111, which is the FDIS version. Table 11.1 shows the current status of ISO 14649 parts including future work items that are either in WD (Working Draft) or CD versions. Part 1 provides an introduction and overview of a data model for CNC including the advantages. Part 10 specifies the process data that is generally needed for NC-programming within all machining technologies and other 1x Parts technology-specific process data such as



(a) One-to-one G-code post processor (b) one-to-many STEP-NC interface

Fig. 11.6 ISO Three levels of ISO 14649 data model

milling, turning, wire EDM, sink EDM, contour cutting, inspection and rapid prototyping. Part 110 describes the requirement model for the machine tool executing the STEP-NC part program, which is currently NWIP (New Work Item Proposal) as of September 2007. Part 1xx describes tools for each process such as Part 111 for cutting tools for milling and Part 121 for cutting tools for turning.

Table 11.1 ISO Current status of Parts in the ISO 14649 (December, 2007)

ISO 14649	Title of documents	Edition	Status
Part 1	Overview & fundamental principles	1	IS
Part 10	General process data	1	IS
Part 11	Process data for milling	1	IS
Part 12	Process data for turning	1	IS
Part 13	Process data for wire-EDM	2	CD
Part 14	Process data for sink-EDM	2	CD
Part 15	Contour cutting	2	WD
Part 16	Process data for inspection	2	WD
Part 17	Process Data for rapid prototyping	2	WD
Part 110	Machine tools for general process	2	NWIP
Part 111	Tools for milling machines	1	FDIS
Part 121	Tools for turning machines	1	IS

IS : International Standard, CD : Committee Draft,
WD : Working Draft, NWIP : New Work Item Proposal,
FDIS: Final Draft International Standard

11.4.1 Part 1: Overview and Fundamental Principles

Part 1 describes an introduction and overview of a data model for CNC, AAM (Application Activity Model), and usage scenarios. It describes the fact that the ISO

14649 data model is composed of three levels as shown in Fig. 11.7. Level A deals with the modeling of the manufacturing technologies in the Application Reference Models (ARMs) with a precise description in EXPRESS schemas. Level A is the responsibility of ISO/TC184/SC 1/WG 7. Level B deals with integration and compatibility with ISO 10303, based on the Application Interpreted Models (AIMs) that map the ARMs to the set of ISO 10303 integrated resources. Level B is the responsibility of ISO/TC 184/SC 4. Level B covers also the data exchange and compatibility needs. Based on actual STEP standards, different data formats can be used in the databases and to transfer exchangeable data to the CNC controllers, such as ISO 10303-21, ISO 10303 SDAI database and the most actual and advanced ISO 10303 data server with EXPRESS-X queries and data formatted in XML (ISO 10303-28). Level C deals with adoption software, which is the implementation of Level A or B in controllers. CNC manufacturers or third parties are responsible for implementing Level C.

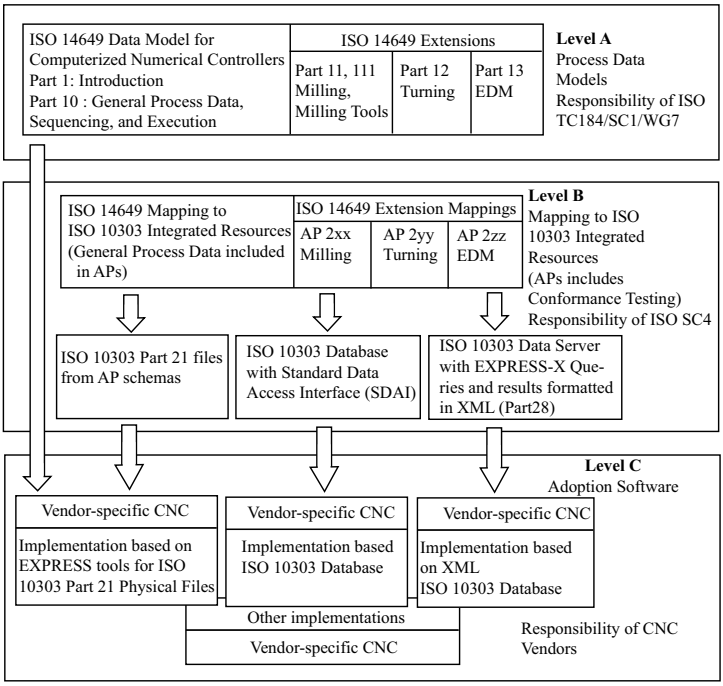


Fig. 11.7 ISO three levels of ISO 14649 data model

The most important feature of the ISO 14649 data model is to remedy the shortcomings of ISO 6983 by specifying machining processes rather than machine tool motion, using the object-oriented concept of the *workingstep*. *Workingsteps* are the essential building blocks of manufacturing tasks. Each *workingstep* describes a single manufacturing operation using one cutting tool. Figure 11.8 shows the overall

structure of ISO 14649 from the viewpoint of the *workingstep*. The order of execution of manufacturing operations is given by the order of *workingstep*s that a *workplan* has. The project entity serves as a starting point for executing the part program. The *workingstep* describes what (*machining_feature*) is machined and how (*machining_operation*) to remove with (*machining_tools*) which tools. *Machining_operation* includes *machining_tool*, *machine_functions*, *machining_strategy* and other process data.

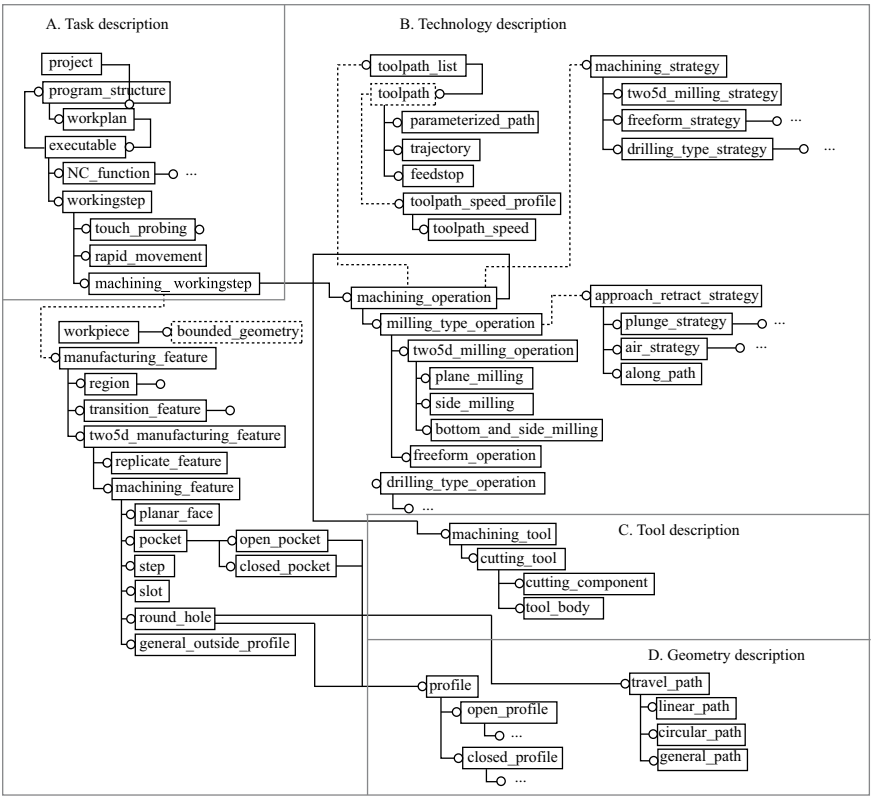


Fig. 11.8 Overall architecture of ISO 14649 ARM

11.4.2 Part 10: General Process Data

Part 10 specifies the execution sequence of the part program and the general process data for NC-programming. The sequence of execution of the part program is repre-

sented by *program_structure*, and the general process data by *manufacturing_feature* and *machining_operation*, respectively.

Figure 11.9 shows the information contents for the sequence of the part program. *Program_structure*s are used to build logical blocks for structured programming of the manufacturing operation. The *program_structure*s, not the list of manufacturing features, have authority over the actual manufacturing sequence. Non-linear process planning, one of the most powerful advantages of ISO 14649, is possible by using *non-sequential*, *parallel*, *selective* entities. The most important structure is the workplan that provides a linear sequence of *executable*s. The *workingstep*s represent the essential building blocks of an ISO 14649 part program. They can either be technology-independent actions, like *rapid_movement*s or *probing_operation*s, or *machining_workingstep*s that describe manufacturing or handling operations that involve interpolating axes. *NC-function* describes manufacturing or handling operations that do not involve interpolation of axes, for example, switching operations or other singular-event machine functionality.

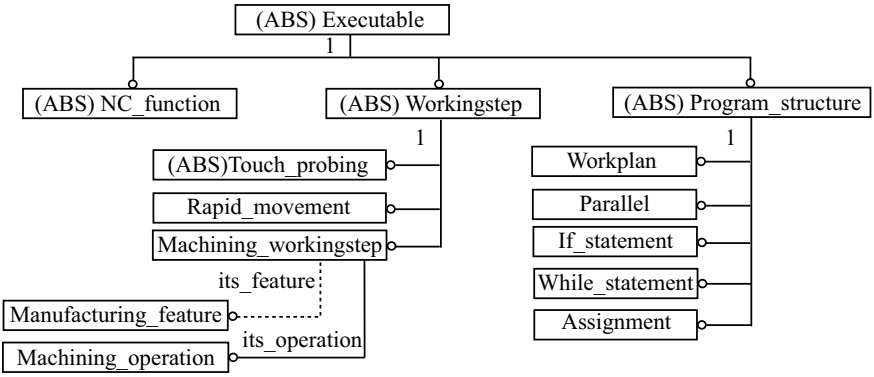


Fig. 11.9 EXPRESS-G diagram for executables

Related to the general process data, the major entities defined in Part 10 are *manufacturing_feature*, which defines 2.5D and 3D machining features, machining_operation that defines the types of the operations, *machining_strategy* that defines various strategies, technology that defines feedrate and spindle, *machine_function*s that defines coolant, chip removal, and *tool_path* that defines pre-defined tool paths. Part 10 has the common operational information for all machining, while detailed information for each manufacturing type, such as milling or turning, are defined in each Part of ISO 14649. For example, Fig. 11.10 shows the structure of *manufacturing_feature*. *Manufacturing_feature* has four subtypes and each subtype has its own subtypes. In detail, *Two5D_manufacturing_feature* has two subtypes, *machining_feature* and *replicate_feature* and *machining_feature* has many subtypes for milling features such as *step*, *slot*, *pocket*, *hole*, and so on.

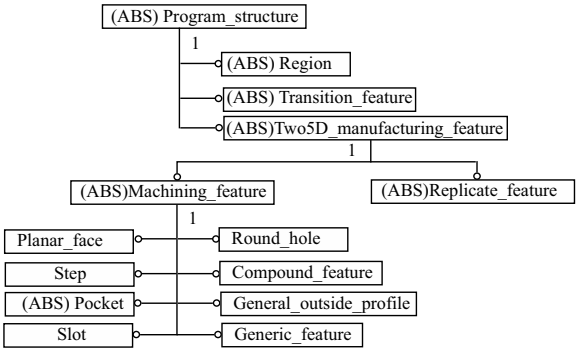


Fig. 11.10 EXPRESS-G diagram for manufacturing features

11.4.3 Part 11: Process Data for Milling

Part 11 specifies the technology-specific data elements needed as process data for milling. It can be used for milling operations on all types of machines, such as milling machines, machining centers, and lathes with motorized tools capable of milling (*i.e.* complex machine tools). Part 11 describes the technology-specific data types representing the machining process for milling and drilling. Figure 11.11 shows the structure of the milling_machining_operation. It is a subtype of machining_operation, defined in Part 10 and has three subtypes: freeform_operation, two5d_milling_operation, and the drilling_type_operation. Each of these has its own associated machining strategy. Examples include freeform_strategy, two5d_milling_strategy or drilling_type_machining_strategy, respectively, as shown in Fig. 11.12. In addition, Part 11 also defines elements such as approach_retract_strategy at the beginning or end of the operation, milling_machine_functions for milling, and milling_technology for milling.

11.4.4 Part 12: Process Data for Turning

Part 12 defines the technology-specific data elements needed as process data for turning. The relationship between Part 10 and Part 12 is similar to that of Part 10 and Part 11. The difference is that turning_feature is defined in Part 12. As turning features are different from milling features and can be machined only in turning machines, they are defined in Part 12. Figure 11.13 shows the structure of the turning_feature and Fig. 11.14 shows the operations defined in Part 12.

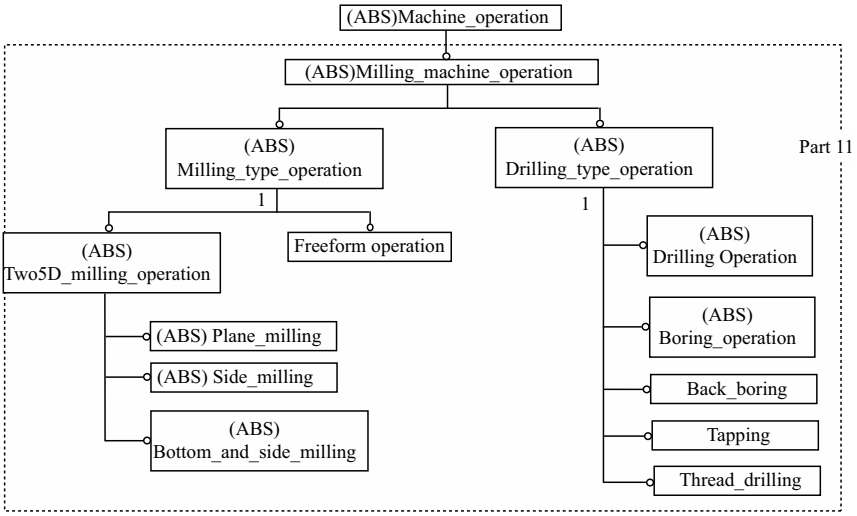


Fig. 11.11 EXPRESS-G diagram for machining operation in Part 11

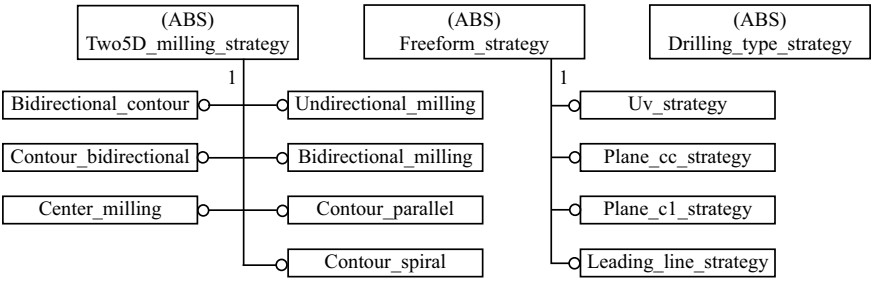


Fig. 11.12 EXPRESS-G diagram for machining strategy in Part 11

11.4.5 Tools for Milling and Turning

This section deals with Part 111: "Tools for milling machines" and Part 121: "Tools for turning machines".

Part 111 and Part 121 define data elements describing cutting tool data for milling machine tools and machining centers and for turning machine tools, respectively. In ISO 6983, the tool is defined by its identifier (*e.g.* T8) and no further information concerning the tool type or geometry is given. This information is part of the tool setup sheet, which is supplied with the NC-program to the machine. However, ISO 14649 includes this information in the part program, such as tool identifier; tool type; tool geometry; application-dependent expected tool life. These data elements can be used as criteria to select one of several operations; they do not describe complete information of a particular tool. Therefore, leaving out optional attributes gives the controller more freedom to select from a larger set of tools. Part 10 defines machining_tool as

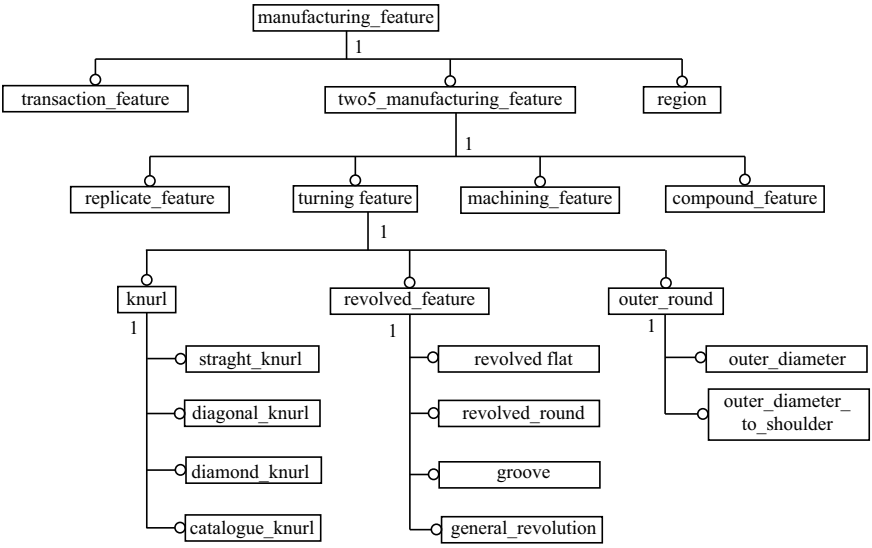


Fig. 11.13 EXPRESS-G diagram for turning_feature

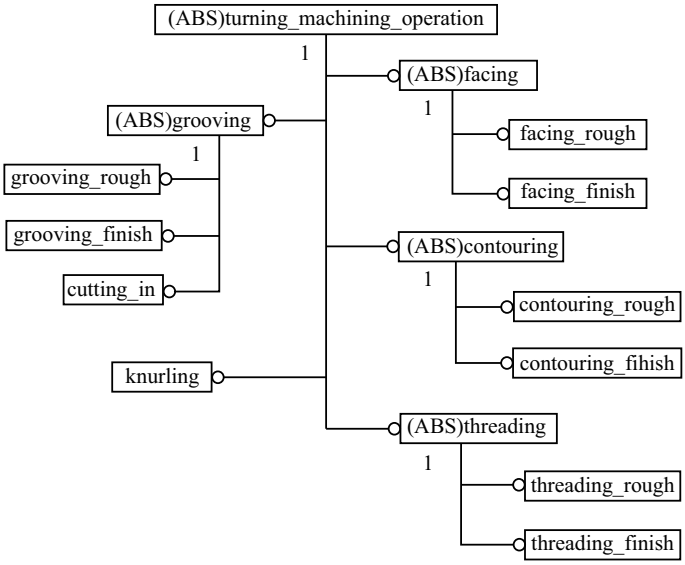


Fig. 11.14 EXPRESS-G diagram for turning_machining_operation

a supertype of milling_machine_cutting_tools and turning_machine_cutting_tools that are defined in Part 111 and Part 121 respectively. Figures 11.15 and 11.16 show the structure of the milling_machine_cutting_tool and turning_machine_cutting_tool elements.

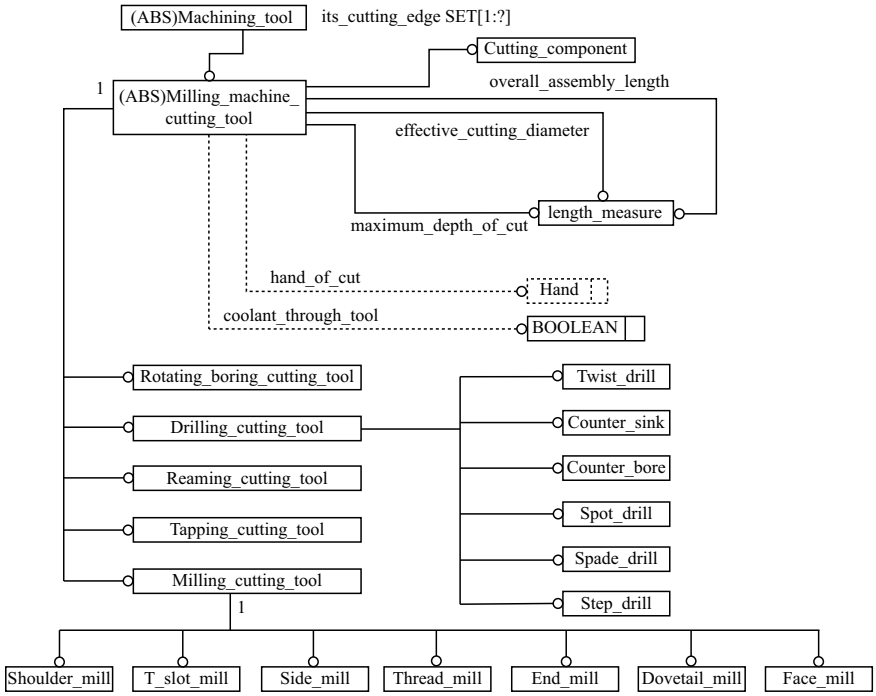


Fig. 11.15 EXPRESS-G diagram for milling machine cutting tool

11.5 Part Programming

Based on the data model, the STEP-NC part program is represented as a physical file according to ISO 10303 Part 21: Clear Text Encoding Rule. As shown in Fig. 11.18, the STEP-NC part program is divided into the header section and the data section. The header section includes information with regard to the part program itself, such as the author information, schema information and version of the part program. The data section includes all the information about the manufacturing such as process sequence, manufacturing feature, operation type, machining strategy, machining technology, machine function, workpiece and geometry. In this subsection, STEP-NC part programs for milling and turning will be described.

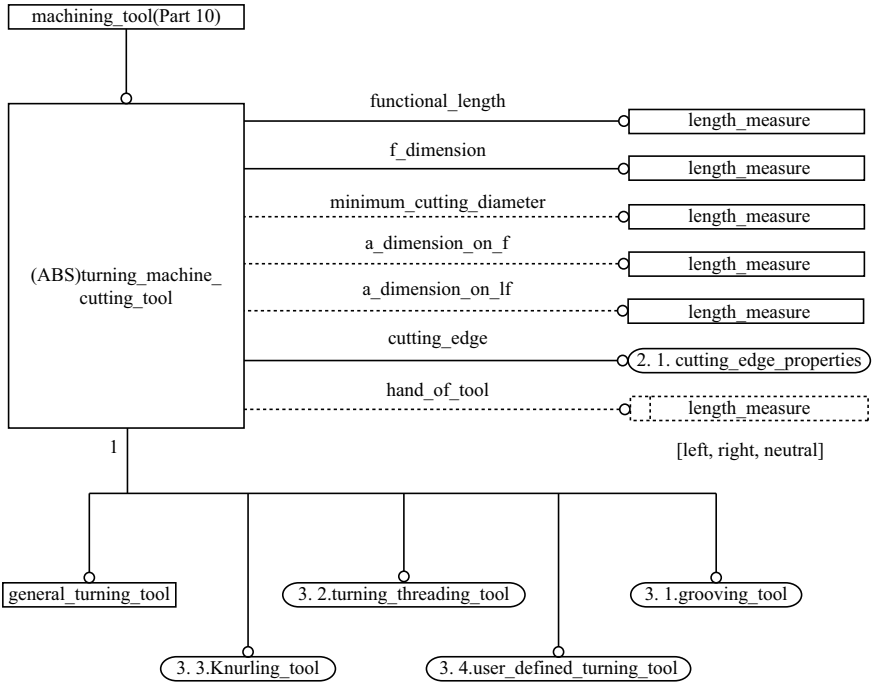


Fig. 11.16 EXPRESS-G diagram for turning machine cutting tool

11.5.1 Part Programming for the Milling Operation

Figure 11.17 shows a simple example for milling, described in Annex E of ISO 14649 Part 11. Figure 11.18 shows the overall structure of the STEP-NC part program for the test part of Fig. 11.17. Note that the part program of Fig. 11.18 is just a fraction of the whole program in order to reduce space. For the full version of this part program, please refer to Annex E of ISO 14649 Part 11.

The shape of Fig. 11.17 includes a plane at the top face (*planar_face*), a rectangular pocket (*closed_pocket*) and a hole (*round_hole*). In this section, machining sequences and detailed information about a rectangular pocket and its machining operation will be explained.

"Sequences" noted in Fig. 11.18 shows information about the machining sequence that is used to machine the test part. Every STEP-NC part program starts with the *project* entity (#1). The main purposes of the *project* are to define the sequence of machining processes by using the *main_workplan* (#2) attribute and to define the workpiece information by using the *workpiece* (#4) attribute, which will be explained later. In this example, five *machining_workingsteps* are executed sequentially. Firstly, the finishing operation for the *planar_face* at the top (#10) is executed, and then the drilling operation (#11) and reaming operation (#12) are executed sequentially

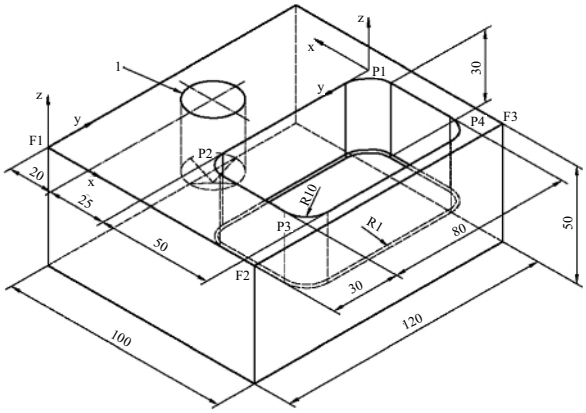


Fig. 11.17 Simple example test part for milling

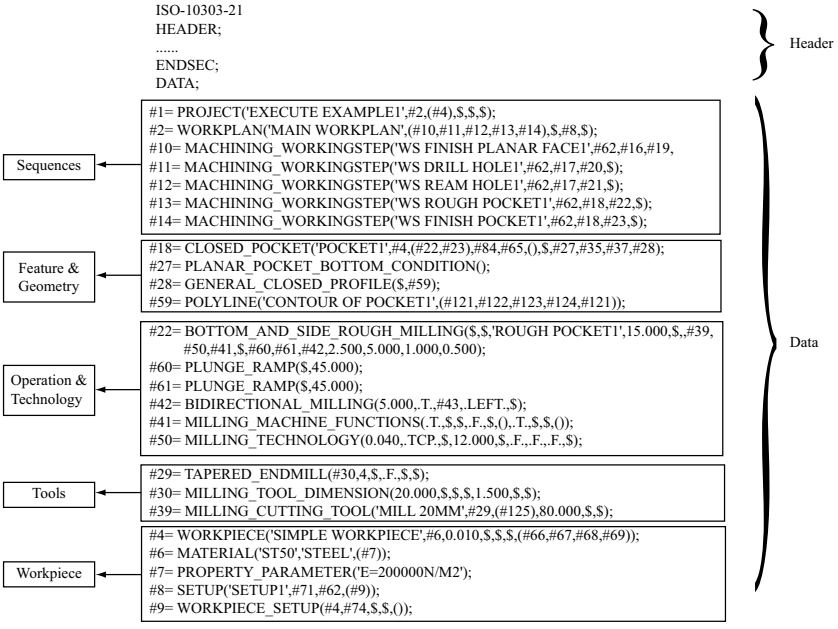


Fig. 11.18 ISO 14649 part program for test part for milling

for the *round_hole*. Finally roughing (#13) and finishing (#14) operations for the *closed_pocket* are executed.

"Feature and geometry" shows feature information in the STEP-NC part program, especially *closed_pocket*. In the part program, the bottom of the pocket is defined as the *planar_pocket_bottom_condition* (#27). The *general_closed_profile* (#28), more especially *polyline* (#59), is used for the contour of the *closed_pocket*.

Table 11.2 Process plan for the closed_pocket

machine parameter	Closed pocket	
	Bottom and side rough milling	Bottom and side finish milling
Tool	Taper End mill 20.0	Taper End mill 6.0
Retract plane	30	30
ADC	4	1
RDC	3	1
Strategy	bidirectional_milling	Contour_bidirectional
Approach	Plunge_zigzag	Plunge_zigzag
Retract	Plunge_ramp	Plunge_ramp
Bottom allowance	1	0
Side allowance	1	0
Feedrate	250	250
Spindle speed	500	500
Coolant	On	on
Chip removal	On	on

Table 11.2 shows the process plan to remove the closed pocket of Fig. 11.17. In this example, the part program for the roughing operation will be explained. Machining type is given by the *bottom_and_side_rough_milling* entity (#22) that has axial depth information (4.0), radial depth information (3.0) and finishing allowance for the wall (1.0) and bottom (1.0), the starting point and the overcut length.

The *machining_strategy* defines the method to execute the given machining operation. The *bidirectional_milling* entity (#42) is used in the process plan of Table 11.2. It defines the direction of the machining, step-over direction and so on. If these values are omitted, the CNC can decide these values autonomously. The *milling_technology* entity (#50) information defines machining conditions such as feed and spindle. Feed can be defined by using *feedrate* or *feedrate_per_tooth* and the speed of the spindle can be defined by using *spindle* or *cut_speed*. Additional information such as the concurrent movement of spindle and feed, the override of the feed and spindle can be defined. In this example, *feed_per_tooth* is used to define feed and *cut_speed* is used to define the cutting speed of the spindle. The *milling_machine_function* entity (#41) defines the activity of the machine tool such as air pressure, coolant, chip removal and so on. In Table 11.2, coolant and chip removal are used during machining. For the *machining_tool*, *taper_endmill* (#29) is used. It defines the diameter (20.0), edge radius (1.5), overall length (80.0) and number of cutting teeth (4).

Information about the raw material of the part is defined by the *workpiece* entity in STEP-NC. In the existing method, G-code, there is no workpiece information. Only the operator knows the workpiece information and decides the cutting conditions by considering that information and generates the G-code. However, STEP-NC supports the initial and final shape of the raw workpiece, material of the workpiece, chucking position of the workpiece and so on. In this example, the material of the workpiece is steel named 'ST-50' and the initial shape of the workpiece is a block whose size is $100.0 \times 120.0 \times 50.0$.

11.5.2 Part Programming for the Turning Operation

Figure 11.19 shows a simple part for turning operation, described in the Annex D of ISO 14649 Part 12. Figure 11.20 shows the overall structure of the STEP-NC part program for the test part. The full version can be found in Annex D of ISO 14649 Part 12.

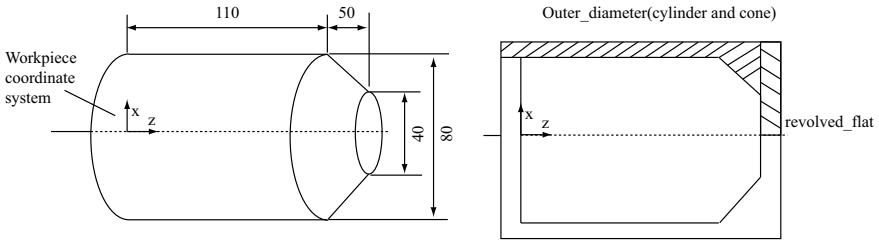


Fig. 11.19 ISO Three levels of ISO 14649 data model

The overall structure of the part program is similar to that for milling operations. The differences are the machining features, machining operations, machining tools that are used in turning. Therefore, turning feature (*outer_diameter*), turning operation (*contouring_rough*) and turning tool (*general_turning_tool*) are explained here briefly.

The shape of Fig. 11.19 includes an end face (*revolved_flat*, #10), a cylinder and a cone (*outer_diameter*, #11 and #12). For the machining cylinder part (*outer_diameter*, #12), the *contouring_rough* (#22) operation is used. For the machining strategy, *unidirectional_turning* (#54) is assigned to execute *contouring_rough* (#22). *Unidirectional_turning* includes length of overcut, depth of cut (3 mm), change amount of feed, lift height (2 mm), feed direction, back path direction, stepover direction and the feed for each direction. For the cutting condition, *turning_technology* (#43) 0.3 mm per revolution is set as feed and 500 RPM is set as the spindle speed in the manner of constant spindle speed. For the machine function, *turning_machine_function* (#40) defines that coolant should be used to carry out contouring_rough. For the cutting tool, *general_turning_tool* (#100) is used and the

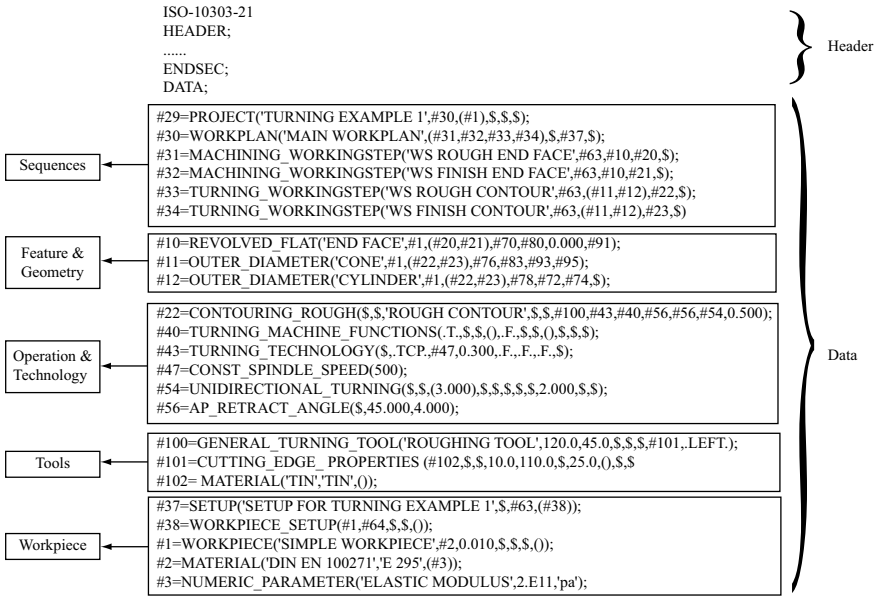


Fig. 11.20 ISO 14649 part program for test part for turning

overall length and width of its holder are 120 mm and 45 mm respectively. Also, *general_turning_tool* uses an insert which has cutting edge length (10.0 mm), side cutting edge angle (110.0°) and end cutting edge angle (25.0°).

11.6 STEP-CNC System

As the new language is established, increasing attention is being paid to the development of a new CNC, STEP-CNC (or STEP-compliant CNC), operating based on ISO 14649. Since the new language accommodates various pieces of information about 'what-to-make' (*i.e.*, product information including 3D geometry) and 'how-to-make' (process plan), STEP-CNC can undertake various intelligent functions that cannot be performed by conventional CNC operation based on ISO 6983. In this subsection, the types of STEP-CNC and their architectures and related technology will be explained.

As shown in Fig. 11.21, STEP-CNC has two types of interface bus, an external bus and an internal bus. The external bus, noted as "STEP based New Programming Language (ISO 14649)" in Fig. 11.21, connects CNC and the CAD/CAPP/CAM system. The information in the STEP-NC part program is interpreted and saved in the database according to its type *e.g.* CAD DB, CAPP DB, and CAM DB. The

internal bus, noted as Soft Bus (CORBA) in Fig. 11.21, makes it possible for the various intelligent modules on the inside of the CNC controller to communicate with each other.

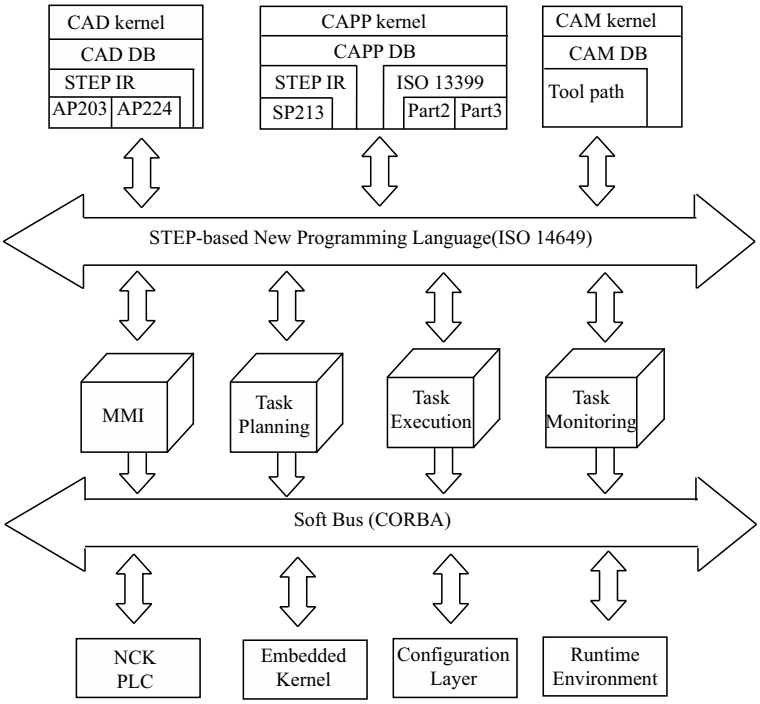


Fig. 11.21 STEP-NC interface architecture

Considering the architecture, STEP-NC technology requires various technologies such as STEP interface technology, Autonomous machining technology, Open Architectural Controller technology, CNC technology, and CAD/ CAM/CAPP technology, as shown in Fig. 11.22. These technologies can be classified into three types; 1) ISO 14649 related technologies, such as STEP interface technology and feature based CAD/CAM/CAPP technology; 2) ISO 14649 based intelligent and autonomous technologies, such as Open-architecture Soft-NC; NCK, PLC, Motion control, Autonomous task planning, On-line tool path generation, Feature-based execution, Task monitoring, and Emergency handling; 3) Computer-aided programming technologies for generating STEP-NC part programs such as shopfloor programming systems. Details about open architecture controllers and soft-NC were explained in the previous chapter, this section shows the types and architectures of STEP-CNC.

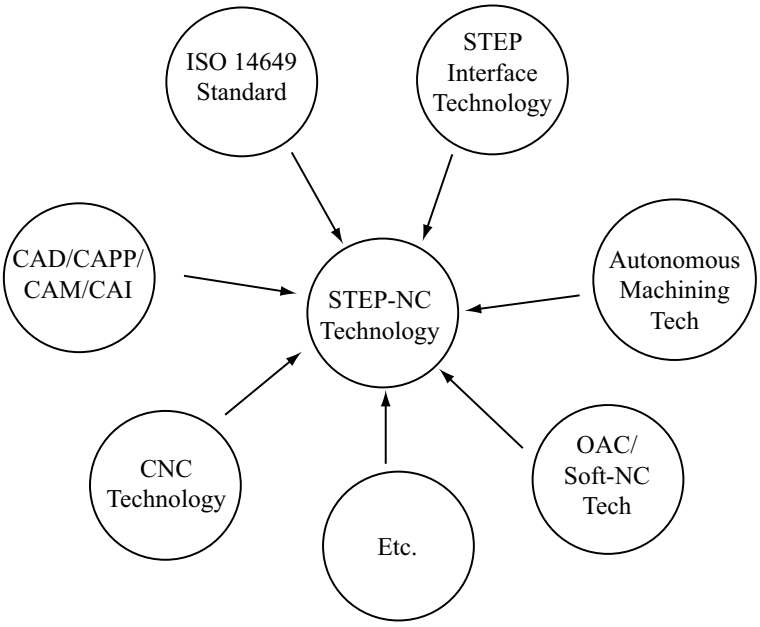


Fig. 11.22 STEP-NC related technologies

11.6.1 Types of STEP-CNC

Depending on how STEP-NC is implemented on the CNC, there are three types of STEP-CNC: (1) conventional control, (2) new control, and (3) new intelligent control, as shown in Fig. 11.23.

Type 1 simply incorporates ISO 14649 in a conventional controller via post-processing. In this case, conventional CNC can be used without modification. Strictly speaking, this cannot be considered as a STEP-compliant CNC as it should at least be able to read ISO 14649 code. Type 2, the 'New Control', has a STEP-NC interpreter in it, through which the programmed workingstep is executed by the CNC kernel with built-in toolpath generation capability. Type 2 is the basic type where the motion is executed 'faithfully' based on the machining strategy and sequence as specified by the ISO 14649 part program. In other words, it does not have intelligent functions other than the toolpath generation capability. Most of the STEP-NC prototypes developed up to the present time fall into this category.

Type 3, much more promising than the predecessors, is the 'New Intelligent Control' (Fig. 11.23), in which CNC is able to perform machining tasks 'intelligently' and 'autonomously' based on the comprehensive information of ISO 14649. Some examples of intelligent functions are automatic feature recognition, automatic collision-free toolpath generation including approach and retract motion, automatic tool selection, automatic cutting condition selection, status monitoring and automatic recovery, and machining status and result feedback.

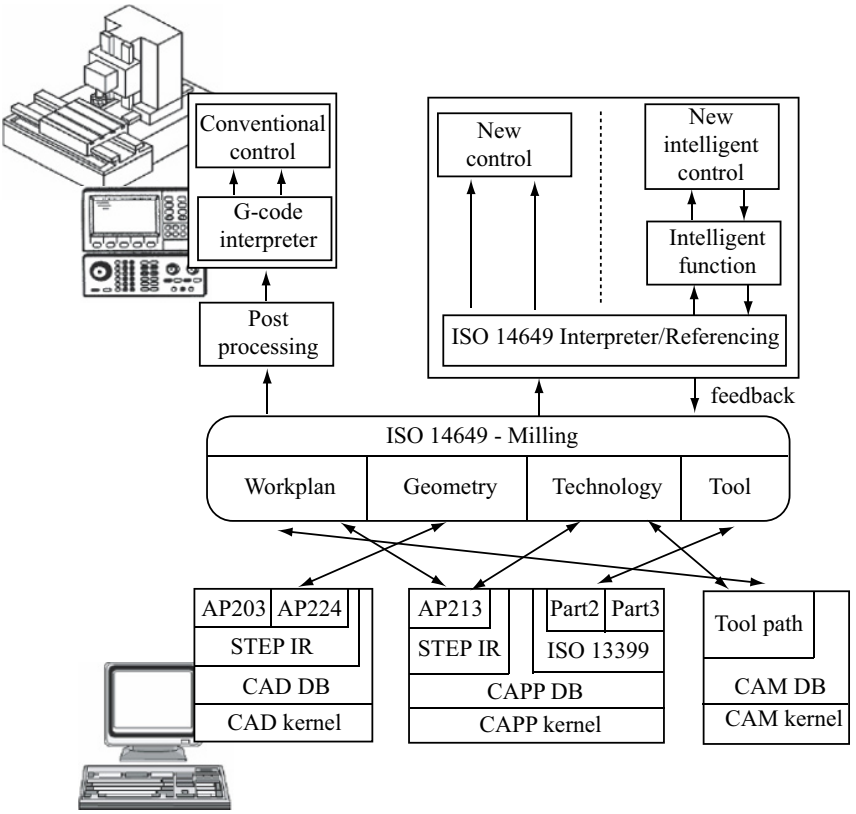


Fig. 11.23 Three types of STEP-CNC

11.6.2 Intelligent STEP-CNC Systems

The requirements for the next-generation CNC are 1) from the data-level point of view, CAD data interface with a standard schema, internet interface, seamless information exchange should be considered, 2) from the functional-level point of view, intelligence including autonomy, multi-functionality, change/failure recovery, high speed machining, and learning should be concerned, 3) from the implementation level point of view, software-based CNC, open and modular architecture, and user configurable structure are to be provided. If those requirements are satisfied, the next-generation CNC can communicate with higher-level manufacturing systems bidirectionally, maximize the control function of the machine tools, and be re-configured according to user requirements and application areas.

An example of the functional architecture of the STEP-compliant intelligent CNC (Intelligent STEP-NC) is shown in Fig. 11.24. This is composed of 1) Control modules covering various intelligent control functions, such as monitoring, decision making, execution, and so on, 2) SFP/TPG (shopfloor programming/toolpath generation)

modules, which are extended HMIs comprehensively covering part programming and toolpath generation based on a STEP-NC data model, 3) Common DB modules providing comprehensive data for the SFP/TPG and control modules, 4) non-machining modules such as Setup Manager, Inspector, and Learner.

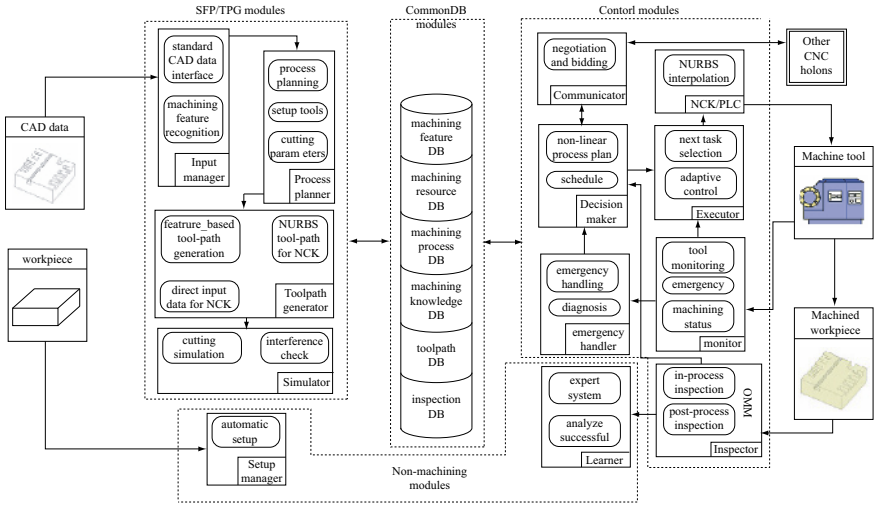


Fig. 11.24 A functional architecture of intelligent STEP-CNC

The control modules involve intra-task management of the CNC such as Decision Maker, Executor, NCK/PLC, Monitor, Emergency Handler, and Inspector.

- *Decision Maker:* This schedules the task, selecting the next task from various alternatives from a non-linear process plan. The non-linear process plan includes alternative process plans, and can be represented by an AND-OR-type graph to be explained later. One of the critical decisions is to assign the priorities between the scheduled task and the newly invoked task by the emergency handler and the inspector.
- *Executor:* This converts the task into commands and passes them to NCK/PLC. If the task is a machining operation, it retrieves the corresponding toolpath from the Tool-Path DB and passes it to NCK/PLC. If the task is a tool change, it finds the tool in the tool magazine and passes it to NCK/PLC. Executor keeps track of the commands executed by NCK/PLC for adaptive control.
- *NCK/PLC:* NCK interprets the toolpath commands and executes them by activating the servo mechanism, while PLC executes machinery commands, such as tool change and workpiece loading/unloading. For free-form surface machining, NCK is capable of NURBS interpolation in which accurate and high-speed machining can be carried out with reduced data.
- *Monitor:* The entire machining status is continuously monitored by capturing information from sensor signals. Tool monitoring and emergency detection are cru-

cial tasks. The results are sent to the emergency handler and/or the decision maker accordingly.

- *Emergency Handler:* In case of an emergency, which is monitored and reported by the monitor, the emergency handler makes a diagnosis and decides what to do about it. The result is sent to the decision maker for the final decision and scheduling. For example, in the case of tool breakage, the emergency handler retracts the tool, and checks if an alternative tool is available in the tool magazine (through Machine Resource DB). If one is available the operation is resumed with the alternative tool, otherwise it reports to the decision maker and waits for a final decision. The emergency handler can be thought of as a subtype of the decision maker, specializing in dealing with emergency.
- *Inspector:* In-process and post-process inspections are carried out automatically by the inspector. In either case, inspection is done on the machine tool by OMM (on-machine measurement). The inspector generates the toolpath for the touch probe and stores the data into the Inspection DB. Any geometrical errors between the designed part and the machined part are found by comparing the data of the inspection DB with that of the Machining Feature DB.

The SFP/TPG modules incorporate the CAM functions into the shopfloor programming system based on the STEP-NC data model. These include Input Manager, Process Planner, Toolpath Generator, and Simulator.

- Input Manager. The roles of the input manager are CAD data interface handling and machining feature recognition. It translates standard CAD data (STEP, AP203) into built-in geometric modeling kernel data, recognizes the machining features, and extracts the feature attributes required for machining. Output is stored in the Machining Feature DB.
- Process Planner. This determines the processing sequence, operations, fixtures, setups and cutting tools required to machine the features. The processing sequence is represented by a non-linear process plan so that the decision maker can select an appropriate plan at the time of execution. Optimal cutting parameters, machining strategies and tools for operations are determined using the Machining Knowledge DB. For this, a knowledge-based process planning system is required. Output is stored in the Machining Process DB.
- Tool-Path Generator. This generates toolpaths both for machining and measurement. It can generate a complete path including approach, departure, and connection path between the machining or measurement paths. The generated toolpaths are stored in the Tool-Path DB, which is accessed by NCK/PLC. As NCK/PLC is able to interpret NURBS curves directly, the toolpath generator does not segment the toolpath of a freeform curve into lines/arcs.
- Simulator. Prior to actual machining, it is necessary to perform a cutting simulation to verify the given toolpath and to detect any possible errors. The simulator finds undercut or gouging and tool interference by cutting simulation. In addition to error detection in the toolpath, optimal feedrate is calculated by using the required material removal rate during the solid cutting simulation. Output is stored in the Tool-Path DB and the Machining Process DB.

The other functions are as follows:

- Setup Manager. This supports the part setup operation. Once the part is loaded onto the machine, it finds the datum position by moving a touch probe using the workpiece and fixture geometry information.
- Learner: Information captured during machining is analyzed by an expert algorithm, and stored in the Machining Knowledge DB.
- Common DB modules: These DB modules are the repositories of data that are generated, updated, and retrieved by control modules and SFP/TPG modules. Machining feature DB, machining process DB, toolpath DB, and inspection DB are short-term databases and machine resource DB and machining knowledge DB are long-term databases. On completion of the part machining, the short-term database is cleared.
- Communicator. The communicator is responsible for the interactions with external units, such as the CAD/CAM system, shopfloor control system, and human operator:

  1. When requested by the CAD/CAM system, the CNC sends the part program in the current CNC DB.
  2. When requested by the shopfloor control system, it reports the current status including the progress of machining, and problems that occurred during machining.
  3. When the execution of a certain operation is impossible due to unexpected problems it sounds an alarm for operator attention.

Assuming that the intelligent STEP-NC presented is developed, an operational scenario is shown in Fig. 11.25 to illustrate how it works. The part programmer (user) designs a part to be machined as a workpiece in a CAD system supporting an AP 203 data model. Then, the user goes to a shopfloor programming (SFP) system installed in either an offline CAM system (external SFP) or a CNC system (built-in SFP). Then, the input manager recognizes the machining features and stores them in the machining feature DB. For each machining feature, a process plan is specified in the process planner module in terms of workingstep including machining operation and strategy together with cutting tools and cutting conditions specified in the process planner module. Considering the shape of the machining features, the user provides an alternative sequence of workingsteps graphically. Then, the CNC generates the toolpath for the cutter and touch probe (using its toolpath generator), which can be shown graphically by the simulator. After verification of the toolpath, the operation is started by pressing the cycle start button. When a tool breakage is detected, it stops the operation and invokes the emergency handling mechanism, followed by reporting to the decision maker. After the emergency case has been solved, when the inspection workingstep is required, the decision maker orders the inspector to invoke the necessary action.

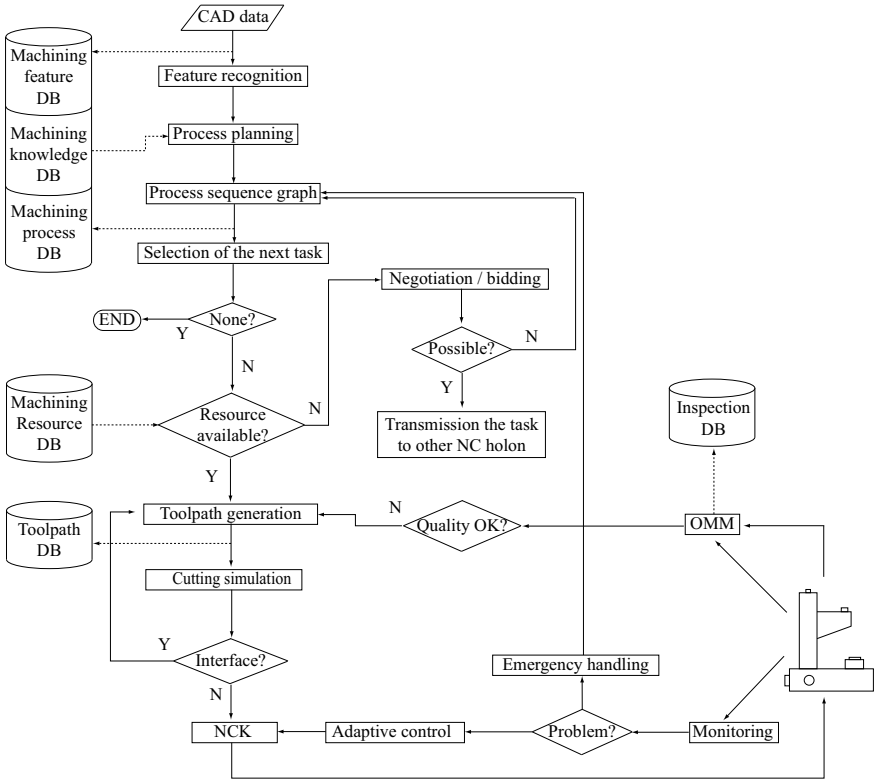


Fig. 11.25 The operation scenario in intelligent STEP-NC

11.7 Worldwide Research and Development

Due to its enormous impact STEP-NC draws keen attention from academic communities as well as major industries worldwide. They have different perspectives from each other. This difference is well reflected in the current state of STEP-NC R&D efforts throughout the world. In this section, we will introduce several representative researches, even though a large number of passionate endeavors are on-going worldwide.

11.7.1 WZL-Aachen University (Germany)

Research at WZL has focused on optimizing manufacturing planning by close coupling of a CAM System and CNC Controller. This is depicted in Fig. 11.26 in the form of a CAM client on the CNC. Since the main requirement is to assure the usability of existing machine tools and controllers, a post-processor is still neces-

sary to translate the process information into the data format of the specific CNC. However, even with the step of post-processing it is possible to enable interoperable process planning with seamless bidirectional data flow on a high information level if the post-processing of the information occurs as close to the specific CNC and as late as possible before beginning the manufacturing operation. If each CNC has its own, customized post-processor, then the input information can be controller independent. Information that cannot be transferred to and from the controller with the NC program file, can be transferred via direct software interfaces between the CAM System and CNC (CAM–CNC Coupling). This brings the high-level information of the CAM system to the shopfloor level and the CNC. Thus, this allows enriched information management at the machine tool level as well as feedback of process information to the CAM system.

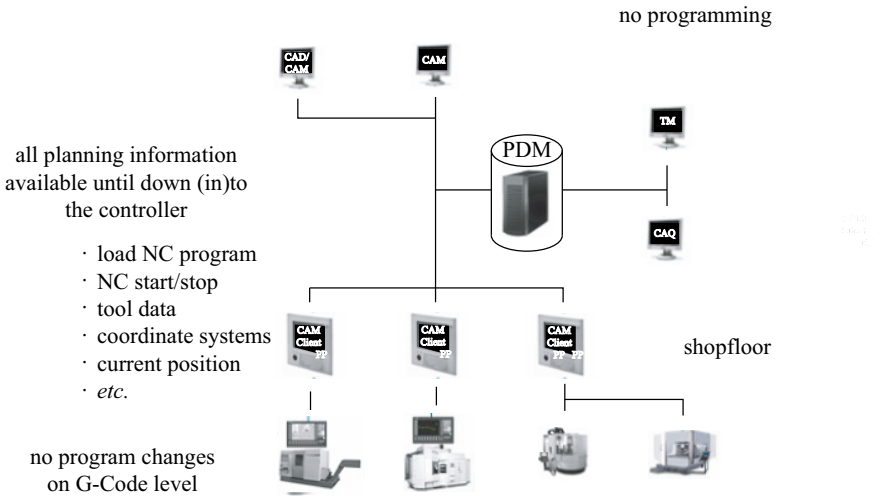


Fig. 11.26 CAM–CNC coupling based on consistent data management

Such a CAM client system might take the form of an integration framework that allows integrating software solutions of different providers (*e.g.* toolpath planning functionalities, 3D simulation of the NC program, acquisition of the real geometry of the workpiece and its consideration for toolpath planning, provision of geometry information for NC integrated collision avoidance systems). A possible detailed layout of such a system and its seamless PDM integration with all other process planning software systems in order to enable true interoperable machining based on common and consistent data is one of the current research topics at WZL.

11.7.2 ISW-University of Stuttgart (Germany)

The Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) at the University of Stuttgart researches in the area of the CAD/ CAPP/ CAM/ CNC process chain. The work focuses on methodologies, data models and software tools to utilize bidirectional information exchange between CNC and a unified manufacturing process planning database capturing STEP-NC information as illustrated in Fig. 11.27.

During the EU STEP-NC project and together with POSTECH of Korea during the IMS/EU STEP-NC project, ISW developed a STEP-NC data model for turning. To verify the turning data model, ISW developed a Computer–Aided Planning demonstrator for turning, "STEPturn", and a software module to convert STEP-NC data into the Siemens ShopTurn CNC data format. For the purpose of optimization of machining processes, *e.g.* in a small-batch manufacturing environment, the feature-based process model of STEP-NC is being utilized to structure process data acquired in open CNCs and open servo drive controllers. Relating this information about executed machining workingsteps to the corresponding manufacturing features and machining operations as well as additional context information, like the executing machine tool, helps to build a comprehensive manufacturing knowledge database.

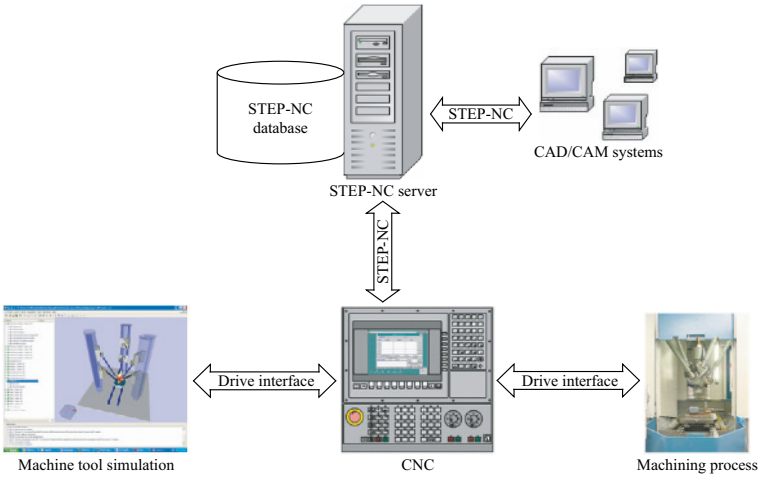


Fig. 11.27 Infrastructure to acquire process data

11.7.3 POSTECH (South Korea)

The National Research Laboratory for STEP-NC Technology (NRL-SNT) at POSTECH has made the following achievements related to STEP-NC technology:

- Development of Korea STEP-NC: STEP-CNC system for milling
- Development of TurnSTEP: STEP-CNC system for turning
- Development of the data model for turning (ISO14649 Part 12 and 121) with ISW-University of Stuttgart
- Suggestion and reflection on revision of the ISO14649 data model for milling
- Promotion of international and domestic seminars for STEP-NC

NRL-SNT developed two types of STEP-CNC system: Korea STEP-NC for milling [Suh, *et al.*, 2003 [140]] and TurnSTEP for turning [Suh, *et al.*, 2006 [13]]. The following issues have been considered in designing the architecture of STEP-CNC and are also technical contributions for implementation of STEP-NC.

- Full compliance with ISO14649 and STEP APs
- Suite of STEP-manufacturing
- Distributed architecture for e-manufacturing
- Extension to intelligent/autonomous CNC execution
- Feature recognition/mapping capability
- Tolerance handling capability
- Optimization of the machining sequence for the CNC controller
- Internet interfacing
- XML support
- Accommodation of conventional CNC
- Automated/interactive generation capability

Korea STEP-NC is an integrated system including CAD/CAM/CNC modules based on the open-modular architecture. It is composed of five modules as shown in Fig. 11.28: i) PosSFP (Shop Floor Programming), ii) PosTPG (Tool–Path Generation), iii) PosTPV (Tool–Path Viewer), iv) PosMMI (Man–Machine Interface), and v) PosCNC. For communication between these modules CORBA is used. Korea STEP-NC is capable of execution of STEP-NC code without G-code and for direct interpolation of STEP-NC toolpaths using Soft-NC technology.

TurnSTEP for rotational parts fully supports ISO14649 Part 12 and 121 as a means for verifying the data model. It is composed of three subsystems: i) CGS (Code-Generating System), ii) CES (Code-Editing System), and iii) ACS (Autonomous Control System), as illustrated in Fig. 11.29.

The three subsystems interface to the Internet together with a CAD system generating a part-geometry file, and the STEP-NC repository. CGS is used for generating neutral (hardware independent) part programs, and CES is for customizing the neutral part program for the machine tools that will be used for executing the STEP-NC code. Finally, ACS is used for controlling the machine tools based on the hardware converted STEP-NC code. The developed STEP-NC repository enables data sharing

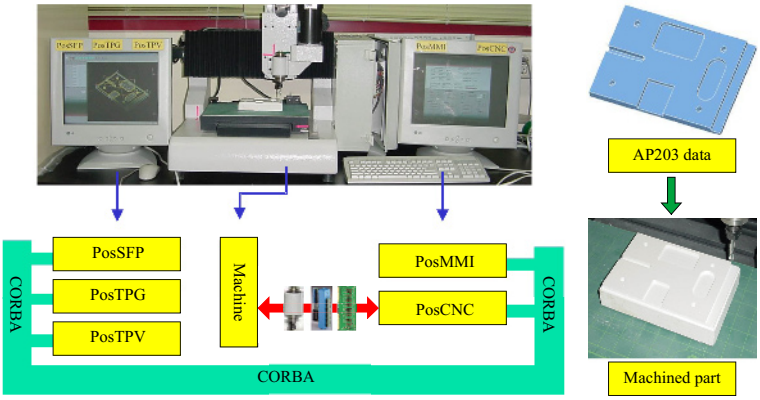


Fig. 11.28 The prototype Korea STEP-NC and the machined part

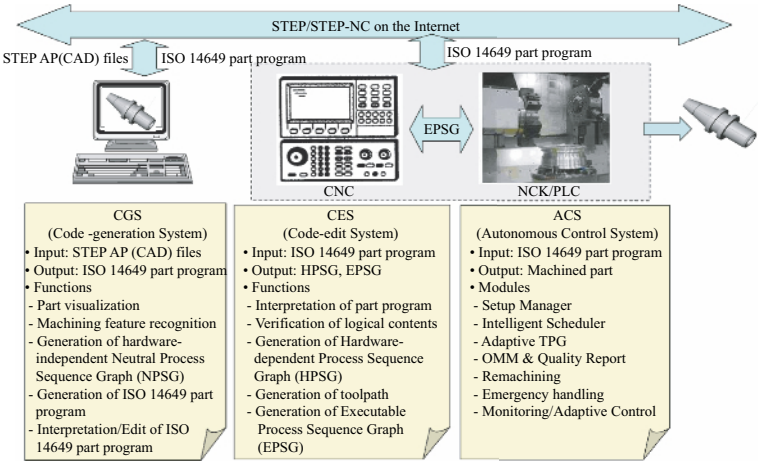


Fig. 11.29 Three subsystems of TurnSTEP

anytime, anywhere and on any platform. In addition, by expressing STEP data using XML as a core technology of the repository, product data can be easily stored and shared across the Web. A translator has been developed to convert STEP data in the clear text format into XML and *vice versa*.

11.7.4 Ecole Polytechnic Fédérale of Lausanne (Switzerland)

STEP-NC work at the EPFL concentrated on EDM with other Swiss partners. As well as control based on STEP-NC, design features, feature-based process planning

and optimization methods are being developed. For manufacturing, possible feature information from CAD may or may not be useful for manufacturing, depending on the reasons for which they were introduced. Current work is on Malcolm Sabin's back-building process planning method, involving recognizing and selecting sets of features and removing them successively until the desired stock is reached. The features removed are recorded for organization into a 'micro' process plan for machining using STEP-NC. This work is also related to another on-going project, on eco-evaluation, where it is planned to define methods for adaptive control, optimizing ecological parameters, based on STEP-NC.

11.7.5 University of Bath (UK)

Research at the University of Bath is developing a novel universal manufacturing platform that utilizes the STEP-NC data models and accentuates it with the functionality of mobile agents and manufacturing knowledge-bases. Figure 11.30 illustrates the conceptual view for the platform where various CAx applications can exchange information seamlessly. In addition to CAD, CAM, CAPP and CNC interfaces, business applications such as ERP, scheduling and costing can also exchange information with the various systems. This allows the systems to link business information to the manufacturing data and resources.

In order to achieve full interoperability, the platform requires abstraction of resources, encoding relevant knowledge in a standardized manner and communication infrastructure to transfer data from one application to another. The STEP-NC data model is utilized as the basis for the representation of manufacturing knowledge contained within the platform. An XML-based structure to represent resources has therefore been developed to support encoding of the various CAx system capabilities. The open approach used in the development of the XML resource schema allows it to be modified to comply with the new standards currently being developed for resource representation.

11.7.6 NIST (USA)

Interoperability between discrete parts manufacturing equipment is a large part of NIST's standards work conducted by the Manufacturing Engineering Laboratory (MEL). MEL's Smart Machining Systems program is focusing on issues relevant to CNC interoperability. These smart machining systems are envisioned to know and communicate their capabilities and condition to monitor and optimize their operations autonomously, to assess the quality of their output and to learn and improve themselves over time. The program considers "smart data" to be vital to achieving smart machines.

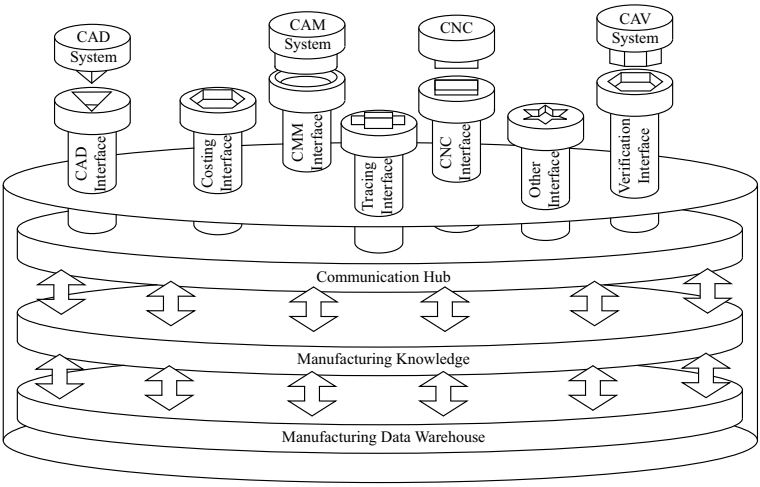


Fig. 11.30 Universal manufacturing platform architecture

The NIST Advanced Technology Program (ATP) funded a project to validate the use of STEP-NC in manufacturing applications. This project, the Model-Driven Intelligent Control of Manufacturing (also known as the "Super Model" project), began in 1999 with the goal of using STEP-NC and other standards to develop an open database of all the information necessary to design and manufacture a part. While NIST understands the value of standards-based data exchange, it is the "smart data" component that is expected to revolutionize machining. Toward this end, NIST has developed a dynamic optimizer that uses physics-based models of machining, coupled with measurements of machine tool performance and tool characteristics, to generate optimal speed and feed settings that reduce cycle times compared with conservative handbook values. The Matlab-based optimizer was recently coupled with a STEP-NC front end that takes a process plan for turning, extracts relevant information for optimization, runs the optimizer, and merges the optimized parameters back into the original STEP-NC file.

11.8 Future Prospects

Research and development on STEP-Manufacturing has been actively pursued and it has been demonstrated to work in practice both internationally and locally. At present, an effort has been made to apply the techniques to real industrial areas. However, truly, it is hard to realize full STEP-Manufacturing in one step due to the time, cost and technological difficulties. For this reason, the authors suggested the STEP-Manufacturing Roadmap more focused on the STEP-NC domain, as shown in Table 11.3. This roadmap is composed of three steps as the specific approach

methodology for the formalization of the STEP-Manufacturing environment. The roadmap takes into consideration the following itemized strategies:

- Collaborative participation with many manufacturing-related companies
  - Collaborative interaction with design-engineering-machining company chains, CAD/CAM software users, CNC controller developers, CNC machine tool users and/or builders
- Inducement toward an information-oriented and international
  - Spread to information-oriented company and information exchange among collaborating companies
  - Gradual extension from local cluster to global environment and from metal working to other industrial sectors
- Consideration of compact and economical research and development
  - Practical use from conventional products to new intelligent STEP-based products
  - Inducement of implementation from partial to whole
  - Technology and service offered through Web services
  - Maximum utilization of accumulated know-how from R&D organizations

Despite the short history of STEP-NC and on-going development of this standard, a large number of research works have been carried out across the whole world. From the perspective of the STEP-NC data model, milling and turning data models have been published as International Standards, EDM is in the process of being introduced, and other data models including the machine tool data model, inspection and rapid prototyping are currently in progress. Simultaneously, the second edition versions of some ISO 14649 parts have been under development in order to complement the first versions.

From the perspective of STEP-CNC systems, current research for the first type of STEP-CNC, which simply incorporates ISO 14649 in a conventional controller via post processing, have been carried out by the consortia that are composed of many CAD, CAM, CNC vendors and user groups. With the development of STEP-NC technology, the second type of STEP-CNC, having an ISO 14649 interpreter, will replace G-code-based controllers or the first type of STEP-CNC. Finally, the third type STEP-CNC that enables performance of ‘intelligent’ and ‘autonomous’ machining based on the comprehensive information will dominate the CNC market. Considering the current momentum of research in STEP-NC areas, these challenges will come true within a score of years.

Table 11.3 STEP-Manufacturing roadmap

		1 step (the beginning period)	2 step (the employment period)	3 step (the completion period)
Objective/ Benefit		STEP-Mfg infra introduction through the minimum investment	Merit acquisition of STEP-Mfg by STEP-Mfg settlement	e-Mfg paradigm implementation based on STEP-Mfg
Time frame		2 year (TBA)	3~4 year (TBA)	After 5 year (TBA)
Infra range		Intranet (in company)	Internet (in local area)	Internet (international)
Information exchange level		Hybrid (STEP, STEP-NC, G-code)	Partial STEP-Mfg (STEP AP203, ISO14649)	Full STEP-Mfg (STEP APs, ISO14649, ...)
Implementation level	CNC	Type 1 (conventional control) via post-processing	Type 2 (new control) via new & w/STEP-NC interpreter (Siemens)	Type 3 (intelligent control) via new & intelligent controller (TurnSTEP-ACS)
	CAD/ CAM	Legacy software with STEP-NC interface (ST-Plan, ST-Machine)	STEP & STEP-NC based CAPP/CAM (PosSFP, TurnSTEP-CGS)	CAPP/CAM for intelligent STEP-Mfg (TurnSTEP-CES)
Required technology	STEP-x interface	STEP-NC interpreter, Post-processor for Type 1 (STEP-NC → G-code)	STEP & STEP-NC interpreter, STEP-NC converter (G-code → STEP-NC)	STEP, STEP-NC interpreter
	Web service	Web-service build-up in server side (settlement of web-service range)	STEP-Mfg application build-up in client side	Client-Server harmonization and improvement
	DB build-up	Local DB	Global DB (STEP-Mfg repository)	Global DB (STEP-Mfg repository)
Role division	Company	Intranet infra in company, STEP-Mfg introduction	STEP-Mfg infra employment	e-Mfg infra employment
	R&D center	STEP-Mfg component technology research and spread	Component technology development, Conformance verification	Verification of reliability, conformance, interoperability
	Government	STEP-Mfg introduction support, local IT infra build-up business	Infra technology employment business, Local IT infra build-up business	Commercial use business, certification business, IT infra enlargement (nation)

Appendix A
Turning and Milling G-code System

A.1 Turning

Table A.1 G-codes for turning

G-code	Grp.	Function	Format
G00	1	Rapid traverse	[X_/U_][Y_/V_][Z_/W_]
G01	1	Linear interpolation	[X_/U_][Y_/V_][Z_/W_]
G02	1	Circular interpolation in clockwise direction	[X_/U_][Y_/V_][Z_/W_] [R_/I_J_K_]
G03	1	Circular interpolation in counter-clockwise direction	[X_/U_][Y_/V_][Z_/W_] [R_/I_K_]
G04	0	Dwell	[X_/U_/P_]
G10	0	Programmable data input	P_[X_/U_][Y_/V_][Z_/W_] [R_/C_]Q_
G17	16	Selecting XY plane	
G18	16	Selecting ZX plane	
G19	16	Selecting YZ plane	
G20	6	Inch (or SI) system	
G21	6	Metric system	
G22	9	Stored stroke check function on	[X_/U_][Y_/V_][Z_/W_] I_J_K_
G23	9	Stored stroke check function off	
G25	8	Spindle vibration monitoring off	
G26	8	Spindle vibration monitoring on	
G27	0	Moving to origin and check	[X_/U_][Y_/V_][Z_/W_]

Table A1 (continued)

G28	0	Moving to origin	[X_/U_][Y_/V_][Z_/W_]
G29	0	Moving from origin	[X_/U_][Y_/V_][Z_/W_]
G30	0	Moving to 234 origin	P_[X_/U_][Y_/V_][Z_/W_]
G31	0	Skip	P_[X_/U_][Y_/V_][Z_/W_]
G32	1	Thread cutting	[X_/U_][Y_/V_][Z_/W_]
G34	1	Variable lead thread cutting	[X_/U_][Y_/V_][Z_/W_]K_
G36	0	Tool radius compensation on in X-direction	[X_/U_][Y_/V_][Z_/W_]
G37	0	Tool radius compensation on in Z-direction	[X_/U_][Y_/V_][Z_/W_]
G40	7	Tool radius compensation off	
G41	7	Tool radius compensation on left side	
G42	7	Tool radius compensation on right side	
G50	0	Setting up work coordinate system	[X_/U_][Y_/V_][Z_/W_]
G52	0	Setting up local coordinate system	[X_/U_][Y_/V_][Z_/W_]
G53	0	Setting up machine coordinate system	[X_/U_][Y_/V_][Z_/W_]
G54	14	Selecting work coordinate system	[X_/U_][Y_/V_][Z_/W_]
G55	14	Selecting work coordinate system	[X_/U_][Y_/V_][Z_/W_]
G56	14	Selecting work coordinate system	[X_/U_][Y_/V_][Z_/W_]
G57	14	Selecting work coordinate system	[X_/U_][Y_/V_][Z_/W_]
G58	14	Selecting work coordinate system	[X_/U_][Y_/V_][Z_/W_]
G59	14	Selecting work coordinate system	[X_/U_][Y_/V_][Z_/W_]
G65	0	Calling macro	P_L_A_B_C_D_E_F_H_M _Q_R_S_T_U_V_W_X_Y_Z _I_I_..J_J_..K_K_..
G66	12	Calling macro modal	P_L_A_B_C_D_E_F_H_M _Q_R_S_T_U_V_W_X_Y_Z _I_I_..J_J_..K_K_..
G67	12	Macro call off	
G68	4	Mirror image on	
G69	4	Mirror image off	
G70	0	Finish cut cycle on	P_Q_
G71	0	Outer diameter/Internal diameter turning cycle	U_R_ P_Q_U_W_
G72	0	Rough facing cycle	W_R_ P_Q_U_W_

Table A1 (continued)

G73	0	Patten repeating cycle	U_W_R_ P_Q_U_W_
G74	0	End face peck drilling cycle	R_ [X_/U_][Y_/V_][Z_/W_] P_Q_R_
G75	0	Drilling cycle on external and internal side	R_ [X_/U_][Y_/V_][Z_/W_] P_Q_R_
G76	0	Complex threading cycle	P_ _ _Q_R_ [X_/U_][Y_/V_][Z_/W_] P_Q_R_
G80	10	Canned cycle cancel	
G83	10	Cycle for face drilling for drilling	[X_/U_][Y_/V_][Z_/W_] R_Q_K_
G84	10	Cycle for face tapping	[X_/U_][Y_/V_][Z_/W_] R_P_K_
G85	10	Cycle for face boring	[X_/U_][Y_/V_][Z_/W_] R_Q_K_
G87	10	Cycle for side boring	[X_/U_][Y_/V_][Z_/W_] R_Q_K_
G88	10	Cycle for side tapping	[X_/U_][Y_/V_][Z_/W_] R_P_K_
G89	10	Cycle for side boring	[X_/U_][Y_/V_][Z_/W_]R_K_
G90	1	Outer diameter/internal diameter cutting cycle	[X_/U_][Y_/V_][Z_/W_]R_
G92	1	Threading cycle	[X_/U_][Y_/V_][Z_/W_]R_
G94	1	End face turning cycle	[X_/U_][Y_/V_][Z_/W_]R_
G96	2	Constant surface speed control	
G97	2	Constant surface speed control cancel	
G98	5	Feed per minute	
G99	5	Feed per revolution	
G107	22	Cylindrical interpolation	C_
G112	20	Polar coordinate interp-olation	
G113	20	Polar coordinate interp-olation cancel	

A.2 Milling

Table A.2 G-codes for milling

G-code	Grp.	Meaning	Data elements
G00	1	Rapid traverse	X_Y_Z_
G01	1	Linear interpolation	X_Y_Z_
G02	1	Circular interpolation in clockwise direction	X_Y_Z_[R_/I_J_K_]
G03	1	Circular interpolation in counter clockwise direction	X_Y_Z_[R_/I_J_K_]
G04	0	Dwell	[X_/P_]
G10	0	Programmable data input	L_P_R_
G15	17	Polar coordinate command	
G16	17	Polar coordinate command cancel	
G17	16	Selecting XY plane	
G18	16	Selecting ZX plane	
G19	16	Selecting YZ plane	
G20	6	Input in inches	
G21	6	Input in mm	
G22	9	Stored stroke check function on	X_Y_Z_I_J_K_
G23	9	Stored stroke check function off	
G27	0	Reference position return check	X_Y_Z_
G28	0	Automatic return to reference position	X_Y_Z_
G29	0	Movement from reference position	X_Y_Z_
G30	0	2nd, 3rd, and 4th reference position return	P_X_Y_Z_
G31	0	Skip	P_X_Y_Z_
G33	1	Threading	X_Y_Z_
G39	0	Tool radius compensation: corner circular interpolation	[X_Y_Z_/I_J_K_]
G40	7	Tool radius compensation off	
G41	7	Tool radius compensation on in left side	
G42	7	Tool radius compensation on in right side	
G43	13	Tool length compensation +	X_Y_Z_H_
G44	13	Tool length compensation –	X_Y_Z_H_

Table A2 (continued)

G49	13	Tool length compensation off	
G50	11	Scaling off	
G51	11	Scaling on	X_Y_Z_[P_/I_J_K]
G52	0	Setting local coordinate system	X_Y_Z_
G53	0	Setting machine corrdinate system setting	X_Y_Z_
G54	14	Selecting work coordinate system	X_Y_Z_
G55	14	Selecting work coordinate system	X_Y_Z_
G56	14	Selecting work coordinate system	X_Y_Z_
G57	14	Selecting work coordinate system	X_Y_Z_
G58	14	Selecting work coordinate system	X_Y_Z_
G59	14	Selecting work coordinate system	X_Y_Z_
G61	15	Exact stop mode on	
G62	15	Automatic corner override mode on	
G63	15	Tapping mode	
G64	15	Cutting mode	
G65	0	Macro call	P_L_A_B_C_D_E_F_H_M _Q_R_S_T_U_V_W_X_Y_Z _I_I_..J_J_..K_K_..
G66	12	Macro modal call	P_L_A_B_C_D_E_F_H_M _Q_R_S_T_U_V_W_X_Y_Z _I_I_..J_J_..K_K_..
G67	12	Macro modal call cancel	
G68	18	Coordinate system rot.	X_Y_Z_R_
G69	18	Coordinate system rotation cancel	
G73	10	Peck drilling cycle	X_Y_Z_R_Q_K_
G74	10	Left-handed tapping cycle	X_Y_Z_R_P_K_
G76	10	Fine boring cycle	X_Y_Z_R_Q_K_
G80	10	Canned cycle cancel	
G81	10	Drilling cycle or spot boring cycle	X_Y_Z_R_K_
G82	10	Drilling cycle or counter boring cycle with dwell	X_Y_Z_R_P_K_
G83	10	Peck drilling cycle	X_Y_Z_R_Q_K_
G84	10	Tapping cycle	X_Y_Z_R_P_K_
G85	10	Boring cycle	X_Y_Z_R_K_

Table A2 (continued)

G86	10	Boring cycle	X_Y_Z_R_K_
G87	10	Back boring cycle	X_Y_Z_R_Q_K_
G88	10	Boring cycle with dwell	X_Y_Z_R_P_K_
G89	10	Boring cycle with dwell	X_Y_Z_R_P_K_
G90	3	Absolute programming	
G91	3	Incremental programming	
G92	0	Setting for workpiece coordinate system	X_Y_Z_
G94	5	Feed per minute	
G95	5	Feed per revolution	
G96	2	Constant surface speed ctl.	
G97	2	Constant surface speed control cancel	
G98	19	Canned cycle: return to initial level	
G99	19	Canned cycle: return to R point level	
G107	22	Cylindrical interpolation	C_
G112	20	Polar coordinate interpolation mode on	
G113	20	Polar coordinate interpolation mode off	
G84.2	10	Rigid tapping cycle	X_Y_Z_R_P_K_F_
G84.3	10	Left-handed rigid tapping cycle	X_Y_Z_R_P_K_F_

A.3 Classification of G-code Groups

The group of G codes can be divided into two groups; one-shot group and modal group. The modal group consists of a variety of groups. 'One-shot group' means the set of G-codes that has an influence on a single block. Unlike one-shot G-codes, the G-codes in the modal group continue to have an influence on the next blocks until the cancel command is called.

Table A.3 G-code grouping

Group	Command
0	One-shot command
1	Feed command
2	Constant surface speed command
3	Absolute/Incremental programming command
4	Mirror image command
5	Feed unit selection command
6	Programming unit selection command
7	Tool radius compensation command
8	Spindle vibration detection command
9	Stroke limit input command
10	Cycle code command
11	Scaling command
12	Macro call command
13	Tool length compensation command
14	Work coordinate system selection command
15	Cutting mode command
16	Plane selection command
17	Polar coordinate command
18	Coordinate system rotation command
19	Return position setting command for drilling cycle
20	Polar coordinate system command
21	High-speed machining command
22	Cylindrical interpolation command
23	Skip command

*Note: the group number can vary depending on the CNC makers and is not fixed.

Bibliography

1. Altintas, Y., Munasinghe, W. K. “A Hierarchical Open-Architecture CNC System for Machine Tools”, Annals of CIRP, Vol. 43/1, pp. 349–354, 1994.
2. Altintas, Y., Erol, N. A. “Open Architecture Modular Tool Kit for Motion and Machining Process Control”, Annals of CIRP, Vol. 47/1, pp. 295–300, 1998.
3. Aronsoon, R. B. “Machine tool 101: part 1, trends”, Manufacturing engineering, pp. 31–36, January 1994.
4. Aronsoon, R. B. “Machine tool 101: part 5, controls”, Manufacturing engineering, pp. 53–57, May 1994.
5. Åström, K. J., Hägglund, T. “Automatic Tuning of Simple Regulators with Specifications on Phase and Amplitude Margins”, Automatica, Vol. 20, No. 5, pp. 646–651, 1984.
6. Auslander, D. M., Tham, C. H. “Real-Time Software for Control”, Prentice Hall, Upper Saddle River, NJ, 1990.
7. Babb, M. “IEC 1131-3: A Standard Programming Resource for PLCs”, Control Engineering, pp. 67–76, February 1996.
8. Bollinger, J. G., Duffie, N. A. “Computer Control of Machines and Processes”, Addison Wesley, Boston, MA., 1989.
9. Brecher, C., Voss, M., Almeida, C. “Component-based Open Control Systems using Open Source Software”, Proceedings Third CIRP International Conference on Reconfigurable Manufacturing Systems, May 2005.
10. Brouer, N., Weck, M. “Feature-Oriented Programming Interface of an Autonomous Production Cell”, Fourth IFAC Workshop, pp. 223–228, 1997.
11. Chang, C. H., Melkanoff, M. A. “NC Machine Programming and Software Design”, Prentice Hall, Upper Saddle River, NJ., 1989.
12. Chang, H., Lee, K. “Design and implementation of real time multitasking kernel under DOS environment”, Journal of Korean Automatic Control and System Engineering, Vol. 3, No. 4, pp. 373–380, 1997.
13. Choi, I., Suh, S., Kim, K., Song, M., Jang, M., Lee, B. “Development process and data management of TurnSTEP: a STEP-compliant CNC system for turning”, International Journal of Computer Integrated Manufacturing, Vol. 19, No. 6, pp. 546–558, 2006.
14. Chon, D., Kim, H. “Generating velocity profile for NURBS curve by look-ahead method”, Proceedings Fall Conference of KSPE, pp. 977–980, 1997.
15. Chung, C., Lee, K., Lee, B. “Software design and implementation for robot controller based on real time operating system”, Proceedings Conference of KACC, pp. 1246–1251, 1994.
16. Chu, J., Lee, H., Lee, Y., Jun, K. “A new contour error modeling method for cross coupling control”, Journal of Control, Automation and System Engineering, Vol. 3, No. 4, pp. 389–397, 1997.
17. Comer, D., Fossum, T. V. “Operating System Design, Volume I The XINU Approach”, Prentice Hall, 1988.

18. Daihatsu Motor Company, “Method of Control of NC Machine Tools”, US patent 4445182, 1984.
19. Danielsson, P. E. “Incremental Curve Generation”, IEEE Transactions on Computers, Vol. C-19, No. 9, pp. 783–793, 1970.
20. Deitel, H. M. “An Introduction to Operating Systems”, Addison Wesley, Boston, MA., 1984.
21. Deshayes, L., Donmez, A., Welsch, L., Ivester, R. “Robust Optimization for Smart Machining Systems: An Enabler for Agile Manufacturing”, Proceedings ASME International Mechanical Engineering Congress and Exposition, November 2005.
22. Fanuc, “Method for Compensating for Servo Delay caused at an ARC or Corner”, US patent 4543625, 1985.
23. Fanuc, “Acceleration/Deceleration System for A Numerical Controller”, US patent 4652804, 1987.
24. Fanuc, “Blank Profile Specifying Method”, US patent 4669041, 1987.
25. Fanuc, “Automatic Machining Process Determination Method in an Automatic Programming System”, US patent 4723203, 1988.
26. Fanuc, “Conversational-type Programming Apparatus”, US patent 4823253, 1989.
27. Fanuc, “Automatic Programming System”, US patent 4939635, 1990.
28. Fanuc, “Acceleration/Deceleration Control Apparatus”, US patent 4994978, 1991.
29. Fanuc, “System for processing MST function command”, US patent 4982335, 1991.
30. Fanuc, “Programming method for drilling holes”, Patent applied for 92-10197, 1992.
31. Fanuc, “Part Profile Input Method”, US patent 5089950, 1992.
32. Fanuc, “Acceleration/Deceleration Control Method for A Numerical Control Device”, US patent 5218281, 1993.
33. Fanuc, “Graphic mechanism for checking interferences”, Patent applied 94-700698, 1994.
34. Fanuc, “Conversational automatic programming system”, Patent applied 94-702618, 1994.
35. Fanuc, “Numerical controller capable of estimating finishing time of machining operation”, Patent applied 94-704020, 1994.
36. Fanuc, “Feedforward control method for a servomotor”, US patent 5448145, 1995.
37. Fanuc, “Free-Form Curve Interpolation Method and Apparatus”, US patent 5815401, 1998.
38. Fanuc, “Numerical control system”, Patent applied 88-700333, 1988.
39. Fanuc, “Free Curve Interpolation Apparatus and Interpolation Method”, US patent 5936864, 1999.
40. Final Report of ESPRIT Projects 6379 & 9115, “Open system architecture for controls within automation systems: OSACA & Final report”, ESPRIT, April 1996.
41. Fujita, S., Yoshida, T., “OSEC: Open System Environment for Controller”, Proceedings Seventh International Machine Tool Engineering Conference, pp. 234–244, 1996.
42. GE Fanuc Automation North America, Inc., “Description Manual, Series 21/210-Model B”, 1996.
43. GMPTG, “Open modular architecture controls at GM POWERTRAIN - Technology and Implementation: version 1.0”, White paper downloaded from http://www.arcweb.com/omac, May 1996.
44. Hägglund, T., Åström, K. J. “Industrial Adaptive Controller Based on Frequency Response Techniques”, International Federation of Automatic Control, pp. 599–609, 1991.
45. Hardwick, M. “Justifying the STEP-NC savings”, Fourth MDICM IRB Meeting, June 2001.
46. Heusinger, S., Rosso-Jr, R., Klemm, P., Newman, S., Rahimifard, S. “Integrating the CAx process chain for STEP-compliant NC manufacturing of asymmetric parts”, International Journal of Computer Integrated Manufacturing, Vol. 19, No. 6, pp. 533–545, 2006.
47. Hughes Aircraft Company, “Robot Axis Controller Employing Feedback and Open Loop (feed forward) Control”, US patent 4912753, 1990.
48. Hurco Companies, Inc., “CNC Control System”, US patent 5453933, 1995.
49. Hyundai Motor Company, “Users’ manual for HiTROL-M100”, 2000.
50. Hyundai Motor Company, “Programming manual for HiTROL M-100”, 2000.
51. Hyundai Motor Company, “Operating manual for HiTROL M-100”, 2000.
52. Hyundai Precision Co. Ltd, “Implementation method for CNC controller using single CPU”, Korea Patent applied 99-79125, 1999.

53. Hyundai Precision Co. Ltd, “CNC controller having real time measurement capability”, Korea Patent applied 99-79804, 1999.
54. Hyundai Precision Co. Ltd, “Turning CNC with tool path generation method”, Korea Patent applied 99-74236, 1999.
55. ICOM, “Method and Apparatus for Creating Custom Displays for Monitoring Ladder Logic Programs”, US patent 4991076, 1991.
56. Im, S., Kang, S., Choi, J. “Machining cycle for rough cut and finish cut of turning operation with collision-avoidance capability”, Proceedings of Conference of KSPE, pp. 1050–1053, 1997.
57. ISO/FDIS 14649-1, “Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 1: Overview and fundamental principles”, ISO/TC 184/SC 1/WG 7, 2002.
58. ISO/FDIS 14649-10, “Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data”, ISO/TC 184/SC 1/WG 7, 2004.
59. ISO/FDIS 14649-11, “Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling”, ISO/TC 184/SC 1/WG 7, 2004.
60. ISO/FDIS 14649-12, “Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 12: Process data for turning”, ISO/TC 184/SC 1/WG 7, 2005.
61. ISO/FDIS 14649-111, “Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 111: Tools for milling machines”, ISO/TC 184/SC 1/WG 7, 2004.
62. ISO/IS 14649-121, “Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 121: Tools for turning machines”, ISO/TC 184/SC 1/WG 7, 2005.
63. Kang, C. “Machine tool technology: present and prospect”, Journal of Korea Society of Precision Engineering, Vol. 13, No. 3, pp. 9–25, 1996.
64. Kang, J., Suh, S., “Machinability and set-up orientation for 5-axis numerically controlled machining of free surfaces”, International Journal of Advanced Manufacturing Technology, Vol. 13, No. 5, pp. 311–325, 1997.
65. Kang, S., Lee, J., Choi, J. “Shop floor programming system for turning operations”, Proceedings Conference of KSPE, pp. 707–712, 1995.
66. Kang, S. “NURBS interpolator for open architecture CNC control system”, Proceedings Eleventh Conference of KACC, pp. 656–659, October 1996.
67. Kim, D., Song, J., Kim, S. “Software-based acceleration and deceleration method for CNC machine and industrial robot”, Journal of Korea Electrical Engineering, pp. 562–572, 1992.
68. Kim, H. J., Kim, G. T., Choi, B. W. “Architecture Design for Open Intelligent Manufacturing System”, Proceedings Fourth IFAC Workshop, pp. 335–338, 1997.
69. Kim, H. “Characteristic analysis for system development”, Proceedings Conference of KACC, pp. 22–27, 1998.
70. Kim, J., Im, Y., Chung, S. “Real time control system for multi-motor manipulation”, Proceedings Eleventh Conference of KACC, pp. 1048–1051, 1996.
71. Kim, S. “Asymptotic PID control based on fuzzy inference and modified Ziegler–Nichols method”, Journal of KSPE, Vol. 13, No. 5, pp. 74–83, 1996.
72. Kim, S. “Small motor control”, Sungandang Publishing Co., Seoul, 2000.
73. Koren, Y. “Interpolator for a Computer Numerical Control System”, IEEE Transactions on Computers, Vol. C-25, No. 1, pp. 32–37, 1976.
74. Koren, Y. “Cross coupled biaxial computer control for manufacturing system”, ASME Transactions, Journal of Dynamic Systems, Measurement and Control, Vol. 102, No. 4, pp. 265–272, December 1980.
75. Koren, Y., Masory, O. “Reference-pulse circular interpolators for CNC systems”, Journal of Engineering for Industry, Vol. 103, pp. 131–136, 1981.

76. Koren, Y. “Computer Control of Manufacturing System”, McGraw Hill, New York, NY., 1983.
77. Koren, Y., Lo, C. “Variable gain cross coupling controller for contouring”, CIRP Annals, Vol. 104, pp. 371–374, August 1991.
78. Koren, Y., Lo, C. “Advanced Controllers for Feed Drives”, CIRP Annals, Vol. 41, pp. 689-697, 1992.
79. Koren, Y., Pasek, Z. J., Ulsoy, A. G., Benchetrit, U. “Real-Time Open Control Architecture and System Performance”, CIRP Annals, Vol. 45/1, pp. 377–380, 1996.
80. Koren, Y. “Control of machine tools”, Journal of Manufacturing Science and Engineering, Vol. 119, pp. 749–755, November 1997.
81. Kuo, B. C. “Automatic Control Systems”, Fourth Edition, Prentice-Hall, Upper Saddle River, NJ., 1981.
82. Lavallee, R. “Soft Logic’s New Challenge: Distributed Machine Control”, Control Engineering, pp. 51–58, August 1996.
83. Lee, B. “NC readings”, Sungandang Publishing Co., Seoul, 1995.
84. Lee, J., Park, H., Huh, W. “Tracking control of servo system with two degrees-of-freedom”, Proceedings Eleventh Conference of KACC, pp. 844-847, 1996.
85. Lewis, R.W. “Programming Industrial Control Systems using IEC 1131-3”, IEC, 1995.
86. LG Industrial Electronics Ltd., “Method for generating acceleration and deceleration for CNC machine tools”, Patent applied 96-15124, 1996.
87. Lo, C. C. “Feedback Interpolators for CNC Machine Tools”, Journal of Manufacturing Science and Engineering, Vol. 119, pp. 587–592, 1997.
88. Lynch, M. “Computer Numerical Control, Advanced Techniques”, McGraw Hill, New York, NY., 1993.
89. Lyu J., Kang S., Chon, Y. “Minimum time pocket machining cycle for conversational programming system”, Proceedings Conference of KACC, pp. 848–851, 1996.
90. Mazak, “Programming Manual for Mazatrol T32-2”, 1990.
91. MDSI, “Open CNC Demo V5.1.1”, www.misi2.com, 2000.
92. Mitsubushi Denki, “Numerical control machining method”, Patent applied 84-6114, 1984.
93. Mitsubishi Denki, “Numerically Controlled Machining Method using Primary and Compensating cutters”, US patent 4713747, 1987.
94. Mitsubishi Denki, “Machining data editing method for CNC controller”, Patent applied 91-1508, 1991.
95. Na, I., Choi, J., Chang, T., Choi, B., Song, O. “Contouring error analysis for PID control of machining center”, Journal of ICASE, pp. 32–39, 1997.
96. NAF Controls AB, “Method and an Apparatus in Tuning a PID Regulator”, US patent 4549123, 1985.
97. National Instruments Corp., “System and Method for Closed Loop Autotuning of PID Controllers”, US patent 6081751, June 2000.
98. NRL-SNT, www.step-nc.com, 2008.
99. Newman, S., Ali, L., Brail, A., Brecher, C., Klemm, P., Liu, R., Nassehi, A., Nguyen, V., Proctor, F., Rosso-Jr, R., Stroud, I., Suh, S., Vitr, M., Wang, L., Xu, X. “The Evolution of CNC Technology from Automated Manufacture to Global Interoperable Manufacturing”, Second International Conference on Changeable, Agile, Reconfigurable and Virtual Production, July 2007.
100. Newman, S., Nassehi, A., Kumar, T., Vichare, P. “Universal Manufacturing Platform for Global CNC Machining”, International Workshop on Ubiquitous Manufacturing, February 2008.
101. Ohyama, M., Tamai, S. “AC servo system: theory and design in practice”, Jonghap Publishing Co., Seoul, 1990.
102. Oki Electric, “System for Interpolating an ARC for a Numerical Control System”, US patent 4243924, 1981.
103. Olsson, G., Piani, G. “Computer Systems for Automation and Control”, Prentice Hall, 1992.
104. www.arcweb.com/omac, 2000.

105. OMG, “CORBA 3.0 New Components Chapters OMG TC Document ptc/99-10-04”, October 1999.
106. OMG, “The Common Object Request Broker: Architecture and Specification Revision 2.4”, October 2000.
107. OSE consortium, “Development of OSEC (Open System Environment for Controller)”, OSEC Project technical report, October 1998.
108. Park, C. “Cycle machining method for numerical control”, Patent applied 96-5564, 1996
109. Park, S., Jung, S. “Priority determination method under real time CORBA environment”, Journal of Korea Electrical Engineering, Vol. 38, No. 4, pp. 59–71, 2001.
110. Piegl , L., Tiller, W. “ The NURBS Book”, Second Edition, Springer Verlag, London, 1995.
111. Poo, A. N., Bolinger, J. G., Younkin, G.W. “Dynamic errors in type I contouring systems”, IEEE Transaction on Industrial Application, Vol. T.8, No. 4, pp. 477–484, 1972.
112. Pritschow, G., Philipp, W. “Research on the Efficiency of Feedforward Controllers in M Direct Drives”, CIRP Annals, Vol. 41/1, pp. 411–415, 1992.
113. Pritschow, G., Daniel, G., Junghans, G., Sperling, W. “Open System Controllers-A Challenge for the Future of the Machine Tool Industry”, CIRP Annals, Vol. 42/1, pp. 449–452, 1993.
114. Richard, J., Nguyen, V., Stroud, I. “Standardization of the Manufacturing Process: IMS/EU STEP-NC project about Wire EDM Process”, Intelligent Manufacturing System Conference, May 2004.
115. Rober S. J., Shin, Y. C. “Modeling and Control of CNC Machine using A PC-based Open Architecture Controller”, Mechatronics, Vol. 5, No. 4, pp. 401–420, 1995.
116. Samsung electronic, “Acceleration and deceleration method for servo motor”, Patent applied 96-24775, 1996.
117. Samsung electronic, “Tool path generation method for turning operations”, Patent applied 98-40717, 1998.
118. Schmitz, D., Khosla, P. “CHIMERA: A Real-Time Programming Environment for Manipulator Control”, Proceedings IEEE Conference of Robotics Automation, pp. 846–852, 1989.
119. Schofield, S., Wright, P. “Open Architecture Controllers for Machine Tools, Part 1: Design Principles”, Journal of Manufacturing Science and Engineering, Vol. 120, pp. 417–424, 1998.
120. Servo Motor, www.tao.co.kr, 2001.
121. Shin, D., Chung, S. “Design of digital filter for acceleration and deceleration control of servo motor”, Journal of KSPE, Vol. 14, No. 9, pp.52–60, 1997.
122. Shin, S., Suh, S., Stroud, I. “Reincarnation of G-code based part programs into STEP-NC for turning applications”, Computer Aided Design, Vol. 39, pp.1–16, 2007.
123. Siemens, “Method for limiting the rate-of-change of acceleration in numerical driving systems”, US patent 5073748, 1991.
124. Siemens, “Sinumerik 840D OEM package NCK: User’s Manual”, Preliminary Edition, March 1995.
125. Silberschatz, A., Peterson, J., Galvin, P. “Operating System Concepts”, Third Edition, Addison Wesley, Boston, MA., 1991.
126. Srinivasan, K., Kulkarni, P. “Cross coupled control of biaxial feed drive servomechanism”, Transactions of ASME, Journal of Dynamic Systems, Measurement and Control, Vol. 112, No. 2, pp. 225–232, June 1990.
127. Srinivasan, K., Tsao, T. C. “Machine tool feed drives and their control: A survey of the state of the art”, Journal of Manufacturing Science and Engineering, Vol. 119, pp.743–748, November 1997.
128. STEP tools, www.steptools.com, 1999.
129. Stewart, D., Volpe, R., Khosla, P. “Design of Dynamically Reconfigurable Real-Time Software using Port-Based Objects”, IEEE Transaction on Software Engineering, Vol. 23, No. 12, pp. 759–776, 1997.
130. Stroud, I., Xirouchakis, P. “Strategy features for communicating aesthetic shapes for manufacturing”, International Journal of Computer Integrated Manufacturing, Vol. 19, No. 6, pp. 639–649, 2006.
131. Suh, S., Lee, K. “A prototype CAM system for 4-axis NC machining of rotational-free-surfaces”, Journal of Manufacturing Systems, Vol. 10, No. 4, pp. 322–331, August 1991.

132. Suh, S., Cho, J., Hascoet, J. “Incorporation of tool deflection in tool path computation: simulation and analysis”, Journal of Manufacturing Systems, Vol. 15, No. 3, 1996.
133. Suh, S., Lee, S., Lee, J. “Compensating probe radius in free surface modeling with CMM: simulation and experiment”, International Journal of Production Research, Vol. 34, No. 2, pp. 507–523, 1996.
134. Suh, S., Lee, J. “Interference-free tool-path planning for flank milling of twisted ruled surfaces”, International Journal of Advanced Manufacturing Technology, Vol. 14, No. 11, pp. 795–805, 1998.
135. Suh, S., Lee, J., Kim, S., “Multiaxis machining with additional-axis NC system: theory and development”, International Journal of Advanced Manufacturing Technology, Vol. 14, No. 12, pp. 865–875, 1998.
136. Suh, S. “Numerical Control and Integrated Manufacturing Systems”, Second Edition, POSTECH Press, September 1999.
137. Suh, S. “STEP-NC Technology: Present and prospect”, Journal of KSPE, Vol. 17, No. 5, pp. 8–14, 2000.
138. Suh, S., Jih, W., Hong, H., Chung, D., “Sculptured surface machining of spiral bevel gears with CNC milling”, International Journal of Machine Tools & Manufacture, Vol. 41, pp. 833–850, May 2001.
139. Suh, S. and Cheon, S. “A Framework for an Intelligent CNC and Data Model”, International Journal of Advanced Manufacturing Technology, Vol. 19, pp. 727–735, 2002.
140. Suh, S., Lee, B., Chung, D., Cheon, S. “Architecture and implementation of a shop-floor programming system for STEP-compliant CNC”, Computer Aided Design, Vol. 35, pp. 1069–1083, 2003.
141. Suh, S., Lee, B. “STEP-Manufacturing Roadmap”, Proceedings Korea CAD/CAM Conference, January 2004.
142. Suh, S., Chung, D., Lee, B., Shin, S., Choi, I., Kim, K. “STEP-compliant CNC system for turning: Data model, architecture, and implementation”, Computer Aided Design, Vol. 38, pp. 677–688, 2006.
143. Suh, S., Stroud, I. “A new model for machine data transfer”, ISO Focus, pp. 24–26, 2007.
144. Sung, W. “A study on surface machining with CNC machining center”, MS Thesis, Seoul National University, 1998.
145. Szafarczyk, M., Klein, B., Szala, W. “Extension of Typical CNC Systems by External Controllers”, CIRP Annals, Vol. 38/1, pp. 351–354, 1989.
146. Szyperski, C., “Component Software: Beyond Object-Oriented Programming”, Addison Wesley, Boston, MA., 1999.
147. Teltz, R., Elbestawi, M. “Design Basis and Implementation of an Open Architecture Machine Tool Controller”, Transactions of NAMRI/SME, Vol. XXV, pp. 299–304, 1997.
148. Timmerman, M., *et al.*, “Windows NT real-time extensions better of worse?”, Real-Time Magazine 98-3, pp. 11–19, 1998.
149. Timmerman, M., *et al.*, “Is Windows CE 2.0 a real threat to the RTOS World?”, Real-Time Magazine 98-3, pp. 20–24, 1998.
150. Tomizuka, M. “Zero phase error tracking algorithm for digital control”, ASME Transactions, Journal of Dynamic Systems, Measurement and Control, Vol. 109, pp. 65–68, 1987.
151. Toshiba, “Numerical Control Unit”, US patent 5994863, 1999.
152. University of Utah Research, “Method and system for spline interpolation, and their use in CNC”, US patent 5726896, 1998.
153. U.S. Philips Corp., “Machining apparatus wherein ARC length along a tool path is determined in relation to a parameter which is a monotonic function of time”, US patent 5321623, 1994.
154. VenturCom, “RTX 4.1 manual”, 1997.
155. Weck, M., Bibring, H. “Handbook of Machine Tools, Vol. 3: Automation and Control”, John Wiley & Sons, Hoboken, NJ., 1984.
156. Wosnik, M., Kramer, T., Selig, A., Klemm, P. “Enabling feedback of process data by use of STEP-NC”, International Journal of Computer Integrated Manufacturing, Vol. 19, No. 6, pp. 559–569, 2006.

157. Xu, X., Wang, H., Mao, J., Newman, S., Kramer, T., Proctor, F., Michaloski, J. "STEP-compliant NC research: the search for intelligent CAD/CAPP/CAM/CNC integration", International Journal of Production Research, Vol. 43, No. 17, pp. 3703–3743, 2005.
158. Xu, X., Proctor, F., Klemm, P., Suh, S. "STEP-compliant process planning and manufacturing", International Journal of Computer Integrated Manufacturing, Vol. 19, No. 6, pp. 491–494, September 2006.
159. Yamazaki, K. "Open Architecture CNC Controller in the USA", Proceedings Seventh International Machine Tool Engineering Conference, pp. 204–218, 1996.
160. Yang, D., Kong, T. "Parametric interpolator versus linear interpolator for precision CNC machining", Computer Aided Design, Vol. 26, No. 3, pp. 225–233, 1994.
161. Yeh, S., Hus, P. "The speed-controlled interpolator for machining parametric curve", Computer Aided Design, Vol. 31, No. 5, pp. 349–357, 1999.
162. Zahavi, R. "Enterprise Application Integration with CORBA: Component and Web-Based Solutions", John Wiley & Sons, 2000.
163. Ziegler, J. G., Nichols, N. B. "Optimum Settings for Automatic Controllers", ASME Transactions, pp. 759–768, 1942.
164. Ziegler, J. G., Nichols, N. B., "Process Lags in Automatic-Control Circuits", ASME Transactions, pp. 433–444, 1943.

Index

AAM, 404
absolute-type encoder, 13
AC servo motor – synchronous, 11
acc/dec control, 107
acc/dec control – algorithm, 109
acc/dec control – block overlap, 126–128
acc/dec control – digital circuit, 112
acc/dec control – digital filter, 109
acc/dec control – exponential, 117–120
acc/dec control – filter, 109
acc/dec control – functions, 200, 216
acc/dec control – implementation, 199, 202, 215
acc/dec control – input/output, 199, 215
acc/dec control – linear, 112–114
acc/dec control – machining error, 121–126
acc/dec control – S-shape, 114–116
acceleration, 107–114
acceleration - deceleration controller, 24
adaptive control module, 157
ADCAI, 107–128, 187
ADCBI, 107, 108, 128–155, 187, 211
ADCBI-type NCK – architecture, 211
address, 237
aging, 328
AGV, 4
AIM, 404
and – AND, 260
and not – ANDN, 261
and stack – ANDS, 266
application layer, 275
approximation errors, 77
APT, 281
architecture – system hardware, 344
area for data input, 273
area for machine operation, 273
area for MPG handling, 273
area for status display, 271
ARM, 400, 404
automatic gain tuning, 168
automatic programming, 278, 280

ball screw, 15
ball screw mechanisms, 4, 6
basic instruction, 250, 256
Bath–United Kingdom, 427
binary semaphore, 330
block classification – normal-normal, 133–136
block classification – normal-short, 138, 139
block classification – short-normal, 136–138
block classification – short-short, 140, 141
block overlap, 126, 132
block overlap control, 132
block record buffer, 359, 374
block record memory, 65, 67

CAD, 4
CAI, 4, 6
CAM, 4
CAN Bus, 22
cancel mode, 50
CAPP, 4, 278
cascade loop structure, 173
cascade structure, 159
causal FIR, 176
causal FIR filter, 177
causal/noncausal FIR, 176
chord height error, 90
circular slot cycle, 55
classification of continuous blocks, 132–141
clock manager, 323
closed loop, 18, 19
closed-type CNC system, 387
CMM, 4, 6

CNC, 7, 8, 21, 22
CNC architecture design, 315
CNC control loop, 17
CNC system – architecture, 348
CNC system – closed, 390
CNC system – communication data classification, 370
CNC system – components, 19
CNC system – modeling, 356
CNC system – progress, 29
code interpreter, 33
common bus type, 344
common element, 242, 245
communication – inter-process, 323
communication model, 242
compatibility, 391
compensation function, 50
compiling method, 233
complex fixed cycle, 305
computer aided programming technologies, 416
configuration model, 242
connectivity, 241
constant surface speed control function, 53
constructed geometry method, 300
context switching, 322
context switching time, 341
continuous mode, 126
contour control, 69, 160, 161
contour error, 160
control – contour, 160, 161
control – D, 164
control – derivative, 164
control – feedback, 179
control – feedforward, 171, 173–178, 182
control – PI, 164
control – point-to-point, 160, 161
control – position, 161
control – tracking, 160, 161
control system – PID, 157
controller – derivative, 164
controller – feedback, 161
controller – P, 161
controller – PI, 161
controller – PID, 162
conversational programming, 279, 283
convolution, 109
coordinate system, 40
CORBA, 416
corner machining cycle, 310
corner speed, 142, 144, 148
corner speed – acute angle, 142, 144
corner speed – speed difference, 144, 145
counter, 235
counting semaphore, 330
coupling, 16
CPU unit, 231, 232
create event service, 384
critical section, 334
current control loop, 159
curvature, 103
cutting, 3
cutting angle, 307
cutting condition database, 301
cutting edge angle, 307
cutting feature, 290
cutting machines, 3
cyclic task – high priority, 358
cyclic task – low priority, 358
cylindrical interpolation, 46

D control, 164
DA-BA-SA, 397
DC servo motor, 10
DDA, 70–73
DDA – algorithm, 77, 78
DDA – hardware, 75
DDA – integrator, 72, 76
DDA – interpolation, 73, 79
deadlock, 336
deceleration, 107–114
derivative control, 164
derivative controller, 164
derivative gain, 164
design of PC-NC and open CNC, 353
design of system kernel, 361
development of the machining cycle, 305
device manager, 323
digital differential analyzer, 70–73, 75, 76, 78, 79
digital filter, 109, 110
direct access method, 369
direct search, 82
direct search algorithm, 77, 78, 85
direct search interpolation, 84, 85
distributed system, 317
DPM, 359
drawing instruments, 4
drilling cycle, 297
drilling sequence, 312
driving motor and sensor, 9
driving system components, 8
DRV, 33, 34
dry run, 57
dual port memory, 359
dwell, 49
dwell function, 49
dynamic priority scheduling, 328

e-manufacturing, 397
EDM machines, 3
embedded motion controller, 353
embroidery machines, 4
encoder, 12
EPFL–Switzerland, 426
error – contour, 160
error – position, 160
error – trajectory, 160
error compensation module, 157
error handler, 63
ethernet, 354
Euler algorithm, 92, 93
Euler algorithm – improved, 92, 93
Euler method, 77, 96
event, 331
event handler, 377
event service, 385
event-driven scheduling, 328
exact stop, 49
exact stop mode, 126, 127
EXAPT, 281
executor, 62
executor basic commands, 256
executor implementation example, 254
executor programming sequence, 253
exponential-type Acc/Dec control, 117
exponential-type acc/dec control, 117
exponential-type acc/dec pulse profile, 111
EXPRESS schema, 404
extensibility, 391

FA, 4
face milling pattern, 56
FANUC 0 series, 350
FANUC 150i, 350
FAPT, 281
feature mode, 289
feed function, 48
feedback control, 159, 179
feedback control following error, 179
feedback controller, 161
feedforward, 59
feedforward control, 171, 173–178, 182
feedforward control following error, 182
feedrate, 69, 86
fine boring cycle, 55
fine interpolation, 96
fine interpolator, 203
fine interpolator – functions, 204
fine interpolator – implementation, 203
fine interpolator – input/output, 204
fine interpolator – verification, 205
first-come, first-served scheduling, 327
fixed cycle function, 53
fixed sample time scheduling, 328
fixed-priority scheduling, 328
flexibility, 391
flexible coupling, 16
FMS, 4
following error, 179, 183
following error analysis, 179
full open CNC, 393
function block diagram – FBD, 246, 247
functional instruction, 250
functions – ACCDEC_Expo, 202
functions – ACCDEC_Expo_B0, 202
functions – ACCDEC_Expo_ES, 202
functions – ACCDEC_Linear, 201
functions – ACCDEC_Linear_B0, 201
functions – ACCDEC_Linear_ES, 201
functions – ACCDEC_Scurve, 201
functions – ACCDEC_Scurve_B0, 202
functions – ACCDEC_Scurve_ES, 201
functions – ARoughInterpolation, 198
functions – CircleNormalBlock, 220
functions – CircleSmallBlock, 220
functions – CircularIPO_Pre, 223
functions – CWCCWInterpolation, 199
functions – DetermineIBlockVelocity, 214
functions – DetermineVelocityBetweenCC, 215
functions – DetermineVelocityBetweenCL, 215
functions – DetermineVelocityBetweenLC, 214
functions – DetermineVelocityBetweenLL, 214
functions – DetermineVelocityProfile, 216
functions – ecal, 202
functions – FIPO, 204
functions – FIPO_Linear, 204
functions – FIPO_Moving, 205
functions – lcal, 202
functions – LinearInterpolation, 199
functions – LinearIPO_Pre, 223
functions – LineNormalBlock, 217
functions – LineSmallBlock, 218
functions – LookAhead, 214
functions – Mapping, 226
functions – POS, 209
functions – RoughInterpolation, 223
functions – scal, 202

G-code, 37, 397, 398, 431, 434, 437
G00, 43, 47
G01, 44, 47
G02, 44, 47
G03, 44, 47
G04, 49
G09, 48, 49

G15, 41
G31, 56
G33, 50
G40, 50
G41, 50
G42, 50
G43, 51
G44, 51
G49, 51
G50.1, 42
G51, 41
G51.1, 42
G54, 41
G55, 41
G56, 41
G57, 41
G58, 41
G59, 41
G61, 48, 126
G62, 49
G63, 49
G64, 48, 49, 126
G68, 42
G70, 54
G71, 54
G72, 54
G73, 54
G74, 54
G75, 54
G76, 54
G80, 54
G81, 54
G82, 54
G83, 54
G84, 54
G84.2, 54
G84.3, 54
G85, 54
G86, 54
G87, 54
G88, 54
G89, 54
G90, 41, 43, 45, 54
G91, 41, 43, 45
G92, 54
G94, 54
G96, 53
G97, 53
G98, 54
G99, 54
G&M code, 397
G&M code – difficult traceability, 398
G&M code – information loss, 397
G&M code – lack of interoperability, 398
G&M code – non-compatibility, 398
G&M-code interpreter, 62
gain – derivative, 164
gain – proportional, 163
gain – tuning, 166–168
gain – tuning automatic, 168, 169
gain – tuning Ziegler–Nichols, 167
Giddings and Lewis, 8
GPMC, 29
graphic representation, 234

hard real-time system, 319
hardware interpolator, 70
hardwired NC, 7
helical interpolation, 45
hierarchical structure, 273
hybrid loop, 19

I control, 162
ICS, 396
IEC1131, 241–245, 247
IEC1131-3, 27, 241–243, 247
IEC1131-3 PLC languages, 246
IEC1131-3 software model, 243
IKF, 175
improved Euler algorithm, 92, 93
improved Euler method, 96
improved Tustin algorithm, 95, 96, 195
IMS, 399
incremental-type encoder, 12
induction-type AC servo motor, 10
induction-type servo motor, 11
input unit, 230, 231
inspection, 4
instruction list – IL, 247
instruction list –IL, 246
intelligent and autonomous technologies, 415
intelligent STEP-CNC system, 418
inter-module communication, 371
inter-process communication, 323, 337, 338
inter-task communication, 381
InterBus-S, 22
interchangeability, 391
interference space angle, 307
interlock function, 237
internal block memory, 64
interoperability, 391
interpolation - sampled data, 77
interpolation errors, 77
interpolation functions, 42
interpolator, 24, 69–79, 81–106, 188
interpolator – hardware, 70–75
interpolator – implementation, 188
interpolator – input/output, 196

interpolator – software, 75–79, 81–90, 92–106
interpretative method, 232
interpreter, 24, 33
interpreter – execution, 191
interpreter – input/output, 192
interpreter – structure, 188
introduction to NC systems, 3
inverse compensation filter, 175
IPC, 323, 337, 338
ISO 10303, 399
ISO 14649, 396, 399
ISO 6983, 397
ISR, 323
ISW–Stuttgart, 424

Jacquard, 8
jig and fixture, 4

Kearney and Tracker, 8
kernel layer, 275, 276
key performance indices, 340

ladder diagram – LD, 234, 235, 246, 247, 253
language-type programming, 279, 280
latency time, 378
linear interpolation, 73
linear movement guide, 15
linear type acc/dec control, 112
linear type acc/dec pulse profile, 111
linear-circular overlap, 141, 142
LINUX, 356
LM guide, 15
loader, 27
local coordinate system, 40
look ahead, 57, 145–155
look ahead algorithm, 147
look ahead function, 49
look ahead module, 213
look-ahead module – functions, 214
look-ahead module – implementation, 213
look-ahead module – input/output, 213
loop cycle time, 361
loop driver mechanism, 366
loosely coupled type, 345
LSI, 7

M address, 37
M-code, 238, 397, 398
M02, 50
M19, 54
M30, 50, 53
machine coordinate system, 40
machine lock, 57
machine tool, 3
machine tool PLC programming, 235
machines – cutting, 3, 4
machines – EDM, 3
machines – embroidery, 4
machines – milling, 3
machines – mother, 3
machines – non-cutting, 3
machines – press, 3
machines – turning, 3
machines – woodworking, 4
machining center sequence flow, 240
machining cycle for arbitrary shape, 306
machining error, 121–124, 126
machining feature, 405, 406
machining geometry definition, 299
machining operation, 405, 406
machining operation cycle, 296
machining strategy, 290
machining strategy data, 301–303, 305
machining tool, 405
macro executor, 63
main program, 39
manual programming, 278
mapping – functions, 226
mapping module, 225
mapping module – input/output, 225
material removal rate, 420
maximum allowable acceleration, 144
maximum allowable error, 101
Mazatrol conversational system, 289
memory manager, 322
message system, 338
method for specifying part shape, 295
milling cycle, 298
milling machines, 3
MMC, 33, 34
MMI, 21, 22, 28, 29, 271–286, 288–311, 313
MMI – monitoring and alarm functions, 23
MMI – operation functions, 22
MMI – parameter setting functions, 23
MMI – program editing functions, 23
MMI – service and utility functions, 23
MMI function, 22, 271
MMI unit, 22
mnemonic, 234, 253
modal code, 37
modularity, 391
module – function, 360, 403
monotonic scheduling, 361
mother machines, 3
moving average method, 97
MPG, 273
MTB, 387
multi-processing hardware, 344

multi-processing system, 317
multi-programming system, 317
mutual exclusion, 335

NC, 7
NC machine tools – history, 7
NC machines, 3, 4
NC systems, 4
NCK, 21, 22, 24, 26–29, 33, 34, 109, 159, 187–226
NCK function, 23
NCK unit, 24
NIST–USA, 427
non-causal FIR, 176, 177
non-causal FIR filter, 177
non-cutting machines, 3
non-cyclic task, 357
non-pre-emption scheduler, 327
normal block, 130
numerical control kernel, 21, 22, 24, 26–29, 109, 159, 187–226
NURBS, 59–61, 99–101, 103, 105
NURBS – interpolation, 59, 98, 99, 102
NURBS – interpolation algorithm, 101
NURBS – surface machining, 61

OAC, 30
offline tasks, 4
offset cancel mode, 51
offset mode, 51
on-machine measurement, 420
online tasks, 4
open CNC system, 387, 389
open environment common interface controller, 392
open environment controller, 392
open loop, 19
open MMI, 392
open modular architecture controller, 392
open system interface, 375
operating system, 317
operating system configuration, 347
operation sequence control, 305
or – OR, 262
or not – ORN, 263
or stack – ORS, 267
oriented geometry method, 300
OS layer, 275, 277
output unit, 230
overlap between a linear and a circular profile, 141

P control, 162, 163
P controller, 161
painting, 3
parallel programming, 320
parser, 62
Parsons, 8
part program, 34–37, 39–42
part program for the milling operation, 411
part programming, 410
part programming for the turning operation, 414
partially open CNC, 392
path generator, 63
PC NC, 353
PC-based MMI, 275
performance – key indices, 340
PI control, 164
PI controller, 161
PID, 157, 162–166
PID controller, 162, 164–166
PID controller for the discrete time domain, 164
PLC, 21, 22, 24, 25, 27–29, 229–250, 253–269, 284
PLC – Executer, 27
PLC – loader, 27
PLC – program tasks, 364
PLC –programmer, 27
PLC compiler, 233
PLC configuration elements, 248
PLC element, 230
PLC function, 25
PLC program executor, 248, 362
PLC program interpolator, 233
PLC programmer, 250
PLC programming, 234, 238
PLC programming signal definition, 239
PLC system, 249
PLC system functions, 240, 249
PLC unit, 27
PMSM, 8
point-to-point control, 69, 160, 161
portability, 241, 391
position control, 160, 161
position control loop, 159
position controller, 24, 157, 208
position controller – functions, 209
position controller – implementation, 208
position controller – input/output, 209
position controller – verification, 209
position error, 160
post-line tasks, 4
post-processing, 398
Postech–Korea, 425
Pratt and Whitney, 8
pre-emption scheduler, 326

pre-emptive multi-tasking, 361
press machines, 3
priority, 381
priority scheduling, 327, 365
process coordinator, 322
process creation, 324
process management, 323
process manager, 322
process planning, 4, 278
process scheduling, 325
process state transition, 324
process synchronization, 330
process termination, 324
Profi-Bus, 22
PROFIBUS, 375
profile machining cycle, 297
program executor, 250
program structure, 35
program verification, 56
programmable logic control, 229–250, 253–269, 284
programming – automatic, 278, 280
programming – conversational, 279
programming – language-type, 279, 280
programming – manual, 278
programming – parallel, 320
programming – real-time, 320
programming – sequential, 320
programming language, 232, 234, 242, 244, 245
programming method comparison, 284
programming methods, 299, 300
programming model, 244
programming procedure, 292
proportional control, 162, 163
proportional gain, 163
punch press, 9

radial error, 90
rate monotonic, 328
read – RD, 256
read not – RDN, 257
read not stack – RDNS, 265
read stack – RDS, 264
real time extension, 356, 378
real-time control system, 28
real-time OS, 315, 316, 318, 320, 322, 325, 326, 329, 332, 333, 335, 339, 341–346, 348–351
real-time OS – structure, 321
real-time programming, 320
reference pulse interpolator, 76, 86
reference pulse method, 78
reference word interpolation, 90
reference word interpolator, 76, 87, 88
reference word interpolator for circles, 88
reference word interpolator for lines, 87
relay gain tuning, 168
relay method, 168
remaining pulse, 195
request/answer method, 369
resolver, 14
resource protection, 334
resources, 334
resources – system, 334
reusability, 391
ring buffer, 188, 338, 377
ring menu structure, 273
robots, 3
rough input, 196
rough interpolator, 193, 222
rough interpolator – circular interpolation, 195
rough interpolator – functions, 223
rough interpolator – implementation, 193, 222
rough interpolator – input/output, 222
rough interpolator – linear interpolation, 193
rough output, 198
RS 274, 397
RT LINUX, 356
RTOS, 315, 316, 318, 320, 322, 325, 326, 329, 332, 333, 335, 339, 341–346, 348–351
RTOS kernel, 321

S-code, 53, 238
S-shape type acc/dec control, 114
S-shape type acc/dec pulse profile, 111
Sabin, 427
sampled data interpolation, 77, 86, 96
scalability, 391
scaling function, 41
scheduling, 327, 328
scheduling – event-driven, 328
scheduling – first-come, first-served, 327
scheduling – fixed sample time, 328
scheduling – priority, 327
scheduling – time-slice, 327
self-waking thread, 369
semaphore, 330, 365
semaphore shuffling time, 341
semi-closed loop, 18
sequence of part programming, 278
sequential programming, 320
SERCOS, 22, 389
servo, 8, 10
servo controller, 158, 159
servo driving mechanism, 8
servo motor, 8, 10
SFC, 241, 245

SFP, 6, 421
shared memory, 337, 376, 383
shop floor programming, 415, 421
shopfloor programming, 6
short block, 130
Siemens 840C, 350
Siemens 840D, 350
signal, 331
simple fixed cycle, 305
single block, 57
SISO, 162
skip function, 56
soft bus, 30, 416
soft PLC, 247–249
soft real-time system, 319
Soft-NC, 30, 248, 353, 355, 357, 359, 362–364, 366–369, 372–377, 388, 392, 393, 416
software model, 242
softwired NC, 7
SOP, 284, 285
speed control loop, 159
speed feedforward controller, 177
speed profile, 129
speed profile generation, 129–132
speed sensor, 15
speed within block, 151
spindle, 9
spindle function, 53
spindle motor, 9
spindle orientation function, 53
spindle position function, 53
spline interpolation, 47
stack register, 254
stairs approximation, 77, 79, 82, 83
stairs approximation algorithm, 78
stairs approximation interpolator, 79
standard bus type, 344
standard communication protocol, 22
standard geometry method, 300
standardization, 241, 391
start-up mode, 50
statement list representation, 234
static priority scheduling, 328
STEP, 398, 399
STEP compliant CNC, 397
STEP manufacturing, 399
step response method, 167
STEP-CNC, 397, 415, 417
STEP-NC, 395–430
STEP-NC data model, 396
STEP-NC technology, 397
structure of a real-time OS, 321
structure of MMI system, 275
structured text – ST, 246, 247
subprogram, 39, 40
symbolic conversational system, 280
symmetry, 42
synchronous-type servo motor, 10, 11
system call, 321
system hardware architecture, 344
system resources, 334
system response, 361, 365

T-code, 238
tacho generator, 15
tapping machine, 9
task dispatch latency time, 341
task priority, 381
task scheduling, 28
task scheduling – priority, 28
task switching time, 340
task synchronization, 331, 365, 378
Taylor algorithm, 93, 96
Taylor method, 77
threading, 50
time sharing system, 317
time-slice scheduling, 327
timer, 235
timer handler, 377
tool database, 301
tool function, 50
tool length compensation function, 51
tool offset database, 301
tool radius compensation, 50
tool sequence database, 301
torque feedforward controller, 178
tracking control, 160, 161
trajectory error, 160
turning fixed cycle, 305
turning machines, 3
Tustin algorithm, 94–96
Tustin algorithm – improved, 95
Tustin method, 77
type of STEP-CNC, 417

ultimate sensitivity method, 167
United States Air Force, 8
user input, 299
user programming languages, 245

virtual mode, 355
VME bus, 349

Weck, 175
welding, 3
woodworking machines, 4
WOP, 6, 284
workingstep, 405

workpiece coordinate system, 40
workplan, 405
workshop oriented programming, 6, 284
write – WR, 258
write not – WRN, 259
WZL–Aachen, 422

Yasnac, 234

zero-phase error-tracking control, 174, 175
Ziegler–Nichols method, 166, 167
ZPETC, 174, 175