

Alexander Barkalov
Larysa Titarenko

Logic Synthesis for Compositional Microprogram Control Units

Lecture Notes Electrical Engineering

Volume 22

Alexander Barkalov · Larysa Titarenko

Logic Synthesis for Compositional Microprogram Control Units

 Springer

Prof. Dr. Alexander Barkalov
University of Zielona Gora
Inst. Computer Engin. and
Electronics
Podgorna Street 50
65-246 Zielona Gora
Poland
a.barkalov@iie.uz.zgora.pl

Dr. Larysa Titarenko
University of Zielona Gora
Inst. Computer Engin. and
Electronics
Podgorna Street 50
65-246 Zielona Gora
Poland
l.tittarenko@iie.uz.zgora.pl

ISBN: 978-3-540-69283-6

e-ISBN: 978-3-540-69285-0

Library of Congress Control Number: 2008930211

© 2008 Springer-Verlag Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: eStudio Calamar S.L.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Introduction

One of the very important parts of any digital system is the control unit, coordinating interplay of other system blocks. As a rule, control units have irregular structure, which makes process of their logic circuits design very sophisticated. In case of complex logic controllers, the problem of system design is reduced practically to the design of control units. Actually, we observe a real technical boom connected with achievements in semiconductor technology. One of these is the development of integrated circuit known as the "systems-on-a-programmable- chip" (SoPC), where the number of elements approaches one billion. Because of the extreme complexity of microchips, it is very important to develop effective design methods oriented on particular properties of logical elements. Solution of this problem permits improving functional capabilities of the target digital system inside single SoPC chip. As majority of researches point out, design methods used in case of industrial packages are, in case of complex digital system design, far from optimal. Similar problems concern the design of control units with standard field-programmable logic devices (FPLD), such as PLA, PAL, GAL, CPLD, and FPGA. Let us point out that modern SoPC are based on CPLD or FPGA technology. Thus, the development of effective design methods oriented on FPLD implementation of logic circuits used in the control units still remains the problem of great importance.

One possible way to optimise the control units (such characteristics as the size or performance of its logic circuit) is the adaptation of its structure to the particular properties of interpreted control algorithms. In our book, control algorithms are represented by the graph-schemes of algorithms (GSA). This choice is based on obvious fact that such specification provides simple explanation of methods proposed by the authors. We restrict ourselves to the case of the so called linear GSA, where the number of operator vertices is not less than 75% of the total number of all algorithm vertices. The compositional microprogram control units (CMCU) are proposed for interpretation of linear control algorithms. The compositional microprogram control unit has two blocks: the finite state machine, which addresses microprogram microinstructions, and the microprogram control unit including control memory, which keeps the microprogram corresponding to the initial GSA. The mode of natural addressing of microinstructions is used in the CMCU, where multidirectional

microprogram transitions depending on the arbitrary number of logical conditions are executed during one cycle of CMCU operation. This organisation permits to get a control unit with the following positive features:

1. The finite state machine of CMCU generates transitions addresses only if natural order of microinstructions' execution is violated. It permits to reduce the hardware amount, in comparison with other control units designed without assuming linearity of the initial GSA. This idea has been checked many times in case of control unit design with counters or shift registers used to keep the codes of FSM states.
2. The format of microinstructions in case of CMCU includes only the operational part and number of microinstructions kept in its control memory, which does not exceed the total number of operator vertices in the initial GSA. It permits to reach the minimum possible size of control memory, in comparison with other known models of microprogram control units.
3. Microprogram transitions are always executed during one cycle of CMCU operation. This time does not depend on the number of logical conditions determining a particular transition. It permits to minimize the average number of cycles needed for algorithm execution, in comparison with other models of microprogram control units.

Organization of the control unit proposed in the book increases regularity of the circuit, because the system of microoperations is implemented using standard blocks, such as PROM or RAM. At the same time, irregular part of the system described by means of Boolean functions is reduced. It permits to decrease the total number of logical elements (PAL, GAL, PLA, FPGA) in comparison with other models of finite state machines. Let us point out that the CMCU can be viewed as Moore FSM, in which codes of states are concatenations of information about the OLC code and address of microinstruction. These devices can be used to organize control processes not only in computers, but also in other digital systems, including complex controllers. The methods of synthesis and design presented in the book are not oriented to any particular set of logical elements, but to construction of tables describing the behaviour of CMCU blocks. These tables are used to find the systems of Boolean functions, which can be used to implement logic circuits of particular blocks. In order to implement corresponding circuits, this information should be transformed using data formats of particular industrial CAD systems. This step is beyond the scope of this book, in which the following information is presented:

Chapter 1 introduces such basic topics, as principles of microprogram control and specification of the control unit behaviour using the graph-scheme of algorithm. Next, some methods of control algorithm interpretation, such as finite-state machines (FSM) and microprogram control units (MCU), are discussed. Last part of the chapter is devoted to the organization principles of compositional microprogram control units, which can be viewed as compositions of finite-state machine and microprogram control unit. These control units provide efficient interpretation of the so-called linear GSA, in which long sequences of operator vertices can be found. These sequences are called operational linear chains (OLC). Microinstructions

corresponding to the components of OLC are addressed using the principle of natural microinstruction addressing. It permits to use the counter to keep microinstruction addresses and to simplify the combinational part of control unit, as compared with the classical Moore FSM. The Mealy FSM is used in CMCU to address microinstructions. It permits to calculate the transition address during one cycle of control unit's operation. Due to this feature, performance of the CMCU (proportional to the number of cycles needed to execute the control algorithm) is better than performance of the equivalent MCU with natural microinstruction addressing.

Chapter 2 discusses contemporary field-programmable logic devices and their evolution, starting from the simplest programmable logic devices, such as PLA, PAL, GAL and PROM, and finishing with very sophisticated chips such as CPLD and FPGA. This analysis shows particular features of different elements and permits to optimize the control unit logic circuits, in which some particular elements are used. The CMCU has some features of both FSM and MCU. Therefore main design and optimisation methods applied in case of these two types of control units are presented in the main part of the chapter.

Chapter 3 is devoted to the design and optimisation of some basic CMCU structures. Corresponding methods are discussed for the CMCU basic structure as well as for the CMCU with common memory. The problem of the set of operator vertices of the initial GSA is solved first. The resulted partition includes minimum possible number of OLCs. Next, the method of natural microinstruction addressing is discussed and its solution in case of CMCU given. It is shown that optimization methods used in the case of Moore FSM can be used to optimize the CMCU hardware amount. All methods presented in this book are oriented towards decreasing of hardware amount in the CMCU logic circuits.

Chapter 4 is devoted to the design methods based on the so-called code sharing, in which the microinstruction address is represented by concatenation of the OLC code and the code of its component. This representation makes possible using special methods of Moore FSM optimization adapted to the peculiarities of CMCU. In this case, the OLC codes are viewed as analogs of the states codes of Moore FSM. This approach permits to reduce the number of inputs and outputs of the combinational part of CMCU. Additional decrease can be achieved due to the application of elementary OLCs having only one input. In this case, the address of first component of each OLC is represented by all zeros and permits to diminish the number of functions generated by FSM.

Chapter 5 is devoted to optimization methods based on the transformation of codes. This method can be applied, when the FSM performing microinstruction addressing generates functions, which are subsequently loaded into two different memory blocks (the register and the counter). Some of these functions are treated as primary objects and the others as secondary objects. The FSM combinational part generates codes of primary objects and some additional variables only. This information is used to generate codes of secondary objects and permits to reduce the number of inputs in the block generating codes of secondary objects, in comparison with the initial combinational part of FSM and to reduce the hardware amount of resulting CMCU logic circuit.

Chapter 6 considers some optimisation methods used to reduce the size of CMCU control memory keeping the microprogram. These methods are based on the use of special address transformer permitting to keep the control memory size, which is the same as in case of the CMCU basic structure. One of the methods is oriented towards keeping only the original sets of microoperations and some additional variables in the control memory, in order to provide natural addressing and operation termination operating modes. The second approach assumes that a special CMCU block, which is not the part of its control memory, generates additional variables mentioned above. The methods proposed here permit to reduce control memory volume in comparison with the CMCU basic structure. Negative feature of this approach is decreasing of the CMCU performance because duration of the cycle becomes greater than in case of the CMCU basic structure.

Chapter 7 deals with multilevel implementation of CMCU logic circuits. These methods are based on some well-known ideas taken from the literature devoted to optimization of FSM and MCU. They are of course adapted to the particular conditions of the CMCU operation. All these methods lead to the increase of cycle time, in comparison with the CMCU basic structure. They can be applied, when minimum hardware amount is the main goal of a particular design.

Chapter 8 is devoted to CMCU optimization, based on modification of the microinstruction format. Proposed modifications permit to eliminate code transformers from the CMCU and provide reduction of hardware amount of circuits used in the FSM used for microinstruction addressing, as compared with the CMCU basic structure. This kind of optimisation leads to the increasing of the number of cycles, needed for execution of the control algorithms. This transformation causes sometimes the increase of control memory size. Next, the possibility of multilevel CMCU implementation is discussed and the method of optimal structure choice proposed. A particular CMCU structure is considered as optimal, if it guarantees minimum hardware amount and sufficient performance. This chapter is based on the results of common research performed with J. Bieganowski (Poland).

We hope that our book will be interesting and useful for students and postgraduates in the area of Computer Science and for designers of modern digital devices. We think that compositional microprogram control units enlarge the class of models applied for implementation of control units with modern field-programmable logic devices.

Acknowledgements

Several people helped us with preparation of this manuscript. Our PhD students Mr Jacek Bieganowski and Mr. Sławomir Chmielewski worked with us on initial planning of this work, distribution of tasks during the project, and final assembly of this book.

We also thank Professor Marian Adamski for his support and special attention to this work. His guidelines in making this book useful for students and practitioners were very helpful in the organization of this book.

Contents

1	Methods of interpretation of control algorithms	1
1.1	Principle of microprogram control	1
1.2	Control algorithm interpretation with finite state machines	4
1.3	Control algorithm interpretation with microprogram control units ..	11
1.4	Organization of compositional microprogram control units	22
	References	26
2	Synthesis of control units with field-programmable logic devices	27
2.1	Evolution of field-programmable logic devices	27
2.2	Optimization of microprogram control units	34
2.3	Optimization of Mealy finite state machines	42
2.4	Optimization of Moore finite state machines	50
2.5	Control unit design with FPLDs	56
	References	59
3	Synthesis of basic circuits of compositional microprogram control units	65
3.1	Synthesis of compositional microprogram control unit with basic structure	65
3.2	Synthesis of CMCU with common memory	72
3.3	Optimization of CMCU with common memory logic circuit	81
	References	98
4	Synthesis of compositional microprogram control units with code sharing	99
4.1	Synthesis of CMCU basic model with code sharing	99
4.2	Optimization of logic circuit of CMCU with code sharing	107
4.3	Synthesis of CMCU with elementary operational linear chains	120
4.4	Logic circuit optimization for CMCU with elementary OLC	125
	References	134

5	Synthesis of compositional microprogram control units with object transformation	137
5.1	Optimization principles for CMCU with object transformation	137
5.2	Objects transformation for CMCU with basic structure	140
5.3	Object transformation in CMCU with codes sharing	149
	References	157
6	Control memory optimization for CMCU with code sharing	159
6.1	Principles of control memory optimization	159
6.2	Synthesis of CMCU with generation of microinstruction addresses .	164
6.3	Synthesis of CMCU with addressing of expanded microinstructions	171
6.4	Synthesis of CMCU with generation of addresses of collections of microoperations	180
6.5	Combined application of different object transformation methods for CMCU	188
	References	196
7	Synthesis of CMCU with coding of logical conditions and collections of microoperations	197
7.1	Coding of logical conditions for CMCU with basic structure	197
7.2	Encoding of logical conditions for basic models of CMCU	205
7.3	Encoding collections of microoperations in CMCU	215
7.4	Synthesis of multilevel circuits of CMCU	225
	References	234
8	Synthesis of CMCU with modified system of microinstructions	235
8.1	Synthesis of CMCU with dedicated area of inputs	235
8.2	Optimization of compositional microprogram control unit with the dedicated input area	246
8.3	Minimization of the number of feedback signals in CMCU	255
8.4	Synthesis of multilevel circuits for CMCU with modified system of microinstructions	263
	References	267
	Conclusion	269
	References	270
	Index	271

Symbols

Γ	graph–scheme of algorithm
b_0	start vertex of GSA
b_E	end vertex of GSA
B_1	set of GSA operator vertices
B_2	set of GSA conditional vertices
$E = \{\langle b_t, b_q \rangle\}$	set of GSA arcs
$X = \{x_1, \dots, x_L\}$	set of logical conditions
$Y = \{y_1, \dots, y_N\}$	set of microoperations
$Y_q \subseteq Y$	collection of microoperations
$a_m \in A$	internal state of FSM
$K(a_m)$	code of internal state $a_m \in A$
$\Phi = \{\varphi_1, \dots, \varphi_R\}$	set of input memory functions
A_m	conjunction of state variables $T_r \in T$ corresponding to the code of state $a_m \in A$
$\Pi_A = \{B_1, \dots, B_I\}$	partition of set A into the classes of pseudoequivalent states
FY	operational part of the microinstruction, which contains information about microoperations to be executed
FX	field of logical conditions with information about logical condition $x_t^j \in X$, which is checked at time t
y_E	special signal used to terminate a control unit operation
$\alpha_g = \langle b_{g_1}, \dots, b_{g_{F_g}} \rangle$	operational linear chain of GSA Γ
D^g	set of operator vertices, which are components of OLC α_g
I_g^j	input j of OLC α_g
O_g	output of OLC α_g
$C = \{\alpha_1, \dots, \alpha_G\}$	set of operational linear chains of GSA Γ
$I(\Gamma)$	set of inputs of operational linear chains of GSA Γ
$O(\Gamma)$	set of outputs of operational linear chains of GSA Γ
$A(I_g^j)$	address of the input j of OLC $\alpha_g \in C$
$\tau = \{\tau_1, \dots, \tau_{R_1}\}$	set of state variables encoding states of CMCU addressing FSM

$T = \{T_1, \dots, T_{R_2}\}$	outputs of the counter CT, which determine next microinstruction address for CMCU
$M(\Gamma)$	set of main inputs of GSA Γ
$U_i(\Gamma_j)$	compositional microprogram control unit U_i which interprets GSA
R_{FB}^i	the number of feedback inputs of the combinational circuit of CMCU U_i
$R_{FB}^i(\Gamma_j)$	the corresponding symbol for R_{FB}^i in case of CMCU $U_i(\Gamma_j)$
$H_i(\Gamma_j)$	the total number of terms in systems of Boolean functions implemented by combinational circuit CC of CMCU $U_i(\Gamma_j)$
$S_i(\Gamma_j)$	the number of input variables of combinational circuit CC of CMCU $U_i(\Gamma_j)$
$t_i(\Gamma_j)$	the number of output variables of combinational circuit CC of CMCU $U_i(\Gamma_j)$
$\Pi_c = \{B_i, \dots, B_l\}$	partition of the set $C' \subset C$ by classes of pseudoequivalent operational linear chains of GSA Γ
$C^1 \subseteq C$	set of OLC, such that their outputs have no direct connections with the final vertex of transformed GSA
$\Gamma_j(U_i)$	graph-scheme of algorithm transformed for interpretation by CMCU U_i
$K(\alpha_g)$	code of operational linear chain $\alpha_g \in C$
C_E	set of elementary operational linear chains of GSA Γ
$IM_E(\Gamma)$	set of main inputs of elementary operational linear chains

Abbreviations

AMP	area of microprogram
AT	address transformer
CAMI	counter of microinstruction address
CC	combinational circuit
CCS	special combinational circuit which generates variables y_0 and y_E
CFA	sequencer calculating transition address (address of the next microinstruction to be executed)
CM	control memory of control unit
CMCU	compositional microprogram control unit
CMO	block for generation of microoperations
CPLD	Complex Programmable Logic Devices
CT	counter
DC	decoder
DAI	dedicated input area
EOLC	elementary operational linear chain
FSM	finite state machine
FPGA	field-programmable gate arrays
FPLD	field-programmable logic devices
GAL	generic array logic
GSA	graph-scheme of algorithm
LGSA	linear graph-scheme of algorithm
LUT	look-up table
MCU	microprogram control unit
MX	multiplexer
OLC	operational linear chain
PAL	programmable array logic
PLA	programmable logic array
PLD	programmable logic device
PLS	programmable logic sequencers
ROM	programmable read-only memory
RG	register

RAM	random-access memory
RAMI	register of microinstruction address
ROM	read-only memory
SBF	system of Boolean functions
SOP	sum of products (disjunctive normal form)
SoPC	system-on-a-programmable chip
ST	FSM structure table
STF	system of transition formulae
TAS	address transformer of microinstruction address to state code
TC	transformer of code
TF	fetch flip-flop used to organize the stop mode of a control unit
TOK	transformer of OLC code in the OLC component code
TSA	code transformer from state code to microinstruction address
VGSA	vertical graph-scheme of algorithm

Chapter 1

Methods of interpretation of control algorithms

Abstract The chapter introduces such basic topics, as principles of microprogram control and specification of the control unit behavior using the graph-scheme of algorithm. Next, some methods of control algorithm interpretation, such as finite-state machines (FSM) and microprogram control units (MCU), are discussed. Last part of the chapter is devoted to the organization principles of compositional microprogram control units, which can be viewed as compositions of finite-state machine and microprogram control unit. These control units provide efficient interpretation of the so-called linear GSA, in which long sequences of operator vertices can be found. These sequences are called operational linear chains (OLC). Microinstructions corresponding to the components of OLC are addressed using the principle of natural microinstruction addressing. It permits to use the counter to keep microinstruction addresses and to simplify the combinational part of control unit, as compared with the classical Moore FSM. The Mealy FSM is used in CMCU to address microinstructions. It permits to calculate the transition address during one cycle of control unit's operation. Due to this feature, performance of the CMCU (proportional to the number of cycles needed to execute the control algorithm) is better than performance of the equivalent MCU with natural microinstruction addressing.

1.1 Principle of microprogram control

The overwhelming majority of digital systems are organized using the principle of microprogram control [12]. This principle was proposed by M. Wilkes in 1951 and could be explained as follows [1]:

1. Any operation, executed by a digital device, is considered as a complex action, which is represented as the sequence of elementary operations (microoperations) on the words of information (operands).
2. The logical conditions (status signals) are used to control the order of microoperation executions. The values of logical conditions are calculated as some Boolean functions depending on the values of operands.

3. Execution of operations in a digital device is specified by a control algorithm, which is represented in terms of logical conditions and microoperations. It is called "microprogram". Microprogram determines an order of testing the values of logical conditions and sequences of microoperations, which are necessary to get proper operation of the device.
4. The microprogram is used as a particular form of specification of the function of a device and determines the structure of digital device and the order of its time operation sequence.

The principle of microprogram control was developed by Victor Glushkov [7], who proposed representation of any digital system as composition of data-path and control automaton (Fig. 1.1).

The data-path of a digital system receives and keeps words of information (D) to be processed, executes the microoperations Y on these words (operands), estimates the values of logical conditions X and calculates the results of operations R. Control automaton (CA) provides the required time order of distribution of microoperations on the base of the microprogram to be executed. This order is determined by the operation code F and by the values of logical conditions X, which are calculated by the data-path. Moreover, the control automaton cooperates with environment through the input signals CI and output signals CO. One of the input signals, called "Start", serves to initialize the execution of operations determined by the code F. One of the output signals, called "Done", is needed to indicate that a given operation is executed. In the literature, control automaton is called "control unit" [10], and we use this term in our book.

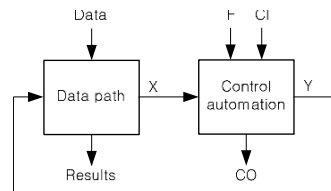


Fig. 1.1 Structure of digital system

The main function of control unit is the analysis of operation code and of the values of logical conditions $X = \{x_1, \dots, x_L\}$. As the result of analysis, microinstructions $Y_q \subseteq Y$ are produced, where $Y = \{y_1, \dots, y_N\}$ is the set of microoperations initializing data processing. The algorithm executed by operational unit is specified using one of the formal methods [3,21]; for example, the language of graph-schemes of algorithms (GSA) has been widely used in cases of practical designs [3]. There are many other approaches for the control algorithm representation. We use the language GSA, because it provides easy explanation for all design methods presented in this book. The second reason of our choice is the existence of simple way to pass from GSA to any other form of control algorithm representation.

Graph-scheme of algorithm Γ is the directed connected graph, characterized by a finite set of vertices including four types of vertices, namely (Fig. 1.2): start (initial vertex), end (final vertex), operator and conditional vertices. These vertices are connected by some paths called arcs.

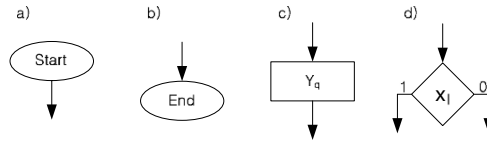


Fig. 1.2 Types of vertices

The start vertex, denoted here by the symbol b_0 , corresponds to the beginning of control algorithm and has no input. The end vertex, denoted here by symbol b_E , corresponds to the end of control algorithm and has no output. The operator vertex $b_l \in B_1$, where B_1 is a finite set of GSA operator vertices, contains a collection of microoperations $Y_q \subseteq Y$. The conditional vertex $b_l \in B_2$, where B_2 is a finite set of GSA conditional vertices, which includes single element $x_l \in X$. It has two outputs, first corresponding to value "1" and second to value "0" of the logical condition. Thus, GSA Γ is characterized by a finite set of vertices $B = B_1 \cup B_2 \cup \{b_0, b_E\}$, connected by arcs taken from a finite set $E = \{ \langle b_l, b_q \rangle \}$, where $b_l, b_q \in B$.

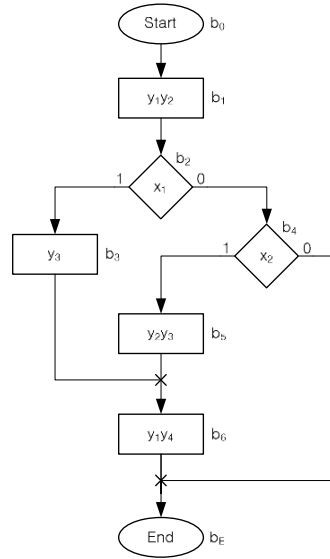
For example, the GSA (Fig. 1.3) is characterized by the following sets:

- the set of vertices $B = \{b_0, b_1, \dots, b_6, b_E\}$;
- the set of arcs $E = \{ \langle b_0, b_1 \rangle, \langle b_1, b_2 \rangle, \dots, \langle b_6, b_E \rangle \}$;
- the set of microoperations $Y = \{y_1, \dots, y_4\}$;
- the set of logical conditions $X = \{x_1, x_2\}$.

The control algorithm can be implemented either as a program (program interpretation) or as a network of logical elements connected in some way (hard-wired interpretation). In this book we discuss the second way of control algorithm implementation. These methods are based either on the model of a finite state-machine (called sometimes structural automaton or an automaton with hard-wired logic) or on the principle of keeping the microprogram in a special control memory (automaton with programmed logic) [7, 17].

Methods of data-path design are not discussed in this book. They could be found in many fundamental books, as for example, in [1, 12, 14]. Let us discuss the classical methods of control units design.

Fig. 1.3 Graph-scheme of algorithm Γ_1



1.2 Control algorithm interpretation with finite state machines

The finite state machine (FSM) [3] implementing control algorithm is represented by classical model of sequential circuit, which can be treated as the composition of combinational circuit CC and register RG (Fig. 1.4).

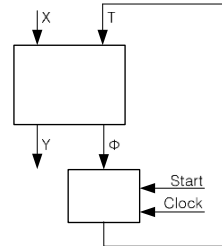


Fig. 1.4 Structural diagram of finite state machine

Presence of register RG in this structure can be explained in the following way. A time-distributed microinstruction sequence $Y(0), Y(1), \dots, Y(t)$, where t is the automaton time determined by synchronization pulse "Clock", appears on the output of FSM. The initial time $t = 0$ is determined by a single-shot pulse "Start". To produce such a sequence, some information about history of the system operation is needed. This sequence is determined by input signals $X(0), \dots, X(t - 1)$ for previous moments of time. It means that output signal $Y(t)$ at time t is determined by the following formula

$$Y(t) = f(X(0), \dots, X(t - 1), X(t)). \tag{1.1}$$

Expressions of this kind are very cumbersome and could not be realized in hardware, especially if they contain cycles with unpredictable number of iterations. In practice, the history is described by special internal states of the FSM, from the set of states $A = \{a_1, \dots, a_M\}$. States $a_m \in A$ are encoded by binary codes $K(a_m)$ having not less than

$$R = \lceil \log_2 M \rceil \quad (1.2)$$

bits, where $\lceil A \rceil$ is the least integer, greater than or equal to A . This function is known as a ceil function or ceiling [14]. Elements of the set of state variables $T = \{T_1, \dots, T_R\}$ are used to encode the states of FSM. The code of current state is kept in register RG, which includes R synchronous flip-flops with common timing signal "Clock". The code of initial state $a_1 \in A$ is loaded into RG using pulse "Start". The content of RG can be changed by pulse "Clock" on the base of input memory functions, which form the set of input memory functions $\Phi = \{\phi_1, \dots, \phi_R\}$. As a rule, the register RG is implemented using D flip-flops [14, 20].

The combinational circuit CC produces the following input memory functions

$$\Phi = \Phi(T, X) \quad (1.3)$$

and the output functions, which depend strongly on the FSM model in use [3]. In case of the Mealy FSM, output functions Y are represented as

$$Y = Y(T, X). \quad (1.4)$$

In case of the Moore FSM, output functions depend only on the states variables:

$$Y = Y(T). \quad (1.5)$$

The method of FSM logic circuit implementation on the base of GSA includes the following steps [3]:

- construction of marked GSA Γ ;
- state assignment (encoding of the states);
- construction of the structure table of FSM;
- construction of systems Φ and Y using the structure table of FSM;
- implementation of FSM logic circuit using some logical elements.

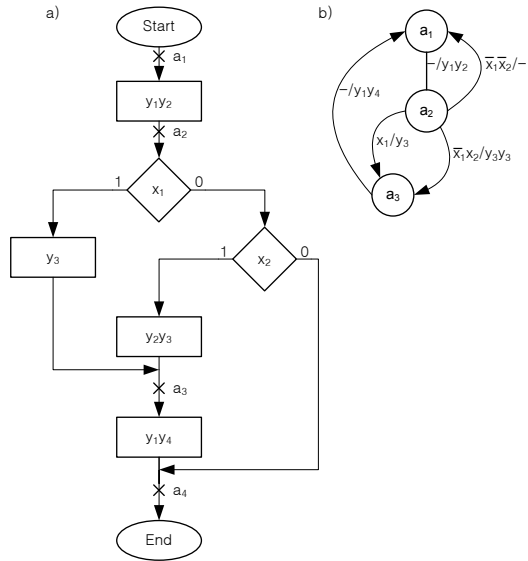
Let us discuss some examples of FSM synthesis using GSA Γ_1 to represent the control algorithm.

In case of the Mealy FSM, marked GSA is constructed using the following rules [3]:

- the output of the vertex b_0 and input of the vertex b_E are marked by an initial state a_1 ;
- inputs of vertices $b_i \in B$, connected with outputs of operator vertices, are marked by unique states a_2, \dots, a_M (only once for each input).

Application of this procedure to GSA Γ_1 leads to the marked GSA Γ_1 (Fig. 1.5a), corresponding to the graph of Mealy FSM S_1 (Fig. 1.5b).

Fig. 1.5 Marked GSA Γ_1 **a** and graph of Mealy FSM S_1 **b**



The vertices of this graph correspond to the states of FSM S_1 , whereas its arcs correspond to the transitions between states. Each arc is marked by a pair (input signal, output signal). Input signal X_h ($h = 1, \dots, H$) corresponds to some conjunction of variables from the set X (direct values of variables or their complements). Output signal $Y_h \subseteq Y$ corresponds to some microoperations $y_n \in Y$, which are written into an operator vertex of the transition h of Mealy FSM ($h = 1, \dots, H$). It follows from analysis of the marked GSA Γ_1 that the FSM S_1 is characterized by sets $X = \{x_1, x_2\}$, $Y = \{y_1, \dots, y_4\}$, $A = \{a_1, a_2, a_3\}$ and has $H = 5$ transitions.

There are many methods of state encoding [3, 5, 14, 20, 22], which are oriented on optimization of hardware amount of the logic circuit CC. Their applications depend strongly on logical elements used to implement this circuit. The problem is known as the state assignment [14]. Let us try a trivial encoding approach first, using minimum possible number of state variables to encode the states. In case of the FSM S_1 we have $M = 3$, $R = 2$, $T = \{T_1, T_2\}$. The states are encoded as: $K(a_1) = 00$, $K(a_2) = 01$ and $K(a_3) = 10$. Let us point out that the code $K(a_1)$ of initial state $a_1 \in A$ should include R zeros to simplify the initialization of FSM operation by the pulse "Start".

An FSM structure table (ST) can be viewed as the FSM graph represented by a list of interstate transitions. This table contains an additional column Φ_h with input memory functions. They are equal to 1, in order to change the states of particular FSM memory flip-flops. This table includes the following columns [3]: a_m is the current state of FSM; $K(a_m)$ is the code of state; $a_s \in A$ is the next state of FSM (state of transition); $K(a_s)$ is the code of this state; X_h is the input signal causing transition $\langle a_m, a_s \rangle$; Y_h is the output signal produced during the transition $\langle a_m, a_s \rangle$; Φ_h is the collection of input memory functions, which are equal to 1 in order to

change the register content from $K(a_m)$ into $K(a_s)$; $h = \overline{1, H}$ is the current transition number.

Structure table is constructed in trivial way using the automaton graph; in case of FSM S_1 , this table contains $H = 5$ lines (Table 1.1).

Table 1.1 Structure table of Mealy FSM S_1

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Y_h	Φ_h	h
a_1	00	a_2	01	1	y_1y_2	D_2	1
a_2	01	a_3	10	x_1	y_3	D_1	2
		a_3	10	\bar{x}_1x_2	y_2y_3	D_1	3
		a_1	00	$\bar{x}_1\bar{x}_2$	y_1y_4	-	4
a_3	10	a_1	00	1	-	-	5

Functions (1.3) - (1.4) are derived from the automaton structure table as the sums of products (SOP) depending on the following product terms:

$$F_h = A_m X_h (h = 1, \dots, H). \quad (1.6)$$

In this formula A_m is the conjunction of state variables $T_r \in T$ corresponding to the code of state $a_m \in A$ from line h of the structure table:

$$A_m = T_{l_{m1}}^1 \dots T_{l_{mR}}^R, \quad (1.7)$$

where $l_{mr} \in \{0, 1\}$ is the value of bit r in the code $K(a_m)$, $T_r^0 = \bar{T}_r$, $T_r^1 = T_r$ ($r = 1, \dots, R; m = 1, \dots, M$). Systems (1.3) - (1.4) are represented in the form:

$$\varphi_r = \bigvee_{h=1}^H C_{rh} F_h (r = 1, \dots, R); \quad (1.8)$$

$$y_n = \bigvee_{h=1}^H C_{rh} F_h (r = 1, \dots, N), \quad (1.9)$$

where $C_{rh}(C_{nh})$ is a Boolean variable equal to one, iff (if and only if) the line h of the structure table includes the variable $\varphi_r(y_n)$.

For example, from Table 1.1 we get the following equation system: $F_1 = \bar{T}_1 \bar{T}_2$; $F_2 = \bar{T}_1 T_2 x_1$; $F_3 = \bar{T}_1 T_2 \bar{x}_1 x_2$; $F_4 = \bar{T}_1 T_2 \bar{x}_1 \bar{x}_2$; $F_5 = \bar{T}_1 \bar{T}_2$; $y_1 = F_1 \vee F_4$; $y_2 = F_1 \vee F_3$; $y_3 = F_2 \vee F_3$; $y_4 = F_4$; $D_1 = F_2 \vee F_3$; $D_2 = F_1$.

Implementation of FSM circuit depends strongly on particular properties of logical elements in use. This step will be considered later.

The marked GSA of Moore FSM is constructed using the following rules [3]:

- the vertices b_0 and b_E are marked by an initial state a_1 ;
- the operator vertices $b_t \in B_1$ are marked by the unique states a_2, \dots, a_M .

Application of this procedure to the GSA Γ_1 results in producing the marked GSA Γ_1 (Fig. 1.6a) corresponding to the state diagram of Moore FSM (Fig. 1.6b). It

follows from Fig. 1.6b that the vertices of Moore FSM graph are marked by output signals $y_n \in Y$ and the graph arcs are marked only by input signals, which determine interstate transitions. As we can see, the Moore FSM S_2 is described by sets $X = \{x_1, x_2\}$, $Y = \{y_1, \dots, y_4\}$, $A = \{a_1, \dots, a_5\}$, and has $H = 7$ transitions.

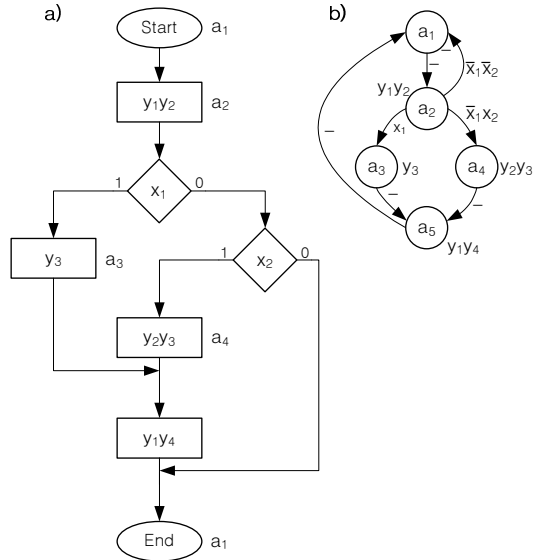


Fig. 1.6 Marked GSA Γ_1 **a** and graph of Moore FSM S_2 **b**

In case of the Moore FSM S_2 , $R = 3$, $T = T_1, T_2, T_3$. Let us encode the states $a_m \in A$ as: $K(a_1) = 000, \dots, K(a_5) = 100$. The structure table of Moore FSM is constructed using either an automaton graph (state diagram) or a marked GSA. This table includes the columns: a_m , $K(a_m)$, $K(a_s)$, X_h , Φ_h , h . Information about output signals is placed into the column a_m [3]. The structure table of the Moore FSM S_2 includes $H = 7$ lines (Table 1.2).

Table 1.2 Structure table of Moore FSM S_2

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Φ_h	h
$a_1(-)$	000	a_2	001	1	D_3	1
$a_2(y_1y_2)$	001	a_3	010	x_1	D_2	2
		a_4	011	\bar{x}_1x_2	D_2D_3	3
		a_1	000	$\bar{x}_1\bar{x}_2$	-	4
$a_3(y_3)$	010	a_5	100	1	D_1	5
$a_4(y_2y_3)$	011	a_5	100	1	D_1	6
$a_5(y_1y_4)$	100	a_1	000	1	-	7

Equation systems (1.3) and (1.5) are found using the structure table; system (1.3) depends on product terms (1.6) and its SOP is similar to system (1.8). Functions (1.5) can be represented in the form

$$y_n = \bigvee_{m=1}^M C_{nm} A_m (n = 1, \dots, N), \quad (1.10)$$

where C_{nm} is the Boolean variable equal to one, iff microoperation $y_n \in Y$ is executed, when FSM is in the state $a_m \in A$.

For example, in case of Moore FSM S_2 , the following system of expressions can be extracted from Table 1.2: $F_1 = \bar{T}_1 \bar{T}_2 \bar{T}_3$, $F_2 = \bar{T}_1 \bar{T}_2 \bar{T}_3 x_1, \dots, F_7 = T_1 \bar{T}_2 \bar{T}_3$; $D_1 = F_5 \vee F_6$; $D_2 = F_2 \vee F_3$; $D_3 = F_1 \vee F_3$; $A_1 = \bar{T}_1 \bar{T}_2 \bar{T}_3 \dots$, $A_5 = T_1 \bar{T}_2 \bar{T}_3$; $y_1 = A_2 \vee A_5$; $y_2 = A_2 \vee A_4$; $y_3 = A_3 \vee A_4$; $y_4 = A_5$.

Automata S_1 and S_2 are equivalent in the sense that they interpret the same GSA Γ_1 . Comparison of automata S_1 and S_2 leads to the following conclusions satisfied for all equivalent Mealy and Moore automata:

- Moore FSM has, as a rule, more states and transitions than the equivalent Mealy FSM;
- system of output signals of Moore FSM has regular form, because it depends only on the states of FSM.

Let us point out that model of Moore FSM is used more often in practical design [20] because it offers more stable control units than the control units based on the Mealy FSM model. Moreover, system (1.5) is regular, what means that it is specified for more than 50% of all possible input assignments. This regularity makes possible implementation of this system using either read-only memory (ROM) chips or random-access memory (RAM) blocks [6, 8].

The number of product terms in the input memory functions system can be reduced due to the existence of pseudoequivalent states of Moore FSM [5, 8]. The states $a_m, a_s \in A$ are called pseudoequivalent states of Moore FSM, if there exist the arcs $\langle b_i, b_t \rangle, \langle b_j, b_t \rangle \in E$, where vertex $b_i \in B_1$ is marked by state $a_m \in A$ and vertex $b_j \in B_1$ by state $a_s \in A$. Thus, the states a_3 and a_4 of the Moore FSM S_2 are pseudoequivalent states. They can not be treated as equivalent states [3], because of different output signals generated by these states. As follows from Table 1.2, the columns a_s of structure table for the states a_3 and a_4 contain the same information.

Let $\Pi_A = \{B_1, \dots, B_I\}$ be a partition of set A into the classes of pseudoequivalent states. For example, in case of Moore FSM S_2 we have $\Pi_A = \{B_1, \dots, B_4\}$, with $B_1 = \{a_1\}$, $B_2 = \{a_2\}$, $B_3 = \{a_3, a_4\}$, $B_4 = \{a_5\}$. The number of terms in system Φ can be reduced due to optimal state encoding [5], when the codes of pseudoequivalent states from some class $B_i \in \Pi_A$ belong to a single generalized interval of an R -dimensional Boolean space. For example, the well-known algorithms NOVA [14] or ASYL [18, 19] can be used for the state encoding mentioned above.

The optimal state encoding for the Moore FSM S_2 is shown in the Karnaugh map of Fig. 1.7.

As follows from Fig. 1.7, the class B_1 corresponds to the interval $K(B_1) = 000$, $B_2 \rightarrow K(B_2) = *01$, $B_3 \rightarrow K(B_3) = *1*$, $B_4 \rightarrow K(B_4) = 1**$, where sign "*" is

Fig. 1.7 Optimal state encoding for the Moore FSM S_2

		T_2T_3			
		00	01	11	10
T_1	0	a_1	a_2	a_3	a_4
	1	a_5	*	*	*

determines "don't care" value of state variable $T_r \in T$. These intervals can be considered as the codes of classes $B_i \in \Pi_A$. Let us construct a transformed structure table of Moore FSM with the following columns: $B_i, K(B_i), a_s, K(a_s), X_h, \Phi_h, h$. To do this we replace the column a_m by the column B_i , and the column $K(a_m)$ by the column $K(B_i)$. If the structure table transformed in this way contains equal lines, only one of them should find place in the final transformed table. For example, the transformed structure table of Moore FSM S_2 (Table 1.3) contains $H = 6$ lines.

The transformed structure table serves as the base to form product terms (1.6), but now these terms include variables $l_{mr} \in \{0, 1, *\}$, where $T_r^0 = \bar{T}_r, T_r^1 = T_r, T_r^* = 1$ ($m = 1, \dots, M; r = 1, \dots, R$). Presence of "don't care" input assignments makes possible reduction of the number of product terms in system (1.8), which has only H_0 terms. Thus, in case of the Moore FSM S_2 we have: $F_1 = \bar{T}_1\bar{T}_2\bar{T}_3; F_1 = \bar{T}_2T_3x_1; F_3 = \bar{T}_2T_3\bar{x}_1x_2; F_4 = \bar{T}_2T_3\bar{x}_1\bar{x}_2; F_5 = T_2; F_6 = T_1$.

We find that terms F_4 and F_6 are not the parts of SOP (1.8).

Table 1.3 Transformed structure table of Moore FSM S_2

B_i	$K(B_i)$	a_s	$K(a_s)$	X_h	Φ_h	h
B_1	000	a_2	001	1	D_3	1
B_2	*01	a_3	011	x_1	D_2D_3	2
		a_4	010	\bar{x}_1x_2	D_2	3
		a_1	000	$\bar{x}_1\bar{x}_2$	-	4
B_3	*1*	a_5	100	1	D_1	5
B_4	1**	a_1	000	1	-	6

It was shown in [4] that optimal state encoding permits to compress the transformed structure table of Moore FSM up to corresponding size of the equivalent Mealy FSM structure table.

As a rule, models of FSM are used for implementation of fast operational units [1]. If system performance is not important for a project, the control unit can be implemented as a microprogram control unit (MCU).

1.3 Control algorithm interpretation with microprogram control units

In 1951 M. Wilkes proposed to use the intermediate level for execution of computer program instructions. This level was called microprogram level or firmware. Each instruction of the high-level programming language was interpreted here by a special microprogram. These microprograms were kept in the separate control memory (CM) as a sequence of microinstructions. The microinstructions are organized according to the operational-address principle [1, 10]. These microprograms are interpreted by microprogram control units (MCU) [7]. The typical method of MCU design includes the following steps [7, 8]:

- transformation of initial graph-scheme of algorithm;
- generation of microinstructions with given format;
- microinstruction addressing;
- encoding of operational and address parts of microinstructions;
- construction of control memory content;
- synthesis of logic circuit of MCU using given logical elements.

The mode of microinstruction addressing affected tremendously the method of MCU synthesis [7]. Three particular addressing modes are used most often nowadays [1]:

- compulsory addressing of microinstructions;
- natural addressing of microinstructions;
- combined addressing of microinstructions.

As a rule, microinstruction formats include the following fields: FY , FX , FA_0 and FA_1 . The field FY , operational part of the microinstruction, contains information about microoperations $y_n \in Y (t = 0, 1, \dots)$, which are executed in cycle t of control unit operation. The field FX contains information about logical condition $x_i^t \in X$, which is checked at time $t (t = 0, 1, \dots)$. The field FA_0 contains next microinstruction address A^{t+1} (transition address), either in case of unconditional transition (go to type), or if $x_i^t = 0$. The field FA_1 contains next microinstruction address for the case when $x_i^t = 1$. The fields FX , FA_0 and FA_1 form the address part of microinstruction.

Consider an example of MCU design with compulsory microinstruction addressing S_3 interpreting FSA I_1 (Fig. 1.3). The microinstruction format is shown in Fig. 1.8.

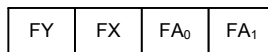


Fig. 1.8 Format of microinstructions with compulsory addressing

The address of next microinstruction A^{t+1} is determined by contents of the fields $[FX]^t$, $[FA_0]^t$ and $[FA_1]^t (t = 0, 1, \dots)$ using the following rules:

$$A^{t+1} = \begin{cases} [FA_0]^t, & \text{if } [FX]^t = 0; \\ [FA_0]^t, & \text{if } x_j^t = 0; \\ [FA_1]^t, & \text{if } x_j^t = 1. \end{cases} \quad (1.11)$$

First line of expression 1.11 determines the address of transition in case of unconditional jump, whereas the second and third lines determine this address for the conditional jump.

Structural diagram of MCU with compulsory microinstruction addressing (Fig. 1.19) includes the following blocks [1, 7]:

- sequencer CFA, calculating transition address from (1.11);
- register of microinstruction address RAMI, keeping address A^t ;
- control memory CM, keeping microinstructions;
- block of microoperation generation CMO;
- fetch flip-flop TF used to organize the stop mode of the MCU.

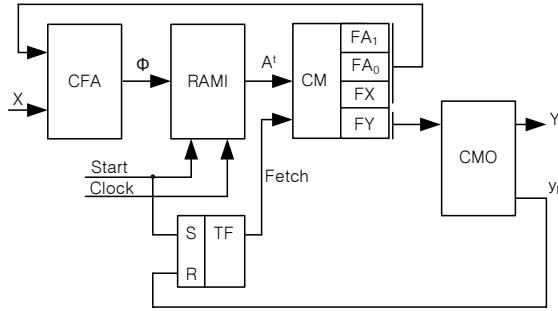


Fig. 1.19 Structural diagram of MCU with compulsory addressing

The control unit S_3 operates as follows. The pulse "Start" is used to load the address of first microinstruction to be executed (start address) into RAMI. At the same time the flip-flop TF is set up; signal Fetch=1 initiates reading of a microinstruction from the control memory. Let some address A^t be located in the register RAMI at time t ($t = 0, 1, \dots$). Corresponding microinstruction is then fetched from the memory CM. The operational part of this microinstruction is next transformed by the block CMO into microoperations $y_n \in Y$, which are directed to a system data-path. The sequencer CFA processes both the microinstruction address part and logical conditions X to produce the functions Φ , which form a transition address A^{t+1} sent into register RAMI. This address is loaded into RAMI by synchronization pulse "Clock". If the end of microprogram is reached, then special signal y_E is generated to clear the flip-flop TF. It causes termination of microinstruction fetching from memory CM, which means the end of MCU operation.

The transformation of initial GSA Γ is executed using the following rules [7]:

- if there is an arc $\langle b_q, b_E \rangle \in E$, such that $b_q \in B_1$, the variable y_E is assigned to the vertex b_q ;

- if there is an arc $\langle b_q, b_E \rangle \in E$, such that $b_q \in B_2$, an additional operator vertex b_{Q+1} ($Q = |B| - 2$) with the variable y_E is inserted into GSA Γ , and the arc $\langle b_q, b_E \rangle$ is replaced by arcs $\langle b_q, b_{Q+1} \rangle$ and $\langle b_{Q+1}, b_E \rangle$.

Therefore, the transformation of GSA for MCU with compulsory addressing of microinstructions is necessary to organize the ending mode of the MCU. Thus, transformation of the GSA Γ_1 is reduced to inserting the variable y_E into the vertex $b_6 \in B_1$ and adding the vertex b_7 . The transformed GSA $\Gamma_1(S_3)$ thus obtained is shown in Fig. 1.10. Generation of microinstructions with compulsory addressing is reduced to successive analysis of pairs of vertices $\langle b_q, b_t \rangle \in E$. All possible vertices pair configurations are shown in Fig. 1.11.

There are four possible configurations:

- $b_q, b_t \in B_1$ (Fig. 1.11a). In this case the vertex $b_q \in B$ corresponds to microinstruction with empty fields FX and FA_1 , whereas its field FY contains set of microoperations Y_q and field FA_0 contains the microinstruction address, corresponding to vertex $b_t \in B_1$. The analysis should be continued for the vertex $b_t \in B_1$. If, in such a pair, second vertex is the final vertex of GSA ($b_t = b_E$), then the vertex b_q corresponds to microinstruction with empty fields FX , FA_0 and FA_1 ;
- $b_q \in B_1, b_t \in B_2$ (Fig. 1.11b). In this case, the pair of vertices corresponds to one microinstruction with all fields containing useful information;
- $b_q, b_t \in B_2$ (Fig. 1.11c). In this case the vertex $b_t \in B_2$ corresponds to microinstruction with empty field FY , and the analysis should be continued for the vertex $b_q \in B_2$;
- $b_q \in B_2, b_t \in B_1$ (Fig. 1.11d). In this case the analysis should be continued for the both vertices of the pair.

Let us denote microinstructions by symbols O_m ($m = 1, \dots, M$); now the following microinstructions can be generated using the transformed FSA $\Gamma_1(S_2)$: $O_1 = \langle b_1, b_2 \rangle$, $O_2 = \langle b_3, \rangle$, $O_3 = \langle \emptyset, b_4 \rangle$, $O_4 = \langle b_5, \emptyset \rangle$, $O_5 = \langle b_6, \emptyset \rangle$, $O_6 = \langle b_7, \emptyset \rangle$.

Addresses of microinstructions with compulsory addressing can be appointed in the following manner. Each microinstruction O_m corresponds (one-to-one) to a binary code A_m with $R = \lceil \log_2 M \rceil$ bits ($m = 1, \dots, M$). A microinstruction with start address is determined by the arc $\langle b_0, b_q \rangle \in E$. In the case under consideration there is the arc $\langle b_0, b_1 \rangle \in E$ (Fig. 1.10), and therefore the start address belongs to the microinstruction O_1 , corresponding to the pair with vertex $b_1 \in B_1$. All other microinstructions are addressed in arbitrary manner.

The microprogram of MCU $S_3(\Gamma_1)$ includes $M = 6$ microinstructions, thus $R = 3$; and it is clear that $A_1 = 000$. Let $A_2 = 001, \dots, A_6 = 101$.

Because the control memory can keep only some bit strings, the encoding of operational and address parts of microinstructions is necessary to load microinstructions into control memory. Addressing of microinstructions gives information, which should be written into the fields FA_0 and FA_1 . There are many methods to encode operational part of microinstructions [7, 11]. Let us choose the one-hot encoding of microoperations to design the control memory of MCU $S_3(\Gamma_1)$, where $S_3(\Gamma_1)$ means that the GSA Γ_1 is interpreted by MCU with compulsory addressing

Fig. 1.10 Transformed GSA $\Gamma_1(S_3)$

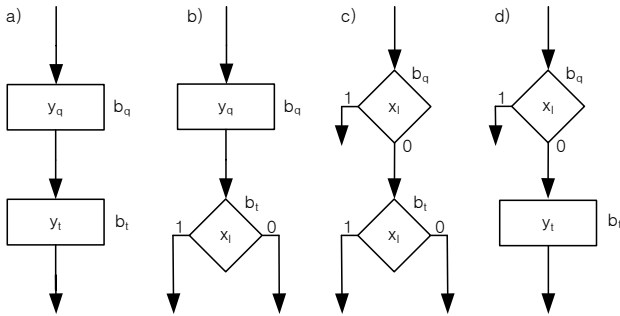
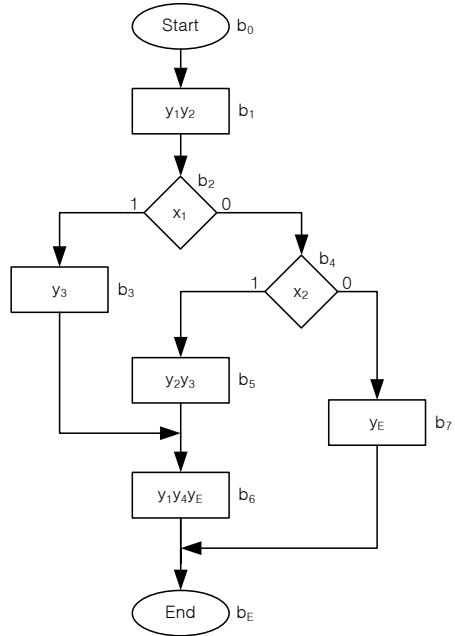


Fig. 1.11 Possible vertices pair configurations

of microinstructions. In case of one-hot encoding the length (bit capacity) n_1 of the field FY is determined by the following formula:

$$n_1 = N + 1. \tag{1.12}$$

For MCU $S_3(\Gamma_1)$ this formula gives the value $n_1 = 5$.

Let us encode logical conditions $x_l \in X$ using binary codes with minimum length (called sometimes minimal-length codes)

$$n_2 = \lceil \log_2(L + 1) \rceil. \tag{1.13}$$

The value 1 is added into (1.13) in order to take into account the code for unconditional jump, when $[FX] = \emptyset$. For MCU $S_3(\Gamma_1)$ this formula gives the value $n_2 = 2$. Let $K(\emptyset) = 00$; $K(x_1) = 01$; $K(x_2) = 10$.

Construction of the control memory content permits to form a table with lines keeping microinstruction addresses and binary codes of particular microinstructions. Control memory of MCU S_3 keeps M microinstructions with

$$n_3 = n_1 + n_2 + 2R \quad (1.14)$$

bits. In case of MCU $S_3(\Gamma_1)$ this formula gives the value $n_3 = 13$. The control memory content for MCU $S_3(\Gamma_1)$ is shown in Table 1.4.

In this table, microinstruction addresses are represented by variables from the set $A = \{a_1, a_2, a_3\}$, whereas the codes of microinstructions are represented by variables from the set $V = \{v_1, \dots, v_{13}\}$, where $|A| = R$, $|V| = n_3$. The last column of the table contains formulae of transitions for microinstructions, which are direct analogues of the formulae of transitions for operators of GSA [3].

Analysis of this table shows the main drawbacks of MCU with compulsory addressing of microinstructions, such as:

- an empty field FY for microinstructions, corresponding to the pairs $\langle \emptyset, b_t \rangle$, where $b_t \in B_2$;
- empty fields FX and FA_1 for microinstructions, corresponding to the pairs $\langle b_t, \emptyset \rangle$, where $b_t \in B_1$.

It results in the inefficient use of control memory volume, but a positive feature of compulsory addressing is the minimum number of microinstructions for the particular GSA, in comparison with MCU with other modes of microinstruction addressing [7].

Table 1.4 Control memory content for MCU $S_3(\Gamma_1)$

Address $a_1 a_2 a_3$	FY $v_1 v_2 v_3 v_4 v_5$	FX $v_6 v_7$	FA_0 $v_8 v_9 v_{10}$	FA_1 $v_{11} v_{12} v_{13}$	Formula of transitions
000	11000	01	010	001	$O_1 \rightarrow \bar{x}_1 O_3 \vee x_3 O_2$
001	00100	00	100	000	$O_2 \rightarrow O_5$
010	00000	10	101	011	$O_3 \rightarrow \bar{x}_2 O_6 \vee x_2 O_4$
011	01100	00	100	000	$O_4 \rightarrow O_5$
100	10011	00	000	000	$O_5 \rightarrow End$
101	00001	00	000	000	$O_6 \rightarrow End$

Synthesis of the logic circuit of MCU S_3 is reduced to the implementation of block CFA using standard multiplexers and control memory using standard memory blocks, such as PROM or RAM chips [1]. Let us point out that some logical elements should be used to implement the block CMO [3]. Assume that content of the field FA_1 is loaded into register RAMI if $z_1 = 1$, otherwise (if $z_1 = 0$) RAMI is

loaded from the field FA_0 of current microinstruction. Thus, expression (1.11) can be represented as

$$A^{t+1} = \bar{z}_1[FA_0] \vee z_1[FA_1]. \quad (1.15)$$

The variable $z_1 = 1$, if a logical condition to be checked is equal to 1; it means that

$$z_1 = \bigvee_{l=1}^L V_l x_l, \quad (1.16)$$

where V_l is a conjunction of variables $v_r \in V$, corresponding to the code $K(x_l)$ ($l = 1, \dots, L$).

In case of the MCU $S_3(\Gamma_1)$ expression (1.15) is represented as a system of equations

$$\begin{aligned} a_1 &= \bar{z}_1 v_8 \vee z_1 v_{11}; \\ a_2 &= \bar{z}_1 v_9 \vee z_1 v_{12}; \\ a_3 &= \bar{z}_1 v_{10} \vee z_1 v_{13}, \end{aligned} \quad (1.17)$$

and expression (1.16) has now the form

$$z_1 = \bar{v}_6 \bar{v}_7 0 \vee \bar{v}_6 v_7 x_1 \vee v_6 \bar{v}_7 x_2. \quad (1.18)$$

This formula specifies standard multiplexer with two control inputs and three informational inputs in use. The first term of expression (1.18) corresponds to unconditional jump. Symbol "0" represents the fact that logical 1 should be connected with informational input of the multiplexer corresponding to code 00; variables a_r from (1.17) coincide with variables D_r ($r = 1, \dots, R$). Expressions (1.17) – (1.18) determine the logic circuit of sequencer CFA of the MCU $S_3(\Gamma_1)$, shown in Fig. 1.12. Operation of this circuit can be easily deduced from Fig. 1.12.

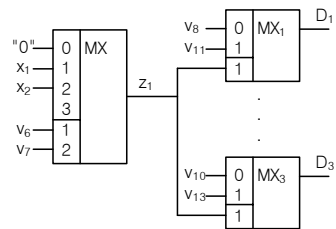


Fig. 1.12 Logic circuit of CFA of MCU $S_3(\Gamma_1)$

Some problems of the control memory implementation are discussed in Chapter 2 of this book.

There are two microinstruction formats in case of natural microinstruction addressing [1]: operational microinstructions corresponding to operator vertices of GSA Γ and control microinstructions corresponding to conditional vertices of GSA Γ (Fig. 1.13).

Fig. 1.13 Microinstruction formats for MCU with natural addressing of microinstructions

0	FY	
1	FX	FA ₀

First bit of each format represents field FA , used to recognize the type of microinstruction. Let $FA = 0$ correspond to operational microinstruction and $FA = 1$ to control microinstruction. As follows from Fig. 1.13, next address is not included in operational microinstructions. The same is true for the case, when a logical condition to be checked is equal to 1. In both cases mentioned above current address A^t is used to calculate next address:

$$A^{t+1} = A^t + 1. \tag{1.19}$$

Hence the following rule is used for next address calculation

$$A^{T+1} = \begin{cases} A^t + 1, & \text{if } [FA]^t = 0; \\ A^t + 1, & \text{if } (x_j^t = 1) \wedge ([FA]^t = 1); \\ [FA_0]^t, & \text{if } (x_j^t = 0) \wedge ([FA]^t = 1); \\ [FA_0]^t, & \text{if } ([FX]^t = \emptyset) \wedge ([FA]^t = 1). \end{cases} \tag{1.20}$$

Analysis of (1.20) shows that MCU with natural addressing of microinstructions should include a counter CAMI. Corresponding structure is shown in Fig. 1.14.

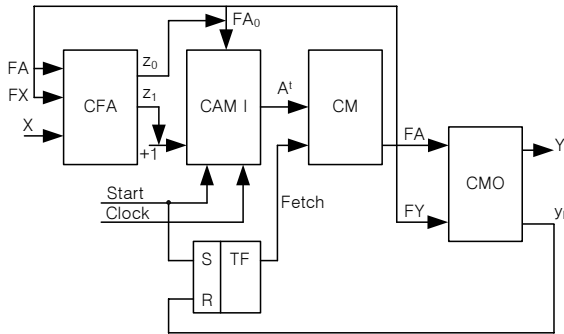


Fig. 1.14 Structural diagram of MCU with natural addressing of microinstructions

This MCU operates in the following manner. The pulse "Start" initiates loading of start address into CAMI. At the same time flip-flop TF is set up. Let an address A^t be located in CAMI at time t ($t = 0, 1, \dots$). If this address determines an operational microinstruction, the block CMO generates microoperations $y_n \in Y$, and the sequencer CFA produces signal z_1 . If this address determines a control microinstruction, microoperations are not generated, and the sequencer produces either signal z_0 (corresponding to an address loaded from the field FA_0), or signal z_1 (it corresponds

to adding 1 to the content of CAMI). The content of counter CAMI can be changed by pulse "Clock". If variable y_E is generated by CMO, then the flip-flop TF is cleared and operation of MCU terminated.

Let symbol S_4 stands for this kind of MCU. Now we use an example of MCU $S_4(\Gamma_1)$ to discuss some particular problems of such design.

The transformation of initial GSA is executed in two consecutive steps. First step involves the same transformations as in case of MCU S_3 . Addressing conflicts between microinstructions [7, 8] are eliminated during the second step. Let us point out that in case of MCU S_4 operational microinstructions correspond to operator vertices $b_q \in B_1$ and control microinstructions correspond to conditional vertices $b_q \in B_2$. Nature of addressing conflicts is the consequence of implicit transition addresses, as expressed by (1.19).

Let some GSA include two arcs $\langle b_i, b_q \rangle, \langle b_j, b_q \rangle \in E$, where $b_i, b_j \in B_1$ (Fig. 1.15a). Let indexes of vertices, corresponding microinstructions and microinstruction addresses be the same and take $A_i = 100$. According to (1.19) we find that $A_q = 101$, which means that the address A_j should be equal to 100. Thus, microinstructions O_i and O_j should have the same address. We call this situation addressing conflict. Some conditional vertex b_l with logical condition x_0 should be inserted in the initial GSA to eliminate this conflict. This condition corresponds to the unconditional jump, when $FX = \emptyset$ (Fig. 1.16a).

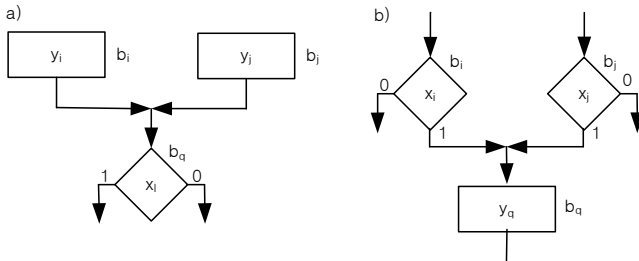


Fig. 1.15 Addressing conflicts in MCU S_4

Now, if $A_i = 100$, we have $A_q = 101$, and the field FA_0 of microinstruction O_i contains address $A_q = 101$. Addressing conflict is possible also between control microinstructions (Fig. 1.15b), and its elimination requires inserting an additional vertex (Fig. 1.16b).

Let us point out that GSA subgraphs, similar to ones shown in Fig. 1.15, can have arbitrary number of vertices.

Addressing conflicts can arise also among operational microinstructions and control microinstructions [7].

The transformed GSA $\Gamma_1(S_4)$ contains $M = 8$ vertices (Fig. 1.17). As the result of transformation, variable y_E is inserted into vertex b_6 , vertex b_7 with y_E is added, and vertex b_8 is also added, to eliminate addressing conflict between microinstructions corresponding to vertices b_3 and b_5 .

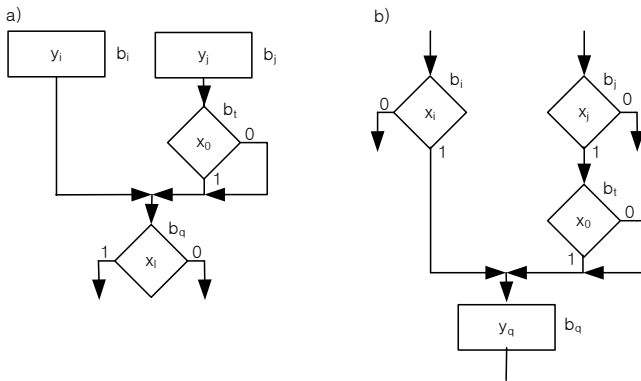


Fig. 1.16 Elimination of addressing conflicts

As it was mentioned above, each operator vertex corresponds to an operational microinstruction and each conditional vertex corresponds to a control microinstruction. It means that microinstructions are generated in a very simple way. For example, the microprogram of MCU $S_4(\Gamma_1)$ includes $M = 8$ microinstructions.

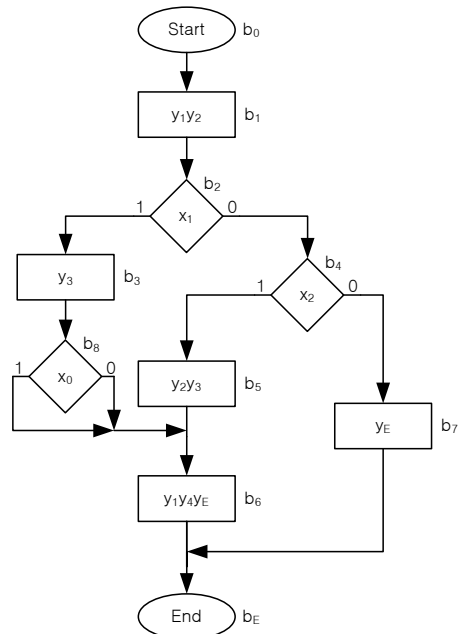


Fig. 1.17 Transformed GSA $\Gamma_1(S_4)$

Generation of special microinstruction sequences is needed in case of natural addressing of microinstructions. These sequences are created as follows. Let us build

a set $I(\Gamma)$, elements of which are inputs of the sequences. Vertex $b_q \in B_1 \cup B_2$ is the input of a sequence, if the input of this vertex is connected either with the output of vertex b_0 or with the output of conditional vertex, marked as "0".

In case of the MCU $S_4(\Gamma_1)$, the set $I(\Gamma) = \{b_1, b_4, b_7\}$ and microinstruction sequences are started by corresponding microinstructions. End points of these sequences are microinstructions corresponding to vertices connected with final vertex b_E , or conditional vertex with x_0 . Let α_g denote a microinstruction sequence. There are three such sequences in case of MCU $S_4(\Gamma_1)$, namely $\alpha_1 = \langle O_1, O_2, O_3, O_8 \rangle$, $\alpha_2 = \langle O_4, O_5, O_6 \rangle$, $\alpha_3 = \langle O_7 \rangle$. The zero address is assigned to the microinstruction, corresponding to this vertex b_i , where $\langle b_0, b_i \rangle \in E$. Addresses of next microinstructions belonging to this sequence are calculated according to (1.19). The address of current sequence input is calculated by adding 1 to the address of last microinstruction from previous sequence, and so on. Application of this procedure to the case of MCU $S_4(\Gamma_1)$, when $R = 3$, gives the microinstruction addresses, shown in Table 1.5.

Table 1.5 Microinstruction addresses of MCU $S_4(\Gamma_1)$

O_m	A_m	O_m	A_m
O_1	000	O_5	101
O_2	001	O_6	110
O_3	010	O_7	111
O_4	100	O_8	011

Encoding of operational and address parts of microinstructions is executed in the same manner as in case of MCU S_3 . Let us take the case of MCU $S_4(\Gamma_1)$, and use one-hot codes for microoperations ($n_1 = 5$), as well as minimal-length codes for logical conditions ($n_2 = 2$). Let corresponding codes for both MCU $S_3(\Gamma_1)$ and $S_4(\Gamma_1)$ be the same.

Construction of the control memory content is executed due to the fact that the usage of microinstruction bits depends on microinstruction type. The control memory of MCU S_4 contains M microinstructions with

$$n_4 = \max(n_1 + 1, n_2 + R + 1) \quad (1.21)$$

bits; for example, for MCU $S_4(\Gamma_1)$ it can be found that $n_4 = 6$. The control memory content of MCU $S_4(\Gamma_1)$ is shown in Table 1.6.

Let us discuss now the design of sequencer CFA for MCU S_4 . Variable $z_1 = 1$ should be generated either if $x'_t = 1$ or when an operational microinstruction is executed at time t . Thus, the logical expression for calculation of z_1 can be obtained by the following transformation of expression (1.16):

$$z_1 = \left(\bigvee_{l=1}^L V_l x_l \right) \vee \bar{v}_1, \quad (1.22)$$

where $v_1 = 0$ corresponds to FA=0. It is clear that $z_0 = \bar{z}_1$.

Table 1.6 Control memory content of MCU $S_4(\Gamma_1)$

Address	FA	FX FY	FA_0 FY	Formula of transitions
$a_1a_2a_3$	v_1	v_2v_3	$v_4v_5v_6$	
000	0	11000		$O_1 \rightarrow O_2$
001	1	01100		$O_2 \rightarrow \bar{x}_1O_4 \vee x_1O_3$
010	0	00100		$O_3 \rightarrow O_8$
011	1	00110		$O_8 \rightarrow \bar{x}_0O_6 \vee x_0O_6$
100	1	10111		$O_4 \rightarrow \bar{x}_2O_7 \vee x_2O_5$
101	0	01100		$O_5 \rightarrow O_6$
110	0	10011		$O_6 \rightarrow End$
111	0	00001		$O_7 \rightarrow End$

Let the counter CAMI has input C_1 , used to increment the counter content and input C_2 to load the input parallel code into the counter under the influence of pulse "Clock". The corresponding Boolean expressions for C_1 and C_2 have the form:

$$\begin{aligned} C_1 &= z_1 \cdot Clock, \\ C_2 &= \bar{z}_1 \cdot Clock. \end{aligned} \tag{1.23}$$

Expressions (1.23) serve to design the logic circuit of sequencer CFA for MCU $S_4(\Gamma_1)$ shown in Fig. 1.18.

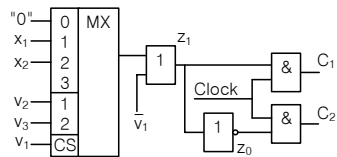


Fig. 1.18 Implementation of the block CFA for MCU $S_4(\Gamma_1)$

In this circuit multiplexer MX is active if $v_1 = 1$ is applied to the $\text{\textcircled{S}}enable\checkmark$ input CS of the chip. It corresponds to a control microinstruction. Remaining elements of this circuit follow directly from expressions (1.22) and (1.23). The methods of control memory implementation will be discussed later.

The comparative analysis of Tables 1.4 and 1.6 shows that MCU S_4 is characterized by longer microprogram, than the equivalent MCU S_3 . In case of MCU S_4 , control algorithm execution requires more time, than in case of the equivalent MCU S_3 . A positive feature of MCU S_4 is smaller microinstruction length. In case of our example we find that $n_3 = 2.17n_4$.

Microprogram control units with combined microinstruction addressing (Fig. 1.19) represent a compromise settlement with average number of microinstructions, of bit capacity and of control algorithm execution time.

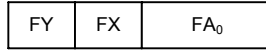


Fig. 1.19 Microinstruction format with combined addressing

$$A^{t+1} = \begin{cases} [FA_0]^t, & \text{if } |FX|^t = 0; \\ [FA_0]^t, & \text{if } x_j^t = 0; \\ [FA_1]^t, & \text{if } x_j^t = 1. \end{cases} \quad (1.24)$$

It follows from (1.24), that addressing conflicts are possible only between microinstructions with $[FX] \neq \emptyset$. A design method, which can be used for MCU with combined microinstruction addressing, can be found in [1, 7, 8].

Microprogram control units were very popular in the past [2, 13, 20], but they have one serious disadvantage, namely inferior performance in comparison with the equivalent finite state machines. As a rule, only single logical condition is checked during one cycle of MCU operation. Thus, multidirectional transitions depending on $k > 1$ logical conditions need $k > 1$ cycles for its execution, and the controlled data path will have $k - 1$ idle cycles, when its resources are not in use. Positive feature of MCU is the use of regular control memory to implement the microinstruction system. Besides, the sequencer CFA is very simple and can be implemented using standard multiplexers. As a rule, any change in the control algorithm leads to the redesign of corresponding FSM, but only small modifications of the control memory content in the equivalent MCU are needed.

1.4 Organization of compositional microprogram control units

The properties of the interpreted control algorithm have great influence on the hardware amount of corresponding control unit [4]. One of such properties is the existence of operational linear chains corresponding to the paths of GSA, which include operator vertices only. Let us call a GSA Γ the linear GSA (LGSA), if the number of its operator vertices exceeds 75% of the total number of vertices. Existence of operational linear chains allows simplification of input memory functions and reduction of hardware amount in the logic circuit of control unit. In this case either shift register or up counter [15, 16] is used to keep state codes.

One of the approaches for linear GSA interpretation is the use of compositional microprogram control units (CMCU), which can be viewed as a composition of the finite state machine and microprogram control unit [9]. These units have several particularities, distinguishing them from other control units:

1. Microinstruction format includes the operational part only. It permits to minimize the control word bit capacity. Thus control words kept in control memory have minimum possible length in comparison with all organizations of MCU mentioned above.

2. Microprograms have minimum possible length (the number of microinstructions), because the CMCU control memory is free from control microinstructions.
3. Multidirectional transitions are executed in one cycle of CMCU operation. It provides minimum time of control algorithm interpretation. Thus, such control units have similar performance, as compared with the equivalent FSM.

Let us introduce some definitions used in this book.

Definition 1.1. An operational linear chain (OLC) of GSA Γ is a finite vector of operator vertices $\alpha_g = \langle b_{g_1}, \dots, b_{g_{F_g}} \rangle$, such that an arc $\langle b_{g_i}, b_{g_{i+1}} \rangle \in E$ corresponds to each pair of adjacent vertices $b_{g_i}, b_{g_{i+1}}$, where i is the component number of vector α_g . Let D^g be a set of operator vertices, which are components of OLC α_g .

Definition 1.2. An operator vertex $b_q \in D^g$ is called an input of OLC α_g , if there is an arc $\langle b_t, b_q \rangle \in E$, such that $b_t \notin D^g$.

Definition 1.3. An input $b_q \in D^g$ is called a main input of OLC α_g , if GSA Γ does not include an arc $\langle b_t, b_q \rangle \in E$ such that $b_t \in B_1$.

Definition 1.4. An operator vertex $b_q \in D^g$ is called an output of OLC α_g , if there is an arc $\langle b_q, b_t \rangle \in E$, where $b_t \notin D^g$.

It follows from the basic properties of GSA [3] that each OLC α_g corresponding to definitions given above should have at least one input and exactly one output. Let I_g^j stand for input j of OLC α_g and O_g for its output. Let inputs of OLC α_g form a set $I(\alpha_g)$.

For GSA Γ we have the following sets:

1. A set of OLC $C = \{\alpha_1, \dots, \alpha_G\}$, satisfying the following condition

$$\begin{aligned} D^1 \cup \dots \cup D^G &= B_1; \\ |D^i \cap D^j| &= 0 (i \neq j; i, j \in \{1, \dots, G\}); \\ G &\rightarrow \min. \end{aligned} \tag{1.25}$$

2. A set of inputs $I(\Gamma)$ of the operational linear chains of GSA Γ :

$$I(\Gamma) = \bigcup_{g=1}^G I(\alpha_g). \tag{1.26}$$

3. A set of outputs $O(\Gamma)$ of the operational linear chains of GSA Γ :

$$O(\Gamma) = \{O_1, \dots, O_G\}. \tag{1.27}$$

Let the natural microinstruction addressing be executed for microinstructions corresponding to the adjacent components of each OLC $\alpha_g \in C$:

$$A(b_{g_{i+1}}) = A(b_{g_i}) + 1 (i = 1, \dots, F_g - 1). \tag{1.28}$$

In expression (1.28) symbol $A(b_{g_i})$ stands for the address of microinstruction corresponding to component i of vector $\alpha_g \in C$, where $i = 1, \dots, F_g - 1$. In this case GSA Γ can be interpreted by compositional microprogram control unit with basic structure of Fig. 1.20 [9]. Let us denote it as unit U_1 .

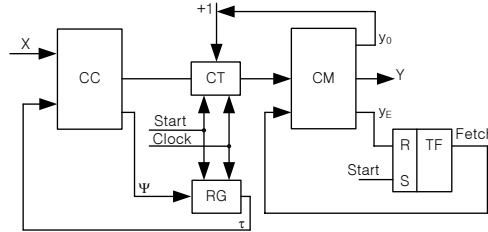


Fig. 1.20 Structural diagram of compositional microprogram control unit with basic structure

In the unit U_1 , combinational circuit CC and register RG form a finite state machine S_1 , which will be called microinstruction addressing unit or FSM S_1 . Counter CT, control memory CM and flip-flop TF form microprogram control unit S_2 with natural microinstruction addressing. The unit U_1 operates in the following manner. The pulse "Start" initializes following actions: the zero code of FSM S_1 initial state is loaded into register RG; start address of microprogram is loaded into counter CT; flip-flop TF is set up (Fetch=1). If Fetch=1, microinstructions can be fetched out of the control memory. Let at time t ($t = 0, 1, 2, \dots$) the code of state $a_m \in A_1$, where A_1 is a set of FSM S_1 states, be loaded into register RG and address $A(I_g^j)$ of the input j of OLC $\alpha_g \in C$ be loaded into the counter CT. Current microinstruction is read out of CM and its microoperations $y_n \in Y$ initialize some actions of the data-path. If this input is not the output of current OLC $\alpha_g \in C$ ($I_g^j \neq O_g$), additional variable $y_0 = 1$ is generated by MCU S_2 . If $y_0 = 1$, content of register RG is unchangeable and 1 is added to the content of counter CT. It corresponds to a transition between adjacent components of OLC $\alpha_g \in C$. If the output O_g is reached, then $y_0 = 0$. In this case circuit CC generates Boolean functions:

$$\Phi = \Phi(\tau, X), \quad (1.29)$$

$$\Psi = \Psi(\tau, X), \quad (1.30)$$

where $\tau = \{\tau_1, \dots, \tau_{R_1}\}$ is a set of state variables encoding states $a_m \in A_1$. The minimum number of these variables is determined as

$$R_1 = \lceil \log_2 M_1 \rceil, \quad (1.31)$$

where $M_1 = |A_1|$. If there is a transition from output O_g to some input under influence of some values of logical conditions, functions (1.29) determine the address of this input $I_g^j \in I(\Gamma)$ which is to be loaded into the counter. Functions (1.30) calculate the code of next state $a_s \in A_1$ to be loaded into RG. Content of both CT and RG is changed by the pulse "Clock". Outputs of the CT, $T = \{T_1, \dots, T_{R_2}\}$ determine next microinstruction address. This set includes

$$R_2 = \lceil \log_2 M_2 \rceil \quad (1.32)$$

variables, where $M_2 = |B_1|$. If CT contains the address of microinstruction corresponding to vertex $b_q \in B_1$ such that $\langle b_q, b_E \rangle \in E$, some additional variable $y_E = 1$ is generated. If $y_E = 1$, the flip-flop TF is cleared. Thus Fetch=0 and microinstruction fetching from the control memory is terminated.

As follows from (1.29), FSM S_1 of unit U_1 implements any multidirectional microprogram transition between output $O_g \in O(\Gamma)$ and input $I_i^j \in I(\Gamma)$ in one cycle of operation. At the same time MCU S_2 implements addressing rule (1.28), used to organize transitions between microinstructions corresponding to adjacent components of OLC $\alpha_g \in C$. Therefore, control memory CM should only keep microoperations $y_n \in Y$ and additional variables y_0, y_E . In other words, an address part is absent in the microinstruction format in case of CMCU U_1 . The main disadvantage of CMCU U_1 is the loss of universality, because changes in the interpreted microprogram lead to the redesign of circuit CC. Fortunately, as it will be shown below, current achievements in semiconductor technology permit to eliminate this drawback.

If we treat the concatenation of register RG and counter CT as the code of internal CMCU state, then the CMCU is a Moore FSM, in which each state is represented by the concatenation of FSM S_1 state and by the address of MCU S_2 microinstruction. Let us point out that all these models (FSM, MCU and CMCU) can be used to implement control algorithms of any digital system, including computers and complex industrial controllers. Some problems, which should be solved during the design of compositional microprogram control units, are listed below:

1. Problem of finding the minimum partition C of a set of operator vertices of GSA Γ , such that each class of this partition corresponds to some OLC.
A resulting set of OLC C should be a partition of set B_1 , because in other cases, the same operator vertices may appear in different OLCs. It results in the increase of both microprogram length (the number of microinstructions) and size of the control memory. It follows from the principle of CMCU operation, that amount of hardware in the logic circuit of FSM S_1 depends on the total number of OLC inputs. Thus minimizing hardware amount requires finding partition C , called *minimum partition*, which minimizes the total number of OLC inputs.
2. Problem of microinstruction addressing for MCU S_2 . This problem corresponds to the one we meet in the MCU design with natural microinstruction addressing and could be solved using the well-known methods.
3. Problem of transformation of the initial GSA Γ . The transformation should be made in such a manner that output signals Φ of FSM S_1 , which interprets transformed GSA $\Gamma(U_1)$, form address $A(I_g^j)$ of input I_g^j for each OLC $\alpha_g \in C$ to be loaded into the counter CT. Transitions between operator vertices of the transformed GSA $\Gamma(U_1)$ should correspond to transitions between outputs and inputs of the operational linear chains of initial GSA Γ .

Methods of synthesis and optimization suitable for logic circuits of FSM and MCU depend significantly on logical elements in use. The methods discussed in our book are oriented towards field-programmable logic devices (FPLD) ; and correspond to recent state-of-the-art in the design technique. These devices are

very complex, and before discussing the compositional microprogram unit design, let us analyze both FPLD and various methods of hardware optimization oriented towards these elements.

References

1. M. Adamski and A. Barkalov. *Architectural and Sequential Synthesis of Digital Devices*. University of Zielona Góra Press, 2006.
2. A. Agrawala and T. Rauscher. *Foundations of Microprogramming*. Academic Press, New York, 1976.
3. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
4. A. A. Barkalov. *Development of formalized methods of structure synthesis for compositional automata*. PhD thesis, Donetsk:DonSTU, 1994. (in Russian).
5. A. A. Barkalov. Principles of optimization of logic circuit of Moore FSM. *Cybernetics and System Analysis*, (1):65–72, 1998. (in Russian).
6. A. A. Barkalov. *Synthesis of Control Units with PLDs*. Donetsk National Technical University, 2002. (in Russian).
7. A. A. Barkalov and A. V. Palagin. *Synthesis of Microprogram Control Units*. IC NAS of Ukraine, Kiev, 1997. (in Russian).
8. A. A. Barkalov, M. Węgrzyn, and R. Wiśniewski. Partial reconfiguration of compositional microprogram control units implemented on fpgas. In *Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems (Brno)*, pages 116–119, 2006.
9. A.A. Barkalov. Microprogram control unit as composition of automate with programmable and hardwired logic. *Automatics and computer technique*, (4):36–41, 1983. (in Russian).
10. A. Clements. *The Principles of Computer Hardware*. Oxford University Press, Inc., New York, 2000.
11. S. Dasgupta. The organization of microprogram stores. *ACM Computing Survey*, (24):101–176, 1979.
12. D. Gajski. *Principles of Digital Design*. Prentice Hall, New York, 1997.
13. M. J.Flynn and R.F. Rosin. Microprogramming: An introduction and a viewpoint. *IEEE transactions on Computers*, C–20(7):727–731, 1971.
14. G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw–Hill, 1994.
15. C. Papachristou. Hardware microcontrol schemes using PLAs. In *Proceeding of 14th Microprogramming Workshop*, number 2, pages 3–15, 1981.
16. C. Papachristou and S. Gambhir. A microsequencer architecture with firmware support for modular microprogramming. *ACM SIGMICRO Newsletters*, 13(4):105–113, 1982.
17. A. Salisbury. *Microprogrammable Computer Architectures*. Am Elstein, New York, 1976.
18. G. Saucier, M. Depaulet, and P. Sicard. Asyl: a rule-based system for controller synthesis. *IEEE Transactions on Computer-Aided Design*, 6(11):1088–1098, 1987.
19. G. Saucier, P. Sicard, and L. Bouchet. Multi-level synthesis on programmable devices in the ASYL system. In *Proceedings of Euro ASIC*, pages 136–141, 1990.
20. V. A. Skljarov. *Synthesis of automata on matrix LSI*. Nauka i Technika, Minsk, 1984. (in Russian).
21. V. Solovjev and M. Czyzy. Synthesis of sequential circuits on programmable logic devices based on new models of finite state machines. In *Proceedings of the EUROMICRO Conference, Milan*, pages 170 – 173, 2001.
22. V. V. Solovjev. *Design of Digital Systems Using the Programmable Logic Integrated Circuits*. Hot line – Telecom, Moscow, 2001. (in Russian).

Chapter 2

Synthesis of control units with field-programmable logic devices

Abstract The chapter discusses contemporary field-programmable logic devices and their evolution, starting from the simplest programmable logic devices, such as PLA, PAL, GAL and PROM, and finishing with very sophisticated chips such as CPLD and FPGA. This analysis shows particular features of different elements and permits to optimize the control unit logic circuits, in which some particular elements are used. The CMCU has some features of both FSM and MCU. Therefore main design and optimization methods applied in case of these two types of control units are presented in the main part of the chapter.

2.1 Evolution of field-programmable logic devices

This book deals mostly with synthesis methods oriented towards logic devices, which are programmed by the end user. These devices are called field-programmable logic devices (FPLD) [65]. The programmability of FPLD is intended for the hardware level, contrary to microprocessors, which run programs but have fixed hardware. A FPLD is the general purpose chip whose hardware can be configured by the end user to implement some particular project. Our interest can be explained by domination of FPLD in the design of modern digital devices [94].

Historically first representatives of FPLD are programmable read-only memory chips (PROM), manufactured by Harris Semiconductor in 1970 [65]. They included a fixed array of AND gates (AND-array) followed by a programmable array of OR gates (OR-array), as shown in Fig. 2.1.

The AND-array implements address decoder DC, having S inputs and $q = 2^S$ outputs, where each output corresponds to the unique address of memory cell. The content of OR-array is programmable and the sign 'X' in Fig. 2.1 shows the programmable connection.

This architecture perfectly fits for implementation of a system of Boolean functions $Y = \{y_1, \dots, y_N\}$ depending on Boolean variables from the set $X = \{x_1, \dots, x_L\}$, represented by a truth table [66]. In this case, the system to be implemented can be

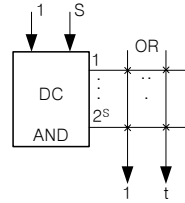


Fig. 2.1 Architecture of PROM

represented by the table with

$$H = 2^L \tag{2.1}$$

rows, where each row includes L input columns and N output columns. Let us denote this system of Boolean functions (SBF) as $Y(L, N)$ and discuss its implementation with $\text{PROM}(S, t)$, where $\text{PROM}(S, t)$ means that PROM chip has S inputs and t outputs. Combinations of parameters S, t, L, N give the following implementations of SBF.

1. In case when $S \geq L, t \geq N$, the system $Y(L, N)$ can be implemented in a trivial way using only one $\text{PROM}(S, t)$ chip. This implementation is shown in Fig. 2.2, where logical variables X are connected with PROM address inputs, and functions Y appear on the PROM outputs.

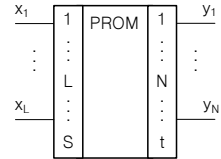


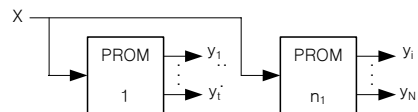
Fig. 2.2 Trivial implementation of SBF with PROM

2. In case when $S \geq L, t < N$, it is necessary to have

$$n_1 = \left\lceil \frac{N}{t} \right\rceil \tag{2.2}$$

$\text{PROM}(S, t)$ chips to implement $Y(L, N)$. Address inputs of all chips are connected with logical variables $x_i \in X$, and each chip generates up to t output functions $y_n \in Y$ (Fig. 2.3). This approach is called sometimes the "expansion of PROM outputs" [15].

Fig. 2.3 Implementation of SBF with expansion of PROM outputs. The value of i can be calculated as $i = t(n_1 - 1) + 1$



3. If $S < L, t \geq N$, the expansion of PROM inputs [15] is used and

$$n_2 = \left\lfloor \frac{2^L}{2^S} \right\rfloor = \lceil H/q \rceil \quad (2.3)$$

PROM chips is necessary to implement the system $Y(L, N)$ of Fig. 2.4.

In this circuit, the $L - S$ leftmost bits of input assignment $\langle x_1, \dots, x_L \rangle$ form a set of variables X^1 , which are connected with inputs of decoder DC, having n_2 outputs. Outputs of the DC are connected with "enable" inputs of corresponding PROMs. Address inputs of all chips are connected with S rightmost bits of input assignment and these variables form a set X^2 . Partial functions Y^i are generated as outputs of microchip i and correspond to subtables of the truth table with rows from $q(i - 1)$ to qi . As it can be seen from Fig. 2.4, OR-gates are used to produce the final values of functions $y_n \in Y$. The approach considered above is rather theoretical, because this level of the circuit could be implemented using the three-stable outputs of PROM chips [65].

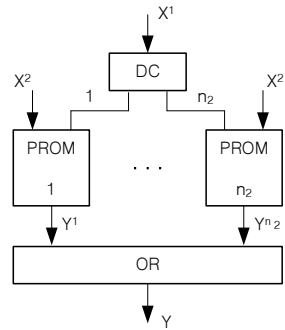


Fig. 2.4 Implementation of SBF with expansion of PROM inputs

4. If $S < L, t < N$, then

$$n_3 = n_1 \cdot n_2 \quad (2.4)$$

PROM chips are necessary for implementation of SBF. In this case both methods of expansion (of outputs and inputs) are used together.

Thanks to regularity of their structure, PROM chips find wide application for implementation of tabular functions. Main drawback of the PROM is doubling of their capacity, if the number of inputs is incremented by 1. Besides, PROMs can not be used for implementation of the SBF satisfying the following condition

$$H_1 \ll H \quad (2.5)$$

where H_1 is the number of input assignments, such that at least one of functions $y_n \in Y$ is equal to 1.

Programmable logic arrays (PLA) were introduced in the mid 1970s by Signetics [64] and were oriented to implementation of SBF, for which condition (2.5) is satisfied. Particular property of PLA is the programmability of both AND- and OR-arrays (Fig. 2.5), providing greater flexibility, than the PROM.

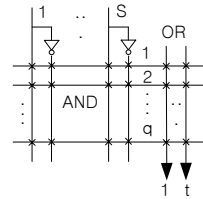


Fig. 2.5 Architecture of PLA

Thanks to programmability of both arrays, PLA can be applied to implement SBF represented by the minimal sum-of-products [66]. Programmability of the AND-array leads however to the increase of chip area and reduction of both speed and parameter q of the resulting circuit, in comparison with PROM-implementations [12].

Let the PLA with S inputs, t outputs and q terms be denoted by $PLA(S, t, q)$ and discuss how they can be used for the SBF $Y(L, N, H_1)$ implementation. The following possible combinations of SBF and PLA parameters are listed below.

1. If $S \geq L, t \geq N, q \geq H_1$, the SBF Y is implemented in trivial way using one PLA chip. Structure of resulting circuit is similar to the one shown in Fig. 2.2, where PLA is used instead of PROM.
2. If $S \geq L, t < N, q \geq H_1$, logic circuit is implemented using n_1 PLA chips, where the value of n_1 is determined by (2.2), and the circuit structure is similar to that from Fig. 2.3.
3. If $S \geq L, t \geq N, q < H_1$, the "expansion of PLA terms" approach should be used [12], and the circuit can be implemented using

$$n_4 = \left\lceil \frac{H_1}{q} \right\rceil \tag{2.6}$$

$PLA(S, t, q)$ chips. Implementation of logic circuit in this case is similar to the one, shown in Fig. 2.6, but decoder DC is absent, because inputs of all microchips are connected with the same logical conditions X .

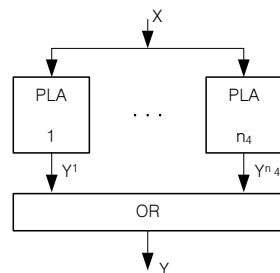


Fig. 2.6 Implementation of SBF with expansion of PLA terms

4. If $S \geq L, t < N, q < H_1$, both expansion methods mentioned above should be applied simultaneously. Minimization of hardware amount can be made with

application of sophisticated design methods [12], based on the search of some partitions on the set of SBF terms.

More complex synthesis methods are used to implement SBF Y , when the following condition is satisfied

$$S < L. \quad (2.7)$$

In this case we have also the second condition

$$F_{\max} \leq L, \quad (2.8)$$

where F_{\max} is the maximal number of literals [66] in the terms of SBF Y . If this condition holds, the initial SBF can be implemented by the single-level circuit, shown in Fig. 2.7.

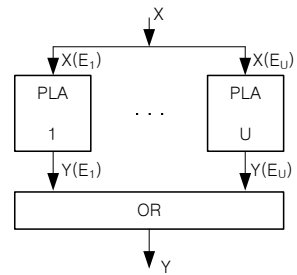


Fig. 2.7 Single-level implementation of SBF with PLA

During the design of logic circuit, a partition Π_F of the set of terms F should be found, in which $|F| = H_1$, and the number of blocks U attains minimum [12]. Let $X(E_u)$ be the set of logical conditions, which form the terms of set $E_u \in \Pi_F \{E_1, \dots, E_U\}$, and $Y(E_u)$ be a set of functions depending on terms $E_u \in \Pi_F$. Partition Π_F should satisfy the following conditions:

$$\begin{aligned} |X(E_u)| &\leq S, \\ |Y(E_u)| &\leq t, \quad (u = 1, \dots, U) \\ |E_u| &\leq q, \\ U &\rightarrow \min. \end{aligned} \quad (2.9)$$

Many different approaches to the solution of problem (2.9) minimizing the value of U are known [5, 90]. If condition (2.8) is violated, SBF Y can be implemented as a multilevel circuit [12], in which resulting digital system performance is reduced.

It is clear, that PLA allows the implementation of combinational circuits only. In case of sequential circuit implementation, the outputs of PLA should be connected to the external register. This disadvantage can be eliminated by adding flip-flops inside the chip, at each output of PLA. These chips are called registered PLA or programmable logic sequencers (PLS) [65]. Design methods oriented towards the PLS use decomposition of initial GSA into subgraphs in such a way that FSM corresponding to each subgraph can be implemented using only one PLS chip [12, 94].

As a rule, real digital devices are specified by SBF with limited number of terms, where the following condition is satisfied

$$|H(y_n)| \leq 16, \quad (2.10)$$

in which $H(y_n)$ is a set of terms, which are used in the SOP as products of Boolean function $y_n \in Y$. Analysis of this condition shows that PLA have redundancy of connections, because any term of PLA can be connected with arbitrary output of the chip.

Programmable array logic (PAL) chips, introduced by Monolithic Memories in 1978 [65], were oriented to SBF implementation satisfying (2.10). The PAL chips contain programmable AND-arrays and t fixed OR-arrays (Fig. 2.8).

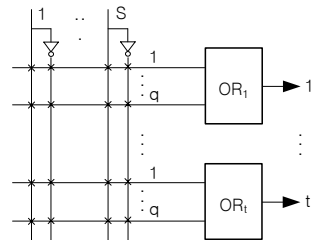


Fig. 2.8 Architecture of PAL

Design methods used in case of PAL are oriented on minimizing $|H(y_n)|$ up to some fixed value q , determining the number of AND-arrays connected with a single OR-array [63, 94]. To broaden the application area of these chips, additional elements such as flip-flops, logic gates and multiplexers were added to each PAL output. This new output cell was called Macrocell. It has a feedback path from the output of the cell to the AND-array. All connections inside the macrocell were also programmable and therefore PAL is more flexible than other solutions. These PAL cells were called generic array logic (GAL) and were introduced by Lattice in 1985 [65, 71]. Let us point out, that GAL chips are still manufactured in standalone packages [71] by Lattice, Atmel, TI, etc. Typical example of GAL device is the GAL16V8 chip, which has 16 inputs, 8 outputs and 20 pins. This device has 8 input pins and 8 bidirectional input/output pins, which can be used either as inputs or as outputs.

The chips PLA, PLS, PAL, GAL belong to the class of Simple Programmable Logic Devices (SPLD), which have at most 40 inputs/outputs and are equivalent to at most 500 NAND-gates with two inputs [65]. Development of semiconductor technology was yielded in producing Complex Programmable Logic Devices (CPLD), which can be viewed as array structures of macrocells as their elements. The simplified architecture of CPLD with PAL macrocells is shown in Fig. 2.9.

In this CPLD each macrocell PAL_i ($i = 1, \dots, I$) is connected to S fixed chip inputs and programmable input/outputs IO_i . Outputs of the blocks can be used as input information for switch matrix SM. First CPLD were MegaPAL of MMI devices [65]. Now several companies such as Altera, Xilinx, Cypress, Atmel, Lattice

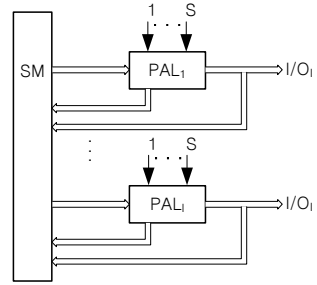


Fig. 2.9 Simplified architecture of CPLD

manufacture CPLD chips [1, 2, 65, 94]. A typical example of CPLD is the Xilinx XC9500, where PLD devices, similar to 36V18 GAL, are used. Modern CPLD chips contain some additional features, like JTAG support and interface to other logic standards. Let us point out that such CPLD as CoolRunner family (Xilinx) are based on PLA blocks instead of PAL [3]. For example, CPLD of the CoolRunner XPLA3 family include from 32 to 512 PLA blocks, which can replace from 750 to 12 0000 equivalent gates. Such devices can have from 36 to 260 input/output pins and the current consumption less than 0.1 mA.

Field-programmable gate arrays (FPGA) were introduced by Xilinx in 1985 [65]. They differ from CPLD chips in architecture, because they are aimed at the implementation of high-performance and large-size circuits. The simplest FPGA includes programmable logic blocks PLB, based on the look-up table element LUT, flip-flop and multiplexer (Fig. 2.10).

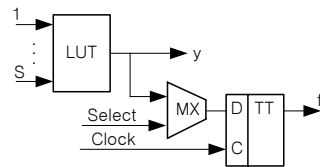


Fig. 2.10 Architecture of programmable logic block

In such a circuit, the LUT element implements an arbitrary Boolean function $y = y(x_1, \dots, x_S)$. The signal "Select" controls multiplexer MX and the choice of either combinational or register mode of the PLB output. The "Clock" pulse is used for the timing of flip-flop TT. Let us point out that LUT elements can be implemented using either random-access memory blocks or programmable multiplexers [64]. For example, FPGAs of Virtex E family include from 384 to 16 224 logic blocks, from 1 728 to 73 008 logic cells, that is the equivalent of 72K to 4M system gates. They have from 176 to 804 input/output pins, include from 1 392 to 64 896 flip-flops and operate with maximal internal frequency up to 240 MHz [3]. The blocks are connected using matrix of programmable interconnections MPI. The wires of MPI form a grid, in which PLBs are represented by nodes and input/output blocks represented by endings (Fig. 2.11).

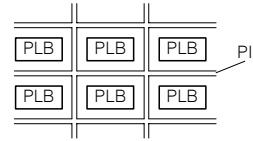


Fig. 2.11 Simplified architecture of FPGA

Design process oriented on FPGA is very complex. First of all, initial SBF should be transformed in such a manner, that each subfunction can be implemented using a single PLB. As a rule, typical PLB has at most 5 inputs [65]. Methods of functional decomposition are used [82] in order to transform the initial SBF. Resulting Boolean functions $y_n \in Y$ are implemented using multilevel circuits. Next, the routing and placement procedures are applied and sophisticated optimization tasks are solved during this design step. Let us point out that using FPGAs is impossible without efficient CAD tools, but fortunately many of them are available [65]. Nowadays, FPGA microchips are manufactured by several companies, such as Altera, Xilinx, Actel, Atmel, Lucent and QuickLogic.

Development of different CPLD and FPGA families culminated in the appearance of a system-on-a-programmable chip (SoPC) [65, 79] with several hundred millions of transistors [52]. Modern SoPCs include tools for implementation of arbitrary logic (PAL, PLA or FPGA) and system memory (dedicated PROM or RAM blocks) with microprocessors. They are characterized by large variety of architectures and their progress can be seen on the Web-sites of corresponding industrial companies.

The most important conclusion following from this short analysis is that modern FPLDs include several tools for implementation of arbitrary logic represented in the SOP form (PAL, PLA, FPGA) and of memory blocks suitable for implementation of tabular functions (PROM, RAM). It leads us to conclude that the CMCU structure perfectly fits to modern logical elements. The problems of hardware optimization are always very important regardless of characteristics of the FPLDs in use. Due to irregular character of the control unit circuit structure, it cannot be implemented as IP-cores. In consequence the design process involving control units appears to be a real bottleneck of modern system design. The CMCU have some features of both FSM and MCU. Hence it would be useful to consider some optimization methods used in case of the control units with FPLDs .

2.2 Optimization of microprogram control units

The principle of microprogram control was proposed in 1951 [100, 101]. Microprogramming was widely used in computers starting from 1964, when IBM announced the System/360 [29,44,70,74,95]. After spectacular success of the IBM System/360 line, microprogramming became main implementation technique for most computers. Very fast and very simple computers were the only exception. This situation lasted for about 20 years and, for example, practically all models of the IBM/370

and all models of the DEC PDP-11 (aside from the PDP-11/20) were microprogrammed. The DEC VAX 11/780 can be viewed as a peak of the microprogramming era. It appeared in 1978 and included 4K read-only control memory with 96 bits per word and additional 1K read/write words were available for the end user [17]. The analysis of early microprocessors shows that some models were microprogrammed (for example, Intel 8080 and Zilog Z80), and some were hardwired (for example, MC6800) [8]. The development of semiconductor technology stimulated present situation, in which the principle of microprogram control is still used in computers, but only few modern computers use microprogramming for implementation of their instruction sets [89]. Microprogramming is still used in different Intel-86 compatible microprocessors like the Pentium 4 and AMD Athlon [89]. On the other hand, all RISC microprocessors are hardwired [89], as well as recent processors for IBM System/390 [25]. Nevertheless, we can still use microprogramming to implement control units of all digital systems.

Basic optimization methods for MCU were developed in the 1970s and 1980s [6–8, 15, 17, 50, 54, 89], but are still discussed today [5, 17]. Formal methods of MCU design can be found, for example, in [5, 15]. One of the main goals of MCU optimization is to obtain better performance of the controlled digital systems. To achieve this goal, the time of cycle and/or the number of cycles needed to execute the control algorithm should be reduced. The second goal is reduction of control memory size. One solution of this problem is the length (bit capacity) reduction of control words used for implementation of particular microprogram. Let us discuss the second problem, because it is still of great importance, even after the appearance of CPLDs and FPGAs.

One of the important factors, which affected the control word length, is the organization of microinstruction operational part. There are three main operational part organization principles [17]:

- one-hot encoding of microoperations ;
- binary encoding of collections of microoperations;
- encoding of the fields of compatible microoperations.

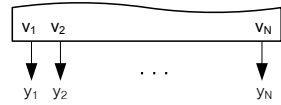
Let us discuss application of these methods using an example of the MCU S_4 with the following collections of microoperations in the interpreted control algorithm: $Y_1 = \emptyset$, $Y_2 = \{y_7\}$, $Y_3 = \{y_1, y_4, y_6\}$, $Y_4 = \{y_2\}$, $Y_5 = \{y_1, y_2, y_4, y_6\}$, $Y_6 = \{y_2, y_5, y_6, y_8\}$, $Y_7 = \{y_3, y_8\}$, $Y_8 = \{y_3, y_6, y_8\}$, $Y_9 = \{y_3, y_7, y_9\}$, $Y_{10} = \{y_7, y_9, y_5\}$, $Y_{11} = \{y_1, y_7\}$ Thus, the control algorithm implemented by MCU S_4 has the following parameters: number of microoperations $N = 9$ and number of collections of microoperations $Q = 11$.

In case of one-hot encoding of microoperations the bit capacity of the operational part of microinstructions is equal to

$$m_1 = N. \quad (2.11)$$

It means that the microoperation y_n corresponds to the element v_n (one bit of the control memory output word) of the set of encoding variables V (Fig. 2.12).

Fig. 2.12 Organization of operational part of microinstructions with one-hot encoding of microoperations



Main advantage of this approach, in comparison with other operational part organizations, is maximal flexibility of microprogramming. It means that any modification of the microprogram is reduced to the modification of control memory content. Let us point out that appearance of reprogrammable FPLD reduces the significance of this factor. Besides, this method is characterized by maximum performance of the controlled digital system. The main drawback of such organization is maximal size of control memory and its inefficient usage because the following condition is always satisfied

$$\max(|Y_1|, \dots, |Y_Q|) \ll N. \tag{2.12}$$

This method can be used only when minimal cycle time is accepted as the control unit design criterion.

For *binary encoding of collections of microoperations*, the number of encoding bits is equal to

$$m_2 = \lceil \log_2 Q \rceil. \tag{2.13}$$

Each collection $Y_q \subseteq Y$ corresponds to a binary code $K(Y_q)$ with m_2 bits. Elements of a set V are used for this encoding. Generation of microoperations $y_n \in Y$ is executed by additional circuit CFO, including decoder DC and coder CD (Fig. 2.13). The decoder DC generates some variables B_q corresponding to the codes of collections $K(Y_q)$:

$$B_q = \bigwedge_{i=1}^{m_2} v_i^{l_i} (q = 1, \dots, Q). \tag{2.14}$$

In (2.14) variable $l_i \in \{0, 1\}$ represents the value of bit i of the code $K(Y_q)$, where $v_i^0 = \bar{v}_i, v_i^1 = v_i (i = 1, \dots, m_2)$. The coder CD generates Boolean functions

$$y_n = \bigvee_{q=1}^Q C_{nq} B_q (n = 1, \dots, N), \tag{2.15}$$

where C_{nq} is the Boolean variable, equal to 1 iff $y_n \in Y$.

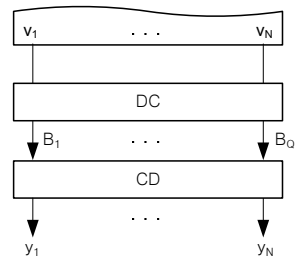


Fig. 2.13 Generation of microoperations by CMO

Application of this organization permits to minimize the length of control word, but the performance of such MCU is reduced in comparison with MCU with one-hot encoding, due to the existence of additional system $\langle DC, CD \rangle$. This drawback can be partially eliminated, if the encoding of collections $Y_q \subseteq Y$ is executed in such a manner, that each Boolean function for each microoperation $y_n \in Y$ is represented by one term of the SOP (2.15) [17].

In case of MCU S_4 we have $m_2 = 4$ and the following SOPs: $y_1 = B_2 \vee B_5 \vee B_{11}$; $y_2 = B_4 \vee B_5 \vee B_6$; $y_3 = B_7 \vee B_8 \vee B_9$; $y_4 = B_3 \vee B_5$; $y_5 = B_6 \vee B_{10}$; $y_6 = B_3 \vee B_5 \vee B_6 \vee B_8$; $y_7 = B_2 \vee B_9 \vee B_{10} \vee B_{11}$; $y_8 = B_6 \vee B_7 \vee B_8$; $y_9 = B_9 \vee B_{10}$. Let us encode the collections $Y_q \subseteq Y$, as shown in the Karnaugh map of Fig. 2.14.

		$v_3 v_4$			
		00	01	11	10
$v_1 v_2$	00	Y_1	Y_4	*	Y_7
	01	Y_3	Y_5	Y_6	Y_8
	11	*	Y_{11}	Y_{10}	Y_9
	10	Y_2	*	*	*

Fig. 2.14 Encoding of collections of microoperations for MCU S_4

Taking into account the don't care input assignments shown as "*", the system of microoperations $y_n \in Y$ of MCU S_4 has the form: $y_1 = v_2 \bar{v}_3$; $y_2 = \bar{v}_1 v_4$; $y_3 = v_3 \bar{v}_4$; $y_4 = \bar{v}_1 v_2 \bar{v}_3$; $y_5 = v_3 v_4$; $y_6 = \bar{v}_1 v_2$; $y_7 = v_1$; $y_8 = \bar{v}_1 v_3$; $y_9 = v_1 v_3$. In this particular case the system $\langle DC, CD \rangle$ is represented by an array of AND-gates and provides decrease of hardware amount and better performance of the control unit.

This encoding problem can be solved using well-known algorithm ESPRESSO [64, 68]. Let us point out that this encoding algorithm can be modified to meet particular properties of logical elements used to implement the circuit, and in consequence:

- when using PLA, encoding is executed in such a manner, that SOP for each formula (2.15) has at most $\lceil q/t \rceil$ terms, where t, q is the number of PLA outputs and terms respectively;
- when using PAL, encoding is executed in such way that SOP for each formula (2.15) has at most q terms, where q is the number of terms in PAL macrocells;
- when using FPGA, encoding is executed in such a manner, that each microoperation $y_n \in Y$ can be implemented using one LUT element.

Let us point out that PROM can be used to implement CMO circuit. In this case, the system of microoperations should be represented in tabular form.

The method of *encoding of the fields of compatible microoperations* represents some compromise between hardware and performance in comparison with two approaches mentioned above. It is based on the partition of set Y into equivalence classes B_1, \dots, B_l , where each class includes only the microoperations, which are not executed concurrently. It means that microoperations from one class always belong to different collections $Y_q \subseteq Y$. Each block $B_i \in \Pi_Y$, where Π_Y is the partition

of the set Y , corresponds to the field Y^i in the microinstruction format. Bit capacity r_i of the field Y^i is determined as

$$r_i = \lceil \log_2(|B_i| + 1) \rceil \quad (i = 1, \dots, I). \tag{2.16}$$

Let us point out that 1 in (2.16) is added to cover the case, when microoperations from a given class do not belong to the particular collection $Y_q \subseteq Y$. Each microoperation $y_n \in B_i$ corresponds to one binary code $K(y_n)$ having r_i bits ($i = 1, \dots, I$). The number of elements in the set of encoding variables V is given as

$$m_3 = \sum_{i=L}^I r_i. \tag{2.17}$$

The microoperations belonging to class $B_i \in \Pi_Y$ are generated by correspondent individual decoder DC_i (Fig. 2.15).

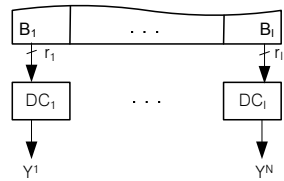


Fig. 2.15 Organization of MCU with encoding of the fields of compatible microoperations

The main goal of finding partition Π_Y is the minimization of parameter (2.17). This problem was formulated in [86], where this method was proposed. First solution of this task was reduced to the classical task of partition with minimal cost. There are many different solutions of the same problem, which can be found, for example, in [17].

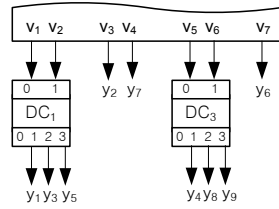
For the case of MCU S_4 , solution gives the partition $\Pi_Y = \{B_1, \dots, B_4\}$, where $Y^1 = \{y_1, y_3, y_5\}$, $Y^2 = \{y_2, y_7\}$, $Y^3 = \{y_4, y_8, y_9\}$, $Y^4 = \{y_6\}$. Result of the microoperation encoding is shown in Table 2.1.

Analysis of this table shows that the CMO of MCU S_4 includes two decoders, because they are not needed for the fields B_2 and B_4 (Fig. 2.16).

Table 2.1 Encoding of the fields of compatible microoperations of MCU S_4

B_1		Y^1		B_2		Y^2		B_3		Y^3		B_4		Y^4	
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	y_1	0	1	y_7	0	1	y_4	1	y_6	1	y_8	1	y_9	1
1	0	y_3	1	0	y_7	1	0	y_8	—	—	—	—	—	—	—
1	1	y_5	1	1	*	1	1	y_9	—	—	—	—	—	—	—

Fig. 2.16 Organization of CMO with encoding of the fields of compatible microoperations for MCU S_4



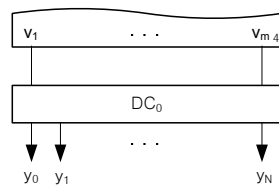
This approach provides less flexibility than the strategy of one-hot encoding of microoperations, because any modification of the sets $Y_q \subseteq Y$ may violate the compatibility for some fields of microinstruction. In this case some redesign of CMO is needed. Let us point out that modern FPLD provide very easy way to solve this problem due to their reprogramming ability. If the partition is performed in such a way that each field of microinstruction is assigned to one functional unit of the data-path (for example, microinstruction includes fields of adder, shifter, and so on), resulting control unit is more flexible. In this case modification of the system of microoperations does not violate the compatibility of microoperations. Disadvantage of this approach is the increase of operational part length, as compared with classical solutions [17].

The method of initial GSA verticalization is proposed in [17]. In this case all microoperations of GSA are compatible and the operational part of each microinstruction includes only one field having the bit capacity

$$m_4 = \lceil \log_2(N + 1) \rceil. \tag{2.18}$$

It permits to use only one decoder DC_0 for implementation of the CMO. This decoder has m_4 inputs and $N + 1$ outputs (Fig. 2.17). The purpose of additional signal y_0 will be explained later.

Fig. 2.17 Organization of CMO for MCU with full compatibility of microoperations



In the microprogramming theory a microprogram is called a vertical microprogram if each its microinstruction includes one microoperation only. Similarly, let a GSA Γ be called a vertical GSA (VGSA), if condition (2.19) holds for each vertex $b_q \in B_1$, and

$$|Y(b_q)| \leq 1. \tag{2.19}$$

Here $Y(b_q) \subseteq Y$ is the collection of microoperations from $b_q \in B_1$. As a rule, an arbitrary GSA Γ is not a vertical GSA, and should be transformed, to make all

microoperations compatible. This transformation is called a verticalization of GSA [17] and gives the vertical GSA $V(\Gamma)$ on the base of GSA Γ .

The verticalization is reduced to splitting the vertex $b_q \in B_1$ into $n_q = |Y(b_q)|$ operator vertices $b_q^1, \dots, b_q^{n_q}$, corresponding to the following condition:

$$\begin{aligned} |Y(b_q^i)| &= 1 & (i = 1, \dots, n_q); \\ Y(b_q^i) \cap Y(b_q^j) &= \emptyset & (i, j \in \{1, \dots, n_q\}, i \neq j); \\ Y(b_q^1) \cup \dots \cup Y(b_q^{n_q}) &= Y(b_q). \end{aligned} \quad (2.20)$$

Operator vertices of GSA Γ are transformed one-by-one and the following procedure is executed for each $b_q \in B_1$ [17]:

1. If $|Y(b_q)| \leq 1$, then go to step 7.
2. Exclude the arc $\langle b_t, b_q \rangle$ from the set of arcs E and add the arc $\langle b_t, b_q^1 \rangle$ to the set E .
3. Include the arcs $\langle b_q^i, b_q^{i+1} \rangle$, where $i = 1, \dots, n_q - 1$, to the set E .
4. Exclude the arc $\langle b_q, b_n \rangle$ from the set E and include the arc $\langle b_q^{n_q}, b_n \rangle$ to the set E .
5. Exclude the vertex b_q from the set of operator vertices B_1 and include the vertices $b_q^1, \dots, b_q^{n_q}$ to the set B_1 .
6. Include a unique microoperation $y_n \in Y(b_q)$ to each vertex $b_q^i (i = 1, \dots, n_q)$ in order to satisfy condition (2.20).
7. Ending the vertex $b_q \in B_1$ transformation.

Applying this procedure to the operator vertex b_3 of GSA Γ (Fig. 2.18a) we get a subgraph of vertical GSA $V(\Gamma)$ (Fig. 2.18b). The purpose of additional signal y_0 will be discussed later.

It follows from Fig. 2.18b, that microoperations y_2, y_3, y_4 are generated in series. In consequence, corresponding actions of the data-path are also executed in series and performance of the system is reduced. Besides, data dependence [17] between microoperations $y_n \in Y(b_q)$ may occur and they could not be executed consequently. For example, if there are microoperations $y_1, y_2 \in Y(b_2)$, such that $y_1 \# S := A + B$, $y_2 \# A := S \& C$, due to data dependence, any execution order of these microoperations ($\langle y_1, y_2 \rangle$ or $\langle y_2, y_1 \rangle$) leads to faulty calculations.

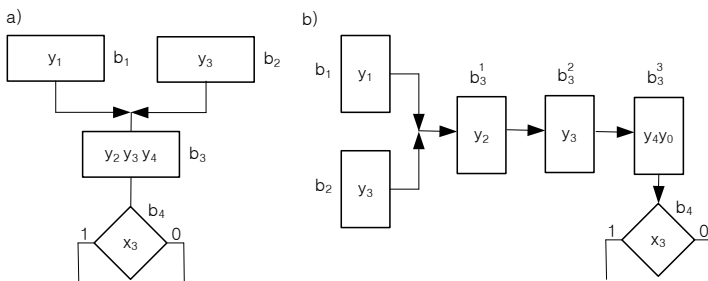


Fig. 2.18 Subgraph of GSA Γ before **a** and after **b** verticalization

The drawbacks mentioned above can be eliminated, if additional register RA is introduced between the control unit and data-path [17]. Microoperations $y_n \in Y(b_q)$ are loaded into this register one-by-one; special signal y_0 is generated simultaneously with generation of the microoperation from vertex $b_q^{n_q}$, initializing the data path operation. The tools of independent synchronization, available in modern FPLD, allow very easy implementation of this mode [65].

Let t_1, t_2 be average execution time of the algorithm represented by GSA Γ and VGSA $V(\Gamma)$ respectively, t_3, t_4 – the cycle time of data-path and MCU. Let each collection $Y(b_q)$ include k microoperations. Then k cycles of MCU are executed, in average, for each cycle of the data-path. Parameters t_1 and t_2 can be calculated as follows:

$$\begin{aligned} t_1 &= n(t_3 + t_4); \\ t_2 &= n(t_3 + kt_4), \end{aligned} \quad (2.21)$$

where n is an average number of digital system cycles needed to execute an algorithm represented by GSA Γ . As a rule, $t_3 > t_4$, let $t_3 = mt_4$. Let us find the relation between average times of algorithm execution for digital systems with control algorithms represented by GSA Γ and VGSA $V(\Gamma)$ correspondingly. This relation takes the form:

$$\eta = \frac{t_1}{t_2} = \frac{n(mt_4 + t_4)}{n(mt_4 + kt_4)} = \frac{m+1}{m+k}. \quad (2.22)$$

As can be seen in Fig. 2.19, where diagram of (2.22) is shown, if the control unit operation speed is greater, in comparison with the data-path operation speed (greater m), and if less microoperations in average each operator vertex includes (smaller k), then smaller is the digital system performance decrease, due to verticalization of the initial GSA Γ .

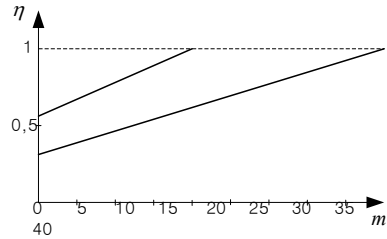


Fig. 2.19 Diagram of relation η

One method of reducing microinstruction length is the implicit representation of logical conditions [15]. In this case, code $K(x_l)$ of corresponding logical condition to be checked is placed in some dedicated bits of microinstruction address. This step allows excluding the field FX from microinstruction format. For example, the modified MCU structure with compulsory microinstruction addressing is shown in Fig. 2.20.

Operation principle of the MCU of Fig. 2.20 can be explained very easily. Most important is the fact that address register can be used to store some additional information. We use this property for optimization of the CMCU basic structure.

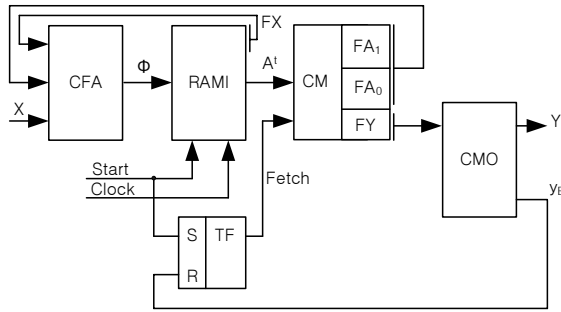


Fig. 2.20 Organization of MCU with compulsory microinstruction addressing and implicit representation of logical conditions

Many optimization methods are oriented towards checking more than one logical condition during one MCU operation cycle [17]. They are not interesting for us, because the FSM with microinstruction addressing, in case of CMCU, can check an arbitrary number of logical conditions $x_l \in X$ during one operation cycle.

2.3 Optimization of Mealy finite state machines

Synthesis methods, used for finite state machines with FPLDs, are very similar to the well known methods of combinational circuit design. Design of the FSM involves however execution of two specific steps, namely minimization of the number of states and the state assignment (state encoding). Successful solution of these problems permits optimizing such characteristics of control unit as the chip area occupied by FSM logic circuit, maximal frequency of its operation, and so on. Numerous methods of state minimization can be found in the literature, for example, in [68, 94]. We do not discuss them in our book, assuming that the FSM designed by means of the marked GSA have minimal possible number of states. Let us characterize briefly some methods of state assignment. This domain of computer science is explored for more than 40 years [11, 39, 94]. Modern state assignment methods are oriented towards the following goals:

- decrease of the chip area, occupied by control unit logic circuit [14, 21, 30, 40, 41, 43, 48, 49, 73, 77, 88, 94, 98],
- increase of the control unit performance [31, 32, 40, 43, 49, 87, 99],
- decrease of the consumed power [10, 26, 28, 51, 71, 99],
- increase of the circuit testability [37, 69].

Logical elements, used to implement FSM logic circuit have significant influence on the state assignment [55]. If the CPLD chips based on PAL or PLA macrocells are used to implement the FSM logic circuit, corresponding state assignment methods are oriented towards two-level minimization of the FSM combinational

part [4, 11, 12, 18, 34–36, 56–63, 78, 81, 90–94]. If, on the other hand, combinational part of FSM is implemented with FPGAs, corresponding state assignment methods involve the multi-level minimization [20, 47, 49, 53, 75, 76, 82, 85]. At present, the state assignment methods, based on symbolic minimization of systems of Boolean functions representing FSM combinational part are the most popular [37, 40, 67, 68, 97]. Besides, genetic algorithms are widely used to solve this problem [27, 28, 42, 96, 102].

Many modern CAD systems include special state encoding tools. The most popular among them, SIS [87, 88], is the developed version of the well known MIS II system [22], and uses two state assignment algorithms [68]:

- algorithm NOVA, oriented on two-level FSM implementation;
- algorithm JEDI, oriented on multi-level FSM implementation.

There are also such well-known systems as ASYL [83, 84], which is oriented to FSM circuit implementation with PLA and PROM, and the systems MUSTANG [38] and MUSE [40], oriented to FPGA. Some original optimization and synthesis methods are used in the systems DOMAIN [64], oriented to FPGA, and ZUBR [94], oriented to CPLD with PAL macrocells.

Let us discuss general approaches used to organize Mealy FSM logic circuits. Single-level combinational circuit (Fig. 2.7) corresponds to the single-level Mealy FSM logic circuit implementation (Fig. 2.21), which is a generalization of different structures described in [12, 15, 90, 94].

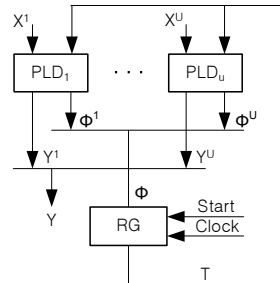


Fig. 2.21 Single-level Mealy FSM logic circuit implementation

In this figure $X^u \subseteq X$ is a set of logical conditions, used as literals of the SOP for Boolean functions $Y^u \subseteq Y$ and $\Phi^u \subseteq \Phi$, generated by microchip PLD_u ($u = 1, \dots, U$). Functions Y^u belong to a set of microoperations Y ; functions Φ^u to a set of input memory functions Φ . Codes of FSM states are kept in register RG and the set of state variables T is used to represent these codes.

For example, if PLAs are used to implement the single-level FSM circuit, a partition Π_F of the set of SOP is to be found. This partition should include minimal number of block terms E_1, \dots, E_U , where each block satisfies the following inequalities [12]:

$$\begin{aligned}
 |X^u| + R &\leq S; \\
 |Y^u| + |\Phi^u| &\leq t; \\
 |E_u| &\leq q.
 \end{aligned}
 \tag{2.23}$$

Constraints (2.23) should be modified according to the properties of logical element in use [94]. All these methods are based on using special partitions, which can be found in [68].

Blocks PLD_1, \dots, PLD_U form the combinational part of FSM, which is very often called a P block. The single-level FSMs are sometimes called P FSMs [90]. Main advantage of the P FSMs is very good performance, but it is sometimes accompanied by very high redundancy of logical elements in the FSM circuit, because different blocks can have the same input and output information [15].

There are many methods applied in order to minimize the hardware amount of Mealy FSM logic circuits [15, 90, 94]. Let us discuss some of them in detail.

Replacement of logical conditions. Let the transitions from state $a_m \in A$ depend on logical conditions x_l , of a set $X(a_m)$ with L_m elements ($m = 1, \dots, M$). As a rule, condition $L_m \ll L$ holds for real control algorithms [12]. If $G = \max(L_1, \dots, L_M)$, we create a new set of variables $P = \{p_1, \dots, p_G\}$. Let us replace the set X by the set P . For this end, for each state $a_m \in A$ we replace variable $x_l \in X(a_m)$ by the variable $p_g \in P$, such that if $A_m = 1$, then $p_g = x_l$ ($g = 1, \dots, G; l = 1, \dots, L$). It means that we have the following equation

$$p_g = \bigvee_{m=1}^M \bigvee_{l=1}^L C_{ml} A_m x_l (g = 1, \dots, G), \quad (2.24)$$

where C_{ml} is a Boolean variable, equal to 1 iff variable $p_g \in P$ replaces the variable $x_l \in X$ in the state $a_m \in A$.

After this change of variables we can replace the systems

$$Y = Y(T, X), \quad (2.25)$$

$$\Phi = \Phi(T, X) \quad (2.26)$$

by the following ones

$$P = P(T, X), \quad (2.27)$$

$$Y = Y(T, P), \quad (2.28)$$

$$\Phi = \Phi(T, P). \quad (2.29)$$

Systems (2.27) – (2.29) describe the so-called MP Mealy FSM [12], having the structure shown in Fig. 2.22. Here the block M implemented with multiplexers generates functions (2.27), and the block P implemented with PLDs generates functions (2.28) – (2.29).

It is obvious that multiplexers of the block M can be implemented using PLA, PAL or FPGA. Different methods of hardware optimization for the block M are known [15]. For example, in case of optimal state encoding, information about logical conditions to be checked is coded in the leftmost bits of state codes. The FSMs based on this principle are called MPC Mealy FSM. Second approach is connected with transformation of state codes into the codes of logical conditions. The FSM based on this principle is called a MPL Mealy FSM.

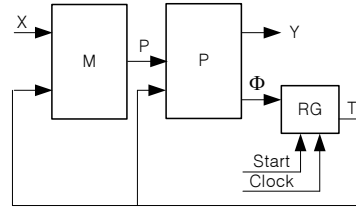


Fig. 2.22 Organization of MP Mealy FSM

Besides, optimization of the block M can be executed due to the transformation of initial GSA in such a way, that each FSM state satisfies the following condition

$$|X(a_m)| \leq i. \tag{2.30}$$

This approach leads to the families of MiP-, MiPC- and MiPL Mealy FSMs, where $i = 1, \dots, G$, where the value of G depends on characteristics of the initial GSA.

Let us point out, that replacement of logical conditions makes sense, when PLDs in use have restricted number of inputs.

Encoding of collections of microoperations. System (2.25) corresponds to one-hot encoding of microoperations. Let us encode each collection of microoperations $Y_q \subseteq Y$ ($q = 1, \dots, Q$) by the binary code $K(Y_q)$ with $R_1 = \lceil \log_2 Q \rceil$ bits and use the variables $z_r \in Z$, where $|Z| = R_1$. In this case, Mealy FSM can be implemented as the PY Mealy FSM (Fig. 2.23).

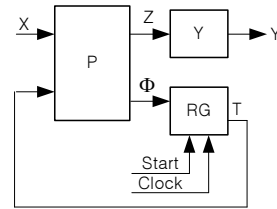


Fig. 2.23 Organization of PY Mealy FSM

In PY Mealy FSM, block P implements input memory functions represented by (2.26) and the system of encoding variables

$$Z = Z(T, X). \tag{2.31}$$

A block Y implements the system of microoperations, represented as

$$Y = Y(Z). \tag{2.32}$$

This approach can be applied for all types of FPLDs and the best way to implement the block Y is to use memory blocks, such as PROMs or RAMs.

Encoding of the fields of compatible microoperations. This method is the same as the one used in case of MCU. Its application, in case of Mealy FSM, gives the PD Mealy FSM, shown in Fig. 2.24.

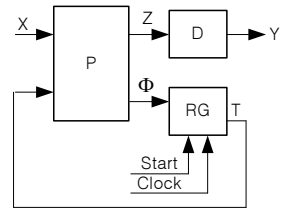


Fig. 2.24 Organization of PD Mealy FSM

In the PD Mealy FSM, block P implements functions (2.26), (2.31), and the block D with decoders generates microoperations represented by (2.32). Decoders of the block D can be of course implemented using any PLDs.

Let I be the minimal number of the classes of compatible microoperations, which can be found using classical methods [17]. In this case, block D includes I decoders. The number of decoders can be reduced using the transformation of initial GSA Γ [4, 15], permitting to reduce the number of compatibility classes in comparison with this fixed value of I . The extreme case of this transformation is the vertical GSA $V(\Gamma)$, with only one class of compatible microoperations ($I = 1$). Thus, transformation of GSA Γ leads to PD_{1-} , PD_{2-} , ..., PD_{I-} Mealy FSM with different hardware amounts.

Encoding of structure table lines. This method is based on encoding of each line of the structure table F_h by a binary code $K(F_h)$ with $R_1 = \lceil \log_2 H \rceil$ bits; using variables $z_r \in Z$ [90]. We obtain the PH Mealy FSM (Fig. 2.25), in which the block P implements system (2.31), and block H systems (2.32) and

$$\Phi = \Phi(Z). \tag{2.33}$$

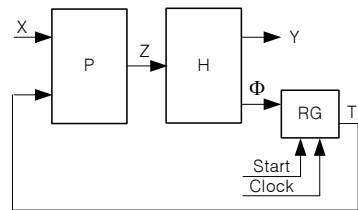


Fig. 2.25 Organization of PH Mealy FSM

Application of PROMs or RAMs chips offers the simplest way to implement the block H.

Non-standard representation of terms. According to [13, 15], the terms corresponding to the structural table lines can be represented as pairs $\langle a_m, a_s \rangle$, where a_m is a current state of FSM and a_s is next state ($a_m, a_s \in A$). This approach permits to simplify block CMO and leads to PR Mealy FSM (Fig. 2.26).

Here, the block P implements the system (2.26), register RG_1 keeps the code of next state, represented by state variables $T_r \in T$, register RG_2 keeps the code of current state, represented by state variables $v_r \in V$ ($|T| = |V| = R$), and block R

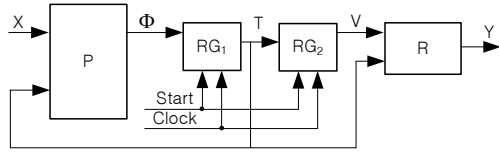


Fig. 2.26 Organization of PR Mealy FSM

generates the functions

$$Y = Y(V, T). \tag{2.34}$$

Comparison of systems (2.25) and (2.34) shows that variables V replace variables X in (2.34). As $R \ll L$, this replacement allows decreasing of hardware amount in comparison with the equivalent P Mealy FSM. Besides, the block P of PR Mealy FSM is not used to generate additional variables Z , which are formed by PY- or PD Mealy FSM. It gives the reduction of hardware amount, in comparison with FSMs mentioned above.

If the terms, corresponding to lines of structure table, are represented by pairs $\langle K(a_m), K(Y_q) \rangle$, we obtain the PT Mealy FSM (Fig. 2.27) [15].

In this case, the block P implements functions (2.31), block Y generates microoperations Y represented by (2.32), and block H implements functions Φ , represented as:

$$\Phi = \Phi(T, Z). \tag{2.35}$$

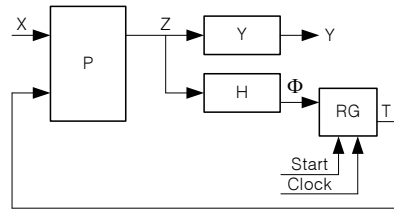


Fig. 2.27 Organization of PT Mealy FSM

Comparison of systems (2.26) and (2.35) shows that in case of PT Mealy FSM, variables Z replace variables X in the system Φ . Since $|Z| \ll L$, this approach allows reduction of hardware amount, in comparison with the equivalent P- and PY Mealy FSMs.

Transformation of object codes. This approach, which can be found in [15, 16], defines the internal states and collections of microoperations as objects of FSM. Some additional variables $v_r \in V$ may be needed to express one object as a function of another object. This approach gives the PF Mealy FSM of the first type (Fig. 2.28) or second type (Fig. 2.29), denoted as PF_1 and PF_2 Mealy FSM respectively.

$$T = T(V, Z). \tag{2.36}$$

It is clear that PF_1 Mealy FSM can be considered as the development of PR Mealy FSM and PF_2 Mealy FSM as some development of PT Mealy FSM. Let us

Fig. 2.28 Organization of PF_1 Mealy FSM

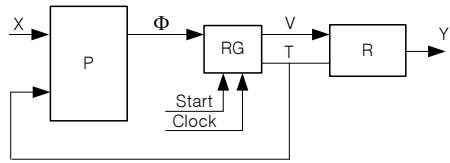
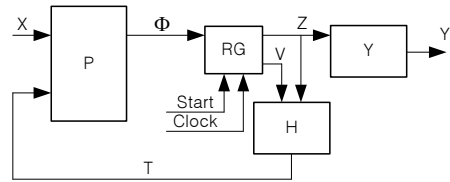


Fig. 2.29 Organization of PF_2 Mealy FSM



point out that blocks P of PF_1 - and PF_2 Mealy FSMs generate more output variables than their analogies, but their blocks R and H have smaller number of inputs.

Combined application of the two methods explained above leads to the three- and four-level organizations of Mealy FSM logic circuits [15]. All possible structures of Mealy FSM logic circuits are shown in Table 2.2.

Table 2.2 Multilevel organizations of Mealy FSM

Levels	A	B	C	D
Blocks	M_1	P	H	2 Y 1
	M_1C		R	2 D_1 1
	M_1L		Y	1 \vdots
	\vdots		D_1	1 D_l 1
	M_G		\vdots	
	M_GG		D_l	1
	M_GL			

Generation of multilevel Mealy FSM logic circuit structures can be interpreted as a word-formation process [15], where the level A is used as a prefix of word S, the level B as its base, the level C either as its suffix or ending, and the level D as its ending. It is clear that the base should always be present in a word, but other attributes can be absent. The digit 1 in Table 2.2 means that a given block is treated as ending of a word and digit 2 means that a given block is treated as its suffix. For example, the word $S = M_1 * P * H * Y$ determines Mealy FSM with replacement of logical conditions and initial GSA transformation. Each transition depends here on at most one logical condition. Encoding of lines of the structure table, encoding of collections of microoperations and organization of the FSM is shown in Fig. 2.30. In this M_1PHY , Mealy FSM variables from a set Z encode the structure table lines and variables from a set V encode collections of microoperations $Y_q \subseteq Y$.

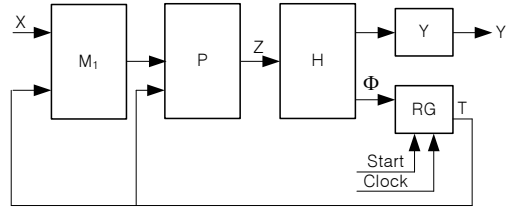


Fig. 2.30 Organization of M_1PHY Mealy FSM

Let us form Table 2.3 to calculate the total number of possible organizations of FSM circuit. Let V_j be the number of certain structures of FSM circuit with j levels and V_0 be the total number of possible organizations of the FSM circuit with j levels ($j = 1, \dots, 4$). It can be found that Table 2.3 represents

$$V_0 = V_1 + \dots + V_4 = 4 + 18G + I + 9IG \tag{2.37}$$

different structures of the Mealy FSM logic circuit organizations. It can be found in [11] that for the FSM of average complexness we have $I = G = 6$ and in consequence expression (2.37) gives $V_0 = 442$ different structures of Mealy FSM logic circuit organizations.

If the interpreted GSA Γ is a linear GSA, the FSM logic circuit hardware amount can be reduced due to the use of a counter instead of a register to keep the FSM state codes [15]. This control unit is called C FSM [15]. Design process of the C FSM involves generation of some linear sequences of states $\alpha_g = \langle a_{g_1}, \dots, a_{g_{F_g}} \rangle$, where a transition $\langle a_{g_i}, a_{g_{i+1}} \rangle$ ($a_g, a_{g_i} \in A$) exists for any pair of adjacent components of a vector ($i = 1, \dots, F_g$).

Table 2.3 Estimation of total number of Mealy FSM logic circuit organization

Single level	k_1	Two levels	k_2	Three levels	k_3	Four levels	k_4		
P	1	M_gP	G	M_gPH	G	M_gPHY	G		
		M_gCP	G	M_gCPH	G	M_gCPHY	G		
		M_gLP	G	M_gLPH	G	M_gLPHY	G		
		PH	1	M_gPR	G	M_gPHD_i	IG		
		PR	1	M_gCPR	G	M_gCPHD_i	IG		
		PY	1	M_gLPR	G	M_gLPHD_i	IG		
		PD_i	1	M_gPY	G	M_gPRY	G		
				M_gCPY	G	M_gCPRY	G		
				M_gLPY	G	M_gLPRY	G		
				M_gPD_i	IG	M_gPRD_i	IG		
				M_gCPD_i	IG	M_gCPRD_i	IG		
				M_gLPD_i	IG	M_gLPRD_i	IG		
		$V_1 = 1$		$V_2 = 3 + 3G + I$		$V_3 = 9G + 3IG$		$V_4 = 6G + 6IG$	

Obviously, the linear sequences of states are some analogues of operational linear chains of CMCU, but there is one difference: conditional transitions between states

of such linear sequences are possible. Design of the C Mealy FSM requires special state encoding, for which the following condition holds:

$$K(a_{g_{i+1}}) = K(a_{g_i}) + 1, \quad (2.38)$$

where $i = 1, \dots, F_g$, $g = 1, \dots, G$, G is the total number of sequences for GSA Γ . Let a symbol P_c stand for the Mealy FSM with counter CT keeping state codes. Organization of P_c Mealy FSM is shown in Fig. 2.31.

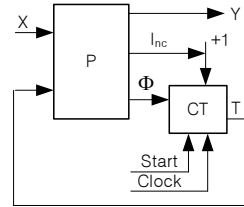


Fig. 2.31 Organization of P_c Mealy FSM

Block P generates functions (2.25) – (2.26) using a special signal

$$I_{nc} = f(T, X), \quad (2.39)$$

which serves to increment the content of counter CT in order to obtain mode (2.38). All optimization methods mentioned above can be used to reduce the hardware amount of the C FSM logic circuit. In consequence we can double the value of V_0 .

Such variety of Mealy FSM logic circuit organizations requires the development of formal methods used to find the structure with minimal hardware amount and performance corresponding to technical requirements of particular task. Let us point out that solution of this problem does not exist up to now.

2.4 Optimization of Moore finite state machines

In the simplest case a Moore FSM can be implemented as a single-level sequential device (Fig. 2.32), called by analogy to the single-level Mealy FSM a P Moore FSM [15, 90].

Blocks PLD_u implement input memory functions Φ^u , belonging to a set Φ , and output functions Y^u , from a subset Y_0 of a set Y ($u = 1, \dots, U$). Blocks $PROM_j$ implement output functions Y^{U+j} ($j = 1, \dots, J$), belonging to the complement of the set Y_0 up to the set Y . As in case of P Mealy FSM, this circuit organization is characterized by maximal performance due to maximal hardware amount. Optimization of Moore FSM logic circuits can be realized using some methods discussed in Section 2.3, as well as some specific approaches based on existence of pseudoequivalent

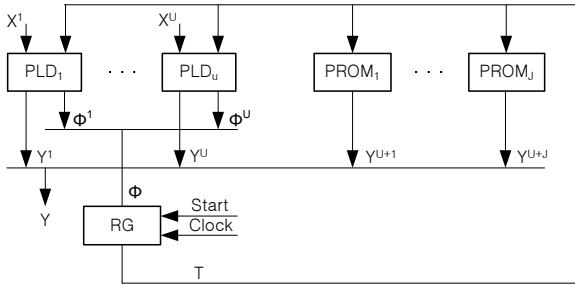


Fig. 2.32 Single-level Moore FSM implementation with PLD

states [13]. Let us point out that classes of pseudoequivalent states are equivalent to the input constraints [31, 67, 68, 98] used, for example, in the program NOVA [99].

There are three main methods used to reduce the hardware amount of Moore FSM logic circuit. They are based on partition of the set of FSM states A , into the classes of pseudoequivalent states with I blocks, considered previously in [14, 15]. We have:

- optimal state assignment;
- transformation of state codes into codes of classes of pseudoequivalent states;
- transformation of initial GSA.

The principle of *optimal state encoding* was already discussed in Section 1.2. Application of this method gives the $P_E Y$ Moore FSM of Fig. 2.33. Let us point out that the structures of P and $P_E Y$ Moore FSM are identical.

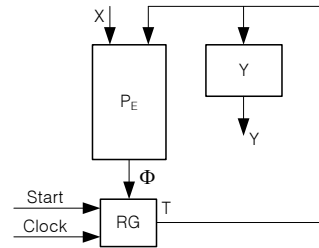


Fig. 2.33 Organization of Moore FSM with optimal state encoding

The block P_E of Fig. 2.33 implements the system Φ specified by (2.26), and block Y corresponds to

$$Y = Y(T). \tag{2.40}$$

Obviously, some functions $y_n \in Y$ can be implemented using free resources of the block P , but in order to simplify our discussion, this fact is not shown in Fig. 2.33.

Transformation of state codes into codes of the classes of pseudoequivalent states can be used when optimal encoding is not possible for some classes $B_i \in \Pi_A$.

Assume that for some Moore FSM S we have got the following partition: $\Pi_A = \{B_1, \dots, B_4\}$, $B_1 = \{a_1\}$, $B_2 = \{a_2, a_3, a_4\}$, $B_3 = \{a_5, a_6, a_7\}$, $B_4 = \{a_8\}$. Now $R = 3$ and Karnaugh map (Fig. 2.34) shows the result of optimal state encoding for the Moore FSM S .

		T_2T_3			
		00	01	11	10
T_1	0	a_1	a_2	a_3	a_4
	1	a_5	a_6	a_7	a_8

Fig. 2.34 Optimal state encoding for the Moore FSM S

It follows from the Karnaugh map that class B_1 corresponds to the code $K(B_1) = 000$, class B_2 is split by classes $B_2^1 = \{a_2\}$ with code $K(B_2^1) = 001$ and $B_2^2 = \{a_3, a_4\}$ with code $K(B_2^2) = 01*$. The class B_3 is split by classes $B_3^1 = \{a_5\}$ with code $K(B_3^1) = 100$ and $B_3^2 = \{a_6, a_7\}$ with code $K(B_3^2) = 1*1$ and the class B_4 corresponds to code $K(B_4) = 110$. Let H_i be the number of transitions from state $a_m \in B_i$ ($i = 1, \dots, I$) and assume that in case of Moore FSM S we have: $H_1 = 3, H_2 = 6, H_3 = 4, H_4 = 2$. In consequence, the structure table of an equivalent Mealy FSM has $H_0 = 15$ lines. Transformed structure table of the Moore FSM S has $H = H_1 + 2H_2 + 2H_3 + H_4 = 25$ lines, which means that this result is far from optimum.

Let us encode each class $B_i \in \Pi_A$ by a binary code $K(B_i)$ with $R_0 = \lceil \log_2 I \rceil$ bits and use variables $\tau_r \in \tau$ for the encoding, where $|\tau| = R_0$. Now, Moore FSM can be implemented as $P_T Y$ Moore FSM [15], where code transformer TC implements the transformation of state codes into codes of the classes of pseudoequivalent states (Fig. 2.35).

In case of $P_T Y$ Moore FSM, block P_T implements the following input memory functions

$$\Phi = \Phi(\tau, X), \tag{2.41}$$

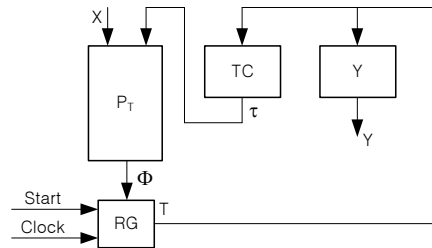


Fig. 2.35 Organization of Moore FSM with transformation of state codes

Block Y implements (2.39), and TC implements the system

$$\tau = \tau(T). \tag{2.42}$$

This approach allows reduction of the number of lines of the Moore FSM structure table down to H_0 .

In case of Moore FSM S, there are four equivalence classes ($I = 4$) and therefore $R_0 = 2$, $\tau = \{\tau_1, \tau_2\}$. Let these classes be encoded as follows: $K(B_1) = 00, \dots, (B_4) = 11$. Now, Table 2.4 represents the behavior of block TC.

Table 2.4 Table of code transformer of the Moore FSM S

a_m	$K(a_m)$	B_i	$K(b_i)$	τ_m	m	a_m	$K(a_m)$	B_i	$K(b_i)$	τ_m	m
a_1	000	B_1	00	τ_1	1	a_5	100	B_3	10	τ_1	5
a_2	001	B_2	01	τ_2	2	a_6	101	B_3	10	τ_1	6
a_3	010	B_2	01	τ_2	3	a_7	110	B_3	10	τ_1	7
a_4	011	B_2	01	τ_2	4	a_8	111	B_4	11	$\tau_1 \tau_2$	8

In this table, column τ_m contains variables $\tau_r \in \tau$, equal to 1 in code $K(b_i)$, where $a_m \in B_i$. From this table we find system (2.42), in which; for example, $\tau_1 = A_5 \vee \dots \vee A_8 = T_1$. Let us point out that minimization of system (2.42) is needed only for the implementation using PAL, PLA or FPGA macrocells.

This approach provides minimal hardware amount of the block P, but some FPLD resources are needed to implement the block TC. Let us point out that introduction of TC does not deteriorate FSM performance in comparison with $P_E Y$ Moore FSM, because variables $\tau_r \in \tau$ are generated concurrently with microoperations $y_n \in Y$.

Method of **initial GSA transformation** permits to get an FSM logic circuit with average characteristics in comparison with two other methods discussed in this section [15]. Let us consider the subgraph of GSA Γ (Fig. 2.36a), corresponding to some class $B_2 \in \Pi_A$ of Moore FSM S.

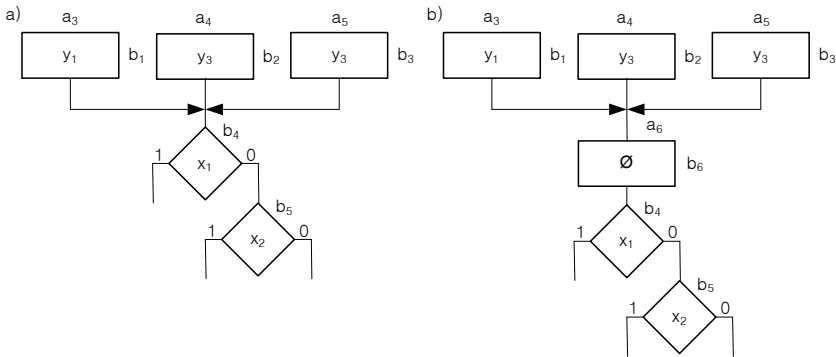


Fig. 2.36 Subgraph of GSA Γ before **a** and after **b** transformation

The class $B_2 = \{a_3, a_4, a_5\}$ corresponds to the subtable of FSM structure table including $H_2 = 3 \cdot 3 = 9$ lines. Let us introduce vertex b_6 (Fig. 2.36b). Now, the

class B_2 corresponds to a subtable of FSM structure table having $H_2 = 3 \cdot 1 = 3$ lines, and state a_6 corresponds to a subtable with $H_6 = 3$ lines. Next, the number of FSM transitions corresponding to the subgraph of GSA Γ is reduced to $\Delta H = H_2 - H_6 = 6$ lines. Usually, the number of FCM structure table lines is reduced to

$$\Delta H = \sum_{i=1}^I H_i(K_i - 1) \quad (2.43)$$

where H_i is the number of transitions from some state $a_m \in B_i$, $K_i = |B_i|$ ($i = 1, \dots, I$). This approach leads to $P_F Y$ Moore FSM with the same logic circuit structure as in case of $P_E Y$ Moore FSM given in Fig. 2.33.

Further logic circuit optimization can be reached due to increase the number of its levels. Let us discuss briefly same approaches which make it possible.

Replacement of logical conditions is executed similarly as in case of MP Mealy FSM and leads to the MPY Moore FSM of Fig. 2.37.

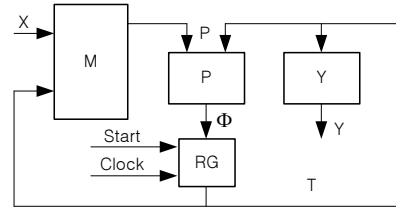


Fig. 2.37 Organization of MPY Moore FSM

The hardware amount of block M can be reduced using either a special state encoding (see Section 2.3), resulting in the MPCY Moore FSM, or generation of logical condition codes (see Section 2.3), resulting in the MPLY Moore FSM. The number of inputs in block M can be changed due to the GSA transformation (see Section 2.3), after which we obtain $M_g PCY$ -, $M_g PLY$ - or $M_g PY$ Moore FSM ($g = 1, \dots, G$). All these methods can be applied for each of the PY-, $P_E Y$ - and $P_F Y$ Moore FSMs.

Maximal encoding of collections of microoperations involves introducing a special transformer TC_1 generating codes of microoperation collections with $R_1 = \lceil \log_2 Q \rceil$ bits, where Q is the number of collections, obtained from state codes. This approach gives the PTY Moore FSM.

Encoding of the fields of compatible microoperations involves introducing a special transformer TC_2 converting state codes into codes of the fields of compatible microoperations. This approach gives the PTD Moore FSM.

In both cases some additional variables from a set Z are used for encoding microoperations or their collections. Generalized structure of the Moore FSM is shown in Fig. 2.38. In case of maximal encoding of collections of microoperations blocks CMO and TC_i represent blocks Y and TC_1 respectively. In case of maximal encoding of the fields of compatible microoperations, the blocks CMO and TC_i represent blocks Y and TC_2 correspondingly.

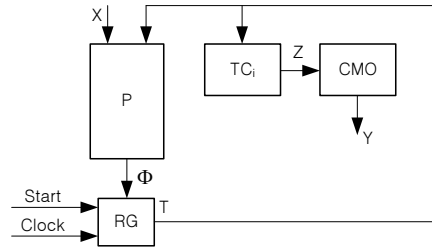


Fig. 2.38 Generalized structure of Moore FSM logic circuit

Both approaches belong to the class of object transformation methods, where states are transformed into microoperations.

Encoding of structure table lines has no sense in case of Moore FSM, because in this case block P generates the input memory functions Φ only.

Transformation of object codes is reduced either to the transformation of binary codes of microoperation collections and of some additional variables V into state codes (PY_A Moore FSM), or to the transformation of the fields of compatible microoperations and of additional variables V into state codes (PD_A Moore FSM). The generalized structure of the Moore FSM logic circuit for these cases is shown in Fig. 2.39.

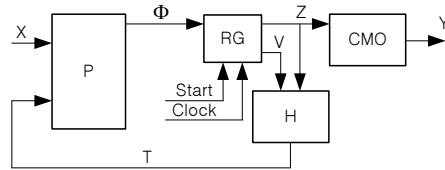


Fig. 2.39 Generalized structure of Moore FSM logic circuit with object transformation

Codes of microoperation collections are represented here by variables from a set Z and block H generates the functions

$$T = T(V, Z). \tag{2.44}$$

Block CMO represents either the block Y (PY_A Moore FSM), or the block D (PD_A Moore FSM).

Mutual application of optimization methods mentioned above leads to three- and four-level structures of Moore FSM logic circuits. All possible structures are listed in Table 2.5.

Comparison of Tables 2.2 and 2.5 leads to the conclusion that implementation of the Moore FSM logic circuit requires 4 times greater number of structures than in case of the equivalent Mealy FSM. It is due to the fact that basic level B of the Moore FSM contains 4 different types of blocks P, whereas the other levels of both tables have equal number of elements. Transformation of expression (2.37) allows estimating the number of different structures of Moore FSM logic circuit, which is equal to:

Table 2.5 Multilevel structures of Moore FSM logic circuits

Levels	A	B	C	D
Blocks	M_1	P	Y 1	Y 1
	M_1C	P_E	D_1 1	D_1 1
	M_1L	P_T	\vdots	\vdots
	\vdots	P_Γ	D_I 1	D_I 1
	M_G		TC_1 2	
	M_GC		TC_2 2	
	M_GL			

$$V_0 = 16 + 72G + 4I + 36IG. \quad (2.45)$$

In case of FSM with $G = I = 6$, expression (2.45) gives $V_0 = 1768$ different structures of Moore FSM logic circuit.

Let us point out that this number can be doubled using linear GSA, because in this case a counter or a register can be used to keep the state codes [15].

2.5 Control unit design with FPLDs

Design is considered here as some process generating detailed specifications, allowing fabricating the target product with specific properties and figures of merit, such as performance and area. This process can be reduced to repeated decomposition of the initial object specification into sub-objects. It is accomplished when each sub-object can be implemented using standard library elements. Each design step is accompanied by synthesis, which means that an object function is transformed into its structure. Next design step is the analysis, during which verification of synthesis results is executed. Results of analysis may require repeated synthesis with optimization of some figures of merit. Let us discuss some types of models, abstraction levels and digital system specifications, using some results found in the literature [65, 68].

Huge complexity of modern digital systems requires application of computer-aided design (CAD) tools, which help us to represent physical objects by their models. A circuit model can be viewed as some abstraction describing selected features of the system, which are sufficient to perform synthesis procedure. There are two forms of model classification, namely levels of abstraction and views. The following levels and views are used in models of digital devices:

1. Architectural level, where a device performs some set of operations, such as data computation or transfer. This level is called a-level.
2. Logic level, where a device evaluates a set of logical functions, such as AND, OR, NOT. This level is called l-level.

3. Geometrical level, where a device is represented as a set of geometrical entities, such as floor plans or layouts. This level is called g-level.
4. Behavioral view describes the device function regardless of its implementation. This view is called b-view.
5. Structural view describes a model as interconnection of specific components. This view is called s-view.
6. Physical view describes the device with help of some physical objects, such as transistors. This view is called p-view.

Models of different levels can be seen under different views. This fact is reflected in Fig. 2.40, taken from [68]. This representation is often referred as Gajski and Kuhn Y-chart.

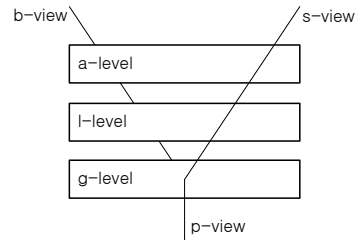


Fig. 2.40 Abstraction levels and views of digital device specification

Synthesis of control units belongs to the logic level of digital system design. Model of the control unit can be represented either as a finite state machine state diagram, or as a netlist, or as a text using some standard hardware description language (HDL). In our book we have chosen one of the register transfer level languages, namely the language of graph-scheme of algorithms, as the tool of initial specification of control unit. This language is chosen due to its clearness allowing simple explanation of particular properties of compositional microprogram control units design and optimization. The outcome of logic design is the structure specification of a target circuit, as for example the gate-level netlist.

Analysis of the literature, such as for example [9, 19, 45, 46, 65, 80], shows that design process used for digital devices is a formalized automatic process oriented to particular logical elements. As a rule, an object to be designed is specified using some HDL language, such as VHDL [23, 72] or Verilog [24], or System C [23]. These languages allow control unit description on the register transfer level (RTL). This description is next transformed using some standard synthesis tools. If the field-programmable gate arrays chips are used as logical elements, simplified diagram of the design process has the form shown in Fig. 2.41.

The control unit specification on RTL level is simulated using simulation part 1 of particular CAD tools. All deviations from the target device function are corrected during this process. This correct model is next processed by the synthesis part of CAD system, which performs the project optimization (minimizing the area and maximizing performance) and generates the netlist of control unit. This netlist serves as a base for more detailed simulation of the circuit by simulation part 2 of

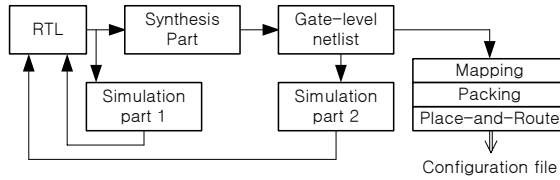


Fig. 2.41 Digital device design process with FPGA

the CAD system. The corrected netlist is next used to generate the configuration file. This step consists on finding correspondence between logical elements and LUT elements (mapping). These LUT elements are then combined into logical blocks (packing), which are placed on the chip area (placement) and connected using some resources of the programmable interconnection array (routing).

The outcome of this process is the configuration file, used to program the FPGA chip. Design process of the control unit implemented with CPLD chips is very similar to the one from Fig. 2.41. This process has however some particular features described in [65]. Moreover, all tools used in the design process of this kind are objects of constant improvement, because semiconductor industry offers frequently new architectures of FPLD chips.

Program tools oriented towards the design using FPLD are developed by numerous companies. For example, Altera manufactured such systems as MAX+PLUS II, Quartus, Quartus II ; Xilinx proposed the ES Series . There exist several powerful tools used to specify control units by state diagrams, such as Foundation developed by Aldec for Xilinx; system Renoir of Mentor Graphics; programs StateCAD from the package Workview of Viewlogic and other [1–3]. There exist also numerous academic systems, oriented on the FPGA design (as for example, DOMAIN system [64]), or CPLD based with PAL macrocells (as for example, ZUBR [93]), or CPLD based on PLA macrocells (for example, SIS [87, 88]).

Multiplicity of existing logical elements and design tools bring us to the conclusion that every useful book on this area should include synthesis methods, useful for creating some CAD tools. These methods should be free from any specific features of the families of logical devices produced by particular industrial companies. We have chosen this way, because the second solution implies rigid orientation towards some particular chips and CAD tools. In that case our book would be the object of interest to a limited circle of specialists involved in particular design processes and using chips and CAD tools manufactured by a particular company. As the consequence of our choice, outcomes of all original synthesis methods of compositional microprogram control units can be presented in the form of tables, which can be used to obtain systems of Boolean functions describing some parts of CMCUs. It is clear that these Boolean functions can be represented in the form of netlists. Thus, we can say that our book is devoted to methods of logical synthesis of the compositional microprogram control units. Their behavior is specified by means of the language of graph-schemes of algorithms. Specifications of all CMCU blocks are represented in tabular form, allowing generation of systems of Boolean functions

and netlists of logical elements and their interconnections. As a rule, the design methods proposed in this book allow to reduce the number of function arguments and are oriented towards FPGA implementation. Corresponding design methods allowing reduction of the number of product terms in these functions are oriented towards CPLD with PAL and PLA macrocells. In order to unify the diagrams used in the book, we use the PLA implementation of the combinational parts of CMCU, but the reader should remember that all types of logical elements can be used to implement this part of the system he wants to design.

References

1. <http://www.altera.com>.
2. <http://www.atmel.com>.
3. <http://www.xilinx.com>.
4. A. A. Barkalov and A. Bucowiec. Synthesis of mealy finite-states machines for interpretation of verticalized flow-charts. *Theoretical and applied informatics*, 39–51, 2005.
5. M. Adamski and A. Barkalov. *Architectural and Sequential Synthesis of Digital Devices*. University of Zielona Góra Press, 2006.
6. T. Agerwala. Microprogram optimization: a survey. *IEEE Transactions of Computers*, (10):962–973, 1976.
7. A. Agrawala and T. Rauscher. *Foundations of Microprogramming*. Academic Press, New York, 1976.
8. F. Anceau. *The Architecture of Microprocessors*. Addison-Wesley, Workingham, 1986.
9. P. Asahar, S. Devidas, and A. Newton. *Sequential Logic Synthesis*. Kluwer Academic Publishers, Boston, 1992.
10. P. Bacchetta, L. Daldos, D. Sciuto, and C. Silvano. Low-power state assignment techniques for finite state machines. In *Proc. of the IEEE Inter. Symp. on Circuits and Systems (ISCAS'2000)*, volume 2, pages 641–644, 2000.
11. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
12. S. I. Baranov and V.A. Skljarov. *Digital Units on Programmable LSI with Matrix Structure*. Radio i Swiaz, 1986.
13. A. A. Barkalov. *Development of formalized methods of structure synthesis for compositional automata*. PhD thesis, Donetsk:DonSTU, 1994. (in Russian).
14. A. A. Barkalov. Principles of optimization of logic circuit of Moore FSM. *Cybernetics and System Analysis*, (1):65–72, 1998. (in Russian).
15. A. A. Barkalov. *Synthesis of Control Units with PLDs*. Donetsk National Technical University, 2002. (in Russian).
16. A. A. Barkalov and A.A. Barkalov. Design of Mealy finite-state machines with transformation of object codes. *Inter. Journal "Applied Mathematics and Computer Science"*, 15(1):151–158, 2005.
17. A. A. Barkalov and A. V. Palagin. *Synthesis of Microprogram Control Units*. IC NAS of Ukraine, Kiev, 1997. (in Russian).
18. A. A. Barkalov, M. Węgrzyn, and R. Wiśniewski. Partial reconfiguration of compositional microprogram control units implemented on fpgas. In *Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems (Brno)*, pages 116–119, 2006.
19. M. Bolton. *Digital System Design with Programmable Logic*. Addison-Wesley, Boston, 1990.
20. B.W. Bomar. Implementation of microprogrammed control in FPGAs. *IEEE Transactions on Industrial Electronics*, 49(2):415–422, 2002.

21. R. Brayton, G. Hatchel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, 1984.
22. R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: a multi-level logic optimization system. *IEEE Transactions on Computer-Aided Design*, 6:1062–1081, 1987.
23. S. Brown and Z. Vernesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw–Hill, 2000.
24. S. Brown and Z. Vernesic. *Fundamentals of Digital Logic with Verilog Design*. McGraw–Hill, 2003.
25. C. Webb C and J. Liptay. A high-frequency custom cmos s/390 microprocessor. *IBM Journal of research and Development*, 41(4/5):463–473, 1997.
26. C. Cao, B. O’Nils, and D. Oelmann. A tool for low-power synthesis of FSMs with mixed synchronous/asynchronous state memory. In *Proc. of Norchip Conf.*, pages 199–202, 2004.
27. S. Chattopadhyay. Area conscious state assignment with flip-flop and output polarity selection for finite state machines synthesis – a genetic algorithm. *The Computer Journal*, 48(4):443–450, 2005.
28. S. Chattopadhyay and P. Chaudhuri. Genetic algorithm based approach for integrated state assignment and flipflop selection in finite state machines synthesis. In *Proc. of the IEEE Inter. Conf. on VLSI Design*, pages 522–527, Los Alamitos, 1998. IEEE Computer Society.
29. Y. C. Chu. *Computer Organization and Microprogramming*. Prentice Hall, 1972.
30. M. Ciesielski and S. Jang. PLADE: a two-stage PLA decomposition. *IEEE Transactions on Computer-Aided Design*, 11(8), 1992.
31. R. Czerwinski and D. Kania. State assignment method for high speed FSM. In *Proc. of Programmable Devices and Systems [33]*, pages 216–221.
32. R. Czerwinski and D. Kania. State assignment method for high speed FSM. In *Proc. of Programmable Devices and Systems [33]*, pages 216–221.
33. R. Czerwinski and D. Kania. State assignment method for high speed FSM. In *Proc. of Programmable Devices and Systems*, pages 216–221, 2004.
34. D. Debnath and T. Sasao. Multiple-valued minimization to optimize PLA with output EXOR gates. In *Proc. of IEEE Inter. Symp. on Multiple-Valued Logic*, pages 99–104, 1999.
35. Sasao T. Debnath D. Doutput phase optimization for and-or-exor plas with decoders and its application to design of adders. *IFICE Transactions on Information and Systems*, E88-D(7):1492–1500, 2005.
36. S. Deniziak and K. Sapiiecha. An efficient algorithm of perfect state encoding for cpld based systems. In *Proceedings of IEEE Workshop on Design and Diagnostic of Electronic Circuits and Systems (DDECS’98)*, pages 47–53, 1998.
37. S. Devadas and H. Ma. Easily testable PLA-based finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(6):604–611, 1990.
38. S. Devadas, H. Ma, A. Newton, and A. Sangiovanni-Vincentelli. MUSTANG: State assignment of finite state machines targeting multilevel logic implementation. *IEEE Transactions on Computer-Aided Design*, 7(12):1290–1300, 1988.
39. S. Devadas and A. Newton. Exact algorithms for output encoding, state assignment, and four-level boolean minimization. *IEEE Transactions on Computer-Aided Design*, 10(1):143–154, 1991.
40. X. Du, G. Hachtel, B. Lin, and A. Newton. MUSE: a multilevel symbolic encoding algorithm for state assignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):28–38, 1991.
41. B. Escherman. State assignment for hardwired vlsi control units. *ACM Computing Surveys*, 25(4):415–436, 1993.
42. S. Goren and F. Ferguson. CHESMIN: a heuristic for state reduction of incompletely specified finite state machines. In *Proc. of the Design, Automation and Test in Europe Conf. and Exhibition (DATE’02)*, pages 248–254, 2002.
43. B. Gupta, H. Narayanan, and M. Desai. A state assignment scheme targeting performance and area. In *Proc. of 12th Inter. Conf. On VLSI Design*, pages 378–383, 1999.
44. S. Habib. *Microprogramming and Firmware Engineering Methods*. John Wiley and Sons, New York, 1988.

45. S. Hassoun and T. Sasao. *Logic synthesis and verification*. Kluwer Academic Publishers, 2002.
46. G. Hatchel and F. Somenzi. *Logic synthesis and verification algorithms*. Kluwer Academic Publishers, 2000.
47. E. Hryniewicz and D. Kania. Impact of decomposition direction on synthesis effectiveness. In *Proc. of Programmable Devices and Systems(PDS'03)*, pages 144–149, 2003.
48. H. Hu, H. Xue, and J. Bian. A heuristic state assignment algorithm targeting area. In *Proc. of 5th Inter. Conf. on ASIC*, volume 1, pages 93–96, 2003.
49. J. Huang, J. Jou, and W. Shen. ALTO: An iterative area/performance algorithms for LUT-based FPGA technology mapping. *IEEE Transactions on VLSI Systems*, 18(4):392–400, 2000.
50. S. Husson. *Microprogramming: Principles and Practices*. Prentice Hall, Englewood Cliffs, NY, 1970.
51. A. Iranli, P. Rezvani, and M. Pedram. Low power synthesis of finite state machines with mixed D and T flip-flops. In *Proc. of the Asia and South Pacific– DAC*, pages 803–808, 2003.
52. H. Iwai. Future CMOS scaling. In *Proc. 11th Conf. Mixed Design of Integrated Circuits and Systems, MIXDES 2004*, pages 12–18, Szczecin, Poland, 2004. Department of Microelectronics and Computer Science, Technical University of Łódź.
53. J. Jenkins. *Design with FPGAs and CPLDs*. Prentice Hall, New York, 1995.
54. M. J.Flynn and R.F. Rosin. Microprogramming: An introduction and a viewpoint. *IEEE transactions on Computers*, C–20(7):727–731, 1971.
55. T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. *A Synthesis of Finite State Machines: Functional Optimization*. Kluwer Academic Publishers, Boston, 1998.
56. D. Kania. Two-level logic synthesis on PAL-based CPLD and FPGA using decomposition. In *Proc. of 25th Euromicro Conference*, pages 278–281, 1999.
57. D. Kania. Two-level logic synthesis on PALs. *Electronic Letters*, (17):879–880, 1999.
58. D. Kania. Coding capacity of PAL-based logic blocks included in CPLDs and FPGAs. In *Proc. of IFAC Workshop on Programmable Devices and Systems (PDS'2000)*, pages 164–169. Elsevier Science, 2000.
59. D. Kania. Decomposition-based synthesis and its application in PAL-oriented technology mapping. In *Proc. of 26-th Euromicro Conference*, pages 138–145. Maastricht: IEEE Computer Society Press, 2000.
60. D. Kania. An efficient algorithm for output coding in PAL-based CPLDs. *International Journal of Engineering*, 15(4):325–328, 2002.
61. D. Kania. Logic synthesis of multi-output functions for PAL-based CPLDs. In *Proc. of IEEE Inter. Conf. on Field-Programmable Technology*, pages 429–432, 2002.
62. D. Kania. An efficient approach to synthesis of multi-output boolean functions on PAL-based devices. In *IEEE Proc. – Computer and Digital Techniques*, volume 150, pages 143–149, 2003.
63. D. Kania. *Logic synthesis for PAL-oriented programmable structures*. Zeszyty naukowe Politechniki Śląskiej, Gliwice, 2004. (in Polish).
64. T. Łuba, K. Jasiński, and B. Zbierchowski. *Specialized digital circuits in PLD i FPGA structures*. Wydawnictwo Komunikacji i Łączności, 1997. (in Polish).
65. C. Maxfield. *The Design Warrior's Guide to FPGAs*. Academic Press, Inc., Orlando, FL, USA, 2004.
66. E. McCluskey. *Logic Design Principles*. Prentice Hall, Englewood Cliffs, 1986.
67. G. De Micheli. Symbolic design of combinational and sequential logic implemented by two-level macros. *IEEE Transactions on Computer-Aided Design*, 5(9):597–616, 1986.
68. G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw–Hill, 1994.
69. S. Park, S. Yang, and S. Cho. Optimal state assignment technique for partial scan designs. *Electronic Letters*, (18):1527–1529, 2000.
70. D. Patterson and J. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, San Moteo, CA, 1998.

71. C. Pedram and A. Despain. *Low-power state assignment targeting two- and multilevel logic implementations*, volume 17. 1998.
72. V. Pedroni. *Circuit Design with VHDL*. MIT Press, Cambridge, 2004.
73. I. Pomerancz and K. Cheng. STOIC: state assignment based on output/input functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and System*, 12(8):1123–1131, 1993.
74. E. Pugh, L. Johnson, and J. Palmer. *IBM's 360 and Early 370 Systems*. MIT Press, Cambridge, MA, 1991.
75. M. Rawski, T. Luba, Z. Jachna, and P. Tomaszewicz. *Design of Embedded Control Systems*, chapter The influence of functional decomposition on modern digital design process, pages 193–203. Springer, Boston, 2005.
76. M. Rawski, H. Selvaraj, and T. Luba. An application of functional decomposition in ROM-based FSM implementation in FPGA devices. *Journal of System Architecture*, 51(6-7):423–434, 2005.
77. J. Rho, F. Hatchel, R. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Transactions on Computer-Aided Design*, 13(2):167–177, 1994.
78. R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for pla optimization. *IEEE Transactions on Computer-Aided Design*, 6(5):727–750, 1987.
79. K. Sakamura. Future SoC possibilities. *IEEE Micro.*, (5):7, 2002.
80. Z. Salcic. *VHDL and FPLDs in Digital Systems Design, Prototyping and Customization*. Kluwer Academic Publishers, 1998.
81. T. Sasao. Input variable assignment and output phase optimization of pla optimization. *IEEE Transactions on Computers*, 33(10):879–894, 1984.
82. T. Sasao. *Switching Theory for Logic Synthesis*. Kluwer Academic Publishers, 1999.
83. G. Saucier, M. Depaulet, and P. Sicard. Asyl: a rule-based system for controller synthesis. *IEEE Transactions on Computer-Aided Design*, 6(11):1088–1098, 1987.
84. G. Saucier, P. Sicard, and L. Bouchet. Multi-level synthesis on programmable devices in the ASYL system. In *Proceedings of Euro ASIC*, pages 136–141, 1990.
85. C. Scholl. *Functional Decomposition with Application to FPGA Synthesis*. Kluwer Academic Publishers, Boston, 2001.
86. S. Schwartz. An algorithm for minimizing read-only memories for machine control. *IEEE 10th Annual Symposium on Switching and Automata Theory*, pages 28–33, 1968.
87. E. Sentowich, K. Singh, L. Lavango, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Bryton, and A. Sangiovanni-Vincentelli. SIS: a system for sequential circuit synthesis. Technical report, University of California, Berkely, 1992.
88. E. Sentowich, K. Singh, L. Lavango, C. Moon, R. Murgai, S. Saldanha, H. Savoj, P. Stephan, R. Bryton, and A. Sangiovanni-Vincentelli. SIS: a system for sequential circuit synthesis. In *Proc. of the Inter. Conf. of Computer Design (ICCD'92)*, pages 328–333, 1992.
89. B. Shriver and B. Smith. *The anatomy of a High-performance Microprocessor: A Systems Perspective*. IEEE Computer Society Press, Los Alamitos, CA, 1998.
90. V. A. Skljarov. *Synthesis of automata on matrix LSI*. Nauka i Technika, Minsk, 1984. (in Russian).
91. V. Solovjev and M. Czyzy. Refined CPLD macrocells architecture for effective FSM implementation. In *Proc. of the 25th EUROMICRO Conference*, volume 1, pages 102–109, Milan, Italy, 1999.
92. V. Solovjev and M. Czyzy. The universal algorithm for fitting targeted unit to complex programmable logic devices. In *Proc. of the 25th EUROMICRO Conference*, volume 1, pages 286–289, Milan, Italy, 1999.
93. V. Solovjev and M. Czyzy. Synthesis of sequential circuits on programmable logic devices based on new models of finite state machines. In *Proceedings of the EUROMICRO Conference, Milan*, pages 170 – 173, 2001.
94. V. V. Solovjev. *Design of Digital Systems Using the Programmable Logic Integrated Circuits*. Hot line – Telecom, Moscow, 2001. (in Russian).

95. S. Tucker. Microprogram control for system/360. *IBM System Journal*, 6(4):222–241, 1967.
96. G. Venkatamaran, S. Reddy, and I. Pomerancz. GALLOP: genetic algorithm based low power fsm synthesis by simultaneous partitioning and state assignment. In *Proc. of 16th Inter. Conf. on VLSI Design*, pages 533–538, 2003.
97. T. Villa, T. Kam, R. Brayton, and A. Sangiovanni-Vincentelli. *A Synthesis of Finite State Machines: Logic Optimization*. Kluwer Academic Publishers, Boston, 1998.
98. T. Villa, T. Saldachna, R. Brayton, and A. Sangiovanni-Vincentelli. Symbolic two-level minimization. *IEEE Transactions on Computer-Aided Desig*, 16(7):692–708, 1997.
99. T. Villa and A. Sangiovanni-Vincentelli. NOVA: State assignment of finite state machines for optimal two-level logic implementation. *IEEE Transactions on Computer-Aided Design*, 9(9):905–924, 1990.
100. M. Wilkes. The best way to design an automatic calculating machine. In *Proc. of Manchester University Computer Inaugural Conference*, 1951.
101. M. Wilkes and J. Stringer. Microprogramming and the design of the control circuits in an electronic digital computer. In *Proc. of Cambridge Philosophical Society*, volume 49, pages 230–238, 1953.
102. Y. Xia and A. Almani. Genetic algorithm based state assignment for power and area optimization. volume 149, pages 128–133, 2002.

Chapter 3

Synthesis of basic circuits of compositional microprogram control units

Abstract The chapter is devoted to the design and optimization of some basic CMCU structures. Corresponding methods are discussed for the CMCU basic structure as well as for the CMCU with common memory. The problem of the set of operator vertices of the initial GSA is solved first. The resulted partition includes minimum possible number of OLCs. Next, the method of natural microinstruction addressing is discussed and its solution in case of CMCU given. It is shown that optimization methods used in the case of Moore FSM can be used to optimize the CMCU hardware amount. All methods presented in this book are oriented towards decreasing of hardware amount in the CMCU logic circuits.

3.1 Synthesis of compositional microprogram control unit with basic structure

As was pointed out in Chapter 1, the compositional microprogram control unit with basic structure (Fig. 1.20) includes combinational circuit CC, which generates input memory functions for register $RG(\Psi)$ and counter $CT(\Phi)$, control memory CM, which keeps microoperations $y_n \in Y$ together with additional variables y_0 (timing control of RG and CT) and y_E (control of fetch flip-flop TF). This device implements the following systems of Boolean functions:

$$\Phi = \Phi(\tau, X), \quad (3.1)$$

$$\Psi = \Psi(\tau, X), \quad (3.2)$$

$$Y = Y(T), \quad (3.3)$$

$$y_0 = y_0(T), \quad (3.4)$$

$$y_E = y_E(T). \quad (3.5)$$

Here $\tau = \{\tau_1, \dots, \tau_{R_1}\}$ is a set of RG outputs, used for encoding M_1 states of addressing FSM S_1 ; $T = \{T_1, \dots, T_{R_2}\}$ is a set of CT outputs, used for addressing

microinstructions kept in the control memory. Total number of microinstructions equals $M_2 = |B_1|$ and R_1, R_2 can be calculated using the expressions:

$$R_1 = \lceil \log_2 M_1 \rceil, \quad (3.6)$$

$$R_2 = \lceil \log_2 M_2 \rceil. \quad (3.7)$$

It follows from Section 1.4 that the following problems should be solved during the CMCU U_1 logic circuit design:

1. Construction of partition $C = \{\alpha_1, \dots, \alpha_G\}$ of the set of operator vertices of initial GSA Γ with minimal possible G . This problem is represented by expression (1.25).
2. Natural addressing of microinstructions, corresponding to operator vertices of the initial GSA Γ . This problem is represented by expression (1.28).
3. Construction of control memory content.
4. Construction of transformed GSA $\Gamma(U_1)$.
5. Synthesis of logic circuits of S_1 and S_2 units with given logical elements.

Let us discuss solutions of these problems [3] and illustrate them using example of CMCU $U_1(I_2)$ design, with initial GSA I_2 shown in Fig. 3.1.

Construction of the partition C satisfying condition (1.25) is proposed in [6]. It includes the following steps:

1. Construct the set of main inputs $M(\Gamma)$ of GSA Γ .
2. Put $g = 1$.
3. Take an arbitrary vertex b_q from the set $M(\Gamma)$ and exclude this vertex from the set $M(\Gamma)$. Let us call the vertex b_q a base vertex of OLC α_g .
4. Find the vertex $b_t = pr_2\langle b_q, b_t \rangle$, where $\langle b_q, b_t \rangle \in E$. Let us call this step moving through GSA down.
5. If $b_t \in B_2$, or $b_t = b_E$, or $b_t \in D^i$ ($i \neq g$), stop construction of OLC α_g ($O_g = b_t$). Go to point 7.
6. Include the vertex b_t in OLC α_g after the vertex b_q . Go to point 4, using the vertex b_t as the base vertex of OLC α_g for moving through GSA down.
7. If $M(\Gamma) = \emptyset$, than go to point 8, else make $g := g + 1$ and go to point 3.
8. End.

Let us call this construction procedure as procedure P_1 . Application of P_1 to GSA I_2 gives the set $C = \{\alpha_1, \dots, \alpha_4\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1^1 = b_1$, $O_1 = b_2$; $\alpha_2 = \langle b_4, b_5, b_6, b_7 \rangle$, $I_2^1 = b_4$, $I_2^2 = b_6$, $I_2^3 = O_2 = b_7$; $\alpha_3 = \langle b_{11}, \dots, b_{14} \rangle$, $I_3^1 = b_{11}$, $O_3 = b_{14}$; $\alpha_4 = \langle b_9 \rangle$, $I_4^1 = O_4 = b_9$. In consequence, the graph-scheme of algorithm I_2 has the following characteristics: $G = 4$, $I(I_2) = \{b_1, b_4, b_6, b_7, b_{11}, b_9\}$, $O(I_2) = \{b_2, b_7, b_9, b_{14}\}$. Let us point out that initial GSA Γ should be transformed to provide the stop mode of CMCU. This transformation is executed in the same way as in case of the microprogram control unit with compulsory addressing of microinstructions. In case of GSA I_2 , it reduces to the insertion of variable y_E into vertex b_9 (Fig. 3.1).

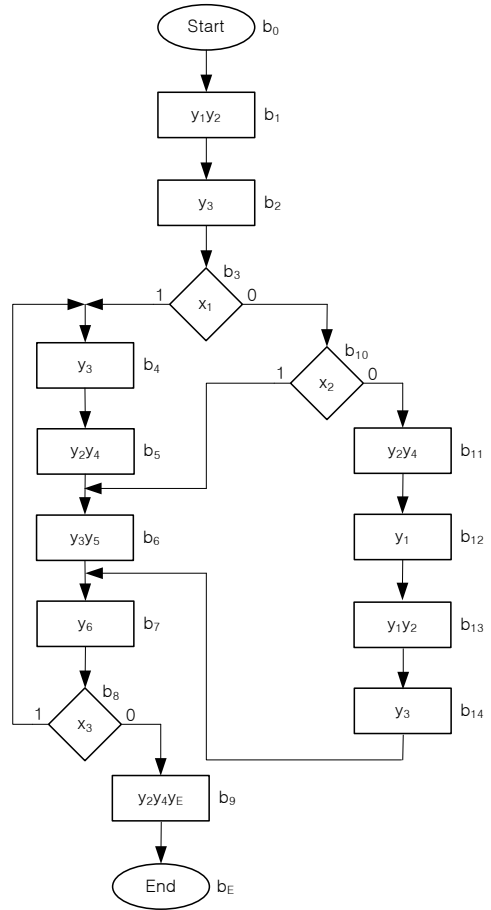


Fig. 3.1 Initial graph-scheme of algorithm I_2

Microinstruction addressing described by (1.28) can be performed by means of the procedure used in case of the MCU with natural addressing of microinstructions, described in [6]. It includes the following steps:

1. Construction of a vector α which is the concatenation of OLCs $\alpha_q \in C$, where the first component is the vertex $b_q \in B_1$, such that $\langle b_0, b_q \rangle \in E$. It should be pointed out that some additional vertex connected with output of initial vertex should be inserted into GSA Γ in case, when output of the initial vertex is connected with input of a conditional vertex.
2. Numeration of components of the vector α using the consecutive integers $0, \dots, M_2 - 1$.
3. Replacement of the number i of the component b_q of vector α by its binary equivalent $A(b_q)$ interpreted as an address of corresponding microinstruction $Y(b_q)$.

Let us call this addressing method as procedure P_2 . In case of the CMCU $U_1(I_2)$ we have $R_2 = 4$, $T = \{T_1, \dots, T_4\}$ and the result of procedure P_2 is shown in Fig. 3.2.

The microinstruction format of CMCU includes the operational part only. The control memory content of CMCU $U_1(I_2)$ is shown in Table 3.1, where variable y_0 is included in all microinstructions $Y(b_q)$, such that $b_q \notin O(\Gamma)$. The column "Comment" contains information about particular vertex corresponding to given address, belonging to the sets $I(\Gamma)$ and $O(\Gamma)$, and to the particular OLC including this vertex.

T_3T_4	00	01	11	10
00	A(b_1)	A(b_2)	A(b_5)	A(b_4)
01	A(b_6)	A(b_7)	A(b_{12})	A(b_{11})
11	*	*	*	*
10	A(b_{13})	A(b_{14})	*	A(b_9)

Fig. 3.2 Microinstruction addresses of CMCU $U_1(I_2)$

Table 3.1 Control memory content of CMCU $U_1(I_2)$

Address	Content	Comment
0000	y_0, y_1, y_2	$b_1 I_1^1 \alpha_1$
0001	y_3	$b_2 O_1$
0010	$y_0 y_3$	$b_4 I_2^1 \alpha_2$
0011	$y_0 y_2 y_4$	b_5
0100	$y_0 y_3 y_5$	$b_6 I_2^2$
0101	y_6	$b_7 I_2^3 O_2$
0110	y_2, y_4, y_E	$b_9 I_3^1 O_3 \alpha_3$
0111	$y_0 y_2 y_4$	$b_{11} I_4^1 \alpha_4$
1000	$y_0 y_1$	b_{12}
1001	$y_0 y_1 y_2$	b_{13}
1010	y_3	$b_{14} O_4$

Let us analyze main operation principles of CMCU U_1 and formulate basic requirements for the transformed GSA $\Gamma(U_1)$.

1. The finite state machine S_1 generates input memory functions serving to load into CT the addresses of OLC inputs only. All other addresses are generated by unit S_2 according to (1.28). Thus, the transformed GSA $\Gamma(U_1)$ should include only the operator vertices corresponding to OLC inputs.
2. If GSA Γ includes a path from vertex $b_q \in O(\Gamma)$ to vertex $b_l \in I(\Gamma)$, passing through some conditional vertices, the transformed GSA $\Gamma(U_1)$ should include the same path. Thus, the sets of conditional vertices are identical for GSA Γ and $\Gamma(U_1)$.

3. If counter CT contains an address $A(I_g^j)$ of the input j of OLC $\alpha_g \in C$ ($j = 1, \dots, J_g$), the state of FSM S_1 remains unchanged till the output address $A(O_g)$ of this OLC is reached. It means that one state may correspond to all inputs of any OLC. Thus, outputs of vertices corresponding to inputs of OLC $\alpha_g \in C$ can be combined together in the transformed GSA $\Gamma(U_1)$.

According to these requirements, we can propose the following procedure P_3 for construction of the transformed GSA $\Gamma(U_1)$ [5]:

1. Construct transition formulae [2] for vertices $b_q \in O(\Gamma)$ and b_0 of the initial GSA Γ .
2. Replace vertices $b_i \in I(\Gamma)$ from the left part of each transition formula by the inputs, corresponding to these vertices. Replace vertex $b_q = O_g$ from the right part of each transition formula by all inputs of OLC $\alpha_g \in C$.
3. Construct the transformed GSA using the system of transformed transition formulae and methods given in [2].
4. If operator vertex b_i of the transformed GSA corresponds to input I_g^j of OLC $\alpha_g \in C$, then write the input memory functions, which are equal to 1, into this vertex b_i to the address $A(I_g^j)$.

In case of CMCU $U_1(I_2)$, the initial system of transition formulae (STF) has the form:

$$\begin{aligned} b_0 &\rightarrow b_1; b_2 \rightarrow x_1 b_4 \vee \bar{x}_1 x_2 b_6 \vee \bar{x}_1 x_2 b_{11}; \\ b_7 &\rightarrow x_3 b_4 \vee \bar{x}_3 b_9; b_{14} \rightarrow b_7; b_9 \rightarrow b_E. \end{aligned}$$

Execution of point 2 of procedure P_3 leads to STF of the transformed GSA $\Gamma_2(U_1)$:

$$\begin{aligned} b_0 &\rightarrow I_1^1; I_1^1 \rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_2^2 \vee \bar{x}_1 \bar{x}_2 I_4^1; \\ I_2^1, I_2^2, I_2^3 &\rightarrow x_3 I_2^1 \vee \bar{x}_3 I_3^1; I_3^1 \rightarrow b_E; \\ I_4^1 &\rightarrow I_2^3. \end{aligned}$$

In case of CMCU $U_1(I_2)$, the microinstruction address includes $R_2 = 4$ bits and therefore the set of input memory functions includes also 4 elements ($\Phi = \{D_1, D_2, D_3, D_4\}$). The transformed GSA $\Gamma_2(U_1)$ is shown in Fig. 3.3.

Symbols of OLC inputs of the initial GSA are shown near corresponding operator vertices $b_i \in B_1$ of the transformed GSA, which makes finding the content of these vertices much easier. For example, in case of vertex I_2^1 , we find the address $A(I_2^1) = 0010$ from Table 3.1 and place the function $D_3 \in \Phi$ into the vertex I_2^1 . Contents of all other vertices are formed in the same way.

Synthesis of FSM S_1 logic circuit is executed using the well-known methods given in [2]:

1. Construction of marked GSA $\Gamma(U_1)$. For example under discussion, the states are shown in Fig. 3.3.

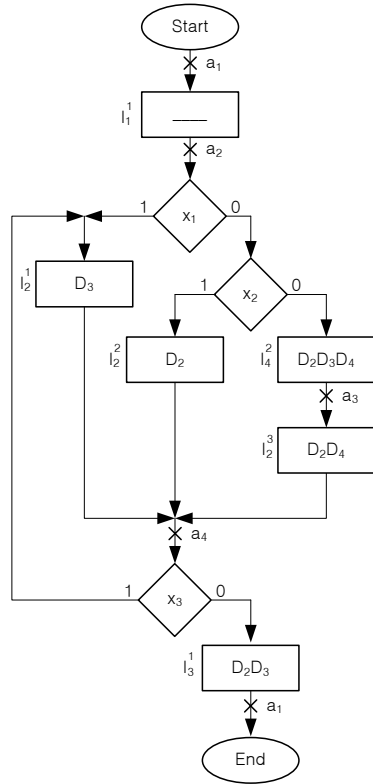


Fig. 3.3 Transformed GSA $F_2(U_1)$

2. State encoding for FSM S_1 . In our case there are 4 internal states and $A = \{a_1, \dots, a_4\}$, $M_1 = 4$, $R_1 = 2$, $\tau = \{\tau_1, \tau_2\}$. Let these states be encoded as: $K(a_1) = 00, \dots, K(a_4) = 11$. The set of input memory functions for register RG is: $\Psi = \{D_5, D_6\}$.
3. Construction of structure table of FSM S_1 . This table contains $H = 7$ lines (Table 3.2).

Table 3.2 Structure table of FSM S_1 of CMCU $U_1(F_2)$

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Φ_h	Ψ_h	h
a_1	00	a_2	01	1	–	D_6	1
a_2	01	a_4	11	x_1	D_3	D_5D_6	2
		a_4	11	\bar{x}_1x_2	D_2	D_5D_6	3
		a_3	10	$\bar{x}_1\bar{x}_2$	$D_2D_3D_4$	D_4D_5	4
a_3	10	a_4	11	1	D_2D_4	D_5D_6	5
a_4	11	a_4	11	x_3	D_3	D_5D_6	6
		a_1	00	\bar{x}_3	D_2D_3	–	7

4. Construction of systems Φ and Ψ . The systems of Boolean functions (3.1) and (3.2) depend on conjunctive terms represented in the form (1.6), where conjunction A_m represented as (1.7) includes variables $\tau_r \in \tau$. From Table 3.2 we can get, for example, the following Boolean expression:

$$D_2 = F_3 \vee F_4 \vee F_5 \vee F_7 = \bar{\tau}_1 \tau_2 \bar{x}_1 \vee \tau_1 \bar{\tau}_2 \vee \tau_1 \tau_2 \bar{x}_3.$$

Now the implementation of CMCU U_1 logic circuit is reduced to implementation of systems (3.1) – (3.2) using PLDs and control memory, corresponding to systems (3.3) – (3.5), using PROM or RAM chips.

A special unit is needed to provide the synchronization mode of CMCU U_1 . Assume that the counter CT has two inputs for synchronization, with: $C_1\#CT := CT + 1$ and $C_2\#CT := \langle \Phi \rangle$ (loading of a parallel code). Then, pulse Clock should appear at the input C_1 , if $y_0 = 1$, and at the input C_2 , if $y_0 = 0$. Therefore, synchronization unit is described by the following expressions:

$$\begin{aligned} C_1 &= y_0 \cdot \text{Clock}; \\ C_2 &= \bar{y}_0 \cdot \text{Clock}. \end{aligned} \tag{3.8}$$

The pulse C_2 should appear also at the timing input of register RG.

Let both systems (3.1) and (3.2) be implemented with PLA chips. In this case the CMCU $U_1(I_2)$ logic circuit has the form shown in Fig. 3.4.

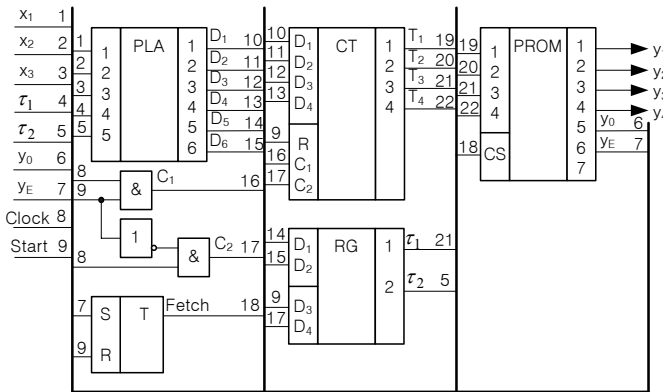


Fig. 3.4 Logic circuit of CMCU $U_1(I_2)$

The parameters of PLA and PROM chips are chosen in such a manner that only one chip can be used to implement a particular part of logic circuit. It is made to avoid complex optimization procedures.

The main advantage of CMCU U_1 is minimal possible number of feedback variables (state variables). Using a special RG and increasing the number of combinational circuit outputs, with respect to the minimal number R_2 , is needed however to address the microinstructions. When condition (3.9) is satisfied

$$R_1 = R_2, \quad (3.9)$$

the CMCU U_1 can not be considered as the efficient interpretation of any particular control algorithm, because the number of CC outputs is twice R_2 . To interpret the control algorithms satisfying (3.9), a compositional microprogram control unit with common memory is proposed [3]. In this CMCU, counter CT is used as a source of both microinstruction address and the state code of FSM S_1 . This principle can be used to interpret an arbitrary GSA Γ , even in case, when condition (3.9) is not satisfied.

3.2 Synthesis of CMCU with common memory

The structural diagram of CMCU with common memory (called CMCU U_2) is shown in Fig. 3.5.

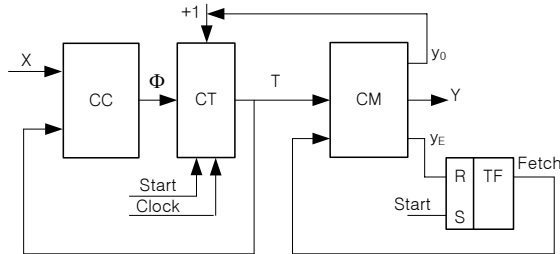


Fig. 3.5 Structural diagram of CMCU U_2

In this case, combinational circuit CC implements the system of input memory functions of the counter CT:

$$\Phi = (T, X), \quad (3.10)$$

remaining blocks execute the same functions as in case of the CMCU U_1 .

The address of first microinstruction of the particular microprogram is loaded into CT using pulse "Start". This pulse causes also the set up of flip-flop TF (it means that Fetch=1). Current microinstruction is fetched out of the control memory CM. If it includes the variable y_0 , content of the counter is incremented and unconditional jump executed between microinstructions corresponding to the components of the same OLC $\alpha_g \in C$. If $y_0 = 0$, functions (3.10) will change content of the counter using pulse "Clock". It corresponds to the transition from output of OLC $\alpha_g \in C$. If $y_E = 1$, the microprogram is ended. In this case flip-flop TF is reset and the operation of CMCU terminated.

As can be seen from comparison of (3.1) and (3.10), multidirectional transitions are executed during one cycle of both CMCU U_1 and U_2 . Output functions of the

control memory are represented by (3.3) – (3.5) and therefore parameters of CM for CMCU U_1 and U_2 are the same.

The synthesis method of CMCU $U_2(\Gamma)$ includes the following steps:

1. Preliminary transformation of the initial GSA Γ .
2. Construction of OLC C set for the transformed GSA $\Gamma(U_2)$.
3. Natural addressing of microinstructions.
4. Construction of control memory content.
5. Construction of transition table for CMCU U_2 .
6. Synthesis of logic circuit with given logical elements.

Let us discuss an example of CMCU U_2 design using the graph-scheme of algorithm Γ_3 (Fig. 3.6), where only the operator vertices are numbered.

Preliminary transformation of GSA Γ CA Γ is executed in the following manner (procedure P_4):

- if there is an arc $\langle b_0, b_q \rangle \in E$, where $b_q \in B_2$, vertex $b_t \in B_1$ is introduced into GSA Γ , where $Y(b_t) = \emptyset$, and initial arc $\langle b_0, b_q \rangle \in E$ is replaced by a pair of new arcs $\langle b_0, b_t \rangle$ and $\langle b_t, b_q \rangle$;
- if there is an arc $\langle b_t, b_E \rangle \in E$, where $b_t \in B_2$, the vertex $b_q \in B_1$ with y_E is introduced into GSA Γ and initial arc $\langle b_t, b_E \rangle \in E$ is replaced by two arcs $\langle b_t, b_q \rangle$ and $\langle b_q, b_E \rangle$;
- if there is an arc $\langle b_t, b_E \rangle \in E$, where $b_t \in B_1$, the variable y_E is inserted into operator vertex b_t .

The first transformation is essential for organization of conditional transitions, after reset of the counter using pulse "Start". The second and third transformations are essential to organize the termination mode of CMCU. Let us point out that GSA can include more than one arc connecting conditional vertices with final vertex (Fig. 3.7a), but only single additional vertex with y_E is introduced into transformed GSA and it is common for all such conditional vertices (Fig. 3.7b).

Therefore, the transformed GSA $\Gamma(U_2)$ includes at most $|B_1| + 2$ operator vertices, where B_1 is a set of operator vertices of the initial GSA Γ .

In case of the GSA Γ_3 , this transformation is reduced to insertion of the vertex b_{18} , connected to the output of vertex b_0 , and the vertex b_{19} , connected to the input of vertex b_E (Fig. 3.8). Thus, the transformed GSA $\Gamma_3(U_2)$ includes $M_2 = 19$ operator vertices, corresponding to 19 microinstructions kept in the control memory CM. Therefore, $R_2 = 5$ variables are sufficient to form the set $T = \{T_1, \dots, T_5\}$, necessary to address all microinstructions of CMCU $U_2(\Gamma_3)$.

Construction of the set of OLC reduces to the application of procedure P_1 to the transformed GSA $\Gamma(U_2)$. In case under consideration we have $C = \{\alpha_1, \dots, \alpha_7\}$, with $\alpha_1 = \langle b_{18} \rangle, I_1^1 = O_1 = b_{18}; \alpha_2 = \langle b_1, b_2, b_3 \rangle, I_2^1 = b_1, O_2 = b_3; \alpha_3 = \langle b_4, b_5, b_6 \rangle, I_3^1 = b_4, O_3 = b_6; \alpha_4 = \{b_7, \dots, b_{10}\}, I_4^1 = b_7, I_4^2 = b_9, O_4 = b_{10}; \alpha_5 = \{b_{11}, \dots, b_{14}\}, I_5^1 = b_{11}, I_5^2 = b_{13}, O_5 = b_{14}; \alpha_6 = \langle b_{15}, b_{16}, b_{17} \rangle, I_6^1 = b_{15}, O_6 = b_{17}; \alpha_7 = \langle b_{19} \rangle, I_7^1 = O_7 = b_{19}$.

Natural addressing of microinstructions is reduced here to the application of procedure P_2 . For our example microinstruction addresses of the CMCU $U_2(\Gamma_3)$ are

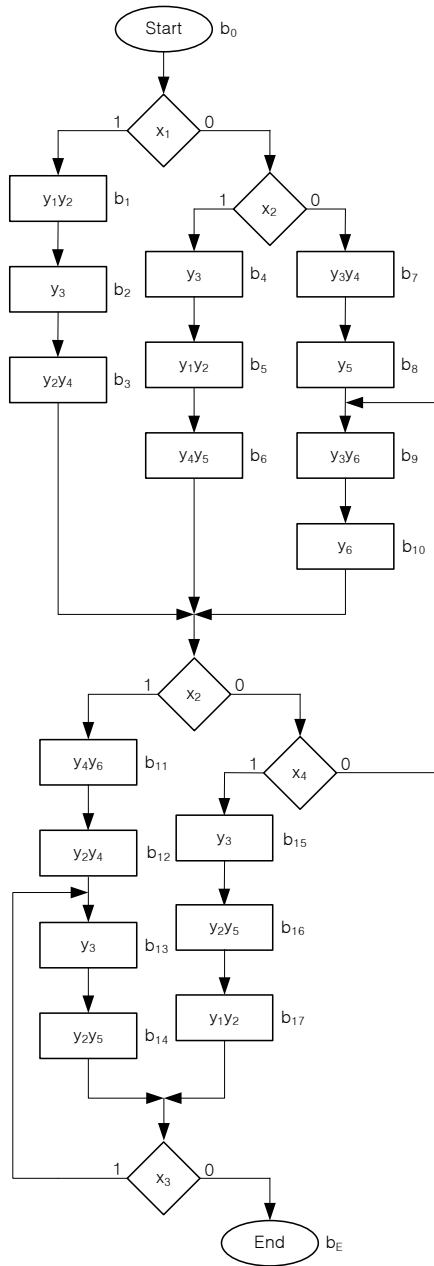


Fig. 3.6 Initial graph-scheme of algorithm I_3

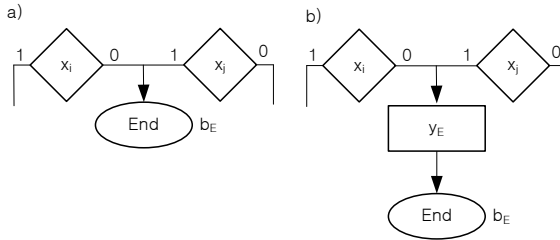


Fig. 3.7 Fragment of initial GSA before **a** and after **b** transformation

shown in Fig. 3.9. This figure represents the addresses using a kind of modified Karnaugh map with all possible input assignments. Symbol "*" is used to indicate the "don't care" assignments. The symbols of operator vertices are written in corresponding cells of the table.

Construction of control memory content for CMCU U_2 is executed in the same way as in case of CMCU U_1 . In our example the control memory contains $M_2 = 19$ microinstructions. The format of each microinstruction includes only the field FY. This table is not presented here because its construction is obvious.

Construction of CMCU transition table. The addressing finite state machine S_1 of CMCU U_2 is represented by the Moore FSM, with states corresponding to the operational linear chains $\alpha_g \in C$, and the state codes corresponding to addresses of OLC output microinstructions. Let $C^1 \subseteq C$ be a set of OLC, such that their outputs have no direct connections with the final vertex of transformed GSA $U_2(\Gamma)$. Procedure P_5 is then proposed to construct a transition table of CMCU U_2 in the following steps:

- construct the system of transition formulae for outputs of OLC $\alpha_g \in C^1$;
- replace the vertex in the left part of each transition formula by symbol of corresponding OLC output;
- replace vertices in the right part of each transition formula by symbols of corresponding OLC inputs;
- construct the CMCU transition table with the columns $O_g, A(O_g), I_m^i, A(I_m^j), X_h, \Phi_h, h$, where $A(O_g)$ is an address of OLC $\alpha_g \in C^1$ output; $A(I_m^j)$ is an address of OLC $\alpha_m \in C$ input; X_h is an input signal, which determines the transition from O_g into I_m^j ; Φ_h is a set of input memory functions, which are equal to 1 in order to change the content of counter CT from $A(O_g)$ to $A(I_m^j)$; $h = 1, \dots, H$ is a transition number.

Application of points 1 and 2 of procedure P_5 to the GSA $I_3(U_2)$ leads to the following STF:

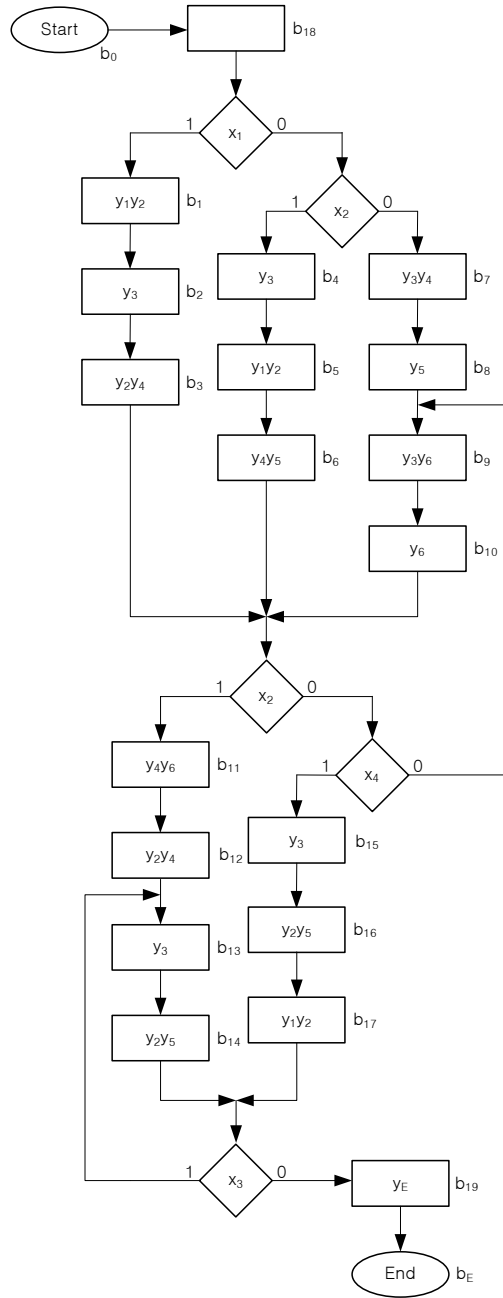


Fig. 3.8 Transformed GSA $\Gamma_3(U_2)$

$T_1T_2T_3$								
T_4T_5	000	001	010	011	100	101	110	111
00	b_{18}	b_4	b_8	b_{12}	b_{16}	*	*	*
01	b_1	b_5	b_9	b_{13}	b_{17}	*	*	*
11	b_2	b_6	b_{10}	b_{14}	b_{19}	*	*	*
10	b_3	b_7	b_{11}	b_{15}	*	*	*	*

Fig. 3.9 Microinstruction addresses of CMCU $U_2(I_3)$

$$\begin{aligned}
 O_1 &\rightarrow x_1I_2^1 \vee \bar{x}_1x_2I_3^1 \vee \bar{x}_1\bar{x}_2I_4^1; \\
 O_2 &\rightarrow x_2I_5^1 \vee \bar{x}_2x_4I_6^1 \vee \bar{x}_2\bar{x}_4I_4^2; \\
 O_3 &\rightarrow x_2I_5^1 \vee \bar{x}_2x_4I_6^1 \vee \bar{x}_2\bar{x}_4I_4^2; \\
 O_4 &\rightarrow x_2I_5^1 \vee \bar{x}_2x_4I_6^1 \vee \bar{x}_2\bar{x}_4I_4^2; \\
 O_5 &\rightarrow x_3I_5^2 \vee \bar{x}_3I_7^1; \\
 O_6 &\rightarrow x_3I_5^2 \vee \bar{x}_3I_7^1.
 \end{aligned} \tag{3.11}$$

Let us point out that the OLC $\alpha_7 \notin C^1$, and in consequence transitions from the output O_7 are not listed in (3.11). The transition table of CMCU $U_2(I_3)$ (Table 3.3) contains $H = 16$ lines. Each line corresponds to one term of the system (3.11). All addresses in the columns of Table 3.3 are taken from Fig. 3.9, the set of input memory functions includes 5 elements, and $\Phi = \{D_1, \dots, D_5\}$. The total number of lines H is equal to the number of terms in (3.11).

Synthesis of CMCU logic circuit is reduced to implementation of SBF (3.10) using PLDs and implementation of control memory CM using PROMs or RAMs. System (3.10) is constructed using transition table in the following form:

$$\varphi_r = \bigvee_{h=1}^H C_{rh} O_g^h X_h (r = 1, \dots, R_2), \tag{3.12}$$

where C_{rh} is a Boolean variable equal to 1 iff line h of the transition table contains variable φ_r ; O_g^h is a conjunction of state variables $T_r \in T$, corresponding to address $A(O_g)$ from line h of the table ($h = 1, \dots, H$).

Using Table 3.3, we can get, for example, the Boolean function:

$$D_1 = \bar{T}_1T_2T_3T_4\bar{T}_5\bar{x}_3 \vee T_1\bar{T}_2\bar{T}_3\bar{T}_4T_5\bar{x}_3.$$

Logic circuit of the compositional microprogram control unit $U_2(I_3)$ is shown in Fig. 3.10.

As in previous case and all cases presented below, parameters of both PLA and PROM are sorted out in such a manner, that each block of any control unit discussed in this book can be implemented using a single PLD chip. Let us introduce symbols of some parameters needed to compare different structures of compositional microprogram control units:

Table 3.3 Transition table of CMCU $U_2(\Gamma_3)$

O_g	$A(O_g)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
O_1	00000	I_2^1	00001	x_1	D_5	1
		I_3^1	00100	\bar{x}_1x_2	D_3	2
		I_4^1	00111	$\bar{x}_1\bar{x}_2$	$D_3D_4D_5$	3
O_2	00011	I_5^1	01011	x_2	$D_2D_4D_5$	4
		I_6^1	01111	\bar{x}_2x_4	$D_2D_3D_4D_5$	5
		I_4^2	01001	$\bar{x}_2\bar{x}_4$	D_2D_5	6
O_3	00110	I_5^1	01011	x_2	$D_2D_4D_5$	7
		I_6^1	01111	\bar{x}_2x_4	$D_2D_3D_4D_5$	8
		I_4^2	01001	$\bar{x}_2\bar{x}_4$	D_2D_5	9
O_4	01010	I_5^1	01011	x_2	$D_2D_4D_5$	10
		I_6^1	01111	\bar{x}_2x_4	$D_2D_3D_4D_5$	11
		I_4^2	01001	$\bar{x}_2\bar{x}_4$	D_2D_5	12
O_5	01110	I_5^2	01101	x_3	$D_2D_5D_3$	13
		I_7^1	10010	\bar{x}_3	D_1D_4	14
O_6	10001	I_5^2	01101	x_3	$D_2D_5D_3$	15
		I_7^1	10010	\bar{x}_3	D_1D_4	16

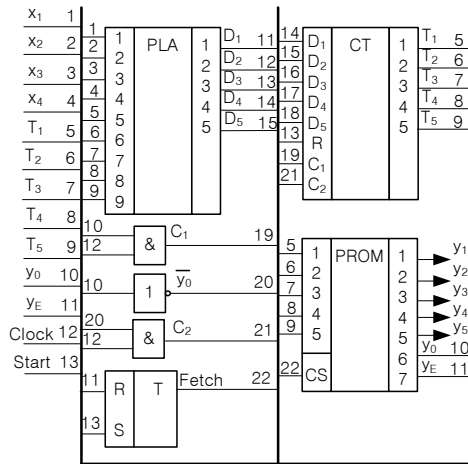


Fig. 3.10 Logic circuit of CMCU $U_2(\Gamma_3)$

- R_{FB}^i is the number of feedback inputs of the combinational circuit of CMCU U_i ($i = 1, 2, \dots$);
- $R_{FB}^i(\Gamma_j)$ is the corresponding symbol for R_{FB}^i in case of CMCU $U_i(\Gamma_j)$;
- $H_i(\Gamma_j)$ is the total number of terms in systems of Boolean functions implemented by combinational circuit CC of CMCU $U_i(\Gamma_j)$;
- $S_i(\Gamma_j)$ is the number of input variables of combinational circuit CC of CMCU $U_i(\Gamma_j)$;
- $t_i(\Gamma_j)$ is the number of output variables of combinational circuit CC of CMCU $U_i(\Gamma_j)$;

If the symbols listed above use Γ instead of Γ_j , the common value of corresponding parameter is taken.

Let us introduce, for the block of GSA, some graphical symbol (Fig. 3.11), corresponding to OLC $\alpha_g \in C$ with inputs $I_g^1, \dots, I_g^{K_g}$ and output O_g .

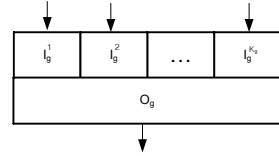


Fig. 3.11 Block of GSA Γ corresponding to OLC α_g

This block represents the part of GSA Γ , operator vertices of which are replaced by blocks. Let us add the formula $O_7 \rightarrow b_E$ to system (3.11) and construct the block GSA Γ_3 (Fig. 3.12). Analysis of Table 3.3 and Fig. 3.12 shows that FSM S_1 of CMCU U_2 is a Moore FSM, having the states corresponding to blocks of the interpreted graph-scheme of algorithm Γ . Specific property of this FSM is that the codes of its current states $a_m \in A$ are determined by output addresses of corresponding OLC, whereas codes of its next states are determined by input addresses of the operational linear chains $\alpha_g \in C$.

Addressing FSM $S_1(U_2)$ of CMCU U_2 has the following parameters:

$$R_{FB}^2 = R_2; \quad (3.13)$$

$$S_2(\Gamma) = R_2 + L; \quad (3.14)$$

$$t_2(\Gamma) = R_2; \quad (3.15)$$

$$H_2(\Gamma) = \sum_{g=1}^G C^g H_g, \quad (3.16)$$

where C^g is a Boolean variable, which is equal to 1 iff $\alpha_g \in C^1$; H_g is the number of transitions from output O_g ($g = 1, \dots, G$).

In order to compare the characteristics of FSM $S_1(U_1)$ and $S_1(U_2)$, let us construct the transformed GSA $\Gamma_3(U_1)$ and mark it with states of Mealy FSM (Fig. 3.13).

In our case, $R_1 = 2$, and comparison of both GSAs shows that a single state of FSM $S_1(U_1)$ may correspond to more than one block of GSA $\Gamma(U_2)$. Let us call OLC, $\alpha_i, \alpha_j \in C$, a pseudoequivalent OLC, if their outputs are connected with the input of the same GSA vertex of GSA Γ . Obviously, the existence of pseudoequivalent OLC can be used to optimize combinational part of FSM $S_1(U_2)$. Finite state machine $S_1(U_1)$ has the following parameters:

$$R_{FB}^1 = R_1; \quad (3.17)$$

$$S_1(\Gamma) = R_1 + L; \quad (3.18)$$

$$t_1(\Gamma) = R_1 + R_2; \quad (3.19)$$

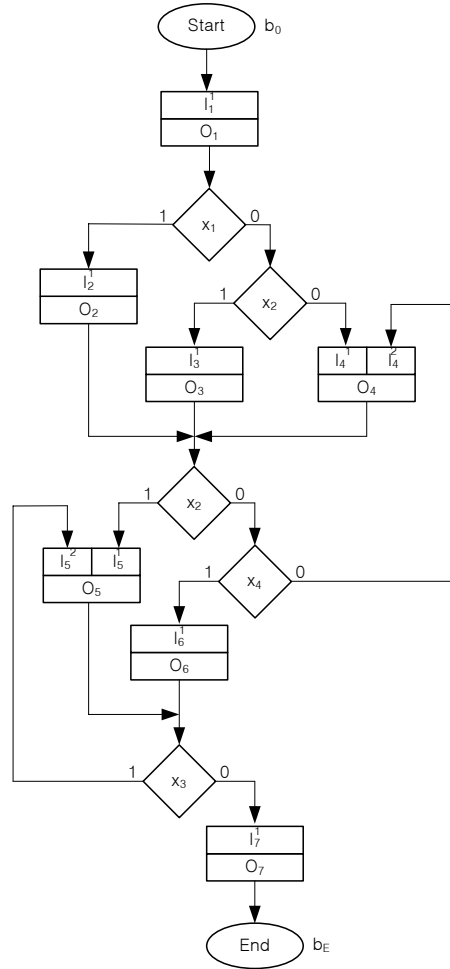


Fig. 3.12 Block representation of GSA Γ_3

$$H_1(\Gamma) = \sum_{m=1}^M H_m, \tag{3.20}$$

where H_m is the number of transitions from the state $a_m \in A$. Comparison of expressions (3.13) – (3.20) shows that the following relations are true for CMCU $U_1(\Gamma)$ and $U_2(\Gamma)$:

$$S_1(\Gamma) \leq S_2(\Gamma); \tag{3.21}$$

$$t_1(\Gamma) > t_2(\Gamma); \tag{3.22}$$

$$H_1(\Gamma) \leq H_2(\Gamma). \tag{3.23}$$

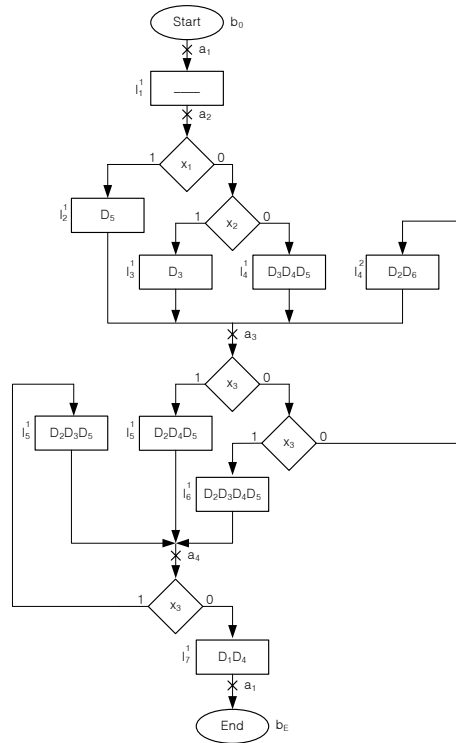


Fig. 3.13 Transformed GSA $F_3(U_1)$

Thus, the values of $S_2(\Gamma)$ and $H_2(\Gamma)$ should be reduced to allow practical use of the compositional microprogram control unit U_2 . All methods proposed further are devoted to solve this problem.

3.3 Optimization of CMCU with common memory logic circuit

Methods of Moore FSM logic circuit optimization [4] can be applied to decrease the hardware amount of the CMCU U_2 . These methods should be adapted to take account of the fact that the main input address determines here unambiguously the output address of corresponding operational linear chain and vice versa. This property results in the necessity of operation by blocks of vertices, rather than by separate vertices. The following optimization methods considering this property are discussed in our book:

- special addressing of microinstructions;
- optimal addressing of microinstructions;
- transformation of microinstruction addresses;
- transformation of OLC output codes;

- transformation of transformed GSA.

Let us discuss these methods in details.

Special addressing of microinstructions [1] is based on identification of the operational linear chains $\alpha_g \in C$ using

$$R_3 < R_2 \tag{3.24}$$

bits of microinstruction address. For the best case, this parameter can be found from the following equation:

$$R_3 = \lceil \log_2 G_1 \rceil, \tag{3.25}$$

where $G_1 = |C^1|$ is the number of OLCs in a set $C^1 \subseteq C$. This method is based on finding a set $T' \subseteq T$, where $T = \{T_1, \dots, T_{R_2}\}$ is the set of address bits and $|T'| = R_3$. Elements of the set T are used to address microinstructions and elements of set T' are used for identification of the outputs of OLC $\alpha_g \in C$. This approach leads to the compositional microprogram control unit U_3 (Fig. 3.14).

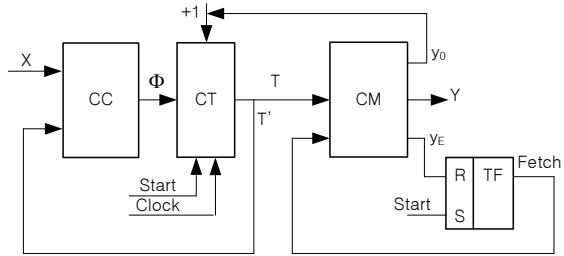


Fig. 3.14 Structural diagram of CMCU U_3

Operation mode of CMCU U_3 is the same as in case of CMCU U_2 . Obviously, synthesis methods for both control units are also identical; the only exception is the addressing procedure in use.

Let us analyze the outcome of microinstruction addressing for the CMCU $U_2(I_3)$ shown in Fig. 3.9. In this case the OLC $\alpha_7 \notin C^1$, $|C^1| = 6$ and $R_3 = 3$. Therefore, the outputs of OLC $\alpha_g \in C^1$ can be identified using $R_3 = 3$ variables and the set $T' = \{T_1, T_2, T_3\} \subset T$. Let us use the elements of $T' \subseteq T$ to represent the output codes $K(O_g)$ for OLC $\alpha_g \in C^1$.

In our example, the outputs of OLC $\alpha_g \in C^1$ correspond to codes $K(O_1) = K(O_2) = 000$, $K(O_3) = 001$, $K(O_4) = 010$, $K(O_5) = 011$, $K(O_6) = 100$. Thus, application of procedure P_2 does not allow unique identification of the OLC α_1 and α_2 . Analysis of Fig. 3.9 shows that it is the consequence of the fact that the outputs of both OLCs belong to the same generalized interval $000**$ of the Boolean space. Obviously, microinstruction addressing for CMCU U_3 should be executed in such a manner that the output of any OLC $\alpha_g \in C$ belongs to a unique R_3 - dimensional generalized interval of an R_2 - dimensional Boolean space. The following procedure P_6 is proposed for this microinstruction addressing:

1. Put $R_0 = R_3$ and apply procedure P_2 .

2. Construct an addressing table with 2^{R_0} columns, marked by variables T_1, \dots, T_{R_0} , and with $2^{R_2-R_0}$ rows, marked by variables T_i, \dots, T_{R_2} , where $i = R_0 + 1$.
3. If outputs of OLC $\alpha_i, \alpha_j \in C^1$, where $j > i$, belong to the same column of the table, make shift of information to the right starting from main input of OLC $\alpha_j \in C^1$. Fill up empty cells of the table by the sign "*". Continue the shift till outputs of OLC $\alpha_i, \alpha_j \in C^1$ are in different columns of addressing table.
4. If the shift resulted in spillover from the address space, find $R_0 := R_0 + 1$.
5. If $R_0 < R_2$, go to point 2.
6. End.

Application of procedure P_2 to the GSA Γ_3 results in microinstruction addressing (Fig. 3.9) with $R_0 = R_3 = 3$. Analysis of this addressing table shows that vertices $b_{18} = O_1$ and $b_3 = O_2$ are placed in the same column. First shift to the right gives the addressing table of Fig. 3.15a, where outputs O_1 and O_2 are placed in different columns. Now, outputs $O_2 = b_3$ and $O_3 = b_6$ are in the same column. Second shift to the right starting from vertex b_4 results in the addressing table of Fig. 3.15b, where outputs $O_5 = b_{14}$ and $O_6 = b_{17}$ are in the same column. Third shift to the right starting from the vertex b_{15} gives the final table of Fig. 3.15c. Now, the outputs of OLC $\alpha_j \in C^1$ are located in different columns of addressing table and each output has a unique code, namely: $K(O_1) = 000, K(O_2) = 001, K(O_3) = 010, K(O_4) = 011, K(O_5) = 100, K(O_6) = 101$. Therefore, OLC outputs are univocally identified using minimal possible number of variables R_3 .

In order to implement combinational circuit CC of CMCU U_3 we should construct a system of transition formulae and a corresponding transition table. In our example, the system of transition formulae is represented by (3.11). In the transition table of CMCU U_3 column $A(O_g)$ is replaced by column $K(O_g)$.

The final transition table allows finding the system of input memory functions:

$$\varphi_r = \bigvee_{h=1}^{H_3(\Gamma)} C_{rh} O_g^h X_h \quad (r = 1, \dots, R_0), \quad (3.26)$$

where O_g^h is a conjunction of variables $T_r \in T'$, corresponding to the output code of OLC $\alpha_j \in C^1$ from the line h of transition table ($h = 1, \dots, H_3(\Gamma)$). System (3.25) can be represented in the form:

$$\Phi = \Phi(T', X). \quad (3.27)$$

In our example, the transition table (Table 3.4) includes $H_3(\Gamma_3) = 16$ lines. The addresses of OLC inputs and outputs are taken here from the addressing table shown in Fig. 3.15c.

The parameters of combinational circuit CC of compositional microprogram control unit U_3 can be identified as:

$$S_3(\Gamma) = R_0 + L(R_0 \leq R_2); \quad (3.28)$$

$$t_3(\Gamma) = t_2(\Gamma); \quad (3.29)$$

$$H_3(\Gamma) = H_2(\Gamma). \quad (3.30)$$

a) $T_1T_2T_3$

T_4T_5 \ $T_1T_2T_3$	000	001	010	011	100	101	110	111
00	18	3	7	11	15	*	*	*
01	*	4	8	12	16	*	*	*
11	1	5	9	13	17	*	*	*
10	2	6	10	14	18	*	*	*

b) $T_1T_2T_3$

T_4T_5 \ $T_1T_2T_3$	000	001	010	011	100	101	110	111
00	18	3	6	10	14	19	*	*
01	*	*	7	11	15	*	*	*
11	1	4	8	12	16	*	*	*
10	2	5	9	13	17	*	*	*

c) $T_1T_2T_3$

T_4T_5 \ $T_1T_2T_3$	000	001	010	011	100	101	110	111
00	18	3	6	10	14	17	*	*
01	*	*	7	11	*	19	*	*
11	1	4	8	12	15	*	*	*
10	2	5	9	13	16	*	*	*

Fig. 3.15 Addressing of microinstructions of CMCU $U_3(T_3)$

Thus, application of special microinstruction addressing allows higher number of combinational circuit CC outputs, in comparison with the CMCU U_2 . It is clear that the following condition is satisfied:

$$S_1(\Gamma) \leq S_3(\Gamma) \leq S_2(\Gamma). \quad (3.31)$$

As follows from (3.30), special microinstruction addressing does not affect the number of lines in the transition table.

Optimal microinstruction addressing is oriented towards decrease of the number of inputs and terms of combinational circuit CC [1]. Let us address microinstructions in such a way that output addresses of pseudoequivalent OLC $\alpha_j \in C^1$ belong to the same generalized interval of R_2 -dimensional Boolean space. Now, the interval including pseudoequivalent OLC $\alpha_g, \alpha_j \in B_i$ is considered as the code $K(B_i)$ of a class $B_i \in \Pi_c$, where $\Pi_c = \{B_i, \dots, B_l\}$ is a set partition of the set $C' \subset C$ by classes of pseudoequivalent OLCs. Last approach leads to CMCU U_4 , having the same structure as the compositional microprogram control unit U_2 (Fig. 3.14).

The CMCU $U_4(\Gamma)$ synthesis includes the following steps:

Table 3.4 Transition table of CMCU $U_3(I_3)$

O_g	$K(O_g)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
O_1	000	I_2^1	00010	x_1	D_4	1
		I_3^1	00110	\bar{x}_1x_2	D_3D_4	2
		I_4^1	01001	$\bar{x}_1\bar{x}_2$	D_2D_5	3
O_2	001	I_5^1	01101	x_2	$D_2D_3D_5$	4
		I_6^1	10010	\bar{x}_2x_4	D_1D_4	5
		I_4^2	01011	$\bar{x}_2\bar{x}_4$	$D_2D_4D_5$	6
O_3	010	I_5^1	01101	x_2	$D_2D_3D_5$	7
		I_6^1	10010	\bar{x}_2x_4	D_1D_4	8
		I_4^2	01011	$\bar{x}_2\bar{x}_4$	$D_2D_4D_5$	9
O_4	011	I_5^1	01101	x_2	$D_2D_3D_5$	10
		I_6^1	10010	\bar{x}_2x_4	D_1D_4	11
		I_4^2	01011	$\bar{x}_2\bar{x}_4$	$D_2D_4D_5$	12
O_5	100	I_5^2	01111	x_3	$D_2D_3D_4D_5$	13
		I_7^1	10010	\bar{x}_3	D_1D_4	14
O_6	101	I_5^2	01111	x_3	$D_2D_3D_4D_5$	15
		I_7^1	10010	\bar{x}_3	D_1D_4	16

1. Construction of OLC set for GSA Γ (procedure P_1).
2. Construction of partition Π_c of the set $C^1 \subset C$.
3. Optimal addressing of microinstructions, corresponding to operator vertices $b_q \in B_1$, which are components of OLC $\alpha_g \in C^1$.
4. Construction of the control memory content.
5. Construction of the transition table of CMCU.
6. Synthesis of CMCU logic circuit using given logical elements.

Let us apply this method to the design of CMCU $U_4(I_3)$. In this case procedure P_1 gives the set $C = \{\alpha_1, \dots, \alpha_7\}$, where $\alpha_1 = \langle b_{18} \rangle$, $\alpha_2 = \langle b_1, b_2, b_3 \rangle$, $\alpha_3 = \langle b_4, b_5, b_6 \rangle$, $\alpha_4 = \langle b_7, \dots, b_{10} \rangle$, $\alpha_5 = \langle b_{11}, \dots, b_{14} \rangle$, $\alpha_6 = \langle b_{15}, b_{16}, b_{17} \rangle$, $\alpha_7 = \langle b_{19} \rangle$. Let us point out that the OLC $\alpha_7 \notin C^1$.

Partition Π_c can be obtained simply by using the transformed GSA $\Gamma(U_4)$, which is the same as GSA $\Gamma(U_2)$. In our example, partition $\Pi_c = \{B_1, B_2, B_3\}$ is formed, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6\}$. Let m_i be the cardinality number of class $B_i \in \Pi_c$, and L_i maximal number of components for OLC $\alpha_g \in B_i$ ($i = 1, \dots, I$). Each class $B_i \in \Pi_c$ corresponds to a single unique generalized interval of the R_2 -dimensional Boolean space, iff each class $B_i \in \Pi_c$ corresponds to a subtable of Karnaugh map of the size:

$$V_i = 2^{r_i} \cdot L_i (i = 1, \dots, I), \quad (3.32)$$

where $r_i = \lceil \log_2 m_i \rceil$. In case of the compositional microprogram control unit $U_4(I_3)$ we have $V_1 = 1$, $V_2 = 4 \cdot 4 = 16$, $V_3 = 2 \cdot 4 = 8$. It is clear that optimal addressing is possible here, because for $R_2 = 5$, the Karnaugh map has 32 cells and we have

$$2^{R_2} > \sum_{i=1}^I V_i. \quad (3.33)$$

Optimal microinstruction addressing can be executed using some modifications of well-known algorithms given in [7]. One possible version of optimal microinstruction addressing for the CMCU $U_4(I_3)$ is shown in Fig. 3.16.

$T_1T_2T_3$ T_4T_5	000	001	011	010	110	111	101	100
00	b_{18}	b_{19}	*	*	b_7	*	b_{11}	*
01	*	*	b_1	b_4	b_8	*	b_{12}	b_{15}
11	*	*	b_2	b_5	b_9	*	b_{13}	b_{16}
10	*	*	b_3	b_6	b_{10}	*	b_{14}	b_{17}

Fig. 3.16 Optimal microinstruction addressing for CMCU $U_4(I_3)$

We get from Fig. 3.16 that $K(B_1) = 00***$ (the address A_{19} is treated as the don't care input assignment because only transitions from outputs of OLC $\alpha_g \in C^1$ are used), $K(B_2) = *1***$, $K(B_3) = 10***$.

Transition table of CMCU $U_4(\Gamma)$ can be constructed using the following procedure:

- construct the system of transition formulae for outputs of OLC $\alpha_g \in C^1$ (procedure P_5);
- replace the OLC outputs in the left part of each transition formula by the symbol of corresponding class $B_i \in \Pi_c$;
- if the transformed transition formulae include k equal transitions, only one of them should remain in the final system;
- use this system for construction of the transition table with columns: B_i , $K(B_i)$, I_m^j , $A(I_m^j)$, X_h , Φ_h , h .

Let us denote this procedure as P_7 . Its application to GSA $I_3(U_4)$ gives the following system of transition formulae:

$$\begin{aligned} B_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_3^1 \vee \bar{x}_1 \bar{x}_2 I_4^1; \\ B_2 &\rightarrow x_2 I_5^1 \vee \bar{x}_2 x_4 I_6^1 \vee \bar{x}_2 \bar{x}_4 I_4^2; \\ B_3 &\rightarrow x_3 I_5^2 \vee \bar{x}_3 I_7^1. \end{aligned} \quad (3.34)$$

System (3.34) is constructed using (3.11); and it serves to construct the transition table of CMCU $U_4(I_3)$, including $H_4(I_3) = 8$ lines (Table 3.5).

This table is now used to construct system (3.27), serving as the base to implement logic circuit of CMCU $U_4(\Gamma)$. This system has the form:

$$\varphi_r = \bigvee_{h=1}^{H_4(\Gamma)} C_{rh} B_i^h X_h \quad (\Gamma = 1, \dots, R_2), \quad (3.35)$$

where B_i^h is a conjunction of variables corresponding to the code $K(B_i)$ of class $B_i \in \Pi_C$ from the line h of transition table ($h = 1, \dots, H_4(\Gamma)$). It means that conjunction B_i^h has the form:

$$B_i^h = \bigwedge_{r=1}^{R_2} T_r^{l_{ri}} \quad (i = 1, \dots, I), \quad (3.36)$$

where $l_{ri} \in \{0, 1, *\}$ is a value of the bit r of code $K(B_i)$, $T_r^0 = \bar{T}_r$, $T_r^1 = T_r$, $T_r^* = 1$ ($r = 1, \dots, R_2$).

Table 3.5 Transition table of CMCU $U_4(I_3)$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
B_1	00***	I_2^1	00101	x_1	D_3D_5	1
		I_3^1	01001	\bar{x}_1x_2	D_2D_5	2
		I_4^1	11000	$\bar{x}_1\bar{x}_2$	D_1D_2	3
B_2	*1***	I_5^1	10100	x_2	D_1D_3	4
		I_6^1	10001	\bar{x}_2x_4	D_1D_5	5
		I_4^2	11010	$\bar{x}_2\bar{x}_4$	$D_1D_2D_4$	6
B_3	10***	I_5^2	10110	x_3	$D_1D_3D_4$	7
		I_7^1	00100	\bar{x}_3	D_3	8

Synthesis of CMCU U_4 logic circuit is reduced to the implementation of system (3.35) using PLD chips and of control memory using PROM chips. In this case we can construct, for example, the Boolean function $D_4 = B_2^6X_6 \vee B_3^7X_7 = T_2\bar{x}_2\bar{x}_4 \vee T_1\bar{T}_2x_3$, using Table 3.5.

In our example $T' = \{T_1, T_2\}$, thus $R_{FB}^4(I_3) = 2$ and we have the relation

$$R_{FB}^4(I_3) \leq R_2. \quad (3.37)$$

Let us point out that this value $R_{FB}^4(I_3) = 2$ is equal to $R_{FB}^1(I_3)$, which represents the minimum.

The number of lines in transition table of CMCU U_4 is determined as

$$H_4(\Gamma) = \sum_{i=1}^I k_i H_i, \quad (3.38)$$

where k_i is the number of generalized intervals in R_2 -dimensional Boolean space, including microinstruction addresses corresponding to the components of OLC $\alpha_g \in B_i$; H_i is the number of transitions from the output of any OLC $\alpha_g \in B_i$ ($i = 1, \dots, I$). In our example the condition

$$k_i = 1 \quad (i = 1, \dots, I) \quad (3.39)$$

is satisfied for all classes $B_i \in \Pi_c$ and $H_4(\Gamma_3) = 8 < H_1(\Gamma_3) = 9$.

Thus, if condition (3.39) is true, the CMCU $U_4(\Gamma)$ is characterized by minimal values of $S_4(\Gamma)$, $t_4(\Gamma)$ and $H_4(\Gamma)$, in comparison with the equivalent CMCU $U_1 - U_3$. There are some GSAs, for which condition (3.39) is false. In this case, the values of parameters $S_4(\Gamma)$ and $H_4(\Gamma)$ are far from minimum. For example, if $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6, \alpha_7\}$, $C = \{\alpha_1, \dots, \alpha_8\}$, $H_1 = 3$, $H_2 = 5$, $H_3 = 4$, $R_2 = 4$, minimal number of transitions is $H_4(\Gamma) = 12$. For optimal microinstruction addressing we have $k_1 = 1$, $k_2 = k_3 = 2$, and therefore $H_4(\Gamma) = 21 > H_1(\Gamma) = 13$.

Transformation of microinstruction addresses [3] is oriented towards reduction of hardware amount in CMCU logic circuit, if condition (3.39) is not satisfied. This method consists on transformation of microinstruction addresses of OLC outputs into codes of the classes of pseudoequivalent OLC.

Let us encode each class $B_i \in \Pi_c$ by a binary code $K(B_i)$ with

$$R_4 = \lceil \log_2 I \rceil \quad (3.40)$$

bits and use variables $\tau_r \in \tau$, where $|\tau| = R_4$, for encoding the classes $B_i \in \Pi_c$. Now, the initial GSA Γ is interpreted by CMCU U_5 (Fig. 3.17), where an address transformer AT transforms output addresses of OLC $\alpha_g \in B_i$ into the codes of classes $B_i \in \Pi_c$.

In compositional microprogram control unit U_5 , combinational circuit CC implements the input memory functions

$$\Phi = \Phi(\tau, X), \quad (3.41)$$

and address transformer AT implements functions of the system

$$\tau = \tau(T). \quad (3.42)$$

Let us point out that the control memory CM implements always functions (3.3)–(3.5). The CMCU U_5 operates as follows.

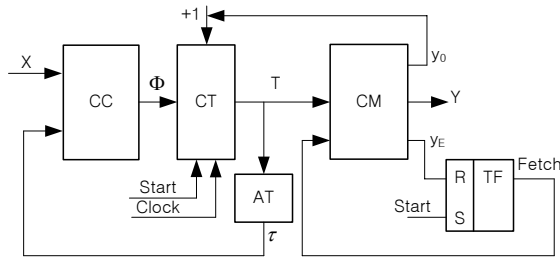


Fig. 3.17 Structural diagram of CMCU U_5

Pulse "Start" is used to load a zero address into the counter CT and to set up the flip-flop TF (it gives Fetch=1). If microinstruction $Y(b_q)$, where $b_q \neq O_g$ ($g = 1, \dots, G_1$), is read out of the CM, signal y_0 is generated and content of the counter

is incremented, in order to address next microinstruction, corresponding to next component of the current OLC. If $b_q = O_g$, a transition address is generated by combinational circuit CC using outputs of the address transformer AT and logical conditions. If microinstruction with y_E is read out of the CM, the flip-flop TF is cleared and operation of CMCU terminated.

Synthesis of CMCU $U_5(\Gamma)$ includes the following steps:

1. Preliminary transformation of GSA Γ .
2. Construction of the set of OLC for the transformed GSA $\Gamma(U_5)$.
3. Natural addressing of microinstructions.
4. Construction of control memory content.
5. Construction of partition Π_c of the set C^1 .
6. Encoding of the classes $B_i \in \Pi_c$.
7. Construction of the table of address transformer AT.
8. Construction of transition table of CMCU U_5 .
9. Synthesis of logic circuit of the CMCU.

Obviously, the first four steps give equal results for both CMCU $U_2(\Gamma)$ and $U_5(\Gamma)$. Let us discuss an example of the CMCU $U_5(\Gamma_3)$ design, starting from point 5 of the design method presented above.

Construction of partition Π_c of the set $C^1 = \{\alpha_1, \dots, \alpha_6\}$ results in the partition $\Pi_c = \{B_1, B_2, B_3\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6\}$. This step is executed in a trivial way using only the definition of pseudoequivalent operational linear chains.

Encoding of the classes of pseudoequivalent OLC is executed in a trivial way too. In case of the CMCU $U_5(\Gamma_3)$ we have $R_4 = 2$, $\tau = \{\tau_1, \tau_2\}$. Let the classes have the codes: $K(B_1) = 00$, $K(B_2) = 01$, $K(B_3) = 10$.

Construction of address transformer table is reduced to construction of the table with columns $O_g, A(O_g), B_i, K(B_i), \tau_g, g$, where column τ_g includes functions $\tau_r \in \tau$, equal to 1 in the code $K(B_i)$ from line g of the table ($g = 1, \dots, G_1$).

In case of the CMCU $U_5(\Gamma_3)$, output addresses are taken from the addressing table shown in Fig. 3.9. The address transformer table for the CMCU $U_5(\Gamma_3)$ includes $G_1 = 6$ lines (Table 3.6).

Let us point out that functions $\tau_r \in \tau$ are specified here only for 6 from all 32 possible input assignments. Thus, it is better to use some PLD chips instead of PROM chips to implement the logic circuit of address transformer for our particular example. Of course, in some other cases, PROM chips could be preferable. It depends on characteristics of the interpreted graph-scheme of algorithm. If PLD chips are used, the insignificant input assignments can be used to minimize disjunction normal forms of the functions $\tau_r \in \tau$. We can get, for example, the following expressions from Table 3.6:

$$\begin{aligned}\tau_1 &= \bar{T}_1 T_2 T_3 T_4 \bar{T}_5 \vee T_1 \bar{T}_2 \bar{T}_3 \bar{T}_4 T_5; \\ \tau_2 &= \bar{T}_1 \bar{T}_2 \bar{T}_3 T_4 \vee \bar{T}_1 \bar{T}_2 \bar{T}_3 T_4 \bar{T}_5 \vee \bar{T}_1 T_2 \bar{T}_3 T_4 \bar{T}_5.\end{aligned}\tag{3.43}$$

Let us construct the Karnaugh map for function τ_1 (Fig. 3.18). Using this map, the minimal form of function τ_1 can be obtained:

Table 3.6 Table of address transformer of CMCU $U_5(I_3)$

O_g	$A(O_g)$	B_i	$K(B_i)$	τ_g	g
O_1	00000	B_1	00	-	1
O_2	00011	B_2	01	τ_2	2
O_3	00110	B_2	01	τ_2	3
O_4	01010	B_2	01	τ_2	4
O_5	01110	B_3	10	τ_1	5
O_6	10001	B_3	10	τ_1	6

$$\tau_1 = T_2T_3 \vee T_1. \tag{3.44}$$

$T_4T_5 \backslash T_1T_2T_3$		$T_1T_2T_3$							
		000	001	010	011	100	101	110	111
00	0	0	*	*	*	*	*	*	*
	01	*	*	*	*	*	*	*	1
11	0	0	*	*	*	*	*	*	*
	10	*	0	1	0	*	*	*	*

Fig. 3.18 Karnaugh map for function τ_1

The same approach gives the minimal form of function τ_2 :

$$\tau_2 = \bar{T}_1T_5 \vee \bar{T}_2T_3 \vee T_2\bar{T}_3. \tag{3.45}$$

Construction of transition table is executed by analogy with the case of CMCU U_4 and both tables include the same columns. Transition table of the CMCU $U_5(I_3)$ includes $H_5(I_3) = 8$ lines (Table 3.7).

Table 3.7 Transition table of CMCU $U_5(I_3)$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
B_1	00	I_2^1	00001	x_1	D_5	1
		I_3^1	00100	\bar{x}_1x_2	D_2	2
		I_4^1	00111	$\bar{x}_1\bar{x}_2$	$D_3D_4D_5$	3
B_2	01	I_5^1	01011	x_2	$D_2D_4D_5$	4
		I_6^1	01111	\bar{x}_2x_4	$D_2D_3D_4D_5$	5
		I_4^2	01001	$\bar{x}_2\bar{x}_4$	D_2D_5	6
B_3	10	I_5^2	01101	x_3	$D_2D_3D_5$	7
		I_7^1	10010	\bar{x}_3	D_1D_4	8

This table serves as a base to get the system (3.41), represented in the form:

$$\varphi_r = \bigvee_{h=1}^{H_5(\Gamma)} C_{rh} B_i^h X_h \quad (r = 1, \dots, R_2). \quad (3.46)$$

All terms of (3.46) have the same meaning as in case of (3.35); only the term (3.47) given below is different:

$$B_i^h = \bigwedge_{r=1}^{R_4} \tau_r^{l_{ri}}, \quad (3.47)$$

where $l_{ri} \in \{0, 1\}$ is the bit r of code $K(B_i)$, $\tau_r^0 = \bar{\tau}_r$, $\tau_r^1 = \tau_r$ ($r = 1, \dots, R_4$). For example, from Table 3.7 we find that $D_1 = \tau_1 \bar{\tau}_2 \bar{x}_3$.

Synthesis of logic circuit of CMCU is reduced to the implementation of functions (3.41) – (3.42) using PLD chips and functions (3.3) – (3.5) using PROM chips. Logic circuit of the CMCU $U_5(I_3)$ is shown in Fig. 3.19.

Analysis of Fig. 3.19 shows that PLA implementing the address transformer AT has only 4 inputs, because functions (3.44) and (3.45) do not depend on variable T_4 .

General analysis of CMCU U_5 leads to the conclusion that its combinational part is characterized by minimal values of inputs, outputs and terms, as compared with all CMCU considered above:

$$\begin{aligned} R_{FB}^5 &= R_4 = R_1; \\ t_5(\Gamma) &= R_2; \\ H_5(\Gamma) &= H_4(\Gamma) \text{ if } k_i = 1 \quad (i = 1, \dots, I). \end{aligned} \quad (3.48)$$

We should also remember that CMCU U_5 includes an address transformer AT. In consequence, this method is useful only if the total hardware amount of blocks CC and AT is smaller then the hardware amount of combinational circuit CC of other compositional microprogram control units. As we know, the hardware amount can be calculated either as the chip area or as the number of chips, needed for implementation of the CMCU logic circuit.

Transformation of output codes of OLC is oriented to reduction of the hardware amount in combinational circuit CC [4]. Main idea of this method consists on application of special microinstruction addressing (procedure P_6) and on the transformation of output code $K(O_g)$ into the code of class $B_i \in \Pi_c$, resulting in the CMCU U_6 of Fig. 3.20.

The only difference between CMCU U_6 and U_5 is the application of special microinstruction addressing, resulting in replacement of (3.42) by the system of Boolean functions:

$$\tau = \tau = (T'). \quad (3.49)$$

Both CMCU U_5 and U_6 operate in the same manner, and corresponding design methods differ only in microinstruction addressing. Let us consider these peculiarities using an example of CMCU $U_6(I_3)$ synthesis. In this case, application of procedure P_6 gives the microinstruction addresses shown in Fig. 3.15c. Table of the

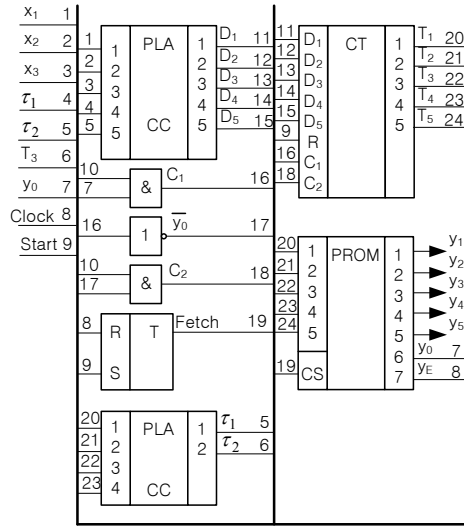


Fig. 3.19 Logic circuit of CMCU $U_5(I_3)$

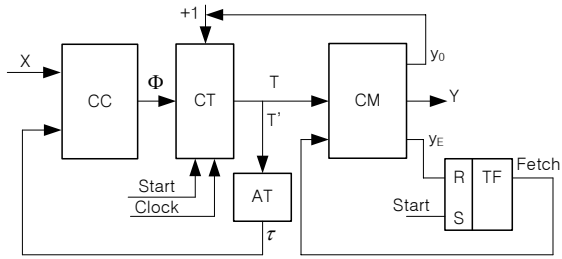


Fig. 3.20 Structural diagram of CMCU U_6

address transformer AT of CMCU U_5 should be replaced by corresponding table of code transformer TC, with columns: $O_g, K(O_g), B_i, K(B_i), \tau_g, g$.

Analysis of Table 3.6 shows that the number of terms in system (3.42) can be reduced using some modification of the well-known state assignment method of FSM with D flip-flops, used to implement its memory [2]. We use this method formulated as follows: the more elements some class $B_i \in \Pi_c$ include, the more zeros should its code have. Application of this approach for the CMCU $U_6(I_3)$ gives codes: $K(B_1) = 10, K(B_2) = 00, K(B_3) = 01$. These codes are used in Table 3.8.

From Table 3.8 we get system (3.48) and in particular: $\tau_1 = \overline{T_1} \overline{T_2} \overline{T_3}; \tau_2 = T_1 \overline{T_2}$. Logic circuits for both CMCU $U_5(I_3)$ and $U_6(I_3)$ are practically identical, but block TC can be implemented using PLA chip with three outputs and two terms. Therefore, application of special microinstruction addressing leads to the decrease of hardware amount in this particular case.

Transformation of initial GSA is reduced to introduction of some extra vertices corresponding to additional OLC $\alpha_g \in C^2$ in GSA $\Gamma(U_2)$, where

$$|C^2| \leq I. \tag{3.50}$$

Table 3.8 Table of code transformer TC of CMCU $U_6(\Gamma_3)$

O_g	$K(O_g)$	B_i	$K(B_i)$	τ_g	g
O_1	000	B_1	10	τ_1	1
O_2	001	B_2	00	–	2
O_3	010	B_2	00	–	3
O_4	011	B_2	00	–	4
O_5	100	B_3	01	τ_2	5
O_6	101	B_3	01	τ_2	6

This idea is an analogue of the one used for optimization of Moore FSM logic circuit [4] and is presented for the block GSA Γ_3 (Fig. 3.12). In this case OLC $\alpha_2, \alpha_3, \alpha_4 \in B_2$ and total number of transitions from these OLC outputs is equal to 9. Let us add an extra block to this GSA Γ_3 , corresponding to the OLC $\alpha_8 \in C^2$ with input I_8^1 and output O_8 (Fig. 3.21).

It is clear that the transformation of a subgraph i of GSA makes sense only if condition (3.51) is satisfied:

$$\Delta H_i > 0. \tag{3.51}$$

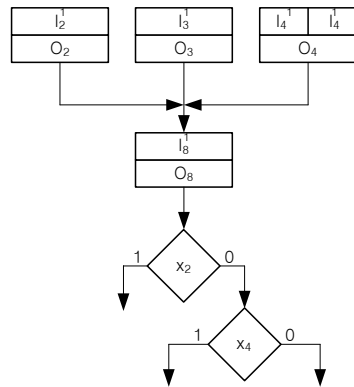


Fig. 3.21 Transformation of subgraph of block GSA Γ_3

Introduction of extra OLC should not cause exceeding the minimal value of R_2 , thus cardinality number of set C^2 should satisfy the following condition:

$$|C^2| \leq 2^{R_2} - M_2. \tag{3.52}$$

If condition (3.52) holds, transformation of GSA is executed for all subgraphs satisfying condition (3.51). Otherwise, the subgraphs should be ranked in order of decreasing ΔH_i value and only first K subgraphs from this list should be transformed, where

$$K = 2^{R_2} - M_2. \tag{3.53}$$

Let us call this method procedure P_8 . Application of this procedure to the GSA Γ_3 is shown in Fig. 3.21.

In this case formula (3.53) gives the value $K = 32 - 19 = 13$. The number of classes in partition Π_c is equal to 3, and therefore condition (3.52) holds and the transformation is possible for all classes $B_i \in \Pi_c$. In our last example we have the following values of parameters $|B_i|, H_i$ and ΔH_i : $|B_1| = 1, H_1 = 3, \Delta H_1 = -1$; $|B_2| = 3, H_2 = 3, \Delta H_2 = 3$; $|B_3| = 2, H_3 = 2, \Delta H_3 = 0$. It means that the transformation has sense only for the class $B_2 \in \Pi_c$. Thus only one extra OLC is added and $C^2 = \{\alpha_8\}$.

Let the CMCU based on procedure P_8 be denoted by U_7 . It is clear, that structures of both CMCU U_2 and U_7 are the same. Optimization of the number of combinational circuit CC inputs can be executed due to application of special addressing for microinstructions, corresponding to operator vertices of the transformed GSA $\Gamma(U_7)$.

Synthesis of CMCU $U_7(\Gamma)$ includes the following steps:

1. Preliminary transformation of initial GSA Γ (procedure P_4) and construction of GSA $\Gamma(U_2)$.
2. Construction of the set C for GSA $\Gamma(U_2)$ (procedure P_1).
3. Construction of partition Π_c .
4. Construction of transformed GSA $\Gamma(U_7)$ (procedure P_8).
5. Special microinstruction addressing (procedure P_6).
6. Construction of the control memory content.
7. Construction of CMCU transition table.
8. Synthesis of CMCU logic circuit.

Let us apply this method to the case of CMCU $U_7(\Gamma_3)$. Application of points 1–3 gives the GSA $\Gamma_3(U_2)$ shown in Fig. 3.8, the set $C = \{\alpha_1, \dots, \alpha_7\}$ and partition $\Pi_c = \{B_1, B_2, B_3\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6\}$.

Construction of transformed GSA have been already discussed and is reduced to introduction of vertex b_{20} into the GSA $\Gamma_3(U_2)$. The result of this GSA transformation is shown in Fig. 3.22.

Special microinstruction addressing is executed in the same way as in previous example, with OLC outputs encoded using

$$R_5 = \lceil \log_2(G_1 + G_2) \rceil \quad (3.54)$$

bits, where $|G_2| = |C^2|$. Result of procedure P_6 used for CMCU $U_7(\Gamma_3)$ is shown in Fig. 3.23.

As follows from Fig. 3.23, outputs of OLC $\alpha_g \in C^1 \cup C^2$ have the following codes: $K(O_1) = 000, K(O_2) = 001, \dots, K(O_6) = 101, K(O_8) = 110$.

Construction of transition table of CMCU is executed using the same approach as the one used in case of CMCU U_3 . In this case, transition table has 13 lines (Table 3.9).

Usually, the number of transition table lines for CMCU U_7 can be found using the following expression:

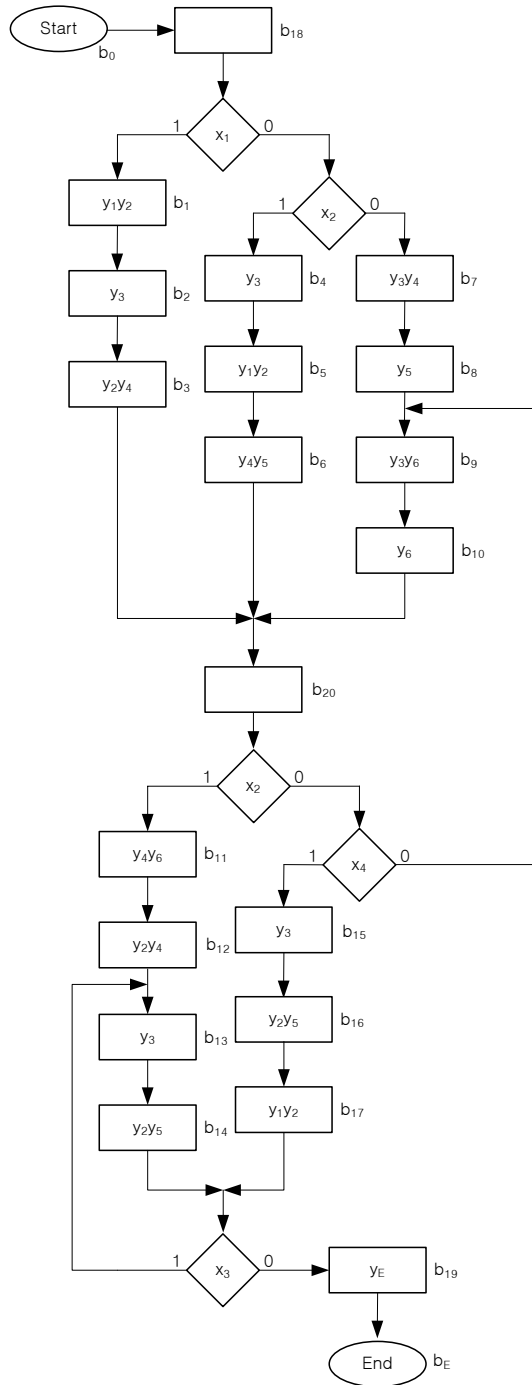


Fig. 3.22 Transformed GSA $I_3 (U_7)$

$T_1T_2T_3$		000	001	010	011	100	101	110	111
		T_4T_5							
00	00	18	3	6	10	14	17	20	*
	01	*	*	7	11	*	*	19	*
11	11	1	4	8	12	15	*	*	*
	10	2	5	9	13	16	*	*	*

Fig. 3.23 Special microinstruction addressing for CMCU $U_7(\Gamma_3)$

$$H_7(\Gamma) = H_2(\Gamma) - \sum_{i=1}^I \Delta H_i \cdot C_i, \tag{3.55}$$

where C_i is a Boolean variable, equal to 1 iff the class $B_i \in \Pi_c$ was transformed.

Table 3.9 Transition table of CMCU $U_7(\Gamma_3)$

O_g	$K(O_g)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
O_1	000	I_2^1	00010	x_1	D_4	1
		I_3^1	00110	\bar{x}_1x_2	D_3D_4	2
		I_4^1	01001	$\bar{x}_1\bar{x}_2$	D_2D_5	3
O_2	001	I_8^1	11000	1	D_1D_2	4
O_3	010	I_8^1	11000	1	D_1D_2	5
O_4	011	I_8^1	11000	1	D_1D_2	6
O_5	100	I_5^2	01101	x_3	$D_2D_3D_4D_5$	7
		I_7^1	10010	\bar{x}_3	D_1D_4	8
O_6	101	I_5^2	01111	x_3	$D_2D_3D_4D_5$	9
		I_7^1	10010	\bar{x}_3	D_1D_4	10
O_8	110	I_5^1	01101	x_2	$D_2D_3D_5$	11
		I_6^1	10010	\bar{x}_2x_4	D_1D_4	12
		I_4^2	01011	$\bar{x}_2\bar{x}_4$	$D_2D_4D_5$	13

This transition table serves as the base for construction of the system (3.27), with functions represented in the form (3.25).

Synthesis of logic circuit of CMCU $U_7(\Gamma)$ is executed in the same manner as in case of CMCU $U_3(\Gamma)$.

Let us point out that the transformation of GSA allows to reduce the number of transition table lines in comparison with $H_3(\Gamma)$, but this merit is accompanied with some negative effect, namely increase of the number of cycles needed to execute the transformed control algorithm.

In Table 3.10 the main characteristics of CMCU $U_1 - U_7$ are compared. Symbol H_{\min} stays here for the number of transition table lines in case of CMCU $U_1(\Gamma)$, whereas symbol H_{\max} is used for the number of transition table lines in case of CMCU $U_2(\Gamma)$. Analysis of this table leads to the conclusion that CMCU $U_4(\Gamma)$

has the best characteristics, but as should be remembered, these characteristics are achievable only if condition (3.39) is satisfied. In case of CMCU $U_5(\Gamma)$ and $U_6(\Gamma)$ the main characteristics are also minimal, but the use of additional blocks, consuming either some area of the chip (if the control unit is implemented as a part of digital system on single chip) or some additional chips if standard PLDs should be used to implement logic circuit of the CMCU. Control unit $U_7(\Gamma)$ has an average hardware amount, but performance of the controlled digital system decreases due to higher number of cycles required to execute the control algorithm. Thus, a priori choice of the best model of CMCU is not possible. Here, the expression "a priori" is the equivalent of "without implementation of CMCU logic circuit". Final choice of the best model of CMCU can be made only after implementation of logic circuits for all microprogram control units $U_1 - U_7$, using particular programmable logic devices and particular control algorithm. As always in this book we assume that the best model of CMCU is the one, which combines minimal possible hardware amount (of all models analyzed) with performance meeting the initial requirements of the digital system designer.

Table 3.10 Comparative characteristic of CMCU $U_1 - U_7$

Model	Boolean systems	Parameters of PLDs	Comments
U_1	$\Phi = \Phi(\tau, X) \quad \Psi = \Psi(\tau, X)$	$S_1(\Gamma) = R_1 + L$	Minimum
		$t_1(\Gamma) = R_1 + R_2$	Maximum
		$H_1(\Gamma) = H_{\min}$	Minimum
U_2	$\Phi = \Phi(T, X)$	$S_2(\Gamma) = R_2 + L$	Maximum
		$t_2(\Gamma) = R_2$	Minimum
		$H_2(\Gamma) = H_{\max}$	Maximum
U_3	$\Phi = \Phi(T', X)$	$S_3(\Gamma) = R_3 + L$	$\geq S_1(\Gamma)$
		$t_3(\Gamma) = R_2$	Minimum
		$H_3(\Gamma) = H_{\max}$	Maximum
U_4	$\Phi = \Phi(T', X)$	$S_4(\Gamma) = R_4 + L$	$\geq S_1(\Gamma)$
		$t_4(\Gamma) = R_2$	Minimum
		$H_4(\Gamma) = H_{\min}$	Minimum if (3.39)
U_5	$\Phi = \Phi(\tau, X) \quad \tau = \tau(T)$	$S_5(\Gamma) = R_4 + L$	Minimum
		$t_5(\Gamma) = R_2$	Minimum
		$H_5(\Gamma) = H_{\min}$	Minimum
U_6	$\Phi = \Phi(\tau, X) \quad \tau = \tau(T')$	$S_6(\Gamma) = R_4 + L$	Minimum
		$t_6(\Gamma) = R_2$	Minimum
		$H_6(\Gamma) = H_{\min}$	Minimum
U_7	$\Phi = \Phi(T', X)$	$S_7(\Gamma) = R_5 + L$	$\geq S_1(\Gamma)$
		$t_7(\Gamma) = R_2$	Minimum
		$H_{\min} < H_7(\Gamma) \leq H_{\max}$	Average

References

1. M. Adamski and A. Barkalov. *Architectural and Sequential Synthesis of Digital Devices*. University of Zielona Góra Press, 2006.
2. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
3. A. A. Barkalov. *Development of formalized methods of structure synthesis for compositional automata*. PhD thesis, Donetsk:DonSTU, 1994. (in Russian).
4. A. A. Barkalov. Principles of optimization of logic circuit of Moore FSM. *Cybernetics and System Analysis*, (1):65–72, 1998. (in Russian).
5. A. A. Barkalov. *Synthesis of Control Units with PLDs*. Donetsk National Technical University, 2002. (in Russian).
6. A.A. Barkalov. Microprogram control unit as composition of automate with programmable and hardwired logic. *Automatics and computer technique*, (4):36–41, 1983. (in Russian).
7. T. Łuba, K. Jasiński, and B. Zbierchowski. *Specialized digital circuits in PLD i FPGA structures*. Wydawnictwo Komunikacji i Łączności, 1997. (in Polish).

Chapter 4

Synthesis of compositional microprogram control units with code sharing

Abstract The chapter is devoted to the design methods based on the so-called code sharing, in which the microinstruction address is represented by concatenation of the OLC code and the code of its component. This representation makes possible using special methods of Moore FSM optimization adapted to the peculiarities of CMCU. In this case, the OLC codes are viewed as analogs of the states codes of Moore FSM. This approach permits to reduce the number of inputs and outputs of the combinational part of CMCU. Additional decrease can be achieved due to the application of elementary OLCs having only one input. In this case, the address of first component of each OLC is represented by all zeros and permits to diminish the number of functions generated by FSM.

4.1 Synthesis of CMCU basic model with code sharing

One of the main goals of all methods discussed in previous chapters is minimizing the number of feedback variables used as inputs of combinational circuit CC for generation of transition addresses. One solution of this problem is the method of code sharing proposed in [3] and adapted to the peculiarities of compositional microprogram control units.

Let each operational linear chain $\alpha_g \in C$ corresponds to a binary code $K(\alpha_g)$ with the following number of bits:

$$R_G = \lceil \log_2 G \rceil, \quad (4.1)$$

where $G = |C|$, and let variables $\tau_r \in \tau$ be used for this encoding ($|\tau| = R_G$). Let L_g be the number of components in OLC $\alpha_g \in C$. We find the following relation:

$$L_{\max} = \max(L_1, \dots, L_G). \quad (4.2)$$

Let components $b_q \in D^s$ of OLC $\alpha_g \in C$ be encoded by binary codes $K(b_q)$ with the following number of bits:

$$R_7 = \lceil \log_2 L_{\max} \rceil. \quad (4.3)$$

Let some variables $T_r \in T$, where $|T| = R_7$, be used for this encoding. Let us encode the components in such a way that any pair of adjacent components $b_t, b_q \in D^8$ of OLC $\alpha_g \in C$, where $\langle b_t, b_q \rangle \in E$, satisfies the condition:

$$K(b_q) = K(b_t) + 1. \quad (4.4)$$

In this case an address $A(b_q)$ of microinstruction, corresponding to vertex $b_q \in D^8$, can be represented by a concatenation of the codes of OLC $\alpha_g \in C$ and its component $K(b_q)$:

$$A(b_q) = K(\alpha_g) * K(b_q), \quad (4.5)$$

where $*$ is a sign of concatenation. The address presentation (4.5) is called code sharing [3, 7].

If microinstruction addresses are given in the form (4.5), the initial GSA Γ is interpreted by CMCU with code sharing (Fig. 4.1), denoted in this book as CMCU U_8 .

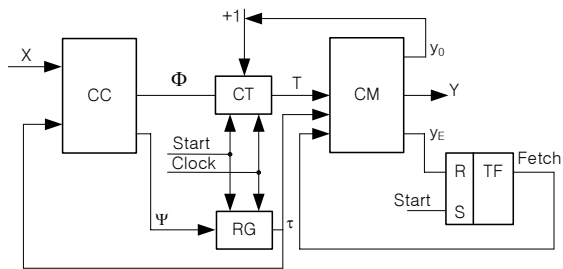


Fig. 4.1 Structural diagram of CMCU U_8

In the CMCU U_8 , combinational circuit CC implements the following input memory functions for counter CT

$$\Phi = \Phi(\tau, X), \quad (4.6)$$

and input memory functions for register RG

$$\Psi = \Psi(\tau, X). \quad (4.7)$$

Control memory CM of CMCU U_8 implements the system of output functions

$$Y = Y(\tau, T); \quad (4.8)$$

$$\begin{aligned} y_0 &= y_0(\tau, T); \\ y_E &= y_E(\tau, T). \end{aligned} \quad (4.9)$$

The CMCU U_8 operates in the following manner. Zero codes are loaded into both RG and CT using pulse "Start". It corresponds to the address of first microinstruction

of a particular microprogram. This pulse causes also the set up of flip-flop TF (now Fetch=1) and microinstructions can be read from the CM. Current microinstruction is fetched out of the control memory CM. If concatenation of contents of register RG and counter CT forms an address $A(b_q)$, where $b_q \in D^g$ and $b_q \neq O_g$, variable $y_0 = 1$ is generated together with microoperations $y_n \in Y(b_q)$. If $y_0 = 1$, content of RG remains unchanged, but the content of CT is incremented according to the addressing mode (4.4). If concatenation of the contents of register RG and counter CT forms an address $A(b_q)$ of OLC $\alpha_g \in C$ output, then $y_0 = 0$. In this case transition address is generated by combinational circuit CC. If the concatenation of contents of register RG and counter CT forms an address $A(b_q)$, where $\langle b_q, b_E \rangle \in E$, signal y_E is generated, flip-flop TF is reset and operation of CMCU terminated.

Such organization of CMCU gives the following advantages:

1. Microinstruction addresses are not used as state codes of addressing FSM. States codes correspond to the codes of OLC and can be chosen arbitrarily. It means that all known methods used for optimization of Moore FSM logic circuit can be used to optimize the logic circuit of addressing FSM S_1 of CMCU with code sharing.
2. Combinational circuit CC uses only R_6 feedback variables and reduces the number of inputs, in comparison with CMCU U_2 .
3. Microinstruction address has

$$R_8 = R_6 + R_7 \quad (4.10)$$

bits and if the following condition holds

$$R_8 = R_2, \quad (4.11)$$

the number of outputs of combinational circuit CC attains minimum.

If however condition (4.11) is violated, and the following inequality is satisfied

$$R_8 > R_2, \quad (4.12)$$

control memory size is increased

$$2^{R_8 - R_2} \quad (4.13)$$

times in comparison with its minimal value, determined as

$$V_{\min} = 2^{R_2} \cdot m_Y, \quad (4.14)$$

where m_Y is the number of bits (length) of the control memory word. It depends on encoding of collections of microoperations. Therefore, application of code sharing makes sense only if condition (4.11) is satisfied.

Synthesis of CMCU U_8 includes the following steps:

- Preliminary transformation of initial GSA Γ .
- Construction of the set C for transformed GSA $\Gamma(U_8)$.
- Encoding of operational linear chains $\alpha_g \in C$.

- Encoding of OLC $\alpha_g \in C$ components.
- Construction of control memory content.
- Construction of CMCU $U_8(\Gamma)$ transition table.
- Synthesis of CMCU logic circuit using given logical elements.

Let us discuss application of this method for synthesis of CMCU $U_8(\Gamma_4)$, with the transformed GSA $\Gamma_4(U_8)$ shown in Fig. 4.2.

It should be pointed out that transformation of the initial GSA is reduced to application of procedure P_4 . Obviously, in the case of GSA Γ_4 , variable y_E was inserted into the vertices b_{14} and b_{17} of the initial GSA in order to organize the stop mode of CMCU $U_8(\Gamma_4)$.

Construction of the set of OLC is executed using procedure P_1 . In our example we get $C = \{\alpha_1, \dots, \alpha_6\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1^1 = b_1$, $O_1 = b_2$, $L_1 = 2$; $\alpha_2 = \langle b_3, b_4, b_5 \rangle$, $I_2^1 = b_3$, $I_2^2 = O_2 = b_5$, $L_2 = 3$; $\alpha_3 = \langle b_6, b_7 \rangle$, $I_3^1 = b_6$, $O_3 = b_7$, $L_3 = 2$; $\alpha_4 = \langle b_8, b_9, b_{10} \rangle$, $I_4^1 = b_8$, $O_4 = b_{10}$, $L_4 = 3$; $\alpha_5 = \langle b_{11}, \dots, b_{14} \rangle$, $I_5^1 = b_{11}$, $O_5 = b_{14}$, $L_5 = 4$; $\alpha_6 = \langle b_{15}, b_{16}, b_{17} \rangle$, $I_6^1 = b_{15}$, $O_6 = b_{17}$, $L_6 = 3$.

Encoding of OLC is executed in a trivial way. In present example we have $G = 6$, and $R_6 = 3$, $\tau = \{\tau_1, \tau_2, \tau_3\}$. Let us encode the OLC $\alpha_g \in C$ as: $K(\alpha_1) = 000, \dots, K(\alpha_6) = 101$.

Encoding of OLC components is executed in such a way, that condition (4.4) takes place for all components. To achieve this, it is sufficient to assign the number 0 to the first components of all OLCs, the number 1 to the second components of all OLCs, and so on. Binary representations of these numbers using R_7 bits are treated as the codes required for these components.

In present example $L_{\max} = 4$, $T = \{T_1, T_2\}$ and codes of components are shown in Table 4.1.

Table 4.1 Codes of components for CMCU $U_8(\Gamma)$

$K(b_q)$	α_1	α_2	α_3	α_4	α_5	α_6
00	b_1	b_3	b_6	b_8	b_{11}	b_{15}
01	b_2	b_4	b_7	b_9	b_{12}	b_{16}
10	–	b_5	–	b_{10}	b_{13}	b_{17}
11	–	–	–	–	b_{14}	–

Encoding of both OLCs $\alpha_g \in C$ and their components gives the table of Fig. 4.3, representing microinstruction addresses (4.5).

Using this table we find, for example, that $A(b_4) = 00101$, $A(b_{17}) = 10110$, and so on.

Construction of control memory content is executed using the same approach as for CMCU U_1-U_8 .

Construction of transition table of CMCU U_8 is performed in the same order as in case of CMCU U_2 , but the OLC outputs in the left parts of transition formulae are replaced this time by symbols of corresponding OLC $\alpha_g \in C^1$. In consequence,

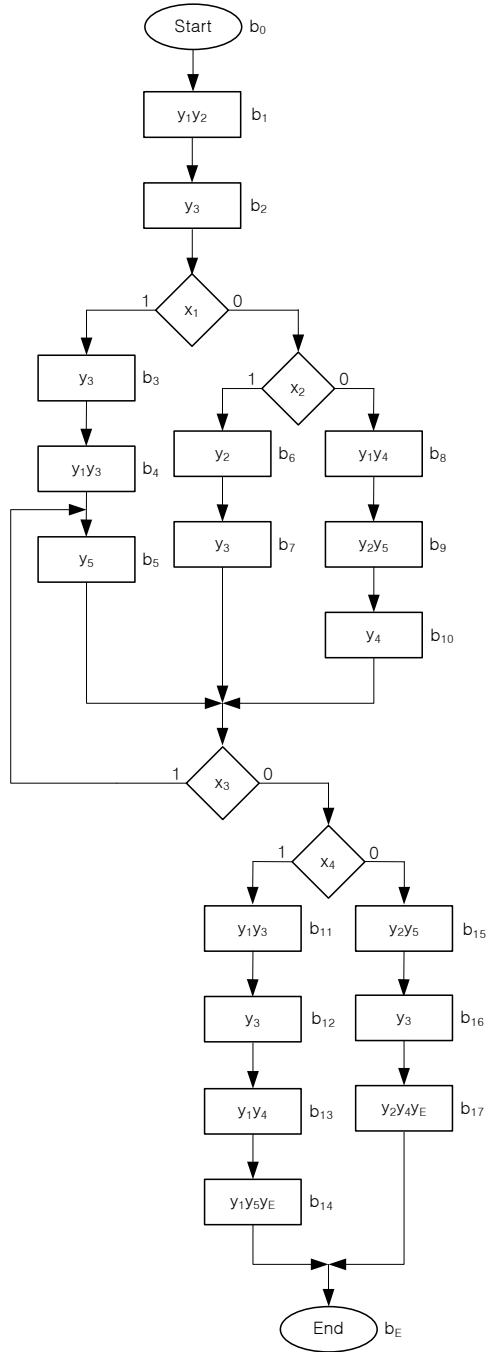


Fig. 4.2 Transformed GSA $\Gamma_4(U_8)$

$T_1T_2T_3$		T_4T_5							
		000	001	010	011	100	101	110	111
00	00	b_1	b_3	b_6	b_8	b_{11}	b_{15}	*	*
	01	b_2	b_4	b_7	b_9	b_{12}	b_{16}	*	*
11	11	*	b_5	*	b_{10}	b_{13}	b_{17}	*	*
	10	*	*	*	*	b_{14}	*	*	*

Fig. 4.3 Microinstruction addresses for CMCU $U_8(\Gamma)$

the following system of transition formulae can be found for CMCU $U_8(\Gamma)$, where set $C^1 = \{\alpha_1, \dots, \alpha_4\}$:

$$\begin{aligned}
 \alpha_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_3^1 \vee \bar{x}_1 \bar{x}_2 I_4^1; \\
 \alpha_2 &\rightarrow x_3 I_2^2 \vee \bar{x}_3 x_4 I_5^1 \vee \bar{x}_3 \bar{x}_4 I_6^1; \\
 \alpha_3 &\rightarrow x_3 I_2^3 \vee \bar{x}_3 x_4 I_5^1 \vee \bar{x}_3 \bar{x}_4 I_6^1; \\
 \alpha_4 &\rightarrow x_3 I_2^2 \vee \bar{x}_3 x_4 I_5^1 \vee \bar{x}_3 \bar{x}_4 I_6^1.
 \end{aligned} \tag{4.15}$$

Transition table of CMCU U_8 includes the following columns: α_g , $K(\alpha_g)$, I_m^j , $A(I_m^j)$, X_h , Ψ_h , Φ_h , h , where column Ψ_h contains input memory functions used to load OLC code $K(\alpha_m)$ into register RG, and column Φ_h contains input memory functions necessary to load the code $K(b_q)$ of some component of OLC α_m into counter CT, where $b_q = I_m^j$, $h = 1, \dots, H_8(\Gamma)$. The transition table of CMCU $U_8(\Gamma)$ has $H_8(\Gamma) = 12$ lines (Table 4.2).

Table 4.2 Transition table of CMCU $U_8(\Gamma)$

α_g	$K(\alpha_g)$	I_m^j	$A(I_m^j)$	X_h	Ψ_h	Φ_h	h
α_1	000	I_2^1	00100	x_1	D_3	–	1
		I_3^1	01000	$\bar{x}_1 x_2$	D_2	–	2
		I_4^1	01100	$\bar{x}_1 \bar{x}_2$	$D_2 D_3$	–	3
α_2	001	I_2^2	00110	x_3	D_3	D_4	4
		I_5^1	10000	$\bar{x}_3 x_4$	D_1	–	5
		I_6^1	10100	$\bar{x}_3 \bar{x}_4$	$D_1 D_3$	–	6
α_3	010	I_2^2	00110	x_3	D_3	D_4	7
		I_5^1	10000	$\bar{x}_3 x_4$	D_1	–	8
		I_6^1	10100	$\bar{x}_3 \bar{x}_4$	$D_1 D_3$	–	9
α_4	011	I_2^2	00110	x_3	D_3	D_4	10
		I_5^1	10000	$\bar{x}_3 x_4$	D_1	–	11
		I_6^1	10100	$\bar{x}_3 \bar{x}_4$	$D_1 D_3$	–	12

This transition table serves for construction of systems (4.6)–(4.7), depending on the terms

$$F_h = A_g^h X_h \quad (h = 1, \dots, H_8(\Gamma)), \tag{4.16}$$

where A_g^h is a conjunction of variables $\tau_r \in \tau$, corresponding to the code $K(\alpha_g)$ of OLC $\alpha_g \in C^1$ from the line h of the table. From Table 4.2 we find, for example, that $D_4 = \bar{\tau}_1 \bar{\tau}_2 \tau_3 x_3 \vee \bar{\tau}_1 \tau_2 \bar{\tau}_3 x_3 \vee \bar{\tau}_1 \tau_2 \tau_3 x_3$.

Synthesis of logic circuit of CMCU reduces to the implementation of systems (4.6)–(4.7) using PLD chips and of systems (4.8)–(4.9) using PROM chips. Logic circuit of CMCU $U_8(\Gamma_4)$ is shown in Fig. 4.4.

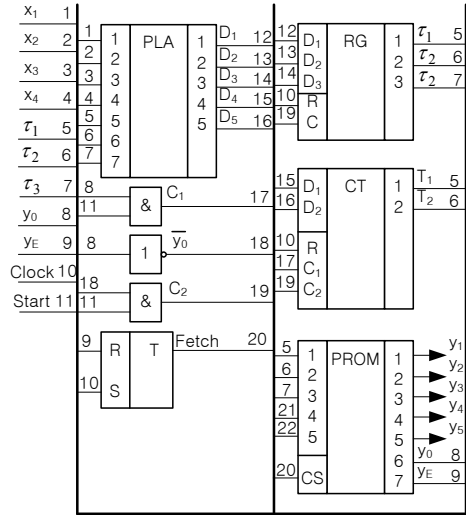


Fig. 4.4 Logic circuit of CMCU $U_8(\Gamma_4)$

In present case condition (4.11) holds and minimal values of both the number of combinational circuit outputs and size of control memory are obtained. It is possible, however, that condition (4.11) is violated, as can be seen from addressing table of CMCU $U_8(\Gamma_A)$ (Fig. 4.5).

As follows from Fig. 4.5, the number of microinstructions $M_2 = 19$ and $R_2 = 5$, but $L_{max} = 6$ (for the OLC α_3), and in consequence $R_7 = 3$. In this case the set C includes $G = 6$ operational linear chains, shown in Fig. 4.5, and $R_6 = 3$ variables are necessary for the encoding. Therefore, $R_6 + R_7 = 6$ and condition (4.11) is violated. According to (4.13), the control memory size grows twice, in comparison with V_{min} .

In order to satisfy condition (4.11), some of the initial OLCs should be transformed in such a way that the value of L_{max} could be reduced. Let us call this transformation a procedure P_9 , having the following steps:

1. Find the OLC $\alpha_g \in C$ with $L_g = L_{max}$.
2. Divide the OLC α_g by two OLCs α_g and α_{G+1} in such a way that L_g is reduced twice.
3. Put $G := G + 1$
4. Find new value of parameter L_{max} .
5. Find new value of parameter R_8 .
6. If $R_8 = R_2$, go to point 8.

$\tau_1\tau_2\tau_3$	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8
$T_1T_2T_3$	000	001	010	011	100	101	110	111
000	b_1	b_4	b_6	b_{12}	b_{13}	b_{18}	*	*
001	b_2	b_5	b_7	*	b_{14}	b_{19}	*	*
010	b_3	*	b_8	*	b_{15}	*	*	*
011	*	*	b_9	*	b_{16}	*	*	*
100	*	*	b_{10}	*	b_{17}	*	*	*
101	*	*	b_{11}	*	*	*	*	*
110	*	*	*	*	*	*	*	*
111	*	*	*	*	*	*	*	*

Fig. 4.5 Microinstruction addresses for CMCU $U_8(\Gamma_A)$

7. If $L_{\max} = 1$, go to point 8, else go to point 1.
8. End.

In case of CMCU $U_8(\Gamma_A)$, this transformation can be performed as follows. In the first step, the OLC α_3 is chosen (because $L_3 = L_{\max} = 6$), and two new OLCs are constructed, namely $\alpha_3 = \langle b_6, b_7, b_8 \rangle$ and $\alpha_7 = \langle b_9, b_{10}, b_{11} \rangle$. Next, the OLC α_5 is chosen (because $L_5 = L_{\max} = 5$), and two new OLC are constructed: $\alpha_5 = \langle b_{13}, b_{14}, b_{15} \rangle$, $\alpha_8 = \langle b_{16}, b_{17} \rangle$. Now the following values are got: $L_{\max} = 3$, $R_7 = 2$, $G = 8$, $R_6 = 3$, $R_8 = 5 = R_2$. Their analysis shows that application of code sharing method can be chosen. The microinstruction addresses for CMCU $U_8(\Gamma_A)$ are shown in Fig. 4.6. They were obtained after application of procedure P_9 to the initial set of operational linear chains.

$\tau_1\tau_2\tau_3$	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8
T_1T_2	000	001	010	011	100	101	110	111
00	b_1	b_4	b_6	b_{12}	b_{13}	b_{16}	b_9	b_{16}
01	b_2	b_5	b_7	*	b_{14}	b_{17}	b_{10}	b_{17}
11	b_3	*	b_8	*	b_{15}	*	b_{11}	*
10	*	*	*	*	*	*	*	*

Fig. 4.6 Microinstruction addresses for CMCU $U_8(\Gamma_A)$ after transformation of OLC

If condition

$$L_{\max} > 1 \tag{4.17}$$

is violated, then CMCU U_8 degenerates into the Moore FSM (Fig. 4.7).

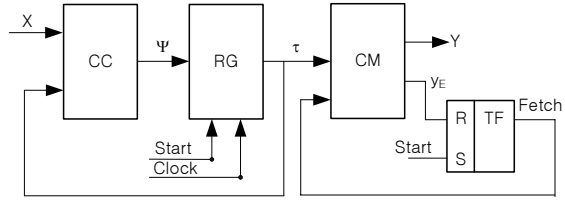


Fig. 4.7 Structural diagram of degenerated CMCU U_8

The CMCU U_8 can be considered as an intermediate variant between classic Moore FSM (Fig. 4.7), and CMCU U_2 , where automaton memory of FSM S_1 is represented by a counter (Fig. 3.5).

Let us construct the block GSA Γ_4 and mark it by states of Mealy FSM (Fig. 4.8). Its analysis shows that $R_{FB}^1(\Gamma_4) = 2$, $H_1(\Gamma_4) = 7$, $S_1(\Gamma_4) = 6$, $t_1(\Gamma_4) = 7$. Therefore, the following relations are true for CMCU U_1 and U_8 :

$$R_{FB}^1(\Gamma) \leq R_{FB}^8(\Gamma); \quad (4.18)$$

$$t_1(\Gamma) > t_8(\Gamma) = R_2; \quad (4.19)$$

$$H_1(\Gamma) < H_8(\Gamma) = H_{\max}. \quad (4.20)$$

The form of relations (4.18) and (4.20) follows from the fact that the addressing FSM S_1 of CMCU U_1 is the Mealy FSM, but this very FSM is the Moore FSM in case of CMCU U_8 . Obviously, parameter values of the addressing FSM of CMCU U_8 can be reduced by means of well-known optimization methods developed for the Moore FSM [2, 4], which should be adapted to the peculiarities of the code sharing. This optimization should be oriented towards reduction of parameters $R_{FB}^8(\Gamma)$ and $H_8(\Gamma)$.

4.2 Optimization of logic circuit of CMCU with code sharing

The codes of OLC $\alpha_g \in C^1$ for CMCU U_8 are analogues of the Moore FSM state codes. In consequence, three methods serving to reduce the hardware amount of the CMCU U_8 combinational circuit can be proposed:

- optimal encoding of OLC $\alpha_g \in C^1$;
- code transformation of OLC $\alpha_g \in C^1$ into the codes of the classes of pseudoequivalent OLC;
- transformation of initial GSA $\Gamma(U_8)$.

Optimal encoding of OLC [6] can be obtained in the same way as the codes $K(\alpha_g)$ of OLC $\alpha_g \in B_i$, where $B_i \in \Pi_c$, Π_c is a partition of the set C^1 into the classes of pseudoequivalent OLC, belonging to the same generalized interval of R_6 -dimensional Boolean space. In this case, interval $K(B_i)$ is considered as the code of the class $B_i \in \Pi_c$. The OLC encoding can be performed using some well-known

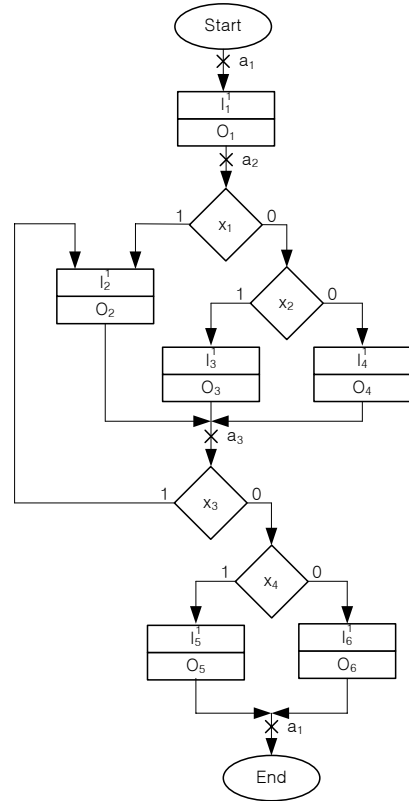


Fig. 4.8 Block presentation of GSA Γ_4

methods, such as the one described in [1], or as the algorithm ESPRESSO [8, 10]. Optimal encoding of OLC results in the CMCU U_9 of Fig. 4.9.

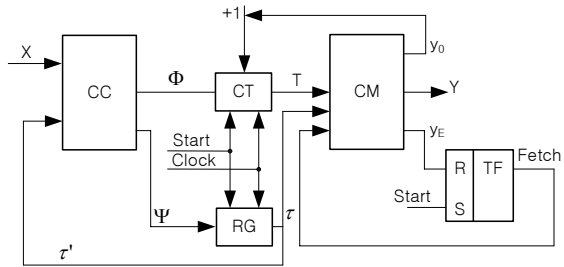


Fig. 4.9 Structural diagram of CMCU U_9

The set $\tau' \subseteq \tau$ includes variables $\tau_r \in \tau$, which are significant variables in the sense of [9], used for determination of the codes $K(B_i)$ of classes $B_i \in \Pi_c$. Comparison of CMCU U_8 and U_9 shows the identity of their operation modes, but in later case, combinational circuit CC implements the functions:

$$\Phi = \Phi(\tau', X), \quad (4.21)$$

$$\Psi = \Psi(\tau', X). \quad (4.22)$$

Synthesis of CMCU $U_9(\Gamma)$ includes the following steps:

1. Transformation of initial GSA Γ (procedure P_4).
2. Construction of OLCs for the transformed GSA $\Gamma(U_9)$ (procedure P_1).
3. Construction of partition Π_c on the set C^1 .
4. Optimal encoding of OLC $\alpha_g \in C^1$ and arbitrary encoding of operational linear chains $\alpha_g \notin C^1$.
5. Encoding of components of OLC $\alpha_g \in C$.
6. Construction of the control memory content.
7. Construction of CMCU $U_9(\Gamma)$ transition table.
8. Synthesis of CMCU logic circuit with given elements.

Let us discuss an example of the CMCU $U_9(\Gamma_4)$ synthesis, with sets $C = \{\alpha_1, \dots, \alpha_6\}$ and $C^1 = \{\alpha_1, \dots, \alpha_4\}$. The partition of set C^1 into classes of pseudo-equivalent OLCs is: $\Pi_c = \{B_1, B_2\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$. We have $R_6 = 3$ and one possible variant of OLC $\alpha_g \in C$ encoding is shown in Fig. 4.10.

$\tau_2 \tau_3$		00	01	11	10
τ_1	0	α_1	α_2	α_3	α_5
	1	*	α_4	*	α_6

Fig. 4.10 Optimal encoding of OLC for CMCU $U_9(\Gamma_4)$

All intervals occupied by codes of OLC $\alpha_g \notin C^1$ are considered as insignificant input assignments, because transitions from these OLC outputs are not included in the transition table of CMCU U_9 . Thus, in our case, we have the following codes: $K(B_1) = **0$, $K(B_2) = **1$, and, $\tau' = \{\tau_3\}$.

Encoding of the components of OLC $\alpha_g \in C$ is executed in the same way, as for CMCU U_8 . Microinstruction addresses for the CMCU $U_9(\Gamma_4)$ are shown in Fig. 4.11, from which we can get for example: $A(b_{12}) = 01001$, $A(b_{17}) = 11010$, and so on.

Construction of control memory content for CMCU U_9 is executed using the same approach as for other CMCU discussed above. It could be done simply replacing vertices $b_q \in B_1$ by corresponding sets of microoperations from the addressing table. In the case of CMCU $U_9(\Gamma_4)$ it is shown in Fig. 4.12.

It follows from Fig. 4.12, for example, that microoperations y_0, y_2, y_5 should be placed in the control memory cell addressed by 10101, and so on.

Construction of transition table for the CMCU $U_9(\Gamma_4)$ is executed using procedure P_7 , as in case of the CMCU U_4 transition table. The initial system of transition formulae (4.15) is replaced by STF (4.23):

$\tau_1\tau_2\tau_3$		000	001	010	011	100	101	110	111
		T_1T_2							
00		b_1	b_3	b_{11}	b_6	*	b_8	b_{15}	*
01		b_2	b_4	b_{12}	b_7	*	b_9	b_{16}	*
11		*	b_5	b_{13}	*	*	b_{10}	b_{17}	*
10		*	*	b_{14}	*	*	*	*	*

Fig. 4.11 Microinstruction addresses for CMCU $U_9(\Gamma_4)$

$\tau_1\tau_2\tau_3$		000	001	010	011	100	101	110	111
		T_1T_2							
00		$y_0y_1y_2$	y_0y_4	$y_0y_1y_3$	y_0y_2	*	$y_0y_1y_4$	$y_0y_2y_5$	*
01		y_3	$y_0y_1y_3$	y_0y_3	y_3	*	$y_0y_2y_5$	y_0y_3	*
11		*	*	$y_0y_1y_4$	*	*	y_4	$y_2y_4y_5$	*
10		*	*	$y_1y_5y_5$	*	*	*	*	*

Fig. 4.12 Control memory content for CMCU $U_9(\Gamma_4)$

$$\begin{aligned}
 B_1 &\rightarrow x_1I_2^1 \vee \bar{x}_1x_2I_3^1 \vee \bar{x}_1\bar{x}_2I_4^1; \\
 B_2 &\rightarrow x_3I_2^2 \vee \bar{x}_3x_4I_5^1 \vee \bar{x}_3\bar{x}_4I_6^1,
 \end{aligned}
 \tag{4.23}$$

which is used for construction of Table 4.3.

This transition table serves for construction of Boolean systems (4.21)–(4.22), which depend on terms

$$F_h = B_i^h X_h \quad (h = 1, \dots, H_9(\Gamma)),
 \tag{4.24}$$

where

$$B_i^h = \bigwedge_{r=1}^{R_6} \tau_r^{hir} \quad (i = 1, \dots, I).
 \tag{4.25}$$

Table 4.3 Transition table of CMCU $U_9(\Gamma_4)$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Ψ_h	Φ_h	h
B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Ψ_h	Φ_h	h
B_1	**0	I_2^1	00100	x_1	D_3	–	1
		I_3^1	01100	\bar{x}_1x_2	D_2D_3	–	2
		I_4^1	10100	$\bar{x}_1\bar{x}_2$	D_1D_3	–	3
B_2	**1	I_2^2	00110	x_3	D_3	D_4	4
		I_5^1	01000	\bar{x}_3x_4	D_2	–	5
		I_6^1	11000	$\bar{x}_3\bar{x}_4$	D_2D_3	–	6

Conjunction (4.25) corresponds to code $K(B_i)$ of some class $B_i \in \Pi_c$ from the line h of transition table, $l_{ir} \in \{0, 1, *\}$ is the value of bit r of the code $K(B_i)$, where $\tau_r^0 = \bar{\tau}_r$, $\tau_r^1 = \tau_r$, $\tau_r^* = 1$ ($r = 1, \dots, R_6$). The following expression can be found, for example, using Table 4.3: $D_2 = F_2 \vee F_5 \vee F_6 = \bar{\tau}_3 \bar{x}_1 x_2 \vee \tau_3 \bar{x}_3 x_4 \vee \tau_3 \bar{x}_3 \bar{x}_4$.

Synthesis of logic circuit of CMCU $U_9(\Gamma_4)$ consists in implementation of systems (4.24)–(4.25) with given PLD chips and control memory implementation using either PROM or RAM chips.

The parameters of the combinational circuit CC of CMCU U_9 are determined by the following expressions:

$$R_{FB}^9 = |\tau'| \leq R_6; \quad (4.26)$$

$$t_9(\Gamma) \leq R_2; \quad (4.27)$$

$$H_9(\Gamma) = \sum_{i=1}^I k_i H_i. \quad (4.28)$$

Here k_i is the number of intervals of an R_6 -dimensional Boolean space with codes of OLC $\alpha_g \in B_i$, where $B_i \in \Pi_c$; H_i is the number of transitions from the output of any OLC $\alpha_g \in B_i$.

In this case all parameters (4.26)–(4.28) have minimal possible values among all corresponding parameters of the CMCU U_1 – U_7 , namely: $R_{FB}^9(\Gamma_4) = 1 < R_{FB}^1(\Gamma_4) = 2$, $t_9(\Gamma_4) = 4 < R_2$, $H_9(\Gamma_4) = 6 < H_1(\Gamma) = 7$. Unfortunately, the optimal OLC encoding with $k_i = 1$ for all classes $B_i \in \Pi_c$ is not always possible. For example, if $C = \{\alpha_1, \dots, \alpha_8\}$, $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6, \alpha_7\}$, $C^1 = \{\alpha_8\}$, optimal encoding gives $k_i = 2$ for the classes $B_2, B_3 \in \Pi_c$ and the increase of parameter values for combinational CC circuit.

Let us point out, that $t_9(\Gamma) < R_2$ for the CMCU $U_9(\Gamma_4)$, but this result was obtained by chance.

Transformation of OLC codes is executed by analogy with the transformation of Moore FSM states, discussed in Section 2.4. Let each class $B_i \in \Pi_c$ correspond to a binary code $K(B_i)$ with R_4 bits, determined from the formula (3.40) as $R_4 = \lceil \log_2 I \rceil$, where $I = |\Pi_c|$. Let variables $z_r \in Z$ be used for this encoding, where $|Z| = R_4$. The GSA Γ is interpreted here by CMCU U_{10} (Fig. 4.13). The code transformer TC generates the codes of $B_i \in \Pi_c$ classes, on the base of the OLC $\alpha_g \in B_i$ codes.

The operation principle of both CMCU U_{10} and U_5 is identical. In case of CMCU U_{10} , block TC generates the functions

$$Z = Z(\tau), \quad (4.29)$$

and combinational circuit CC implements the systems:

$$\begin{aligned} \Phi &= \Phi(Z, X), \\ \Psi &= \Psi(Z, X). \end{aligned} \quad (4.30)$$

Synthesis of CMCU $U_{10}(\Gamma)$ includes the following steps:

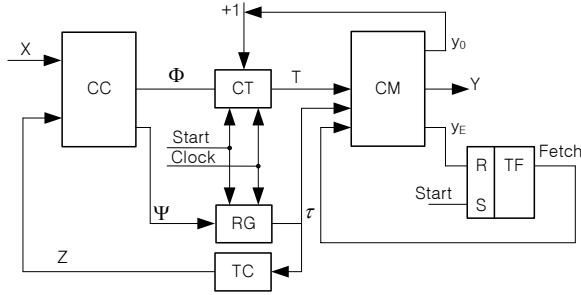


Fig. 4.13 Structural diagram of CMCU U_{10}

1. Transformation of initial GSA (procedure P_4).
2. Construction of OLC set for transformed GSA $\Gamma(U_{10})$.
3. Construction of partition Π_c of the set C^1 .
4. Encoding of OLC $\alpha_g \in C$ and of their components.
5. Encoding of classes $B_i \in \Pi_c$.
6. Construction of the control memory content.
7. Construction of CMCU transition table.
8. Construction of the table of code transformer TC.
9. Synthesis of CMCU logic circuit with given elements.

Let us discuss application of this method for synthesis of CMCU U_{10} , using the GSA $\Gamma_5(U_9)$ (Fig. 4.14).

Application of procedure P_1 to the GSA $\Gamma_5(U_9)$ results in the set $C = \{\alpha_1, \dots, \alpha_8\}$, where $\alpha_1 = \langle b_1, b_2 \rangle, I_1^1 = b_1, O_1 = b_2, L_1 = 2$; $\alpha_2 = \langle b_3, b_4, b_5 \rangle, I_2^1 = b_3, I_2^2 = O_2 = b_5, L_2 = 3$; $\alpha_3 = \langle b_6, b_7 \rangle, I_3^1 = b_6, O_3 = b_7, L_3 = 2$; $\alpha_4 = \langle b_8, b_9, b_{10} \rangle, I_4^1 = b_8, O_4 = b_{10}, L_4 = 3$; $\alpha_5 = \langle b_{11}, b_{12} \rangle, I_5^1 = b_{11}, O_5 = b_{12}, L_5 = 2$; $\alpha_6 = \langle b_{13}, b_{14}, b_{15} \rangle, I_6^1 = b_{13}, O_6 = b_{15}, L_6 = 3$; $\alpha_7 = \langle b_{16}, \dots, b_{19} \rangle, I_7^1 = b_{16}, I_7^2 = b_{18}, O_7 = b_{19}, L_7 = 4$; $\alpha_8 = \langle b_{20} \rangle, I_8^1 = O_8 = b_{20}, L_8 = 1$.

It follows from Fig. 4.15 that class B_1 corresponds to the code $K(B_1) = 000$, the class B_2 to the code $K(B_2) = 1**$ (input assignment 110 is considered as "don't care"), the class B_3 to two generalized intervals of 3-dimensional Boolean space ($R_6 = 3$) and to the codes: $K(B_3^1) = 01*$ and $K(B_3^2) = *10$. Encoding of the components of OLC $\alpha_g \in C$ results in microinstruction addresses, shown in Fig. 4.16, where we get the following sets and parameters: $\tau = \{\tau_1, \tau_2, \tau_3\}$, $R_6 = 3$, $T = \{T_1, T_2\}$, $R_7 = 2$ ($L_{\max} = L_7 = 4$).

Let us encode the classes $B_i \in \Pi_c$ in the following way: the more elements the class $B_i \in \Pi_c$ includes, the more zeros its code contains. In our case $I = 3$, $R_4 = 2$, $Z = \{z_1, z_2\}$ and classes $B_i \in \Pi_c$ have the codes: $K(B_1) = 10$, $K(B_2) = 00$, $K(B_3) = 01$. We call this approach a special encoding of equivalence classes.

As in previous cases, construction of the control memory content is executed in a trivial way. Transition table of CMCU U_{10} includes the columns B_i , $K(B_i)$, I_m^j , $A(I_m^j)$, X_h , Ψ_h , Φ_h , h , and is constructed with help of procedure P_7 . In case of CMCU $U_{10}(\Gamma_5)$, corresponding system of transition formulae is:

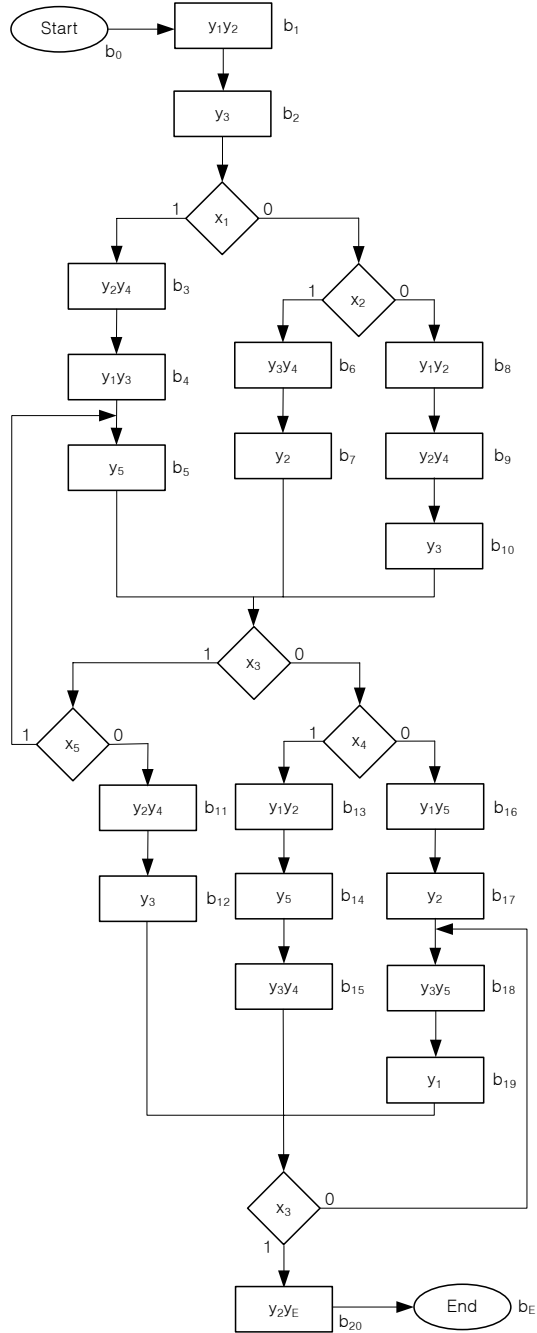


Fig. 4.14 Transformed GSA $I_3(U_9)$

Fig. 4.15 Optimal encoding of OLC for CMCU $U_{10}(\Gamma_5)$

$\tau_2\tau_3$	00	01	11	10
τ_1				
0	α_1	α_5	α_6	α_7
1	α_2	α_3	α_4	α_8

$\tau_1\tau_2\tau_3$	α_1	α_5	α_7	α_6	α_2	α_3	α_8	α_4
T_1T_2	000	001	010	011	100	101	110	111
00	b_1	b_{11}	b_{16}	b_{13}	b_3	b_6	b_{20}	b_8
01	b_2	b_{12}	b_{17}	b_{14}	b_4	b_7	*	b_9
11	*	*	b_{18}	b_{15}	b_5	*	*	b_{10}
10	*	*	b_{19}	*	*	*	*	*

Fig. 4.16 Microinstruction addresses for CMCU $U_{10}(\Gamma_5)$

$$\begin{aligned}
 B_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_3^1 \vee \bar{x}_1 \bar{x}_2 I_4^1; \\
 B_2 &\rightarrow x_3 x_5 I_2^2 \vee x_3 \bar{x}_5 I_5^1 \vee \bar{x}_3 x_4 I_6^1 \vee \bar{x}_3 \bar{x}_4 I_7^1; \\
 B_3 &\rightarrow x_3 I_8^1 \vee \bar{x}_3 I_7^2.
 \end{aligned}$$

This system corresponds to Table 4.4 with $H_{10}(\Gamma_5) = 9$ lines.

Table of code transformer TC is built by analogy with the table of OLC output codes transformer for CMCU U_6 and includes columns $\alpha_g, K(\alpha_g), B_i, K(B_i), Z_g, g$, column Z_g with variables $z_r \in Z$, equal to 1 in the code $K(B_i)$ of line g of the table ($g = 1, \dots, G_1$). In our case the table of TC includes $G_1 = 7$ lines (Table. 4.5).

Transition table of CMCU is now used to construct functions (4.30), with terms:

$$F_h = Z_i^h X_h \quad (h = 1, \dots, H_{10}(\Gamma)), \quad (4.31)$$

where Z_i^h is a conjunction of variables $z_r \in Z$, corresponding to code $K(B_i)$ of class $B_i \in \Pi_c$ from line h of the table. We can get (Table 4.4), for example, the expression $D_4 = F_4 \vee F_9 = \bar{z}_1 \bar{z}_2 x_3 x_5 \vee \bar{z}_1 z_2 \bar{x}_3$.

Table of CMCU code transformer serves to build functions (4.29), where the terms A_g are determined as:

$$A_g = \bigwedge_{r=1}^{R_6} \tau_r^{l_{gr}} \quad (g = 1, \dots, G_1), \quad (4.32)$$

and $l_{gr} \in \{0, 1\}$ is the value of the bit r of code $K(\alpha_g)$, $\tau_r^0 = \bar{\tau}_r$, $\tau_r^1 = \tau_r$ ($r = 1, \dots, R_6$). For example, we get from Table 4.5 that: $z_2 = A_5 \vee A_6 \vee A_7$.

Optimal encoding of OLC permits to reduce the number of terms in (4.29). For example, using OLC codes from Fig. 4.15, the following minimal form of the Boolean function z_2 can be found: $z_2 = \bar{\tau}_1 \tau_3 \vee \bar{\tau}_1 \tau_2$.

As in previous cases, block CC is implemented with PLD chips and the control memory with PROM or RAM chips. Block TC can be also implemented using PLD

Table 4.4 Transition table of CMCU $U_{10}(I_5)$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Ψ_h	Φ_h	h
B_1	10	I_2^1	10000	x_1	D_1	–	1
		I_3^1	10100	$\bar{x}_1 x_2$	$D_1 D_3$	–	2
		I_4^1	11100	$\bar{x}_1 \bar{x}_2$	$D_1 D_2 D_3$	–	3
B_2	00	I_2^2	10010	$x_3 x_5$	D_1	D_4	4
		I_5^1	00100	$x_3 \bar{x}_5$	D_3	–	5
		I_6^1	01100	$\bar{x}_3 x_4$	$D_2 D_3$	–	6
		I_7^1	01000	$\bar{x}_3 \bar{x}_4$	D_2	–	7
B_3	01	I_8^1	11000	x_3	$D_1 D_2$	–	8
		I_7^2	01010	\bar{x}_3	D_2	–	9

Table 4.5 Table of code transformer of CMCU $U_{10}(I_5)$

α_g	$K(\alpha_g)$	B_i	$K(B_i)$	Z_g	g
α_1	000	B_1	10	Z_1	1
α_2	100	B_2	00	–	2
α_3	101	B_2	00	–	3
α_4	111	B_2	00	–	4
α_5	001	B_3	01	Z_2	5
α_6	011	B_3	01	Z_2	6
α_7	010	B_3	01	Z_2	7

chips (if system Z includes less than 50% of all possible terms) or PROM chips (otherwise). In this case, system (4.29) includes 3 terms and therefore PLA implementation can be used. Logic circuit of CMCU $U_{10}(I_5)$ is shown in Fig. 4.17.

We find that the parameters of CMCU $U_{10}(I_5)$ and $U_1(I_5)$, are: $R_{FB}^{10}(I_5) = 2$, $t_{10}(I_5) = 4$, $H_{10}(I_5) = 9$; $R_{FB}^1(I_5) = 2$, $t_1(I_5) = 7$, $H_1(I_5) = 10$. This comparison shows the following common rules:

$$R_{FB}^{10} = R_4 = R_1; \quad (4.33)$$

$$t_{10}(\Gamma) \leq R_2; \quad (4.34)$$

$$H_{10}(\Gamma) = \sum_{i=1}^I H_i \leq H_1(\Gamma). \quad (4.35)$$

Thus, combinational circuit CC of CMCU U_{10} has minimal possible parameter values, but the use of additional block TC, consuming additional hardware, is necessary. These values could be reduced, due to optimal OLC encoding and special encoding of equivalence classes $B_i \in \Pi_c$. For example, if the classes B_2 and B_3 are interchanged (Fig. 4.15), function z_2 would be equal to: $z_2 = \tau_1$.

Transformation of GSA $\Gamma(U_8)$ consists on introduction of additional operator vertices and gives smaller number of lines in the transition table, without using additional TC block. Let symbol U_{11} stay for CMCU, based on this property. Structural diagrams of both CMCU U_8 and U_{11} are the same. As in case of CMCU U_7 , extra

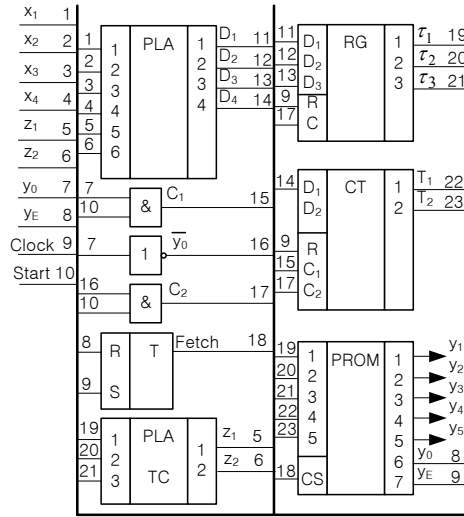


Fig. 4.17 Logic circuit of CMCU $U_{10}(I_5)$

vertices correspond to additional OLC $\alpha_g \in C^2$. Synthesis method for the CMCU $U_{11}(\Gamma)$ is an analogue of the one used for CMCU $U_7(\Gamma)$, but condition (3.52) takes now the form:

$$|C^2| = 2^{R_6} - G. \quad (4.36)$$

If the number of new vertices exceeds the value determined by (4.36), the number of bits in OLC code increases and, therefore, the number of bits for microinstruction addressing will also exceed the minimal value R_2 .

Synthesis of CMCU $U_{11}(\Gamma)$ includes the following steps:

1. Transformation of initial GSA Γ (procedure P_4) and construction of the transformed GSA $\Gamma(U_2)$.
2. Construction of the set of OLC (procedure P_1).
3. Construction of partition Π_c .
4. Construction of transformed GSA $\Gamma(U_{11})$ (procedure P_8).
5. Optimal encoding of OLC $\alpha_g \in C^1$.
6. Encoding of components of OLC $\alpha_g \in C^1 \cup C^2$.
7. Construction of control memory content.
8. Construction of transition table for CMCU.
9. Synthesis of logic circuit for CMCU.

Let us discuss application of this method for synthesis of CMCU U_{11} interpreting the GSA I_6 . This GSA is transformed into GSA $I_6(U_2)$ using procedure P_4 (Fig. 4.18).

Application of procedure P_1 to GSA $I_6(U_2)$ gives the OLC set $C = \{\alpha_1, \dots, \alpha_7\}$, where $\alpha_1 = \langle b_1, b_2 \rangle, I_1^1 = b_1, O_1 = b_2, L_1 = 2; \alpha_2 = \langle b_3, b_4, b_5 \rangle, I_2^1 = b_3, O_2 = b_5, L_2 = 3; \alpha_3 = \langle b_6, b_7 \rangle, I_3^1 = b_6, O_3 = b_7, L_3 = 2; \alpha_4 = \langle b_8, \dots, b_{11} \rangle, I_4^1 = b_8, I_4^2 = b_{10}, O_4 = b_{11}, L_4 = 4; \alpha_5 = \langle b_{12}, b_{13} \rangle, I_5^1 = b_{12}, O_5 = b_{13}, L_5 = 2; \alpha_6 = \langle b_{14}, b_{15}, b_{16} \rangle,$

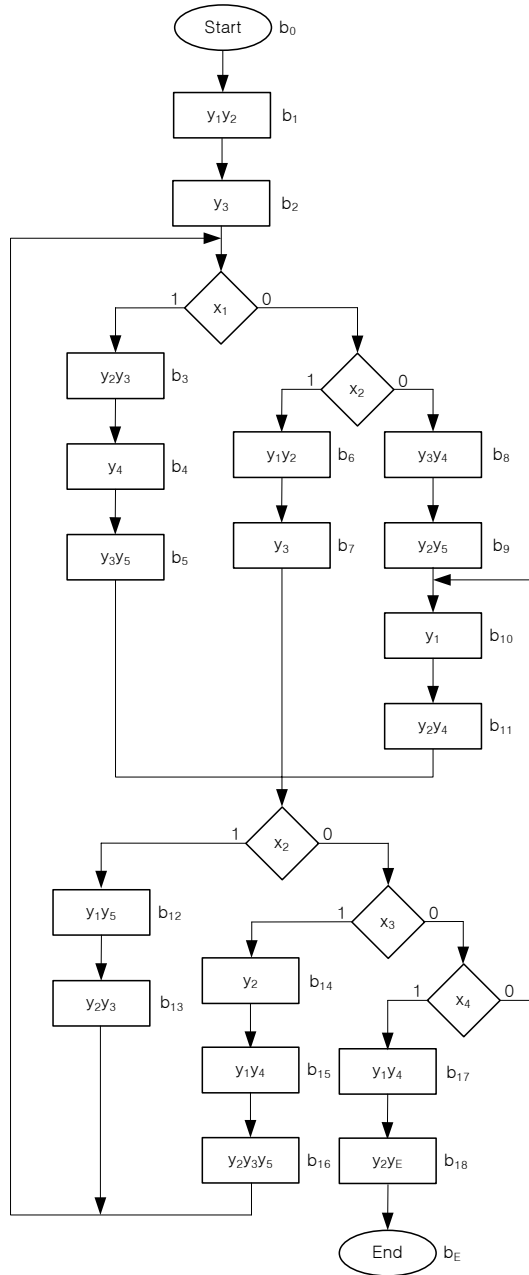


Fig. 4.18 Transformed GSA $\Gamma_6(U_2)$

$I_6^1 = b_{14}$, $O_6 = b_{16}$, $L_6 = 3$; $\alpha_7 = \langle b_{17}, b_{18} \rangle$, $I_7^1 = b_{17}$, $O_7 = b_{18}$, $L_7 = 2$. The output of the OLC α_7 is connected to the final vertex b_E , and therefore $C^1 = \{\alpha_1, \dots, \alpha_6\}$. Partition of the OLC set includes 2 classes, namely $\Pi_c = \{B_1, B_2\}$, with $B_1 = \{\alpha_1, \alpha_5, \alpha_6\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$.

Classes of partition Π_c have the following parameters: $|B_1| = 3$, $|H_1| = 3$, $\Delta H_1 = 3$, $|B_2| = 3$, $H_2 = 4$, $\Delta H_2 = 5$. It means that the transformation of GSA makes sense for both classes.

According to (4.36), the GSA $\Gamma_6(U_2)$ can be transformed by introduction of $2^3 - 7 = 1$ additional operator vertices. In other words, the queue $\gamma = \langle B_2, B_1 \rangle$ should be organized. Blocks $B_i \in \Pi_c$ are listed in the order of diminishing parameter $\Delta H_i > 0$. Thus, only the subgraph of GSA corresponding to the class $B_2 \in \Pi_c$ can be transformed. The transformed subgraph of GSA $\Gamma_6(U_2)$ is shown in Fig. 4.19. This transformation introduces the vertex b_{19} , all other connections in the GSA $\Gamma_6(U_2)$ being the same as in the GSA $\Gamma_6(U_2)$.

Further decrease of the number of transitions is possible due to optimal encoding of OLC $\alpha_g \in C^1 \cup C^2$, where the set $C^2 = \{\alpha_8\}$ with an extra OLC $\alpha_8 = \langle b_{19} \rangle$, $I_8^1 = O_8 = b_{19}$. Code $K(\alpha_7)$ corresponds to insignificant input assignment, because $\alpha_7 \notin C^1$. Outcome of optimal OLC encoding is shown in Fig. 4.20.

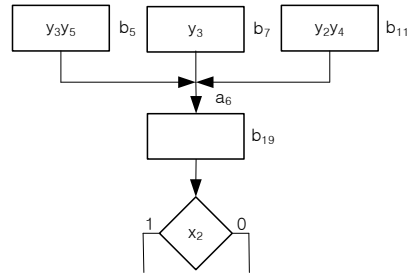


Fig. 4.19 Transformed subgraph of GSA $\Gamma_6(U_{11})$

$\tau_2 \tau_3$	00	01	11	10
0	α_1	α_5	α_6	α_7
1	α_2	α_3	α_4	α_8

Fig. 4.20 Optimal encoding of OLC for CMCU $U_{11}(\Gamma_6)$

Introduction of the OLC α_8 results in new partition $\Pi_c = \{B_1, B_2, B_3\}$, where $B_3 = \{\alpha_8\}$. It follows from Fig. 4.20 that $K(B_1) = 0**$ (code $K(\alpha_7)$ is considered as insignificant input assignment), $K(B_3) = *10$, block B_2 belongs to the two generalized intervals of code space and thus is split into two blocks, namely: B_2^1 coded as $K(B_2^1) = 10*$ and B_2^2 coded as $K(B_2^2) = 1*1$.

Encoding of OLC components is executed in a standard way and the result is shown in Fig. 4.21.

Construction of the transition table is the same as in case of CMCU U_9 . After transformation of the initial SFT the following system of transition formulae is obtained:

$$\begin{aligned}
 B_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_3^1 \vee \bar{x}_1 \bar{x}_2 I_4^1; \\
 B_2^1 &\rightarrow I_8^1; \\
 B_2^2 &\rightarrow I_8^1; \\
 B_3 &\rightarrow x_2 I_5^1 \vee \bar{x}_2 x_3 I_6^1 \vee \bar{x}_2 \bar{x}_3 x_4 I_7^1 \vee \bar{x}_2 \bar{x}_3 \bar{x}_4 I_7^2.
 \end{aligned} \tag{4.37}$$

$\tau_1 \tau_2 \tau_3$	α_1	α_5	α_7	α_6	α_2	α_3	α_8	α_4
$T_1 T_2$	000	001	010	011	100	101	110	111
00	b_1	b_{12}	b_{17}	b_{14}	b_3	b_6	b_{19}	b_8
01	b_2	b_{13}	b_{18}	b_{15}	b_4	b_7	*	b_9
11	*	*	*	b_{16}	b_5	*	*	b_{10}
10	*	*	*	*	*	*	*	b_{11}

Fig. 4.21 Microinstruction addresses for CMCU $U_{11}(I_6)$

System (4.37) corresponds to the CMCU $U_{11}(I_6)$ transition table with $H_{11}(I_6) = 9$ lines corresponding to the number of terms in the system of transition formulae (4.37). In case of CMCU $U_8(I_6)$, the transition table has $H_8(I_6) = 22$ lines, but in case of CMCU $U_1(I_6)$ it has only $H_1(I_6) = 8$ lines. Therefore, the combined application of GSA transformation and optimal OLC encoding allows reducing the number of lines in the CMCU $U_{11}(I_6)$ transition table approximately to the same value as in case of the structure table of addressing FSM S_1 in CMCU $U_1(I)$.

Let us point out, that extra vertices can be treated as additional components of OLC, available already before the transformation. For example, vertex b_{19} (Fig. 4.19) can be treated as output of some OLC $\alpha_g \in B_2$. Introduction of vertex b_{19} , either in the OLC α_2 or in the OLC α_3 , does not lead to higher L_{\max} , but introduction of this vertex into the OLC α_4 increases this parameter and now $R_7 = 3$. Advantage of this approach consists on preservation of the number of OLC in comparison with the initial GSA. The analysis shows that the following rules should be used during transformation of the initial GSA (procedure P_{10}):

- if the introduction of additional vertex does not increase the value of R_7 , this vertex is added in some OLC, already available;
- if the introduction of additional vertex increases the value of R_7 and introduction of additional OLC does not increase the value of R_6 , then this vertex is treated as the additional OLC;
- if the introduction of additional vertex increases the values of both R_6 and R_7 , the vertex is not added.

Let us take the case when vertex b_{19} is inserted into the OLC α_2 and we have the new OLC $\alpha_2 = \langle b_3, b_4, b_5, b_{19} \rangle$ with $I_2^1 = b_3$, $I_2^2 = O_2 = b_{19}$. Now, new

partition $\Pi_c = \{B_1, B_2, B_3\}$ can be formed, where $B_1 = \{\alpha_1, \alpha_5, \alpha_6\}$, $B_2 = \{\alpha_2\}$, $B_3 = \{\alpha_3, \alpha_4\}$, and the transition table is represented by the following STF:

$$\begin{aligned} B_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_3^1 \vee \bar{x}_1 \bar{x}_2 I_4^1; \\ B_2 &\rightarrow x_2 I_5^1 \vee \bar{x}_2 x_3 I_6^1 \vee \bar{x}_2 \bar{x}_3 x_4 I_7^1 \vee \bar{x}_2 \bar{x}_3 \bar{x}_4 I_7^2; \\ B_3 &\rightarrow I_2^2. \end{aligned}$$

This system corresponds to the transition table having 8 lines, the same as the structure table of CMCU $U_1(\Gamma_6)$. Control unit U_{11} is characterized by the following parameters:

$$R_{FB}^{11} = R_6; \tag{4.38}$$

$$t_{11}(\Gamma) \leq R_2; \tag{4.39}$$

$$H_{11}(\Gamma) = \sum_{i=1}^{I_1} k_i H_i, \tag{4.40}$$

where I_1 is the new value of the cardinal number of partition Π_c after introduction of additional vertices. Let us point out, that condition (4.38) holds only when procedure P_{10} is used. Otherwise the number of feedback variables can exceed the value of R_6 .

4.3 Synthesis of CMCU with elementary operational linear chains

The method of synthesis of CMCU with elementary OLC is proposed in [3, 5]. Operational linear chain is called an elementary OLC (EOLC) if it has only one single input. Structural diagram of CMCU U_{12} is shown in Fig. 4.22 and represents basic structure of the CMCU with EOLC.

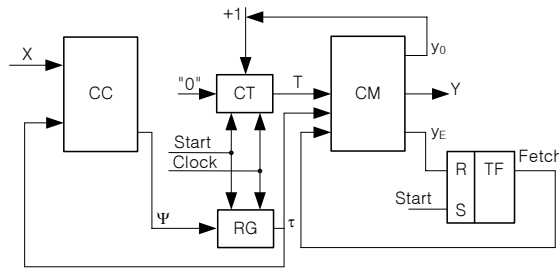


Fig. 4.22 Structural diagram of CMCU U_{12}

Let C_E be a set of elementary OLC of GSA Γ , including G_E elements. Let us encode each EOLC by a binary code with

$$R_9 = \lceil \log_2 G_E \rceil \quad (4.41)$$

bits and use variables $\tau_r \in \tau$ for this encoding, where $|\tau| = R_9$. Let us encode components of OLC by binary codes $K(b_q)$ with R_7 bits and use variables $T_r \in T$, where $|T| = R_7$. Let us represent the address $A(b_q)$ of microinstruction corresponding to vertex $b_q \in D^8$ ($g = 1, \dots, G_E$) in the form (4.5). In this case CMCU $U_{12}(\Gamma)$ operates in the following manner.

Pulse "Start" initiates loading of zero codes into register RG and counter CT. Simultaneously, flip-flop TF is set up and allows fetching microinstructions from the control memory CM. If contents of RG and CT form an address $A(b_q)$, where $b_q \neq O_g$ ($g = 1, \dots, G_E$), additional variable y_0 is generated together with microoperations from the set $Y(b_q)$. If $y_0 = 1$, the pulse "Clock" causes increment of CT, which corresponds to the unconditional jump (4.4). If contents of RG and CT form an address $A(b_q)$, where $b_q = O_g$ ($g = 1, \dots, G_E$), variable y_0 is not generated. In this case pulse "Clock" causes reset of CT, corresponding to an input code of some EOLC. At the same time functions (4.7) change the content of RG. If microinstruction with $y_E = 1$ is fetched from the control memory, flip-flop TF is reset and operation of CMCU terminated.

Analysis of CMCU U_{12} shows that its combinational circuit CC has R_9 outputs, which is the minimum for this parameter. The corresponding microinstruction address has

$$R_{10} = R_9 + R_7 \quad (4.42)$$

bits and if condition

$$R_{10} = R_2 \quad (4.43)$$

holds, expression (4.42) gives the smallest possible number of address bits. Application of this method has no sense if condition

$$R_{10} > R_2 \quad (4.44)$$

is satisfied, because in this case the size of control memory exceeds the minimum V_{\min} , determined by (4.14).

Synthesis of CMCU $U_{12}(\Gamma)$ is an analogue of the one used for CMCU $U_8(\Gamma)$, but the set C_E is used in former case instead of the set of OLC C . Synthesis of the CMCU $U_{12}(\Gamma)$ includes the following steps:

1. Preliminary transformation of initial GSA (procedure P_4).
2. Construction of the set of elementary OLC C_E for GSA $\Gamma(U_{12})$.
3. Encoding of elementary OLC.
4. Encoding of components for elementary OLC.
5. Construction of the control memory content.
6. Construction of transition table for CMCU $U_{12}(\Gamma)$.
7. Synthesis of logic circuit of CMCU.

Construction of set C_E requires some modification of procedure P_1 , leading to procedure P_{11} , and uses the set of main inputs $IM_E(\Gamma)$ of elementary OLC as a base for construction of the chains. Main input of EOLC is an operator vertex, with the

input connected either to the output of initial or conditional vertices or with outputs of more operator vertices. Procedure P_{11} includes the following steps:

1. Construction of the set $IM_E(\Gamma)$.
2. Put $g := 1$
3. Choice of the vertex $b_q \in B_1$ with the least value of subscript in the set $IM_E(\Gamma)$. Removing of this vertex from the set $IM_E(\Gamma)$.
4. Insertion of the vertex b_q into the set D^g , used as a base vertex b_N of elementary OLC α_g .
5. Choice of some vertex b_t , such that $\langle b_N, b_t \rangle \in E$.
6. If $b_t \in B_2$ or $b_t = b_E$, or $b_t \in IM_E(\Gamma)$, go to point 8, else the vertex b_t should be included in the constructed OLC α_g after the vertex b_N .
7. Fix vertex b_t as a new base vertex b_N of EOLC α_g and go to point 5.
8. If $IM_E(\Gamma) = \emptyset$, go to point 10.
9. Put $g := g + 1$ and go to point 3.
10. End.

All other steps of synthesis procedure for CMCU $U_{12}(\Gamma)$ are executed by analogy with corresponding steps given already for CMCU $U_8(\Gamma)$. Let us consider application of this method for synthesis of CMCU U_{12} , using the transformed GSA $\Gamma_7(U_2)$ shown in Fig. 4.23.

Construction of the set of elementary OLC. In this case we have the set of inputs $IM_E(\Gamma) = \{b_1, b_3, b_4, b_7, b_{10}, b_{11}b_{14}, b_{18}\}$, therefore, the set of EOLC includes $G_E = 8$ elements. Application of procedure P_{11} results in the set $C_E = \{\alpha_1, \dots, \alpha_8\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1 = b_1$, $O_1 = b_2$, $L_1 = 2$; $\alpha_2 = \langle b_3 \rangle$, $I_2 = O_2 = b_3$, $L_2 = 1$; $\alpha_3 = \langle b_4, b_5, b_6 \rangle$, $I_3 = b_4$, $O_3 = b_6$, $L_3 = 3$; $\alpha_4 = \langle b_7, b_8, b_9 \rangle$, $I_4 = b_7$, $O_4 = b_9$, $L_4 = 3$; $\alpha_5 = \langle b_{10} \rangle$, $I_5 = O_5 = b_{10}$, $L_5 = 1$; $\alpha_6 = \langle b_{11}, b_{12}, b_{13} \rangle$, $I_6 = b_{11}$, $O_6 = b_{13}$, $L_6 = 3$; $\alpha_7 = \langle b_{14}, \dots, b_{17} \rangle$, $I_7 = b_{14}$, $O_7 = b_{17}$, $L_7 = 4$; $\alpha_8 = \langle b_{18} \rangle$, $I_8 = O_8 = b_{18}$, $L_8 = 1$. Let us point out, that inputs of EOLC have only the subscripts.

Encoding of EOLC is executed in a trivial way. In case of the CMCU $U_{12}(\Gamma_7)$, $R_9 = 3$ bits is sufficient to encode EOLC, thus $\tau = \{\tau_1, \tau_2, \tau_3\}$. Let the EOLC have the following codes: $K(\alpha_1) = 000, \dots, K(\alpha_8) = 111$.

Encoding of components is executed using the same approach as in case of CMCU U_8 . In this case it can be found that $L_{\max} = 4$, $R_7 = 2$ and $T = \{T_1, T_2\}$. Microinstruction addresses for the CMCU $U_{12}(\Gamma_7)$ are shown in Fig. 4.24.

Construction of control memory content is reduced here to the replacement of vertices $b_q \in B_1$ by microoperations taken from corresponding set $Y(b_q)$. This step is executed in a trivial way.

Construction of transition table of CMCU U_{12} is executed by analogy with corresponding synthesis step for CMCU U_8 . The transition formulae are constructed only for EOLC $\alpha_g \in C_E^1$, where $C_E^1 \subseteq C_E$ is a set of EOLC, such that their outputs do not include variable y_E . In case of CMCU $U_{12}(\Gamma_7)$ we have $C_E^1 = \{\alpha_1, \dots, \alpha_7\}$ and the system of transition formulae includes $|C_E^1| = 7$ expressions:

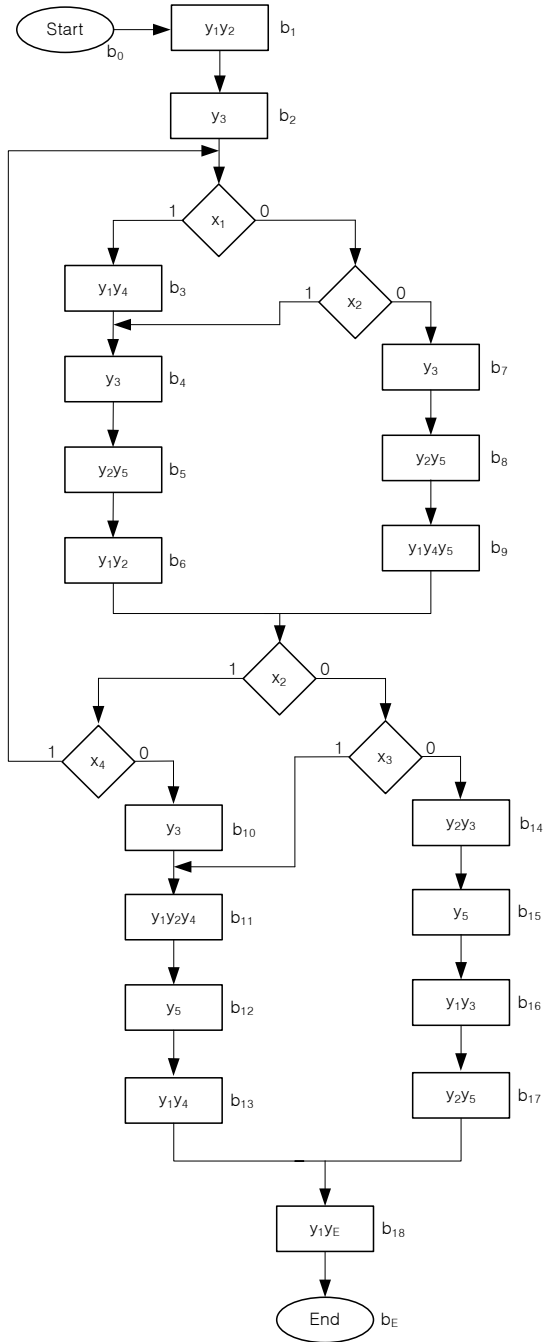


Fig. 4.23 Transformed GSA $F_7(U_2)$

$\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110	111
00	b_1	b_3	b_4	b_7	b_{10}	b_{11}	b_{14}	b_{18}
01	b_2	*	b_5	b_8	*	b_{12}	b_{15}	*
11	*	*	b_6	b_9	*	b_{13}	b_{16}	*
10	*	*	*	*	*	*	b_{17}	*

Fig. 4.24 Microinstruction addresses for CMCU $U_{12}(I_7)$

$$\begin{aligned}
 \alpha_1 &\rightarrow x_1 I_2 \vee \bar{x}_1 x_2 I_3 \vee \bar{x}_1 \bar{x}_2 I_4; \\
 \alpha_2 &\rightarrow I_3; \\
 \alpha_3 &\rightarrow x_2 x_4 I_3 \vee x_2 \bar{x}_4 I_5 \vee \bar{x}_2 x_3 I_6 \vee \bar{x}_2 \bar{x}_3 I_7; \\
 \alpha_4 &\rightarrow x_2 x_4 I_3 \vee x_2 \bar{x}_4 I_5 \vee \bar{x}_2 x_3 I_6 \vee \bar{x}_2 \bar{x}_3 I_7; \\
 \alpha_5 &\rightarrow I_6; \\
 \alpha_6 &\rightarrow I_8; \\
 \alpha_7 &\rightarrow I_8.
 \end{aligned} \tag{4.45}$$

System (4.45) corresponds to the transition table of CMCU $U_{12}(I_7)$, including $H_{12}(I_7) = 15$ lines (Table 4.6).

Table 4.6 Table of code transformer TC for CMCU $U_{20}(I_{10})$

α_g	$K(\alpha_g)$	I_m^j	$A(I_m^j)$	X_h	Ψ_h	h
α_1	000	I_2	00100	x_1	D_3	1
		I_3	01000	$\bar{x}_1 x_2$	D_2	2
		I_4	01100	$\bar{x}_1 \bar{x}_2$	$D_2 D_3$	3
α_2	001	I_3	01000	1	D_2	4
α_3	010	I_3	01000	$x_2 x_4$	D_2	5
		I_5	10000	$x_2 \bar{x}_4$	D_1	6
		I_6	10100	$\bar{x}_2 x_3$	$D_1 D_3$	7
		I_7	11000	$\bar{x}_2 \bar{x}_3$	$D_1 D_2$	8
α_4	011	I_3	01000	$x_2 x_4$	D_2	9
		I_5	10000	$x_2 \bar{x}_4$	D_1	10
		I_6	10100	$\bar{x}_2 x_3$	$D_1 D_3$	11
		I_7	11000	$\bar{x}_2 \bar{x}_3$	$D_1 D_2$	12
α_5	100	I_6	10100	1	$D_1 D_3$	13
α_6	101	I_8	11100	1	$D_1 D_2 D_3$	14
α_7	110	I_8	11100	1	$D_1 D_2 D_3$	15

In contrast to the transition table of CMCU U_8 , this table of CMCU U_{12} does not include the column Φ_h , because all components corresponding to inputs of EOLC have only zero codes. This transition table is the base to construct functions of system (4.7), the terms of which are determined by (4.16). For example, the following

Boolean expression can be extracted from Table 4.6: $D_1 = F_6 \vee F_7 \vee F_8 \vee F_{10} \vee \dots \vee F_{15} = \bar{\tau}_1 \tau_2 \bar{\tau}_3 x_2 \bar{x}_4 \vee \dots \vee \tau_1 \tau_2 \bar{\tau}_3$.

Synthesis of logic circuit of CMCU U_{12} is reduced to implementation of system (4.7) using PLD chips and systems (4.8)–(4.9) using either PROM or RAM chips. Logic circuit of the CMCU $U_{12}(I_7)$ is shown in Fig. 4.25.

Let us find the values of some parameters of the CMCU $U_8(I_7)$: $G = 6$, $R_{FB}^8(\Gamma) = 3$; $t_8(I_7) = 3 + 2 = 5$; $H_8(I_7) = 12$. It is helpful to use the following general relations for comparison of CMCU U_8 and U_{12} :

$$R_{FB}^8 = R_{FB}^{12}; \quad (4.46)$$

$$t_8(\Gamma) > t_{12}(\Gamma); \quad (4.47)$$

$$H_8(\Gamma) = H_{12}(\Gamma). \quad (4.48)$$

Inequality (4.47) turns into equality if the following condition

$$\lceil \log_2 G \rceil = \lceil \log_2 G_E \rceil \quad (4.49)$$

is satisfied. Parameter $t_{12}(\Gamma)$ has the minimal possible value among all CMCU considered already. The number of transitions for CMCU U_{12} can be estimated as

$$H_{12}(\Gamma) = H_8(\Gamma) + (G_E - G) + J_E, \quad (4.50)$$

where J_E is a variable equal to the number of OLC $\alpha_g \in C^1$ with $L_g > 1$. The second term of expression (4.50) determines the number of new members corresponding to unconditional jumps connected with transformation of initial OLC into EOLC. In our case it can be found that $G_E - G = 2$, $J_E = 1$. It means that $H_{12}(\Gamma) - H_8(\Gamma) = 3$, as was already proved by previous calculations.

As in case of CMCU U_8 , the number of feedback signals and lines of transition table of CMCU U_{12} can be reduced due to application of Moore FSM optimization methods adapted to peculiarities of CMCU with elementary operational linear chains.

4.4 Logic circuit optimization for CMCU with elementary OLC

In CMCU U_{12} the codes of OLC $\alpha_g \in C_E$ are analogues of state codes of Moore FSM S_1 . Therefore, three methods can be proposed for reducing the hardware amount of CMCU U_{12} logic circuit:

- optimal encoding of elementary OLC $\alpha_g \in C_E^1$;
- transformation of codes of EOLC $\alpha_g \in C_E^1$ into codes of the classes of pseudoequivalent EOLC;
- transformation of graph-scheme of algorithm $\Gamma(U_{12})$.

Optimal encoding of EOLC requires finding a partition Π_E of the set EOLC C_E^1 into the classes of pseudoequivalent EOLC. As in case of pseudoequivalent OLC,

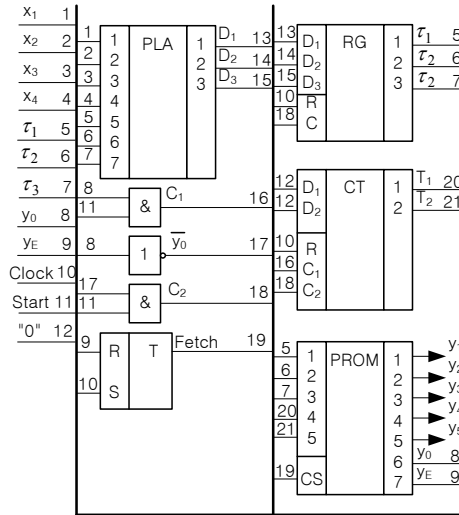


Fig. 4.25 Logic circuit of CMCU $U_{12}(I_7)$

elementary OLC $\alpha_i, \alpha_j \in C_E^1$ are called pseudoequivalent EOLC, if their outputs are connected with input of the same vertex of GSA Γ . In case of GSA Γ_7 , where $\Pi_c = \{B_1, \dots, B_5\}$, partition $\Pi_c = \{B_1, \dots, B_5\}$ can be found, for which $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2\}$, $B_3 = \{\alpha_3, \alpha_4\}$, $B_4 = \{\alpha_5\}$, $B_5 = \{\alpha_6, \alpha_7\}$. The encoding is executed in such a manner, that codes $K(\alpha_g)$ of EOLC $\alpha_g \in B_i$ belong to the same generalized interval of a R_9 - dimensional Boolean space. This approach results in the CMCU U_{13} of Fig. 4.26.

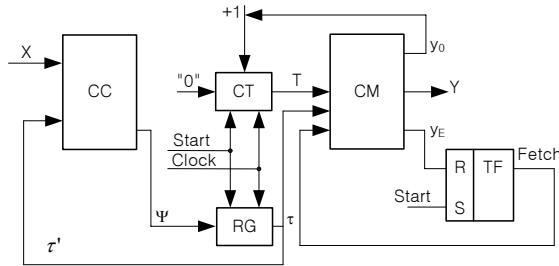


Fig. 4.26 Structural diagram of CMCU U_{13}

As in case of CMCU U_9 , the set $\tau' \subseteq \tau$ includes variables $\tau_r \in \tau$, which are significant for codes $K(B_i)$ of classes $B_i \in \Pi_E$. Combinational circuit CC of CMCU U_{13} generates functions (4.22). Synthesis methods used for both CMCU U_9 and U_{13} are identical, but the later uses elementary OLC instead of OLC and partition Π_E instead of partition Π_c . Outcome of optimal encoding of EOLC for the CMCU $U_{13}(I_7)$ is shown in Fig. 4.27.

$\tau_2 \tau_3$		00	01	11	10
τ_1	0	α_1	α_2	α_3	α_6
	1	α_8	α_5	α_4	α_7

Fig. 4.27 Optimal encoding of EOLC for CMCU $U_{13}(F_7)$

The code $K(\alpha_8) = 100$ is treated as insignificant input assignment and therefore classes $B_i \in \Pi_E$ are determined by the codes: $K(B_1) = *00$, $K(B_2) = 001$, $K(B_3) = *11$, $K(B_4) = 10*$, $K(B_5) = *10$. It means that in this case equality $\tau = \tau'$ holds and the number of feedback variables in combinational circuit CC of the CMCU $U_{13}(F_7)$ is not decreased due to optimal encoding of elementary OLC.

Transition table of CMCU $U_{13}(F_7)$ is constructed using procedure P_7 . It leads to replacement of initial system (4.45) by the following system of transition formulae:

$$\begin{aligned}
 B_1 &\rightarrow x_1 I_2 \vee \bar{x}_1 x_2 I_3 \vee \bar{x}_1 \bar{x}_2 I_4; \\
 B_2 &\rightarrow I_3; \\
 B_3 &\rightarrow x_2 x_4 I_3 \vee x_2 \bar{x}_4 I_5 \vee \bar{x}_2 x_3 I_6 \vee \bar{x}_2 \bar{x}_3 I_7; \\
 B_4 &\rightarrow I_6; \\
 B_5 &\rightarrow I_8.
 \end{aligned} \tag{4.51}$$

System (4.51) corresponds to transition table of CMCU $U_{13}(F_7)$, which includes $H_{13}(F_7) = 10$ lines (Table 4.7).

Table 4.7 Table of code transformer TC for CMCU $U_{20}(F_{10})$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Ψ_h	h
B_1	*00	I_2	00100	x_1	D_3	1
		I_3	01100	$\bar{x}_1 x_2$	$D_2 D_3$	2
		I_4	11100	$\bar{x}_1 \bar{x}_2$	$D_1 D_2 D_3$	3
B_2	001	I_3	01100	1	$D_2 D_3$	4
B_3	*11	I_3	01100	$x_2 x_4$	$D_2 D_3$	5
		I_5	10100	$x_2 \bar{x}_4$	$D_1 D_3$	6
		I_6	01000	$\bar{x}_2 x_3$	D_2	7
		I_7	11000	$\bar{x}_2 \bar{x}_3$	$D_1 D_2$	8
B_4	10*	I_6	01000	1	D_2	9
B_5	*10	I_8	10000	1	D_1	10

Functions (4.22) depend on terms (4.24). For example, the following expression for the first input memory function can be obtained from Table 4.7: $D_1 = F_3 \vee F_6 \vee F_8 \vee F_9 = \bar{\tau}_2 \bar{\tau}_3 \bar{x}_1 \bar{x}_2 \vee \tau_1 \tau_2 x_2 \bar{x}_4 \vee \tau_2 \tau_3 \bar{x}_2 \bar{x}_3 \vee \tau_2 \bar{x}_3$.

Parameters of combinational circuit CC, determined for CMCU $U_{13}(F_7)$, satisfy the relations:

$$R_{FB}^{13} = |\tau'| \leq R_9; \tag{4.52}$$

	$z_2 z_3$	00	01	11	10
z_1	0	B_3	B_5	B_2	B_1
	1	B_4	*	*	*

Fig. 4.29 Encoding of classes for CMCU $U_{14}(F_7)$

Transition table for the CMCU $U_{14}(F_7)$ is built using system (4.51) and includes $H_{14}(F_7) = 10$ lines (Table 4.8). It allows getting functions (4.30), depending on the terms

$$F_h = Z_i^h X_h \quad (h = 1, \dots, H_{14}(F)), \tag{4.56}$$

where conjunction Z_i^h has the form:

$$Z_i^h = \bigwedge_{r=1}^{R_{11}} z_r^{l_{ir}} \quad (i = 1, \dots, I_E). \tag{4.57}$$

Here $l_{ir} \in \{0, 1, *\}$ is the value of bit r of the code $K(B_i)$ from line h of the transition table, $z_r^0 = \bar{z}_r$, $z_r^1 = z_r$, $z_r^* = 1$ ($r = 1, \dots, R_{11}$). For example, the following equation can be found from Table 4.8: $D_1 = F_3 \vee F_6 \vee F_8 \vee F_{10} = z_2 z_3 \bar{x}_1 \bar{x}_2 \vee \dots \vee \bar{z}_2 z_3$.

Table 4.8 Transition table of CMCU $U_{14}(F_7)$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Ψ_h	h
B_1	*10	I_2	00100	x_1	D_3	1
		I_3	01100	$\bar{x}_1 x_2$	$D_2 D_3$	2
		I_4	11100	$\bar{x}_1 \bar{x}_2$	$D_1 D_2 D_3$	3
B_2	*11	I_3	01100	1	$D_2 D_3$	4
B_3	000	I_3	01100	$x_2 x_4$	$D_2 D_3$	5
		I_5	10100	$x_2 \bar{x}_4$	$D_1 D_3$	6
		I_6	01000	$\bar{x}_2 x_3$	D_2	7
		I_7	11000	$\bar{x}_2 \bar{x}_3$	$D_1 D_2$	8
		I_8	10000	1	D_1	10
B_4	1**	I_6	01000	1	D_2	9
B_5	*01	I_8	10000	1	D_1	10

Table of code transformer TC is constructed by analogy to the corresponding table of CMCU U_{10} . In this case, table of TC has $G_1 = 7$ lines (Table 4.9) and is used to construct system (4.29), with terms determined by analogy with (4.32):

$$A_g = \bigwedge_{r=1}^{R_9} \tau_r^{l_{gr}} \quad (g = 1, \dots, G_1). \tag{4.58}$$

For example, from Table 4.9 we get: $z_1 = \tau_1 \bar{\tau}_2$ (taking into account the insignificant input assignment 100) and $z_2 = \bar{\tau}_1 \bar{\tau}_2$; $z_3 = \bar{\tau}_1 \bar{\tau}_2 \tau_3 \vee \tau_2 \bar{\tau}_3$. We point out that functions of this system may in general include up to $G_1 = 7$ terms, each term with up to $R_9 = 3$ literals. Thus, the disjunctive normal forms of functions (4.29) include

Table 4.9 Table of code transformer of CMCU $U_{14}(I_7)$

α_g	$K(\alpha_g)$	B_i	$K(B_i)$	Z_g	g
α_1	000	B_1	*10	Z_2	1
α_2	001	B_2	*11	Z_2Z_3	2
α_3	011	B_3	000	–	3
α_4	111	B_3	000	–	4
α_5	101	B_4	1**	Z_1	5
α_6	010	B_5	*10	Z_3	6
α_7	110	B_5	*10	Z_3	7

up to $G_1 \cdot R_9 = 21$ literals. In our case this value is reduced to 7, due to optimal encoding of both elementary OLCs and their classes.

Comparison of Tables 4.7 and 4.9 shows that the use of TC has no sense in this particular case, because it does not lead to decrease neither the number of lines of transition table nor the length of code $K(B_i)$. Therefore, use of the CMCU $U_{13}(I_7)$ gives the logic circuit with best parameters, in case of interpretation GSA I_7 using the CMCU model. In general case, CMCU U_{14} has the following parameters:

$$R_{FB}^{14} = R_{11} \leq R_9; \quad (4.59)$$

$$t_{14}(\Gamma) = R_{11} \leq R_9; \quad (4.60)$$

$$H_{14}(\Gamma) = \sum_{i=1}^{I_E} H_i \leq H_{13}(\Gamma). \quad (4.61)$$

Transformation of GSA $\Gamma(U_8)$ is reduced to insertion of additional operator vertices, corresponding to elementary OLC $\alpha_g \in C_E^2$. The number of lines in the transition table can be reduced without using of TC block. This approach leads to the CMCU U_{15} , having similar structure, as CMCU U_{12} . Synthesis of CMCU U_{15} includes the following steps:

1. Transformation of initial GSA Γ (procedure P_4).
2. Construction of the set EOLC C_E (procedure P_{11}).
3. Construction of partition Π_E for the set C_E^1 .
4. Transformation of GSA $\Gamma(U_2)$: procedure P_8 .
5. Optimal encoding of EOLC $\alpha_g \in C_E \cup C_E^2$.
6. Encoding of components of EOLC $\alpha_g \in C_E \cup C_E^2$.
7. Construction of the control memory content.
8. Construction of transition table of CMCU.
9. Synthesis of logic circuit of CMCU.

Let us apply this approach to logic circuit of CMCU $U_{15}(I_8)$, where the GSA $I_8(U_2)$ is shown in Fig. 4.30.

Application of procedure P_{11} gives the set $C_E = \{\alpha_1, \dots, \alpha_7\}$, where $\alpha_1 = \langle b_1, b_2 \rangle, I_1 = b_1, O_1 = b_2, L_1 = 2; \alpha_2 = \langle b_3, \dots, b_6 \rangle, I_2 = b_3, O_2 = b_6, L_2 = 4; \alpha_3 = \langle b_7, b_8 \rangle, I_3 = b_7, O_3 = b_8, L_3 = 2; \alpha_4 = \langle b_9, \dots, b_{12} \rangle, I_4 = b_9, O_4 = b_{12}, L_4 = 4;$

$\alpha_5 = \langle b_{13}, b_{14}, b_{15} \rangle, I_5 = b_{13}, O_5 = b_{15}, L_5 = 3; \alpha_6 = \langle b_{16}, b_{17} \rangle, I_6 = b_{16}, O_6 = b_{17}, L_6 = 2; \alpha_7 = \langle b_{18}, b_{19} \rangle, I_7 = b_{18}, O_7 = b_{19}, L_7 = 2$. Let us point out that the set $C_E^1 = \{\alpha_1, \dots, \alpha_6\}$ and partition $\Pi_E = \{B_1, B_2\}, B_1 = \{\alpha_1\}, B_2 = \{\alpha_1, \dots, \alpha_6\}$, is obtained in this case.

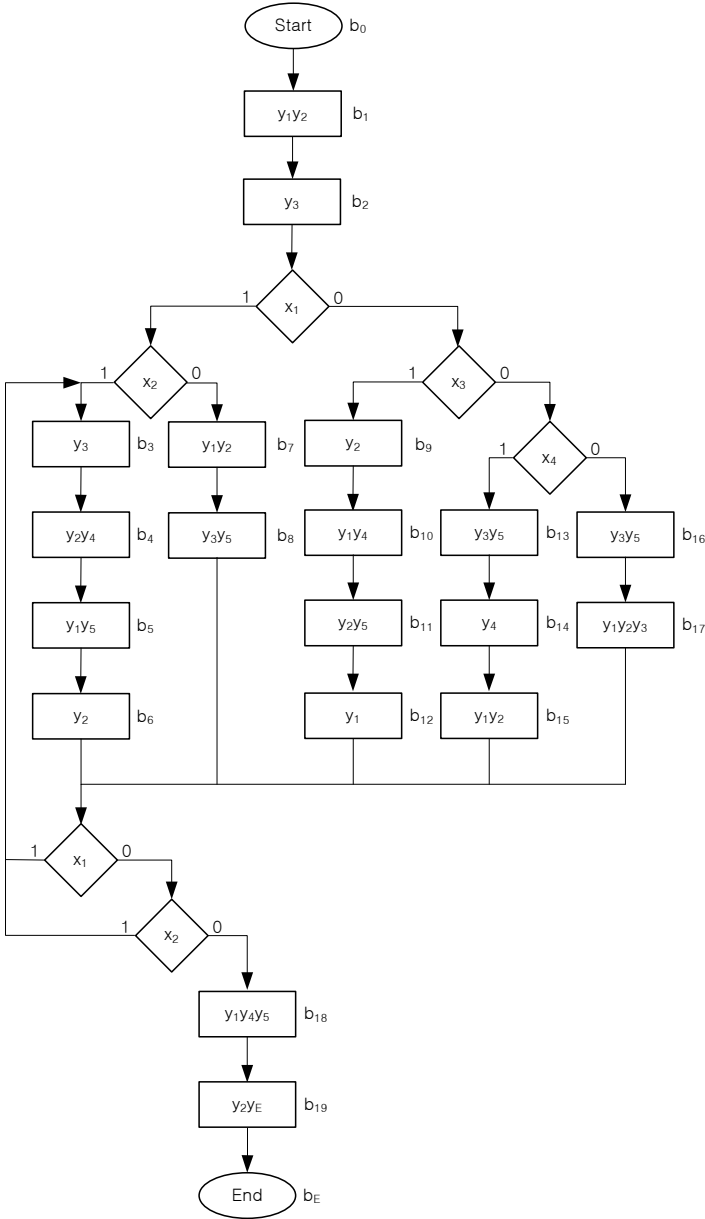


Fig. 4.30 Transformed GSA $\Gamma_8(U_2)$

Analysis of class B_2 shows that $|B_2| = 5, H_2 = 2, \Delta H_2 = 7$. It means that transformation of the GSA $\Gamma_8(U_2)$ makes sense.

Application of procedure P_8 results in appearance of the vertex b_{20} , corresponding to EOLC α_8 , where $I_8 = O_8 = b_{20}, L_8 = 1$ (Fig. 4.31). It means that the set $C_E^2 = \{\alpha_8\}$ should be constructed.

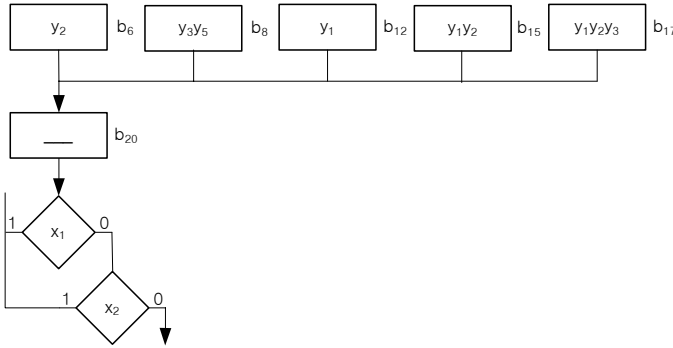


Fig. 4.31 Subgraph of transformed GSA $\Gamma_8(U_{15})$

Outcome of the optimal encoding of elementary OLC $\alpha_g \in C_E \cup C_E^2$ is shown in Fig. 4.32.

$\tau_2 \tau_3$		00	01	11	10
τ_1	0	α_1	α_2	α_3	α_4
	1	α_8	α_5	α_6	α_7

Fig. 4.32 Optimal encoding of EOLC for CMCU $U_{15}(\Gamma_8)$

In this case $G_E = 8, R_9 = 3, \tau = \{\tau_1, \tau_2, \tau_3\}$. As follows from Fig. 4.32, class $B_2 \in \Pi_E$ is split into classes: $B_2^1 = \{\alpha_2, \alpha_3, \alpha_5, \alpha_6\}, B_2^2 = \{\alpha_4\}$ with binary codes $K(B_2^1) = **1$ and $K(B_2^2) = *1*$ (code $K(\alpha_7)$ is treated here as insignificant input assignment). The class $B_1 \in \Pi_E$ corresponds to code $K(B_1) = 000$ and new class $B_3 = \{\alpha_8\}$ corresponds to the code $K(B_3) = 1*0$. The microinstruction addresses of Fig. 4.33 were found after encoding the components of EOLC $\alpha_g \in C_E \cup C_E^2$ (in this case $R_7 = 2, T = \{T_1, T_2\}$).

As in all previous cases, construction of the control memory content is executed in a trivial way. Procedure P_7 is used to construct transition table of CMCU U_{15} . In this example, system of transition formulae has the form:

$$\begin{aligned}
 B_1 &\rightarrow x_1x_2I_2 \vee x_1\bar{x}_2I_3 \vee \bar{x}_1x_3I_4 \vee \bar{x}_1\bar{x}_3x_4I_5 \vee \bar{x}_1\bar{x}_3\bar{x}_4I_6; \\
 B_2^1 &\rightarrow I_8; \\
 B_2^2 &\rightarrow I_8; \\
 B_3 &\rightarrow x_1I_2 \vee \bar{x}_1x_2I_2 \vee \bar{x}_1\bar{x}_2I_7.
 \end{aligned}
 \tag{4.62}$$

$\tau_1\tau_2\tau_3$ T_1T_2	000	001	010	011	100	101	110	111
00	b_1	b_3	b_9	b_7	b_{20}	b_{13}	b_{18}	b_{16}
01	b_2	b_4	b_{10}	b_8	*	b_{14}	b_{19}	b_{17}
11	*	b_5	b_{11}	*	*	b_{15}	*	*
10	*	b_6	b_{12}	*	*	*	*	*

Fig. 4.33 Microinstruction addresses for CMCU $U_{15}(I_8)$

System (4.62) corresponds to the transition table with $H_{15}(I_8) = 10$ lines. In case of CMCU U_{12} , the transition table includes $H_{12}(I_8) = 20$ lines, in case of CMCU U_{13} it includes $H_{13}(I_8) = 15$ lines, and in case of CMCU U_{14} this table includes $H_{14}(I_8) = 8$ lines (due to the use of additional block TC). In general case, CMCU U_{15} is characterized by the following parameters:

$$R_{FB}^{15} = R_9; \tag{4.63}$$

$$t_{15}(\Gamma) = R_9 < R_2; \tag{4.64}$$

$$H_{15}(\Gamma) = \sum_{i=1}^{I_E} k_i H_i, \tag{4.65}$$

where $I_E = |II_E|$ after introduction of an extra EOLC.

Let us construct the table with parameters of different models of CMCU with code sharing (Table 4.10), where parameters $S_i(\Gamma)$, $t_i(\Gamma)$ and $H_i(\Gamma)$ for $i = 8, \dots, 15$ can be found together with comments allowing comparison of these parameters with other models of CMCU. As in case of corresponding Table 3.10 for CMCU $U_1 - U_7$, only the parameters of combinational circuit CC are shown in Table 4.10. It follows from the fact that the size of control memory CM is the same for all equivalent control units.

It follows from Table. 4.10 that models U_i ($i = 8, \dots, 11$) are analogues of models U_{i+4} , but different approaches to find the set of operational linear chains are used. Models U_{10} and U_{14} have minimal hardware amount among all models from the class considered, which is the consequence of using additional transformer TC, consuming some additional resources. Models based on EOLC have minimal number of CC outputs, but need more inputs and terms in comparison with their analogues based on OLC.

Introduction of extra vertices leads to CMCU with average values of analyzed parameters, but is connected with the decrease of resulting digital system performance.

Choice of particular model of CMCU depends on characteristics of the initial GSA, logical elements and optimality criterion assumed for the designed control unit.

Table 4.10 Comparative characteristics of CMCU U_8-U_{15}

Model	Systems	Parameters	Comments
U_8	$\Phi = \Phi(\tau, X)$ $\Psi = \Psi(\tau, X)$	$S_8(\Gamma) = R_6 + L$ $t_8(\Gamma) = R_6 + R_7 = R_2$ $H_8(\Gamma) = H_{\max}$	Average Maximum Maximum
U_9	$\Phi = \Phi(\tau', X)$ $\Psi = \Psi(\tau', X)$	$S_9(\Gamma) = R_{FB}^9(\Gamma) + L \leq S_8(\Gamma)$ $t_9(\Gamma) \leq R_2$ $H_9(\Gamma) = H_{\min} k_i = 1$	Average Average Minimum
U_{10}	$\Phi = \Phi(z, X)$ $\Psi = \Psi(z, X)$	$S_{10}(\Gamma) = R_4 + L$ $t_{10}(\Gamma) \leq R_2$ $H_{10}(\Gamma) \leq H_1(\Gamma)$	Minimum Average Minimum
U_{11}	$\Phi = \Phi(\tau', X)$ $\Psi = \Psi(\tau', X)$	$S_{11}(\Gamma) = R_6 + L$ $t_{11}(\Gamma) \leq R_2$ $H_{10}(\Gamma) \leq H_{11}(\Gamma) < H_{\max}$	Average Average Average
U_{12}	$\Psi = \Psi(\tau, X)$	$S_{12}(\Gamma) = R_9 + L \geq S_8(\Gamma)$ $t_{12}(\Gamma) = R_9$ $H_{12}(\Gamma) = H_{\max}$	Maximum Minimum Maximum
U_{13}	$\Psi = \Psi(\tau', X)$	$S_{13}(\Gamma) = R_9 + L$ $t_{13}(\Gamma) = R_9 < R_2$ $H_{13}(\Gamma) > H_{11}(\Gamma)$	Maximum Minimum Average
U_{14}	$\Psi = \Psi(z, X)$	$S_{14}(\Gamma) = R_{11} + L < R_{12}(\Gamma)$ $t_{14}(\Gamma) = R_9$ $H_{14}(\Gamma) \leq H_{13}(\Gamma)$	Average Minimum Average
U_{15}	$\Psi = \Psi(\tau, X)$	$S_{15}(\Gamma) = R_9 + L$ $t_{15}(\Gamma) = R_9$ $H_{15}(\Gamma) \geq H_{14}(\Gamma)$	Maximum Minimum Average

References

1. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
2. A. A. Barkalov. Principles of optimization of logic circuit of Moore FSM. *Cybernetics and System Analysis*, (1):65–72, 1998. (in Russian).
3. A. A. Barkalov. *Synthesis of Control Units with PLDs*. Donetsk National Technical University, 2002. (in Russian).
4. A. A. Barkalov, M. Wêgrzyn, and R. Wiśniewski. Partial reconfiguration of compositional microprogram control units implemented on fpgas. In *Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems (Brno)*, pages 116–119, 2006.
5. A.A. Barkalov and R. Wiśniewski. Optimization of compositional microprogram control unit with elementary operational linear chains. *Control systems and machines*, (5):25–29, 2004.
6. A.A. Barkalov and R. Wiśniewski. Optimization of compositional microprogram control units implemented on system-on-chip. *Theoretical and applied informatics*, (9):7–22, 2005.

7. A.A. Barkalova, L.A. Titarenko, and M. Kolopencyk. Design of compositional microprogram control units with expanded microinstructions and elementary OLCs. *Measurements. Automation. Control.*, 53(5):72–74, 2007.
8. T. Łuba, K. Jasiński, and B. Zbierchowski. *Specialized digital circuits in PLD i FPGA structures*. Wydawnictwo Komunikacji i Łączności, 1997. (in Polish).
9. E. McCluskey. *Logic Design Principles*. Prentice Hall, Englewood Cliffs, 1986.
10. G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw–Hill, 1994.

Chapter 5

Synthesis of compositional microprogram control units with object transformation

Abstract The chapter is devoted to optimization methods based on the transformation of codes. This method can be applied, when the FSM performing microinstruction addressing generates functions, which are subsequently loaded into two different memory blocks (the register and the counter). Some of these functions are treated as primary objects and the others as secondary objects. The FSM combinational part generates codes of primary objects and some additional variables only. This information is used to generate codes of secondary objects and permits to reduce the number of inputs in the block generating codes of secondary objects, in comparison with the initial combinational part of FSM and to reduce the hardware amount of resulting CMCU logic.

5.1 Optimization principles for CMCU with object transformation

Main conception of object transformation [3,4] is the following one. Let some combinational circuit CC implement the systems of Boolean functions

$$A = A(C), \quad (5.1)$$

$$B = B(C), \quad (5.2)$$

where $|A| = t_1$, $|B| = t_2$, $|C| = S_1$ (Fig. 5.1a).

In this case, CC has S_1 inputs and $t_1 + t_2$ outputs. Let us call systems A and B the objects of device CC. The number of outputs of the CC can be reduced, due to presentation of functions of the system B, using the functions of system A and some extra object V. In this case, device CC_1 implements functions (5.1) and some extra functions

$$V = V(C). \quad (5.3)$$

In this case, system (5.2) transforms to the form:

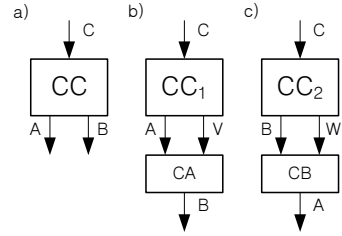


Fig. 5.1 Principle of objects transformation

$$B = B(A, V), \quad (5.4)$$

which is implemented by a code transformer CA (Fig. 5.1b). Assuming $|V| = t_3$, the number of outputs of CC_1 can be estimated as $t_1 + t_3$, and the code transformer CA has $t_1 + t_3$ inputs. If $t_3 < t_2$ and $S_1 > t_1 + t_3$, introduction of the code transformer CA reduces both the number of functions implemented by the device CC and the number of input variables in the system B.

Let us represent function A using function B and some additional system of functions W, where $|W| = t_4$. In this case CC_2 implements both the system (5.2) and extra system

$$W = W(C). \quad (5.5)$$

System (5.1) takes the form:

$$A = A(B, W). \quad (5.6)$$

System (5.1) is implemented by code transformer CB (Fig. 5.1c). Now the device CC implements $t_2 + t_4$ functions of S_1 input variables, and the block CB implements t_1 functions, of $t_2 + t_4$ input variables. If $t_4 < t_1$ and $S_1 > t_2 + t_4$, introduction of the block CB reduces both the number of functions implemented by the device CC and the number of input variables of the system A.

This transformation makes sense only if the total hardware amount either in the system $\langle CC_1, CB \rangle$ or in the system $\langle CC_2, CB \rangle$ is smaller, than the hardware amount of the initial combinational circuit CC. Let us point out that introduction of code transformers leads to deterioration of CMCU performance, due to its longer cycle time. Let us analyze models U_1-U_{15} to find out, for which of them it would be reasonable to use the object transformation. Of course, only the CC should be analyzed.

In case of CMCU U_1 , the combinational circuit CC generates input memory functions, Φ and Ψ . It means that two kinds of transformation are possible in this case, namely the transformation of state code into the address of OLC input and the inverse transformation. Thus, two objects exist in case of CMCU U_1 , namely the state codes of the addressing FSM S_1 and input addresses of OLC $\alpha_g \in C$.

In CMCU $U_2 - U_7$ the combinational circuit CC generates input memory functions Φ . It means that only a single object exists in these cases and the object transformation has no sense.

In CMCU $U_8 - U_{11}$ the circuit CC generates functions Φ and Ψ . It means that there are two objects and the transformation is possible. The transformation of input address of OLC into code of OLC is not possible however, because only the codes of

components are generated by functions Φ . In consequence, the bit capacity of new object code is equal to bit capacity of OLC code. Therefore, only the transformation of OLC code into component codes is possible in these cases.

In CMCU $U_{12} - U_{15}$ the circuit CC generates only one object and object transformation makes no sense. Therefore, the following object transformations are possible for the CMCU $U_1 - U_{15}$:

- transformation of state codes into addresses of OLC inputs (U_1);
- transformation of addresses of OLC inputs into state codes (U_1)
- transformation of OLC codes into codes of OLC components ($U_8 - U_{11}$).

In case of CMCU with code sharing, if condition

$$R_6 + R_7 > R_2 \quad (5.7)$$

holds, the number of outputs of the circuit CC is minimal, but the size of control memory increases drastically in comparison with its minimal value V_{\min} . Let us remember, that $R_6 = \lceil \log_2 G \rceil$ is the number of bits in codes of $OLC \alpha_g \in C$, $R_7 = \lceil \log_2 L_{\max} \rceil$ is the number of bits in codes of components of OLC, where L_{\max} is the maximal number of components in OLC $\alpha_g \in C$, $R_2 = \lceil \log_2 M_2 \rceil$ is the minimal number of bits in microinstruction address, $M_2 = |B_1|$, where B_1 is a set of operator vertices of the interpreted GSA. In case of code sharing an address $A(b_q)$ of microinstruction $Y(b_q)$, corresponding to vertex $b_q \in B_1$, is determined by concatenation of the two codes, namely:

$$A(b_q) = K(\alpha_g) * K(b_q), \quad (5.8)$$

where $K(\alpha_g)$ is the code of OLC $\alpha_g \in C$, $K(b_q)$ is the code of component of OLC $\alpha_g \in C$, corresponding to the vertex b_q . Let $C(b_q)$ be an address of microinstruction $Y(b_q)$ with R_2 bits. If condition (5.7) holds, the size of control memory CM can be reduced, due to introduction of the address transformer AT, which transforms two presentation forms of the same object. In this case, microinstruction address is used as this object [2]. The address is not determined here by variables T (output of the counter CT) and τ (output of the register RG), as for the circuit shown in Fig. 5.2a, but by some functions V , generated by the address transformer AT of Fig. 5.2b.

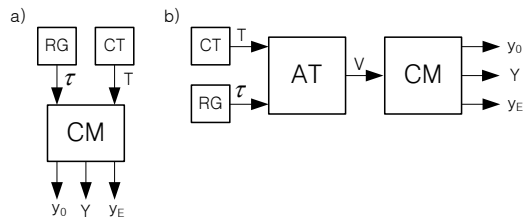


Fig. 5.2 Address transformation for CMCU with code sharing

The address transformer AT generates functions

$$V = V(\tau, T), \quad (5.9)$$

which are treated as address bits to form an address $C(b_q)$ with R_2 bits. This approach allows to reduce the control memory size V_{\min} , if condition (5.7) holds. Let us point out, that application of AT increases the cycle time of CMCU.

In case of CMCU with elementary OLC, the control memory size is increased in comparison with V_{\min} under the condition

$$R_9 + R_7 > R_2 \quad (5.10)$$

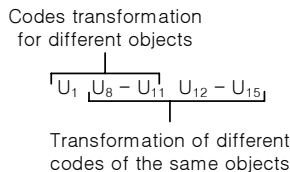
where R_9 is the number of bits in the code of EOLC $\alpha_g \in C_E$. This value can be minimized through the use of address transformer AT.

Thus, two types of object transformation can be applied to reduce the hardware amount of the logic circuit of compositional microprogram control unit:

- codes transformation for different objects (U_1, U_8-U_{11});
- transformation of different codes of the same object (U_8-U_{15}).

Let us point out, that only the first approach is possible in case of CMCU U_1 , and it gives the three-level structure of logic circuit. Only the second approach can be used in the cases of CMCU $U_{12}-U_{15}$. It leads also to the three-level structure of logic circuits. Both approaches can be used for the CMCU U_8-U_{11} . If only one of them is applied, resulting circuit has a three-level structure. If both methods are applied simultaneously, we obtain the four-level structure. All possibilities mentioned above are shown in Fig. 5.3.

Fig. 5.3 Possible transformations of object codes for CMCU



5.2 Objects transformation for CMCU with basic structure

Let us discuss the organization principle of CMCU U_{16} (Fig. 5.4), produced by adding special code transformer TSA to the CMCU U_1 . This extra block TSA transforms state codes of addressing FSM S_1 into the microinstruction addresses of microprogram control unit S_2 .

In case of CMCU U_{16} , combinational circuit CC generates input memory functions for register RG, which keeps codes $K(a_m)$ of FSM S_1 . These functions are represented by the following system:

$$\Psi = \Psi(\tau, X). \quad (5.11)$$

The combinational circuit CC generates also some additional variables $v_r \in V$, where $|V| = R_{12}$, used to identify microinstruction addresses. These functions are represented by the system:

$$V = V(\tau, X). \quad (5.12)$$

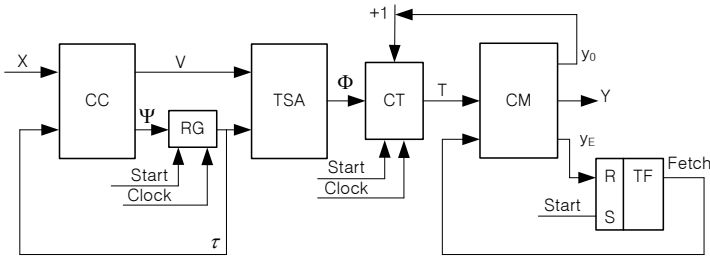


Fig. 5.4 Structural diagram of CMCU U_{16}

Code transformer TSA generates input memory variables of the counter CT

$$\Phi = \Phi(\tau, V). \quad (5.13)$$

The functions (5.13) represent microinstructions addresses corresponding to the inputs of operational linear chains $\alpha_g \in C$.

Functions of all other corresponding blocks are the same for both CMCU U_1 and U_{16} . Let us find some formula to express the parameter R_{12} .

In the transformed GSA $\Gamma(U_1)$, mark of the state $a_m \in A$ is the same at least for all inputs of a single OLC $\alpha_g \in C$. Let $B(a_m)$ be a set of OLC inputs, marked by state $a_m \in A$, and K_m be the number of elements in this set. It is clear that maximal value of parameter K_m can be found as:

$$K_{\max} = \max(K_1, \dots, K_{M_1}), \quad (5.14)$$

where $M_1 = |A|$. Now parameter R_{12} can be found in the form:

$$R_{12} = \lceil \log_2 K_{\max} \rceil. \quad (5.15)$$

Let us encode each input $I_g^j \in B(a_m)$ by a binary code $K(I_g^j)$ with R_{12} bits. In this case microinstruction address $A(b_q)$ corresponding to operator vertex $b_q = I_g^j$ is determined as the concatenation

$$A(b_q) = K(a_m) * K(I_g^j), \quad (5.16)$$

where $*$ is a concatenation sign. Expression (5.16) can be used to form the truth table of code transformer TSA. Synthesis of CMCU $U_{16}(\Gamma)$ includes the following steps:

1. Application of procedures $P_1 - P_4$ to initial GSA Γ .
2. Construction of the control memory content.

3. Construction of the pairs $\langle \text{state of FSM } S_1, \text{input of OLC } \alpha_g \in C \rangle$ for transformed GSA $\Gamma(U_{16})$.
4. Encoding of states $a_m \in A$ and OLC inputs $I_g^j \in I(\Gamma)$.
5. Construction of transition table for CMCU U_{16} .
6. Construction of the table of code transformer TSA.
7. Synthesis of CMCU logic circuit with given logical elements.

Application of procedures $P_1 - P_4$ gives the following results for some graph-scheme of the algorithm Γ_9 :

1. Set of OLC $C = \{\alpha_1, \dots, \alpha_5\}$, where $\alpha_1 = \langle b_1, b_2, b_3 \rangle$, $I_1^1 = b_1$, $I_1^2 = O_1 = b_3$; $\alpha_2 = \langle b_4, \dots, b_7 \rangle$, $I_2^1 = b_4$, $I_2^2 = b_5$, $I_2^3 = O_2 = b_7$; $\alpha_3 = \langle b_8, \dots, b_{12} \rangle$, $I_3^1 = b_8$, $O_3 = b_{12}$; $\alpha_4 = \langle b_{13}, \dots, b_{16} \rangle$, $I_4^1 = b_4$, $I_4^2 = b_{15}$, $O_4 = b_{16}$; $\alpha_5 = \langle b_{17}, b_{18}, b_{19} \rangle$, $I_5^1 = b_{17}$, $I_5^2 = O_5 = b_{19}$.
2. Inputs $I(\Gamma_9) = \{b_1, b_3, b_4, b_5, b_7, b_8, b_{13}, b_{15}, b_{17}, b_{19}\}$.
3. Microinstruction addresses $A(b_q)$ having $R_2 = 5$ bits (Fig. 5.5). Let us point out that each OLC $\alpha_g \in C$ is now determined unambiguously by variables T_1, T_2, T_3 .
4. Transformed GSA $\Gamma_9(U_{16})$ is shown in Fig. 5.6. The input memory functions $D_r \in \Phi$ are taken from Fig.5.5. Resulting GSA is marked by states of Mealy FSM and form a set $A = \{a_1, \dots, a_3\}$.

$T_1 T_2 T_3$		$T_1 T_2$							
		000	001	010	011	100	101	110	111
$T_1 T_2$	00	b_1	b_5	b_9	b_{13}	b_{17}	*	*	*
	01	b_2	b_6	b_{10}	b_{14}	b_{18}	*	*	*
	11	b_3	b_7	b_{11}	b_{15}	b_{19}	*	*	*
	10	b_4	b_8	b_{12}	b_{16}	*	*	*	*

Fig. 5.5 Microinstruction addresses for CMCU $U_{16}(\Gamma_9)$

Construction of the control memory content is executed in a standard way and is not discussed in this particular case.

Let us find the following pairs $\langle a_m, I_g^j \rangle$ for the GSA $\Gamma_9(U_{16})$, where each pair contains a state $a_m \in A$ and input I_g^j , corresponding to operator vertex with input marked by a_m . The following pairs are formed in this case: $\beta_1 = \langle a_1, I_4^1 \rangle$, $\beta_2 = \langle a_1, I_4^2 \rangle$, $\beta_3 = \langle a_1, I_5^1 \rangle$, $\beta_4 = \langle a_1, I_5^2 \rangle$, $\beta_5 = \langle a_2, I_1^1 \rangle$, $\beta_6 = \langle a_2, I_1^2 \rangle$, $\beta_7 = \langle a_3, I_2^1 \rangle$, $\beta_8 = \langle a_3, I_2^2 \rangle$, $\beta_9 = \langle a_3, I_2^3 \rangle$, $\beta_{10} = \langle a_3, I_3^1 \rangle$. Now the sets $B(a_1) = \{I_4^1, I_4^2, I_5^1, I_5^2\}$, $B(a_2) = \{I_1^1, I_1^2\}$, $B(a_3) = \{I_2^1, I_2^2, I_2^3, I_3^1\}$ can be formed with $K_1 = K_3 = 4$, $K_2 = 2$, and $K_{\max} = 4$ can be found using formula (5.14).

Encoding of the states $a_m \in A$ and OLC inputs $I_g^j \in I(\Gamma)$ is executed in a trivial way. In case of the CMCU $U_{16}(\Gamma_9)$ it could be found that $M = 3$, $R_1 = 2$, $\tau = \{\tau_1, \tau_2\}$. Let the states have following codes: $K(a_1) = 00$, $K(a_2) = 01$, $K(a_3) = 10$. It

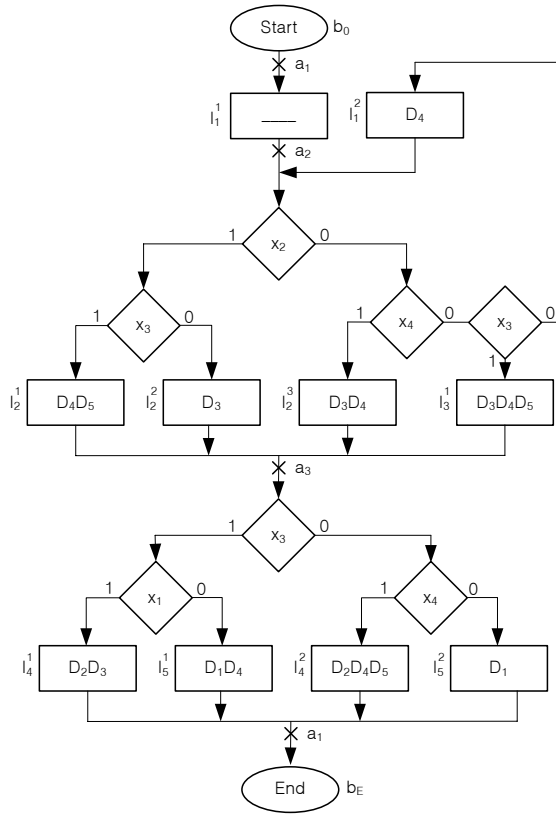


Fig. 5.6 Transformed GSA $I_9(U_{16})$

is sufficient to have $R_{12} = 2$ variables to encode the OLC inputs, when $V = \{v_1, v_2\}$. Codes $K(I_g^j)$ of the OLC inputs are shown in Table 5.1.

Codes of states of the addressing FSM and of OLC inputs are used to find input addresses given by (5.16). It can be found, for example, that $A(I_2^2) = 1001 = K(a_3) * K(I_2^2)$.

Table 5.1 Encoding of OLC inputs for CMCU $U_{16}(I_9)$

$B(a_1)$	$K(I_g^j)$	$B(a_2)$	$K(I_g^j)$	$B(a_3)$	$K(I_g^j)$
I_4^1	00	I_1^1	00	I_2^1	00
I_4^2	01	I_1^2	01	I_2^2	01
I_5^1	10			I_3^3	10
I_5^2	11			I_3^1	11

Transition table of CMCU U_{16} is represented by structure table of the addressing FSM S_1 . It is constructed using standard approach given in [1] and includes

the columns: $a_m, K(a_m), a_s, K(a_s), I_g^j, K(I_g^j), X_h, \Psi_h, V_h, h$, where column V_h contains variables $v_r \in V$, equal to 1 in the code $K(I_g^j)$ from line h of the table ($h = 1, \dots, H_{16}(\Gamma)$). The transition table includes here $H_{16}(\Gamma) = 10$ lines (Table 5.2).

The table of code transformer TSA describes transformation of codes $A(I_g^j)$ into the addresses $A(b_q)$, where $b_q = I_g^j$ ($g = 1, \dots, G$). The number of lines is here the same as the number NP_1 of different pairs $\langle a_m, I_g^j \rangle$, which can be found in the transition table. Table of TSA includes the columns $a_m, K(a_m), I_g^j, K(I_g^j), b_q, A(b_q), \Phi_h, h$ and has $H = NP_1$ lines. The column Φ_h contains variables $D_r \in \Phi$, equal to 1, and used to load an address $A(b_q)$ into the counter CT. In present case, the table of TSA includes $NP_1 = 10$ lines (Table 5.3).

Table 5.2 Transition table of CMCU $U_{16}(\Gamma_9)$

a_m	$K(a_m)$	a_s	$K(a_s)$	I_g^j	$K(I_g^j)$	X_h	Ψ_h	V_h	h
a_1	00	a_2	01	I_1^1	00	1	D_7	–	1
a_2	01	a_3	10	I_2^1	00	x_2x_3	D_6	–	2
		a_3	10	I_2^2	01	$x_2\bar{x}_3$	D_6	v_2	3
		a_3	10	I_2^3	10	\bar{x}_2x_4	D_6	v_1	4
		a_3	10	I_3^1	11	$\bar{x}_2\bar{x}_4x_3$	D_6	v_1v_2	5
		a_2	01	I_1^2	01	$\bar{x}_2\bar{x}_4\bar{x}_3$	D_7	v_2	6
a_3	11	a_1	00	I_4^1	00	x_3x_1	–	–	7
		a_1	00	I_5^1	01	$x_3\bar{x}_1$	–	v_1	8
		a_1	00	I_4^2	10	\bar{x}_3x_4	–	v_2	9
		a_1	00	I_5^2	11	$\bar{x}_3\bar{x}_4$	–	v_1v_2	10

Synthesis of CMCU U_{16} logic circuit is reduced to the implementation of systems (5.11)–(5.13) using PLD chips and of the control memory CM using either PROM or RAM chips. Systems (5.11)–(5.12) are constructed using the transition table 5.2 and, for example, the following Boolean expressions can be found (after minimization):

$$D_6 = \bar{\tau}_1 \tau_2 x_2 \vee \bar{\tau}_1 \tau_2 \bar{x}_2 x_4 \vee \bar{\tau}_1 \tau_2 \bar{x}_2 \bar{x}_4 x_3;$$

$$v_1 = \bar{\tau}_1 \tau_2 \bar{x}_2 x_4 \vee \bar{\tau}_1 \tau_2 \bar{x}_2 \bar{x}_4 x_3 \vee \tau_1 \bar{\tau}_2 x_3 \bar{x}_1 \vee \tau_1 \bar{\tau}_2 \bar{x}_3 \bar{x}_4.$$

The table of TSA is used to construct system (5.13). In this example, we obtain: $D_1 = \bar{\tau}_1 \bar{\tau}_2 v_1 \bar{v}_2 \vee \bar{\tau}_1 \bar{\tau}_2 v_1 v_2 = \bar{\tau}_1 \bar{\tau}_2 v_1$.

In case of the CMCU $U_{16}(\Gamma_9)$, the set of microoperations includes five elements: $Y = \{y_1, \dots, y_5\}$. Corresponding logic circuit of CMCU $U_{16}(\Gamma_9)$ is shown in Fig. 5.7.

In general case, the combinational circuit CC of CMCU U_{16} has the following characteristics:

$$R_{FB}^{16} = R_1; \quad (5.17)$$

$$t_{16}(\Gamma) = R_1 + R_{13}; \quad (5.18)$$

Table 5.3 Table of code transformer TSA for CMCU $U_{16}(\Gamma_9)$

a_m	$K(a_m)$	I_g^j	$K(I_g^j)$	b_q	$A(b_q)$	Φ_h	h
a_1	00	I_4^1	00	b_{13}	01100	D_2D_3	1
a_1	00	I_4^2	01	b_{15}	01110	$D_2D_3D_4$	2
a_1	00	I_5^1	10	b_{17}	10000	D_1	3
a_1	00	I_5^2	11	b_{19}	10010	D_1D_4	4
a_2	01	I_1^1	00	b_1	00000	–	5
a_2	01	I_1^2	01	b_3	00010	D_4	6
a_3	10	I_2^1	00	b_4	00011	D_4D_5	7
a_3	10	I_2^2	01	b_5	00100	D_3	8
a_3	10	I_2^3	10	b_7	00110	D_3D_4	9
a_3	10	I_3^1	11	b_8	00111	$D_3D_4D_5$	10

$$H_{16}(\Gamma) = H_1(\Gamma). \tag{5.19}$$

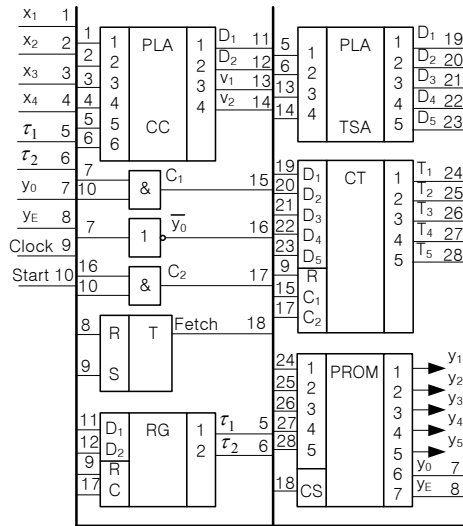


Fig. 5.7 Logic circuit of CMCU $U_{16}(\Gamma_9)$

Structural diagram of CMCU U_{17} is shown in Fig. 5.8, where the address transformer TAS generates state codes using microinstruction addresses.

The CMCU U_{17} is based, therefore, on the transformation: $\langle \text{OLC input address, next state code} \rangle$. In this model, combinational circuit CC generates the input memory functions for counter CT

$$\Phi = \Phi(\tau, X), \tag{5.20}$$

and address transformer TAS generates the input memory functions for register RG

$$\Psi = \Psi(T^l), \tag{5.21}$$

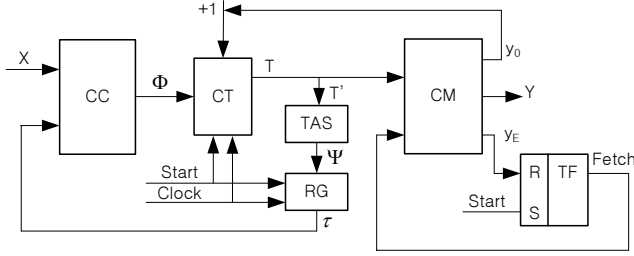


Fig. 5.8 Structural diagram of CMCU U_{17}

where $T' \subseteq T$. The OLC input determines unambiguously the state $a_m \in A$, which marked this input of GSA $\Gamma(U_{17})$, and therefore no additional variables should be used in this case. Let $|T'| = R_{13}$. If condition

$$R_{13} < R_2 \quad (5.22)$$

holds, block TAS has minimal possible number of inputs. In order to satisfy condition (5.22), the first R_{13} leftmost address bits should determine unambiguously the code $K(I_g^j)$ of input for OLC $\alpha_g \in C$. Analysis of Fig. 5.5 shows that $R_{13} = 5$ for the inputs I_1^2 and I_2^1 , because corresponding addresses differ only in the rightmost address bit. Let us apply the algorithm of special microinstruction addressing (procedure P_6), which should be modified as follows: shift to the right is continued till inputs of different OLC $\alpha_g \in C$ are placed in different columns of the addressing table. Let us denote this new procedure as procedure P_{12} . Application of this procedure to microinstruction addresses of the CMCU $U_{16}(I_9)$ gives the new addresses shown in Fig. 5.9.

Synthesis of CMCU $U_{17}(\Gamma)$ includes the following steps:

1. Application of procedures P_1 – P_4 to the initial GSA Γ .
2. Construction of the control memory content.
3. Construction of CMCU transition table.
4. Construction of the table of address transformer TAS.
5. Synthesis of CMCU U_{17} logic circuit with given logical elements.

Let us apply this method to the CMCU $U_{17}(I_9)$, microinstruction addresses of which are shown in Fig. 5.9. Transition table of CMCU U_{17} includes columns a_m , $K(a_m)$, I_g^j , $A(I_g^j)$, X_h , Φ_h , h . In case of the CMCU $U_{17}(I_9)$ it includes $H_{17}(I_9) = 10$ lines (Table 5.4).

This table is the base to construct the functions of system (5.20). For example, the following expression could be found from Table 5.4 (without minimization): $D_1 = \tau_1 \bar{\tau}_2 x_3 x_1 \vee \tau_1 \bar{\tau}_2 x_3 \bar{x}_1 \vee \tau_1 \bar{\tau}_2 \bar{x}_3 x_4 \vee \tau_1 \bar{\tau}_2 \bar{x}_3 \bar{x}_4$. This expression can be minimized to get the final Boolean function: $D_1 = \tau_1 \bar{\tau}_2$.

Table of the address transformer TAS includes the columns I_g^j , $K(I_g^j)$, a_m , $K(a_m)$, Ψ_h , h . Code $K(I_g^j)$ is constructed using insignificant address assignments. For example, as we find from Fig. 5.9: $A(I_5^1) = 10100$, and the inputs of OLC $\alpha_g \in C$ are

$T_1T_2T_3$		000	001	010	011	100	101	110	111
		T_4T_5							
00	00	b ₁	b ₄	b ₈	b ₁₂	b ₁₃	b ₁₇	*	*
	01	b ₂	b ₅	b ₉	*	b ₁₄	b ₁₈	*	*
	11	b ₃	b ₆	b ₁₀	*	b ₁₅	b ₁₉	*	*
	10	*	b ₇	b ₁₁	*	b ₁₆	*	*	*

Fig. 5.9 Microinstruction addresses for CMCU $U_{17}(I_9)$

unambiguously identified by $R_{13} = 3$ address bits $T_1T_2T_3$. In consequence we obtain at first the code $K(I_5^1) = 101**$, and using the generalized interval $111**$ of address space, the code $K(I_5^1) = 1*1**$ is finally obtained. Application of the same approach leads to the following input codes of OLC $\alpha_g \in C$ for control unit $U_{17}(I_9)$: $K(I_1^1) = 000**$, $K(I_2^1) = 000**$, $K(I_2^2) = 001**$, $K(I_2^3) = 001**$, $K(I_3^1) = 001**$, $K(I_3^2) = *10**$, $K(I_4^1) = 1*0**$, $K(I_4^2) = 1*0**$, $K(I_5^1) = 1*1**$, $K(I_5^2) = 1*1**$. Some of these codes are the same and therefore the transformation of only one of them should be shown in the table of TAS. Let us take the input with the least superscript value. This TAS table includes $NP_2 = 5$ lines (Table 5.5).

Table 5.4 Transition table of CMCU $U_{17}(I_9)$

a_m	$K(a_m)$	I_g^j	$A(I_g^j)$	X_h	Φ_h	h
a_1	00	I_1^1	00000	1	-	1
a_2	01	I_2^1	00100	x_2x_3	D_3	2
		I_2^2	00101	$x_2\bar{x}_3$	D_3D_5	3
		I_3^3	00111	\bar{x}_2x_4	$D_3D_4D_5$	4
		I_3^1	01000	$\bar{x}_2\bar{x}_4x_3$	D_2	5
		I_1^2	00010	$\bar{x}_2x_4x_3$	D_4	6
a_3	10	I_4^1	10000	x_3x_1	D_1	7
		I_3^3	10100	$x_3\bar{x}_1$	D_1D_3	8
		I_4^2	10010	\bar{x}_3x_4	D_1D_4	9
		I_5^2	10110	$\bar{x}_3\bar{x}_4$	$D_1D_3D_4$	10

The system (5.21) with $T' = \{T_1, T_2, T_3\}$ is constructed using this table and in particular $D_6 = \bar{T}_1\bar{T}_2\bar{T}_3$; $D_7 = \bar{T}_1\bar{T}_2T_3 \vee \bar{T}_2T_3$.

Logic circuit of the CMCU $U_{17}(I_9)$ is shown in Fig. 5.10.

In general case, combinational circuit CC of CMCU U_{17} has the following characteristics:

$$R_{FB}^{17} = R_1; \tag{5.23}$$

$$t_{17}(\Gamma) = R_2; \tag{5.24}$$

$$H_{17}(\Gamma) = H_1(\Gamma). \tag{5.25}$$

Table 5.5 Table of TAS for CMCU $U_{17}(\Gamma_9)$

I_g^j	$K(I_g^j)$	a_m	$K(a_m)$	Ψ_h	h
I_1^1	000**	a_2	01	D_6	1
I_2^1	001**	a_3	10	D_7	2
I_3^1	*10**	a_3	10	D_7	3
I_4^1	1*0**	a_1	00	-	4
I_5^1	1*1**	a_1	00	-	5

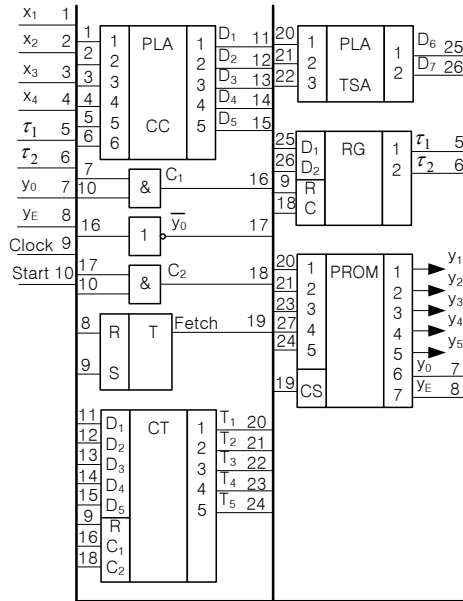


Fig. 5.10 Logic circuit of CMCU $U_{17}(\Gamma_9)$

Let us point out, that both CMCU $U_{16}(\Gamma)$ and $U_{17}(\Gamma)$ have smaller number of outputs than CMCU $U_1(\Gamma)$, and equal number of inputs and lines in the transition table. Optimization of logic circuit can be made with help of either TSA (U_{16}) or TAS (U_{17}), but in both cases some extra hardware is needed. The final choice among models U_1 , U_{16} and U_{17} can be made, therefore, only after implementation of their logic circuits, using particular elements. It should be pointed out, that the use of state transformer TSA increases the cycle time of CMCU $U_{16}(\Gamma)$, in comparison with $U_1(\Gamma)$ and $U_{17}(\Gamma)$.

5.3 Object transformation in CMCU with codes sharing

Application of the transformer of OLC code in the OLC component code for CMCU U_8 results in CMCU U_{18} , where block TOK executes the object transformation described above (Fig. 5.11).

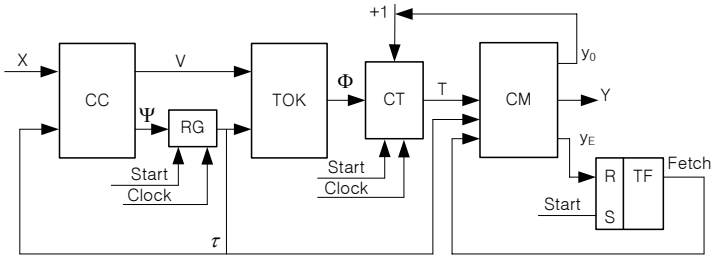


Fig. 5.11 Structural diagram of CMCU U_{18}

Here, the combinational circuit CC generates functions (5.11)–(5.12), where variables $\tau_r \in \tau$ are used to encode OLC $\alpha_g \in C$, and block TOK generates functions (5.13). In this case, system (5.12) includes R_{14} elements, where parameter R_{14} is determined in the following way.

Let $NI_{\max} = \max(NI_1, \dots, NI_G)$, where NI_g is the number of inputs for OLC $\alpha_g \in C$. The parameter R_{14} can be calculated using the following expression:

$$R_{14} = \lceil \log_2 NI_{\max} \rceil. \quad (5.26)$$

Let us encode each input I_g^j of OLC $\alpha_g \in C$ by a binary code $K(I_g^j)$ with R_{14} bits. In this case input address $A(I_g^j)$ is determined by concatenation

$$A(I_g^j) = K(\alpha_g) * K(I_g^j). \quad (5.27)$$

Code transformer TOK generates component code $K(b_q)$, where $b_q = I_g^j$, and this code is loaded into the counter CT.

Synthesis of CMCU $U_{18}(\Gamma)$ includes the following steps:

1. Preliminary transformation of initial GSA Γ (procedure P_4).
2. Construction of the OLC set for transformed GSA.
3. Encoding of operational linear chains $\alpha_g \in C$.
4. Encoding of components of operational linear chains $\alpha_g \in C$.
5. Encoding of the OLC inputs $I_g^j \in I(\Gamma)$.
6. Construction of the control memory content.
7. Construction of transition table for CMCU $U_{18}(\Gamma)$.
8. Construction of table for code transformer TOK.
9. Synthesis of CMCU logic circuit with given elements.

Let us discuss application of this method to the design of CMCU $U_{18}(\Gamma_{10})$, for which the transformed GSA $\Gamma_{10}(U_{18})$ is shown in Fig. 5.12.

In case of the GSA $\Gamma_{10}(U_{18})$ application of procedure P_1 leads to the set $C = \{\alpha_1, \dots, \alpha_8\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1^1 = b_1$, $O_1 = b_2$, $L_1 = 2$; $\alpha_2 = \langle b_3, \dots, b_6 \rangle$, $I_2^1 = b_3$, $I_2^2 = b_5$, $O_2 = b_6$, $L_2 = 4$; $\alpha_3 = \langle b_7, b_8, b_9 \rangle$, $I_3^1 = b_7$, $O_3 = b_9$, $L_3 = 3$; $\alpha_4 = \langle b_{10}, b_{11} \rangle$, $I_4^1 = b_{10}$, $I_4^2 = O_4 = b_{11}$, $L_4 = 2$; $\alpha_5 = \langle b_{12}, b_{13} \rangle$, $I_5^1 = b_{12}$, $O_5 = b_{13}$, $L_5 = 2$; $\alpha_6 = \langle b_{14}, b_{15}, b_{16} \rangle$, $I_6^1 = b_{14}$, $O_6 = b_{16}$, $L_6 = 3$; $\alpha_7 = \langle b_{17}, b_{18}, b_{19} \rangle$, $\alpha_8 = \langle b_{20} \rangle$, $I_8^1 = O_8 = b_{20}$, $L_8 = 1$.

The transformed GSA $\Gamma_{10}(U_{18})$ is characterized by $G = 8$, $R_6 = 3$, which means that $\tau = \{\tau_1, \tau_2, \tau_3\}$. Let us encode OLC $\alpha_g \in C$ in a trivial way: $K(\alpha_1) = 000, \dots, K(\alpha_8) = 111$. Analysis of OLC $\alpha_g \in C$ shows that $L_{\max} = L_2 = 4$, and therefore, $R_7 = 2$, $T = \{T_1, T_2\}$. Microinstruction addresses for the CMCU $U_{18}(\Gamma_{10})$ are shown in Fig. 5.13.

The value of parameter $NI_{\max} = \max(1, 2, 1, 2, 1, 1, 1, 1) = 2$ and therefore $R_{14} = 1$, $V = \{v_1\}$. Now the OLC inputs can be encoded as: $K(I_g^1) = 0$, $K(I_g^2) = 1$ ($g = 1, \dots, 8$). Construction of the control memory content for CMCU $U_{18}(\Gamma_{10})$ is performed in a trivial way. Transition table of CMCU U_{18} is constructed using the same approach, as the one used to construct transition table of CMCU U_2 . First step is reduced to construction of the system of transition formulae for outputs of OLC $\alpha_g \in C^1$. In the discussed case this system is:

$$\begin{aligned} O_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_2^2 \vee \bar{x}_1 \bar{x}_2 x_3 I_3^1 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 I_4^1; \\ O_2, O_3, O_4 &\rightarrow x_3 I_5^1 \vee \bar{x}_3 x_4 I_6^1 \vee \bar{x}_3 \bar{x}_4 I_7^1; \\ O_5, O_6, O_7 &\rightarrow x_2 I_8^1 \vee \bar{x}_2 I_4^2. \end{aligned} \quad (5.28)$$

Transition table is constructed on the base of this system. It includes the columns α_g , $K(\alpha_g)$, α_m , $K(\alpha_m)$, I_m^j , $K(I_m^j)$, X_h , Ψ_h , V_h , h , where α_g is an OLC corresponding to output O_g in the initial SFT; α_m is an OLC corresponding to input I_m^j from the right part of transition formula, such that $O_g = b_i$, $I_m^j = b_q$. The initial GSA Γ includes transition from b_i into b_q , determined by input X_h . The column Ψ_h includes input memory variables $D_r \in \Psi$, corresponding to 1 in the code $K(\alpha_m)$; the column V_h includes input memory variables $v_r \in V$, corresponding to 1 in the code $K(I_m^j)$. The number of lines $H_{18}(\Gamma)$ is determined by the number of terms in the system of transition formulae. In our example, this table includes $H_{18}(\Gamma_{10}) = 19$ lines (Table 5.6).

This table serves as the base for construction of functions (5.11)–(5.12). For example, it could be found (after minimization), that: $D_1 = \bar{\tau}_1 \bar{\tau}_2 \tau_3 \vee \bar{\tau}_1 \tau_2 \bar{\tau}_3 \vee \bar{\tau}_1 \tau_2 \tau_3 \vee \bar{\tau}_1 x_2$; $v_1 = \bar{\tau}_1 \bar{\tau}_2 \bar{x}_2 \bar{x}_1 x_2 \vee \tau_1 \bar{x}_2$.

In this particular case the input assignment 111 is treated as insignificant one, because the OLC $\alpha_8 \notin C^1$. From Table 5.6, we obtain the set of input memory functions $\Psi = \{D_1, D_2, D_3\}$.

Table of code transformer TOK includes the columns α_g , $K(\alpha_g)$, I_g^j , $K(I_g^j)$, b_q , $K(b_q)$, Φ_h , h , where vertex b_q corresponds to input I_g^j , and codes $K(b_q)$ can be found from microinstruction addresses. This table describes the transformation of

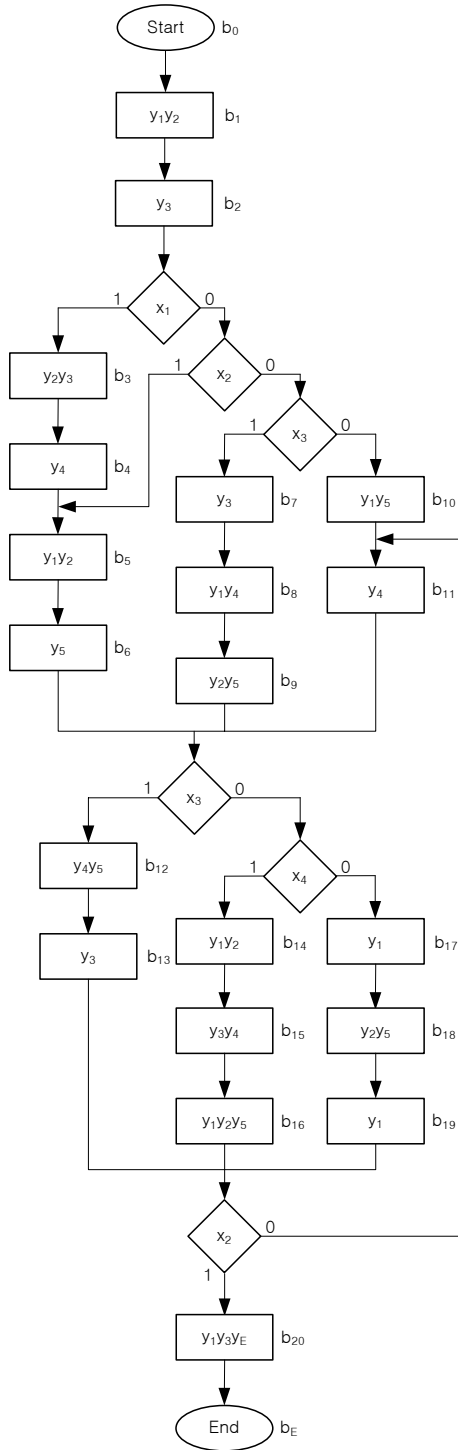


Fig. 5.12 Transformed GSA $\Gamma_{10}(U_{18})$

$\tau_1\tau_2\tau_3$	T_1T_2							
	000	001	010	011	100	101	110	111
00	b ₁	b ₃	b ₇	b ₁₀	b ₁₂	b ₁₄	b ₁₇	b ₂₀
01	b ₂	b ₄	b ₈	b ₁₁	b ₁₃	b ₁₅	b ₁₈	*
11	*	b ₅	b ₉	*	*	b ₁₆	b ₁₉	*
10	*	b ₆	*	*	*	*	*	*

Fig. 5.13 Microinstruction addresses for CMCU $U_{18}(\Gamma_{10})$

Table 5.6 Transition table of CMCU $U_{18}(\Gamma_{10})$

α_g	$K(\alpha_g)$	α_m	$K(\alpha_m)$	I_m^j	$K(I_m^j)$	X_h	Ψ_h	V_h	h
α_1	000	α_2	001	I_2^1	0	x_1	D_3	-	1
		α_2	001	I_2^2	1	\bar{x}_1x_2	D_3	v_1	2
		α_3	010	I_3^1	0	$\bar{x}_1x_2x_3$	D_2	-	3
		α_4	011	I_4^1	0	$\bar{x}_1\bar{x}_2\bar{x}_3$	D_2D_3	-	4
α_2	001	α_5	100	I_5^1	0	x_3	D_1	-	5
		α_6	101	I_6^1	0	\bar{x}_3x_4	D_1D_3	-	6
		α_7	110	I_7^1	0	$\bar{x}_3\bar{x}_4$	D_1D_2	-	7
α_3	010	α_5	100	I_5^1	0	x_3	D_1	-	8
		α_6	101	I_6^1	0	\bar{x}_3x_4	D_1D_3	-	9
		α_7	110	I_7^1	0	$\bar{x}_3\bar{x}_4$	D_1D_2	-	10
α_4	011	α_5	100	I_5^1	0	x_3	D_1	-	11
		α_6	101	I_6^1	0	\bar{x}_3x_4	D_1D_3	-	12
		α_7	110	I_7^1	0	$\bar{x}_3\bar{x}_4$	D_1D_2	-	13
α_5	100	α_8	111	I_8^1	0	x_2	$D_1D_2D_3$	-	14
		α_4	011	I_4^2	1	\bar{x}_2	D_2D_3	v_1	15
α_6	101	α_8	111	I_8^1	0	x_2	$D_1D_2D_3$	-	16
		α_4	011	I_4^2	1	\bar{x}_2	D_2D_3	v_1	17
α_7	110	α_8	111	I_8^1	0	x_2	$D_1D_2D_3$	-	18
		α_4	011	I_4^2	1	\bar{x}_2	D_2D_3	v_1	19

component addresses (5.27) into the component codes with R_7 bits. The number of lines in TOK table can be found from the relation

$$H(U_{18}) = \sum_{i=1}^G NI_g. \tag{5.29}$$

In the discussed case, TOK table includes $H(U_{18}(\Gamma_{10})) = 10$ lines (Table 5.7).

This table is now used to construct functions (5.13). In our case $\Phi = \{D_4, D_5\}$ and the equation: $D_4 = \bar{\tau}_1\bar{\tau}_2\tau_3v_1 \vee \bar{\tau}_1\tau_2\tau_3v_1$, can be extracted from Table 5.7.

Synthesis of logic circuit of CMCU $U_{18}(\Gamma)$ is reduced to implementation of systems (5.11)–(5.13) using PLD chips and control memory with either PROM or RAM chips. Logic circuit of the CMCU $U_{18}(\Gamma_{10})$ is shown in Fig. 5.14.

Table 5.7 Table of code transformer TOK for CMCU $U_{18}(\Gamma_{10})$

α_g	$K(\alpha_g)$	I_g^j	$K(I_g^j)$	b_q	$K(b_q)$	Φ_h	h
α_1	000	I_1^1	0	b_1	00	-	1
α_2	001	I_2^1	0	b_3	00	-	2
α_2	001	I_2^2	1	b_5	10	D_4	3
α_3	010	I_3^1	0	b_7	00	-	4
α_4	011	I_4^1	0	b_{10}	00	-	5
α_4	011	I_4^2	1	b_{11}	10	D_4	6
α_5	100	I_5^1	0	b_{12}	00	-	7
α_6	101	I_6^1	0	b_{14}	00	-	8
α_7	110	I_7^1	0	b_{17}	00	-	9
α_8	111	I_8^1	0	b_{20}	00	-	10

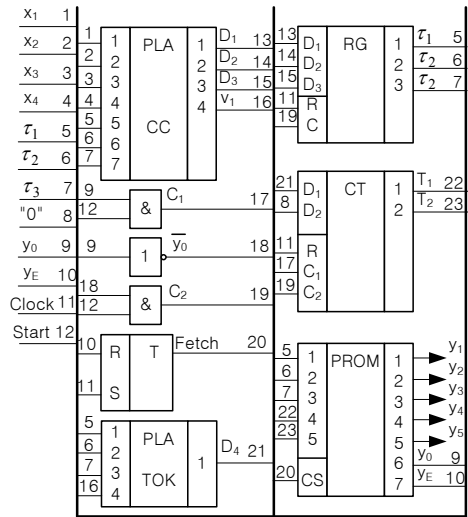


Fig. 5.14 Logic circuit of CMCU $U_{18}(\Gamma_{10})$

$$R_{FB}^{18} = R_6; \tag{5.30}$$

$$t_{18}(\Gamma) = R_6 + R_{14}; \tag{5.31}$$

$$H_{18}(\Gamma) = H_2(\Gamma). \tag{5.32}$$

In case of optimal encoding of OLC $\alpha_g \in C$, introduction of TOK transforms CMCU U_9 into CMCU U_{19} : the structural diagram of the later being the same as for CMCU U_{18} . Synthesis methods for both CMCU U_{19} and U_{18} are practically the same, but in the case of CMCU U_{19} optimal encoding of OLC is executed during step 3 of the synthesis procedure. It allows to reduce the hardware amount of combinational circuit CC, in comparison with the equivalent CMCU U_{18} . If OLC $\alpha_g \in C^1$ of the CMCU $U_{19}(\Gamma_{10})$ is encoded as: $K(\alpha_1) = 000$, $K(\alpha_2) = 100$, $K(\alpha_3) = 110$, $K(\alpha_4) = 101$, $K(\alpha_5) = 001$, $K(\alpha_6) = 010$, $K(\alpha_7) = 011$, $K(\alpha_8) = 111$, the number of lines of the CMCU $U_{19}(\Gamma_{10})$ transition table is equal to $H_{19}(\Gamma_{10}) = 11$. It means

that transition table has now 8 lines less, than the one corresponding to arbitrary OLC encoding (Table 5.6).

Introduction of the code transformer TOK in CMCU U_{10} leads to CMCU U_{20} (Fig. 5.15), where block TC generates the codes of classes $B_i \in \Pi_C$.

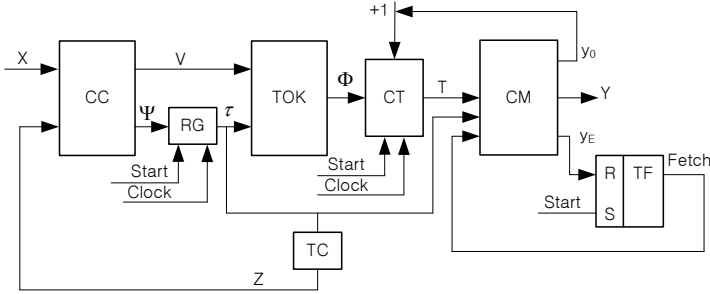


Fig. 5.15 Structural diagram of CMCU U_{20}

In CMCU U_{20} combinational circuit CC generates functions

$$\Psi = \Psi(Z, X), \quad (5.33)$$

$$V = V(Z, Z), \quad (5.34)$$

block TOK implements functions (5.13), and block TC generates functions

$$Z = Z(\tau), \quad (5.35)$$

which are next used to encode classes of pseudoequivalent OLC $\alpha_g \in C^1$. Synthesis method for CMCU $U_{20}(\Gamma)$ includes the following additional steps (in comparison with synthesis method used for CMCU $U_{18}(\Gamma)$):

- construction of the partition Π_C for set C^1 ;
- encoding of the equivalence classes $B_i \in \Pi_C$;
- construction of the table of code transformer TC.

Let us discuss application of this method to synthesis of the CMCU $U_{20}(\Gamma_{10})$. Obviously, outcomes of the steps 1 - 6 are the same for both the CMCU $U_{18}(\Gamma_{10})$ and $U_{20}(\Gamma_{10})$.

It follows from analysis of the GSA $\Gamma_{10}(U_{18})$ that $C^1 = \{\alpha_1, \dots, \alpha_7\}$ and the partition $\Pi_C = \{B_1, B_2, B_3\}$, where $B_1 = \langle \alpha_1 \rangle$, $B_2 = \langle \alpha_2, \alpha_3, \alpha_4 \rangle$, $B_3 = \langle \alpha_5, \alpha_6, \alpha_7 \rangle$. Thus, it could be found that $I = 3$, $R_4 = 2$, $Z = \{z_1, z_2\}$. Let us encode the classes $B_i \in \Pi_C$ using the following principle: the less OLC particular class includes the more zeros its code contains. Now the classes $B_i \in \Pi_C$ have the codes: $K(B_1) = 10$, $K(B_2) = 00$, $K(B_3) = 01$.

It is necessary to replace outputs of OLC $\alpha_g \in C^1$ in the system similar to (5.28) by corresponding classes $B_i \in \Pi_C$ to construct transition table of CMCU $U_{20}(\Gamma)$. In the discussed case system (5.28) is transformed giving the following system:

$$\begin{aligned}
B_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_2^2 \vee \bar{x}_1 \bar{x}_2 x_3 I_3^1 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 I_4^1; \\
B_2 &\rightarrow x_3 I_5^1 \vee \bar{x}_3 x_4 I_6^1 \vee \bar{x}_3 \bar{x}_4 I_7^1; \\
B_3 &\rightarrow x_2 I_8^1 \vee \bar{x}_2 I_4^2.
\end{aligned} \tag{5.36}$$

Transition table of CMCU $U_{20}(\Gamma)$ includes the columns: B_i , $K(B_i)$, α_m , $K(\alpha_m)$, I_m^j , $K(I_m^j)$, X_h , Ψ_h , V_h , h . Some codes $K(B_i)$ may include insignificant variables, as for example in case of the CMCU $U_{20}(\Gamma_{10})$ where the code 11 is not used and we have the codes: $K(B_1) = 1*$ and $K(B_3) = *1$, whereas the code $K(B_2)$ remains unchanged. Transition table of the CMCU $U_{20}(\Gamma_{10})$ includes $H_{20}(\Gamma_{10}) = 9$ lines (Table 5.8). Comparative analysis of Tables 5.6 and 5.8 shows that in our case the number of lines in the transition table reduces more, than twice, and the number of feedback variables is 1.5 times smaller.

Table 5.8 Transition table of CMCU $U_{20}(\Gamma_{10})$

B_i	$K(B_i)$	α_m	$K(\alpha_m)$	I_m^j	$K(I_m^j)$	X_h	Ψ_h	V_h	h
B_1	1*	α_2	001	I_2^1	0	x_1	D_3	–	1
		α_2	001	I_2^2	1	$\bar{x}_1 x_2$	D_3	v_1	2
		α_3	010	I_3^1	0	$\bar{x}_1 \bar{x}_2 x_3$	D_2	–	3
		α_4	011	I_4^1	0	$\bar{x}_1 \bar{x}_2 \bar{x}_3$	$D_2 D_3$	–	4
B_2	00	α_5	100	I_5^2	0	x_3	D_1	–	5
		α_6	101	I_6^1	0	$\bar{x}_3 x_4$	$D_1 D_3$	–	6
		α_7	110	I_7^1	0	$\bar{x}_3 x_4$	$D_1 D_2$	–	7
B_3	*1	α_8	111	I_8^1	0	x_2	$D_1 D_2 D_3$	–	8
		α_4	011	I_4^2	1	\bar{x}_2	$D_2 D_3$	v_1	9

This table is now used to construct functions (5.33)–(5.34). For example, the following equations can be got from Table 5.8 (after minimization): $D_1 = \bar{z}_1 \bar{z}_2 \vee z_2 x_2$; $v_1 = z_1 \bar{x}_1 x_2 \vee z_2 \bar{x}_2$.

Table of code transformer TOK used for the CMCU $U_{20}(\Gamma_{10})$ is the same as the one for the CMCU $U_{18}(\Gamma_{10})$. Table of code transformer TC includes the columns: α_g , $K(\alpha_g)$, B_i , $K(B_i)$, Z_g , g , where $g = 1, \dots, |C^1|$. In this example the table includes $G_1 = 7$ lines (Table 5.9).

Table 5.9 is now used to construct functions (5.35). Taking into account the insignificant input assignment 111, one can obtain the following expressions: $z_1 = \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3$, $z_2 = \tau_1$. Logic circuit of the CMCU $U_{20}(\Gamma_{10})$ is shown in Fig. 5.16, where logic circuits of the blocks CC, TOK and TC are implemented using PLA chips, and control CM is implemented with PROM.

In general case, combinational circuit CC of CMCU U_{20} has the following characteristics:

$$R_{FB}^{20} = R_4; \tag{5.37}$$

$$t_{20}(\Gamma) = R_6 + R_{14}; \tag{5.38}$$

$$H_{20}(\Gamma) = H_{10}(\Gamma). \tag{5.39}$$

Table 5.9 Table of code transformer TC for CMCU $U_{20}(\Gamma_{10})$

α_g	$K(\alpha_g)$	B_i	$K(B_i)$	Z_g	g
α_1	000	B_1	10	z_1	1
α_2	001	B_2	00	–	2
α_3	010	B_2	00	–	3
α_4	011	B_2	00	–	4
α_5	100	B_3	01	z_2	5
α_6	101	B_3	01	z_2	6
α_7	110	B_3	01	z_2	7

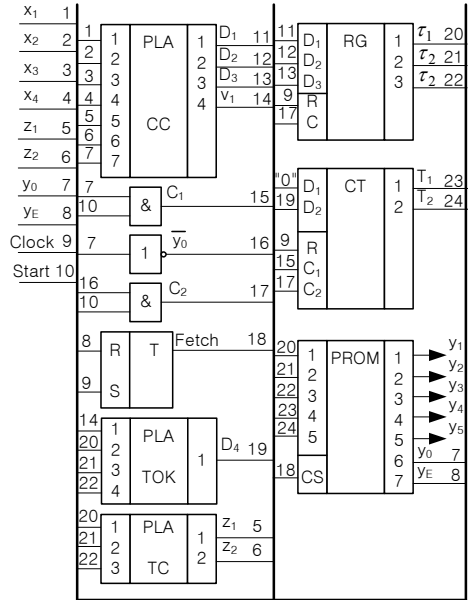
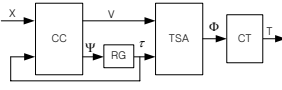
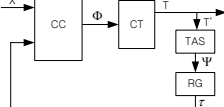
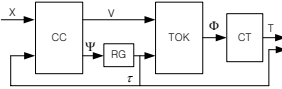
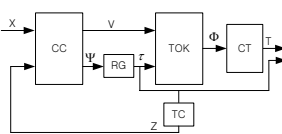


Fig. 5.16 Logic circuit of CMCU $U_{20}(\Gamma_{10})$

Introduction of code transformer TOK in CMCU U_{11} leads to CMCU U_{21} , having the same structural diagram as the CMCU U_{18} . Comparative characteristics of compositional microprogram control units with object transformation is represented in Table 5.10.

As follows from this table, each model of $U_{16}-U_{21}$ has some positive and negative features. In order to choose the best model, it is necessary to find the best basic model for a given GSA and logical elements. This particular model should be modified by introduction of TOK, TSA or TAS.

Table 5.10 Table of code transformer TC for CMCU $U_{20}(\Gamma_{10})$

U_i	Addressing FSM	Parameters of FSM	Comments
U_{16}		$R_{FB}^{16} = R_1$ $t_{16}(\Gamma) = R_1 + R_{13}$ $H_{16}(\Gamma) = H_1(\Gamma)$	Cycle time increases in comparison with cycle time of CMCU U_1 . Combinational circuit CC has nearly minimal number of outputs.
U_{17}		$R_{FB}^{17} = R_1$ $t_{17}(\Gamma) = R_2$ $H_{17}(\Gamma) = H_1(\Gamma)$	Maximal number of circuit CC inputs. Cycle time is the same as for CMCU U_1 .
U_{18} U_{19} U_{21}		$R_{FB}^{18} = R_6$ $t_{18}(\Gamma) = R_6 + R_{14}$ $H_{18}(\Gamma) = H_2(\Gamma)$ $R_{FB}^{19} \leq R_6$ $t_{19}(\Gamma) = t_{18}(\Gamma)$ $H_{19}(\Gamma) \leq H_2(\Gamma)$ $R_{FB}^{21} = R_6$ $t_{21}(\Gamma) = t_{18}(\Gamma)$ $H_1(\Gamma) < H_{21}(\Gamma) \leq H_2(\Gamma)$	Cycle time increases in comparison with the cycle time of CMCU U_8 . Optimal encoding of OLC and initial GSA transformation are oriented towards reduction of the number of lines of transition table.
U_{20}		$R_{FB}^{20} = R_4$ $t_{20}(\Gamma) = t_{18}(\Gamma)$ $H_{20}(\Gamma) = H_1(\Gamma)$	Cycle time increases in comparison with cycle time of CMCU U_{10} . Minimal numbers of inputs, outputs and terms of addressing is connected with use of extra block TC.

References

1. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
2. A. Barkalov and M. Węgrzyn. *Design of Control Units With Programmable Logic*. University of Zielona Góra Press, 2006.
3. A. A. Barkalov. *Synthesis of Control Units with PLDs*. Donetsk National Technical University, 2002. (in Russian).
4. A. A. Barkalov and A.A. Barkalov. Design of Mealy finite-state machines with transformation of object codes. *Inter. Journal "Applied Mathematics and Computer Science"*, 15(1):151–158, 2005.

Chapter 6

Control memory optimization for compositional microprogram control units with code sharing

Abstract The chapter considers some optimization methods used to reduce the size of CMCU control memory keeping the microprogram. These methods are based on the use of special address transformer permitting to keep the control memory size, which is the same as in case of the CMCU basic structure. One of the methods is oriented towards keeping only the original sets of microoperations and some additional variables in the control memory, in order to provide natural addressing and operation termination operating modes. The second approach assumes that a special CMCU block, which is not the part of its control memory, generates additional variables mentioned above. The methods proposed here permit to reduce control memory volume in comparison with the CMCU basic structure. Negative feature of this approach is decreasing of the CMCU performance because duration of the cycle becomes greater than in case of the CMCU basic structure.

6.1 Principles of control memory optimization

Use of code sharing principle allows minimizing parameter values of combinational circuit CC. Unfortunately, if condition

$$R_6 + R_7 > R_2 \quad (6.1)$$

holds, the size of control memory of CMCU $U_8 - U_{11}$ increases in comparison with its value for the equivalent CMCU U_1 . Also, if condition

$$R_9 + R_7 > R_2 \quad (6.2)$$

holds in case of CMCU $U_{12} - U_{15}$, the control memory size also increases. The size of CMCU U_1 control memory is minimal and let us denote it by V_{\min} . It should be remembered that equations (6.1) and (6.2) use the following parameters expressing bit capacity of particular codes: R_6 for OLC code; R_8 for component code of OLC or elementary OLC; R_9 for elementary OLC code; R_2 for address $A(b_q)$ of

microinstruction corresponding to vertex $b_q \in B_1$:

$$R_2 = \lceil \log_2 M_2 \rceil; \quad (6.3)$$

$$R_6 = \lceil \log_2 G \rceil; \quad (6.4)$$

$$R_7 = \lceil \log_2 L_{\max} \rceil; \quad (6.5)$$

$$R_9 = \lceil \log_2 G_E \rceil. \quad (6.6)$$

In these equations $M_2 = |B_1|$, $G = |C|$, $G_E = |C_E|$, L_{\max} is the maximal number of components among either all OLC $\alpha_g \in C$ or elementary OLC $\alpha_g \in C_E$, B_1 is a set of operator vertices of GSA Γ .

Methods discussed in this Chapter are based on our results from [1, 3–9]. If condition (6.1) is satisfied, the three following approaches can be applied to minimize the size of control memory:

- transformation of microinstruction address, represented by concatenation of the OLC code and its component code, into microinstruction address having R_2 bits;
- transformation of microinstruction address, represented by concatenation of OLC code and its component code, into the address of expanded microinstruction;
- transformation of microinstruction address, represented by concatenation of OLC code and its component code, into address of the microinstruction, corresponding to collection of microoperations $y_n \in Y$.

All three approaches are oriented towards transformation of microinstruction address (object) from some particular form of representation (code sharing) into another form of representation, which guarantees preservation of control memory size on V_{\min} level. In some cases, this level can be even make lower. First transformation is connected with introduction of additional address transformer block AT_1 into the CMCU $U_8 - U_{11}$ (Fig. 6.1).

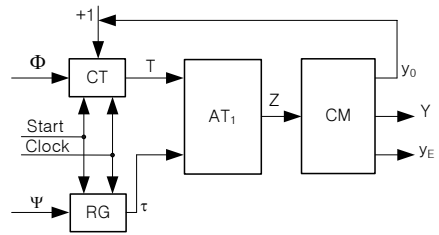


Fig. 6.1 First approach to address transformation

In this case block AT_1 transforms microinstruction address $A(b_q)$, represented by the concatenation

$$A(b_q) = K(\alpha_g) * K(b_q), \quad (6.7)$$

into microinstruction address $B(b_q)$, obtained after application of procedure P_2 . The transformation is reduced to generation of some functions

$$Z = Z(T, \tau), \quad (6.8)$$

used to address microinstructions $Y(b_q)$, $|Z| = R_2$. Thus, application of address transformer AT_1 allows preservation of minimal possible control memory size, if condition (6.1) is satisfied.

Let us call a collection of microoperations $y_n \in Y$ and additional variables y_0, y_E , written in some vertex $b_q \in B_1$, as an *expanded microinstruction* (EMI) and let us denote it by $Y_E(\Gamma)$. We find now a set $Y_E(b_q)$ of expanded microinstructions for GSA Γ , where $|Y_E(\Gamma)| = M_3$. Obviously, the same expanded microinstructions can be written for different operator vertices $b_q \in B_1$; it corresponds to inequality $M_3 \leq M_2$. Let R_{15} be the number of bits sufficient to address all expanded microinstructions. We have

$$R_{15} = \lceil \log_2 M_3 \rceil. \quad (6.9)$$

Expression (6.9) corresponds to the case, when each unique EMI is determined by a unique address of control memory CM. If condition

$$R_{15} < R_2 \quad (6.10)$$

holds, the control memory size for CMCU $U_8 - U_{11}$ can be

$$\Delta_{CM} = 2^{R_2 - R_{15}} \quad (6.11)$$

times less, than the value of parameter V_{\min} . Resulting reduction of the control memory size is connected with the use of address transformer AT_2 in CMCU $U_8 - U_{11}$ (Fig. 6.2).

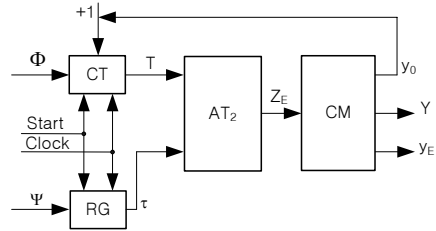


Fig. 6.2 Second approach for address transformation

The address transformer AT_2 generates functions

$$Z_E = Z_E(T, \tau), \quad (6.12)$$

which are used to generate an address $A_E(b_q)$, corresponding to a code $K_E(b_q)$ of expanded microinstruction.

Further optimization of the control memory size is connected with encoding of collections of microoperations $Y(b_q) \subseteq Y$. If some GSA Γ includes M_4 different collections of microoperations, then

$$R_{16} = \lceil \log_2 M_4 \rceil \quad (6.13)$$

binary variables is sufficient for their encoding. If condition

$$R_{16} = R_2 \quad (6.14)$$

holds, the control memory size can be reduced

$$\Delta M_0 = 2^{R_2 - R_{16}} \quad (6.15)$$

times, in comparison with V_{\min} . In this case, control memory CM keeps only microoperations $y_n \in Y$. Special combinational circuit should be used to generate additional variables y_0 and y_E . Let us denote this circuit by CCS. If condition (6.14) takes place, models $U_8 - U_{11}$ can be modified by introducing the address transformer AT_3 and block CCS (Fig. 6.3).

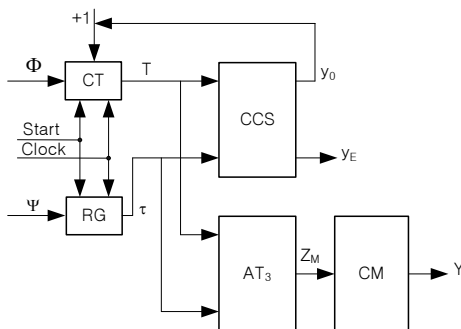


Fig. 6.3 Third approach to address transformation

In this case, block CCS generates functions

$$y_0 = y_0(T, \tau), \quad (6.16)$$

$$y_E = y_E(T, \tau), \quad (6.17)$$

and address transformer AT_3 generates a code $C(b_q)$ of some collection of microoperations on the base of address $A(b_q)$ represented by (6.7). Obviously, this approach offers the greatest possibilities for control memory size optimization. Address transformer AT_3 generates functions

$$Z_M = Z_M(T, \tau), \quad (6.18)$$

used for addressing collections of microoperations, where $|Z_M| = R_{16}$.

Let us use the initial GSA Γ_{11} for further discussion (Fig. 6.4). Application of procedure P_1 for the GSA Γ_{11} results in the following set of operational linear chains: $C = \{\alpha_1, \dots, \alpha_6\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1^1 = b_1$, $O_1 = b_2$, $L_1 = 2$; $\alpha_2 = \langle b_3, b_4, b_5 \rangle$, $I_2^1 = b_3$, $I_2^2 = b_4$, $O_2 = b_5$, $L_2 = 3$; $\alpha_3 = \langle b_6, b_7 \rangle$, $I_3^1 = b_6$, $O_3 = b_7$, $L_3 = 2$; $\alpha_4 = \langle b_8, b_9 \rangle$, $I_4^1 = b_8$, $O_4 = b_9$, $L_4 = 2$; $\alpha_5 = \langle b_{10}, \dots, b_{14} \rangle$, $I_5^1 = b_{10}$, $I_5^2 = b_{12}$, $O_5 = b_{14}$,

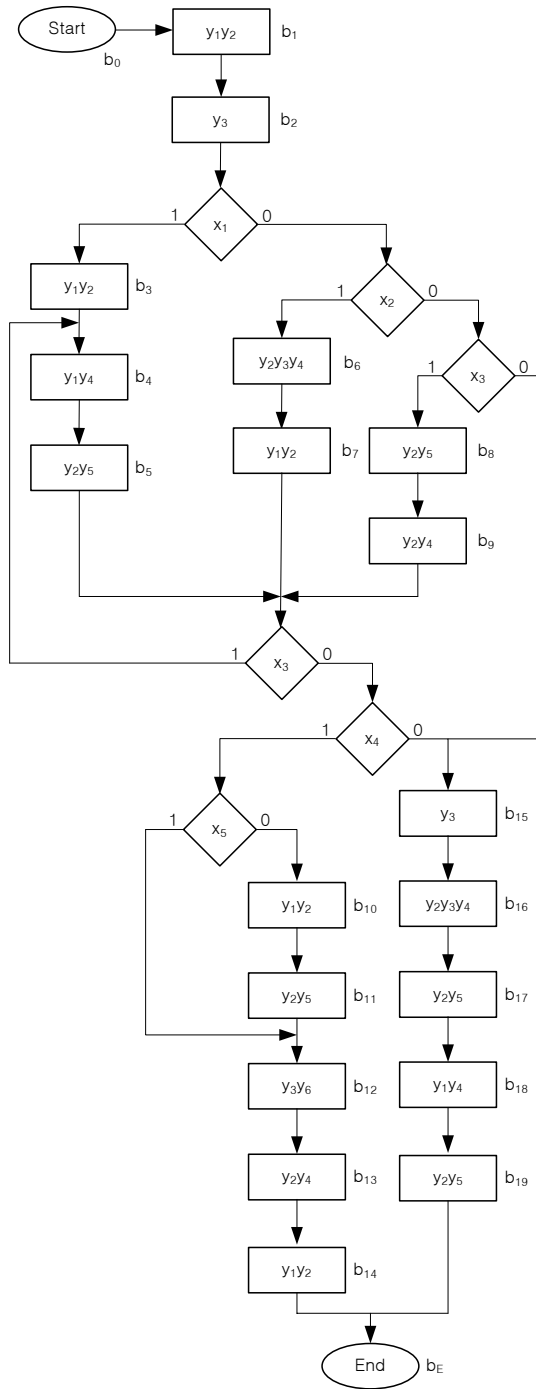


Fig. 6.4 Initial GSA Γ_{11}

$L_5 = 5$; $\alpha_6 = \langle b_{15}, \dots, b_{19} \rangle$, $I_6^1 = b_{15}$, $O_6 = b_{19}$, $L_6 = 5$. In this case $G = 6$, $R_6 = 3$, $L_{\max} = 5$, $R_7 = 3$, $M_2 = 19$, $R_2 = 5$. Thus, condition (6.1) is true and application of the address transformer possible.

Application of procedure P_{11} for the GSA Γ_{11} results in the following set of operational linear chains: $C_E = \{\alpha_1, \dots, \alpha_8\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $L_1 = 2$; $\alpha_2 = \langle b_3 \rangle$, $L_2 = 1$; $\alpha_3 = \langle b_4, b_5 \rangle$, $L_3 = 2$; $\alpha_4 = \langle b_6, b_7 \rangle$, $L_4 = 2$; $\alpha_5 = \langle b_8, b_9 \rangle$, $L_5 = 2$; $\alpha_6 = \langle b_{15}, \dots, b_{19} \rangle$, $L_6 = 5$; $\alpha_7 = \langle b_{12}, b_{13}, b_{14} \rangle$, $L_7 = 3$; $\alpha_8 = \langle b_{15}, \dots, b_{19} \rangle$, $L_8 = 5$. Here $G_E = 8$, $R_9 = 3$, $L_{\max} = 5$, $R_7 = 3$, $M_2 = 19$, $R_2 = 5$. Thus, condition (6.2) holds and the address transformer can also be used.

All approaches mentioned above are connected with decrease of performance, because introduction of address transformer causes the longer CMCU cycle time. If cycle time exceeds maximal possible value determined by the designer of control unit, application of the code sharing approach becomes useless.

All these approaches can be applied to minimize the control memory size of CMCU with elementary OLC only, when condition (6.2) takes place. Structural diagrams of these CMCU are identical to corresponding structures of CMCU based on OLC, but no input memory functions are connected with inputs of the counter CT. Application of the first approach allows preservation of minimal size of the control memory. Application of the second approach reduces the control memory size Δ_{CM} times in comparison with V_{\min} , where Δ_{CM} is determined by (6.1). Application of the third approach reduces the control memory size Δ_{M_0} times in comparison with V_{\min} , where Δ_{M_0} is determined by (6.15).

Synthesis methods applicable for CMCU $U_8 - U_{12}$ are similar to the optimization methods described above. We now take the CMCU U_8 model and use the same explanation. The methods used to reduce the control memory size of the CMCU with elementary OLC will be discussed for the CMCU U_{12} model.

6.2 Synthesis of CMCU with generation of microinstruction addresses

Use of address transformer AT_1 converts CMCU U_8 into CMCU U_{22} , which is shown in Fig. 6.5.

Operation principles are practically identical for both CMCU U_{22} and U_8 , but in case of the CMCU U_{22} , microinstruction address $A(b_q)$ is generated by an address transformer AT_1 . Synthesis method applied for CMCU U_{22} can be viewed as some modification of the synthesis method used already for CMCU U_8 , obtained by adding the steps involving procedure P_2 and construction of the table of address transformer AT_1 . Let us discuss an example of synthesis of the CMCU $U_{22}(\Gamma_{11})$.

It is clear, that the transformation of GSA Γ_{11} is reduced to insertion of variable y_E into operator vertices b_{14} and b_{19} . It was found previously, that in this particular case $R_6 = 3$, $\tau = \{\tau_1, \tau_2, \tau_3\}$, $R_7 = 3$ and $T = \{T_1, T_2, T_3\}$. Let us encode OLC $\alpha_g \in C$ in a trivial way: $K(\alpha_1) = 000, \dots, K(\alpha_6) = 101$. Encoding of components, which satisfies condition (4.4), gives the microinstruction addresses shown in Fig. 6.6.

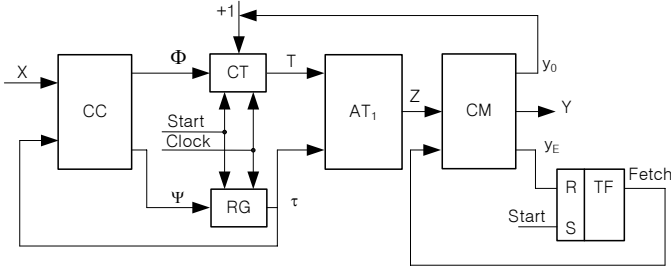


Fig. 6.5 Structural diagram of CMCU U_{22}

$\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110	111
000	b_1	b_3	b_6	b_8	b_{10}	b_{15}	*	*
001	b_2	b_4	b_7	b_9	b_{11}	b_{16}	*	*
010	*	b_5	*	*	b_{12}	b_{17}	*	*
011	*	*	*	*	b_{13}	b_{18}	*	*
100	*	*	*	*	b_{14}	b_{19}	*	*
101	*	*	*	*	*	*	*	*
110	*	*	*	*	*	*	*	*
111	*	*	*	*	*	*	*	*

Fig. 6.6 Microinstruction addresses for CMCU $U_{22}(\Gamma_{11})$

Application of procedure P_2 for this particular case results in microinstruction addresses shown in Fig. 6.7, which are the same as microinstruction addresses of the CMCU $U_1(\Gamma_{11})$.

$z_1z_2z_3$	000	001	010	011	100	101	110	111
00	b_1	b_5	b_9	b_{13}	b_{17}	*	*	*
01	b_2	b_6	b_{10}	b_{14}	b_{18}	*	*	*
10	b_3	b_7	b_{11}	b_{15}	b_{19}	*	*	*
11	b_4	b_8	b_{12}	b_{16}	*	*	*	*

Fig. 6.7 Microinstruction addresses for CMCU $U_1(\Gamma_{11})$

The content of CMCU $U_{22}(\Gamma_{11})$ control memory is identical with the control memory contents of CMCU $U_1 - U_6$ (Table 6.1).

Transition table of CMCU $U_{22}(\Gamma)$ is constructed by analogy to the case of CMCU $U_8(\Gamma)$. In the first step we construct a system of transition formulae for OLC. During the second step, each term of transition formula is replaced by a single line of the transition table. In our case $C^1 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and the system of transition formulae for $a_g \in C^1$ takes the form:

$$\begin{aligned} \alpha_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_3^1 \vee \bar{x}_1 \bar{x}_2 x_3 I_4^1 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 I_6^1; \\ \alpha_2, \alpha_3, \alpha_4 &\rightarrow x_3 I_2^2 \vee \bar{x}_3 x_4 x_5 I_5^2 \vee \bar{x}_3 x_4 \bar{x}_5 I_5^2 \vee \bar{x}_3 x_4 \bar{x}_5 I_5^1 \vee \bar{x}_3 \bar{x}_4 I_6^1. \end{aligned} \quad (6.19)$$

System (6.19) determines the transition table including $H_{22}(\Gamma_{11}) = 16$ lines (Table 6.2). This table is used to construct input memory functions Ψ and Φ , represented by (4.7) and (4.6) respectively. In the case of CMCU $U_{22}(\Gamma_{11})$, the Boolean functions have the form, as for example: $D_5 = F_6 \vee F_{10} \vee F_{14} = \bar{\tau}_2 \bar{\tau}_3 \bar{x}_3 x_4 x_5 \vee \tau_2 \bar{\tau}_3 \bar{x}_3 x_4 x_5 \vee \tau_2 \tau_3 \bar{x}_3 x_4 x_5$. This formula was produced using the property, that codes for all OLC $\alpha_g \in C^1$ include $\tau_1 = 0$.

Table 6.1 Control memory content for CMCU $U_{22}(\Gamma_{11})$

Address $z_1 z_2 z_3 z_4 z_5$	Content	Comments
00000	$y_0 y_1 y_2$	$I_1^1 b_1$
00001	y_3	$O_1 b_2$
00010	$y_0 y_1 y_2$	$I_2^1 b_3$
00011	$y_0 y_1 y_4$	$I_2^2 b_4$
00100	$y_2 y_5$	$O_2 b_5$
00101	$y_0 y_2 y_3 y_4$	$I_3^1 b_6$
00110	$y_1 y_2$	$O_3 b_7$
00111	$y_0 y_2 y_5$	$I_4^1 b_8$
01000	$y_2 y_4$	$O_4 b_9$
01001	$y_0 y_1 y_2$	$I_4^1 b_{10}$
01010	$y_0 y_2 y_5$	b_{11}
01011	$y_0 y_3 y_6$	$I_5^1 b_{12}$
01100	$y_0 y_2 y_4$	b_{13}
01101	$y_1 y_2 y_E$	$O_4 b_{14}$
01110	$y_0 y_3$	$I_6^1 b_{15}$
01111	$y_0 y_2 y_3 y_4$	b_{16}
10000	$y_0 y_2 y_5$	b_{17}
10001	$y_0 y_1 y_4$	b_{18}
10001	$y_2 y_5 y_E$	$O_6 b_{19}$

The address transformer AT_1 is represented by a table with columns b_q , α_g , $K(\alpha_g)$, $K(b_q)$, $A(b_q)$, z_q , q and the operator vertex $b_q \in D^g$ ($g = 1, \dots, G$). In general, this table includes M_2 lines, which in case of the CMCU $U_{22}(\Gamma_{11})$ gives $M_2 = 19$ lines (Table 6.3).

This table serves as the base to construct system (6.8). For example, the equation $z_1 = \tau_1 \bar{\tau}_2 \tau_3 \bar{T}_1 T_2 \bar{T}_3 \vee \tau_1 \bar{\tau}_2 \tau_3 \bar{T}_1 T_2 T_3 \vee \tau_1 \bar{\tau}_2 \tau_3 T_1 \bar{T}_2 \bar{T}_3$ can be obtained from Table 6.3.

Table 6.2 Transition table for CMCU $U_{22}(\Gamma_{11})$

α_g	$K(\alpha_g)$	α_m	$K(\alpha_m)$	I_m^j	$K(I_m^j)$	X_h	Ψ_h	V_h	h
α_1	000	α_2	001	I_2^1	000	x_1	D_3	-	1
		α_3	010	I_3^1	000	\bar{x}_1x_2	D_3	-	2
		α_4	011	I_4^1	000	$\bar{x}_1\bar{x}_2x_3$	D_2D_3	-	3
		α_6	101	I_6^1	000	$\bar{x}_1\bar{x}_2\bar{x}_3$	D_1D_3	-	4
α_2	001	α_2	001	I_2^2	001	x_3	D_3	D_6	5
		α_5	100	I_5^2	010	$\bar{x}_3x_4x_5$	D_1	D_5	6
		α_5	100	I_5^1	000	$\bar{x}_3x_4\bar{x}_5$	D_1	-	7
		α_6	101	I_6^1	000	$\bar{x}_3\bar{x}_4$	D_1D_3	-	8
α_3	010	α_2	001	I_2^2	001	x_3	D_3	D_6	9
		α_5	100	I_5^2	010	$\bar{x}_3x_4x_5$	D_1	D_5	10
		α_5	100	I_5^1	000	$\bar{x}_3x_4\bar{x}_5$	D_1	-	11
		α_6	101	I_6^1	000	$\bar{x}_3\bar{x}_4$	D_1D_3	-	12
α_4	011	α_2	001	I_2^2	001	x_3	D_3	D_6	13
		α_5	100	I_5^2	010	$\bar{x}_3x_4x_5$	D_1	D_5	14
		α_5	100	I_5^1	000	$\bar{x}_3x_4\bar{x}_5$	D_1	-	15
		α_6	101	I_6^1	000	$\bar{x}_3\bar{x}_4$	D_1D_3	-	16

Table 6.3 Table of address transformer for $U_{22}(\Gamma_{11})$

b_q	α_g	$K(\alpha_g)$	$K(b_q)$	$A(b_q)$	z_q	q
b_1	α_1	000	000	00000	-	1
b_2	α_1	000	001	00001	z_5	2
b_3	α_2	001	000	00010	z_4	3
b_4	α_2	001	001	00011	z_4z_5	4
b_5	α_2	001	010	00100	z_3	5
b_6	α_3	010	000	00101	z_3z_5	6
b_7	α_3	010	001	00110	z_3z_4	7
b_8	α_4	011	000	00111	$z_3z_4z_5$	8
b_9	α_4	011	001	01000	z_2	9
b_{10}	α_5	100	000	01001	z_2z_5	10
b_{11}	α_5	100	001	01010	z_2z_4	11
b_{12}	α_5	100	010	01011	$z_2z_4z_5$	12
b_{13}	α_5	100	011	01100	z_2z_3	13
b_{14}	α_5	100	100	01101	$z_2z_3z_5$	14
b_{15}	α_6	101	000	01110	$z_2z_3z_4$	15
b_{16}	α_6	101	001	01111	$z_2z_3z_4z_5$	16
b_{17}	α_6	101	010	10000	z_1	17
b_{18}	α_6	101	011	10001	z_1z_5	18
b_{19}	α_6	101	100	10010	z_1z_4	19

Insignificant input assignments can be used for minimization of functions $z_r \in Z$. The Karnaugh map, generated for our particular case, is shown in Fig. 6.8.

The minimal disjunctive normal form $z_1 = \tau_3\tau_3 \vee \tau_1\tau_2\tau_2$ can be obtained, for example, from the table of Fig. 6.8. This formula corresponds to the first bit of all codes taken from all cells of the Karnaugh map.

$\tau_1\tau_2\tau_3$	000	001	011	010	110	111	101	100
000	00000	00010	00111	00101	*	*	01110	01001
001	00001	00011	01000	00110	*	*	01111	01010
011	*	*	*	*	*	*	10001	01100
010	*	00100	*	*	*	*	10000	01011
110	*	*	*	*	*	*	*	*
111	*	*	*	*	*	*	*	*
101	*	*	*	*	*	*	10010	01101
100	*	*	*	*	*	*	*	*

Fig. 6.8 Karnaugh map for CMCU $U_{22}(\Gamma_{11})$

Synthesis of logic circuit for the CMCU $U_{22}(\Gamma_{11})$ is reduced to the implementation of systems (4.6), (4.7) and (4.8) using PLD chips, and implementation of the control memory CM using either PROM or RAM chips (Fig. 6.9).

By analogy, introduction of address transformer AT_1 converts CMCU U_9 into CMCU U_{23} , CMCU U_{10} into CMCU U_{24} , CMCU U_{11} into CMCU U_{25} . Synthesis methods applied for CMCU $U_{23} - U_{25}$ can be viewed as some modifications of synthesis methods used previously for CMCU $U_9 - U_{11}$ respectively, based on application of procedure P_2 (microinstruction addressing) and on construction of the table of address transformer AT_1 .

Introduction of address transformer AT_1 in CMCU U_{12} leads to the CMCU U_{26} of Fig. 6.10.

Operation principles of both CMCU U_{12} and U_{26} are practically identical, but in case of CMCU U_{26} , address $A(b_q)$ is produced by the address transformer AT_1 . Synthesis method used for CMCU U_{26} is a modification of the one applied for CMCU U_{12} , which consists on using procedure P_2 and the table of address transformer AT_1 . Let us discuss an example, in which this method is applied to the CMCU $U_{26}(\Gamma_{11})$. The set of elementary OLC C_E for the case of GSA Γ_{11} was constructed in Section 6.1, where we obtained: $R_9 = 3; R_7 = 3; R_2 = 5$.

Let us encode elementary OLC $\alpha_g \in C_E$ in a trivial way: $K(\alpha_1) = 000, \dots, K(\alpha_8) = 111$. Encoding of components satisfying condition (4.4) leads to microinstruction addresses for the CMCU $U_{26}(\Gamma_{11})$, shown in Fig. 6.11.

Application of procedure P_2 results in microinstruction addresses shown in Fig. 6.7, and the control memory content for the CMCU $U_{26}(\Gamma_{11})$ is the same as for CMCU $U_{22}(\Gamma_{11})$, and is given in Table 6.1.

Transition table for CMCU $U_{26}(\Gamma)$ is constructed using the same approach as the one, used for CMCU $U_{12}(\Gamma)$. The first step is connected with construction of transition formulae for elementary OLC $\alpha_g \in C_E^1$. The second step consists on replacement of each term of the equation system by a single line of transition table. In

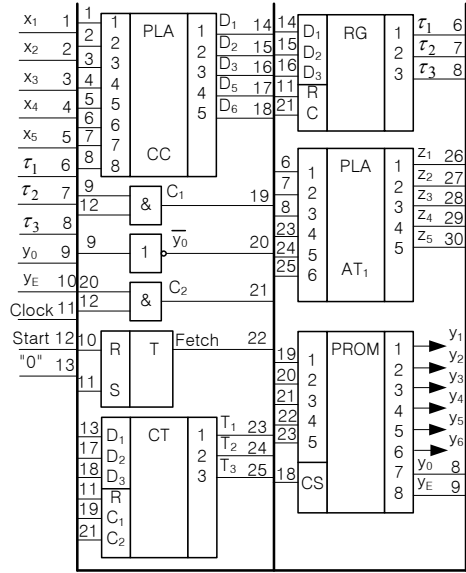


Fig. 6.9 Logic circuit of CMCU U₂₂(I₁₁)

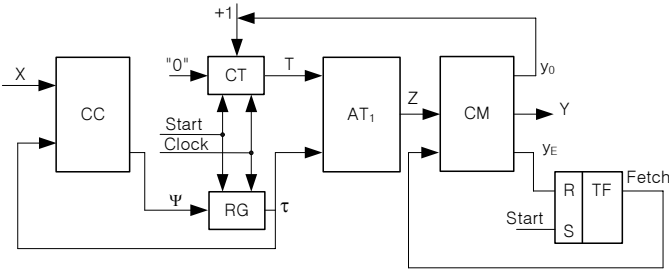


Fig. 6.10 Structural diagram of CMCU U₂₆

case of CMCU U₂₆(I₁₁) we have the set $C_E^1 = \{\alpha_1, \dots, \alpha_6\}$ and system of transition formulae:

$$\begin{aligned}
 \alpha_1 &\rightarrow x_1 I_2 \vee \bar{x}_1 x_2 I_4 \vee \bar{x}_1 \bar{x}_2 x_3 I_5 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 I_8; \\
 \alpha_2 &\rightarrow I_3 \\
 \alpha_3, \alpha_4, \alpha_5 &\rightarrow x_3 I_3 \vee \bar{x}_3 x_4 x_5 I_7 \vee \bar{x}_3 x_4 \bar{x}_5 I_6 \vee \bar{x}_3 x_4 \bar{x}_5 I_5^1 \vee \bar{x}_3 \bar{x}_4 I_8; \\
 \alpha_6 &\rightarrow I_7.
 \end{aligned}
 \tag{6.20}$$

Transition table includes the columns $\alpha_g, K(\alpha_g), \alpha_m, K(\alpha_m), X_h, \Psi_h, h$, and in case of the CMCU U₂₆(I₁₁) it has $H_{26}(I_{11}) = 18$ lines (Table 6.4).

This table serves to construct the input memory functions Ψ . For example, from Table 6.4 we get: $D_1 = F_3 \vee F_4 \vee F_7 \vee F_8 \vee F_9 \vee F_{11} \vee F_{12} \vee F_{13} \vee F_{15} \vee \dots \vee F_{18} = \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 \bar{x}_1 \bar{x}_2 x_3 \vee \dots \vee \tau_1 \bar{\tau}_2 \tau_3$.

$\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110	111
000	b_1	b_3	b_4	b_6	b_8	b_{10}	b_{12}	b_{15}
001	b_2	*	b_5	b_7	b_9	b_{11}	b_{13}	b_{16}
010	*	*	*	*	*	*	b_{14}	b_{17}
011	*	*	*	*	*	*	*	b_{18}
100	*	*	*	*	*	*	*	b_{19}
101	*	*	*	*	*	*	*	*
110	*	*	*	*	*	*	*	*
111	*	*	*	*	*	*	*	*

Fig. 6.11 Microinstruction addresses for CMCU $U_{26}(\Gamma_{11})$

As in case of CMCU U_{22} , address transformer is represented by a table with the columns: $b_q, \alpha_g, K(\alpha_g), K(b_q), A(b_q), z_q, q$. In case of CMCU $U_{26}(\Gamma_{11})$ this table includes $M_2 = 19$ lines (Table 6.5).

In Table 6.5, some bits of codes $K(b_q)$ are marked by symbols *. These symbols correspond to insignificant input assignments for component codes of some EOLC. These assignments allow minimizing the number of literals in the disjunctive normal forms of functions $z_r \in Z$. This table allows to derive functions (6.8). The equation $z_1 = \tau_1 \tau_2 \tau_3 T_2 \vee \tau_1 \tau_2 \tau_3 T_1$ can be found, for example, from Table 6.5 (after minimization).

Synthesis of logic circuit for CMCU $U_{26}(\Gamma)$ is reduced to the implementation of systems (4.6), (4.7) and (6.8) using PLD chips, and implementation of control memory using either PROM or RAM chips.

Introduction of address transformer AT_1 into CMCU $U_{13} - U_{15}$ converts them into CMCU $U_{27} - U_{29}$ respectively. Structural diagrams of CMCU with code sharing and transformation of microinstruction addresses $A(b_q)$ from the form (6.7) into corresponding addresses with R_2 bits are shown in Table 6.6.

Let us point out that only four models among all shown in Table 6.6, namely $U_{22}, U_{24}, U_{26}, U_{28}$ are original ones. All other models are some modifications of them, obtained either using different approach to encoding of OLC (U_{23}), or elementary OLC (U_{27}), or using different method of GSA transformation (U_{25}, U_{29}). The variables used to encode OLC (U_{24}) or EOLC (U_{28}) form a set W , and elements of the set Z are used to address microinstructions kept in the control memory CM.

Table 6.4 Transition table of CMCU $U_{26}(\Gamma_{11})$

α_g	$K(\alpha_g)$	α_m	$K(\alpha_m)$	X_h	Ψ_h	h
α_1	000	α_2	001	x_1	D_3	1
		α_4	011	\bar{x}_1x_2	D_2D_3	2
		α_5	100	$\bar{x}_1\bar{x}_2x_3$	D_1	3
		α_8	111	$\bar{x}_1\bar{x}_2\bar{x}_3$	$D_1D_2D_3$	4
α_2	001	α_3	010	1	D_2	5
α_3	010	α_3	010	x_3	D_2	6
		α_7	110	$\bar{x}_3x_4x_5$	D_1D_2	7
		α_6	101	$\bar{x}_3x_4\bar{x}_5$	D_1D_3	8
		α_8	111	$\bar{x}_3\bar{x}_4$	$D_1D_2D_3$	9
α_4	011	α_3	010	x_3	D_2	10
		α_7	110	$\bar{x}_3x_4x_5$	D_1D_2	11
		α_6	101	$\bar{x}_3x_4\bar{x}_5$	D_1D_3	12
		α_8	111	$\bar{x}_3\bar{x}_4$	$D_1D_2D_3$	13
α_5	100	α_3	010	x_3	D_2	14
		α_7	110	$\bar{x}_3x_4x_5$	D_1D_2	15
		α_6	101	$\bar{x}_3x_4\bar{x}_5$	D_1D_3	16
		α_8	111	$\bar{x}_3\bar{x}_4$	$D_1D_2D_3$	17
α_6	101	α_8	110	1	D_1D_2	18

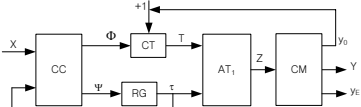
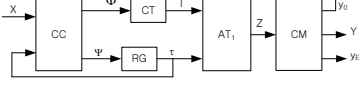
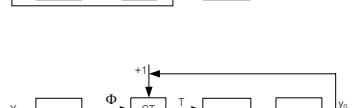
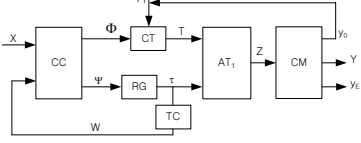
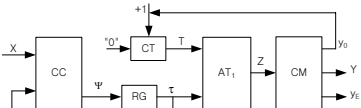
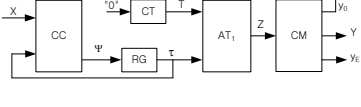
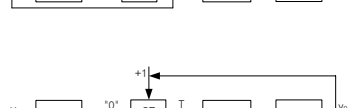
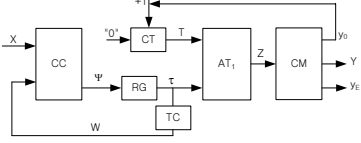
Table 6.5 Table of address transformer for CMCU $U_{26}(\Gamma_{11})$

b_q	α_g	$K(\alpha_g)$	$K(b_q)$	$A(b_q)$	z_q	q
b_1	α_1	000	**0	00000	-	1
b_2	α_1	000	**1	00001	z_5	2
b_3	α_2	001	***	00010	z_4	3
b_4	α_3	010	**0	00011	z_4z_5	4
b_5	α_3	010	**1	00100	z_3	5
b_6	α_4	011	**0	00101	z_3z_5	6
b_7	α_4	011	**1	00110	z_3z_4	7
b_8	α_5	100	**0	00111	$z_3z_4z_5$	8
b_9	α_5	100	**1	01000	z_2	9
b_{10}	α_6	101	**0	01001	z_2z_5	10
b_{11}	α_6	101	**1	01010	z_2z_4	11
b_{12}	α_7	110	*00	01011	$z_2z_4z_5$	12
b_{13}	α_7	110	**1	01100	z_2z_3	13
b_{14}	α_7	110	*1*	01101	$z_2z_3z_5$	14
b_{15}	α_8	111	000	01110	$z_2z_3z_4$	15
b_{16}	α_8	111	*01	01111	$z_2z_3z_4z_5$	16
b_{17}	α_8	111	*10	10000	z_1	17
b_{18}	α_8	111	*11	10001	z_1z_5	18
b_{19}	α_8	111	1**	10010	z_1z_4	19

6.3 Synthesis of CMCU with addressing of expanded microinstructions

Introduction of the address transformer AT_2 into CMCU U_8 gives the CMCU U_{30} with structural diagram shown in Fig. 6.12.

Table 6.6 Structural diagrams of CMCU $U_{22} - U_{29}$

**	Structural diagram	Comments
U_{22}		Analogue of U_8
U_{23}		Analogue of U_9 , set $\tau' \subseteq \tau$ is input of CC
U_{25}		Analogue of U_{11}
U_{24}		Analogue of $U_{10} W = R_4$
U_{26}		Analogue of U_{12}
U_{27}		Analogue of U_{13} , set $\tau' \subseteq \tau$ is input of CC
U_{29}		Analogue of U_{15}
U_{28}		Analogue of $U_{14} W = R_{11}$

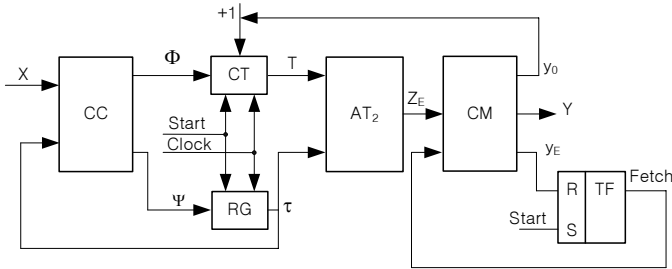


Fig. 6.12 Structural diagram of CMCU U_{30}

Operation principles of both CMCU U_{30} and U_8 are the same, but address transformer AT_2 generates code $K_E(b_q)$ of the expanded microinstruction on the base of address $A(b_q)$, represented in the form (6.7). In case of GSA Γ_{11} , the set of expanded microinstructions $Y_E(\Gamma_{11})$ includes the elements: $Y_1 = \{y_0, y_1, y_2\}$, $Y_2 = \{y_1, y_2\}$, $Y_3 = \{y_1, y_2, y_E\}$, $Y_4 = \{y_0, y_3\}$, $Y_5 = \{y_3\}$, $Y_6 = \{y_0, y_1, y_4\}$, $Y_7 = \{y_0, y_2, y_5\}$, $Y_8 = \{y_2, y_5\}$, $Y_9 = \{y_2, y_5, y_E\}$, $Y_{10} = \{y_0, y_2, y_3, y_4\}$, $Y_{11} = \{y_2, y_4\}$, $Y_{12} = \{y_0, y_2, y_4\}$, $Y_{13} = \{y_0, y_3, y_6\}$. Therefore, $M_3 = 13$, $R_{15} = 4$, condition (6.10) is satisfied, and

this method can be successfully used for the GSA Γ_{11} . Synthesis method used for CMCU U_{30} is a modification of the one applied for CMCU U_8 , when the steps of expanded microinstruction encoding and construction of the table of address transformer AT_2 are added.

By analogy, introduction of address transformer AT_2 into CMCU U_9 results in CMCU U_{31} , having the same structural diagram as CMCU U_{30} . There is one difference however, namely feedback inputs τ' of the combinational circuit CC form some subset of the set τ . Synthesis method used for CMCU U_{31} has the following steps:

1. Transformation of initial GSA Γ (procedure P_4).
2. Construction of the set of OLC (procedure P_1).
3. Construction of the partition Π_C for OLC set C^1 .
4. Optimal encoding of operational linear chains $\alpha_g \in C^1$ and arbitrary encoding of OLC $\alpha_g \notin C^1$.
5. Encoding of components for OLC $\alpha_g \in C$.
6. Encoding of expanded microinstructions.
7. Construction of the control memory content.
8. Construction of transition table for CMCU.
9. Construction of the table for address transformer AT_2 .
10. Synthesis of CMCU logic circuit with given logic elements.

Let us discuss application of this method to the CMCU $U_{31}(\Gamma_{11})$, remembering that the OLC set for the GSA Γ_{11} was already found in Section 6.1.

Partition of the set $C^1 = \{\alpha_1, \dots, \alpha_4\}$ results in the set $\Pi_C = \{B_1, B_2\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$. Result of optimal encoding of OLC $\alpha_g \in C$ is shown in Fig. 6.13. Now, each class $B_i \in \Pi_C$ corresponds to a single generalized interval of a three-dimensional Boolean space.

$\tau_2\tau_3$ τ_1	00	01	11	10
0	α_1	α_5	α_6	*
1	α_2	α_3	α_4	*

Fig. 6.13 Encoding of EOLC for CMCU $U_{31}(\Gamma_{11})$

In this case OLC $\alpha_5, \alpha_6 \notin C^1$ and corresponding codes can be used to minimize the codes $K(B_1)$ and $K(B_2)$. Taking this possibility, we obtain the codes: $K(B_1) = 0**$ and $K(B_2) = 1**$; and in consequence $\tau' = \{\tau_1\}$.

Components of OLC $\alpha_g \in C$ are encoded in a trivial way and corresponding microinstruction addresses are shown in Fig. 6.14.

Encoding of expanded microinstructions is now performed, using the state encoding method presented in [2], namely: the more operator vertices includes expanded microinstruction $Y_m \in Y_E(\Gamma)$ the more zeros its code includes. Let $n_m = |Y_m|$, and the following values of parameter n_m can be found in case of the GSA Γ_{11} :

$\tau_1\tau_2\tau_3$		000	001	010	011	100	101	110	111
$T_1T_2T_3$	000	b_1	b_{10}	*	b_{15}	b_3	b_6	*	b_8
	001	b_2	b_{11}	*	b_{16}	b_4	b_7	*	b_9
	010	*	b_{12}	*	b_{17}	b_5	*	*	*
	011	*	b_{13}	*	b_{18}	*	*	*	*
	100	*	b_{14}	*	b_{19}	*	*	*	*
	101	*	*	*	*	*	*	*	*
	110	*	*	*	*	*	*	*	*
	111	*	*	*	*	*	*	*	*

Fig. 6.14 Microinstruction addresses for CMCU $U_{31}(I_{11})$

$n_1 = 3, n_2 = n_3 = 1, n_4 = n_5 = 1, n_6 = 2, n_7 = 3, n_8 = 1, n_9 = 1, n_{10} = 2, n_{11} = 1, n_{12} = n_{13} = 1.$

The following codes of expanded microinstructions for the CMCU $U_{31}(I_{11})$ were obtained using the approach given above (Fig. 6.15).

z_1z_2		00	01	11	10
z_3z_4	00	Y_1	Y_7	Y_9	Y_6
	01	Y_{10}	Y_3	Y_{11}	Y_4
	11	Y_5	Y_{12}	*	*
	10	Y_2	Y_{13}	*	Y_8

Fig. 6.15 Encoding of expanded microinstructions for CMCU $U_{31}(I_{11})$

These codes represent the control memory addresses of expanded microinstructions. The control memory content is constructed by replacement of the symbols of expanded microinstructions by corresponding collections of microoperations $y_n \in Y$ and variables y_0, y_E (Fig. 6.16).

z_1z_2		00	01	11	10
z_3z_4	00	$y_0y_1y_2$	$y_0y_2y_5$	$y_0y_1y_4$	$y_2y_5y_E$
	01	$y_0y_2y_3y_4$	$y_1y_2y_E$	y_0y_3	y_2y_4
	11	y_1y_2	$y_0y_3y_6$	y_2y_5	*
	10	y_3	$y_0y_2y_4$	*	*

Fig. 6.16 Control memory content for CMCU $U_{31}(I_{11})$

Construction of transition table for CMCU U_{31} is executed using the same method as the one used for CMCU U_9 . In our particular case system (6.19) is replaced by the following system of transition formulae:

$$\begin{aligned} B_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_3^1 \vee \bar{x}_1 \bar{x}_2 x_3 I_4^1 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 I_6^1; \\ B_2 &\rightarrow x_3 I_2^2 \vee \bar{x}_3 x_4 x_5 I_5^2 \vee \bar{x}_3 x_4 \bar{x}_5 I_5^1 \vee \bar{x}_3 \bar{x}_4 I_6^1, \end{aligned} \quad (6.21)$$

which corresponds to the transition table with $H_{31}(\Gamma_{11}) = 8$ lines (Table 6.7).

Table 6.7 Transition table for CMCU $U_{31}(\Gamma_{11})$

B_i	$K(B_i)$	α_m	$K(\alpha_m)$	I_m^j	$K(I_m^j)$	X_h	Ψ_h	Φ_h	h
B_1	0**	α_2	100	I_2^1	000	x_1	D_1	-	1
		α_3	101	I_3^1	000	$\bar{x}_1 x_2$	$D_1 D_3$	-	2
		α_4	111	I_4^1	000	$\bar{x}_1 \bar{x}_2 x_3$	$D_1 D_2 D_3$	-	3
		α_6	011	I_6^1	000	$\bar{x}_1 \bar{x}_2 \bar{x}_3$	$D_2 D_3$	-	4
B_2	1**	α_2	100	I_2^2	001	x_3	D_1	D_6	5
		α_5	001	I_5^2	010	$\bar{x}_3 x_4 x_5$	D_3	D_5	6
		α_5	001	I_5^1	000	$\bar{x}_3 x_4 \bar{x}_5$	D_3	-	7
		α_6	011	I_6^1	000	$\bar{x}_3 \bar{x}_4$	$D_2 D_3$	-	8

This table allows to construct disjunctive normal forms of functions (4.6) and (4.7). The equations $D_3 = \bar{\tau}_1 \bar{x}_1 \vee \tau_1 \bar{x}_3$; $D_5 = \tau_1 \bar{x}_3 x_4 x_5$; can be found, for example, from Table 6.7 (after minimization).

Address transformer AT_2 generates functions (6.12), and its table includes M_2 lines and the columns: b_q , α_g , $K(\alpha_g)$, $K(b_q)$, $K(Y_E)$, z_q , q . Column $K(Y_E)$ of this table contains code of an expanded microinstruction corresponding to some vertex $B_q \in B_2$. In case of CMCU $U_{31}(\Gamma_{11})$ this table includes $M_2 = 19$ lines (Table 6.8).

Synthesis of logic circuit of CMCU U_{31} is reduced to implementation of systems (4.6), (4.7) and (6.12), using PLD chips, and implementation of control memory using either PROM or RAM chips. Logic circuit of the CMCU $U_{31}(\Gamma_{11})$ is shown in Fig. 6.17.

In this particular case, it could be found that $V_{\min} = 32 \times 8 = 256$ bits, and the address transformer AT_2 allows application of code sharing approach as well as reduction of the control memory size up to 128 bits.

Adding the address transformer AT_2 to the CMCU U_{10} we obtain CMCU U_{32} , and adding it to the CMCU U_{11} results in the CMCU U_{33} . Let us point out that introduction of the address transformer AT_2 to any of CMCU $U_8 - U_{11}$ gives reduction of the control memory size in the particular case of GSA Γ_{11} .

Introduction of the address transformer AT_2 into CMCU U_{12} results in CMCU U_{34} , with structural diagram shown in Fig. 6.18.

Operation principles of both CMCU U_{34} and U_{12} are the same, but in the former the address transformer generates code $K_E(b_q)$ of expanded microinstruction on the base of address $A(b_q)$, represented in the form (6.7). Synthesis method used for CMCU U_{34} is a modification of the method applied in case of CMCU U_{12} . This

Table 6.8 Table of address transformer AT_2 for CMCU $U_{31}(I_{11})$

b_q	α_g	$K(\alpha_g)$	$K(b_q)$	$K(Y_E)$	z_q	q
b_1	α_1	000	**0	0000	—	1
b_2	α_1	000	**1	0011	z_3z_4	2
b_3	α_2	100	*00	0000	—	3
b_4	α_2	100	**1	1000	z_1	4
b_5	α_2	100	*1*	1010	z_1z_3	5
b_6	α_3	101	**0	0001	z_4	6
b_7	α_3	101	**1	0010	z_3	7
b_8	α_4	111	**0	0100	z_2	8
b_9	α_4	111	**1	1101	$z_1z_2z_4$	9
b_{10}	α_5	001	000	0000	—	10
b_{11}	α_5	001	*01	0100	z_2	11
b_{12}	α_5	001	*10	0110	z_2z_3	12
b_{13}	α_5	001	*11	0111	$z_2z_3z_4$	13
b_{14}	α_5	001	1**	0101	z_2z_4	14
b_{15}	α_6	011	000	1001	z_1z_4	15
b_{16}	α_6	011	*01	0001	z_4	16
b_{17}	α_6	011	*01	0100	z_2	17
b_{18}	α_6	011	*11	1000	z_1	18
b_{19}	α_6	011	1**	1100	z_1z_2	19

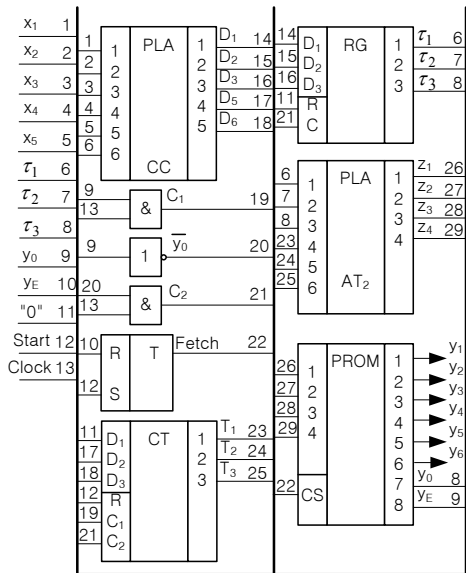


Fig. 6.17 Logic circuit of CMCU $U_{31}(I_{11})$

modification consists on adding the steps of encoding of expanded microinstructions and construction of the table of address transformer AT_2 .

By analogy, introducing of address transformer AT_2 into CMCU U_{13} gives the CMCU U_{35} , having the same structural diagram as in case of CMCU U_{34} . The

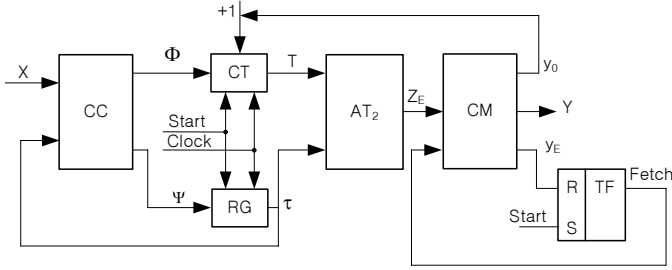


Fig. 6.18 Structural diagram of CMCU U_{34}

variables $\tau' \subseteq \tau$ are used as feedback signals in case of CMCU U_{35} . Synthesis of CMCU U_{35} includes the following steps:

1. Transformation of initial GSA (procedure P_4).
2. Construction of the set of elementary OLC C_E (procedure P_{11}).
3. Construction of partition Π_E for the set $C_E^1 \subset C_E$.
4. Optimal encoding of elementary OLC $\alpha_g \in C_E^1$ and arbitrary encoding of other elementary OLC $\alpha_g \notin C_E^1$.
5. Encoding of the components of elementary OLC $\alpha_g \in C_E$.
6. Encoding of expanded microinstructions.
7. Construction of the control memory content.
8. Construction of transition table for CMCU.
9. Construction of the table for address transformer AT_2 .
10. Synthesis of CMCU logic circuit with given logical elements.

Let us discuss application of this method for synthesis of the CMCU $U_{35}(\Gamma_{11})$ where the EOLC set for the GSA Γ_{11} was found already in Section 6.1.

In case of the GSA $\Gamma_{11}(U_{35})$, partition $\Pi_E = \{B_1, \dots, B_4\}$ can be formed, in which $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2\}$, $B_3 = \{\alpha_3, \alpha_4, \alpha_5\}$, $B_4 = \{\alpha_6\}$, and the set $C_E^1 = \{\alpha_1, \dots, \alpha_6\}$. Corresponding codes of the elementary OLC $\alpha_g \in C_E$ are shown in Fig. 6.19.

$\tau_2 \tau_3$		00	01	11	10
τ_1	0	α_1	α_2	α_8	α_6
	1	α_3	α_4	α_5	α_7

Fig. 6.19 Encoding of EOLC for CMCU $U_{35}(\Gamma_{11})$

Input assignments 011 and 110 are considered here as insignificant ones, because $\alpha_7, \alpha_8 \notin C_E^1$. These assignments give the codes of classes $B_i \in \Pi_E$: $K(B_1) = 000$, $K(B_2) = 0*1$, $K(B_3) = 1**$, $K(B_4) = 01*$. Encoding of EOLC components is executed in a traditional style. In our particular case microinstruction addresses for CMCU $U_{35}(\Gamma_{11})$ are shown in Fig. 6.20.

	$\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110	111
$T_1T_2T_3$	000	b_1	b_3	b_4	b_6	b_8	b_{10}	b_{12}	b_{15}
	001	b_2	*	b_5	b_7	b_9	b_{11}	b_{13}	b_{16}
	010	*	*	*	*	*	*	b_{14}	b_{17}
	011	*	*	*	*	*	*	*	b_{18}
	100	*	*	*	*	*	*	*	b_{19}
	101	*	*	*	*	*	*	*	*
	110	*	*	*	*	*	*	*	*
	111	*	*	*	*	*	*	*	*

Fig. 6.20 Microinstruction addresses for CMCU $U_{35}(\Gamma_{11})$

Sets of expanded microinstructions are the same for both the CMCU $U_{35}(\Gamma_{11})$ and $U_{31}(\Gamma_{11})$. Codes of expanded microinstructions for the CMCU $U_{35}(\Gamma_{11})$ are shown in Fig. 6.15, and its control memory content in Fig. 6.16.

Transition table for CMCU $U_{35}(\Gamma)$ is constructed using transition formulae of EOLC $\alpha_g \in C_E^1$. In our particular case, system (6.20) is replaced by the following system:

$$\begin{aligned}
 B_1 &\rightarrow x_1I_2 \vee \bar{x}_1x_2I_4 \vee \bar{x}_1\bar{x}_2x_3I_5 \vee \bar{x}_1\bar{x}_2\bar{x}_3I_8; \\
 B_2 &\rightarrow I_3; \\
 B_3 &\rightarrow x_3I_3 \vee \bar{x}_3x_4x_5I_7 \vee \bar{x}_3x_4\bar{x}_5I_6 \vee \bar{x}_3\bar{x}_4I_8; \\
 B_4 &\rightarrow I_7.
 \end{aligned}
 \tag{6.22}$$

Transition table of CMCU $U_{35}(\Gamma)$ includes the columns $B_i, K(B_i), \alpha_g, K(\alpha_g), X_h, \Psi_h, h$; and the CMCU $U_{35}(\Gamma_{11})$ includes $H_{35}(\Gamma_{11}) = 10$ lines (Table 6.9).

Table 6.9 Transition table for CMCU $U_{35}(\Gamma_{11})$

B_i	$K(B_i)$	α_g	$K(\alpha_g)$	X_h	Ψ_h	h	
B_1	000	α_2	001	x_1	D_1	1	
			α_4	101	\bar{x}_1x_2	D_1D_3	2
			α_5	111	$\bar{x}_1\bar{x}_2x_3$	$D_1D_2D_3$	3
			α_8	011	$\bar{x}_1\bar{x}_2\bar{x}_3$	D_2D_3	4
B_2	$0*1$	α_3	100	1	D_1	5	
B_3	$1**$	α_3	100	x_3	D_1	6	
			α_7	110	$\bar{x}_3x_4x_5$	D_1D_2	7
			α_6	010	$\bar{x}_3x_4\bar{x}_5$	D_2	8
			α_8	011	$\bar{x}_3\bar{x}_4$	D_2D_3	9
B_4	$01*$	α_7	110	1	D_1D_2	10	

This table allows to obtain the equations for input memory functions Ψ . For example, the equation $D_3 = \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 \vee \tau_1 \bar{x}_3 \bar{x}_4$ can be extracted from lines 1–4 and 9 of Table 6.9 (after minimization). All other functions of system (4.6) can be found using the same approach.

Address transformer AT_2 implements functions of the system (6.12), and the table of AT_2 includes the columns: b_q , α_g , $K(\alpha_g)$, $K(b_q)$, $K(Y_E)$, z_q , q . In case of CMCU $U_{35}(\Gamma_{11})$ this table has $M_2 = 19$ lines (Table 6.10). As in previous case, insignificant input assignments were used for each EOLC $\alpha_g \in C_E$ to reduce the number of literals in the disjunctive normal forms of functions (6.12).

Table of address transformer AT_2 gives the functions $z_r \in Z_E$. For example, the following expression can be found from Table 6.9: $z_1 = \tau_1 \bar{\tau}_2 \bar{\tau}_3 \vee \tau_1 \tau_2 \tau_3 T_3 \vee \bar{\tau}_1 \tau_2 \tau_3 \bar{T}_1 \bar{T}_2 \bar{T}_3 \vee \tau_1 \tau_2 \tau_3 T_2 T_3 \vee \bar{\tau}_1 \tau_2 \tau_3 T_1$. This equation corresponds to lines 4, 5, 9, 15, 18 and 19 of the table.

Table 6.10 Table of address transformer AT_2 for CMCU $U_{35}(\Gamma_{11})$

b_q	α_g	$K(\alpha_g)$	$K(b_q)$	$K(Y_E)$	z_q	q
b_1	α_1	000	**0	0000	–	1
b_2	α_1	000	**1	0011	$z_3 z_4$	2
b_3	α_2	001	***	0000	–	3
b_4	α_3	100	**0	1000	z_1	4
b_5	α_3	100	**1	1010	$z_1 z_3$	5
b_6	α_4	101	**0	0001	z_4	6
b_7	α_4	101	**1	0010	z_3	7
b_8	α_5	111	**0	0100	z_2	8
b_9	α_5	111	**1	1101	$z_1 z_2 z_4$	9
b_{10}	α_6	010	**0	0000	–	10
b_{11}	α_6	010	**1	0100	z_2	11
b_{12}	α_7	110	*00	0110	$z_2 z_3$	12
b_{13}	α_7	110	**1	0111	$z_2 z_3 z_4$	13
b_{14}	α_7	110	*1*	0101	$z_2 z_4$	14
b_{15}	α_8	011	000	1001	$z_1 z_4$	15
b_{16}	α_8	011	*01	0001	z_4	16
b_{17}	α_8	011	*10	0100	z_2	17
b_{18}	α_8	011	*11	1000	z_1	18
b_{19}	α_8	011	1**	1100	$z_1 z_2$	19

Synthesis of CMCU U_{35} logic circuit is reduced to the implementation of systems (4.6) and (6.12) using PLD chips and implementation of the control memory using either PROM or RAM chips. Let us point out that in case of CMCU $U_{35}(\Gamma_{11})$, the combinational circuit CC has 8 inputs (more, than the equivalent CMCU $U_{31}(\Gamma_{11})$) and 3 outputs (less, than the equivalent CMCU $U_{31}(\Gamma_{11})$). The numbers of inputs and outputs of the address transformer AT_2 are the same for both CMCUs mentioned above.

Application of the address transformer AT_2 in CMCU U_{14} results in obtaining CMCU U_{36} , and its application to the CMCU U_{15} results in CMCU U_{37} . Structural

diagrams for models of CMCU $U_{30} - U_{37}$ and their short characteristics are shown in Table 6.10. Let us point out that there are only four original models here; all other can be considered as their modifications due to OLC optimal encoding, or to the introduction of extra OLC, or to optimal encoding and introduction of extra elementary OLC.

Analysis of Tables 6.6 and 6.10 shows that models $U_{22} - U_{29}$ differ from corresponding models $U_{30} - U_{37}$ only by replacement of the address transformer AT_1 by address transformer AT_2 , and of the set Z by set Z_E .

6.4 Synthesis of CMCU with generation of addresses of collections of microoperations

Introduction of the address transformer AT_3 , performing transformation of the microinstruction address $A(b_q)$ into address $C(b_q)$ of collection of microoperations, and of the block generating the control signals CCS, allows conversion of models $U_8 - U_{15}$ into the models of CMCU $U_{38} - U_{45}$ respectively (Table 6.12). Main difference between models $U_{38} - U_{45}$ and their analogues is, that the control memory CM keeps only microoperations $y_n \in Y$, whereas additional variables y_0 (synchronization control) and Y_E (fetching control) are generated by an additional block CCS.

Synthesis methods used for CMCU $U_{38} - U_{45}$ represent some modifications of synthesis methods applied for their analogues $U_8 - U_{15}$ respectively, obtained through the following additional steps:

- construction of the table of address transformer AT_3 ;
- construction of the table of block CCS;
- encoding of collections of microoperations.

For example, combined application of the code transformer TC and address transformer AT_3 for the CMCU with code sharing results in the CMCU $U_{40}(\Gamma)$.

Synthesis method used for CMCU $U_{40}(\Gamma)$ includes the following steps:

1. Transformation of initial GSA (procedure P_4).
2. Construction of the set OLC C for the transformed GSA $\Gamma(U_{40})$.
3. Construction of partition Π_C for the set of OLC C^1 .
4. Encoding of OLCs $\alpha_g \in C$ and their components.
5. Encoding of the equivalence classes $B_i \in \Pi_C$.
6. Encoding of collections of microoperations.
7. Construction of the control memory content.
8. Construction of the transition table for CMCU.
9. Construction of the table of code transformer TC.
10. Construction of the table of address transformer AT_3 .
11. Construction of the table of CCS block.
12. Synthesis of CMCU logic circuit with given logical elements.

Let us discuss example of synthesis of the CMCU $U_{40}(\Gamma_{11})$. As was found in Section 6.1, OLC set for the GSA Γ_{11} includes six elements: $C = \{\alpha_1, \dots, \alpha_6\}$, where $C^1 = \{\alpha_1, \dots, \alpha_4\}$. Partition $\Pi_C = \{B_1.B_2\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, was also obtained. In our example we use the codes of OLC $\alpha_g \in C$ and their components, shown in Fig. 6.6.

In this case we find $I = 2$ classes $B_i \in \Pi_C$ which can be encoded using one variable ($R_4 = 1$), and in consequence $W = \{w_1\}$. The class B_2 includes maximum number of elements, and therefore we encode the classes $B_i \in \Pi_C$ as: $K(B_1) = 1$, $K(B_2) = 0$. In general case, encoding of classes $B_i \in \Pi_C$, follows the rule: the more elements a class includes, the more zeros its code contains.

The following collections of microoperations can be derived from operator vertices of the initial GSA Γ_{11} : $Y_1 = \{y_1, y_2\}$, $Y_2 = \{y_3\}$, $Y_3 = \{y_1, y_4\}$, $Y_4 = \{y_2, y_5\}$, $Y_5 = \{y_2, y_3, y_4\}$, $Y_6 = \{y_2, y_4\}$, $Y_7 = \{y_3, y_6\}$. It means that $M_4 = 7$, $R_{16} = 3$, condition (6.14) holds, and the control memory size can be reduced 4 times in comparison with V_{\min} (according to (6.15)). The following procedure will now be used to optimize logic circuit of the address transformer AT_3 . First, we find the value

Table 6.11 Structural diagrams of CMCU $U_{30} - U_{37}$

Type	Structural diagram	Comments
U_{30}		Analogue of CMCU U_8
U_{31}		Analogue of CMCU U_9 , set $\tau' \subseteq \tau$ is CC input
U_{33}		Analogue of CMCU U_{11}
U_{32}		Analogue of CMCU U_{10} $ W = R_4$
U_{34}		Analogue of CMCU U_{12}
U_{35}		Analogue of CMCU U_{13} , set $\tau' \subseteq \tau$ is CC input
U_{37}		Analogue of CMCU U_{151}
U_{36}		Analogue of U_{14} $ W = R_{11}$

Table 6.12 Structural diagrams of CMCU $U_{38} - U_{45}$

Type	Structural diagram	Comments
U_{38}		Analogue of CMCU U_8
U_{39}		Analogue of CMCU U_9 , set $\tau' \subseteq \tau$ is CC input
U_{41}		Analogue of CMCU U_{11}
U_{40}		Analogue of $U_{10} \mid W \mid = R_4$
U_{42}		Analogue of CMCU U_{12}
U_{43}		Analogue of CMCU U_{13} , set $\tau' \subseteq \tau$ is CC input
U_{45}		Analogue of CMCU U_{15}
U_{44}		Analogue of $U_{14} \mid W \mid = R_{11}$

of parameter n_q , which is equal to the number of operator vertices with collection $Y_q (q = 1, \dots, M_4)$. Next, we make the queue of collections, in the order of decreasing parameter n_q , and finally we encode the collections using the rule mentioned above: the greater is the value of n_q , the more zeros contains the code $K(Y_q)$ of the collection of microoperations $Y_q (q = 1, \dots, M_4)$.

In our current example, parameters n_q have the values: $n_1 = 5, n_2 = 2, n_3 = 2, n_4 = 5, n_5 = 2, n_6 = 2, n_7 = 1$. It allows us to form the queue $\langle Y_1, Y_4, Y_2, Y_3, Y_5, Y_6, Y_7 \rangle$ for encoding collections of microoperations $Y_q \subseteq Y$ for the CMCU $U_{40}(I_{11})$. It results in the codes shown in Fig. 6.21.

Fig. 6.21 Encoding of collections of microoperations for CMCU $U_{40}(\Gamma_{11})$

		z_2z_3			
		00	01	11	10
z_1	0	Y_1	Y_4	Y_5	Y_2
	1	Y_3	Y_6	*	Y_7

Construction of the control memory content is reduced here to the replacement of symbols Y_q by corresponding microoperations $y_n \in Y_q (q = 1, \dots, M_4)$. The corresponding control memory content is shown in Fig. 6.22.

Fig. 6.22 Content of control memory for CMCU $U_{40}(\Gamma_{11})$

		z_2z_3			
		00	01	11	10
z_1	0	y_1y_2	y_2y_5	$y_2y_3y_4$	y_3
	1	y_1y_4	y_2y_4	*	y_3y_6

Transition table is constructed using the same approach as the one used for CMCU U_{10} . In this case, transition table is found using the system of transition formulae (6.21) and includes $H_{40}(\Gamma_{11}) = 8$ lines (Table 6.13).

Table 6.13 Transition table for CMCU $U_{40}(\Gamma_{11})$

B_i	$K(B_i)$	α_m	$K(\alpha_m)$	I_m^j	$K(I_m^j)$	X_h	Ψ_h	Φ_h	h
B_1	1	α_2	001	I_2^1	000	x_1	D_3	-	1
		α_3	010	I_3^1	000	\bar{x}_1x_2	D_2	-	2
		α_4	011	I_4^1	000	$\bar{x}_1\bar{x}_2x_3$	D_2D_3	-	3
		α_6	101	I_6^1	000	$\bar{x}_1\bar{x}_2\bar{x}_3$	D_1D_3	-	4
B_2	0	α_2	001	I_2^2	001	x_3	D_3	D_6	5
		α_5	100	I_5^2	010	$\bar{x}_3x_4x_5$	D_1	D_5	6
		α_5	100	I_5^1	000	$\bar{x}_3x_4\bar{x}_5$	D_1	-	7
		α_6	101	I_6^1	000	$\bar{x}_3\bar{x}_4$	D_1D_3	-	8

This table serves as the base to generate the functions of system (4.30). For example, the formulae $D_1 = z_1\bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{z}_1\bar{x}_3$, $D_5 = z_1\bar{x}_1x_4x_5$ can be extracted from Table 6.13 (after minimization).

The table of address transformer AT_3 is constructed using the same approach as the one used for corresponding tables of AT_1 or AT_2 . This table includes the columns: $b_q, \alpha_g, K(\alpha_g), K(b_q), C(b_q), z_q, q$, where column $C(b_q)$ contains codes $K(Y_g)$ of collections of microoperations from vertex $b_q \in B_1$. The table for CMCU $U_{40}(\Gamma_{11})$ has $M_2 = 19$ lines (Table 6.14).

This table is next used to derive Boolean equations for functions (6.18). For example, after analysis of Table 6.14 the following equation can be found: $z_1 =$

$$\bar{\tau}_1 \bar{\tau}_2 \tau_3 T_3 \vee \bar{\tau}_1 \tau_2 \tau_3 T_3 \vee \tau_1 \bar{\tau}_2 \bar{\tau}_3 T_2 \bar{T}_3 \vee \tau_1 \bar{\tau}_2 \bar{\tau}_3 T_2 T_3 \vee \tau_1 \bar{\tau}_2 \tau_3 T_2 T_3 = \bar{\tau}_1 \tau_3 T_3 \vee \tau_1 \bar{\tau}_2 \bar{\tau}_3 T_2 \vee \tau_1 \bar{\tau}_2 T_2 T_3.$$

Table of code transformer TC represents the generation low for functions (4.29). It includes the columns: α_g , $K(\alpha_g)$, $K(b_q)$, $C(b_q)$, z_q , q . Column W_g contains variables $w_r \in W$, equal to 1 in the code $K(B_i)$, where $\alpha_g \in B_i$. Therefore, in case of CMCU U_{40} system (4.29) is transformed to the form:

$$W = W(\tau). \quad (6.23)$$

In present case this table has $|C^1| = 4$ lines (Table 6.15). The input assignment 100 is insignificant. Using this table, after minimization, the equation $w_1 = \bar{\tau}_2 \bar{\tau}_3$ can be obtained.

Table of the block CCS is now used to find functions (6.16) and (6.17). In order to minimize these functions, it is convenient to represent the table for block CCS in the form of Karnaugh map. In case of the CMCU $U_{40}(\Gamma_{11})$, the Karnaugh map for the block CCS is shown in Fig. 6.23.

Table 6.14 Table of address transformer for CMCU $U_{40}(\Gamma_{11})$

b_q	α_g	$K(\alpha_g)$	$K(b_q)$	$C(b_q)$	z_q	q
b_1	α_1	000	**0	000	—	1
b_2	α_1	000	**1	010	z_2	2
b_3	α_2	001	*00	000	—	3
b_4	α_2	001	**1	100	z_1	4
b_5	α_2	001	*1*	001	z_3	5
b_6	α_3	010	**0	011	$z_2 z_3$	6
b_7	α_3	010	**1	000	—	7
b_8	α_4	011	**0	001	z_3	8
b_9	α_4	011	**1	101	$z_1 z_3$	9
b_{10}	α_5	100	000	000	—	10
b_{11}	α_5	100	*01	001	z_3	11
b_{12}	α_5	100	*10	110	$z_1 z_2$	12
b_{13}	α_5	100	*11	101	$z_1 z_3$	13
b_{14}	α_5	100	1**	000	—	14
b_{15}	α_6	101	000	010	z_2	15
b_{16}	α_6	101	*01	011	$z_2 z_3$	16
b_{17}	α_6	101	*10	001	z_3	17
b_{18}	α_6	101	*11	100	z_1	18
b_{19}	α_6	101	1**	001	z_3	19

For example, cell 001001 of the map corresponds to vertex b_4 of the GSA Γ_{11} , where variable y_0 should be placed. Therefore, variable y_0 is placed in this cell and so on. Two following equations can be obtained from this Karnaugh map: $y_0 = \bar{T}_1 \bar{T}_3 \vee \tau_1 \bar{T}_1 \vee \bar{\tau}_1 \bar{\tau}_2 \tau_3 \bar{T}_2$; $y_E = T_1$.

Synthesis of the CMCU $U_{40}(\Gamma)$ logic circuit is reduced to the implementation of systems Φ , Ψ , Z_M , y_0 , y_E , W using PLD chips and implementation of control memory using PROM or RAM chips (Fig. 6.24).

Table 6.15 Table of code transformer TC for CMCU $U_{40}(\Gamma_{11})$

α_g	$K(\alpha_g)$	B_i	$K(B_i)$	W_g	g
α_1	000	B_1	1	w_1	1
α_2	001	B_2	0	-	2
α_3	010	B_2	0	-	3
α_4	011	B_2	0	-	4

$\tau_1\tau_2\tau_3$	000	001	011	010	110	111	101	100
000	y_0	y_0	y_0	y_0	*	*	y_0	y_0
001	0	y_0	0	0	*	*	y_0	y_0
011	*	0	*	*	*	*	y_0	y_0
010	*	*	*	*	*	*	y_0	y_0
110	*	*	*	*	*	*	*	*
111	*	*	*	*	*	*	*	*
101	*	*	*	*	*	*	*	*
100	*	*	*	*	*	*	y_E	y_E

Fig. 6.23 Karnaugh map for block CCS of CMCU $U_{40}(\Gamma_{11})$

In this case, combinational circuit CC has inputs w_1, x_1, \dots, x_5 and outputs $D_1, D_2, D_3, D_4, D_5, D_6$; block AT_3 has inputs $\tau_1, \tau_2, \tau_3, T_1, T_2, T_3$ and outputs z_1, z_2, z_3 ; block TC has inputs τ_2, τ_3 and output w_1 ; block CCS has inputs $T_1, T_2, T_3, \tau_1, \tau_2, \tau_3$ and outputs y_0, y_E ; control memory has address inputs z_1, z_2, z_3 and six outputs with microoperations y_1, \dots, y_6 .

Synthesis method used for CMCU $U_{44}(\Gamma)$ includes the same steps as the method applied for CMCU $U_{40}(\Gamma)$, but instead of operational linear chains $\alpha_g \in C$ and their classes $B_i \in \Pi_C$, the elementary OLC $\alpha_g \in C_E$ and their classes $B_i \in \Pi_E$ are used respectively. Consider an example of logic circuit synthesis for the CMCU $U_{44}(\Gamma_{11})$, where EOLC sets for GSA Γ_{11} have the form: $C_E = \{\alpha_1, \dots, \alpha_8\}$ and $C_E^1 = \{\alpha_1, \dots, \alpha_6\}$. In this case $I_E = 4$ and $R_{11}, W = \{w_1, w_2\}$.

We encode the classes applying the same principle as the one used for CMCU U_{40} . In this case, codes $K(B_1) = 01, K(B_2) = 10, K(B_3) = 00$ and $K(B_4) = 11$ can be formed. The encoding of collections of microoperations, obtained earlier, is shown in Fig. 6.21, and the control memory content in Fig. 6.22. The codes of elementary OLC and their components are shown in Fig. 6.11.

Transition table for the CMCU U_{44} is constructed using the same approach as the one used for CMCU U_{14} . In our last example, it is made using system (6.22) and includes $H_{44}(\Gamma_{11}) = 10$ lines (Table 6.16).

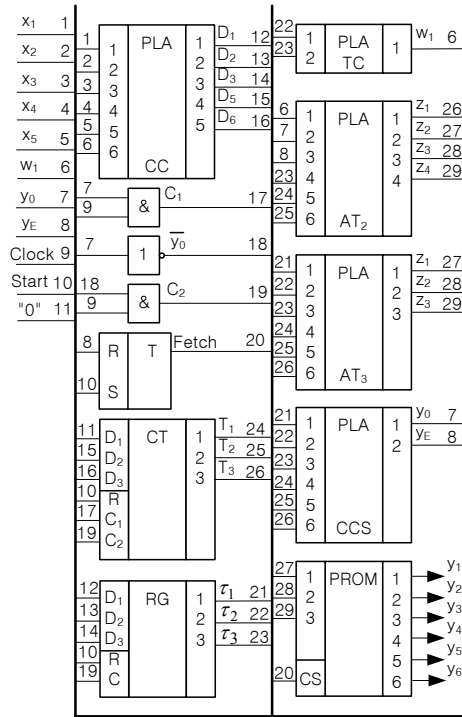


Fig. 6.24 Logic circuit of CMCU $U_{40}(I_{11})$

Using Table 6.16 we can find functions Ψ , for example: $D_1 = \bar{w}_1 w_2 \bar{x}_1 \bar{x}_2 \vee w_1 \bar{w}_2 \bar{x}_3 \vee w_1 w_2$, which corresponds to the lines 3, 4, 7 – 10 (after minimization).

Table of the code transformer TC has in this case $|C_E^1| = 6$ lines (Table 6.17), and serves to find functions W . For example, the following equation $w_1 = \bar{\tau}_1 \bar{\tau}_2 \tau_3 \vee \tau_1 \tau_3$ can be formed using Table 6.17 (after minimization).

Table 6.16 Transition table for CMCU $U_{44}(I_{11})$

B_i	$K(B_i)$	α_g	$K(\alpha_g)$	X_h	Ψ_h	h
B_1	01	α_2	001	x_1	D_3	1
		α_4	011	$\bar{x}_1 x_2$	$D_2 D_3$	2
		α_5	100	$\bar{x}_1 \bar{x}_2 x_3$	D_1	3
		α_8	111	$\bar{x}_1 \bar{x}_2 \bar{x}_3$	$D_1 D_2 D_3$	4
B_2	10	α_3	010	1	D_2	5
		α_3	010	x_3	$D_1 D_2$	6
B_3	00	α_7	110	$\bar{x}_3 x_4 x_5$	D_1	7
		α_6	101	$\bar{x}_3 x_4 \bar{x}_5$	$D_1 D_3$	8
		α_8	111	$\bar{x}_3 \bar{x}_4$	$D_1 D_2 D_3$	9
		α_7	110	1	$D_1 D_2$	10

Table 6.17 Table of code transformer TC for CMCU $U_{44}(\Gamma_{11})$

α_g	$K(\alpha_g)$	B_i	$K(B_i)$	W_g	g
α_1	000	B_1	01	w_2	1
α_2	001	B_2	10	w_1	2
α_3	010	B_3	00	–	3
α_4	011	B_3	00	–	4
α_5	100	B_3	00	–	5
α_6	101	B_4	11	$w_1 w_2$	6

In this particular case table of the address transformer AT_3 includes $M_2 = 19$ lines (Table 6.18).

Table 6.18 Table of address transformer AT_3 for $U_{44}(\Gamma_{11})$

b_q	α_g	$K(\alpha_g)$	$K(b_q)$	$C(b_q)$	z_q	q
b_1	α_1	000	**0	000	–	1
b_2	α_1	000	**1	010	z_2	2
b_3	α_2	001	***	000	–	3
b_4	α_3	010	**0	100	z_1	4
b_5	α_3	010	**1	001	z_3	5
b_6	α_4	011	**0	011	$z_2 z_3$	6
b_7	α_4	011	**1	000	–	7
b_8	α_5	100	**0	001	z_3	8
b_9	α_5	100	**1	101	$z_1 z_3$	9
b_{10}	α_6	101	**0	000	–	10
b_{11}	α_6	101	**1	001	z_3	11
b_{12}	α_7	110	*00	110	$z_1 z_2$	12
b_{13}	α_7	110	**1	101	$z_1 z_3$	13
b_{14}	α_7	110	*1*	000	–	14
b_{15}	α_8	111	000	010	z_2	15
b_{16}	α_8	111	*01	011	$z_2 z_3$	16
b_{17}	α_8	111	*10	001	z_3	17
b_{18}	α_8	111	*11	100	z_1	18
b_{19}	α_8	111	1**	001	z_3	19

This table is now used for generation of functions $z_r \in Z_M$. For example, the following Boolean equation can be extracted from Table 6.18: $z_1 = \bar{\tau}_1 \tau_2 \bar{\tau}_3 \bar{T}_3 \vee \tau_1 \bar{\tau}_2 \bar{\tau}_3 T_3 \vee \tau_1 \tau_2 \bar{\tau}_3 \bar{T}_2 \bar{T}_3 \vee \tau_1 \tau_2 \bar{\tau}_3 T_3 \vee \tau_1 \tau_2 \tau_3 T_2 T_3$. Comparison of the equations obtained for function z_1 shows that in case of CMCU $U_{40}(\Gamma_{11})$ disjunctive normal form of this function includes smaller number of terms, than for the equivalent CMCU $U_{44}(\Gamma_{11})$.

Table of block CCS serves to generate formulae for both additional variables controlling synchronization (y_0) and by fetching y_E . To make their minimizing easier, let us represent these functions using Karnaugh map shown in Fig. 6.25. Obviously,

the well-known algorithms, such as ESPRESSO [10], can be used for computer minimization of functions y_0 and y_E .

$\tau_1\tau_2\tau_3$	000	001	011	010	110	111	101	100
000	y_0	0	y_0	y_0	y_0	y_0	y_0	y_0
001	0	*	0	0	y_0	y_0	0	0
011	*	*	*	*	*	y_0	*	*
010	*	*	*	*	y_E	y_0	*	*
110	*	*	*	*	*	*	*	*
111	*	*	*	*	*	*	*	*
101	*	*	*	*	*	*	*	*
100	*	*	*	*	*	y_E	*	*

Fig. 6.25 Karnaugh map for block CCS of CMCU $U_{44}(\Gamma_{11})$

The following minimal equations can be obtained from this Karnaugh map: $y_0 = \bar{\tau}_1\bar{\tau}_3\bar{T}_3 \vee \bar{\tau}_1\tau_2\bar{T}_3 \vee \tau_1\tau_2\bar{T}_2\bar{T}_3 \vee \tau_1\bar{T}_1\bar{T}_2\bar{T}_3$; $y_E = T_1 \vee \bar{\tau}_3T_2$.

Synthesis of CMCU $U_{44}(\Gamma_{11})$ logic circuit is reduced to hardware implementation of Boolean functions obtained for the given logic elements.

6.5 Combined application of different object transformation methods for CMCU

As was shown in Section 5.1, both transformation methods: using object codes and using different codes of the same object, can be applied to optimize either the combinational circuit CC or the control memory size of CMCU $U_8 - U_{11}$. All models of CMCU $U_{16} - U_{45}$ discussed already have three levels, but combined application of the methods presented in Chapters 5 and 6 results in four-level models. All possible models of CMCU with code sharing are shown in Table 6.19.

As follows from Table 6.19, the first level (A) for CMCU $U_{46} - U_{57}$ is occupied by combinational circuit CC, third level (C) is occupied by one of the address transformers $AT_1 - AT_3$, fourth level (D) is occupied by control memory CM, and second level (B) can be occupied either by a single code transformer TOK, or by two code transformers TOK, TC. The block CCS is always used jointly with address transformer AT_3 . Structural diagrams for logic circuits of the CMCU $U_{46} - U_{57}$ are generated by combination of the blocks taken from corresponding line of the table. For example, structural diagram of CMCU U_{54} is shown in Fig. 6.26.

Table 6.19 Four-level models for CMCU with codes sharing

Type	A	B	C	D	Analogue
U_{46}	CC	TOK	AT_1	CM	U_8
U_{47}	CC	TOK	AT_2	CM	U_8
U_{48}	CC	TOK	AT_3, CCS	CM	U_8
U_{49}	CC	TOK	AT_1	CM	U_9
U_{50}	CC	TOK	AT_2	CM	U_9
U_{51}	CC	TOK	AT_3, CCS	CM	U_9
U_{52}	CC	TOK, TC	AT_1	CM	U_{10}
U_{53}	CC	TOK, TC	AT_2	CM	U_{10}
U_{54}	CC	TOK, TC	AT_3, CCS	CM	U_{10}
U_{55}	CC	TOK	AT_1	CM	U_{11}
U_{56}	CC	TOK	AT_2	CM	U_{11}
U_{57}	CC	TOK	AT_3, CCS	CM	U_{11}

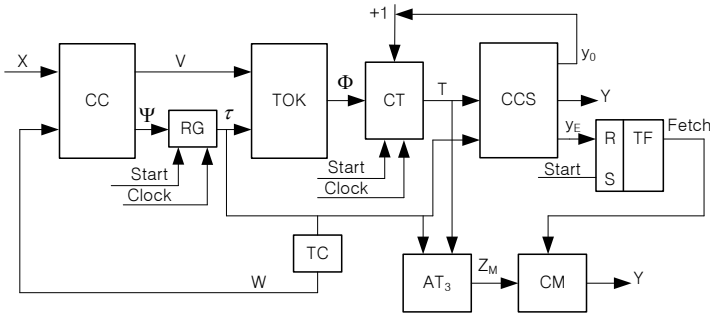


Fig. 6.26 Structural diagram for CMCU U_{54}

Functions of the blocks CC, TC, TOK, AT_3 , CCS, CM are the same, as for corresponding three-level models. Synthesis method applied for the CMCU U_{54} can be generated as the integration of synthesis methods used for corresponding three-level models. This method includes the following steps:

1. Preliminary transformation of initial GSA Γ (procedure P_4).
2. Construction of the set of operational linear chains C (procedure P_1).
3. Encoding of OLC $\alpha_q \in C$ and their components.
4. Encoding of the inputs $I_g^j \in I(\Gamma)$ of operational linear chains.
5. Construction of partition Π_C of the set C^1 .
6. Encoding of equivalence classes $B_i \in \Pi_C$.
7. Construction of transition table of CMCU.
8. Construction of the table of code transformer TC.
9. Construction of the table of code transformer TOK.
10. Encoding of collections of microoperations $Y_q \subseteq Y$.
11. Construction of the control memory content.
12. Construction of the table of address transformer AT_3 .
13. Construction of the table of CCS block.
14. Synthesis of logic circuit of CMCU using the given logic elements.

Let us discuss an example of CMCU $U_{54}(\Gamma_{12})$ synthesis, using the transformed GSA $\Gamma_{12}(U_{54})$ shown in Fig. 6.27.

Application of procedure P_1 results in the following set of operational linear chains $C = \{\alpha_1, \dots, \alpha_8\}$, where $\alpha_1 = \langle b_1, \dots, b_5 \rangle$, $I_1^1 = b_1$, $I_1^2 = b_3$, $O_1 = b_5$, $L_1 = 5$, $NI_1 = 2$; $\alpha_2 = \langle b_6, b_7, b_8 \rangle$, $I_2^1 = b_6$, $O_2 = b_8$, $L_2 = 3$, $NI_2 = 1$; $\alpha_3 = \langle b_9, b_{10} \rangle$, $I_3^1 = b_9$, $O_3 = b_{10}$, $L_3 = 2$, $NI_3 = 1$; $\alpha_4 = \langle b_{11}, b_{12}, b_{13} \rangle$, $I_4^1 = b_{11}$, $I_4^2 = b_{13} = O_4$, $L_4 = 3$, $NI_4 = 2$; $\alpha_5 = \langle b_{14}, b_{15} \rangle$, $I_5^1 = b_{14}$, $I_5^2 = b_{15}$, $O_5 = b_{15}$, $L_5 = NI_5 = 2$; $\alpha_6 = \langle b_{16}, b_{17} \rangle$, $I_6^1 = b_{16}$, $O_6 = b_{17}$, $L_6 = 2$, $NI_6 = 1$; $\alpha_7 = \langle b_{18} \rangle$, $I_7^1 = O_7 = b_{18}$, $L_7 = NI_7 = 1$; $\alpha_8 = \langle b_{19} \rangle$, $I_8^1 = O_8 = b_{19}$, $L_8 = NI_8 = 1$.

Thus, $G = 8$, $R_6 = 3$, $\tau = \{\tau_1, \tau_2, \tau_3\}$ and the trivial encoding of OLC $\alpha_g \in C$ is: $K(\alpha_1) = 000, \dots, K(\alpha_8) = 111$. Analysis of OLC shows that $L_{\max} = 5$, $R_7 = 3$, $T = \{T_1, T_2, T_3\}$. Let us encode components of OLC $\alpha_g \in C$ by codes $K(b_q)$ having R_7 bits. It results in microinstruction addresses shown in Fig. 6.28.

Set of OLC inputs includes, in this example, 11 elements, where $NI_{\max} = 2$, $R_{14} = 1$, $V = \{v_1\}$. Let us encode these inputs I_g^l as: $K(I_g^1) = 0$, $K(I_g^2) = 1$ ($g = 1, \dots, 8$).

For the GSA Γ_{12} we find the set $C^1 = \{\alpha_1, \dots, \alpha_7\}$ and partition $\Pi_C = \{B_1, B_2, B_3\}$ with classes $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6, \alpha_7\}$. Thus, $I = 3$, $R_4 = 2$, $W = \{w_1, w_2\}$. In order to minimize the system W , let us encode the classes $B_i \in \Pi_C$ as: $K(B_2) = 00$, $K(B_3) = 01$.

System of transition formulae for outputs of OLC $\alpha_g \in C^1$ includes three following expressions:

$$\begin{aligned} O_1 &\rightarrow x_1x_2I_1^2 \vee x_1\bar{x}_2I_2^1 \vee \bar{x}_1x_3I_3^1 \vee \bar{x}_1\bar{x}_3x_4I_4^1 \vee \bar{x}_1\bar{x}_3\bar{x}_4I_4^2; \\ O_2, O_3, O_4 &\rightarrow x_2I_5^1 \vee \bar{x}_2x_3I_6^1 \vee \bar{x}_2\bar{x}_3I_7^1; \\ O_5, O_6, O_7 &\rightarrow x_4I_5^2 \vee \bar{x}_4I_8^1. \end{aligned} \quad (6.24)$$

Replacement of outputs O_g by the symbols of corresponding classes B_i , where $\alpha_g \in B_i$, leads to the system (6.25):

$$\begin{aligned} B_1 &\rightarrow x_1x_2I_1^2 \vee x_1\bar{x}_2I_2^1 \vee \bar{x}_1x_3I_3^1 \vee \bar{x}_1\bar{x}_3x_4I_4^1 \vee \bar{x}_1\bar{x}_3\bar{x}_4I_4^2; \\ B_2 &\rightarrow x_2I_5^1 \vee \bar{x}_2x_3I_6^1 \vee \bar{x}_2\bar{x}_3I_7^1; \\ B_3 &\rightarrow x_4I_5^2 \vee \bar{x}_4I_8^1. \end{aligned} \quad (6.25)$$

The system (6.25) allows to construct the transition table for CMCU $U_{54}(\Gamma_{12})$, having $H_{54}(\Gamma_{12})$ lines (Table 6.20).

Using this table we obtain the systems of Boolean functions Ψ and V . Next, we can find, for example, the following equations $D_1 = F_6 \vee \dots \vee F_{10} = \bar{w}_1\bar{w}_2 \vee \bar{w}_1w_2 = \bar{w}_1$; $v_1 = F_1 \vee F_5 \vee F_9 = w_1\bar{w}_2x_1x_2 \vee w_1\bar{w}_2\bar{x}_1\bar{x}_3x_4 \vee \bar{w}_1w_1x_4$.

Table of the code transformer TC describes transformation of OLC codes into the codes of classes of pseudoequivalent OLC. This table includes in general $G_1 = |C^1|$ lines. In the particular case of CMCU $U_{54}(\Gamma_{12})$ we obtain, for example, $G_1 = 7$ lines (Table 6.21).

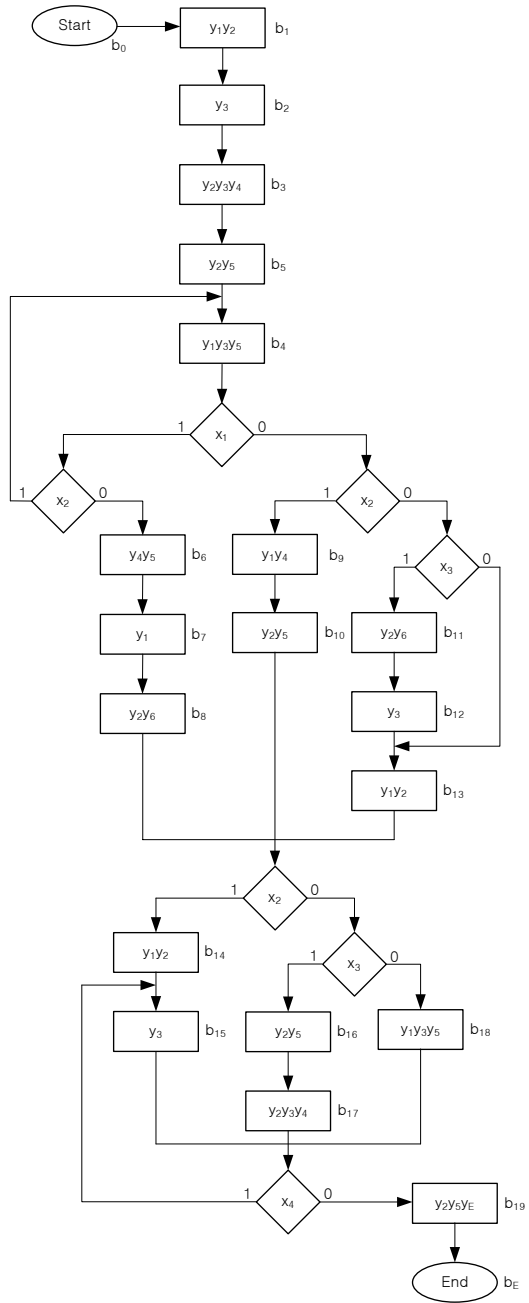


Fig. 6.27 Transformed GSA $\Gamma_{12}(U_{54})$

	$\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110	111
$\tau_1\tau_2\tau_3$	000	b_1	b_6	b_9	b_{11}	b_{14}	b_{16}	b_{18}	b_{19}
	001	b_2	b_7	b_{10}	b_{12}	b_{15}	b_{17}	*	*
	010	b_3	b_8	*	b_{13}	*	*	*	*
	011	b_4	*	*	*	*	*	*	*
	100	b_5	*	*	*	*	*	*	*
	101	*	*	*	*	*	*	*	*
	110	*	*	*	*	*	*	*	*
	111	*	*	*	*	*	*	*	*

Fig. 6.28 Microinstruction addresses for CMCU $U_{54}(\Gamma_{12})$

Table 6.20 Transition table for CMCU $U_{54}(\Gamma_{12})$

B_i	$K(B_i)$	I_m^j	$K(\alpha_m)$	$K(I_m^j)$	X_h	Ψ_h	V_h	h
B_1	10	I_1^2	000	1	x_1x_2	-	v_1	1
		I_2^1	001	0	$x_1\bar{x}_2$	D_3	-	2
		I_3^1	010	0	\bar{x}_1x_3	D_2	-	3
		I_4^1	011	0	$\bar{x}_1\bar{x}_3x_4$	D_2D_3	-	4
		I_4^2	011	1	$\bar{x}_1\bar{x}_3\bar{x}_4$	D_2D_3	v_1	5
B_2	00	I_5^1	100	0	x_2	D_1	-	6
		I_6^1	101	0	\bar{x}_2x_3	D_1D_3	-	7
		I_7^1	110	0	$\bar{x}_2\bar{x}_3$	D_1D_2	-	8
B_3	01	I_5^2	100	1	x_4	D_1	v_1	9
		I_8^1	111	0	\bar{x}_4	$D_1D_2D_3$	-	10

Table 6.21 Table of code transformer TC for CMCU $U_{54}(\Gamma_{12})$

α_g	$K(\alpha_g)$	B_i	$K(B_i)$	W_g	g
α_1	000	B_1	10	w_1	1
α_2	001	B_2	00	-	2
α_3	010	B_2	00	-	3
α_4	011	B_2	00	-	4
α_5	100	B_3	01	w_2	5
α_6	101	B_3	01	w_2	6
α_7	110	B_3	01	w_2	7

The system of Boolean functions W can be obtained from this table. Taking into account the insignificant input assignment 111, the following equations can be found from Table 6.21: $w_1 = \bar{\tau}_1\bar{\tau}_2\bar{\tau}_3$, $w_2 = \tau_1$.

Table of code transformer TOK represents generation of OLC component codes. In the discussed example it includes 11 lines (Table 6.22).

Table 6.22 Table of code transformer TOK for CMCU $U_{54}(\Gamma_{12})$

α_g	$K(\alpha_g)$	I_g^j	$K(I_g^j)$	b_q	$K(b_q)$	Φ_h	h
α_1	000	I_1^1	0	b_1	000	-	1
α_1	000	I_1^2	1	b_3	*10	D_5	2
α_2	001	I_2^1	0	b_6	*00	-	3
α_3	010	I_3^1	0	b_9	**0	-	4
α_4	011	I_4^1	0	b_{11}	*00	-	5
α_4	011	I_4^2	1	b_{13}	*1*	D_5	6
α_5	100	I_5^1	0	b_{14}	**0	-	7
α_5	100	I_5^2	1	b_{15}	**1	D_6	8
α_6	101	I_6^1	0	b_{16}	**0	-	9
α_7	110	I_7^1	0	b_{18}	***	-	10
α_8	111	I_8^1	0	b_{19}	***	-	11

Insignificant input assignments are taken into account in the component codes of Table 6.22. This table permits to obtain the input memory functions Φ and the equations, as for example: $D_5 = \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 v_1 \vee \bar{\tau}_1 \tau_2 \tau_3 v_1$.

There are $M_4 = 8$ different collections of microoperations in operator vertices of the GSA Γ_{12} , namely: $Y_1 = \{y_1, y_2\}$, $n_1 = 4$; $Y_2 = \{y_3\}$, $n_2 = 3$; $Y_3 = \{y_2, y_3, y_4\}$, $n_3 = 2$; $Y_4 = \{y_2, y_5\}$, $n_4 = 2$; $Y_5 = \{y_1, y_3, y_5\}$, $n_5 = 2$; $Y_6 = \{y_4, y_5\}$, $n_6 = 3$; $Y_7 = \{y_1\}$, $n_7 = 1$; $Y_8 = \{y_2, y_6\}$, $n_8 = 2$. Thus, $R_{16} = 3$, $Z_M = \{z_1, z_2, z_3\}$. The queue of collections of microoperations $\langle Y_1, Y_2, Y_6, Y_3, Y_4, Y_5, Y_8, Y_7 \rangle$ should be organized for their encoding. The codes $K(Y_q)$ of collections of microoperations for the CMCU $U_{54}(\Gamma_{12})$ are shown in Fig. 6.29.

Fig. 6.29 Codes of collections of microoperations for CMCU $U_{54}(\Gamma_{12})$

		$z_2 z_3$			
		00	01	11	10
z_1	0	Y_1	Y_2	Y_6	Y_4
	1	Y_3	Y_5	Y_8	Y_7

The control memory content for this case is shown in Fig. 6.30.

Fig. 6.30 Control memory content for CMCU $U_{54}(\Gamma_{12})$

		$z_2 z_3$			
		00	01	11	10
z_1	0	$y_1 y_2$	y_3	$y_4 y_5$	$y_2 y_5$
	1	$y_2 y_3 y_4$	$y_1 y_3 y_5$	$y_2 y_6$	y_1

The table of address transformer AT_3 is the base to derive functions Z_M and includes M_2 lines. In case of CMCU $U_{54}(\Gamma_{12})$, the table of address transformer AT_3 has $M_2 = 19$ lines (Table 6.23). Insignificant input assignments are taken into account

to get the components codes written in Table 6.23. The following equation for function z_1 can be derived, for example, from Table 6.23: $z_1 = \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 T_2 \bar{T}_3 \vee \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 T_1 \vee \bar{\tau}_1 \bar{\tau}_2 \tau_3 T_3 \vee \bar{\tau}_1 \bar{\tau}_2 \tau_3 T_2 \vee \bar{\tau}_1 \tau_2 \tau_3 \bar{T}_2 \bar{T}_3 \vee \tau_1 \bar{\tau}_2 \tau_3 T_3 \vee \tau_1 \tau_2 \bar{\tau}_3$. The terms of this equation correspond to lines 3, 5, 7, 8, 11, 17 and 18 of the table respectively.

As in previous cases, we use the Karnaugh map to represent additional variables y_0 and y_E . This map is constructed on the base of microinstruction addresses. Let us remind the reader, that variable y_0 is inserted in all vertices of GSA, which do not correspond to the OLC outputs (Fig. 6.31). The following Boolean equations can be derived from the Karnaugh map: $y_0 = \tau_2 \bar{T}_1 \bar{T}_3 \vee \bar{\tau}_1 \bar{T}_2 \bar{T}_3 \vee \bar{\tau}_1 \tau_3 \bar{T}_2 \vee \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 \bar{T}_1$; $y_E = \tau_1 \tau_2 \tau_3$.

Synthesis of logic circuit of the CMCU $U_{54}(\Gamma_{12})$ is reduced to implementation of systems $\Psi, V, \Phi, W, Z_M, y_0, y_E$ using PLD chips and implementation of control memory using either RAM or PROM chips (Fig. 6.32).

All models of CMCU listed in Table 6.19 can be synthesized by analogy with the discussed example.

Table 6.23 Table of address transformer AT_3 for CMCU $U_{54}(\Gamma_{12})$

b_q	α_g	$K(\alpha_g)$	$K(b_q)$	$C(b_q)$	z_q	q
b_1	α_1	000	000	000	–	1
b_2	α_1	000	*01	001	z_3	2
b_3	α_1	000	*01	100	z_1	3
b_4	α_1	000	*11	011	$z_2 z_3$	4
b_5	α_1	000	1**	101	$z_1 z_3$	5
b_6	α_2	001	*00	010	z_2	6
b_7	α_2	001	**1	111	$z_1 z_2 z_3$	7
b_8	α_2	001	*1*	110	$z_1 z_2$	8
b_9	α_3	010	**0	000	–	9
b_{10}	α_3	010	**1	010	z_2	10
b_{11}	α_4	011	*00	110	$z_1 z_2$	11
b_{12}	α_4	011	**1	001	z_3	12
b_{13}	α_4	011	*1*	000	–	13
b_{14}	α_5	100	**0	000	–	14
b_{15}	α_5	100	**1	001	z_3	15
b_{16}	α_6	101	**0	011	$z_2 z_3$	16
b_{17}	α_6	101	**1	100	z_1	17
b_{18}	α_7	111	***	101	$z_1 z_3$	18
b_{19}	α_8	111	***	010	z_2	19

$T_1 T_2 T_3$	$\tau_1 \tau_2 \tau_3$							
	000	001	011	010	110	111	101	100
000	y_0	y_0	y_0	y_0	0	y_E	y_0	y_0
001	y_0	y_0	y_0	0	*	*	0	0
011	y_0	0	0	*	*	*	*	*
010	y_0	*	*	*	*	*	*	*
110	*	*	*	*	*	*	*	*
111	*	*	*	*	*	*	*	*
101	*	*	*	*	*	*	*	*
100	*	*	*	*	*	y_E	*	*

Fig. 6.31 Karnaugh map for functions y_0 and y_E of CMCU $U_{54}(I_{12})$

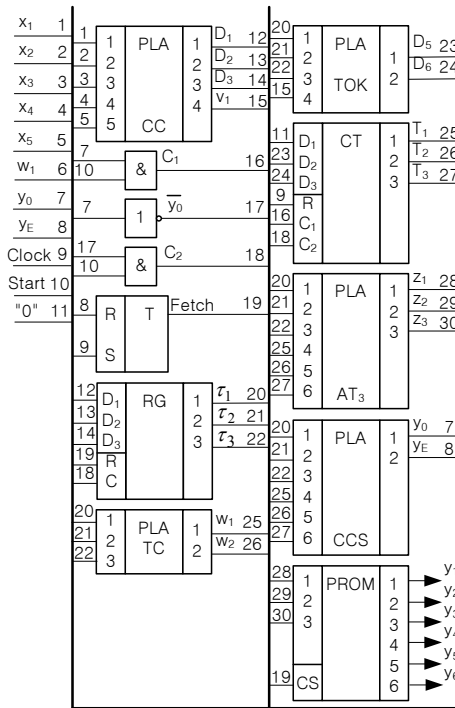


Fig. 6.32 Logic circuit of CMCU $U_{54}(I_{12})$

References

1. A.A. Barkalov and R. Wiśniewski. Synthesis of compositional microprogram control units with function decoder. In *Inter. Workshop Control and Information Technology – IWCIT 2007*, pages 229 – 232. Technical University of Ostrava, 2007.
2. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
3. A. Barkalov and M. Węgrzyn. *Design of Control Units With Programmable Logic*. University of Zielona Góra Press, 2006.
4. A.A. Barkalov and L.A. Titarenko and M. Kołopencyk. Optimization of control unit with code sharing. In *Proc. of the 3rd Inter. Workshop of IFAC Discrete–Event System Design (DESDES' 06)*, pages 195 – 200. University of Zielona Gora Press, 2006.
5. A.A. Barkalov and L.A. Titarenko and M. Kołopencyk. Optimization of control unit with code sharing. In *Proc. of the IEEE East-West Design & Test Workshop (EWDWTW'06)*, pages 171 – 174. Kharkov National University of Radioelectronics, 2006.
6. A.A. Barkalov and L. A. Titarenko. *Measurements, methods, systems and design*, chapter Design of control units with programmable logic devices, pages 371 – 391. Wydawnictwo Komunikacji i Łączności, Warsaw, 2007.
7. A.A. Barkalov, L. A. Titarenko, and J. Bieganowski. Synthesis of control units with modified operational linear chains. *Measurements Automation Control*, 53(5):15 – 17, 2007.
8. A.A. Barkalov, L.A. Titarenko, and R. Wiśniewski. Optimization of address circuit of compositional microprogram unit. In *Proc. of the IEEE East-West Design & Test Workshop (EWDWTW'06)*, pages 167 – 170. Kharkov National University of Radioelectronics, 2006.
9. A.A. Barkalov and R. Wiśniewski. Design of compositional microprogram control units with maximal encoding of inputs. *Radioelectronics and informatics*, (3):79 – 81, 2004.
10. G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw–Hill, 1994.

Chapter 7

Synthesis of CMCU with coding of logical conditions and collections of microoperations

Abstract The chapter deals with multilevel implementation of CMCU logic circuits. These methods are based on some well-known ideas taken from the literature devoted to optimization of FSM and MCU. They are of course adapted to the particular conditions of the CMCU operation. All these methods lead to the increase of cycle time, in comparison with the CMCU basic structure. They can be applied, when minimum hardware amount is the main goal of a particular design.

7.1 Coding of logical conditions for CMCU with basic structure

As we already know, the CMCU U_1 includes Mealy FSM S_1 , and therefore it seems reasonable to start discussion about encoding logical conditions directly from this model of CMCU. We know also that coding of logical conditions is reduced to replacement of logical conditions $x_l \in X$ by some new variables $p_g \in P$. This approach makes sense only if the following condition takes place:

$$|P| \ll |X|. \quad (7.1)$$

Functions $p_g \in P$ are determined using the expression [4]:

$$p_g = \bigvee_{m=1}^M \bigvee_{l=1}^L C_{ml} A_m x_l \quad (g = 1, \dots, G_0), \quad (7.2)$$

where C_{ml} is a Boolean variable, equal to 1 iff variable $x_l \in X$ is replaced by variable $p_g \in P$ for internal state $a_m \in A$; A_m is a conjunction of state variables corresponding to code $K(a_m)$ of state $a_m \in A$. By analogy with the Mealy MP FSM [9], let us denote CMCU U_i based on coding of logical conditions as $MU_i (i = 1, \dots, 57)$. Structural diagram of the CMCU MU_1 is shown in Fig. 7.1.

In CMCU MU_1 block M implements functions

$$P = P(\tau, X) \quad (7.3)$$

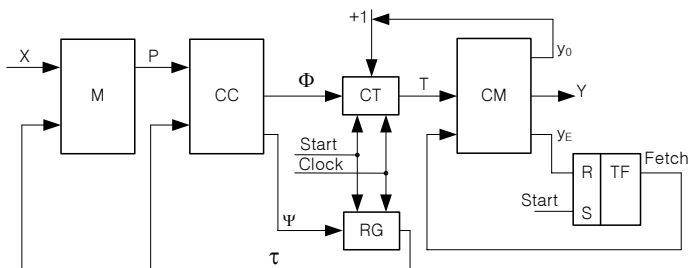


Fig. 7.1 Structural diagram of CMCU MU_1

corresponding to system (7.2), and combinational circuit CC implements functions

$$\Phi = \Phi(\tau, P), \tag{7.4}$$

$$\Psi = \Psi(\tau, P). \tag{7.5}$$

Obviously, operation principles of both CMCU U_1 and MU_1 are the same, but additional step connected with replacement of logical conditions X by new variables P should be executed to design of CMCU MU_1 logic circuit.

Let the Mealy FSM S_1 for CMCU U_1 interpreting some GSA Γ_{13} be represented by the structure table, shown in Table 7.1.

Table 7.1 Structure table of Mealy FSM S_1 for CMCU $U_1(\Gamma_{13})$

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Ψ_h	Φ_h	h
a_1	000	a_2	001	x_1	D_7	D_1D_4	1
		a_3	010	\bar{x}_1x_2	D_6	D_2	2
		a_4	011	$\bar{x}_1\bar{x}_2$	D_6D_7	D_2D_3	3
a_2	001	a_3	010	x_3	D_6	D_1D_3	4
		a_4	011	\bar{x}_3	D_6D_7	D_2	5
a_3	010	a_5	100	1	D_5	D_1D_4	6
a_4	011	a_2	001	1	D_7	D_5	7
a_5	100	a_3	010	x_3x_4	D_6	D_1D_4	8
		a_2	001	$x_3\bar{x}_4$	D_7	D_2	9
		a_6	101	\bar{x}_3	D_5D_7	D_1D_3	10
a_6	101	a_2	001	x_5	D_7	D_2	11
		a_1	000	\bar{x}_5	–	D_1	12

Coding of logical conditions and synthesis of block M are executed using well-known methods presented in [3]. Let us discuss execution of these steps for our particular example.

1. **Construction of set P .** Let us find the cardinality numbers of sets $X(a_m)$, including logical conditions determined transitions from state $a_m \in A$. Parameter G_0 is determined by the following formula:

$$G_0 = \max(|X(a_1)|, \dots, |X(a_{M_1})|). \quad (7.6)$$

From Table 7.1 the following sets can be found: the set of states $A_1 = \{a_1, \dots, a_6\}$, where $M_1 = 6$, and the sets of logical conditions for these states: $X(a_1) = \{x_1, x_2\}$, $X(a_2) = \{x_3\}$, $X(a_3) = X(a_4) = \emptyset$, $X(a_5) = \{x_3, x_4\}$, $X(a_6) = \{x_5\}$. It means that $G_0 = 2$ and $P = \{p_1, p_2\}$, according to (7.6).

2. **Construction of table for encoding of logical conditions.** This table includes columns $a_m, K(a_m), p_1, \dots, p_{G_0}$. Intersection of row a_m and column p_g is marked by logical condition $x_l \in X$, replaced by variable $p_g \in P$ for state $a_m \in A$. Let $X(P_g)$ be a set of logical conditions $x_l \in X$ replaced by variable $p_g \in P$. The table is constructed in such a way that the following condition holds:

$$|X(P_i) \cap X(P_j)| \rightarrow \min, \quad (7.7)$$

where $i \neq j, i, j \in \{1, \dots, G_0\}$. This problem is reduced to classical problem of graph coloring [7].

In the discussed case, encoding table of logical conditions (Table 7.2) satisfies (7.7).

Table 7.2 Encoding table of logical conditions for CMCU $MU_1(\Gamma_{13})$

a_m	$K(a_m)$	p_1	p_2	a_m	$K(a_m)$	p_1	p_2
a_1	000	x_1	x_2	a_4	011	–	–
a_2	001	x_3	–	a_5	100	x_3	x_4
a_3	010	–	–	a_6	101	–	x_5

3. **Construction of system P .** Equations for system (7.2) are derived from encoding table of logical conditions in a trivial way. The following equations can be derived in the case of CMCU $MU_1(\Gamma_{13})$:

$$\begin{aligned} p_1 &= A_1x_1 \vee A_2x_3 \vee A_5x_3; \\ p_2 &= A_1x_2 \vee A_5x_4 \vee A_6x_5. \end{aligned} \quad (7.8)$$

4. **Implementation of block M** is reduced to implementation of each equation of system (7.3) by separate multiplexer. It is clear that multiplexers can be implemented using macrocells of particular PLD chips.

In the discussed example block B is implemented using two multiplexers (Fig. 7.2), having $R_1 = 3$ control inputs and 2^{R_1} data inputs.

The following transformation of structure table should be executed to derive equations of system (7.4): column X_h should be replaced by column P_h , and logical conditions $x_l \in X$ should be replaced by variables $p_g \in P$. This replacement is executed in a trivial way. In our example, the transformation results are shown in Table 7.3.

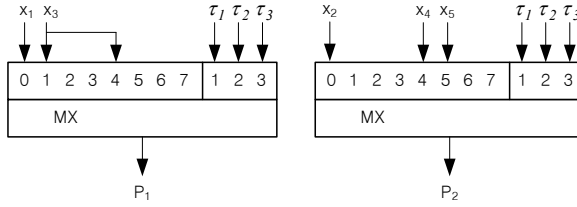


Fig. 7.2 Logic circuit of block M of CMCU $MU_1(\Gamma_{13})$

Table 7.3 Transformed structure table of FSM S_1 for CMCU $MU_1(\Gamma_{13})$

a_m	$K(a_m)$	a_s	$K(a_s)$	P_h	Ψ_h	Φ_h	h
a_1	000	a_2	001	P_1	D_7	D_1D_4	1
		a_3	010	\bar{P}_1P_2	D_6	D_2	2
		a_4	011	$\bar{P}_1\bar{P}_2$	D_6D_7	D_2D_3	3
a_2	001	a_3	010	P_1	D_6	D_1D_3	4
		a_4	011	\bar{P}_1	D_6D_7	D_2	5
a_3	010	a_5	100	1	D_5	D_1D_4	6
a_4	011	a_2	001	1	D_7	D_5	7
a_5	100	a_3	010	P_1P_2	D_6	D_1D_4	8
		a_2	001	$P_1\bar{P}_2$	D_7	D_2	9
		a_6	101	\bar{P}_1	D_5D_7	D_1D_3	10
a_6	101	a_2	001	P_2	D_7	D_2	11
		a_1	000	\bar{P}_2	-	D_1	12

Input memory functions $D_r \in \Psi \cup \Phi$ can be derived as the following disjunctive normal forms:

$$D_r = \bigvee_{r=1}^{R_1+R_2} C_{rh}A_m^hP_h \quad (r = 1, \dots, R_1 + R_2). \tag{7.9}$$

In this equation C_{rh} is a Boolean variable equal to 1 iff the input memory function D_r is placed in the line h of the structure table. For example, the equation $D_5 = \bar{\tau}_1\tau_2\bar{\tau}_3 \vee \tau_1\bar{\tau}_2\bar{\tau}_3\bar{P}_1$ can be derived from Table 7.3.

The remaining synthesis steps for CMCU MU_1 are the same as for CMCU U_1 . Thus, synthesis of CMCU MU_1 includes the following steps:

1. Preliminary transformation of GSA Γ (procedure P_4).
2. Construction of OLC C set (procedure P_1).
3. Addressing of microinstructions (procedure P_2).
4. Construction of the control memory content.
5. Transformation of GSA $\Gamma(U_1)$ (procedure P_3).
6. Construction of structure table for FSM S_1 .
7. Construction of encoding table for logical conditions.
8. Construction of transformed structure table for FSM S_1 .
9. Synthesis of logic circuit of CMCU with given logic elements.

The following values and sets can be derived from Table 7.1: $R_1 = 3, R_2 = 4, \tau = \{\tau_1, \tau_2, \tau_3\}, T = \{T_1, \dots, T_4\}, \Psi = \{D_5, D_6, D_7\}, \Phi = \{D_1, \dots, D_4\}$. The logic

circuit of CMCU $MU_1(\Gamma_{13})$ is shown in Fig. 7.3, assuming that the set of micro-operations $Y = \{y_1, \dots, y_5\}$ can be derived from the operator vertices of interpreted GSA.

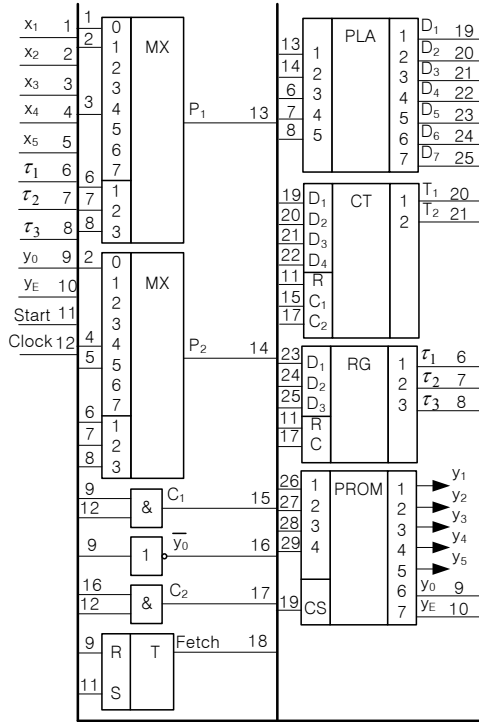


Fig. 7.3 Logic circuit of CMCU $MU_1(\Gamma_{13})$

Analysis of the logic circuit of block M (Fig. 7.2) shows that only 3/8 of potential capabilities of multiplexers are used, because their 5 data inputs are not in use. Let us define the utilization coefficient of multiplexer by the following relation

$$K_g = \frac{L_g}{S} \quad (g = 1, \dots, G_0), \tag{7.10}$$

where $L_g = |X(P_g)|$. It means that the mutual utilization effectiveness of multiplexers of the block M is characterised by the following coefficient:

$$K_M = \left(\sum_{g=1}^{G_0} K_g \right) / G. \tag{7.11}$$

In our example the values of these coefficients can be found as: $K_1 = K_2 = K_M = 0,375$.

Two following methods are proposed in [4] to increase the value of coefficient K_M :

- refined state encoding, when the states with conditional transitions are encoded first;
- transformation of state codes into codes of logical conditions, based on introduction of the special code transformer to the FSM S_1 .

In the first case set of states A_1 is divided in two classes A_1^1 and A_1^2 . Class A_1^1 contains the states with conditional transitions and states with unconditional transitions belong to class A_1^2 . States $a_m \in A_1^1$ are encoded first; the encoding is executed in such a manner, that R_1-R_{17} leftmost bits of the state codes contain zeros, where

$$R_{17} = \lceil \log_2(|A_1^1| + 1) \rceil. \tag{7.12}$$

Digit 1 in (7.12) is added because the code of initial state $a_1 \in A_1$ should contain zeros only. Coefficient K_M is increased, in comparison with arbitrary state encoding used, if the following condition takes place:

$$R_{17} < R_1. \tag{7.13}$$

In our example, there are sets $A_1^1 = \{a_2, a_5, a_6\}$ and $A_1^2 = \{a_3, a_4\}$. It leads to the state codes: $K(a_1) = 000$, $K(a_2) = 001$, $K(a_5) = 010$, $K(a_6) = 011$, $K(a_3) = 100$, $K(a_4) = 101$. Implementation of block M corresponding to this state encoding is shown in Fig. 7.4.

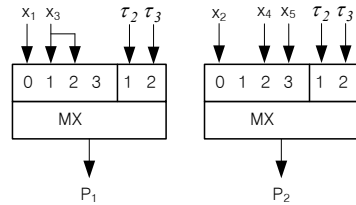


Fig. 7.4 Implementation of block M for refined state encoding

Calculation of K_m coefficients for this circuit gives the following values: $K_1 = K_2 = K_M = 0.75$. Besides, the numbers of control and data inputs are reduced in comparison with logic circuit shown in Fig. 7.2. Obviously, it results in smaller total cost of block M .

Structural diagram of CMCU MU_1 with code transformer TC_1 is shown in Fig. 7.5, where TC_1 generates codes of logical conditions on the base of state codes.

The number of outputs of the code transformer TC_1 is determined by the following formula:

$$R_{18} = \sum_{g=1}^{G_0} R_{18}^g. \tag{7.14}$$

Here symbol R_{18}^g stays for the number of variables used to encode logical conditions $x_l \in X(P_g)$, $g = 1, \dots, G_0$. In our example the following values can be obtained: $|X(P_1)| = 2$, $R_{18}^1 = 1$, $|X(P_2)| = 3$ and $R_{18}^2 = 2$. It results in the set of variables $Z = \{z_1, z_2, z_3\}$. Let us encode logical conditions $x_l \in X$ as shown in Table 7.4.

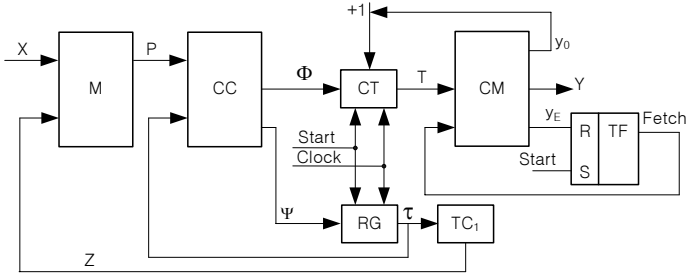


Fig. 7.5 Structural diagram of CMCU with transformation of state codes

Table 7.4 Table of logic conditions encoding for CMCU $MU_1(I_{13})$

$X(P_1)$	$K(x_l)$	$X(P_2)$	$K(x_l)$	
	z_1		z_2	z_3
x_1	0	x_2	0	0
x_2	1	x_4	0	1
-	-	x_5	1	0

For this encoding, the following logic circuit for the block M of CMCU $MU_1(I_{13})$ can be implemented (Fig. 7.6).

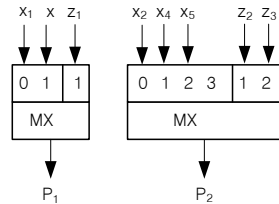


Fig. 7.6 Logic circuit of block M for CMCU $MU_1(I_{13})$

The application of transformer TC_1 gives the following coefficients: $K_1 = 1, K_2 = 0.75, K_M = 0.875$. Thus, the coefficient of effectiveness has the highest value in this last case, but the use of code transformer TC_1 , causes consumption of some resources of the chip.

In order to construct the system of output functions for code transformer TC_1 , namely

$$Z = Z(T), \tag{7.15}$$

it is necessary to find the table of code transformer TC_1 with columns $a_m, K(a_m), Z(P_1), \dots, Z(P_{G_0}), m$, where column $Z(P_g)$ contains variables $z_r \in Z$, encoding logical conditions $x_l \in X(P_g)$. In our case this table includes 4 lines (Table 7.5), equal to the cardinal number of the set A_1^1 .

For the state a_5 we have $p_1 = x_3, K(x_3) = 1$. It means that variable z_1 should be written on the intersection of column $Z(P_1)$ and row a_5 . By analogy, variable z_3

Table 7.5 Table of code transformer TC_1 of CMCU $MU_1(\Gamma_{13})$

a_m	$K(a_m)$	$Z(P_1)$	$Z(P_2)$	m
a_1	000	–	–	1
a_2	001	z_1	–	2
a_5	010	z_1	z_3	3
	011	–	z_2	4

should be written on the intersection of column $Z(P_2)$ and row a_5 because $p_2 = x_4$ and $K(x_4) = 01$. System (7.15), derived from the table of code transformer, gives the following expression:

$$z_r = \bigvee_{m=1}^{M_1} C_{rm} A_m, \tag{7.16}$$

where C_{rm} is a Boolean variable equal to 1 iff variable $z_r = 1$ for state $a_m \in A_1^1$ ($r = 1, \dots, R_{18}$). For example, the following equations can be derived from Table 7.5: $z_1 = \bar{\tau}_2 \tau_3 \vee \tau_2 \bar{\tau}_3$; $z_2 = \tau_2 \tau_3$; $z_3 = \tau_2 \bar{\tau}_3$. Let us point out, that in this particular case, the refined state encoding allows minimization of the number of literals in system (7.15).

Let us denote by the symbol MCU_1 the control unit U_1 with refined state encoding and by MLU_1 the control unit U_1 with transformation of state codes in the codes of logical conditions. Synthesis methods used for CMCU MCU_1 and MLU_1 are some modifications of the synthesis methods applied for CMCU MU_1 .

In the case of CMCU MCU_1 the step of refined state encoding is added after the point 5 of the synthesis method used for CMCU MU_1 . In case of CMCU MLU_1 , two more steps are added, namely coding of logical conditions and construction of the table for code transformer TC_1 .

The number of multiplexers in block M and, therefore, the number of combinational circuit inputs can be reduced due to the transformation of initial GSA [4]. For example, it can be found that $G_0 = 3$ in case of the GSA Γ_{14} fragment shown in (Fig. 7.7).

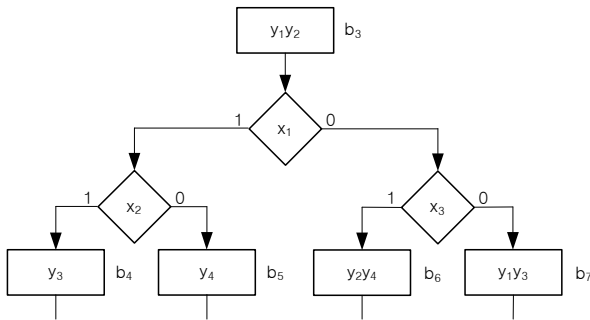


Fig. 7.7 Fragment of GSA Γ_{14}

Let us transform this fragment introducing additional vertex b_{20} (Fig. 7.8), assuming that $M_2 = 19$.

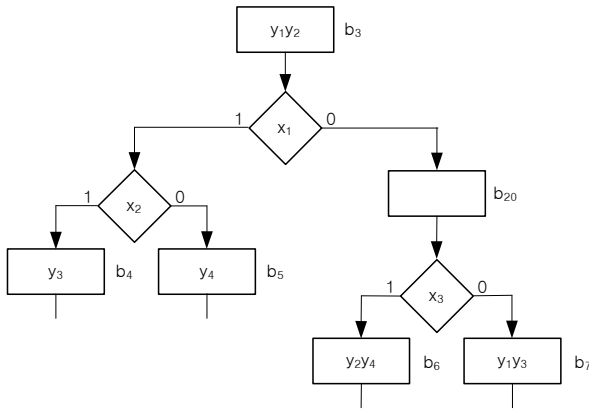


Fig. 7.8 The transformed fragment of GSA Γ_{14}

Now $G_0 = 2$, but introduction of the vertex b_{20} results in the increase of such parameters as the numbers of OLC and states of the Mealy FSM S_1 and the time of control algorithm execution. It is clear, that this transformation allows change the number of combinational circuit inputs from 1 to G_0 , where G_0 is the characteristic of initial GSA Γ . The method of GSA transformation can be applied for any from CMCU MU_1 , MCU_1 and MLU_1 . Let the symbol M_g denote the block M , having g outputs ($g = 1, \dots, G_0$). The application of logical conditions encoding, together with other methods discussed here gives the following models of CMCU: $M_1U_1, \dots, M_{G_0}U_1, \dots, M_{G_0}CU_1, M_1LU_1, \dots, M_{G_0}LU_1$.

Obviously, the transformation of GSA can be applied for any CMCU U_i ($i = 1, \dots, 57$). Let us discuss application possibility of other optimization methods which can be used for basic models of CMCU.

7.2 Encoding of logical conditions for basic models of CMCU

States of Mealy FSM S_1 are sources of codes of logical conditions used in the CMCU MU_1 . One of the three following objects can be used for encoding of logical conditions:

- codes of states $a_m \in A_1$ (MU_1);
- rightmost bits of the state codes (MCU_1);
- transformed state codes (MLU_1).

In consequence, logical conditions $x_l \in X$ can be encoded using feedback variables of the combinational circuit CC either directly (MU_1, MCU_1), or after some

transformation (MLU_1). Analysis of basic models U_1-U_{15} shows that the following objects can be used as a source of logical condition codes:

- codes of states $a_m \in A_1$ of addressing FSM (U_1);
- output addresses of OLC $\alpha_g \in C$ (U_2-U_4, U_7);
- codes of the classes of pseudoequivalent OLC $\alpha_g \in C^1$ (U_5, U_6, U_{10});
- codes of OLC $\alpha_g \in C$ (U_8, U_9, U_{11});
- codes of elementary OLC $\alpha_g \in C_E$ (U_{12}, U_{13}, U_{15});
- codes of the classes of pseudoequivalent EOLC $\alpha_g \in C_E^1$ (U_{14}).

Let us take as objects of CMCU: states $a_m \in A_1$ (U_1), or outputs of OLC $\alpha_g \in C$ (U_2-U_4, U_7), or equivalence classes $B_i \in \Pi_C$ (U_5, U_6, U_{10}), or OLC $\alpha_g \in C$ (U_8, U_9, U_{11}), or elementary OLC $\alpha_g \in C_E$ (U_{12}, U_{13}, U_{15}), or equivalence classes $B_i \in \Pi_E$ (U_{14}). Let us take a source of logical condition code as the object code. Obviously, there are three approaches to logical conditions encoding for CMCU [4], shown in Fig. 7.9:

- trivial encoding of objects (MU_i);
- refined encoding of objects (MCU_i);
- transformation of object codes (MLU_i).

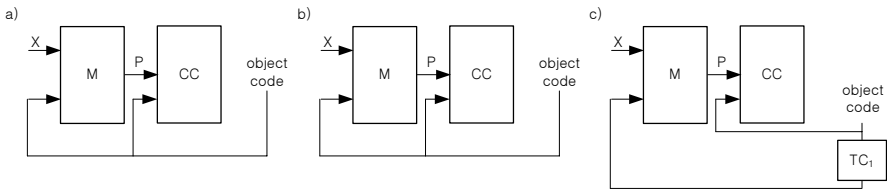


Fig. 7.9 Three approaches to logical conditions encoding

In the first case (Fig. 7.9a) codes of objects are used to generation of both functions P and outputs of the circuit CC. In the second case, refined codes of objects are used to generate functions P, and these refined codes are represented by the rightmost bits of object codes (Fig. 7.9b). In the third case, functions P depend on transformed codes of objects, which are generated by a special code transformer TC_1 (Fig. 7.9c). Application of block TC_1 makes sense only in the case, when total hardware amount used for implementation of logic circuits of blocks M and TC_1 is smaller than the hardware amount used for implementation of the block M in case of CMCU MCU_i .

In all these cases, table of logical conditions encoding should be built, in which first column includes an object, second column contains either the code of object (MU_i), or the refined code of object (MCU_i), or the transformed code of object (MLU_i). Other columns of the table contain variables p_1, \dots, p_{G_0} . System of functions P is generated in the following form:

$$p_g = \bigvee_{k=1}^K C_{kl} O_{kl} \quad (g = 1, \dots, G_0). \tag{7.17}$$

In this formula, C_{kl} is a Boolean variable equal to 1 iff variable $p_g \in P$ replaces logical condition $x_l \in X$ for some object number k ; O_k is a conjunction of variables, representing a code of logical condition $x_l \in X$ for some object number $k(k = 1, \dots, K)$.

In case of CMCU MLU_i , the table of code transformer TC_1 should be built in order to generate refined object codes. This table should include the following columns: an object, object code, $Z(p_1), \dots, Z(p_G)$, where the line k from column $Z(p_g)$ includes variables, equal to 1 in the code of logical condition $x_l \in X$, and some object number $k(k = 1, \dots, K)$. This table serves as the base for derivation of functions

$$z_r = \bigvee_{k=1}^K C_{rk} O_k \quad (r = 1, \dots, R_{18}(MCU_i)), \quad (7.18)$$

where C_{rk} is a Boolean variable, equal to 1 iff the bit r of code $K(x_l)$ of object number k is equal to 1 ($k = 1, \dots, K$); $R_{18}(MCU_i)$ is the cardinal number of set of variables, used to encode logical conditions for CMCU MCU_i ($i = 1, \dots, 57$).

Synthesis method used for CMCU MU_i can be considered as some expansion of the one used for CMCU U_i , where the following steps are added:

- construction of the encoding table of logical conditions;
- transformation of the transition table for CMCU U_i ($i = 1, \dots, 57$).

In case of CMCU MCU_i the stage of refined object encoding should be also added. In case of CMCU MLU_i the steps of encoding logical conditions $x_l \in X$ using additional variables $z_r \in Z$, as well as construction of the table of code transformer TC_1 are also necessary.

Let us now consider an example of CMCU $U_8(\Gamma_{15})$ synthesis using all approaches to encoding of logical conditions. Let Table 7.6 be the transition table for CMCU $U_8(\Gamma_{15})$.

The following sets and parameters can be found from this table for the CMCU $U_8(\Gamma_{15})$: set of OLC $C^1 = \{\alpha_1, \dots, \alpha_6\}$, $R_6 = 3$, sets of input memory functions for register RG, $\Psi = \{D_5, D_6, D_7\}$, and for counter CT, $\Phi = \{D_1, D_2, D_3, D_4\}$, sets of variables $\tau = \{\tau_1, \tau_2, \tau_3\}$, $T = \{T_1, \dots, T_4\}$, $R_7 = 4$, set of logical conditions $X = \{x_1, \dots, x_6\}$ with $L = 6$. Operational linear chains $a_g \in C^1$ are used here as objects, variables $\tau_r \in \tau$ are used for encoding of these objects. Let us discuss synthesis examples for CMCU $M_{G_0}U_8(\Gamma_{15})$, $M_{G_0}CU_8(\Gamma_{15})$ and $M_{G_0}LU_8(\Gamma_{15})$; that is without transformation of the initial GSA $\Gamma_{15}(U_8)$. For the sake of simplicity, we omit subscript G_0 in further considerations.

Structural diagram for CMCU $MU_8(\Gamma)$ is shown in Fig. 7.10.

Analysis of Table 7.6 shows that $G_0 = 2$ and we have the set $P = \{p_1, p_2\}$. Let us construct an encoding table of logical conditions for CMCU $MU_8(\Gamma_{15})$ (Table 7.7).

In this particular case system (7.17) can be written in the form:

$$\begin{aligned} p_1 &= \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 x_1 \vee \bar{\tau}_1 \bar{\tau}_2 \tau_3 x_2 \vee \bar{\tau}_1 \tau_2 \tau_3 x_5; \\ p_2 &= \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 x_3 \vee \bar{\tau}_1 \bar{\tau}_2 \tau_3 x_4 \vee \bar{\tau}_1 \tau_2 \tau_3 x_6 \vee \tau_1 \bar{\tau}_2 \tau_3 x_3. \end{aligned} \quad (7.19)$$

System (7.17) describes logic circuit of block M which, in our case, is shown in Fig. 7.11.

Table 7.6 Table of code transformer TC_1 of CMCU $MU_1(\Gamma_{13})$

α_i	$K(\alpha_i)$	α_j	$K(\alpha_j)$	X_h	Φ_h	Ψ_h	h
α_1	000	α_2	001	x_1	D_4	D_7	1
		α_3	010	$\bar{x}_1 x_3$	D_2	D_6	2
		α_5	100	$\bar{x}_1 \bar{x}_3$	$D_1 D_3$	D_5	3
α_2	001	α_3	010	$x_2 x_4$	$D_3 D_4$	D_6	4
		α_4	011	$x_2 \bar{x}_4$	D_2	$D_6 D_7$	5
		α_6	101	\bar{x}_2	D_1	$D_5 D_7$	6
α_3	010	α_4	011	1	$D_1 D_2$	$D_6 D_7$	7
α_4	011	α_3	010	x_5	$D_3 D_4$	D_6	8
		α_1	000	$\bar{x}_5 x_6$	D_1	-	9
		α_5	100	$\bar{x}_5 \bar{x}_6$	$D_1 D_3$	D_5	10
α_5	100	α_4	011	1	D_2	$D_6 D_7$	11
α_6	101	α_2	001	x_3	D_4	D_7	12
		α_1	000	\bar{x}_3	D_1	-	13

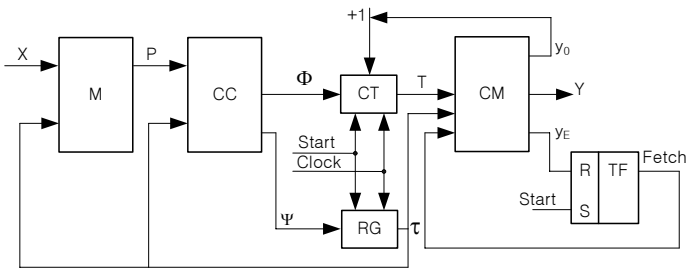


Fig. 7.10 Structural diagram for CMCU MU_8

Table 7.7 Table for encoding logical conditions for $MU_8(\Gamma_{15})$

a_g	$K(a_g)$	p_1	p_2	a_g	$K(a_g)$	p_1	p_2
a_1	000	x_1	x_3	a_4	011	x_5	x_6
a_2	001	x_2	x_4	a_5	100	-	-
a_3	010	-	-	a_6	101	-	x_3

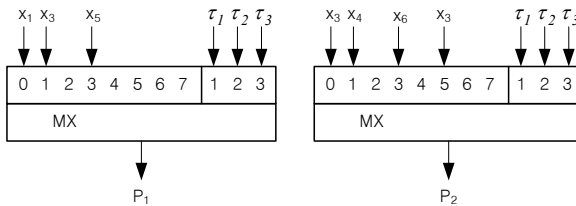


Fig. 7.11 Logic circuit of block M of CMCU $MU_8(\Gamma_{15})$

The following values for coefficients of block M can be found in our particular case: $K_1 = 0.375, K_2 = 0.5, K_M = 0.4375$. It means that less than 50% of multiplexer

potential is used. The coefficient K_M can be increased due to the refined encoding of OLC.

Let us divide set C^1 into two following classes: class C_1^1 (conditional transitions) and class C_2^1 (unconditional transitions). In our case we have the classes $C^1 = \{\alpha_1, \alpha_2, \alpha_4, \alpha_6\}$ and $C_2^1 = \{\alpha_3, \alpha_5\}$. Let us encode OLC $\alpha_g \in C^1$ in the following manner: $K(\alpha_1) = 000, K(\alpha_2) = 001, K(\alpha_4) = 010, K(\alpha_6) = 011, K(\alpha_3) = 100, K(\alpha_5) = 101$. Next we construct the encoding table for logical conditions of CMCU $MCU_8(\Gamma_{15})$, where OLC are represented by refined codes (Table 7.8).

Table 7.8 Table for encoding logical conditions of CMCU $MCU_8(\Gamma_{15})$

α_g	$C(a_g)$	p_1	p_2	α_g	$C(a_g)$	p_1	p_2
α_1	00	x_1	x_3	α_4	10	x_5	x_6
α_2	01	x_2	x_4	α_6	11	-	x_3

In this table symbol $C(a_g)$ stands for the refined code of OLC $\alpha_g \in C$. In this particular case, variables from the set $\tau' = \{\tau_2, \tau_3\}$ are used to represent the codes $C(\alpha_g)$. Structural diagram of CMCU MCU_8 is practically the same as the structural diagram of CMCU MU_8 , but in the former elements of set τ' are connected with the inputs of block M.

In case of CMCU $MCU_8(\Gamma_{15})$, system (7.17) has the form: $p_1 = \bar{\tau}_2 \bar{\tau}_3 x_1 \vee \bar{\tau}_2 \tau_3 x_2 \vee \tau_2 \bar{\tau}_3 x_5; p_2 = \bar{\tau}_2 \bar{\tau}_3 x_3 \vee \bar{\tau}_2 \tau_3 x_4 \vee \tau_2 \bar{\tau}_3 x_6 \vee \tau_2 \tau_3 x_3$ and corresponds to the logic circuit, shown in Fig. 7.12.

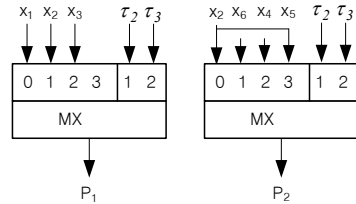


Fig. 7.12 Logic circuit of block M for CMCU $MCU_8(\Gamma_{15})$

Now we have the coefficient values: $K_1 = 0.75, K_2 = 1, K_M = 0.875$. It means that consumption effectiveness of multiplexers is increased twice, in comparison with CMCU $MU_8(\Gamma_{15})$. Obviously, it is the best value of coefficient K_M , possible for our particular example.

Let us construct transformed transition table for CMCU $MCU_8(\Gamma_{15})$. This transformation is reduced to simple replacement of logical conditions $x_l \in X$ by variables $p_g \in P$ and to using refined codes of OLC. This is shown in Table 7.9.

System of Boolean functions (7.4)–(7.5) can be obtained from this table, as for example, the equation: $D_1 = F_3 \vee F_6 \vee F_7 \vee F_9 \vee F_{10} \vee F_{13} = \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 \bar{p}_1 \bar{p}_2 \vee \dots \vee \bar{\tau}_1 \tau_2 \tau_3 \bar{p}_2$.

Table 7.9 Transformed transition table for CMCU $MCU_8(\Gamma_{15})$

α_i	$K(\alpha_i)$	α_j	$K(\alpha_j)$	P_h	Φ_h	Ψ_h	h
α_1	000	α_2	001	p_1	D_4	D_7	1
		α_3	100	$\bar{p}_1 p_2$	D_2	D_5	2
		α_5	101	$\bar{p}_1 \bar{p}_2$	$D_1 D_3$	$D_5 D_7$	3
α_2	001	α_3	100	$p_1 p_2$	$D_3 D_4$	D_5	4
		α_4	010	$p_1 \bar{p}_2$	D_2	D_6	5
		α_6	011	\bar{p}_1	D_1	$D_6 D_7$	6
α_3	100	α_4	010	1	$D_1 D_2$	D_6	7
α_4	010	α_3	100	p_1	$D_3 D_4$	D_5	8
		α_1	000	$\bar{p}_1 p_2$	D_1	-	9
		α_5	101	$\bar{p}_1 \bar{p}_2$	$D_1 D_3$	$D_5 D_7$	10
α_5	101	α_4	010	1	D_2	D_6	11
α_6	011	α_2	001	p_2	D_4	D_7	12
		α_1	000	\bar{p}_2	D_1	-	13

Logic circuit of the addressing finite state machine for CMCU $MCU_8(\Gamma_{15})$ is shown in Fig. 7.13. Such elements of the CMCU circuit as the control memory CM, flip-flop TF and control signal y_E are not shown in Fig. 7.13. Obviously, the initial GSA Γ_{15} is necessary for finding the control memory content, but here we are interested here only in optimization of logic circuit for addressing FSM. As we can see, coding of logical conditions allows to reduce the number of inputs, needed for the combinational circuit CC from $L + R_6 = 9$ to $G_0 + R_6 = 5$, that is 1.5 times.

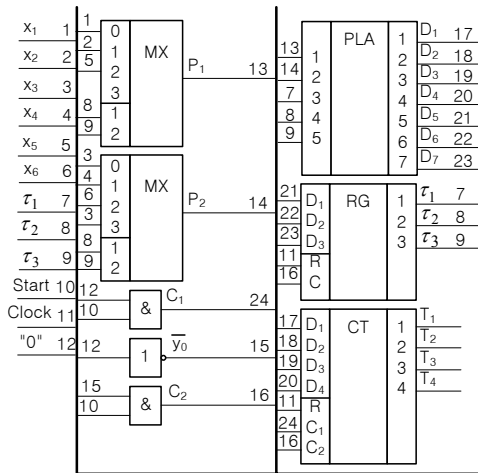


Fig. 7.13 Logic circuit of addressing FSM for CMCU $MCU_8(\Gamma_{15})$

The structural diagram of CMCU MLU_8 is shown in Fig. 7.14. As it was pointed already, there is no sense in introducing the code transformer TC_1 , because it does not lead to performance improvement of the block M.

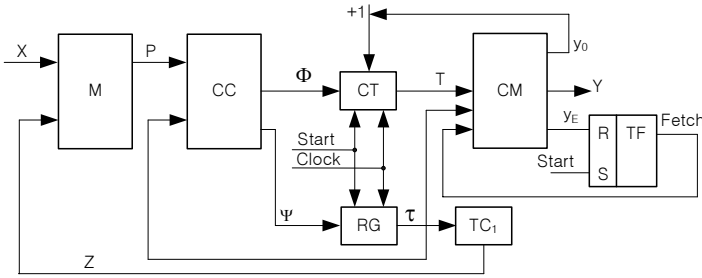


Fig. 7.14 Structural diagram of CMCU MLU_8

If a model of CMCU includes code transformer TC (CMCU U_6 , U_{10} and U_{14}), the two approaches are possible to generate codes of logical conditions. In the first case (parallel transformation), code transformer TC generates variables Z, encoding either OLC (U_5) or their classes (U_6 , U_{10} , U_{14}), and variables W, encoding logical conditions (Fig. 7.15a). In the second case, two different transformers TC and TC_1 are used (Fig. 7.15b), which corresponds to serial transformation.

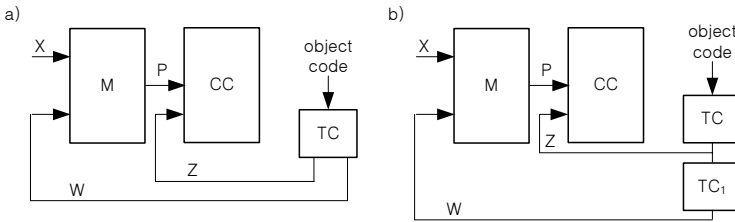


Fig. 7.15 Parallel **a** and serial **b** generation of codes of logical conditions

Let us point out that in case of CMCU U_5 , block TC corresponds to block AT, and functions Z correspond to functions τ . By analogy, functions Z correspond to functions τ for CMCU U_6 . Choice of the type of transformation (parallel or serial approach) depends on the result of hardware amount comparison for both cases.

Obviously, there are no problems with generation of CMCU models for MU_i , MCU_i and MLU_i . The same is true for corresponding synthesis methods. Further optimization of combinational circuit CC is possible due to transformation of the initial GSA. It allows to reduce both the number of inputs of combinational circuit CC and the number of outputs of block M. Introducing operator vertices into the initial GSA leads to increasing the number of bits in corresponding object codes. Besides, this transformation can increase the algorithm execution time. The value of parameter $|P|$ can not be found a priori for the CMCU M_gU_i and for its modifications. The choice can be made only after synthesis of logic circuits for models M_gU_i , when i is fixed and g is changed from 1 to G_0 . The value of g is chosen in such way that it corresponds to logic circuit with minimum hardware amount and performance satisfying corresponding project requirements.

Particular interest presents the case when $P = \{p_1\}$. Here, the number of transitions for any output O_g of OLC $\alpha_g \in C$ (or EOLC $\alpha_g \in C_E$) does not exceed 2 and the combinational circuit CC can be implemented using either PROM or RAM chips [4].

Let us discuss an example of logic circuit design for the CMCU $M_1CU_8(\Gamma_{16})$, using as initial information the transition table of CMCU $U_8(\Gamma_{16})$, where transition from output of any OLC $\alpha_g \in C$ depends on one logical condition only (Table 7.10).

Table 7.10 Transition table for CMCU $U_8(\Gamma_{16})$

α_g	$K(\alpha_g)$	α_m	$K(\alpha_m)$	X_h	Φ_h	Ψ_h	h
α_1	0000	α_2	0001	x_1	-	D_8	1
		α_5	0100	\bar{x}_1	-	D_6	2
α_2	0001	α_3	0010	1	D_3	D_7	3
α_3	0010	α_4	0011	x_3	-	D_7D_8	4
		α_6	0101	\bar{x}_3	-	D_6D_7	5
α_4	0011	α_7	0110	x_2	-	D_6D_7	6
		α_3	0010	\bar{x}_2	D_2D_3	D_7	7
α_5	0100	α_8	0111	1	-	$D_6D_7D_8$	8
α_6	0101	α_9	1000	x_4	-	D_5	9
		α_{10}	1001	\bar{x}_4	-	D_5D_8	10
α_7	0110	α_9	1000	x_4	-	D_5	11
		α_{10}	1001	\bar{x}_4	-	D_5D_8	12
α_8	0111	α_3	0010	1	-	D_7	13
α_9	1000	α_4	0011	x_5	D_2	D_7D_8	14
		α_9	1000	\bar{x}_5	D_2D_4	D_5	15

The following sets, values of main parameters and properties can be found from this table for CMCU $U_8(\Gamma_{16})$: $G_1 = 10$, $X = \{x_1, \dots, x_5\}$, $\Phi = \{D_1, \dots, D_4\}$, $\Psi = \{D_5, \dots, D_8\}$, $R_6 = R_4 = 4$, $T = \{T_1, \dots, T_4\}$, $\tau = \{\tau_1, \dots, \tau_4\}$, $\alpha_{10} \notin C^1$.

Let us check whether the refined OLC encoding makes sense in this particular case. As the set of this OLC is $C_1^1 = \{\alpha_1, \alpha_3, \alpha_4, \alpha_6, \alpha_7, \alpha_9\}$, it means that $R_{17} = 3$ bits is sufficient to encode OLC $\alpha_g \in C_1^1$. It is also less than $R_6 = 4$. Thus, the refined OLC encoding permits to reduce the hardware amount of the block M. The codes of OLC $\alpha_g \in C$ are shown in Fig. 7.16.

The following refined OLC codes depending on elements of the set $\tau' = \{\tau_2, \tau_3, \tau_4\}$ can be found from this Karnaugh map: $C(\alpha_1) = 000$, $C(\alpha_3) = 001, \dots, C(\alpha_9) = 101$. Let us construct the table of logical conditions encoding (Table 7.11).

Table 7.11 Table of encoding of logical conditions for CMCU $M_1CU_8(\Gamma_{16})$

α_g	$C(\alpha_g)$	p_1	α_g	$C(\alpha_g)$	p_1
α_1	000	x_1	α_6	011	x_4
α_2	001	x_2	α_7	100	x_4
α_4	010	x_3	α_8	101	x_5

$\tau_3\tau_4$	00	01	11	10
$\tau_1\tau_2$	00	01	11	10
00	α_1	α_3	α_6	α_4
01	α_1	α_2	α_3	α_5
11	*	*	*	*
10	α_6	α_4	*	*

Fig. 7.16 Refined OLC codes for CMCU $M_1CU_8(\Gamma_{16})$

Table 7.11 serves to generate disjunctive normal forms for the functions

$$p_1 = p_1(\tau', X). \tag{7.20}$$

The equation $p_1 = \bar{\tau}_2\bar{\tau}_3\bar{\tau}_4x_1 \vee \dots \vee \tau_1\bar{\tau}_2\bar{\tau}_3x_5$ can be found from Table 7.11. System (7.20) is used to implement logic circuit of the block M for CMCU M_1CU_8 .

In order to implement logic circuit of the block CC using PROM chips, initial transition table should be transformed. This transformation consists on replacement of the column X_h by column P_1 , and on duplication of table lines, corresponding to unconditional transitions. One of these lines contains $p_1 = 0$, whereas second line contains $p_1 = 1$. It follows from the fact that address of a PROM cell is always determined by concatenation $K(\alpha_g) * p_1$, where $*$ is a concatenation sign. In this particular case, the transformed transition table for CMCU $M_1CU_8(\Gamma_{16})$ includes $2G_1 = 18$ lines (Table 7.12).

Table 7.12 Transformed transition table for CMCU $M_1CU_8(\Gamma_{16})$

α_g	$K(\alpha_g)$	α_m	$K(\alpha_m)$	P_1	Φ_h	Ψ_h	h
α_1	0000	α_2	0111	1	–	$D_6D_7D_8$	1
		α_5	0110	0	–	D_6D_7	2
α_2	0111	α_3	0001	1	D_3	D_8	3
		α_3	0001	0	D_3	D_8	4
α_3	0001	α_4	0010	1	–	D_7	5
		α_6	0011	0	–	D_7D_8	6
α_4	0010	α_7	0100	1	–	D_6	7
		α_3	0001	0	D_2D_3	D_8	8
α_5	0110	α_8	1000	1	–	D_5	9
		α_8	1000	0	–	D_5	10
α_6	0011	α_9	0101	1	–	D_6D_8	11
		α_{10}	1001	0	–	D_5D_8	12
α_7	0100	α_9	0101	1	–	D_6D_8	13
		α_{10}	1001	0	–	D_5D_8	14
α_8	1000	α_3	0001	1	–		15
		α_3	0001	0	–	D_8	16
α_9	0101	α_4	0010	1	D_2	D_7	17
		α_9	0101	0	D_2D_4	D_6D_6	18

7.3 Encoding collections of microoperations in CMCU

Main goal of encoding collections of microoperations is minimizing the control memory size [2,8]. There are two main approaches to this kind of encoding, namely:

- maximal encoding collections of microoperations;
- encoding the fields of compatible microoperations.

In the first case there are two objects for encoding, namely expanded microinstructions $Y_q \subseteq Y \cup \{y_0, y_E\}$ and collections of microoperations $Y_q \subseteq Y$. Let us discuss these methods more thoroughly, because they result in different organizations of block for generation of microoperations (Fig. 7.19).

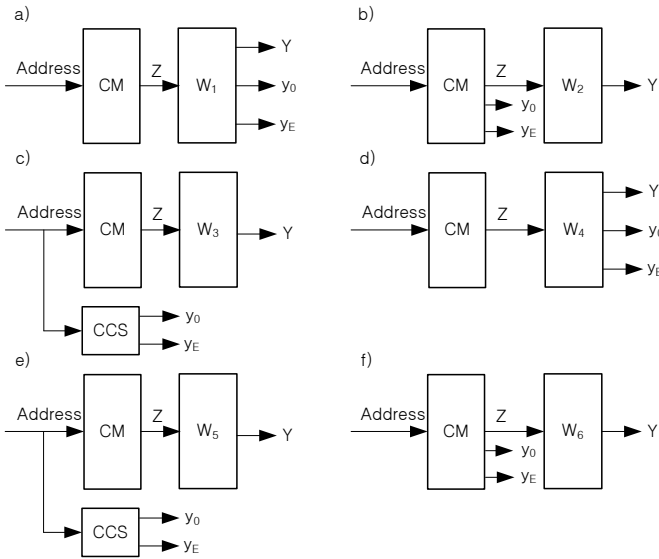


Fig. 7.19 Organization of block for generation of microoperations

In the first case (Fig. 7.19a), each expanded microinstruction Y_q ($q = 1, \dots, M_3$) corresponds to the binary code $K(Y_q)$ having R_{15} bits, where the value of R_{15} is determined using (6.9). Variables $z_r \in Z$ are used for encoding expanded microinstructions, where $|Z| = R_{15}$. Microoperations $y_n \in Y$ and additional variables y_0, y_E are generated by block W_1 , with logic circuit implemented with PROM or RAM chips. The approach results in CMCU U_iW_1 ($i = 1, \dots, 57$).

In the second case, each collection of microoperations $Y_q \subseteq Y$ ($q = 1, \dots, M_4$) is encoded by a binary code $K(Y_q)$ with R_{16} bits, where value of R_{16} is determined by formula (6.13). Variables $z_r \in Z$, where $|Z| = R_{16}$, are used for this encoding. Microoperations $y_n \in Y$ are generated by block W_2 , implemented with PROM or RAM chips; functions y_0, y_E are generated by the control memory CM. This organization results in the CMCU U_iW_2 ($i = 1, \dots, 57$) model, corresponding to Fig. 7.19b.

In the third case (Fig. 7.19c) collections of microoperations are also encoded, but functions y_0, y_E are generated by the block CCS. This approach allows reduction of the control memory size and leads to models of CMCU $U_i W_3$, where $i = 1, \dots, 57$.

In all other cases, the logic circuit of block W_i ($i = 4, 5, 6$) can be implemented using FPGA, PLA or PAL chips. Either expanded microinstructions (Fig. 7.19d) or collections of microoperations (Fig. 7.19e) can be encoded. The encoding is executed in such a way, that maximum possible number of microoperations is implemented using only one macrocell of the particular PLD being in use. If collections of microoperations are encoded, y_0, y_E can be generated either by block CCS (Fig. 7.19e) or by control memory CM (Fig. 7.19f). For example, the algorithm ESPRESSO [6, 7] can be used for this kind of encoding. Application of these methods leads to models of the CMCU $U_i W_j$, where $i = 1, \dots, 57; j = 4, 5, 6$.

Synthesis methods, used for compositional microprogram control units $U_i W_j$ ($i = 1, \dots, 57; j = 4, 5, 6$), can be considered as some modifications of synthesis methods for the corresponding CMCU U_i , where the following steps are added:

- maximal encoding of objects (expanded microinstructions or collections of microoperations);
- construction of the table for block W_j ;
- construction of the table for block CCS (if necessary).

Because of such variety of possible organizations, it is necessary to find the best model, giving minimum hardware amount, as well as satisfactory performance. Hardware amount can be estimated as the number of equivalent gates, needed for logic circuits of different models.

Let the GSA Γ_{17} have the following parameters: $M_2 = 47, R_2 = 6, N = 9$ and let the following expanded microinstructions be derived from the operator vertices of the transformed GSA Γ_{17} : $Y_1 = \{y_0, y_1, y_3\}, Y_2 = \{y_1, y_3\}, Y_3 = \{y_0, y_2, y_4, y_5, y_7\}, Y_4 = \{y_0, y_1, y_3, y_8\}, Y_5 = \{y_1, y_3, y_8\}, Y_6 = \{y_1, y_3, y_8, y_E\}, Y_7 = \{y_0, y_2, y_4, y_7\}, Y_8 = \{y_0, y_1, y_2, y_5\}, Y_9 = \{y_1, y_2, y_5\}, Y_{10} = \{y_1, y_2, y_5, y_E\}, Y_{11} = \{y_0, y_3, y_4, y_8\}, Y_{12} = \{y_0, y_4, y_8, y_9\}, Y_{13} = \{y_4, y_8, y_9\}, Y_{14} = \{y_0, y_5, y_6, y_7\}, Y_{15} = \{y_0, y_6, y_7\}, Y_{16} = \{y_6, y_7\}, Y_{17} = \{y_0, y_9\}, Y_{18} = \{y_9\}$. Let us discuss different organizations of the circuit used for generating microoperations for CMCU $U_1 W_j(\Gamma_{17})$.

In case of CMCU $U_1 W_1(\Gamma_{17})$ the number $R_{15} = 5$ bits is sufficient to encode $M_3 = 18$ expanded microinstructions. To implement the control memory of CMCU $U_1(\Gamma_{17})$ it is necessary to have $V_0 = 2^{R_2} \cdot (N + 2) = 64 \cdot 11 = 704$ bits of PROM. To implement the control memory of CMCU $U_1 W_1(\Gamma_{17})$ we need $V_1 = 2^{R_2} \cdot R_{15} = 64 \cdot 5 = 320$ bits of PROM, and finally, for the implementation of logic circuit of block W_1 it is sufficient to have $K_1 = 2^{R_{15}} \cdot (N + 2) = 32 \cdot 11 = 352$ bits. In general, the number of bits, saved for $U_1 W_1$ in comparison with the CMCU, can be found from the following expression

$$\eta_1 = \frac{V_0}{V_1 + K_1}. \quad (7.21)$$

In the discussed case expression (7.21) gives $\eta_1 = 704/672 = 1.05$. Because $\eta_1 > 1$, application of this encoding makes sense. Let us encode the expanded

microinstructions as follows: $K(Y_1) = 00000, \dots, K(Y_{18}) = 10001$. The table of block W_1 gives equations serving for generation of microoperations $y_n \in Y$ and of the additional functions y_0, y_E :

$$y_n = y_n(Z), y_0 = y_0(Z), y_E = y_E(Z). \quad (7.22)$$

This table includes columns $z_1, \dots, z_{R_{15}}, y_0, y_1, \dots, y_N, y_E$. The first five lines, for our example, are shown in Table 7.13.

Table 7.13 Fragment of the table for block W_1 of CMCU $U_1W_1(\Gamma_{17})$

z_1	z_2	z_3	z_4	z_5	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_E
0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1	0	1	1	0	1	0	0	0
0	0	0	1	1	1	1	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0

Analysis of the expanded microinstructions shows that there are $M_4 = 10$ different collections of microoperations in GSA Γ_{17} , namely: $Y_1 = \{y_1, y_3\}$, $Y_2 = \{y_2, y_4, y_5, y_7\}$, $Y_3 = \{y_1, y_3, y_8\}$, $Y_4 = \{y_2, y_4, y_7\}$, $Y_5 = \{y_1, y_2, y_5\}$, $Y_6 = \{y_3, y_4, y_8\}$, $Y_7 = \{y_4, y_8, y_9\}$, $Y_8 = \{y_5, y_6, y_7\}$, $Y_9 = \{y_6, y_7\}$, $Y_{10} = \{y_9\}$. In case of CMCU $U_1W_2(\Gamma_{17})$, these collections can be encoded using $R_{16} = 4$ variables. It is sufficient to have $V_2 = 2^{R_2} \cdot (R_{16} + 2) = 64 \cdot 6 = 384$ bits to implement the control memory for CMCU $U_1W_2(\Gamma_{17})$, and the logic circuit of block W_2 can be implemented using only $K_2 = 2^{R_{16}} \cdot N = 16 \cdot 9 = 144$ bits. In general case, the number of bits, which can be saved, can be found from the following expression

$$\eta_2 = \frac{V_0}{V_2 + K_2}. \quad (7.23)$$

In our particular case, expression (7.23) gives $\eta_2 = 704/528 = 1.33$. Comparison of the values η_1 and η_2 shows that, in this particular case, the application of block W_2 for the CMCU U_1 is more preferable, than application of block W_1 .

Let us encode collections of microoperations for CMCU $U_1W_2(\Gamma_{17})$ as follows: $K(Y_1) = 0000, \dots, K(Y_{10}) = 1001$. First five lines of the table for block W_2 of the CMCU $U_1W_2(\Gamma_{17})$ with this kind of encoding the collections of microoperations are shown in Table 7.14.

Addresses of microinstructions should be known in case of CMCU $U_1W_3(\Gamma_{17})$ and $U_1W_5(\Gamma_{17})$, if we want to construct the table of block CCS. This step can be executed in trivial way, and therefore it is not discussed here. In general, the number of saved bits η_3 can be found from the following expression:

$$\eta_3 = \frac{2^{R_2} \cdot (N + 2)}{2^{R_2} \cdot R_{16} + 2^{R_2} \cdot N + Q_3}. \quad (7.24)$$

Table 7.14 Fragment of the table for block W_2 of CMCU $U_1W_2(\Gamma_{17})$

z_1	z_2	z_3	z_4	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
0	0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	1	1	0	1	0	0
0	0	1	0	1	0	1	0	0	0	0	1	0
0	0	1	1	0	1	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0	1	0	0	0	0

In this expression parameter Q_3 is equal to the number of bits, needed to implement the logic circuit of block CCS.

If some other PLDs are used to generate microoperations (neither PROM, nor RAM), the encoding methods used for expanded microinstructions and collections of microoperations are the same. Let us discuss an example of implementation of logic circuit for block W_6 of CMCU $U_1W_6(\Gamma_{17})$. Using the algorithm ESPRESSO, the following codes of collections of microoperations can be obtained (Fig. 7.20).

		z_3z_4			
		00	01	11	10
z_1z_2	00	Y_1	Y_3	Y_5	*
	01	*	Y_6	Y_2	Y_8
	11	*	Y_7	Y_4	Y_9
	10	Y_{10}	*	*	*

Fig. 7.20 Codes of collections of microoperations for CMCU $U_1W_6(\Gamma_{17})$

Microoperations $y_n \in Y$ are represented by the following Boolean equations in case of CMCU U_1W_6 :

$$y_n = \bigvee_{q=1}^{M_4} C_{nq}Z_q \quad (n = 1, \dots, N), \tag{7.25}$$

where C_{nq} is a Boolean variable equal to 1 iff $y_n \in Y_q$; Z_q is a conjunction of variables $z_r \in Z$, corresponding to code $K(Y_q)$ of collection of microoperations $Y_q \subseteq Y$. In case of CMCU $U_1W_6(\Gamma_{17})$, system (7.25) takes the form: $y_1 = Z_1 \vee Z_3 \vee Z_5$; $y_2 = Z_2 \vee Z_4 \vee Z_5$; $y_3 = Z_1 \vee Z_3 \vee Z_6$; $y_4 = Z_2 \vee Z_4 \vee Z_6 \vee Z_7$; $y_5 = Z_2 \vee Z_5 \vee Z_8$; $y_6 = Z_8 \vee Z_9$; $y_7 = Z_2 \vee Z_4 \vee Z_8 \vee Z_9$; $y_8 = Z_3 \vee Z_6 \vee Z_7$; $y_9 = Z_7 \vee Z_{10}$.

Using codes from Fig. 7.20 and taking into account the insignificant input assignments, the following final expressions can be got for this particular example: $y_1 = \bar{z}_3\bar{z}_4$; $y_2 = z_1z_2$; $y_3 = \bar{z}_1\bar{z}_3$; $y_4 = z_2z_4$; $y_5 = z_1\bar{z}_3$; $y_6 = z_1\bar{z}_2$; $y_7 = z_1z_4$; $y_8 = \bar{z}_1z_2$; $y_9 = \bar{z}_1z_3$.

Let the disjunctive normal form for the function y_n include H_n terms and each term F_h include m_h literals. In this case, complexness of the function y_n can be estimated using the following expression

$$C_n = \sum_{h=1}^{H_n} m_h + H_n. \quad (7.26)$$

Complexness of logic circuit obtained for block W_6 can be estimated using the following expression

$$V_6 = \sum_{n=1}^N C_n. \quad (7.27)$$

In the discussed example we have $C_n = 2$ for any microoperation $y_n \in Y$ and $V_6 = 18$ bits. The economy coefficient η_6 can be determined using the following expression

$$\eta_6 = \frac{2^{R_2} \cdot (N+2)}{2^{R_2} \cdot (R_{16} + 2) + V_6}. \quad (7.28)$$

In our case it can be found that $\eta_6 = 64 \cdot 11 / (64 \cdot 6 + 18) = 704/402 = 1.75$.

Let us point out, that complexness of block CCS can be estimated using the same approach as the one used for block W_6 . Choice of particular model for implementation of microoperations is reduced to finding the value $\eta_0 = \max(\eta_1, \dots, \eta_6)$. Obviously, the method giving maximum value η_0 of the economy coefficient should be chosen.

In case of encoding the fields of compatible microoperations, set $Y \cup \{y_0, y_E\}$ is divided into classes Y^1, \dots, Y^J , forming a partition Π_Y . Each class $Y^j \in \Pi_Y$ includes microoperations, which are placed into different operator vertices of the transformed GSA [5]. There are many effective methods used to find the partition Π_Y [1], but they are not discussed here. Let class $Y^j \in \Pi_Y$ include m_j elements, denoted here by symbols y_n^j , where $n \in \{1, \dots, N, 0, E\}$, $j = 1, \dots, J$. Let us encode each element y_n^j by a binary code $K(y_n^j)$ with the following number of bits:

$$R^j = \lceil \log_2(m_j + 1) \rceil \quad (j = 1, \dots, J). \quad (7.29)$$

In this case it is sufficient to have R_{19} variables to encode microoperations $y_n \in Y$ and y_0, y_E , where

$$R_{19} = \sum_{j=1}^J R^j. \quad (7.30)$$

These variables form set Z .

Each expanded microinstruction Y_q of CMCU corresponds to the code

$$K(Y_q) = K(y_q^1) * K(y_q^2) * \dots * K(y_q^J), \quad (7.31)$$

where $*$ is the concatenation sign, y_q^j is microoperation $y_n \in Y \cup \{y_0, y_E\}$, such that $y_n \in Y^j$ and $y_n \in Y_q$ ($j = 1, \dots, J$). Codes (7.31) are kept in the control memory CM and microoperations $y_n \in Y^j$ are formed as outputs of decoders DC_j (Fig. 7.21), where $j = 1, \dots, J$.

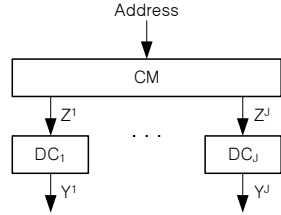


Fig. 7.21 Formation of microoperations for CMCU with encoding the fields of compatible microoperations

As can be seen from Fig. 7.21, variables $Z^j \subseteq Z$ are used to encode microoperations $y_n \in Y^j$, and decoders DC_1, \dots, DC_J form the block D_J . Let us denote CMCU U_i , for which formation of microoperations is described above by symbol $U_i D_J$.

Synthesis methods used for CMCU $U_i D_J (i = 1, \dots, 57)$ can be considered as some expansions of synthesis methods for CMCU U_i , obtained by adding the following steps:

- encoding of microoperations;
- construction of the transformed table of control memory content.

Let us consider an example of synthesis for CMCU $U_1 D_J (\Gamma_{18})$, where the structural diagram of CMCU $U_1 D_J$ is shown in Fig. 7.22, and the transformed GSA $\Gamma_{18}(U_1)$ is shown in Fig. 7.23.

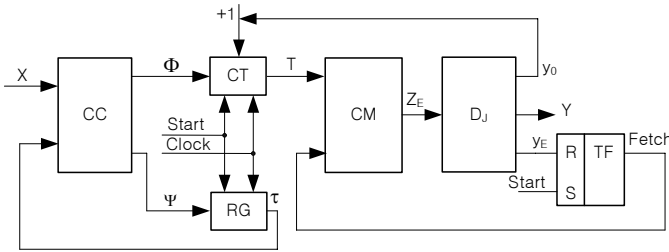


Fig. 7.22 Structural diagram of CMCU $U_1 D_J$

Application of procedure P_1 to GSA $\Gamma_{18}(U_1)$ leads to the set of OLC $C = \{\alpha_1, \dots, \alpha_5\}$, where $\alpha_1 = \langle b_1, b_2 \rangle, I_1^1 = b_1; \alpha_2 = \{b_3, \dots, b_6\}, I_2^1 = b_3; \alpha_3 = \{b_7, b_8, b_9\}, I_3^1 = b_7; \alpha_4 = \langle b_{10}, b_{11} \rangle, I_4^1 = b_{10}, I_4^2 = b_{11}; \alpha_5 = \langle b_{12} \rangle, I_5^1 = b_{12}$. Operator vertices of the GSA $\Gamma_{18}(U_1)$ contain the following expanded microinstructions: $Y_1 = \{y_0, y_1, y_2, y_3\}, Y_2 = \{y_4, y_5, y_6\}, Y_3 = \{y_0, y_7, y_8\}, Y_4 = \{y_0, y_8, y_9, y_{10}\}, Y_5 = \{y_0, y_7, y_8, y_9\}, Y_6 = \{y_9, y_{12}\}, Y_7 = \{y_0, y_6, y_{13}\}, Y_8 = \{y_9, y_{11}\}, Y_9 = \{y_6, y_{13}\}, Y_{10} = \{y_E, y_8, y_{13}\}$. These microinstructions contain microoperations forming the set $Y_0 = \{y_0, y_E, y_1, \dots, y_{13}\}$.

Application of the well-known procedures, given in [5], to the set Y_0 results in the partition $\Pi_Y = \{Y^1, \dots, Y^4\}$, where $Y^1 = \{y_1, y_4, y_7, y_{10}, y_{11}, y_{12}, y_{13}\}, Y^2 = \{y_2, y_5, y_8\}, Y^3 = \{y_3, y_6, y_9\}, Y^4 = \{y_0, y_E\}, m_1 = 7, m_2 = m_3 = 3, m_4 = 2$. Using expressions (7.29)–(7.30), it could be found that $Z^1 = \{z_1, z_2, z_3\}, Z^2 = \{z_4, z_5\}$,

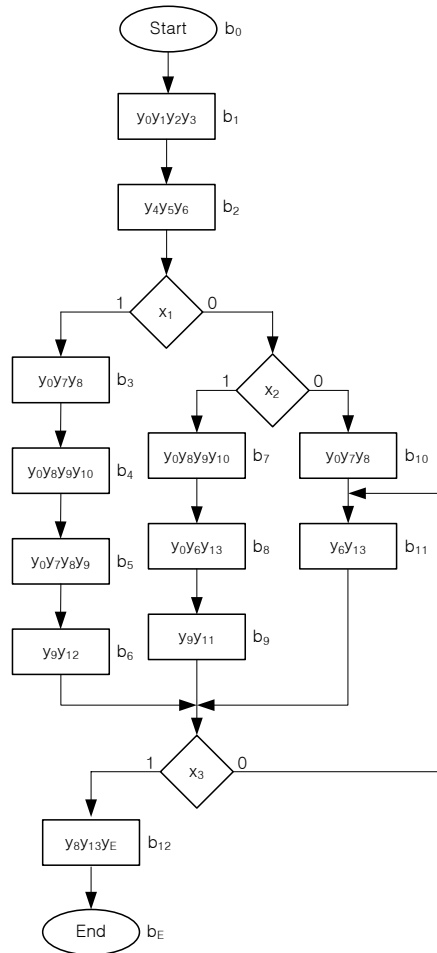


Fig. 7.23 Transformed GSA $\Gamma_{18}(U_1)$

$Z^3 = \{z_6, z_7\}$, $Z^4 = \{z_8, z_9\}$. Microoperation codes for the CMCU $U_1 D_J(\Gamma_{18})$ are shown in Table 7.15, which can be used to determine both the inputs and outputs of decoders designed for block D_J .

Table 7.15 determines the codes given by (7.31), as for example, $K(Y_1) = 001010101 = K(y_1) * K(y_2) * K(y_3) * K(y_0)$. In this table, sign \emptyset corresponds to the case when microoperations of a particular class do not belong to the given expanded microinstruction.

Application of procedure P_2 to GSA $\Gamma_{18}(U_1)$ results in microinstructions addresses shown in Fig. 7.24.

If we want to find control memory content it is sufficient to replace any vertex $b_q \in B_1$ by its code $K(Y_q)$ corresponding to the collection of microoperations from this very vertex (Fig. 7.25).

Table 7.15 Table of microoperation encoding y_q^1 for CMCU $U_1 D_J(\Gamma_{18})$

y_q^1	$K(y_q^1)$	y_q^2	$K(y_q^2)$	y_q^3	$K(y_q^3)$	y_q^4	$K(y_q^4)$
	$z_1 z_2 z_3$		$z_4 z_5$		$z_6 z_7$		$z_8 z_9$
\emptyset	000	\emptyset	00	\emptyset	00	\emptyset	00
y_1	001	y_2	01	y_3	01	y_0	01
y_4	010	y_5	10	y_6	10	y_E	10
y_7	011	y_8	11	y_9	11		
y_{10}	100						
y_{11}	101						
y_{12}	110						
y_{13}	111						

$T_1 T_2$	00	01	11	10
$T_3 T_4$				
00	b_1	b_5	b_9	*
01	b_2	b_6	b_{10}	*
11	b_3	b_7	b_{11}	*
10	b_4	b_8	b_{12}	*

Fig. 7.24 Addresses of microinstructions of CMCU $U_1 D_J(\Gamma_{18})$

$T_1 T_2$	00	01	11	10
$T_3 T_4$				
00	001010101	011111101	101001100	*
01	010101000	110001100	011110001	*
11	011110001	100111101	111001000	*
10	100111101	111001001	111110010	*

Fig. 7.25 Control memory content for CMCU $U_1 D_J(\Gamma_{18})$

Information represented by Fig. 7.25 is used to design logic circuit of the control memory CM.

In order to design logic circuit of block CC, it is necessary to construct Boolean equations for functions Φ and Ψ , depending on variables X and τ . As in all cases discussed previously, the structure table of addressing FSM S_1 for CMCU U_1 is the base to find these equations. The structure table is constructed using block representation of the interpreted GSA, marked by the states of Mealy FSM. In the discussed case, this representation for GSA $\Gamma_{18}(U_1)$ is shown in Fig. 7.26.

Using Fig. 7.26, the following sets and values can be found: $A = \{a_1, a_2, a_3\}$, $M_1 = 3, R_1 = 2, \tau = \{\tau_1, \tau_2\}$. Let $K(a_1) = 00, K(a_2) = 01, K(a_3) = 10$. The transition table of CMCU $U_1 D_J(\Gamma_{18})$ is given below (Table 7.16).

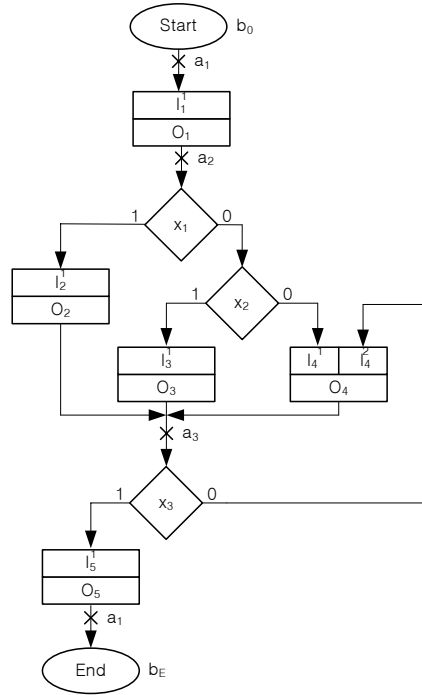


Fig. 7.26 Block representation of GSA $\Gamma_{18}(U_1)$

For example, the following equation can be found using this table: $D_1 = F_4 \vee F_5 \vee F_6 = \bar{\tau}_1 \tau_2 \bar{x}_1 x_2 \vee \tau_1 \bar{\tau}_2$. Logic circuit of CMCU $U_1 D_J(\Gamma_{18})$ is shown in Fig. 7.27, where outputs z_8 and z_9 correspond to signals y_0 and y_E respectively.

The number of control memory outputs can be reduced to the value

$$R_{20} = \lceil \log_2(N + 1) \rceil \tag{7.32}$$

due to verticalization [5] of the transformed GSA. In this case microinstruction format includes fields y_0 and FY (Fig. 7.28).

Table 7.16 Transition table of CMCU $U_1 D_J(\Gamma_{18})$

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Φ_h	Ψ_h	h
a_1	00	a_2	01	1	–	D_6	1
a_2	01	a_3	10	x_1	D_3	D_5	2
		a_3	10	$\bar{x}_1 x_2$	$D_2 D_3$	D_5	3
		a_3	10	$\bar{x}_1 \bar{x}_2$	$D_1 D_4$	D_5	4
a_3	10	a_1	00	x_3	$D_1 D_3 D_4$	–	5
		a_3	10	\bar{x}_3	$D_1 D_3$	D_5	6

The digit 1 is added in expression (7.32) to take into account the existence of variable y_E . The verticalization of GSA is reduced to splitting each operator vertex $b_q \in B_1$ into $|Y(b_q)|$ vertices, where each new vertex includes only one unique microoperation $y_n \in Y \cup \{y_E\}$ [5]. For example, the operator vertex b_1 of GSA $\Gamma_{18}(U_1)$ corresponds to three operator vertices of the verticalized GSA $V\Gamma_{18}(U_1)$ (Fig. 7.29).

In case of CMCU $U_1D_1(\Gamma_{18})$, where symbol D_1 means that $\Pi_Y = Y \cup \{y_E\}$, we can find that $R_{20} = 4$. It means that the control memory size is more than two times smaller, in comparison with this size for the CMCU $U_1D_J(\Gamma_{18})$. The main disadvantage of verticalization is the decrease of CMCU performance, because the average algorithm execution time becomes longer, in comparison with the same parameter of the equivalent CMCU U_1D_J . To make both times closer, some sophisticated mechanism is proposed in [5]. It is based on the data-path launching occurring only when all microoperations $y_n \in Y(b_q)$ are placed in some additional register. We do not discuss this problem in our book.

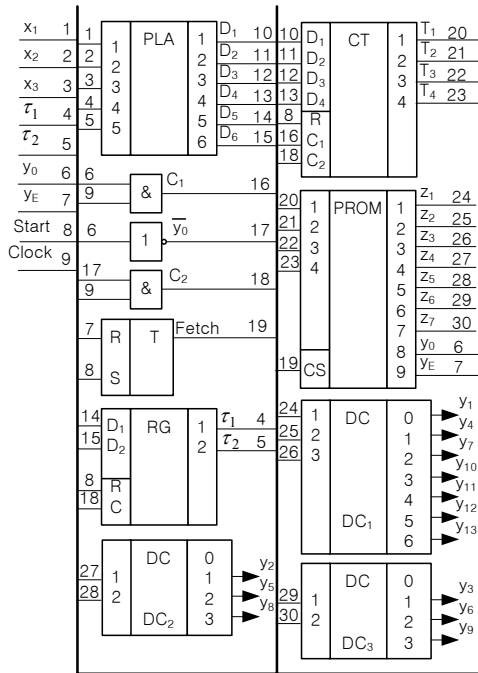
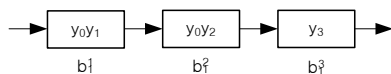


Fig. 7.27 Logic circuit of CMCU $U_1D_J(\Gamma_{18})$



Fig. 7.28 Microinstruction format for CMCU with verticalization of transformed GSA

Fig. 7.29 Splitting of operator vertex b_1 of GSA $\Gamma_{18}(U_1)$

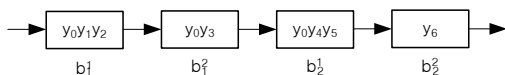


Transformation of GSA can be executed in such a manner, that the number of classes j of partition Π_Y corresponds to the condition

$$1 < j < J, \tag{7.33}$$

which means that parameter j has some intermediate value, between its extremes 1 and J . If, for example, transformation of the GSA $\Gamma_{18}(U_1)$ is executed in such a way that microoperations $y_n \in Y^2 \cup Y^3$ are always placed in the different operator vertices, the partition Π_Y includes three classes only. They have the form: $Y^1, Y^2 = \{y_2, y_3, y_5, y_6, y_8, y_9\}$ and $Y^3 = \{y_0, y_E\}$. This kind of transformation for the case of OLC α_1 is shown in Fig. 7.30.

Fig. 7.30 The transformation of OLC α_1



After applying similar transformation for all OLC $\alpha_g \in C$ we get the partition with the following parameters: $R^1 = 3, R^2 = 3, R^3 = 2$ and $R_{20} = 8$. Let us point out that no solution of the following problem is available today: the choice of some value of the parameter j for which the hardware amount in the system $\langle CM, D_j \rangle$ is optimal, for a given performance level of the resulting control unit, in which the principle of encoding the fields of compatible microoperations is used. Some additional researches should be conducted to solve this particular problem.

7.4 Synthesis of multilevel circuits of CMCU

Combined application of methods discussed previously permits to generate models of CMCU, for which the number of levels is higher than in case of basic CMCU models. Wide variety of possible structures is represented by Table 7.17, which is constructed by analogy with Table 2.2 for the multilevel structures of Mealy FSM, or by Table 2.5 (Moore FSM). The level B is occupied in this table by basic models of CMCU, which are also multilevel structures. The number of levels is different for basic models and varies from two (U_1-U_{15}) to four ($U_{46}-U_{57}$). Thus, the number of possible levels in the discussed structures is changed from two to six. As we remember, parameter G used in Table 7.17 determines the number of variables used to encode logical conditions, and parameter J determines the number of fields of the compatible microoperations. Parameters G and J are calculated using an initial graph-scheme of algorithm Γ . Decrease of the initial values of these parameters is connected with transformation of the initial GSA Γ , leading to reduction of the

resulting digital system performance due in turn to the increase of average number of cycles, necessary for execution of the control algorithm.

Table 7.17 Multilevel models of CMCU

Levels	A	B	C
Blocks	M_1	U_1	W_1
	M_1C	\vdots	\vdots
	M_1L	U_5	W_6
	\vdots		D_1
	M_G		\vdots
	M_GC		D_J
	M_GL		

Let k_i be the number of structures with i levels for some model of CMCU, V_i the total number of CMCU structures with i levels ($i \leq 6$). Now, Table 7.18 will be used to estimate the total number of CMCU structures with i levels ($2 \leq i \leq 6$).

It is clear, that Table 7.18 describes

$$V_0^1 = V_2 + \dots + V_6 = 399 + 1197G + 57J + 171GJ \tag{7.34}$$

different CMCU structures. In case of FSM with average complexness [3] we have $G = J = 6$, and formula (7.34) determines $V^1 = 14079$ different CMCU structures for some initial GSA Γ . The CMCU synthesis method, based on combination of different optimization approaches, can be considered as some expansion of the synthesis method used for a particular basic model of CMCU. For example, let us consider the method applied for CMCU $M_2U_9W_3$, its structural diagram shown in Fig. 7.31.

The expression $M_2U_9W_3$ determines the CMCU with encoded logical conditions, where each transition depends on at most two variables (it is determined by symbol M_2), with code sharing and optimal OLC encoding (determined by symbol U_9) and maximal encoding of collections of microoperations, where the block generating microoperations is implemented using PROM chips and additional signals y_0 and y_E are implemented by an additional block CCS (determined by symbol W_3). Synthesis method for CMCU $M_2U_9W_3$ includes the following steps:

1. Transformation of the initial GSA Γ .
2. Construction of OLC set for transformed GSA $\Gamma(M_2U_9W_3)$.
3. Construction of the partition Π_c for set C^1 .
4. Optimal encoding of OLC $\alpha_g \in C^1$.
5. Encoding components of OLC $\alpha_g \in C$.
6. Encoding collections of microoperations.
7. Encoding logical conditions.

8. Construction of the transition table of CMCU.
9. Construction of the control memory content.
10. Construction of the table for block CCS.
11. Construction of the table for block .
12. Synthesis of the CMCU logic circuit with given logical elements.

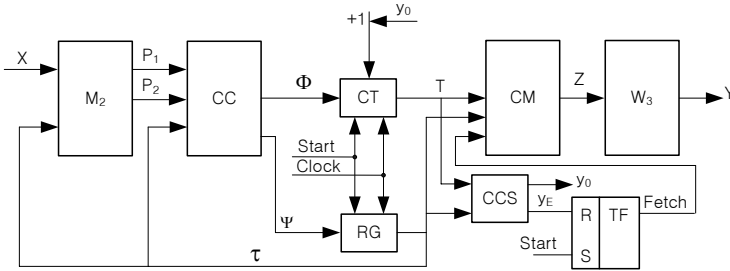


Fig. 7.31 Structural diagram of CMCU $M_2U_9W_3$

Let us consider an example of CMCU synthesis, where the graph-scheme of algorithm is shown in Fig. 7.32.

The transformation of GSA Γ_{19} is reduced here to the introduction of vertex b_{19} (as the first OLC of GSA Γ_{19}), and vertices b_{20} and b_{21} (to satisfy condition), and introduction of variable y_E into the vertex b_{13} and variable y_0 into the vertices, which do not represent the outputs of OLC. After these transformations, the transformed GSA $\Gamma_{19}(M_2U_9W_3)$ is obtained (Fig. 7.33).

Application of procedure P_1 to the transformed GSA $\Gamma_{19}(M_2U_9W_3)$ results in the set $C = \{\alpha_1, \dots, \alpha_8\}$, where $\alpha_1 = \langle b_{19} \rangle, I_1^1 = O_1 = b_{19}; \alpha_2 = \langle b_1, b_2, b_3 \rangle, I_2^1 = b_1, I_2^2 = O_2 = b_3; \alpha_3 = \langle b_4, \dots, b_7 \rangle, I_3^1 = b_4, I_3^2 = b_6; \alpha_4 = \langle b_8, b_9 \rangle, I_4^1 = b_8, O_4 = b_9; \alpha_5 = \langle b_{10}, \dots, b_{13} \rangle, I_5^1 = b_{10}, I_5^2 = b_{12}, O_5 = b_{13}; \alpha_6 = \langle b_{14}, \dots, b_{17} \rangle, I_6^1 = b_{14}, O_6 = b_{17}; \alpha_7 = \langle b_{20} \rangle, I_7^1 = O_7 = b_{20}; \alpha_8 = \langle b_{21} \rangle, I_8^1 = O_8 = b_{21}$. Let us point out that the set C^1 does not include OLC α_5 , because this OLC output contains the variable y_E .

Let us find the partition Π_C of set C^1 . In our case, we obtain $\Pi_C = \{B_1, \dots, B_4\}$, where $B_1 = \{\alpha_1\}, B_2 = \{\alpha_2, \alpha_3, \alpha_4, \alpha_6\}, B_3 = \{\alpha_7\}$ and $B_4 = \{\alpha_8\}$. Outcome of the optimal OLC $\alpha_g \in C$ encoding is shown in the Karnaugh map (Fig. 7.34).

For encoding OLC $\alpha_g \in C$, elements of the set $\tau = \{\tau_1, \tau_2, \tau_3\}$ are used. Let us point out that $R_6 = 3$, because $G = 8$. Taking into account the don't care code $K(\alpha_5) = 011$, the following codes of classes $B_i \in \Pi_C$ can be derived from the Karnaugh map: $K(B_1) = 000, K(B_2) = 1***, K(B_3) = 0*1, K(B_4) = 01*$.

Maximal length of OLC $\alpha_g \in C$ for the discussed example is equal to 4 ($L_{\max} = 4$), hence $R_7 = 2, T = \{T_1, T_2\}$. The codes of OLC components are: $K(b_{19}) = K(b_1) = K(b_4) = K(b_8) = K(b_{10}) = K(b_{14}) = K(b_{20}) = K(b_{21}) = 00; K(b_2) = K(b_5) = K(b_9) = K(b_{11}) = K(b_{15}) = 01; K(b_3) = K(b_6) = K(b_9) = K(b_{12}) = K(b_{16}) = 10; K(b_7) = K(b_{13}) = K(b_{17}) = 11$. The codes, used for OLC and its components, allow to find microinstruction addresses, based on code sharing principle (Fig. 7.35).

Table 7.18 Estimation of the number of CMCU models

Number of levels	Type of model	k_i	V_i		
2	U_1-U_{15}	15	$V_2 = 15$		
3	$M_gU_1-M_gU_{15}$	15G	$V_3 = 120 + 45G + 15J$		
	$M_gCU_1-M_gCU_{15}$	15G			
	$M_gLU_1-M_gLU_{15}$	15G			
	$U_1W_i-U_{15}W_i$	90			
	$U_1D_j-U_{15}D_j$	15J			
	$U_{16}-U_{45}$	30			
	$U_{46}-U_{57}$	12			
	$M_gU_1W_i-M_gU_{15}W_i$	90			
	$M_gCU_1W_i-M_gCU_{15}W_i$	90G			
	$M_gLU_1W_i-M_gLU_{15}W_i$	90G			
4	$M_gU_1D_j-M_gU_{15}D_j$	15GJ	$V_4 = 192 + 360G + 30J + 45GJ$		
	$M_gCU_1D_j-M_gCU_{15}D_j$	15GJ			
	$M_gLU_1D_j-M_gLU_{15}D_j$	15GJ			
	$M_gU_{16}-M_gU_{45}$	30G			
	$M_gCU_{16}-M_gCU_{45}$	30G			
	$M_gLU_{16}-M_gLU_{45}$	30G			
	$U_{16}W_i-U_{45}W_i$	180			
	$U_{16}D_j-U_{45}D_j$	30J			
	$M_gU_{46}-M_gU_{57}$	12G			
	$M_gCU_{46}-M_gCU_{57}$	12G			
	$M_gLU_{46}-M_gLU_{57}$	12G			
	$M_gU_{16}W_i-M_gU_{45}W_i$	180G			
	$M_gCU_{16}W_i-M_gCU_{45}W_i$	180G			
	5	$M_gLU_{16}W_i-M_gLU_{45}W_i$		180G	$V_5 = 72 + 576G + 12J + 90GJ$
		$U_{46}W_i-U_{57}W_i$		72	
		$U_{46}D_j-U_{57}D_j$		12J	
$M_gU_{16}D_j-M_gU_{45}D_j$		30GJ			
$M_gCU_{16}D_j-M_gCU_{45}D_j$		30GJ			
$M_gLU_{16}D_j-M_gLU_{45}D_j$		30GJ			
$M_gU_{46}W_i-M_gU_{57}W_i$		72G			
$M_gCU_{46}W_i-M_gCU_{57}W_i$		72G			
6		$M_gLU_{46}W_i-M_gLU_{57}W_i$	72G	$V_6 = 216G + 36GJ$	
		$M_gU_{46}D_j-M_gU_{57}D_j$	12GJ		
	$M_gCU_{46}D_j-M_gCU_{57}D_j$	12GJ			
	$M_gLU_{46}D_j-M_gLU_{57}D_j$	12GJ			

Operator vertices of the GSA $\Gamma_{19}(M_2U_9W_3)$ include $M_4 = 9$ different collections of microoperations, namely: $Y_1 = \emptyset$, $Y_2 = \{y_1, y_2\}$, $Y_3 = \{y_3\}$, $Y_4 = \{y_4, y_5\}$, $Y_5 = \{y_2, y_5, y_6\}$, $Y_6 = \{y_7\}$, $Y_7 = \{y_8, y_9\}$, $Y_8 = \{y_2, y_8, y_9\}$ and $Y_9 = \{y_1, y_{11}\}$. It is sufficient to have $R_{16} = 4$ variables for encoding these collections and, hence, $Z = \{z_1, \dots, z_4\}$. Let us encode these collections in the following manner: $K(Y_1) = 0000, \dots, K(Y_9) = 1000$.

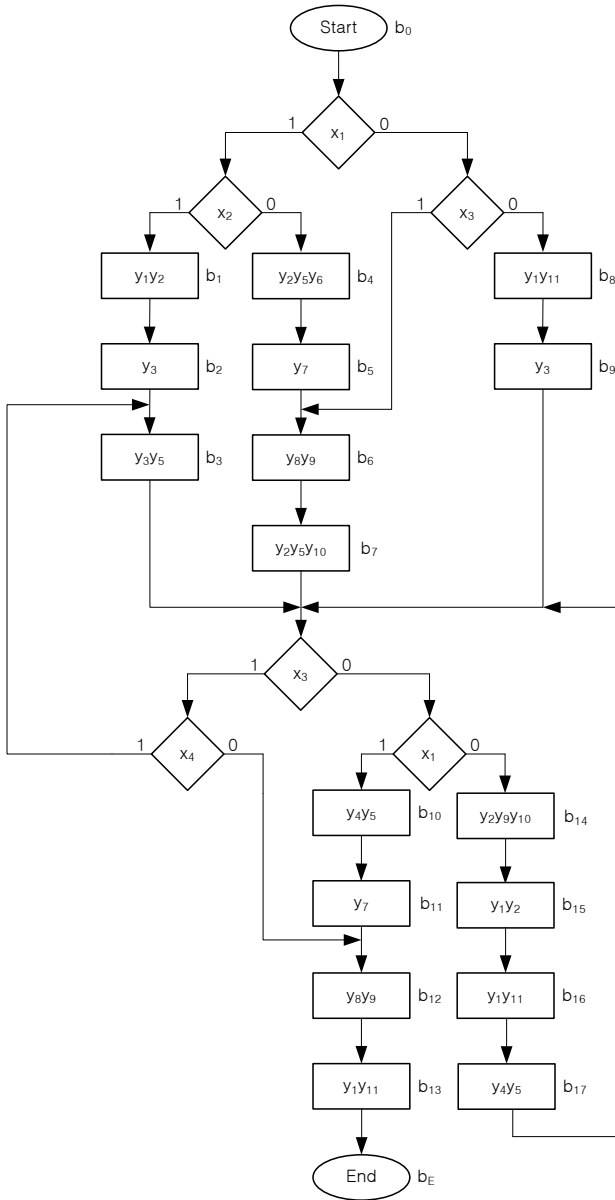


Fig. 7.32 Initial graph-scheme of algorithm Γ_{19}

According to conditions given in this example, we obtain the set $P = \{p_1, p_2\}$ and coding of logical conditions is represented by Table 7.19. This table determines the behavior of multiplexers MX_1 and MX_2 , and, hence, logic circuit of the block M_2 for CMCU $M_2U_9W_3(\Gamma_{19})$.

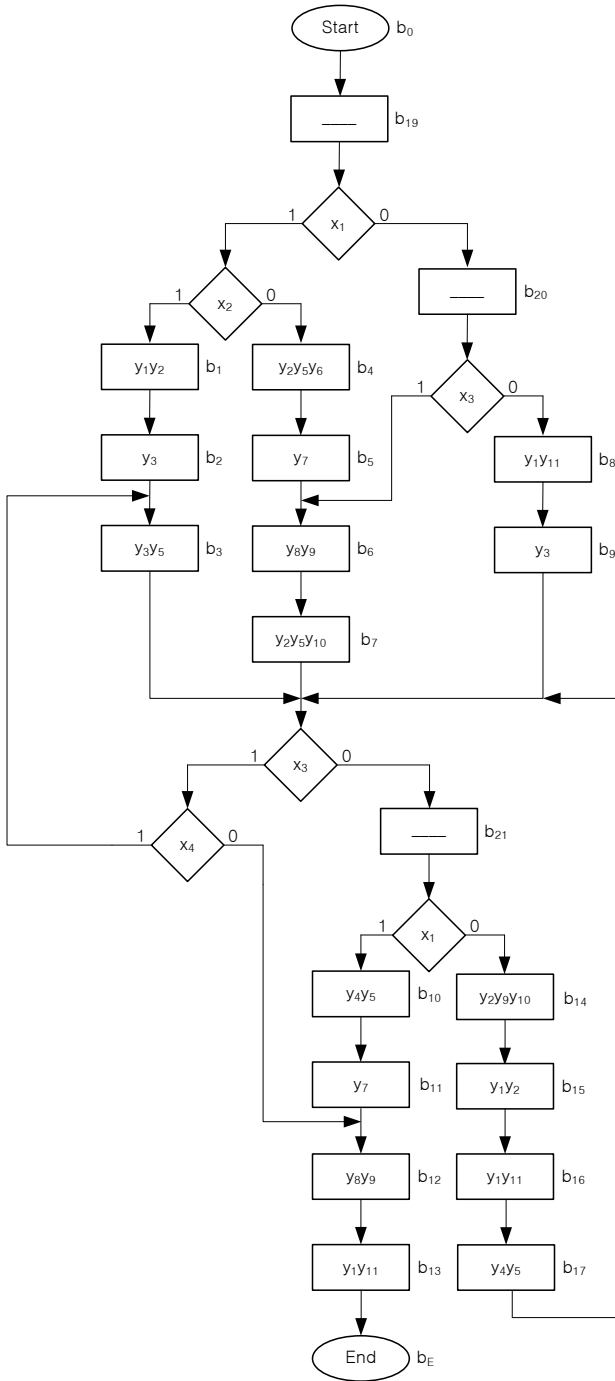


Fig. 7.33 Transformed GSA

Fig. 7.34 Optimal OLC codes for GSA $\Gamma_{19}(M_2U_9W_3)$

$\bar{x}_2\bar{x}_3$ x_1	00	01	11	10
0	α_1	α_7	α_5	α_8
1	α_2	α_3	α_4	α_6

$x_1x_2x_3$ T_1T_2	000	001	010	011	100	101	110	111
00	b_{19}	b_{20}	b_{21}	b_{10}	b_1	b_4	b_{14}	b_8
01	*	*	*	b_{11}	b_2	b_5	b_{15}	b_9
10	*	*	*	b_{12}	b_3	b_6	b_{16}	*
11	*	*	*	b_{13}	*	b_7	b_{17}	*

Fig. 7.35 Microinstruction addresses of CMCU $M_2U_9W_3(\Gamma_{19})$

Table 7.19 Coding of logical conditions for CMCU $M_2U_9W_3(\Gamma_{19})$

α_g	$K(\alpha_g)$	p_1	p_2
α_1	000	x_1	x_2
α_2	001	x_3	x_4
α_3	000	x_3	x_4
α_4	111	x_3	x_4
α_5	011	—	—
α_6	110	x_3	x_4
α_7	001	x_3	—
α_8	010	x_1	—

The following system of transition formulae for OLC $\alpha_g \in C^1$ should be found to construct the transition table of CMCU $M_2U_9W_3(\Gamma_{19})$:

$$\begin{aligned}
 O_1 &\rightarrow x_1x_2I_2^1 \vee x_1\bar{x}_2I_3^1 \vee \bar{x}_1I_7^1; \\
 O_2, O_3, O_4, O_6 &\rightarrow x_3x_4I_2^2 \vee x_3\bar{x}_4I_5^2 \vee \bar{x}_3I_8^1; \\
 O_7 &\rightarrow x_3I_3^2 \vee \bar{x}_3I_4^1; \\
 O_8 &\rightarrow x_1I_5^1 \vee x_1I_6^1.
 \end{aligned}
 \tag{7.35}$$

Next, outputs of OLC $\alpha_g \in C^1$ from the left part of each equation of system (7.35) are replaced by corresponding classes and logical conditions $x_l \in X$ in the right part are replaced by variables $p_g \in P$:

$$\begin{aligned}
 B_1 &\rightarrow p_1p_2I_2^1 \vee p_1\bar{p}_2I_3^1 \vee \bar{p}_1I_7^1; \\
 B_2 &\rightarrow p_1p_2I_2^2 \vee p_1\bar{p}_2 \vee \bar{p}_1I_8^1; \\
 B_3 &\rightarrow p_1I_3^2 \vee \bar{p}_1I_4^1; \\
 B_4 &\rightarrow p_1I_5^1 \vee \bar{p}_1I_6^1.
 \end{aligned}
 \tag{7.36}$$

System (7.36) is now used to construct the table of transitions with following columns: $B_i, K(B_i), I_g^j, A(I_g^j), P_h, \Psi_h, \Phi_h, h$, where input addresses should be taken from Fig. 7.35. In case of the CMCU $M_2U_9W_3(\Gamma_{19})$ this table includes $H_9(\Gamma_{19}) = 10$ lines (Table 7.20).

Table 7.20 Table of transitions for CMCU $M_2U_9W_3(\Gamma_{19})$

B_i	$K(B_i)$	I_g	$A(I_g)$	P_h	Ψ_h	Φ_h	h
B_1	000	I_2^1	10000	p_1p_2	D_1	-	1
		I_3^1	10100	$p_1\bar{p}_2$	D_1D_3	-	2
		I_7^1	00100	\bar{p}_1	D_3	-	3
B_2	100	I_2^2	10010	p_1p_2	D_1	D_4	4
		I_2^5	01110	$p_1\bar{p}_2$	D_2D_3	D_4	5
		I_8^1	01000	\bar{p}_1	D_2	-	6
B_3	0*1	I_3^2	10110	p_1	D_1D_3	D_4	7
		I_4^1	11100	\bar{p}_1	$D_1D_2D_3$	-	8
B_4	01*	I_5^1	01100	p_1	D_2D_3	-	9
		I_6^1	11000	\bar{p}_1	D_1D_2	-	10

It follows from Table 7.20, that $\Psi = \{D_1, D_2, D_3\}$, $\Phi = \{D_4, D_5\}$, and $D_1 = F_1 \vee F_2 \vee F_4 \vee F_7 \vee F_8 \vee F_{10} = \bar{\tau}_1 \bar{\tau}_2 \bar{\tau}_3 p_1 p_2 \vee \dots \vee \tau_1 \tau_2 \bar{p}_1$, for example. It is clear that $D_5 = 0$ and block CC has 4 outputs only.

The control memory content can be found by replacement of vertices $b_q \in B_1$ by corresponding codes of collections of microoperations $K(Y_q)$. The control memory content for our example is shown in Fig. 7.36.

	$\tau_1\tau_2\tau_3$							
T_1T_2	000	001	010	011	100	101	110	111
00	0000	0000	0000	0011	0001	0100	0111	1000
01	*	*	*	0101	0010	0101	0001	0010
10	*	*	*	0110	0011	0110	1000	*
11	*	*	*	1000	*	0111	0111	*

Fig. 7.36 Control memory content for CMCU $M_2U_9W_3(\Gamma_{19})$

Table of block CCS can be found by transformation of Fig. 7.34 into the Karnaugh maps for functions y_0 and y_E . These maps are used to get minimal forms for functions $y_0(\tau, T)$ and $y_E(\tau, T)$. For example, the Karnaugh map for function y_0 in case of the CMCU $M_2U_9W_3(\Gamma_{19})$ is shown in Fig. 7.37.

The following equation $y_0 = \tau_2 \tau_3 \bar{T}_2 \vee \tau_1 \tau_3 \bar{T}_2 \vee \tau_1 \bar{\tau}_3 \bar{T}_1 \vee \tau_1 \tau_2 \bar{T}_1 \vee \bar{\tau}_1 \bar{T}_1 T_2$ can be derived from this map. Using the same approach, equation $y_E = \tau_1 T_1 T_2$ can be also obtained. Both equations are used to design the logic circuit for block CCS.

$\tau_1\tau_2\tau_3$		000	001	010	011	100	101	110	111
		T_1T_2							
00		0	0	1	0	1	1	1	1
01		*	*	1	*	1	0	1	1
10		*	*	0	*	0	*	0	*
11		*	*	1	*	1	*	1	0

Fig. 7.37 Karnaugh map for function y_0

The table of block W_3 includes columns $Y_q, K(Y_q), y_n, q$. This block is represented by Table 7.21.

Table 7.21 Table of block W_3 for CMCU $M_2U_9W_3(I_{19})$

Y_q	$K(Y_q)$	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	q
Y_1	0000	0	0	0	0	0	0	0	0	0	0	0	1
Y_2	0001	1	1	0	0	0	0	0	0	0	0	0	2
Y_3	0010	0	0	1	0	0	0	0	0	0	0	0	3
Y_4	0011	0	0	0	1	1	0	0	0	0	0	0	4
Y_5	0100	0	1	0	0	1	1	0	0	0	0	0	5
Y_6	0101	0	0	0	0	0	0	1	0	0	0	0	6
Y_7	0110	0	0	0	0	0	0	0	1	1	0	0	7
Y_8	0111	0	0	0	0	0	0	0	0	1	1	0	8
Y_9	1000	1	0	0	0	0	0	0	0	0	0	1	9

Logic circuit of the CMCU $M_2U_9W_3(I_{19})$ is shown in Fig. 7.38. The multiplexers MX_1 and MX_2 represent block M_2 . Blocks CC and CCS are implemented using PLA chips. The control memory CM and the block generating microoperations are implemented with PROM chips.

Logic circuit for each model of the CMCU given in table 7.18 can be designed using the same approach as the one used in case of the CMCU $M_2U_9W_3(I_{19})$.

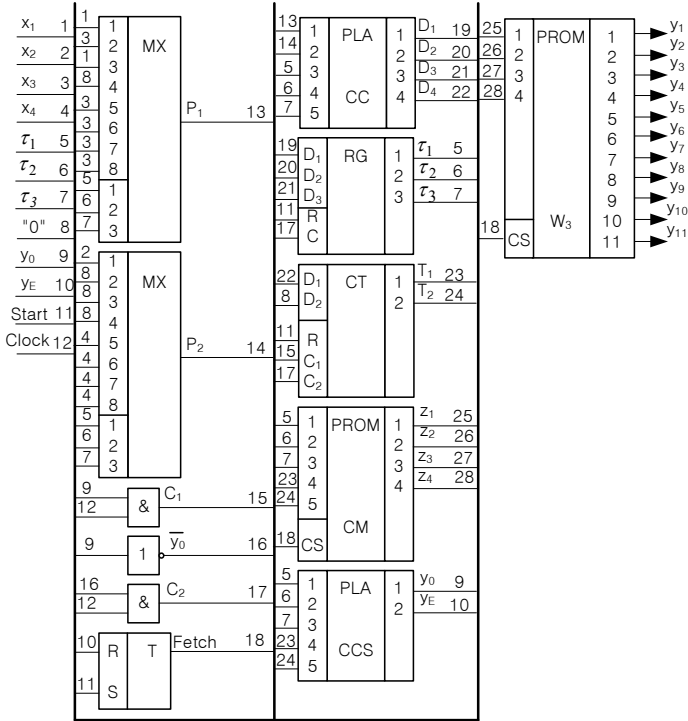


Fig. 7.38 Logic circuit of CMCU $M_2U_9W_3(I_{19})$

References

1. M. Adamski and A. Barkalov. *Architectural and Sequential Synthesis of Digital Devices*. University of Zielona Góra Press, 2006.
2. T. Agerwala. Microprogram optimization: a survey. *IEEE Transactions of Computers*, (10):962–973, 1976.
3. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
4. A. A. Barkalov. *Synthesis of Control Units with PLDs*. Donetsk National Technical University, 2002. (in Russian).
5. A. A. Barkalov and A. V. Palagin. *Synthesis of Microprogram Control Units*. IC NAS of Ukraine, Kiev, 1997. (in Russian).
6. T. Łuba, K. Jasiński, and B. Zbierchowski. *Specialized digital circuits in PLD i FPGA structures*. Wydawnictwo Komunikacji i Łączności, 1997. (in Polish).
7. G. De Micheli. Symbolic design of combinational and sequential logic implemented by two-level macros. *IEEE Transactions on Computer-Aided Design*, 5(9):597–616, 1986.
8. S. Schwartz. An algorithm for minimizing read-only memories for machine control. *IEEE 10th Annual Symposium on Switching and Automata Theory*, pages 28–33, 1968.
9. V. A. Skljarov. *Synthesis of automata on matrix LSI*. Nauka i Technika, Minsk, 1984. (in Russian).

Chapter 8

Synthesis of compositional microprogram control units with modified system of microinstructions

Abstract The chapter is devoted to CMCU optimization, based on modification of the microinstruction format. Proposed modifications permit to eliminate code transformers from the CMCU and provide reduction of hardware amount of circuits used in the FSM used for microinstruction addressing, as compared with the CMCU basic structure. This kind of optimization is leads to the increasing of the number of cycles, needed for execution of the control algorithms. This transformation causes sometimes the increase of control memory size. Next, the possibility of multilevel CMCU implementation is discussed and the method of optimal structure choice proposed. A particular CMCU structure is considered as optimal, if it guarantees minimum hardware amount and sufficient performance. This chapter is based on the results of common research performed with J. Bieganski (Poland).

8.1 Synthesis of CMCU with dedicated area of inputs

All compositional microprogram control units discussed previously have some common feature, namely generation of input addresses by the block CC. This approach can be called hardware address generation, in which the number of outputs in the CC block is equal to R_2 (model U_1 is the only exception). In order to reduce this number, some additional block for address generation is needed (for transformation of object codes). The second approach leads to increasing of the CMCU cycle time, in comparison with its value for CMCU U_1 . In case of the CMCU with elementary OLC and code sharing, the number of CC outputs is smaller than R_2 , but application of these methods can cause either significant increase of the control memory size, in comparison with its minimal value V_{\min} , or an increase of the CMCU cycle time. If the increase of time cycle is not desirable, the number of CC outputs cannot be reduced, in comparison with R_2 . Let us consider how the number of CC outputs can be reduced in cases when application of code sharing leads to introduction of the address transformer AT, but performance of the resulting CMCU cannot be worse,

than in case of the CMCU U_1 . Let us discuss these methods using an example of CMCU U_2 . Our discussion is based on results from [3–5].

In case of CMCU U_2 , the output addresses of OLC $\alpha_g \in C$ are determined by procedure P_2 and, hence, they possess the property of randomness. Application of this procedure does not guarantee that, for example, some bit is equal to zero for all input addresses. Situation of this kind would allow to reduce the number of CC block outputs in comparison with R_2 . Let the set of OLC inputs $I(\Gamma)$ for GSA Γ include I_0 elements, which can be encoded by only R_{21} bits, where

$$R_{21} = \lceil \log_2 I_0 \rceil. \tag{8.1}$$

Obviously the following condition is satisfied for the linear graph-schemes of algorithm, where the number of operator vertices exceeds significantly the number of conditional vertices:

$$R_{21} < R_2. \tag{8.2}$$

Let the following condition (8.3) be satisfied for GSA Γ :

$$\lceil \log_2(I_0 + M_2) \rceil = \lceil \log_2 M_2 \rceil, \tag{8.3}$$

where M_2 is the number of operator vertices. Let us choose I_0 cells of control memory to keep OLC inputs and let these cells have addresses from 0 to $(I_0 - 1)_2$. Let us call this set of cells a dedicated input area (DIA). This fixation of OLC inputs requires execution of unconditional jumps to the real input address, which should be introduced into the special control microinstruction. It leads to some modification of microinstruction formats in comparison with CMCU U_2 [2]. The model of CMCU U_{58} with dedicated input area is shown in Fig. 8.1.

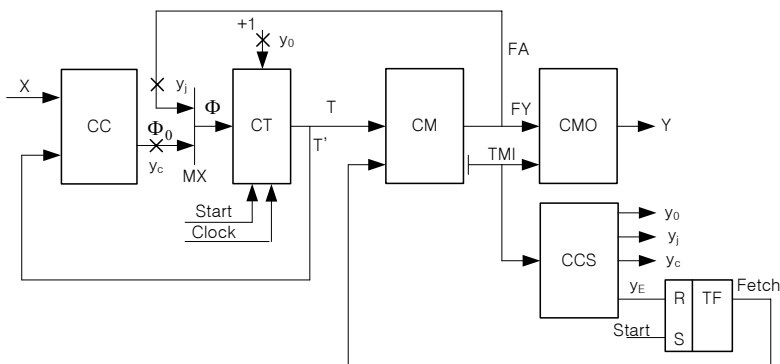


Fig. 8.1 Structural diagram of CMCU U_{58}

Let us discuss particular qualities of CMCU U_{58} in comparison with U_2 . In case of the CMCU U_{58} , there are two formats of microinstructions (Fig. 8.2).

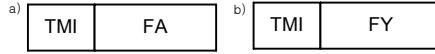


Fig. 8.2 Microinstruction formats for CMCU U_{58}

The control microinstruction, shown in Fig. 8.2a, contains an address field FA with address for transition from the dedicated input area into the area of microprogram (AMP) containing operational microinstructions. This format includes a field of attribute TMI, with all zeroes (TMI=00). Operational microinstruction (Fig. 8.2b) includes the field TMI and an operational part FY. If TMI=01, this microinstruction corresponds to an OLC output, corresponding in turn to $y_c = 1$. If TMI=10, the microinstruction corresponds to some OLC component, which is not an OLC output. It corresponds to $y_0 = 1$. Code TMI=11 indicates that some OLC output connected with vertex b_E is reached. It corresponds to $y_E = 1$. Let us point out that the control microinstruction corresponds to $y_j = 1$.

In case of the control microinstructions, some additional block for generation of microoperations should be used to prevent generation of microoperations $y_n \in Y$ (if $y_j = 1$), because in this case microinstruction would contain information about address of transition only. Multiplexer MX should be used to load into counter CT: either the transition address created by functions Φ_0 ($y_c = 1$), or the address of some cell of the microprogram area, which occupies the field FA of the control microinstruction ($y_j = 1$). Block CCS is used to generate control signals y_0, y_j, y_c, y_E , depending on the content of field TMI.

Compositional microprogram control unit U_{58} operates in the following manner. First, zero code is loaded into the counter CT using pulse "Start", corresponding to the address of main input of OLC $\alpha_1 \in C$, kept in the dedicated input area. At the same time, flip-flop TF is set up and allows microinstruction fetching from the CM control memory (Fetch=1). Current microinstruction is read from the control memory CM and block CCS generates some control signals y_0, y_j, y_c, y_E . If CT contains the address of OLC output, variable $y_c = 1$ is generated together with microoperations $y_n \in Y$. In this case, input memory functions

$$\Phi_0 = \Phi_0(T, X) \quad (8.4)$$

load the address taken from dedicated input area into the counter CT. The signal y_j is generated and an address from AMP is loaded into CT. If the counter CT contains an address of OLC component corresponding to vertex b_q , such that $\langle b_q, b_E \rangle \notin E$ and $b_q \neq O_g$, both microoperations $y_n \in Y(b_q)$ and variable $y_0 = 1$ are generated. In consequence, the counter content is incremented and causes transition to the following microinstruction. If the counter CT contains the address of microinstruction corresponding to vertex b_q , such that $\langle b_q, b_E \rangle \in E$, variable y_E is generated and fetching of microinstructions terminated.

The method of CMCU U_{58} synthesis includes the following steps:

1. Transformation of initial GSA Γ (procedure P_3).
2. Construction of the OLC set using transformed GSA $\Gamma(U_{58})$.

3. Finding addresses for OLC inputs.
4. Microinstruction addressing.
5. Construction of the control memory content.
6. Construction of the transition table of CMCU.
7. Construction of CCS table.
8. Synthesis of CMCU logic circuit using given logical elements.

Let us discuss application of this method for synthesis of the CMCU $U_{58}(I_{20})$, where the transformed GSA $\Gamma_{20}(U_{58})$ is shown in Fig. 8.3.

Application of procedure P_1 to the transformed GSA $\Gamma_{20}(U_{58})$ gives the set $C = \{\alpha_1, \dots, \alpha_6\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1^1 = b_1$, $O_1 = b_2$; $\alpha_2 = \langle b_3, b_4, b_5 \rangle$, $I_2^1 = b_3$, $I_2^2 = O_2 = b_5$; $\alpha_3 = \langle b_6, \dots, b_9 \rangle$, $I_3^1 = b_6$, $I_3^2 = b_8$, $O_3 = b_9$; $\alpha_4 = \langle b_{10}, b_{11} \rangle$, $I_4^1 = b_{10}$, $O_4 = b_{11}$; $\alpha_5 = \langle b_{12}, b_{13} \rangle$, $I_5^1 = b_{12}$, $O_5 = b_{13}$; $\alpha_6 = \langle b_{14}, \dots, b_{17} \rangle$, $I_6^1 = b_{14}$, $O_6 = b_{17}$. Thus, we get the set of inputs $I(I_{20}) = \{b_1, b_3, b_5, b_6, b_8, b_{10}, b_{12}, b_{14}\}$, and the following values can be found: $M_2 = 17$, $R_2 = 15$, $I_0 = 8$, $R_5 = 3$. It means that condition (8.2) holds and application of the method proposed above makes sense. Moreover, because $M_2 + I_0 = 25$, condition (8.3) is satisfied and this method allows to have smaller number of CC inputs, without increasing the length of microinstruction address, in comparison with CMCU $U_2(I_{20})$.

Addressing of OLC inputs is executed in a trivial way, but the address of input I_1^1 should be equal to zero. Let $IA(b_q)$ be the address of input corresponding to vertex $b_q \in B_2$. In case of CMCU $U_{58}(I_{20})$ these addresses are: $IA(b_1) = 000$, $IA(b_3) = 001, \dots, IA(b_{14}) = 111$.

Application of procedure P_2 to GSA $\Gamma_{20}(U_{58})$ results in microinstruction addresses shown in Fig. 8.4.

First line of the table from Fig. 8.4 corresponds to the dedicated input area and each cell of this line contains an address $IA(b_q)$. The rest of lines corresponds to the area of microprogram AMP and each cell for this part of lines contains an address $A(b_q)$. For example, input $I_5^1 = b_{12}$ and its address in DIA is determined as $IA(b_{12}) = 00110$, whereas its address in AMP is $A(b_{12}) = 10011$.

Microinstructions to be kept in the control memory are constructed using the following rules:

- any vertex $b_q \in I(\Gamma)$ from DIA corresponds to a control microinstruction of the unconditional jump, where $[FA] = A(b_q)$;
- if vertex $b_q \in D^g$ is not an output of OLC $\alpha_g \in C$, the control memory cell having address $A(b_q)$ should contain operational microinstruction, where $[TMI] = y_0$;
- if vertex $b_q \in D^g$ is connected with final vertex b_E , the control memory cell with address $A(b_q)$ should contain operational microinstruction, where $[TMI] = y_E$.

Let us denote the construction procedure of the control memory content by symbol P_{13} . Application of procedure P_{13} gives the control memory content shown in Table 8.1.

Let us point out that only 16 cells of the control memory of CMCU $U_{58}(I_{20})$ are shown in Table 8.1. Two bits are used to encode variables y_0, y_j, y_c, y_E , namely m_1 and m_2 . The encoding is executed in such a manner that code 00 corresponds to y_j , code 01 to y_0 , code 10 to y_c , and code 11 to y_E . One-hot encoding approach is used

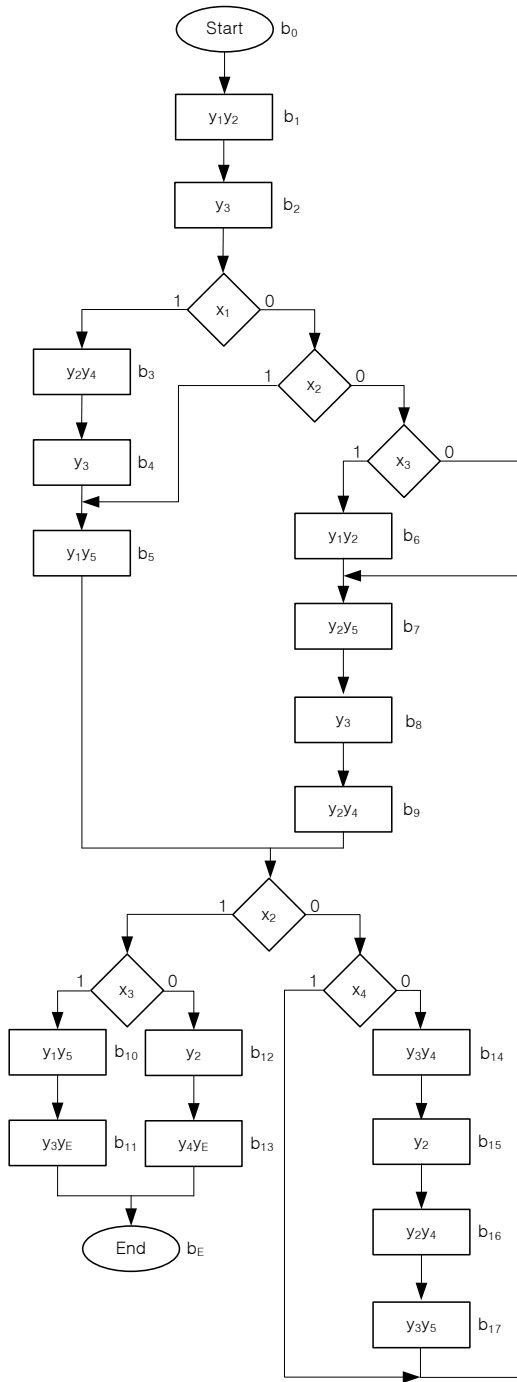


Fig. 8.3 Transformed GSA $I_{20}(U_{58})$

$T_1T_2T_3$		T_4T_5								
		000	001	010	011	100	101	110	111	
00	00	b_1	b_3	b_5	b_6	b_8	b_{10}	b_{12}	b_{14}	DIA
	01	b_1	b_2	b_3	b_4	b_5	b_{18}	b_7	b_8	
11	11	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{19}	b_{15}	b_{16}	AMP
	10	b_{17}	*	*	*	*	*	*	*	

Fig. 8.4 Microinstruction addresses for CMCU $U_{58}(I_{20})$

to encode microoperations, when the bit capacity R_{CM} of the control memory cell is given by the expression

$$R_{CM} = \max(2 + N, 2 + \lceil \log_2(I_0 + M_2) \rceil). \quad (8.5)$$

In this case $R_{CM} = 7$, which means that fields FA and FY are represented by bits $m_3 - m_7$.

Table 8.1 Content of control memory for CMCU $U_{58}(I_{20})$

Address $T_1T_2T_3T_4T_5$	TMI m_1m_2	Content $m_3m_4m_5m_6m_7$	Reference
00000	00	01000	$b_1 \rightarrow A(b_1)$ DIA
00001	00	01010	$b_3 \rightarrow A(b_3)$
00010	00	01100	$b_5 \rightarrow A(b_5)$
00011	00	01101	$b_6 \rightarrow A(b_6)$
00100	00	01111	$b_8 \rightarrow A(b_8)$
00101	00	10001	$b_{10} \rightarrow A(b_{10})$
00110	00	10011	$b_{12} \rightarrow A(b_{12})$
00111	00	10101	$b_{14} \rightarrow A(b_{14})$
01000	01	11000	$b_1 \rightarrow b_2$ AMP
01001	10	00100	$b_2 \rightarrow O_1$
01010	01	01010	$b_3 \rightarrow b_4$
01011	01	00100	$b_4 \rightarrow b_5$
01100	10	10001	$b_5 \rightarrow O_2$
01101	01	11000	$b_6 \rightarrow b_7$
01110	01	01001	$b_7 \rightarrow b_8$
01111	01	00100	$b_8 \rightarrow b_9$

The transition table of CMCU is constructed using the system of transition formulae for outputs of OLC $\alpha_g \in C^1$. In the discussed case we have $C^1 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_6\}$ and the following transition formulae:

$$\begin{aligned}
 O_1 &\rightarrow x_1I_2^1 \vee \bar{x}_1x_2I_2^2 \vee \bar{x}_1\bar{x}_2x_3I_3^1 \vee \bar{x}_1\bar{x}_2\bar{x}_3I_3^2; \\
 O_2, O_3 &\rightarrow x_2x_3I_4^1 \vee x_2\bar{x}_3I_5^1 \vee \bar{x}_2x_4I_3^1 \vee \bar{x}_2\bar{x}_4I_6^1; \\
 O_6 &\rightarrow I_3^2.
 \end{aligned} \quad (8.6)$$

Transition table of the CMCU $U_{58}(I_{20})$ corresponds to system (8.6) and includes $H_{58}(I_{20}) = 13$ lines (Table 8.2). This table is used to obtain the input memory functions for the flip-flops of counter CT (8.4), as for example:

$$D_3^1 = F_4 \vee F_5 \vee F_6 \vee F_8 \vee F_9 \vee F_{10} \vee F_{12} \vee F_{13} = \bar{T}_1 T_2 \bar{T}_3 T_5 \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \dots \vee T_1 T_2 \bar{T}_3 \bar{T}_5.$$

The superscript "1" of function D_3 reflects the fact that D_3 belongs to the set Φ_0 . If this superscript is omitted, we obtain $D_3 \in \Phi$. It can be found from this formula that the address bit $T_4 = 0$ for all outputs of OLC, and, therefore, corresponding variable is absent in system (8.4).

The table for block CCS is constructed in a trivial way and in our particular case it is replaced by the Karnaugh map (Fig. 8.5).

		m_2	
		0	1
m_1	0	y_j	y_o
	1	y_c	y_E

Fig. 8.5 Codes of control variables

Obviously, variables y_0, y_j, y_c, y_E are generated by a decoder with m_1 and m_2 inputs.

Table 8.2 Transition table for CMCU $U_{58}(I_{20})$

O_g	$A(O_g)$	I_m^i	$A(I_m^i)$	X_h	Φ_h	h
O_1	01001	I_2^1	00001	x_1	D_5^1	1
		I_2^2	00010	$\bar{x}_1 x_2$	D_4^1	2
		I_3^1	00011	$\bar{x}_1 \bar{x}_2 x_3$	$D_4^1 D_5^1$	3
		I_3^2	00100	$\bar{x}_1 \bar{x}_2 \bar{x}_3$	D_3^1	4
O_2	01100	I_4^1	00101	$x_2 x_3$	$D_3^1 D_5^1$	5
		I_5^1	00110	$x_2 \bar{x}_3$	$D_3^1 D_4^1$	6
		I_3^1	00011	$\bar{x}_2 x_4$	$D_4^1 D_5^1$	7
		I_6^1	00111	$\bar{x}_2 \bar{x}_4$	$D_3^1 D_4^1 D_5^1$	8
O_3	10000	I_4^1	00101	$x_2 x_3$	$D_3^1 D_5^1$	9
		I_5^1	00110	$x_2 \bar{x}_3$	$D_3^1 D_4^1$	10
		I_3^1	00011	$\bar{x}_2 x_4$	$D_4^1 D_5^1$	11
		I_6^1	00101	$\bar{x}_2 \bar{x}_4$	$D_3^1 D_5^1$	12
O_6	11000	I_3^2	00100	1	D_3^1	13

Logic circuit of CMCU $U_{58}(I_{20})$ is shown in Fig. 8.6. Here, the two-level block AND-OR implements multiplexer MX, outputs of which correspond to the input memory functions of counter CT. The multiplexer is described by the following equations:

$$\begin{aligned}
 D_1 &= y_j \cdot m_3 \vee 0 \cdot y_c; \\
 D_2 &= y_j \cdot m_4 \vee 0 \cdot y_c; \\
 D_3 &= y_j \cdot m_5 \vee D_3^1 \cdot y_c; \\
 D_4 &= y_j \cdot m_6 \vee D_4^1 \cdot y_c; \\
 D_5 &= y_j \cdot m_7 \vee D_5^1 \cdot y_c.
 \end{aligned}
 \tag{8.7}$$

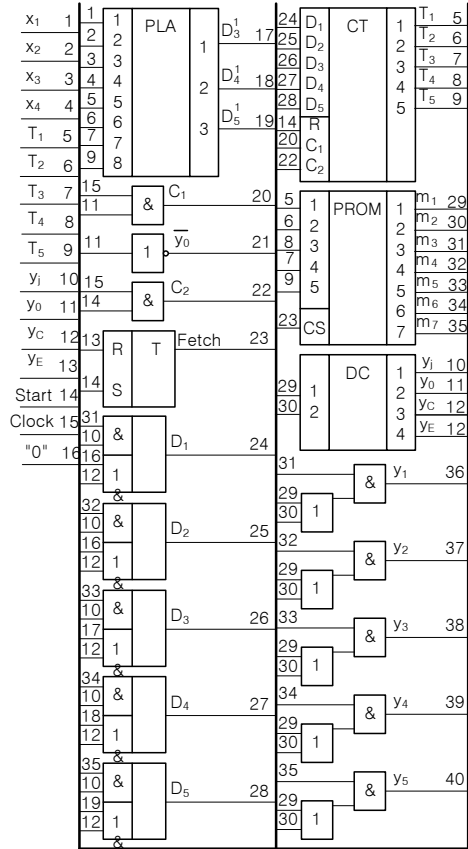


Fig. 8.6 Logic circuit for CMCU $U_{58}(I_{20})$

It is clear, that system (8.7) can be implemented using FPGA, but the logic circuit shown in Fig. 8.6 reflects main principles of CMCU organization only, without its implementation using modern FPLDs.

The system of microoperations is implemented in the following way. It can be seen from the Karnaugh map (Fig. 8.5), that the operational microinstruction is determined by disjunction $m_1 \vee m_2$. Thus, for example, microoperation y_1 is generated if $m_3 = 1$ and $m_1 \vee m_2 = 1$. This analysis leads to the following system:

$$\begin{aligned}
 y_1 &= m_3(m_1 \vee m_2); \\
 y_2 &= m_4(m_1 \vee m_2); \\
 y_3 &= m_5(m_1 \vee m_2); \\
 y_4 &= m_6(m_1 \vee m_2); \\
 y_5 &= m_7(m_1 \vee m_2).
 \end{aligned}
 \tag{8.8}$$

System (8.8) is implemented by the circuit of Fig. 8.6 using AND and OR gates; but can be also implemented with FPGA.

This approach can be applied to obtain some modifications of the CMCU $U_3 - U_6$ models, which are briefly discussed below.

Allocation of the dedicated input area transforms CMCU U_3 into CMCU U_{59} , structural diagram of which is the same as the structural diagram of CMCU U_{58} , but inputs of the block CC of CMCU U_{59} are connected with address variables $T' \subseteq T$. The outcome of special microinstruction addressing for CMCU $U_{59}(I_{20})$ is shown in Fig. 8.7.

$T_1T_2T_3$ T_4T_5		$T_1T_2T_3$							
		000	001	010	011	100	101	110	111
00	00	b_1	b_8	b_1	b_5	b_9	b_{17}	b_{13}	*
	01	b_3	b_{10}	b_2	b_6	b_{14}	b_{10}	*	*
	11	b_5	b_{12}	b_3	b_7	b_{15}	b_{11}	*	*
	10	b_6	b_{14}	b_4	b_8	b_{16}	b_{12}	*	*

Fig. 8.7 Microinstruction addresses for CMCU $U_{59}(I_{20})$

In this particular case, output O_1 is determined unambiguously by the generalized interval of a Boolean space $010**$, output O_2 by $011**$, output O_3 by $100**$, and output O_6 by $101**$. Therefore, inputs of the block CC for the CMCU $U_{59}(I_{20})$ are connected with the variables from set $T' = \{T_1, T_2, T_3\}$. It means that the number of CC inputs is smaller, than in case of CMCU $U_{58}(I_{20})$.

Transformation of the table of Fig. 8.7 into the table shown in Fig. 8.8 results in reduction of the number of address variables connected with CC to only two bits.

$T_1T_2T_3$ T_4T_5		$T_1T_2T_3$							
		000	001	010	011	100	101	110	111
00	00	b_1	b_8	b_{10}	b_{12}	b_1	b_3	b_6	b_{14}
	01	b_3	b_{10}	b_{11}	b_{13}	b_2	b_4	b_7	b_{15}
	11	b_5	b_{12}	*	*	*	b_5	b_8	b_{16}
	10	b_6	b_{14}	*	*	*	*	b_9	b_{17}

Fig. 8.8 New microinstruction addresses for $U_{59}(I_{20})$

Analysis of Fig. 8.8 shows that output O_1 is unambiguously determined by the generalized Boolean interval 100^{**} , output O_2 by interval 101^{**} , output O_3 by interval 110^{**} , and output O_6 by interval 111^{**} . We have $T_1 = 1$ for all outputs of OLC $\alpha_g \in C^1$, and therefore only variables $T_2, T_3 \in T'$ should be connected with the inputs of CC.

Allocation of the dedicated input area transforms U_4 into CMCU U_{60} , with the same structural diagram as in case of CMCU U_{58} , but the application of optimal encoding of OLC $\alpha_g \in C^1$ components allows to reduce the number of terms in (8.4). The outcome of optimal encoding (more correctly, optimal microinstruction addressing) for CMCU $U_{60}(I_{20})$ is shown in Fig. 8.9.

$T_1T_2T_3$	000	001	010	011	100	101	110	111
T_4T_5								
00	b_1	b_8	b_{10}	b_{12}	b_1	b_{14}	b_3	b_6
01	b_3	b_{10}	b_{11}	b_{13}	b_2	b_{15}	b_4	b_7
11	b_5	b_{12}	*	*	*	b_{16}	b_5	b_8
10	b_6	b_{14}	*	*	*	b_{17}	*	b_9

Fig. 8.9 Optimal microinstruction addresses for CMCU $U_{60}(I_{20})$

In the discussed case, partition $\Pi_c = \{B_1, B_2, B_3\}$ can be formed, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3\}$, $B_3 = \{\alpha_6\}$. Analysis of Fig. 8.9 shows that class B_1 is determined by code $K(B_1) = *10^{**}$, class B_2 by code $K(B_2) = *0^{***}$, and class B_3 by code $K(B_3) = *11^{**}$. In this case all address assignments corresponding to the components of OLC $\alpha_g \notin C^1$ are treated as insignificant and are used for optimization of the codes of classes.

Let us transform system (8.6) into the form:

$$\begin{aligned}
 B_1 &\rightarrow x_1 I_2^1 \vee \bar{x}_1 x_2 I_2^2 \vee \bar{x}_1 \bar{x}_2 x_3 I_3^1 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 I_3^2; \\
 B_2 &\rightarrow x_2 x_3 I_4^1 \vee x_2 \bar{x}_3 I_5^1 \vee \bar{x}_2 x_4 I_3^1 \vee \bar{x}_2 \bar{x}_4 I_6^1; \\
 B_3 &\rightarrow I_3^2.
 \end{aligned} \tag{8.9}$$

System (8.9) corresponds to the transition table of CMCU $U_{60}(I_{20})$ with $H_{60}(I_{20}) = 9$ lines (Table 8.3).

This table is used to construct system (8.4). For example, the following equation can be derived from Table 8.3: $D_3^1 = F_4 \vee F_5 \vee F_6 \vee F_8 \vee F_9 = T_2 \bar{T}_3 \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \dots \vee T_1 T_2$. Comparison of equations for the function D_3^1 of CMCU $U_{58}(I_{20})$ and $U_{60}(I_{20})$ shows that in the second case the number of terms is 1.6 times smaller and the number of literals reduced by two elements.

Allocation of the dedicated input area in case of CMCU U_5 turns it into the CMCU U_{61} , with the structural diagram of Fig. 8.10. In this case block CC generates functions

$$\Phi_0 = \Phi_0(T, X). \tag{8.10}$$

Table 8.3 Transition table for CMCU $U_{60}(I_{20})$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
B_1	*10**	I_2^1	00001	x_1	D_5^1	1
		I_2^2	00010	\bar{x}_1x_2	D_4^1	2
		I_3^1	00011	$\bar{x}_1\bar{x}_2x_3$	$D_4^1D_5^1$	3
		I_3^2	00100	$\bar{x}_1\bar{x}_2\bar{x}_3$	D_3^1	4
B_2	*1***	I_4^1	00101	x_2x_3	$D_3^1D_5^1$	5
		I_5^1	00110	$x_2\bar{x}_3$	$D_3^1D_4^1$	6
		I_3^1	00011	\bar{x}_2x_4	$D_4^1D_5^1$	7
		I_6^1	00111	$\bar{x}_2\bar{x}_4$	$D_3^1D_4^1D_5^1$	8
B_3	*11**	I_3^2	00100	1	D_3^1	9

All other blocks of both CMCU U_5 and U_{58} implement similar functions. Synthesis method used for CMCU U_{61} can be interpreted as a modification of the synthesis method applied for CMCU U_{58} and includes some additional steps such as construction of partition Π_c of the set C^1 , encoding of classes $B_i \in \Pi_c$, and construction of the table for block TC.

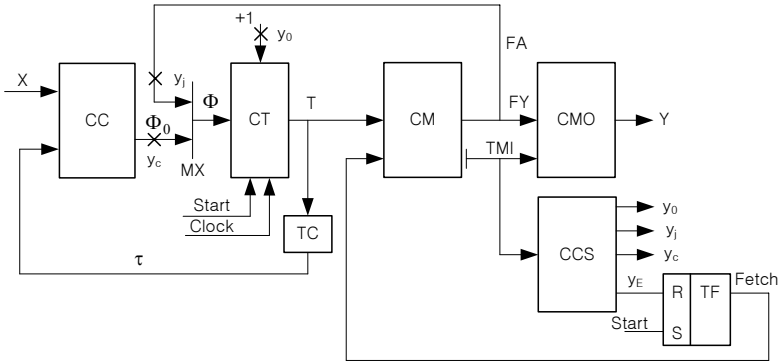


Fig. 8.10 Structural diagram of CMCU U_{61}

Let us consider an example of CMCU $U_{61}(I_{20})$ synthesis. The microinstruction addresses for the CMCU are given in Fig. 8.8. As it was shown earlier, we can get the partition $\Pi_c = \{B_1, B_2, B_3\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3\}$, $B_3 = \{\alpha_6\}$. It is sufficient to have two variables from set $\tau = \{\tau_1, \tau_2\}$ to encode classes $B_i \in \Pi_c$. Let us use the codes: $K(B_1) = 01$, $K(B_2) = 00$, $K(B_3) = 10$. Transition table for CMCU $U_{61}(I_{20})$ is constructed using the system of transition formulae (8.9) and includes $H_{61}(I_{20}) = 9$ lines (Table 8.4).

This table is used to construct equations (8.10). We find, for example, the equation: $D_3^1 = F_4 \vee F_5 \vee F_6 \vee F_7 \vee F_8 \vee F_9 = \bar{\tau}_1 \tau_2 \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{\tau}_1 \bar{\tau}_2 x_2 x_3 \vee \dots \vee \tau_1 \tau_2$.

The corresponding table of code transformer TC is constructed in a traditional way and shown in Table 8.5. This table is used to construct functions $\tau = \tau(T)$, which in our case have the form: $\tau_1 = T_1 T_2 T_3$, $\tau_2 = T_1 \bar{T}_2 \bar{T}_3$. They are used to the synthesis of CMCU $U_{61}(I_{20})$ logic circuit, which is executed as in case of CMCU $U_{58}(I_{20})$.

Allocation of the dedicated input area turns CMCU U_6 into CMCU U_{62} , having the same structural diagram as CMCU U_{58} .

Table 8.4 Transition table for CMCU $U_{61}(I_{20})$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
B_1	01	I_2^1	00001	x_1	D_5^1	1
		I_2^2	00010	$\bar{x}_1 x_2$	D_4^1	2
		I_3^1	00011	$\bar{x}_1 \bar{x}_2 x_3$	$D_4^1 D_5^1$	3
		I_5^2	00100	$\bar{x}_1 \bar{x}_2 \bar{x}_3$	D_3^1	4
B_2	00	I_4^1	00101	$x_2 x_3$	$D_3^1 D_5^1$	5
		I_5^1	00110	$x_2 \bar{x}_3$	$D_3^1 D_4^1$	6
		I_3^1	00011	$\bar{x}_2 x_4$	$D_4^1 D_5^1$	7
		I_6^1	00111	$\bar{x}_2 \bar{x}_4$	$D_3^1 D_4^1 D_5^1$	8

Table 8.5 Table of code transformer for CMCU $U_{61}(I_{20})$

a_g	$C(O_g)$	B_i	$K(B_i)$	τ_g	g
a_1	100**	B_1	01	τ_2	1
a_2	101**	B_2	00	-	2
a_3	110**	B_2	00	-	3
	111**	B_3	10	τ_1	4

The main disadvantage of this approach is the higher number of algorithm execution cycles, due to the existence of control microinstructions. Besides, some additional chip resources are needed to implement the system of microoperations, even in case of hot-one microoperation encoding. The following method can be used to eliminate this disadvantage.

8.2 Optimization of compositional microprogram control unit with the dedicated input area

Let control microinstruction have the following format (Fig. 8.11). In this case, following actions can be executed during one cycle of CMCU operation: generation of microoperations using the code from field FY and generation of transition address using the code from field FA. Both the format of operational microinstruction and

principle of allocation for the first I_0 cells of the control memory for input addresses of OLC $\alpha_g \in C$ are also used here.

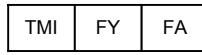


Fig. 8.11 Format of control microinstruction

Application of such control microinstructions transforms the CMCU U_2 into CMCU U_{63} (Fig. 8.12).

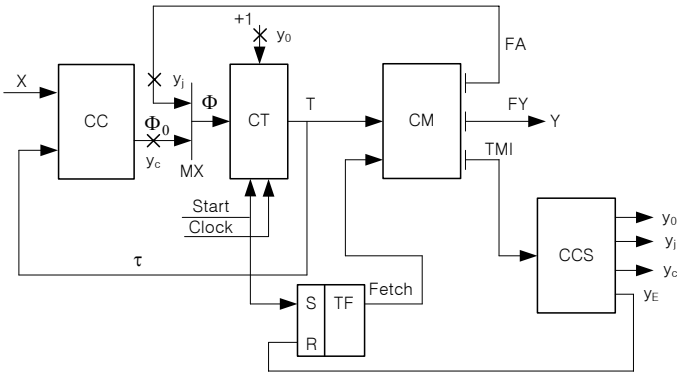


Fig. 8.12 Structural diagram of CMCU U_{63}

Compositional microprogram control unit U_{63} operates as follows. The input address of OLC $\alpha_1 \in C$ is loaded into the counter CT using pulse "Start". At the same time the flip-flop TF is set up. Current microinstruction is fetched from the control memory CM and its field TMI is transformed into the control signals y_0, y_j, y_c, y_E . If signal $y_0 = 1$ is generated simultaneously with microoperations $y_n \in Y$, the content of CT is incremented and corresponds to the transition inside current OLC. If signal $y_j = 1$ is generated, it corresponds to a transition from the dedicated input area DIA into the area of microprogram AMP. In this case, the transition from some input of OLC $\alpha_g \in C$ to next component is executed. If signal $y_c = 1$, it corresponds to the transition from the output of OLC $\alpha_g \in C$ and the content of counter CT is determined by functions Φ_0 . If signal $y_E = 1$, the algorithm execution should be finished. In this case, flip-flop TF is reset and the fetching of microinstructions from control memory is terminated.

Microoperations $y_n \in Y$ are represented by some code in the fixed field FY and therefore the block CMO is absent in case of the hot-one encoding of microoperations (Fig. 8.12). This approach has one serious disadvantage, namely the field FA is not used by microinstructions from the microprogram area AMP. This disadvantage can be partly eliminated due to the partition of control memory CM into two parts (Fig. 8.13). The part CM_1 includes FA field only and therefore information fetching

is executed using the leftmost address bits from set T_j , where $|T_j| = R_{21}$. Both the operational part of microinstructions and the field TMI are kept in the part CM_2 , which is addressed using the whole address.

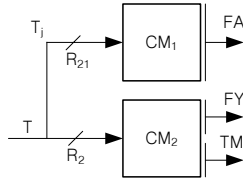


Fig. 8.13 Structural diagram of control memory for CMCU U_{63}

Obviously, fetching of the transition address is executed for all microinstructions, regardless of their type. This address is used only for particular microinstructions, when $y_j = 1$.

The synthesis method used for CMCU U_{63} includes the following steps:

1. Transformation of the initial GSA Γ .
2. Construction of OLC set C for the transformed GSA $\Gamma(U_{63})$.
3. Addressing of inputs for OLC $\alpha_g \in C$.
4. Addressing of microinstructions.
5. Construction of the control memory content for CM_1 .
6. Construction of the control memory content for CM_2 .
7. Construction of the CMCU transition table.
8. Construction of the table for block CCS.
9. Synthesis of CMCU logic circuit for given elements.

Let us discuss the application of this method for synthesis of the CMCU $U_{63}(\Gamma_{20})$, where the transformed GSA $\Gamma_{20}(U_{63})$ is the same as the one shown in Fig. 8.3. Outcomes of the first three synthesis steps are the same for CMCU $U_{58}(\Gamma_{20})$ and $U_{63}(\Gamma_{20})$. Thus, the following set of inputs can be found: $I(\Gamma_{20}) = \{b_1, b_3, b_5, b_6, b_8, b_{10}, b_{12}, b_{14}\}$. In our case the inputs have the addresses: $IA(b_1) = 000, \dots, IA(b_{14}) = 111$.

Microinstruction addressing is executed as follows. First, all main inputs are removed from OLC $\alpha_g \in C$, as the first stage of addressing. Standard addressing procedure is then applied to the transformed OLC $\alpha_g \in C$, as the second stage of addressing. This is the same procedure as the one used in case of the CMCU U_1 .

In this example, removing the main inputs results in the OLC set $C = \{\alpha_1, \dots, \alpha_6\}$, where $\alpha_1 = \langle b_2 \rangle$, $\alpha_2 = \langle b_4, b_5 \rangle$, $\alpha_3 = \langle b_7, b_8, b_9 \rangle$, $\alpha_4 = \langle b_{11} \rangle$, $\alpha_5 = \langle b_{13} \rangle$, $\alpha_6 = \langle b_{15}, b_{16}, b_{17} \rangle$. Addressing the microprogram area AMP starts from the address, which exceeds by 1 the last address taken from the dedicated input area DIA. Resulting microinstruction addresses of the CMCU $U_{63}(\Gamma_{20})$ are shown in Fig. 8.14.

The control memory content of DIA area can be found using the following rules:

$T_1T_2T_3$		000	001	010	011	100	101	110	111
		T_4T_5							
00	b ₁	b ₈	b ₂	b ₈	b ₁₅	*	*	*	
	b ₃	b ₁₀	b ₄	b ₉	b ₁₆	*	*	*	
	b ₅	b ₁₂	b ₅	b ₁₁	b ₁₇	*	*	*	
	b ₆	b ₁₄	b ₇	b ₁₃	*	*	*	*	
DIA					AMP				

Fig. 8.14 Microinstruction addresses for CMCU U_{63}

- if vertex $b_q \neq O_g$ ($g = 1, \dots, G$), field TMI of the memory cell with address $IA(b_q)$ contains code of variable y_j , its field FY contains microoperations $y_n \in Y(b_q)$, and its field FA contains address $A(b_i)$, where $\langle b_q, b_i \rangle \in E$;
- if vertex $b_q = O_g$ ($g = 1, \dots, G$) and $\langle b_q, b_E \rangle \notin E$, field TMI of the memory cell with address $IA(b_q)$ contains code of variable y_j , its field FY contains microoperations $y_n \in Y(b_q)$, and its field FA contains the transition address;
- if vertex b_q is connected with the final vertex b_E , field TMI of the memory cell with address $IA(b_q)$ contains code of variable y_E , its field FY contains microoperations $y_n \in Y(b_q)$, and its field FA is empty.

In our example, content of the control memory CM_1 is shown in Table 8.5. In this case $T_j = \{T_3, T_4, T_5\}$.

Table 8.6 Content of the control memory CM_1 for CMCU $U_{63}(T_{20})$

Address $T_3T_4T_5$	Content $a_1a_2a_3a_4a_5$	Comment
000	01000	$b_1 \rightarrow A(b_2)$
001	01001	$b_3 \rightarrow A(b_4)$
010	01010	$b_5 \rightarrow A(b_5)$
011	01011	$b_6 \rightarrow A(b_7)$
100	01101	$b_8 \rightarrow A(b_9)$
101	01110	$b_{10} \rightarrow A(b_{11})$
110	01111	$b_{12} \rightarrow A(b_{13})$
111	10000	$b_{14} \rightarrow A(b_{15})$

In this table, bits $a_1 - a_5$ represent the transition address and form the field FA.

Construction of the control memory content for AMP area is executed in the same way as in case of CMCU U_{58} . Content of the control memory CM_2 includes both areas DIA and AMP; it is shown in Table 8.6.

As in the previous case, bits m_1, m_2 represent TMI codes, where code 00 corresponds to y_j , code 01 to y_0 , code 10 to y_c , and code 11 to y_E . Bits $m_3 - m_7$ contain hot-one code of the collection of microoperations $y_n \in Y(b_q)$, where $q = 1, \dots, M_2$.

Let us point out that for the vertex b_5 from AMP area, the field $FY = \emptyset$. It corresponds to an idle cycle of the data-path.

Table 8.7 Content of control memory CM_2 for CMCU $U_{63}(I_{20})$

Address $T_1T_2T_3T_4T_5$	Content $m_1m_2m_3m_4m_5m_6m_7$	Comment
00000	0011000	$b_1I_1^1$ DIA
00001	0001010	$b_3I_2^1$
00010	0010001	$b_5I_2^2$
00011	0011000	$b_6I_3^1$
00100	0000100	$b_8I_3^2$
00101	0010001	$b_{10}I_4^1$
00110	0001000	$b_{12}I_5^1$
00111	0000110	$b_{14}I_6^1$
01000	1000100	b_1O_1 AMP
01001	0100100	$b_4 \rightarrow b_5$
01010	1000000	b_5O_2
01011	0101001	$b_7 \rightarrow b_8$
01100	0100100	$b_8 \rightarrow b_9$
01101	1001010	b_9O_3
01110	1100100	$b_{11} \rightarrow End$
01111	1100010	$b_{13} \rightarrow End$
10000	0101000	$b_{15} \rightarrow b_{16}$
10001	0101010	$b_{16} \rightarrow b_{17}$
10010	1000101	$b_{17}O_6$

Transition table for CMCU U_{63} is constructed by analogy with the construction of the corresponding table for CMCU U_{58} . In the discussed case it is the same as in case of CMCU $U_{58}(I_{20})$, represented by Table 8.2. Equations for functions (8.4) for both CMCUs are the same, but the system of formulae for multiplexer MX (8.7) is transformed due to presence of the block CM_1 . In our case this system is transformed into the form:

$$\begin{aligned}
 D_1 &= y_j \cdot a_1 \vee 0 \cdot y_c; \\
 D_2 &= y_j \cdot a_2 \vee 0 \cdot y_c; \\
 D_3 &= y_j \cdot a_3 \vee D_3^1 \cdot y_c; \\
 D_4 &= y_j \cdot a_4 \vee D_4^1 \cdot y_c; \\
 D_5 &= y_j \cdot a_5 \vee D_5^1 \cdot y,
 \end{aligned} \tag{8.11}$$

where variables $a_1 - a_5$ represent the bits of FA field.

Logic circuit of CMCU $U_{63}(I_{20})$ is shown in Fig. 8.15. In this case OLC $\alpha_g \in C^1$ have the following output addresses: $A(O_1) = 01000$, $A(O_2) = 01010$, $A(O_3) = 01101$, $A(O_6) = 10010$. Because each address bit has both direct and complementary values, the inputs of block CC are connected to all $R_2 = 5$ feedback variables. Multiplexer MX is shown here as a block replacing the set of logic elements "AND-OR" from Fig. 8.6. The control memory of CMCU is divided into two blocks (CM_1 and CM_2), contents of which are determined by corresponding tables.

Application of this approach transforms the CMCU U_3 into CMCU U_{64} (special microinstruction addressing and allocation of the dedicated input area), CMCU U_4 into CMCU U_{65} (optimal microinstruction addressing and allocation of dedicated input area), and CMCU U_6 into CMCU U_{67} (transformation of the initial GSA and allocation of the dedicated input area). The structural diagrams for these CMCUs are the same as for CMCU U_{63} , and their synthesis methods are some extensions of methods used for the basic models of CMCU, obtained by adding the steps in which construction of tables for blocks CM_1 and CM_2 is performed.

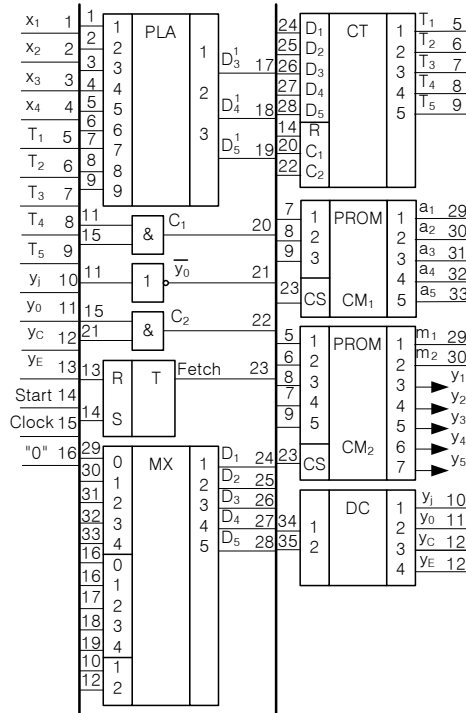


Fig. 8.15 Logic circuit of CMCU $U_{63}(I_{20})$

Use of the control microinstructions having format from Fig. 8.11 transforms CMCU U_5 into CMCU U_{66} with the structural diagram shown in Fig. 8.16. All blocks of CMCU U_{66} play the same roles as the blocks of corresponding basic models of CMCU U_5 and CMCU U_{63} . This synthesis method combines the methods used earlier for both U_5 and U_{63} .

Using the control microinstruction format of Fig. 8.11, instead of the format given in Fig. 8.2, allows to reduce the number of microinstructions in the control memory from I_0 to IO_0 , where IO_0 is the number of OLC inputs, which are also the outputs of the same OLC. It means that condition (8.3) is transformed into the following one:

$$\lceil \log_2(IO_0 + M_2) \rceil = \lceil \log_2 M_2 \rceil. \tag{8.12}$$

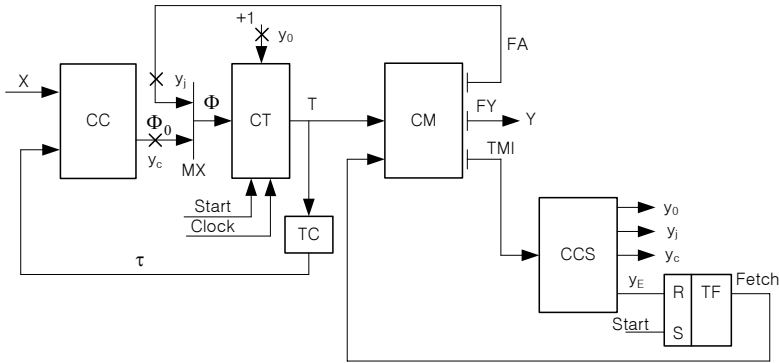


Fig. 8.16 Structural diagram of CMCU U_{66}

Because $IO_0 < I_0$, the probability of satisfying condition (8.12) and hardware amount of CMCU logic circuit can be reduced, due to allocation of the dedicated input area DIA.

The number of microinstructions in the control memory can be reduced to M_2 , if OLC $\alpha_g \in C$ are replaced by elementary OLC $\alpha_g \in C_E$. Application of this approach transforms the CMCU $U_2 - U_7$ into control units $U_{68} - U_{73}$ respectively. The structural diagrams of these new CMCUs are the same as the diagrams of CMCU $U_{62} - U_{67}$ respectively. Synthesis methods applied for CMCU $U_{68} - U_{73}$ differ from corresponding methods used for their basic models only in construction of the EOLC set C_E instead of OLC set. Let us discuss a synthesis example for CMCU $U_{72}(I_{20})$, with the same structural diagram which is shown in Fig. 8.16.

Let us find an EOLC set $C_E = \{\alpha_1, \dots, \alpha_8\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $\alpha_2 = \langle b_3, b_4 \rangle$, $\alpha_3 = \langle b_5 \rangle$, $\alpha_4 = \langle b_6, b_7 \rangle$, $\alpha_5 = \langle b_8, b_9 \rangle$, $\alpha_6 = \langle b_{10}, b_{11} \rangle$, $\alpha_7 = \langle b_{12}, b_{13} \rangle$, $\alpha_8 = \langle b_{14}, \dots, b_{17} \rangle$. In all EOLC $\alpha_g \in C_E$, the first component is an input I_g and last component is an output O_g of the corresponding chain.

Inputs of EOLC $\alpha_g \in C_E$ correspond to the control microinstruction format shown in Fig. 8.11, where the microinstructions are placed in DIA. In this case it is sufficient to use $R_{15} = 3$ variables to address the inputs, and we have the set $\Phi_0 = \{D_3^1, D_4^1, D_5^1\}$. Let us address the microinstructions, as shown in Fig. 8.17.

$T_1 T_2 T_3$		$T_4 T_5$							
		000	001	010	011	100	101	110	111
$T_4 T_5$	00	b_1	b_8	b_2	b_{11}	b_{17}	*	*	*
	01	b_3	b_{10}	b_4	b_{13}	*	*	*	*
	11	b_5	b_{12}	b_7	b_{15}	*	*	*	*
	10	b_6	b_{14}	b_9	b_{16}	*	*	*	*

Fig. 8.17 Microinstruction addresses for CMCU $U_{72}(I_{20})$

For this particular case, the content of control memory CM_1 is represented by Table 8.7, and $T_j = \{T_3, T_4, T_5\}$.

The column FA for address 010 contains all insignificant values, because vertex b_5 is an input of EOLC α_3 and the field FA is ignored. For this field we have $a_1 = 0$, that is the field FA should include only the address bits $a_2 - a_5$. Reduction of the number of address bits allows to reduce the hardware amount of two blocks, CM_1 and MX .

Table 8.8 Content of the control memory CM_1 for CMCU $U_{72}(T_{20})$

Address $T_3T_4T_5$	Content $a_1a_2a_3a_4a_5$	Comment
000	01000	$b_1 \rightarrow A(b_2)$
001	01001	$b_3 \rightarrow A(b_4)$
010	*****	$b_5 = O_3$
011	01010	$b_6 \rightarrow A(b_7)$
100	01011	$b_8 \rightarrow A(b_9)$
101	01100	$b_{10} \rightarrow A(b_{11})$
110	01101	$b_{12} \rightarrow A(b_{13})$
111	01110	$b_{14} \rightarrow A(b_{15})$

Content of the control memory CM_2 for CMCU $U_{72}(T_{20})$ is represented by Table 8.8, having $M_2 = 17$ lines.

Let us find partition Π_E of set C_E^1 into classes of pseudoequivalent EOLC. In our case we get partition $\Pi_E = \{B_1, \dots, B_4\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2\}$, $B_3 = \{\alpha_3, \alpha_5\}$, $B_4 = \{\alpha_4, \alpha_8\}$. Now we construct the system of transition formulae for classes $B_i \in \Pi_E$, which in our case has the form:

$$\begin{aligned}
 B_1 &\rightarrow x_1I_2 \vee \bar{x}_1x_2I_3 \vee \bar{x}_1\bar{x}_2x_3I_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3I_5; \\
 B_2 &\rightarrow I_3; \\
 B_3 &\rightarrow x_2x_3I_6 \vee x_2\bar{x}_3I_7 \vee \bar{x}_2x_4I_5 \vee \bar{x}_2\bar{x}_4I_8; \\
 B_4 &\rightarrow I_5.
 \end{aligned} \tag{8.13}$$

This table is used to derive functions Φ_0 , as for example the expression: $D_3^1 = F_4 \vee F_6 \vee F_7 \vee F_8 \vee F_9 \vee F_{10} = \tau_1 \bar{\tau}_2 \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{\tau}_1 \bar{\tau}_2 \vee \bar{\tau}_1 \tau_2 = \tau_1 \bar{\tau}_2 \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{\tau}_1$ (which is the minimized form of initial expression).

It is necessary to construct the table describing transformation of EOLC codes into codes of classes $B_i \in \Pi_E$, used to design the logic circuit of block TC. In our case this table has 6 lines (Table 8.10).

Analysis of Table 8.10 shows that address bit $a_3 = 0$ and therefore the variable T_3 is excluded from the Boolean equations for functions $\tau_r \in \tau$.

In the synthesis of CMCU $U_{72}(T_{20})$ logic circuit, bits m_1, m_2 represent the code of microinstruction addressing operation. The code 00 determines here the loading of field FA content into the counter CT ($y_j = 1$); code 01 corresponds to a transition within the same EOLC. In this case, content of the counter CT should be

Table 8.9 Content of control memory CM_2 for CMCU $U_{72}(I_{20})$

Address $T_1T_2T_3T_4T_5$	Content $m_1m_2m_3m_4m_5m_6m_7$	Comment
00000	0011000	b_1I_1 DIA
00001	0001010	b_3I_2
00010	1010001	b_5I_3, O_3
00011	0011000	b_6I_4
00100	0000100	b_8I_5
00101	0010001	$b_{10}I_6$
00110	0001000	$b_{12}I_7$
00111	0000110	$b_{14}I_8$
01000	1000100	b_2O_1 AMP
01001	1000100	b_4O_2
01010	1000001	b_7O_4
01011	1001010	b_9O_5
01100	1100100	$b_{11}O_6 \rightarrow End$
01101	1100010	$b_{13}O_7 \rightarrow End$
01110	0101000	$b_{15} \rightarrow b_{16}$
01111	0101010	$b_{16} \rightarrow b_{17}$
10000	1000101	$b_{17}O_8$

Table 8.10 Table of address transformer TC for CMCU $U_{72}(I_{20})$

a_g	$A(O_g)$	B_i	$K(B_i)$	τ_g	g
a_1	01000	B_1	10	τ_1	1
a_2	01001	B_2	11	$\tau_1 \tau_2$	2
a_3	00010	B_3	00	–	3
a_4	01010	B_4	01	τ_2	4
a_5	01011	B_3	00	–	5
a_8	10000	B_4	01	τ_2	6

incremented by 1 ($y_0 = 1$). Code 10 determines loading the EOLC input address using functions Φ_0 , when the output of some EOLC is reached ($y_c = 1$). The code 11 corresponds to the end of CMCU operation, because a particular microprogram is finished ($y_E = 1$).

In case of the control memory CM_1 we have $a_1 = 0$ (Table 8.7) and therefore system (8.11) should be transformed. Now $D_1 = 0$, but other equations remain the same as in system (8.11).

Logic circuit of CMCU $U_{72}(I_{20})$ is shown in Fig. 8.18. As in previous case, the blocks CC, DC and TC are implemented using PLA chips, but PAL macrocells or LUT-elements of FPGA can be also used. Logic circuit for multiplexer MX can be implemented using FPLD elements.

Comparative analysis shows that the use of EOLC instead of OLC allows to reduce the control memory size, but the number of lines in the CMCU transition table becomes higher, making the hardware amount, necessary for implementation of block CC, larger.

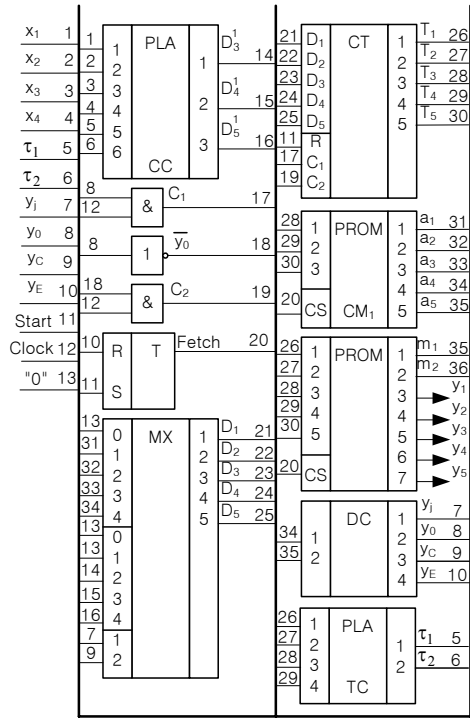


Fig. 8.18 Logic circuit for CMCU $U_{72}(I_{20})$

Further decrease of hardware amount in case of the CMCU with allocation of dedicated input area is possible due to replacement of logical conditions and/or encoding collections of microoperations. Application of these methods makes the control unit cycle time longer and, in consequence, worse resulting digital system performance. These optimization methods are discussed in details below.

Modifications of the microinstruction formats discussed in Sections 8.1 – 8.2 allows to reduce the number of CC outputs, without application of an additional transformer. Obviously, multiplexer MX introduces some additional delay, thus increasing the total cycle time. This is a standard block, however, and its synthesis is much simpler than synthesis of special circuits used as object code transformers.

8.3 Minimization of the number of feedback signals in CMCU

All models of CMCU discussed earlier have the same peculiarity, namely the output addresses have random character. For this reason, R_2 feedback variables (address bits) are connected with CC block inputs of CMCU U_2 . The value of R_2 can be reduced using either special or optimal microinstruction addressing, but positive result of such addressing is not always possible. The number of feedback variables can be reduced to $R_4 = \lceil \log_2 I \rceil$, but it involves the use of special address transformer

TC, consuming additional resources of the chip. In this Section we propose how to reduce the number of feedback variables to R_4 , by means of modification of the microinstruction format.

Let the operational microinstruction include the field of format code TMI and field FY with microoperation code (Fig. 8.19a). Let the control microinstruction include fields TMI and $K(B_i)$ with the code of pseudoequivalent OLC class $B_i \in \Pi_C$ (Fig. 8.19b).

Fig. 8.19 Formats of operational **a** and control **b** microinstructions



Field TMI determines the following control signals: y_0 (increment of the counter content); y_E (end of microprogram); y_c (input memory functions for flip-flops of counter CT are generated by block CC on the base of code $K(B_i)$). Thus, variables y_0 and y_E correspond to the operational microinstruction and y_c determines the control microinstruction of microprogram corresponding to the initial graph-scheme of algorithm.

Let each OLC $\alpha_g \in C^1$ be ended using the control microinstruction with code $K(B_i)$, where $\alpha_g \in B_i$. Thus, the number of microinstructions in a particular microprogram becomes increased by $|C^1|$, in comparison with corresponding value found for CMCU U_2 . Obviously, this approach makes sense if the following condition holds:

$$\lceil \log_2(|C^1| + M_2) \rceil = \lceil \log_2 M_2 \rceil. \tag{8.14}$$

Otherwise, the control memory size will be increased, in comparison with its value for CMCU U_2 . The application of modified microinstruction format (Fig. 8.19) leads to CMCU U_{74} (Fig. 8.20).

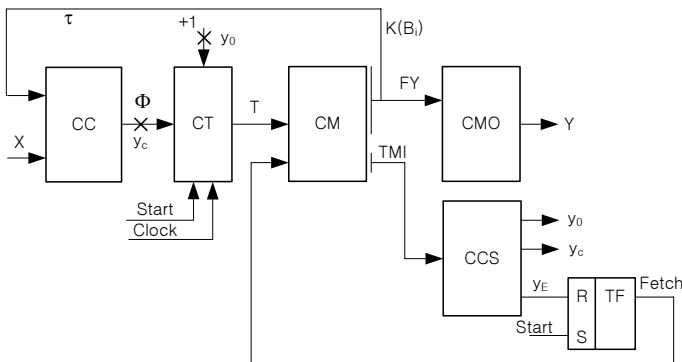


Fig. 8.20 Structural diagram of CMCU U_{74}

In case of CMCU U_{74} , block CC implements the functions

$$\Phi = \Phi(\tau, X), \tag{8.15}$$

where τ is a set of bits from microinstruction field $K(B_i)$. Other blocks of CMCU U_{74} operate in the same manner, as corresponding blocks of other CMCU models with modified microinstruction formats.

Synthesis method used for CMCU U_{74} includes the following steps:

1. Transformation of the initial GSA Γ .
2. Construction of OLC set C for the transformed GSA $\Gamma(U_{74})$.
3. Addressing of microinstructions.
4. Construction of partition Π_C for OLC set C^1 into classes of pseudoequivalent OLC.
5. Coding of the classes $B_i \in \Pi_C$.
6. Construction of the control memory content.
7. Construction of the CMCU transition table.
8. Construction of the table for block CCS.
9. Construction of the table for block CMO.
10. Synthesis of the CMCU logic circuit with given logic elements.

Let us discuss an example of CMCU U_{74} synthesis using GSA Γ_{20} , where the transformed GSA $\Gamma_{20}(U_{74})$ is the same as GSA $\Gamma_{20}(U_{58})$, shown in Fig. 8.3.

Construction of modified OLC set consists on adding some elements, corresponding to OLC output O_g , to the OLC $\alpha_g \in C^1$. These new elements correspond to the control microinstructions. In our example, OLC set $C = \{\alpha_1, \dots, \alpha_6\}$ is formed, where $\alpha_1 = \langle b_1, b_2 \rangle$, $\alpha_2 = \langle b_3, b_4, b_5 \rangle$, $\alpha_3 = \langle b_6, \dots, b_9 \rangle$, $\alpha_4 = \langle b_{10}, b_{11} \rangle$, $\alpha_5 = \langle b_{12}, b_{13} \rangle$, $\alpha_6 = \langle b_{14}, \dots, b_{17} \rangle$. This set includes the subset $C^1 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_6\}$. The modified OLC $\alpha_g \in C^1$ are: $\alpha_1 = \langle b_1, b_2, O_1 \rangle$, $\alpha_2 = \langle b_3, b_4, b_5, O_2 \rangle$, $\alpha_3 = \langle b_6, \dots, b_9, O_3 \rangle$, $\alpha_4 = \langle b_{14}, \dots, b_{17}, O_6 \rangle$.

Using standard addressing procedure for the modified OLC, we get the table shown in Fig. 8.21.

		$T_1T_2T_3$							
		000	001	010	011	100	101	110	111
T_4T_5	00	b_1	b_4	b_7	b_{10}	b_{14}	O_6	*	*
	01	b_3	b_5	b_8	b_{11}	b_{15}	*	*	*
	11	O_1	O_2	b_9	b_{12}	b_{16}	*	*	*
	10	b_3	b_6	O_3	b_{13}	b_{17}	*	*	*

Fig. 8.21 Microinstruction addresses for CMCU $U_{74}(\Gamma_{20})$

The partition $\Pi_c = \{B_1, B_2, B_3\}$ can be found, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3\}$, $B_3 = \{\alpha_6\}$. Variables from set $\tau = \{\tau_1, \tau_2\}$ can be used to encoding classes $B_i \in \Pi_C$. As the CMCU U_{74} does not include any address transformer, coding of classes

$B_i \in \Pi_C$ can be made arbitrarily. Let classes $B_i \in \Pi_C$ have the codes: $K(B_1) = 00$, $K(B_2) = 01$, $K(B_3) = 10$. Including the insignificant input assignment 11, the following final codes can be obtained: $K(B_2) = *1$, $K(B_3) = 1*$.

Length of the control memory word is determined by analogy with (8.5). In our case, the operational microinstruction includes 7 bits, and control microinstruction includes 4 bits. The control memory content of CMCU $U_{74}(I_{20})$ is represented by Table 8.11. Field TMI is represented here by bits m_1 and m_2 , where the code 00 corresponds to y_E , 01 to y_0 , and 11 to y_C . For this encoding, equality $m_1 = 0$ determines an operational microinstruction and $m_2 = 1$ corresponds to a control microinstruction. It allows simplification of the CMO logic circuit responsible for generation of microinstructions. Bits m_3 and m_4 correspond here either to variables τ_1 and τ_2 (for control microinstructions) or to microoperations y_1, y_2 (for operational microinstructions), bits from m_5 to m_7 correspond to microoperations $y_3 - y_5$ respectively and stand for operational microinstructions. If some control memory bit is not used, it is marked by symbol "*" in the table.

Table 8.11 Control memory content for CMCU $U_{74}(I_{20})$

Address $T_1T_2T_3T_4T_5$	TMI m_1m_2	Content $m_3m_4m_5m_6m_7$	Comment
00000	01	11000	$b_1 \rightarrow b_2$
00001	01	00100	
00010	11	00***	O_1
00011	01	01010	$b_3 \rightarrow b_4$
00100	01	00100	$b_4 \rightarrow b_5$
00101	01	10001	$b_5 \rightarrow O_2$
00110	11	*1***	O_2
00111	01	11000	$b_6 \rightarrow b_7$
01000	01	01001	$b_7 \rightarrow b_8$
01001	01	00100	$b_8 \rightarrow b_9$
01010	01	01010	$b_9 \rightarrow O_3$
01011	11	*1***	O_3
01100	01	10001	$b_{10} \rightarrow b_{11}$
01101	00	00100	$b_{11} \rightarrow End$
01110	01	01000	$b_{12} \rightarrow b_{13}$
01111	00	00010	$b_{13} \rightarrow End$
10000	01	00110	$b_{14} \rightarrow b_{15}$
10001	01	01000	$b_{15} \rightarrow b_{16}$
10010	01	01010	$b_{16} \rightarrow b_{17}$
10011	01	00101	$b_{17} \rightarrow O_6$
10100	11	1****	O_6

Transition table for CMCU $U_{74}(I_{20})$ is constructed using the system of transition formulae for classes $B_i \in \Pi_C$. In the discussed case, system (8.9) is used and the output addresses for OLC $\alpha_g \in C$ are taken from Fig. 8.21. The transition table of CMCU $U_{74}(I_{20})$ includes $H_{74}(I_{20}) = 9$ lines (Table 8.12). This table is used to construct system (8.15). The following equation can be derived, for example, from Table 8.12: $D_3^1 = F_2 \vee F_3 \vee F_5 \vee F_6 \vee F_7 = \bar{\tau}_1 \bar{\tau}_2 \bar{x}_1 x_2 \vee \dots \vee \tau_2 \bar{x}_2 x_4$.

Table 8.12 Transition table for CMCU $U_{74}(I_{20})$

B_i	$K(B_i)$	I_m^j	$A(I_m^j)$	X_h	Φ_h	h
B_1	00	I_2^1	00011	x_1	D_4D_5	1
		I_2^2	00101	\bar{x}_1x_2	D_3D_5	2
		I_3^1	00111	$\bar{x}_1\bar{x}_2x_3$	$D_3D_4D_5$	3
		I_3^2	01001	$\bar{x}_1\bar{x}_2\bar{x}_3$	D_2D_5	4
B_2	*1	I_4^1	01100	x_2x_3	D_2D_3	5
		I_5^1	01110	$x_2\bar{x}_3$	$D_2D_3D_4$	6
		I_3^2	00111	\bar{x}_2x_4	$D_3D_4D_5$	7
		I_6^1	10000	$\bar{x}_2\bar{x}_4$	D_1	8
B_3	1*	I_3^2	01001	1	D_2D_5	9

Table for block CCS is constructed in a trivial way; in our example it is replaced by the Karnaugh map shown in Fig. 8.22. Using this map, the following equations can be found: $y_E = \bar{m}_2$, $y_c = m_2$, $y_0 = \bar{m}_1m_2$. Since both y_E and y_0 correspond to the operational microinstruction, it is determined by $m_1 = 0$.

Fig. 8.22 Encoding of variables corresponding to internal control signals

	m_2	0	1
m_1	0	y_E	y_0
	1	*	y_c

The step involving construction of CMO table, determines a system $Y = Y([TMI], [FY])$, where $y_1 = \bar{m}_1m_3$, $y_2 = \bar{m}_1m_4, \dots, y_5 = \bar{m}_1m_7$ (Table 8.13). Logic circuit for CMCU $U_{74}(I_{20})$ is shown in Fig. 8.23.

Table 8.13 Table for block CMO of CMCU $U_{74}(I_{20})$

m_1	y_1	y_2	y_3	y_4	y_5
0	m_3	m_4	m_5	m_6	m_7
1	0	0	0	0	0

Main disadvantage of the proposed approach is higher number of cycles required for control algorithm execution, longer cycle time, and higher number of microinstructions, in comparison with corresponding parameters of CMCU U_2 . As in case of CMCU with the dedicated input area, this disadvantage can be partly eliminated due to use of fields TMI, FY and $K(B_i)$ in a single microinstruction. This approach leads to microinstruction format (Fig. 8.24) used in CMCU U_{75} .

The structural diagram of CMCU U_{75} is shown in Fig. 8.25.

All blocks of CMCU U_{75} execute the same functions as corresponding blocks of CMCU U_{74} . The fields FY and $K(B_i)$ occupy different bits of microinstruction, and

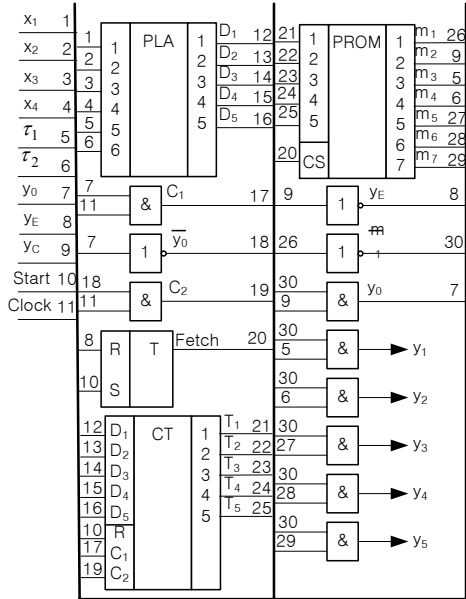


Fig. 8.23 Logic circuit for CMCU $U_{74}(I_{20})$

Fig. 8.24 Microinstruction format for CMCU U_{75}

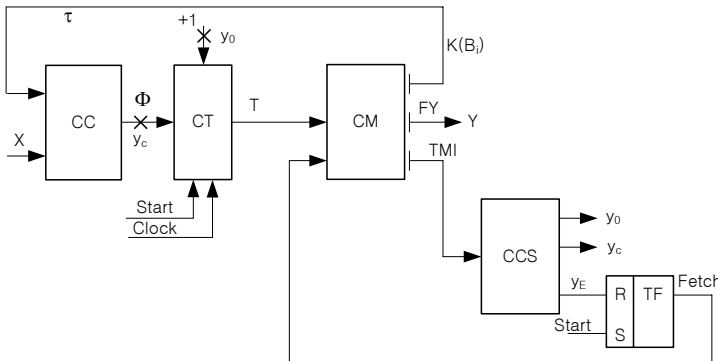


Fig. 8.25 Structural diagram of CMCU U_{75}

therefore block CMO is absent in the case of hot-one encoding of microoperations. It leads to equality of the cycle times for CMCU U_{75} and U_2 . For the same reason, additional control microinstructions corresponding to the outputs of OLC $\alpha_g \in C^1$ are not used. It leads to equality both the number of microinstructions and the algorithm execution time for CMCU U_{75} and U_2 . Of course, these CMCU circuits should interpret the same GSA.

Synthesis method applied for CMCU U_{75} includes steps 1–8 and 10 of the corresponding method used in case of CMCU U_{74} , but traditional OLC (as for CMCU U_2) are generated only after execution of step 2. Let us consider an example of CMCU $U_{75}(I_{20})$ synthesis. Only the results, different from the solutions obtained already for CMCU $U_{74}(I_{20})$, will be discussed below.

Microinstruction addresses for CMCU $U_{75}(I)$ is shown in Fig. 8.26, the control memory content is represented by Table 8.14 and transitions by Table 8.15. The logic circuit of CMCU $U_{75}(I_{20})$ is shown in Fig. 8.27.

$T_1T_2T_3$		000	001	010	011	100	101	110	111
		T_4T_5							
T_4T_5	00	b_1	b_5	b_9	b_{13}	b_{17}	*	*	*
	01	b_3	b_6	b_{10}	b_{14}	*	*	*	*
	11	b_3	b_7	b_{11}	b_{15}	*	*	*	*
	10	b_4	b_8	b_{12}	b_{16}	*	*	*	*

Fig. 8.26 Microinstruction addresses for CMCU $U_{75}(I_{20})$

Table 8.14 Control memory content for CMCU $U_{75}(I_{20})$

Address $T_1T_2T_3T_4T_5$	TMI m_1m_2	FY $m_3m_4m_5m_6m_7$	$K(B_i)$ m_8m_9	Comment
00000	01	11000	**	$b_1 \rightarrow b_2$
00001	11	00100	00	b_2O_1
00010	01	01010	**	$b_3 \rightarrow b_4$
00011	01	00100	**	$b_4 \rightarrow b_5$
00100	11	10001	*1	b_2O_2
00101	01	11000	**	$b_6 \rightarrow b_7$
00110	01	01001	**	$b_7 \rightarrow b_8$
00111	01	00100	**	$b_8 \rightarrow b_9$
01000	11	01010	*1	b_9O_3
01001	01	10001	**	$b_{10} \rightarrow b_{11}$
01010	00	00100	**	$b_{11} \rightarrow End$
01011	01	01000	**	$b_{12} \rightarrow b_{13}$
01100	00	00010	**	$b_{13} \rightarrow End$
01101	01	00110	**	$b_{14} \rightarrow b_{15}$
01110	01	01000	**	$b_{15} \rightarrow b_{16}$
01111	01	01010	**	$b_{16} \rightarrow b_{17}$
10000	11	00101	1*	$b_{17}O_6$

The bits of control memory are marked by "*" in Table 8.14, if they have no definite assignment. Transition tables are practically identical for both CMCU $U_{74}(I_{20})$ and $U_{75}(I_{20})$, but OLC input addresses for these models of CMCU are not the same and therefore corresponding columns $A(I_m^j)$ and Φ_h of their transition tables are different.

Table 8.15 Transition table for CMCU $U_{75}(I_{20})$

Address B_i	TMI $K(B_i)$	FY I_m^j	$K(B_i)$ $A(I_m^j)$	Comment X_h	Φ_h	h
B_1	00	I_2^1	00010	x_1	D_4	1
		I_2^2	00100	\bar{x}_1x_2	D_3	2
		I_3^1	00101	$\bar{x}_1\bar{x}_2x_3$	D_3D_5	3
		I_3^2	00111	$\bar{x}_1\bar{x}_2\bar{x}_3$	$D_3D_4D_5$	4
B_2	*1	I_4^1	01001	x_2x_3	D_2D_3	5
		I_5^1	01011	$x_2\bar{x}_3$	$D_2D_4D_5$	6
		I_3^2	00111	\bar{x}_2x_4	$D_3D_4D_5$	7
		I_6^1	01101	$\bar{x}_2\bar{x}_4$	$D_2D_3D_5$	8
B_3	1*	I_3^2	00111	1	$D_3D_4D_5$	9

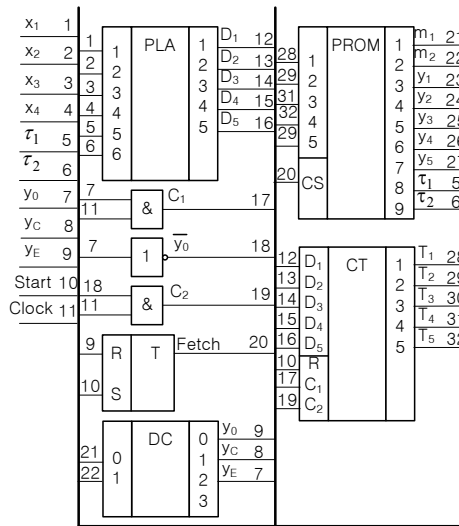


Fig. 8.27 Logic circuit of CMCU $U_{75}(I_{20})$

As in all previous cases, this table is used to construct system (8.15). For example, the equation $D_3 = F_2 \vee F_3 \vee F_4 \vee F_7 \vee F_8 \vee F_9 = \bar{\tau}_1 \bar{\tau}_2 \bar{x}_1 \vee \tau_2 \bar{x}_2 \vee \tau_1$ can be derived from Table 8.15 (after minimization).

The main disadvantage of this approach is the increase of control memory word length, in comparison with CMCU U_2 or U_{74} . This approach allows however to obtain better performance, in comparison with CMCU U_{74} , and smaller chip area occupied by the CMCU circuit, in comparison with CMCU U_2 (absence of TC). The choice among models of CMCU U_2 , U_{74} and U_{75} should be made by the designer, after detailed analysis of requirement specifications for every particular digital system.

Further hardware amount reduction for the CMCU with modified microinstruction formats can be achieved due to the use of multilevel structures of their logic circuits. The methods of logical conditions replacement, encoding collections of microoperations and fields of compatible microoperations, discussed above, can be used for this purpose. Application of these methods leads to longer cycle times and in consequence to lower performance of the designed digital systems. The final choice should be made after analysis of both requirement specifications and used logic elements.

8.4 Synthesis of multilevel circuits for CMCU with modified system of microinstructions

Let us construct Table 8.16 representing models of CMCU with modified system of microinstructions and coding of logical conditions and microoperations. This table is constructed by analogy with Table 7.17. We remember that CMCU U_{75} has two levels, CMCU $U_{63} - U_{74}$ have three levels and CMCU $U_{58} - U_{62}$ have four levels. Therefore, models of CMCU represented by Table 8.16 have from 2 to 6 levels. This table is used in turn to construct Table 8.17, allowing estimation of total number of logic circuit structures for CMCU from Table 8.16.

Table 8.16 Multilevel structures of logic circuits for CMCU with modified system of microinstructions

Levels	A	B	C
Blocks	M_1	U_{58}	W_1
	M_1C		\vdots
	M_1L		W_6
	\vdots	\vdots	D_1
	M_G		\vdots
	M_GC		\vdots
	M_GL	U_{75}	D_J

The number of different CMCU structures with the modified system of microinstructions can be found from Table 8.17 and is equal to:

$$V_0^2 = V_2 + \dots + V_6 = 108 + 378G + 18J + 54GJ. \tag{8.16}$$

According to [1], for the FSM of average complexness we have $G = J = 6$. It can be found from (8.16) that there are $V_0^2 = 4428$ different structures for the discussed CMCU in case of each particular GSA Γ . Using formula (7.34), the total number of different CMCU logic circuit structures can be estimated as

$$V_0 = V_0^1 + V_0^2 = 507 + 1577G + 75J + 225GJ. \tag{8.17}$$

Table 8.17 Estimation of the number of structures for CMCU with modified system of microinstructions

Number of levels	Model of CMCU	k_i	V_i
2	U_{75}	1	$V_2 = 1$
3	$M_g U_{75}$	G	$V_3 = 120 + 45G + 15J$
	$M_g C U_{75}$	G	
	$M_g L U_{75}$	G	
	$U_{75} W_i$	6	
	$U_{75} D_j$	J	
	$U_{63}-U_{74}$	12	
4	$U_{58}-U_{62}$	5	$V_4 = 77 + 54G + 12J + 36GJ$
	$M_g U_{75} W_i$	6G	
	$M_g C U_{75} W_i$	6G	
	$M_g L U_{75} W_i$	6G	
	$M_g U_{75} D_j$	GJ	
	$M_g C U_{75} D_j$	GJ	
	$M_g L U_{75} D_j$	GJ	
	$M_g U_{63}-M_g U_{74}$	12G	
	$M_g C U_{63}-M_g C U_{74}$	12G	
	$M_g L U_{63}-M_g L U_{74}$	12G	
	$U_{63} W_i-U_{74} W_i$	72	
	$U_{63} D_j-U_{74} D_j$	12J	
5	$M_g U_{58}-M_g U_{62}$	5G	$V_5 = 30 + 231G + 5J + 36GJ$
	$M_g C U_{58}-M_g C U_{62}$	5G	
	$M_g L U_{58}-M_g L U_{62}$	5G	
	$M_g U_{63} W_i-M_g U_{74} W_i$	72GJ	
	$M_g C U_{63} W_i-M_g C U_{74} W_i$	72GJ	
	$M_g L U_{63} W_i-M_g L U_{74} W_i$	72GJ	
	$U_{58} W_i-U_{62} W_i$	30	
	$U_{58} D_j-U_{62} D_j$	5J	
	$M_g U_{63} D_j-M_g U_{74} D_j$	12GJ	
	$M_g C U_{63} D_j-M_g C U_{74} D_j$	12GJ	
	$M_g L U_{63} D_j-M_g L U_{74} D_j$	12GJ	
6	$M_g U_{58} W_i-M_g U_{62} W_i$	30G	$V_6 = 90G + 15GJ$
	$M_g C U_{58} W_i-M_g C U_{62} W_i$	30G	
	$M_g L U_{58} W_i-M_g L U_{62} W_i$	30G	
	$M_g U_{58} D_j-M_g U_{62} D_j$	5GJ	
	$M_g C U_{58} D_j-M_g C U_{62} D_j$	5GJ	
	$M_g L U_{58} D_j-M_g L U_{62} D_j$	5GJ	

In case of FSM having average complexity, formula (8.17) determines $V_0 = 18507$ different CMCU logic circuit structures. If we want to synthesize a particular logic circuit, we should apply the optimization methods described above, namely the expanding synthesis methods given before for basic models.

Obviously, the number of different structures estimated by formula (8.17) increases when parameters G and J of the interpreted GSA Γ become higher. For example, if $G = J = 12$ $V_0 = 52731$, that is three times more than for the GSA

having $G = J = 6$. This huge number of possibilities requires some strategy helping to choose the best model of CMCU, that is the one with minimal cost and performance satisfying requirement specifications for each particular control algorithm interpretation.

Strategy adopted for the choice of a particular CMCU model depends on optimality criteria determining the best of all possible solutions. Let us discuss an algorithm oriented towards the choice of model characterized by minimal hardware amount HA_{\min} and which guarantees the required execution time t_A for the interpretation of some graph-scheme of algorithm Γ . Let us use the following notation:

- U_{opt} is a CMCU model with optimal characteristics, that is minimal hardware amount and satisfactory control algorithm execution time;
- i is the number of levels in the CMCU model ($i \leq 6$);
- HA_{\min}^i is the minimal hardware amount for CMCU model U_B^i having i levels;
- t_A^i is the maximal control algorithm execution time, possible if the CMCU model U_B^i is used for interpretation of GSA Γ .

The proposed algorithm (Fig. 8.28) is directed towards finding the best among two-level models and towards consecutive increase of the number of its levels. As we remember, application of the CMCU makes sense only for the interpretation of linear GSA, where the number of operator vertices is not less than 75% of the total number of its vertices. In consequence, the first step of algorithm is the verification, whether the initial GSA to be interpreted is a linear GSA. If the examination result is negative, some other models, different from the CMCU, should be used to interpret this particular algorithm. In this case, operation of searching algorithm is terminated (output "0" for algorithm block 1).

If the examination result is that we deal with a linear GSA (output "1" for algorithm block 1), the analysis of two-levels models is initiated ($i := 2$), because the minimal number of levels in the CMCU models equals two.

In order to find the best model U_B^2 (block 2) it is necessary to analyze all two-level structures, for which the control algorithm execution time does not exceed the value t_A . If more than one model have equal control algorithm execution time, model with minimal hardware amount should be chosen. If there are no two-level models with control algorithm execution time $t_A^c \leq t_A$ (output "0" from block 3), the CMCU model cannot be used and searching for the algorithm is terminated.

If the desired two-level model U_B^2 with $t_A^i \leq t_A$ exists (output "1" from block 3), it should be chosen as an optimal solution U_{opt} . Next, the analysis should start for CMCU models having more than two levels. For all models with the number of levels equal to $i + 1$, the best model is chosen and its parameters HA_{\min}^i and t_A^i found. At the same time, possibility of optimization is analyzed for the model U_{opt} , found after execution of the previous step. This result should be compared with CMCU models having $(i + 1)$ levels and for which neither replacement of logical conditions nor microoperation encoding is used (block 4).

If the best solution found during step i ($i = 2, \dots$) does not guarantees the required performance, previous best solution U_{opt} is used as the final one (output "0" from block 5). Otherwise, hardware amounts found for models U_B^i and U_{opt} should

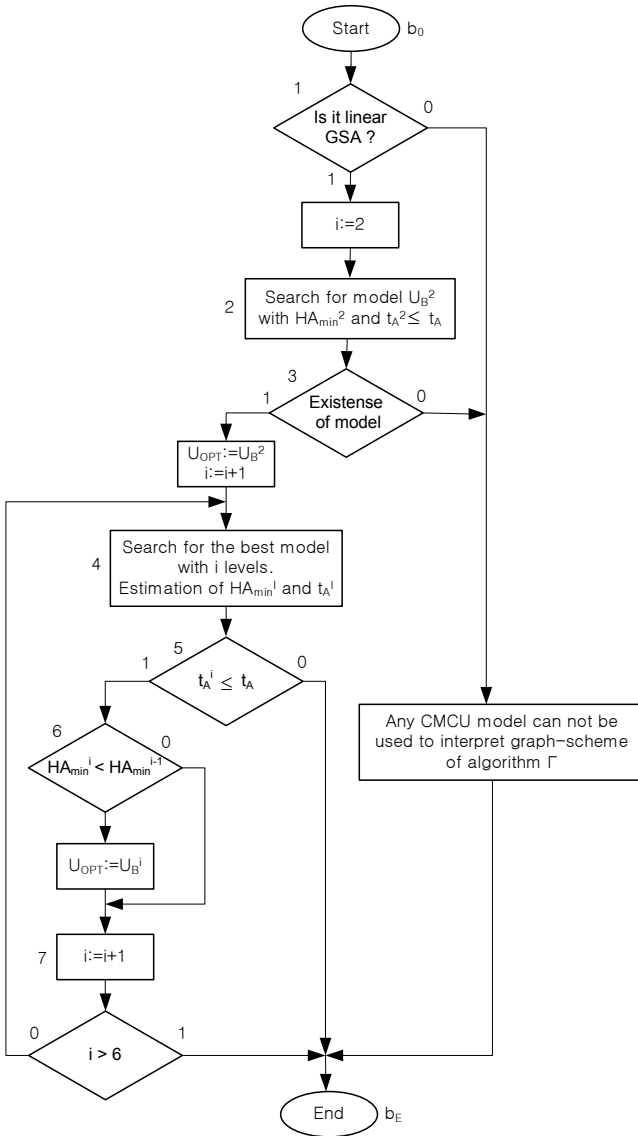


Fig. 8.28 Algorithm for searching the best CMCU model

be compared (block 6). If the hardware amount is reduced (output "1" for block 6), this new model should be used as the best solution ($U_{opt} := U_B^i$). Otherwise, the outcome of previous step is used as the best solution U_{opt} . Next, the number of levels is incremented (block 7) and the process of solution extension (increase of the number of levels in a final CMCU model) is repeated if $i \leq 6$. If however $i > 6$ ($i = 7$), it

means that all possible models have been already analyzed and model U_{opt} should be used to implement the logic circuit of CMCU with given logic elements.

Obviously, the proposed algorithm represents only one of many possible approaches and the problem of finding the best CMCU model is still open for further researches.

References

1. S. I. Baranov. *Logic Synthesis of Control Automata*. Kluwer Academic Publishers, 1994.
2. A. A. Barkalov. *Synthesis of Control Units with PLDs*. Donetsk National Technical University, 2002. (in Russian).
3. A.A. Barkalov and L.A. Titarenko and M. Kołopencyk. Optimization of control memory size of control unit with codes sharing. In *Proc. of the IXth Inter. Conf. CADSM 2007 "The Experience of Designing and Application of CAD Systems in Microelectronics"*, pages 242 – 245, Lviv–Polana, Ukraine, 2007.
4. A.A. Barkalov, L. A. Titarenko, and J. Bieganski. Synthesis of control unit with modified microinstructions. In *Proc. of the Inter. Conf. Mixed Design of Integrated Circuits and Systems – MIXDES 2007*, pages 157 – 160, 2007.
5. A.A. Barkalov, L. A. Titarenko, and J. Bieganski. Synthesis of control unit with modified system of microinstructions. In *Proceedings of IEEE East-West Design & Test Symposium – EWDTs '07*, pages 545 – 549, 2007.

Conclusion

The development of semiconductor technology turns microelectronics into nanoelectronics and results in the appearance of very complex chips including as much as billion elements. Expert's forecasts predict that the number of transistors per chip will exceed 10 billions till 2015 [1, 5]. At the same time there is a tremendous gap in the area of design methods permitting to take advantage of potential chip capabilities [3]. Besides, up-to-day industrial computer-aided design tools do not use many results that could be found in the CAD tools developed by some university scholars. For example, Tadeusz Łuba [4] proved that his design methods oriented towards FPGA technology allow to get logic circuits of digital devices having smaller hardware amount and better performance than similar circuits designed by means of the well-known industrial CAD tools. The same was proved by Dariusz Kania and Valery Solovjov [2, 6] after comparing the outcomes of their own design methods, oriented on the CPLD technology, with the results obtained by means of industrial CAD tools. But as we know, no big company uses their achievements up to now.

We hope that this state of affairs cannot last forever. A moment shall come, when the industry will be in need to exploit design methods and tools developed by the universities. But now we see our task as the development of new models of control units as well as of formal methods of their synthesis. The class of compositional microprogram control units discussed in this book is based on our original results and is oriented towards effective hardware interpretation of the linear control algorithms. This class occupies some intermediate place between finite state machines on the one hand, and microprogram control units on the other hand. It allows using the well-known optimization methods, which proved already to be useful for both extreme classes of the control units.

We hope that this book will be useful for the designers of digital systems and scholars developing synthesis and optimization methods oriented towards the control units. It would permit to take next step towards the convergence of outcomes achieved by university researches with the requirements of industry.

References

1. H. Iwai. Future CMOS scaling. In *Proc. 11th Conf. Mixed Design of Integrated Circuits and Systems, MIXDES 2004*, pages 12–18, Szczecin, Poland, 2004. Department of Microelectronics and Computer Science, Technical University of Łódź.
2. D. Kania. *Logic synthesis for PAL-oriented programmable structures*. Zeszyty naukowe Politechniki Śląskiej, Gliwice, 2004. (in Polish).
3. C. Maxfield. *The Design Warrior's Guide to FPGAs*. Academic Press, Inc., Orlando, FL, USA, 2004.
4. M. Rawski, T. Luba, Z. Jachna, and P. Tomaszewicz. *Design of Embedded Control Systems*, chapter The influence of functional decomposition on modern digital design process, pages 193–203. Springer, Boston, 2005.
5. K. Sakamura. Future SoC possibilities. *IEEE Micro.*, (5):7, 2002.
6. V. V. Solovjev. *Design of Digital Systems Using the Programmable Logic Integrated Circuits*. Hot line – Telecom, Moscow, 2001. (in Russian).

Index

- address transformer, 88, 89
- addressing of microinstructions
 - combined, 21, 22
 - compulsory, 11, 12
 - natural, 16, 17, 73
 - optimal, 81, 86
 - special, 81, 82
- algorithm, 2

- block of GSA, 79
- block representation of GSA, 80
- Boolean
 - function, 32
 - space, 9
 - variable, 7

- class
 - equivalence, 37
 - pseudoequivalent states, 9
- code transformer, 52
- combinational circuit, 4
- complex programmable logic device (CPLD), 32
- compositional microprogram control unit (CMCU), 22
 - with basic structure, 24
 - with code sharing, 99
 - with common memory, 72
- computer aided design system
 - ASYL, 9, 43
 - DOMAIN, 43, 58
 - ES Series, 58
 - MAX+PLUS II, 58
 - NOVA, 9
 - Quartus, 58
 - SIS, 58
 - ZUBR, 43, 58

- control memory content, 11
- control unit, 2

- data-path, 2
- don't care input assignment, 37

- encoding of
 - classes of pseudoequivalent operational linear chains, 89
 - collections of microoperations, 36
 - components of
 - elementary operational linear chains, 130
 - operational linear chains, 109
 - elementary operational linear chains, 120, 122, 126
 - fields of compatible microoperations, 35, 37
 - operational linear chains, 102
- ESPRESSO, 37, 108, 188, 216, 218
- expanded microinstruction, 160, 161

- field-programmable gate arrays (FPGA), 33, 34
- field-programmable logic devices (FPLD), 25, 27, 34, 36, 39, 41, 42
- finite state machine (FSM), 4
 - Mealy, 5
 - Moore, 5

- generalized interval of Boolean space, 9, 82, 84
- generic array logic (GAL), 32, 33
- graph-scheme of algorithm (GSA), 2, 3
 - linear, 1, 22
 - marked, 5
 - transformed, 13, 18
 - vertical, 40, 46

- hardware description language (HDL), 57

- input memory functions, 5
- input of OLC, 23
- Karnaugh map, 9, 37
- logic circuit, 5, 16
- logical condition, 1, 2
- look-up table (LUT) element, 33, 37
- macrocell, 32, 216
- main input of
 - EOLC, 121
 - OLC, 23, 83
- memory
 - common, 72
 - control, 3, 11
- microinstruction
 - control, 16
 - operational, 16
- microinstruction address, 11, 12
- microoperation, 1–3, 6
- microprogram, 2
- microprogram control, 1, 2
- microprogram control unit (MCU), 10, 11
- microprogramming, 34–36, 39
- multiplexer, 15, 16, 21
- one-hot encoding of microoperations, 13, 35
- operational linear chain, 81
- operational linear chain (OLC), 23
- optimal encoding of
 - elementary operational linear chains, 121, 125
 - operational linear chains, 107, 109
 - states, 9
- output of OLC, 23
- partition on
 - classes of pseudoequivalent operational linear chains, 84
 - classes of pseudoequivalent operational linear chains, 125
 - classes of pseudoequivalent states, 9
- product term, 7
- programmable array logic (PAL), 32, 33
- programmable logic array (PLA), 29, 30
- programmable logic sequencer (PLS), 31, 32
- programmable read-only memory (PROM), 27, 28
- pseudoequivalent
 - elementary operational linear chains, 126
 - operational linear chain, 79
 - states, 9
- random-access memory (RAM), 34, 71
- read-only memory (ROM), 9
- register transfer level (RTL), 57
- replacement of logical conditions, 44
- special encoding of equivalence classes, 112
- state
 - assignment, 5, 6, 42, 43
 - code, 72
 - internal, 47
 - pseudoequivalent, 9
 - variables, 5
- structure table of FSM, 70
- sum of products (SOP), 7, 34
- synthesis, 27, 31, 42, 43, 56
- system-on-a-programmable chip (SoPC), 34
- table of
 - address transformer AT, 167
 - address transformer TAS, 148
 - code transformer TC, 156
 - code transformer TOK, 153
 - code transformer TSA, 145
- transformation of
 - addresses of OLC inputs, 139
 - elementary OLC codes, 253
 - initial GSA, 12
 - microinstruction addresses, 88, 160
 - object codes, 149, 206, 235
 - OLC codes, 139, 190
 - OLC output codes, 81
 - structure table, 199
 - transformed GSA, 82
- transformed
 - GSA, 13, 18
 - structure table, 200
- transition
 - formula, 69
 - table, 75, 78
- variable
 - feedback, 71, 255
 - state, 6, 77, 197
- verticalization of GSA, 40, 224