Chapter 2
Field Programmable Gate Arrays in FSM Design

2.1 General Characteristic of FPGAs

The field programmable gate arrays (FPGA) were introduced in 1984 [53] by Xilinx. It is very hard to overestimate the FPGA influence on different areas of electrical engineering [77]. These chips are very flexible due to their property of reconfigurability. Contemporary FPGA chip can replace billions of 2NAND gates [54], called system gates. In the beginning, FPGAs were used for implementing such parts of electrical circuits as glue logic. Now, these chips have up to 7 billion transistors [75], possess clock frequency exceeding the Gigahertz. The most advanced FPGAs have the technology less than 20 nm [2, 52, 58, 64, 89].

First FPGAs, XC2064, were produced by Xilinx in 1985. They included up to 85 000 transistors, 128 logic cells, 64 configurable logic blocks; they have the clock frequency up to 50 MHz. But they made huge progress and now they are very powerful devices.

To show this progress, let us start from the family Spartan-3 of Xilinx [89]. These chips were introduced in 2002, were powered by 1, 2 V and used the 90 nm technology. They included look-up table (LUT) elements having 4 inputs. These chips included up to 104 embedded memory blocks named blocks of RAMs (BRAM). So, it was up to 1,87 Mbits of BRAMs in these FPLDs. They operated with the frequency from 25 MHz till 325 MHz. Each chip was equivalent from 50 000 to 5×10^6 system gates (SG). Some characteristics of Spartan-3 family are shown in Table 2.1.

The last column of Table 2.1 includes the capacity of memory distributed among LUTs of the chip. It is named distributed RAM (DRAM).

In this chapter we discuss only the basic features of FPGAs relevant to implementing logic circuits of control units. Let us analyse peculiarities of LUT-based FPGAs. As a rule, typical FPGAs include four main elements: configurable logic blocks (CLB) based on LUTs; matrix of programmable interconnections (MPI); input-output blocks (IOB) and embedded memory blocks. The organization of an FPGA chip is shown in Fig. 2.1.

© Springer International Publishing Switzerland 2016

A. Barkalov et al., *Logic Synthesis for FPGA-Based Finite State Machines*,
Studies in Systems, Decision and Control 38, DOI 10.1007/978-3-319-24202-6_2

Table 2.1 Characteristics of Spartan-3 family

Device	Number of CLBs	Number of SGs (K)	Capacity of BRAMs, bits (K)	Capacity of DRAM, bits (K)
XC3S50	1 728	50	72	12
XC3S200	4 320	200	216	30
XC3S400	8 064	400	288	56
XC3S1000	17 280	1000	432	120
XC3S1500	29 952	1500	576	208
XC3S2000	46 080	2000	720	320
XC3S4000	62 208	4000	1728	432
XC3S5000	74 880	5000	1872	520

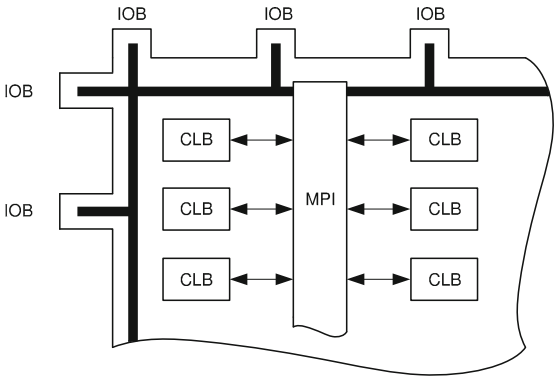


Fig. 2.1 Simplified organization of FPGA

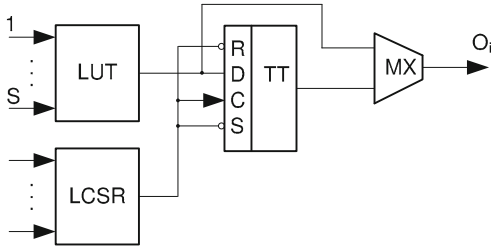


Fig. 2.2 Simplified organization of CLB

As a rule, LUTs are based on RAM having limited amount of inputs S ($S \leq 6$). A single LUT can implement an arbitrary Boolean function depended on L input variables ($L \leq S$) and represented by a truth table. A typical CLB included a single LUT, programmable flip-flop (FF), multiplexer (MX) and logic of clock and set-reset (LCSR). The simplified structure of CLB is shown in Fig. 2.2.

The output of LUT is connected with FF which could be programmed as D, JK or T flip-flop. The FF could be by-passed due to programmable MX. So, the output O_i of a CLB can be either combinational or registered. The existence of flip-flops allows organization of either registers or counters. Both these devices are used for FSM implementation.

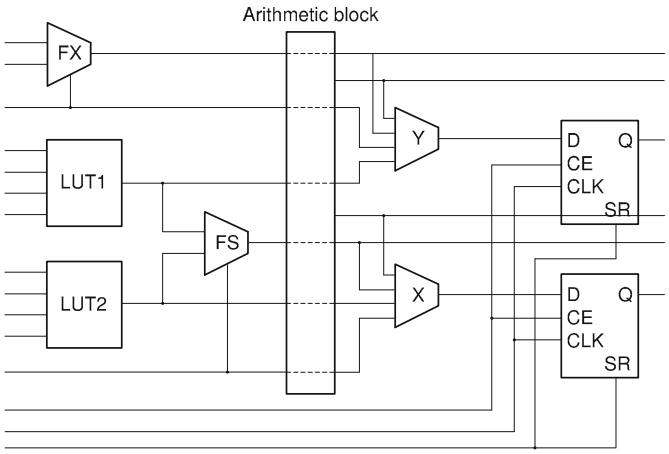


Fig. 2.3 Structure diagram of a slice of Virtex-4 family

With development of technology, more and more sophisticated CLBs were introduced. Let us discuss, for example the CLB of Virtex-7. It includes 4 slices having fast interconnections. A slice includes 2 LUTs, 4 multiplexers, arithmetic logic and 2 programmable flip-flops (Fig. 2.3).

This slice includes 2 LUTs; each of them has $S = 4$ inputs. Each LUT can implement an arbitrary logic function depended on 4 variables. Using the multiplexer FS, both LUTs are viewed as a single LUT having $S = 5$. The multiplexer FX combines together outputs of FS and FX from other slices. So, a slice can implement a Boolean functions depending on 5 variables; two slices on 6 variables; four slices (a CLB) on 7 variables. The arithmetic block allows organizing adders and multipliers. Multiplexers Y and X determine input data for programmable flip-flops. So, each CLB can include either RG or CT.

The number of inputs per a LUT is increased up to 5 for Virtex-5 family, whereas CLBs of Virtex-6 and Virtex-7 include LUTs having $S = 6$. There are different modifications of FPGAs for each family. We do not discuss them. Some characteristics of modern FPGA chips by Xilinx are shown in Table 2.2.

Analysis of Tables 2.1 and 2.2 proves our statement about the tremendous progress in FPGAs. Let us point out that modern chips include blocks of digital signal processors and central processing units. But these blocks are not used for FSM design. So, we do not discuss them.

At present, five companies dominate on the FPGA market: Xilinx, Altera, Lattice Semiconductor, Microsemi and Quicklogic. It can be found all necessary data about products of these companies on their homepages [2, 52, 58, 64, 89].

As follows from Table 2.2, there are powerful EMBs in the modern FPGAs. They have a property of configurability. It means that such parameters as number of memory cells (V) and their outputs (t_F) can be changeable. Of course, there is the constant size (V_o) of an EMB. Now, there are the following typical configurations of EMBs:

Table 2.2 Characteristics of FPGAs by Xilinx

Family	Modification	Number of Slices	Capacity of BRAMs, k bits	Capacity of DRAM, k bits	Technology, nm
Virtex-4	LX	10752–89088	1296–6048	168–1392	90
	SX	10240–24576	2304–5760	160–384	
	FX	5472–63168	648–9936	86–987	
Virtex-5	LX	4800–51840	1152–10368	320–3420	65
	LXT	3120–51840	936–11664	210–3420	
	SXT	5440–37440	3024–18576	520–4200	
	TXT	17280–24320	8208–11664	1500–2400	
	FXT	5120–30720	2448–16416	380–2280	
Virtex-6	LXT	11640–118560	5616–25920	1045–8280	40
	SXT	49200–74400	25344–38304	5090–7640	
	HXT	39360–88560	18144–32832	3040–6370	
	CHT	11640–37680	5616–14976	1045–3650	
Virtex-7	T	44700–305400	14760–46512	3475–21550	28
	XT	64400–135000	31680–64800	6525–13275	
	HT	45000–135000	21600–64800	4425–13275	

Fig. 2.4 Karnaugh map for system Y

T_1 \ T_2T_3	00	01	11	10
0	0	y_1y_2	y_2y_3	y_3
1	y_1y_2	y_1y_4	*	*

36 K ×1, 18 K ×2, 8 K ×4, 4 K ×8, 2 K ×16, 1 K ×32 and 512 ×64 (bits) [2, 52, 58, 64, 89]. Let us discuss how these blocks can be used in logic design.

Let an EMB contain V cells having t_F outputs. Let V_o be a number of cells if there is $t_F = 1$. So, the number V can be determined as

$$V = \lceil V_o / t_F \rceil. \tag{2.1}$$

Let us point out that decreasing t_F by 1 leads to doubling for the number of cells (and vice versa).

Embedded memory blocks could be used for implementing regular functions [8]. Let us discuss the Karnaugh map (Fig. 2.4) corresponding to Table 1.3.

The following functions can be found from this map:

$$\begin{aligned} y_1 &= T_1 \vee \bar{T}_2 T_3; \\ y_2 &= \bar{T}_1 T_3 \vee T_1 \bar{T}_3; \\ y_3 &= T_2;\ y_4 = T_1 T_3. \end{aligned} \tag{2.2}$$

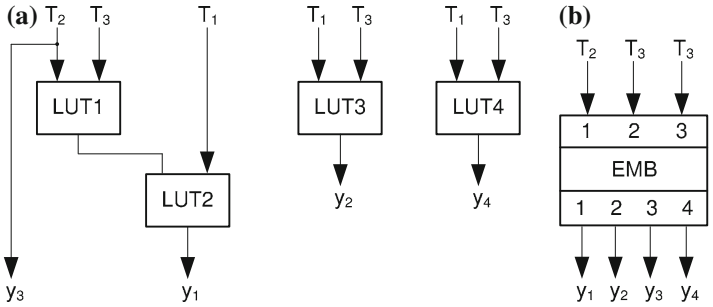


Fig. 2.5 Logic circuit for system (2.2)

Let us use system gates (LUTs with s = 2) for implementing the system (2.2).

From Fig. 2.2 the following negative features could be seen: (1) different propagation time for different output functions; (2) input variables should be connected with more than one logic element. The second feature leads for necessity of bigger value of fan-out for inputs T_1–T_3 than in the case when each input is connected only with a single logic element. Also, it leads to complications in the routing process.

If an EMB is used for implementing the system (2.2), all these problems are absent (Fig. 2.5b). Each input T_r ($r = \overline{1,3}$) is connected only with a single input of EMB. All functions y_n ($n = \overline{1,4}$) have the same propagation time. This example is very simple. But a simple EMB having the configuration 512×64 could replace at least 64 LUTs. It is possible if a system Y depends on 9 inputs and includes up to 64 different functions. Of course, the circuit includes 64 LUTs having $S = 9$. But there is no such LUTs in modern FPGAs. If minimization allows dependence of each function $y_n \in Y$ on 8 variables, then 256 LUTs with $S = 6$ are necessary for creating a logic circuit.

In a typical FPGA 60 % of power is consumed by the programmable interconnects, 16 % is consumed by programmable logic and 14 % by the clock distribution [14]. Replacement of LUTs by EMBs allows decreasing of the number of interconnections. So, it is very important to use EMBs in implementing FSM circuits.

The exceptional complexity of FPGA requires using computer-aided design (CAD) tools for designing logic circuits [41]. It assumes development of formal methods for synthesis and verification of control units [36, 37, 43, 59]. For example, a design process for FPGAs from Xilinx includes the following steps:

1. **Specification of a project**. A design entry can be executed by the schematic editor (if a design is represented by a circuit), or the state editor (if a design is represented by a STG) or a program written with some hardware description language (HDL). The most popular HDLs are VHDL and Verilog [6, 7]. This initial specification is verified and corrected if necessary.
2. **Logic synthesis**. During this step, the package FPGA Express executes synthesis and optimization of an FSM logic circuit. As an outcome of this step, an FPGA

Netlist file is generated. This file is represented in either EDIF or XNF format. During this step, library cells from system and user libraries are used.

3. **Simulation**. The functional correctness of an FSM is checked. This step is executed without taking into account real propagation times in a chip. If the outcome of simulation is negative, then the previous steps should be repeated.
4. **Implementation of logic circuit**. Now the Netlist is translated into an internal format of CAD system. Such physical objects as CLBs and chip pins are assigned for initial physical elements. This step is named the packing. The step of mapping is the first stage of the packing. The mapping refers to the process of associating entities such as the gate-level functions in the gate-level netlist with the LUT-level functions available on the FPGA [53]. It is not a one-to-one mapping because each LUT can be used to represent a number of logic gates [42]. The mapping step gives results for executing the packing. During this step the LUTs and flip-flops are packed into the CLBs. Both mapping and packing steps are very difficult because there are too many variants of their solutions. Following packing the step of place-and-route is executed. Now we know the connection between CLBs and parts of a logic function to be implemented. But there are many ways how these CLBs could be placed in the FPGA. The placement problem is very difficult because hundreds of thousands or even millions CLBs should be placed. During the routing it is necessary to decide how to connect all CLBs for a particular project. This step should be executed in a way giving the maximum possible performance. Obviously, the outcome of placement affects tremendously the outcome of routing.
   When routing is finished, the real performance could be found. Also, the BitStream is formed which will be used for chip programming.
5. **Project verification**. The final simulation is performed where the actual values of delays among the physical elements of a chip are used. If outcome of this step is negative (the actual performance of an FSM is less than it is necessary), then the previous steps of the design process should be repeated.
6. **Chip programming**. This step is connected with the writing of the final bit stream into the chip.

One of the most important roles in the design process plays the step of logic synthesis. Let us analyse the steps and possible solutions for FPGA-based FSMs.

2.2 Trivial Implementing FPGA-Based FSMs

The process of FSM synthesis can be viewed as a transformation of FSM initial specification into structural specifications where elements of lower abstraction levels are used [1, 17, 21]. This process is repeated till each element to be designed is represented by some library element. In this book, we start from GSAs and finish by LUTs and EMBs.

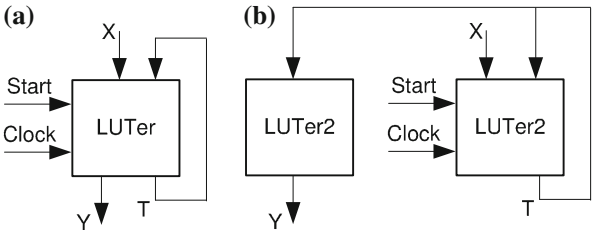


Fig. 2.6 Trivial structural diagrams of LUT-based Mealy (**a**) and Moore (**b**) FSMs

Let us start from LUT-based P Mealy FSM (Fig. 2.6a). It includes a LUTer which is a circuit implemented with LUTs. There is no register in the explicitly as it is in Fig. 1.4. Now, the register is formed by flip-flops distributed among LUT-elements.

Let us discuss an example of design for Mealy FSM $P(\Gamma_1)$. There is the marked GSA Γ_1 shown in Fig. 1.5a. To design the circuit of LUTer, it is necessary to find the functions (1.3) and (1.4). In the case of $P(\Gamma_1)$, these functions are represented as (1.13). Let the symbol $L(D_r)$ stand for the number of literals in the SOP of function $D_r \in \Phi$. Let the symbol $L(y_n)$ stand for the number of literals in the SOP of function $y_n \in Y$. Let the following condition take place:

$$S \geq max(L(D_1), \ldots, L(D_R), L(y_1), \ldots, L(y_N)). \tag{2.3}$$

In (2.3), the symbol S stand for the number of inputs of a LUT. In this case, each function of FSM is implemented using only a single LUT.

There are the following values of $L(D_r)$ and $L(y_n)$ in the case of $P(\Gamma_1)$: $L(D_1) = L(y_4) = 1, L(D_2) = 2, L(D_1) = 4, L(y_2) = L(y_3) = 3$. Let us implement the circuit of $P(\Gamma_1)$ using LUTs having $S = 4$ inputs. It leads to the circuit shown in Fig. 2.7. Let us point out that we use Eq. (1.13).

There are three LUTs in the part implementing functions $y_n \in Y$. There is only a wire T_2 in the circuit for $y_4 = T_2$. There are two LUTs in the part of the circuit implementing input memory functions $D_r \in \Phi$. They have additional inputs for pulses Start and Clock. Of course, they are not address inputs as it follows, for example, from Fig. 2.2. There is $D_1 = T_2$, but it is still necessary a LUT to implement this function.

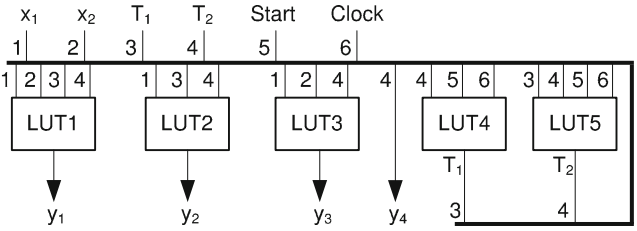


Fig. 2.7 Logic circuit of Mealy FSM $P(\Gamma_1)$

Table 2.3 Table for programming LUT1

T_1	T_2	x_1	x_2	y_1	T_1	T_2	x_1	x_2	y_1	T_1	T_2	x_1	x_2	y_1	T_1	T_2	x_1	x_2	y_1
0	0	0	0	1	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1
0	0	0	1	1	0	1	0	1	0	1	0	0	1	1	1	1	0	1	1
0	0	1	0	1	0	1	1	0	0	1	0	1	0	1	1	1	1	0	1
0	0	1	1	1	0	1	1	1	0	1	0	1	1	1	1	1	1	1	1

To program LUT1–LUT5, they should construct truth tables corresponding to functions y_1, y_2, y_3, D_1 and D_2. Let us discuss this step for the function y_1 from (1.13). It is represented by the following SOP:

$$y_1 = \bar{T}_1\bar{T}_2 \vee T_2\bar{x}_1\bar{x}_2 \vee T_1. \tag{2.4}$$

To form a truth table, this SOP should be transformed into the perfect SOP [55] where each term has exactly $L(y_1)$ literals. To do it, we should find the following functions: $\bar{T}_1\bar{T}_2(x_1 \vee \bar{x}_1)(x_2 \vee \bar{x}_2)$; $T_2\bar{x}_1\bar{x}_2(T_1 \vee \bar{T}_1)$; $T_1(T_2 \vee \bar{T}_2)(x_1 \vee \bar{x}_1)(x_2 \vee \bar{x}_2)$. After transformation (2.4), the table of LUT1 can be constructed (Table 2.3).

The term $\bar{T}_1\bar{T}_2$ is represented by input assignments from 0000 till 0011; the term $T_2\bar{x}_1\bar{x}_2$ by 0100 and 1100; the term T_1 by input assignments from 1000 till 1111.

Using this approach, it can be found tables of programming for each function of FSM. Next, the corresponding bit-streams should be created and loaded into LUTs [77].

The circuit from Fig. 2.7 is a single-level circuit, because there is only a single level of logic among inputs $x_e \in X$, $T_r \in T$ and outputs of FSM. It is possible only if the condition (2.3) takes place. But it is violated very often in practical cases of FSM design [77].

A small amount of inputs per LUT creates a big problem for logic design. Let us consider some input memory function D_1 depending on $L(D_1) = 7$ Boolean variables:

$$D_1 = T_1\bar{T}_2T_3x_1\bar{x}_2 \vee T_1T_2T_3x_3x_4 \vee \bar{T}_1\bar{T}_2\bar{T}_3\bar{x}_1x_3. \tag{2.5}$$

Let LUTs in use have $S = 7$ inputs. In this case, the logic circuit for D_1 includes only a single LUT Fig. 2.8a. Now, let LUT have $S = 6$. In this case the function (2.5) should be transformed. It should be represented by some functions $f_1, f_2, \ldots$ having $L(f_1) \leq 6$, $L(f_2) \leq 6$, and so on. Let us represent the function (2.5) in the following form:

$$D_1 = T_1(\bar{T}_2T_3x_1\bar{x}_2 \vee T_2T_3x_3x_4) \vee \bar{T}_1\bar{T}_2T_3\bar{x}_1x_3 = T_1A \vee B. \tag{2.6}$$

The function D_1 represented as (2.6) requires two LUTs to be implemented. Moreover, this circuit Fig. 2.8b includes two levels of LUTs. It means that the solution corresponding to (2.6) is twice slower.

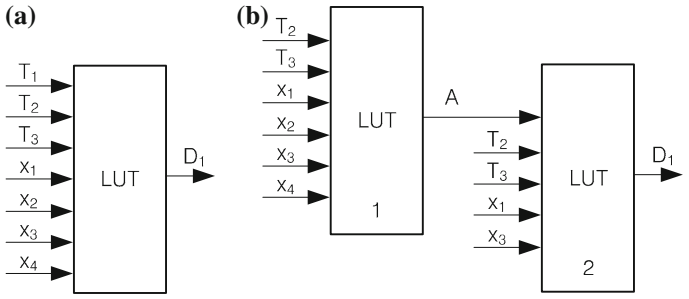


Fig. 2.8 Implementing function D_1 with LUTs having $S = 7$ (**a**) and $S = 6$ (**b**)

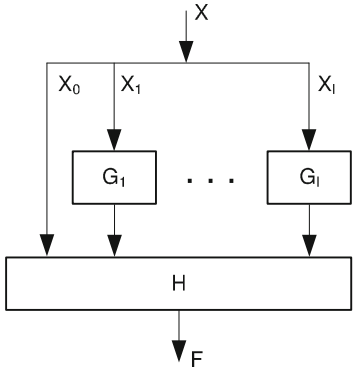


Fig. 2.9 Illustration of the principle of functional decomposition

This approach is named functional decomposition. The principle of functional decomposition is the basic one for FPGA-based design [65, 71]. There are basically approaches targeting only LUTs. But methods exists using EMBs as tools for implementing some subfunctions [18, 66].

In general, the method of functional decomposition is based on representation of a Boolean function $F(X)$ in the following form:

$$F(X) = H(X_0, G_1(X_2), \ldots, G_I(X_I)). \tag{2.7}$$

The Eq. (2.7) corresponds to the implementation of the circuit shown in Fig. 2.9.

The negative influence of functional decomposition is increasing of the propagation time in comparison with a single-level circuit. It follows from comparison of the circuits from Fig. 2.8. As it is mentioned in [65], the methods of functional decomposition are far from ideal. Let us point out that it is very important to decrease the numbers of arguments and product terms in Boolean functions to be implemented. We discuss these methods a bit further.

Now, let us discuss the trivial LUT-based implementing P Moore FSM. Its structural diagram is shown in Fig. 2.6b. The LUTer1 implements the functions (1.5), the LUTer2 the functions (1.4). There is a distributed register hidden among LUTs of LUTer2.

Let us discuss an example of design for Moore FSM $P(\Gamma_1)$. There is the marked GSA Γ_1 shown in Fig. 1.5b. To design the logic circuit, it is necessary to find the functions (1.4) and (1.5). In the discussed case, the system (1.5) is represented by (1.17), whereas the system (1.4) should be derived from Table 1.2. It is the following system:

$$\begin{aligned} D_1 &= \bar{T}_1\bar{T}_2T_3\bar{x_1}\bar{x_2} \vee \bar{T}_1T_2 \vee T_1\bar{T}_2\bar{T}_3; \\ D_2 &= \bar{T}_1\bar{T}_2\bar{T}_3x_1 \vee \bar{T}_1\bar{T}_2\bar{T}_3x_2; \\ D_3 &= \bar{T}_2\bar{T}_3 \vee \bar{T}_1\bar{T}_2T_3\bar{x_1}x_2 \vee \bar{T}_1T_2. \end{aligned} \tag{2.8}$$

If the condition (2.3) takes place, then circuits of LUTer1 and LUTer2 have only a single level of logic.

In the discussed case, it should be $S = 5$. Let us point out that functions (1.5) can be optimized using a proper state assignment.

The following system can be derived from Fig. 2.5b:

$$\begin{aligned} y_1 &= A_2 \vee A_5 \vee A_6; \\ y_2 &= A_2 \vee A_4 \vee A_5; \\ y_3 &= A_3 \vee A_4;\ y_4 = A_6. \end{aligned} \tag{2.9}$$

Let us encode states of Moore FSM $P(\Gamma_1)$ as it is shown in Fig. 2.10.

After minimizing, the following system of equations can be found instead of (2.9):

$$\begin{aligned} y_1 &= T_1; \\ y_2 &= T_2; \\ y_3 &= \bar{T}_1T_3; \\ y_4 &= T_1\bar{T}_2. \end{aligned} \tag{2.10}$$

It is necessary 2 of LUTs having $S = 2$ to implement the system (2.10). The functions $y_1, y_2 \in Y$ are represented just by the outputs of LUTs from the LUTer2. Let us point out that it is necessary 3 of LUTs having $S = 3$ to implement the system (1.17). It shows the big importance of the state assignment step.

FPGA-based devices can be found in many areas where the consumed energy is a critical factor [53]. It concerns portable computing devices, wireless telecommunication equipment, space-based applications [53]. The rising of FPGA complexity leads to increasing the power consumed by FPGA-based devices. It is well known that FSMs consume a significant amount of power in any FPGA-based project [22]. Therefore, minimizing power consumed by the FSMs can significantly reduce the total power consumed by a device.

Fig. 2.10 State codes of Moore FSM $P(\Gamma_1)$

T_1 \ T_2T_3	00	01	11	10
0	a_1	$*$	a_2	a_6
1	a_3	a_4	a_5	$*$

The dynamic power dissipated in CMOS circuits can be expressed by the well-known formula [81]:

$$P = \sum_{n=1}^{N} C_n f_n V_{DD}^2. \tag{2.11}$$

In (2.11), N is the number of elements, C_n is the load capacitance at the output of the element number n, f_n is the frequency of its switching, and V_{DD} is the supply voltage. One of the ways for decreasing the power dissipation is decreasing of the switching activity of flip-flops [81].

There is a very interesting result of investigations conducted by the authors of the article [81]. They found that the smaller FSM circuit consumes less power than its bigger version. It is clear because a smaller circuit needs less interconnections than its bigger counterpart. One of the ways leading to smaller FSM circuits is application of EMBs for implementing some parts of FSM circuits [76]. It is shown that FSM implementation with EMBs provides some benefits compared to synthesis with LUTs [34, 72]. The maximum clock frequency of an FSM implemented in a ROM block is independent on its complexity of course, it is possible if the whole circuit is implementing using just a single EMB. The memory blocks of FPGAs provide control signals that allow for module deactivation when the FSM is inactive. It provides an efficient mechanism for power saving. It has been proved [83] that complex FSMs consume less power when implemented as memory blocks. Let us consider some EMB-based models of FSMs.

In the simplest case, P FSMs can be implemented using a single EMB (Fig. 2.11). There is an external register shown in Fig. 2.11. If an EMB is synchronized, then pulses Clock and Start enter the EMB. But it does not change the design method. The structural diagram (Fig. 2.11) is general for both Mealy and Moore FSMs. We use symbol EMBer to define the circuit implemented with EMBs.

In the case of P Mealy FSM, the EMBer implements systems (1.3) and (1.4). To design it, it is necessary to construct the table of EMBer having columns T, X, Y, Φ, q. Here sets T and X determine addresses of cells, the sets Y and Φ content of cells, q is the number of a cell.

In the case of Mealy FSM $P(\Gamma_1)$, there are the following sets $T = \{T_1, T_2\}$, $X = \{x_1, x_2\}$, $Y = \{y_1, \ldots, y_4\}$ and $\Phi = \{D_1, D_2\}$. So, there are $R_0 = 2$, $L = 2$, $N = 4$. It is necessary to have an EMB with 16 cells having up to 6 bits (Table 2.4).

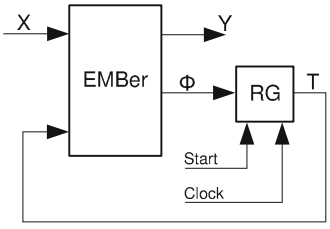


Fig. 2.11 Structural diagram of EMB-based P FSM

Table 2.4 Content of EMBer of Mealy FSM $P(\Gamma_1)$

T	X	Y	Φ	q	T	X	Y	Φ	q
T_1T_2	x_1x_2	$y_1y_2y_3y_4$	D_1D_2		T_1T_2	x_1x_2	$y_1y_2y_3y_4$	D_1D_2	
00	00	1100	01	1	10	00	1001	00	9
00	01	1100	01	2	10	01	1001	00	10
00	10	1100	01	3	10	10	1001	00	11
00	11	1100	01	4	10	11	1001	00	12
01	00	1100	10	5	11	00	0000	00	13
01	01	0110	10	6	11	01	0000	00	14
01	10	0010	10	7	11	10	0000	00	15
01	11	0010	10	8	11	11	0000	00	16

The macrocells 1–4 correspond to row 1 of Table 1.1. The macrocell 5 corresponds to row 4, the macrocell 6 to row 3, macrocells 7, 8 to row 2 of Table 1.1. The macrocells 9–12 correspond to row 5 of Table 1.1. There is no state a_m having the code 11. So, the macrocells 13–16 have no counterparts in Table 1.1. Their content is ignored and can be any. They are filled by zeros in Table 2.4.

EMBs of EMBer (Fig. 2.11) should satisfy to the following condition:

$$2^{R+L} \le V_0. \tag{2.12}$$

In the case of Mealy FSM, there is $R = R_0$. In the case of Moore FSM, there is $R = R_1$. The condition (2.12) shows that an EMB has enough cells to implement a table similar to Table 2.4.

The number of EMBs in EMBer can be found using the following expression:

$$n_{EMB} = \left\lceil \frac{N + R}{t_F} \right\rceil. \tag{2.13}$$

In (2.13) the symbol R has the same meaning as in (2.12). The value of t_F is determined using (2.1). Obviously, it is very important to minimize the number of EMBs in the circuit of EMBer.

There are homogenous circuits corresponding to the diagram shown in Fig. 2.11. In the case of Moore FSM, it is possible to use the heterogeneous approach (Fig. 2.12).

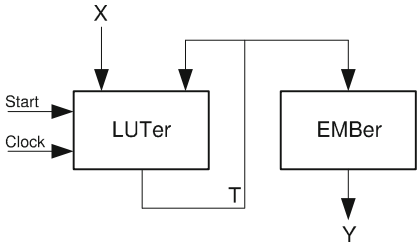


Fig. 2.12 Heterogeneous structure of P Moore FSM

In this circuit, the LUTer implements the system (1.4), whereas the EMBer the system (1.5). It allows implementing the final circuits using less amount of EMBs than in the case of completely EMB-based approach (Fig. 2.11).

Now, the following condition should be true:

$$2^{R_1} \cdot N \leq V_0. \tag{2.14}$$

In this case, it is enough only a single EMB in the EMBer.

For a heterogeneous P Moore FSM, there are the following columns in the table of EMBer: T (address of the cell) and Y (content of the cell). There is $R_1 = 3$ and $N = 4$ for Moore FSM $P(\Gamma_1)$. So it is enough to use an EMB having $V = 8 \times 4 = 32$ bits to implement the circuit of EMBer.

It is very important to diminish the number of LUTs in circuits of LUTers. One of the possible ways is a proper state assignment. Let us discuss this step in more details.

2.3 Methods of State Assignment

If an FSM is specified by a GSA, then such sets as X, Y and A are known. But there are no state codes. To obtain them, the step of state assignment is executed [57]. This step is very important because its outcome has a tremendous influence on the hardware amount (the number of LUTs) in the FSM logic circuit [1]. A strategy of state assignment could target optimization for area, performance, power consumption, or testability.

One of the most popular state assignment algorithms is JEDI which is distributed with the system SIS [73, 74]. JEDI targets a multi-level logic implementation. It is based on the weight assignment for states $a_m \in A$.

The input dominant algorithm assigns higher weights to pairs of present states which assert similar inputs and produce sets of next states. It allows maximizing the size of common cubes in the implemented logic function. The output dominated algorithm assigns higher weights to pairs of next states which are generated by similar input combinations and similar sets of present states. It maximizes the number of common cubes in the logic function. This method can be used for the optimal state assignment based on PES [8]. We discuss this approach a bit later.

In modern industrial packages different state assignment strategies are used. For example, two optimization criteria are used in the design tool XST of Xilinx: maximum performance and minimum hardware [90]. Seven different approaches are used for state assignment. The automatic state assignment is based on some special algorithm proposed by Xilinx. It has been never published. The method of one-hot encoding is based on the following expression:

$$R = M. \tag{2.15}$$

This method is very popular because: (1) it is very simple and (2) each LUT is connected with a flip-flop. So, this conception is implemented very easy in FPGAs. In this case, there is a lot of input memory functions but each of them is relatively small. The compact state assignment is based on the formula (1.6) for a Mealy FSM and the formula (1.14) for a Moore FSM. In this case the number of input memory functions is minimum possible, but they are rather complex. In this book we mostly use this approach and name it a binary state assignment. Two other methods are based on codes either Gray or Johnson. At last, there is so named speed encoding. When the performance is maximized and the sequential encoding based on using of the counter instead of state register.

The master thesis [82] is devoted to investigation of influence of the state assignment method on characteristics of Mealy FSM. The benchmarks from [91] are used in the investigation. The results obtained for Mealy FSM are represented in Table 2.5.

The investigations are executed for the FPGA XC5VLX30 of Xilinx. The first column of Table 2.6 shows the name of a benchmark. The columns "LUT" show numbers of LUTs in the final circuit. The columns "MHz" represent the maximal frequency of operation for final Mealy FSMs.

The best results are produced when the automatic state assignment is used. It gives the best outcomes for area (58.54 % of all benchmarks) and performance (39.02 %). The binary state assignment possesses the second place in this competition. As follows from Table 2.5, the automatic state assignment produces the best results when both area and performance are optimized (29.27 %). The same results are produced for the compact (binary) state assignment. It is interesting that the one-hot state assignment can optimize only one parameter of FSM circuit.

Of course, these results are true only for the chip XC5VLX30. But similar conclusions are made, for example, in [51]. It allows to suggest that these conclusions have a rather common nature.

As it was pointed before, it is very important to diminish the power dissipation in the circuit of FSM [77].

One of the approaches leading to decreasing the power dissipation in FSMs is the energy-saving state assignment [67]. Main works in low-power FSMs compute first the switching activity and transition probabilities [84]. The key idea of these methods is the reduction of the average activity by minimizing the bit changes during state transitions [16, 60]. The state assignment should minimize the Hamming distance between states with high transition probability. Different variations of this approach can be found in many works [3, 23, 32, 61]. There are hundreds of articles devoted to this approach.

Now, let us discuss an approach based on using classes of pseudoequivalent states in Moore FSMs. Remind, that states $a_m, a_s \in A$ belong to the same class of PES if corresponding vertices of a GSA are connected with an input of the same vertex of this GSA [5]. It allows constructing the partition $\Pi_A = \{B_1, \ldots, B_I\}$, where $B_i \in \Pi_A$ is a class of PES. After constructing the partition Π_A, an initial GSA Γ can be transformed into a block GSA $B(\Gamma)$.

Table 2.5 Outcomes of investigations

FSM	Auto		One-hot		Compact		Sequential		Gray		Johnson		Speed1	
	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz
bbara	11	639	9	966	13	635	13	639	19	589	24	545	13	962
bbsse	29	559	29	559	29	582	29	538	31	538	36	408	38	556
bbtas	5	962	8	966	5	966	5	962	5	955	5	962	9	966
beecount	7	952	19	639	7	952	7	952	7	948	21	625	30	583
ese	49	480	52	477	46	463	50	487	46	454	71	434	72	453
dk14	8	945	29	522	8	945	8	945	8	945	19	623	40	512
dk15	7	1062	19	737	7	1062	7	1062	7	1062	7	1062	19	659
dk16	46	556	46	556	15	625	19	506	27	554	86	355	70	399
dk17	6	952	14	669	6	952	6	952	6	952	7	895	27	571
dk27	5	900	8	906	5	897	5	959	5	955	6	899	10	903
dk512	17	730	17	730	7	899	7	895	7	899	21	437	19	790
ex1	64	586	64	586	74	447	67	478	66	406	106	340	72	605
ex4	15	962	15	962	16	626	15	598	14	748	33	546	15	962
ex6	29	553	30	580	20	621	23	615	22	616	36	426	31	598
keyb	56	384	56	384	65	358	71	382	66	447	62	435	85	374
kirkman	51	874	84	1058	53	569	48	880	51	874	112	451	84	1058
lion	3	1084	5	962	3	1080	3	1080	3	1084	3	1084	5	962
mark1	27	726	27	726	19	708	22	622	18	623	27	574	29	959
me	5	1071	8	1071	5	1071	6	1071	5	1071	5	1071	8	1071
opus	22	596	22	754	21	628	26	585	22	596	26	576	26	671
planet	100	888	100	888	138	389	145	417	149	375	192	346	106	637
planet1	100	888	100	888	138	389	145	417	149	375	192	346	106	637

(continued)

Table 2.5 (continued)

FSM	Auto		One-hot		Compact		Sequential		Gray		Johnson		Speed1	
	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz	LUT	MHz
pma	73	554	73	554	115	438	108	367	112	375	121	405	88	559
s1	77	550	77	550	75	447	89	328	105	368	114	361	81	552
s1488	140	425	140	425	141	432	130	394	147	433	192	334	162	458
s1494	124	412	124	412	143	442	135	383	145	383	192	333	152	462
s208	28	559	28	559	13	669	12	716	15	639	29	483	50	386
s27	4	962	21	636	4	962	7	679	4	962	12	664	21	631
s298	362	406	362	406	330	313	264	311	274	314	716	244	399	397
s386	26	577	31	586	28	581	28	558	29	429	43	22	36	441
s420	28	559	28	559	14	629	12	716	15	639	29	483	36	510
s510	42	900	42	900	39	448	53	440	50	427	123	388	42	900
s820	63	429	63	429	85	395	92	441	93	438	98	366	93	399
s832	63	429	63	429	73	412	77	431	87	394	97	335	108	444
sand	99	569	99	569	121	426	125	421	125	438	189	306	103	490
scf	179	676	179	676	202	338	205	349	197	389	337	327	180	561
shiftreg	0	1584	9	1080	0	1584	4	959	4	959	5	902	4	903
sse	29	559	29	559	28	543	37	548	32	540	44	394	36	612
styr	118	430	118	430	127	369	138	363	138	353	181	323	161	454
tav	6	1556	6	1556	6	911	6	911	5	914	5	914	6	1556
tbk	55	406	179	360	71	465	129	342	137	290	295	276	444	342

Fig. 2.13 State codes for Moore FSM $P_0(\Gamma_1)$

T_1 \ T_2T_3	00	01	11	10
0	a_1	a_3	a_4	a_2
1	a_6	$*$	a_5	$*$

Table 2.6 Reduced structure table of Moore FSM $P_0(\Gamma_1)$

B_i	$K(B_i)$	a_s	$K(a_s)$	X_h	Φ_h	h
B_1	$*00$	a_2	010	1	D_2	1
		a_3	001	x_1	D_3	2
B_2	$*10$	a_4	011	$\bar{x_1}x_2$	D_2D_3	3
		a_5	111	$\bar{x_1}\bar{x_2}$	$D_1D_2D_3$	4
B_3	$**1$	a_6	100	1	D_1	5

Let us encode states $a_m \in A$ in such a way that each class $B_i \in \Pi_A$ be represented by the minimal possible amount of generalized intervals of R_1-dimensional Boolean space. Let us name such an approach "optimal state assignment". For example, there is the partition $\Pi_A = \{B_1, \ldots, B_4\}$ in the case of Moore FSM represented by GSA Γ_1 Fig. 1.5b. It corresponds to the BGSA $B(\Gamma_1)$ shown in Fig. 1.7.

Let us point out that the JEDI algorithm [73] can be used for the optimal state assignment. Let us use the symbol $P_0(\Gamma_j)$ to show that a Moore FSM is synthesized using a GSA Γ_j and the principle of optimal state assignment. One of the possible outcomes is shown in Fig. 2.13 for the Moore FSM $P_0(\Gamma_1)$.

Analysis of $B(\Gamma_1)$ shows that there is only an unconditional transition from state $a_6 \in B_4$ into state a_1. There are no functions $D_r = 1$ ($r = \overline{1, R_1}$) during this transition. It means that $K(a_6)$ can be treated as a "don't care" state code.

Taking it into account, the following generalized intervals $K(B_i)$ can be found from the Karnaugh map (Fig. 2.13): $K(B_1) = *00$, $K(B_2) = *10$, $K(B_3) = **1$. These cubes can be treated as codes of classes $B_i \in \Pi_A$. Using these codes and BGSA $B(\Gamma_1)$, the reduced structure table of Moore FSM $P_0(\Gamma_1)$ can be constructed (Table 2.6).

We hope that there is obvious connection among BGSA $B(\Gamma_1)$, the state codes (Fig. 2.13) and Table 2.6. There are the following terms F_h in the case of this table: $F_1 = \bar{T_2}\bar{T_3}$; $F_2 = T_2\bar{T_3}x_1$; $F_3 = T_2\bar{T_3}\bar{x_1}x_2$; $F_4 = T_2\bar{T_3}\bar{x_1}\bar{x_2}$; $F_5 = T_3$.

It allows finding the following system of input memory functions (after minimization):

$$\begin{aligned} D_1 &= T_2\bar{T_3}\bar{x_1}\bar{x_2} \vee T_3; \\ D_2 &= \bar{T_2}\bar{T_3} \vee T_2\bar{T_3}\bar{x_1}; \\ D_3 &= T_2\bar{T_3}. \end{aligned} \tag{2.16}$$

Each function of the system (2.16) is implemented using only a single LUT having $S \leq 4$.

Fig. 2.14 State codes for Moore FSM S_1

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

In the common case, the terms F_h are represented as:

$$F_h = \bigwedge_{r=1}^{R_1} T_r^{e_{ir}} \cdot X_h \quad (h = \overline{1, H_0}). \tag{2.17}$$

In (2.17), $e_{ir} \in \{0, 1, *\}$ is the value of the r-th bit of $K(B_i)$ from the h-th row of a reduced structure table, $T_r^0 = \bar{T}_r$, $T_r^1 = T_r$, $T_r^* = 1$.

It follows from comparison of Tables 1.1 and 2.6 that there is the same amount of rows in both of them. It is true if each class $B_i \in \Pi_A$ is represented by a single cube. Because of it, the symbol H_0 is used in (2.17).

Let us point out that sometimes it is necessary more than one cube to represent some class $B_i \in \Pi_A$. For example, let it be the partition $\Pi_A = \{B_1, \ldots, B_4\}$ for some FSM S_1. Let it be $B_1 = \{a_1\}$, $B_2 = \{a_2, a_3, a_4\}$, $B_3 = \{a_5, a_6, a_7\}$, $B_4 = \{a_8\}$. There is one of the possible outcomes of the optimal state assignment for S_1 shown in Fig. 2.14.

Let us treat as "don't care" the code of the state $a_8 \in A$. Even in this case, the class B_2 is represented by two cubes: $00*$ and $01*$. If there are conditional transitions from the state $a_8 \in B_4$, its code cannot be treated as "don't care". In this case, two classes are represented using two cubes each. The class B_3 is represented by the cubes 11 and $11*$. In this case, there are more than H_0 rows in the reduced structure table. We deal with this case a bit later.

So, the existed methods of state assignment can be divided using the logic elements they target. For example, there are PLA-oriented methods [4, 19, 20, 27, 56, 68–70, 86, 87], as well as PAL-oriented methods [9–14, 24–26, 28, 44–50, 78–80]. A lot of methods deals with FPGA-based FSMs [29–31, 33, 38, 39, 63]. There are a lot of state assignment methods targeting saving the energy consumed by the FSM circuit [3, 40, 62, 85, 88], just to name a few.

Our book is devoted to FPGA-based FSMs. The main goal of all discussed methods is to save the chip area occupied by an FSM circuit. So, we use JEDI-based algorithms for executing the state assignment.

2.4 Hardware Reduction for FPGA-Based FSMs

As it is mentioned in many works, there are the positive back effects of hardware reduction. The solution of this problem leads to increasing of performance and decreasing of power consumption of FSM logic circuits [35, 92]. To decrease

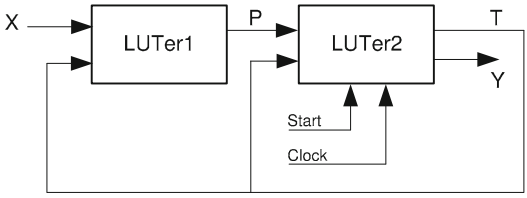


Fig. 2.15 Structural diagram of LUT-based MP Mealy FSM

the amount of hardware, we use three groups of methods: (1) structural decomposition; (2) heterogeneous implementation and (3) optimal encoding of states and other objects of FSMs.

We discussed the method of replacement of logical conditions (RLC) in Sect. 1.5. Let us start with LUT-based MP Mealy FSM (Fig. 2.15).

In this circuit, the LUTer1 is equivalent to BRLC from Fig. 1.21. It implements the system (1.47). The LUTer2 executes functions of BIMF from Fig. 1.21. It generates microoperations (1.48) and input memory functions (1.49). The register RG is distributed among LUTs of LUTer2, so, there are state variables $T_r \in T$ shown as the outputs of LUTer2 (Fig. 2.15).

There are the following steps in design method for MP Mealy FSM:

1. Constructing the set of states for a GSA Γ.
2. Executing the state assignment.
3. Constructing the tables of LUTer1.
4. Constructing the structure table of MP Mealy FSM.
5. Constructing the tables of LUTer2.
6. Implementing the FSM logic circuit.

Let us discuss an example of design for Mealy FSM $MP(\Gamma_4)$, where a GSA Γ_4 is shown in Fig. 2.16. There are marks of states on GSA Γ_4. So, it can be found the set $A = \{a_1, a_2, a_3\}$ with $M_0 = 3$. It is enough $R_0 = 2$ of state variables to encode the states $a_m \in A$. Let us encode them in the following way: $K(a_1) = 00$, $K(a_2) = 01$ and $K(a_3) = 10$.

To construct the table of LUTer1 it is necessary to execute the RLC. There are the following sets $X(a_m)$ in the discussed case: $X(a_1) = \{x_1, x_2\}$, $X(a_2) = \{x_3, x_4\}$ and $X(a_3) = \emptyset$. So, there is $G = 2$ and $P = \{p_1, p_2\}$.

Table 2.7 represents the table of replacement of logical conditions for $MP(\Gamma_4)$. It includes the columns with states, their codes and logical conditions replaced by corresponding variables $p_g \in P$.

There are G circuits in LUTer1. Each of them corresponds to a single function $p_g \in P$. Let $X(p_g) \subseteq X$ be a set of logical conditions replaced by the variable $p_g \in P$. There are G tables representing functions $p_g \in P$. Each table includes the columns $K(a_m)$, $X(p_g)$, p_g. First two columns create the address of the cell, the last column shows its content. There are the following functions derived from Table 2.7.

$$\begin{aligned} p_1 &= \bar{T}_1 \bar{T}_2 x_1 \vee \bar{T}_1 T_2 x_3; \\ p_2 &= \bar{T}_1 \bar{T}_2 x_2 \vee \bar{T}_1 T_2 x_4. \end{aligned} \tag{2.18}$$

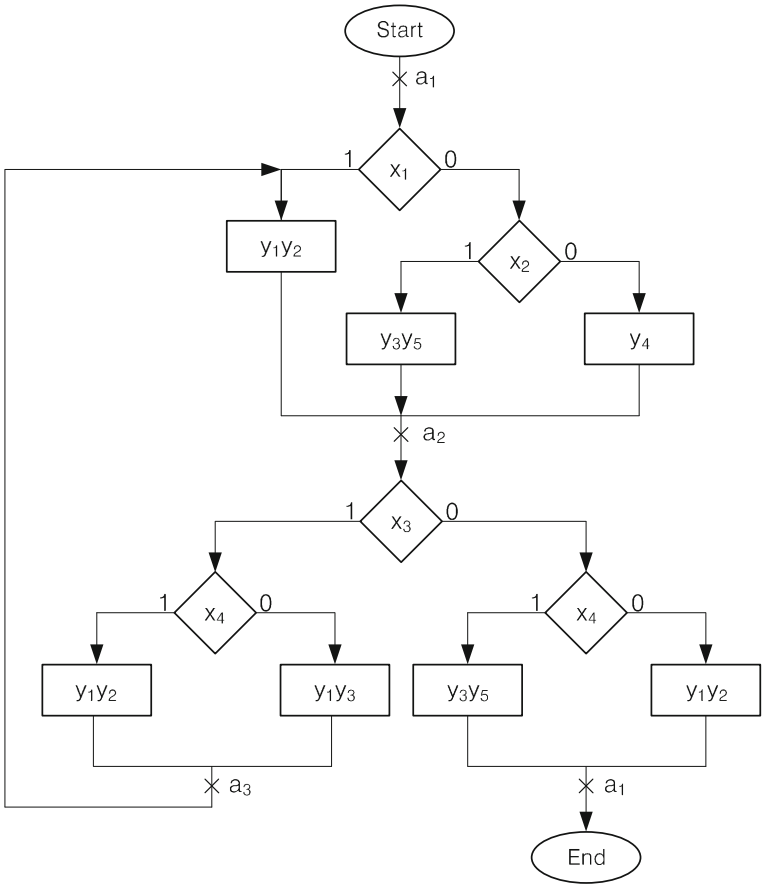


Fig. 2.16 Initial GSA Γ_4

Table 2.7 Table of RLC for Mealy FSM $MP(\Gamma_4)$

a_m	a_1	a_2	a_3
$K(a_m)$	00	01	10
p_1	x_1	x_3	–
p_2	x_2	x_4	–

Table 2.8 represents the function p_1, whereas Table 2.9 the function p_2 from the system (2.18). There are the following sets $X(p_g)$ used in these tables: $X(p_1) = \{x_1, x_3\}$ and $X(p_2) = \{x_2, x_4\}$. There is the obvious connection between system (2.18) and Tables 2.8 and 2.9.

The structure table of MP Mealy FSM includes the same columns as in the case of P Mealy FSM. But the column X_h is replaced by a column P_h. For example, the conjunction x_1x_2 (it determines the transition $\langle a_1, a_2 \rangle$) is replaced by p_1p_2, the

Table 2.8 Table of LUTer1 for function p_1

T	$X(p_1)$	p_1	T	$X(p_1)$	p_1	T	$X(p_1)$	p_1	T	$X(p_1)$	p_1
T_1T_2	x_1x_3		T_1T_2	x_1x_3		T_1T_2	x_1x_3		T_1T_2	x_1x_3	
00	00	0	01	00	0	10	00	0	11	00	0
00	01	0	01	01	1	10	01	0	11	01	0
00	10	1	01	10	0	10	10	0	11	10	0
00	11	1	01	11	1	10	11	0	11	11	0

Table 2.9 Table of LUTer1 for function p_2

T	$X(p_2)$	p_2	T	$X(p_2)$	p_2	T	$X(p_2)$	p_2	T	$X(p_2)$	p_2
T_1T_2	x_2x_4		T_1T_2	x_2x_4		T_1T_2	x_2x_4		T_1T_2	x_2x_4	
00	00	0	01	00	0	10	00	0	11	00	0
00	01	0	01	01	1	10	01	0	11	01	0
00	10	1	01	10	0	10	10	0	11	10	0
00	11	1	01	11	1	10	11	0	11	11	0

Table 2.10 Structure table of Mealy FSM $MP(\Gamma_4)$

a_m	$K(a_m)$	a_s	$K(a_s)$	P_h	Y_h	Φ_h	h
a_1	00	a_2	01	p_1	y_1y_2	D_2	1
		a_2	01	$\bar{p}_1p_2$	y_3y_5	D_2	2
		a_2	01	$\bar{p}_1p_2$	y_4	D_2	3
a_2	10	a_3	10	p_1p_2	y_1y_2	D_1	4
		a_3	10	$p_1\bar{p}_2$	y_1y_3	D_1	5
		a_1	00	$\bar{p}_1p_2$	y_3y_5	–	6
		a_1	00	$\bar{p}_1\bar{p}_2$	y_1y_2	–	7
a_3	10	a_2	01	1	y_1y_2	D_2	8

conjunction $x_1\bar{x_2}$ by $p_1\bar{p}_2$ and so on. Using these rules and table of RLC, it is possible to construct the table of MP Mealy FSM. It is Table 2.10 in the discussed case.

Similar tables are used for constructing tables of LUTer2. There are $R_0 + N$ tables representing LUTer2. Each of them includes columns $K(a_m)$, P (address of the cell) and either $y_n \in Y$ or $D_r \in \Phi$. All these tables are represented by Table 2.11.

The address 0000 corresponds to the row 3 of Table 2.10. So, the corresponding cell contains two ones (y_4 and D_2). The addresses 0010 and 0011 correspond to the row 1 of Table 2.10. Using the same approach, we fill all rows of Table 2.11.

It is possible to implement systems (1.48) and (1.49) by EMBer. It leads to heterogeneous MP Mealy FSM (Fig. 2.17).

In this case, the LUTer implements the system (1.47), whereas the EMBer the systems (1.48) and (1.49). There are the same steps in design of both MP FSMs, but in the heterogeneous case the step 5 is "Constructing the table of EMBer". In the discussed case, table of EMBer is the same as Table 2.11.

Table 2.11 Table of LUTer2 for Mealy FSM $MP(\Gamma_4)$

T	P	y_1	y_2	y_3	y_4	y_5	D_1	D_2	T	P	y_1	y_2	y_3	y_4	y_5	D_1	D_2
T_1T_2	p_1p_2								T_1T_2	p_1p_2							
00	00	0	0	0	1	0	0	1	10	00	1	1	0	0	0	0	1
00	01	0	0	1	0	1	0	1	10	01	1	1	0	0	0	0	1
00	10	1	1	0	0	0	0	1	10	10	1	1	0	0	0	0	1
00	11	1	1	0	0	0	0	1	10	11	1	1	0	0	0	0	1
01	00	1	1	0	0	0	0	0	11	00	0	0	0	0	0	0	0
01	01	0	0	1	0	1	0	0	11	01	0	0	0	0	0	0	0
01	10	1	0	1	0	0	1	0	11	10	0	0	0	0	0	0	0
01	11	1	1	0	0	0	0	1	11	11	0	0	0	0	0	0	0

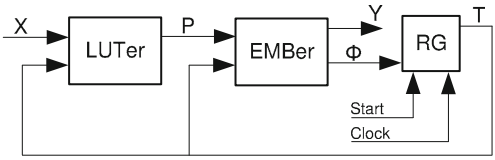


Fig. 2.17 Heterogeneous structure of MP Mealy FSM

Let the following condition take place:

$$R_0 + |X(p_g)| \leq S \quad (g = \overline{1, G}). \tag{2.19}$$

In this case, each function $p_g \in P$ is implemented using a single LUT. If this condition is violated, then it is possible to encode logical conditions [15].

Let us explain this approach on the base of Table 2.7. There are two pairs of conditions in the columns of this table: $\langle x_1, x_2 \rangle$ and $\langle x_3, x_4 \rangle$. Let us denote this number as n_P. Let us encode each pair using R_P variables, where:

$$R_P = \lceil \log_2 n_P \rceil. \tag{2.20}$$

Let us use the variables $z_r \in Z$ for such encoding where $|Z| = R_P$.

In the discussed case, there are $n_P = 2$, $R_P = 1$ and $Z = \{z_1\}$. Let us encode the pairs in the following way: $K(x_1, x_2) = 0$, $K(x_3, x_4) = 1$. Now, the system (1.47) can be replaced by the following system:

$$P = P(Z, X). \tag{2.21}$$

It leads to $M_P P$ Mealy FSM (Fig. 2.18).

In this case, the EMBer implements systems (1.48) and (1.49) and the system:

$$Z = Z(T, P). \tag{2.22}$$

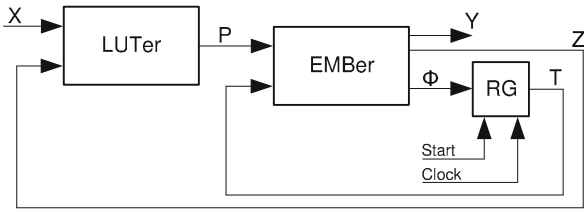


Fig. 2.18 Structural diagram of $M_P P$ Mealy FSM

Table 2.12 Table of RLC for Mealy FSM $M_P P(\Gamma_4)$

$\langle x_e, x_m \rangle$	$\langle x_1, x_2 \rangle$	$\langle x_3, x_4 \rangle$
$K(x_e, x_m)$	0	1
p_1	x_1	x_3
p_2	x_2	x_4

There is the following table of RLC in the case of $M_P P(\Gamma_1)$ (Table 2.12).
Now, there are the following functions implemented by the LUTer:

$$\begin{aligned} p_1 &= \bar{z_1} x_1 \vee z_1 x_3; \\ p_2 &= \bar{z_1} x_2 \vee z_1 x_4. \end{aligned} \tag{2.23}$$

Each function (2.21) is implemented by a single LUT if the following condition takes place:

$$R_P + |X(p_g)| \leq S. \tag{2.24}$$

Of course, this approach has sense if there is

$$R_P < R_0. \tag{2.25}$$

Let us discuss the MP Moore FSM (Fig. 2.19) implemented as a heterogeneous circuit. Here the LUTer1 implements the system (1.47), the LUTer2 the system (1.49) and the EMBer the system (1.5).

To minimize the circuit of LUTer1, it is possible to apply the encoding of logical conditions. It leads to $M_P P$ Moore FSM (Fig. 2.20).

This approach is a general one. It can be used in any model of FSM to diminish the hardware amount. It uses two main issues: (1) the structural decomposition and (2) the heterogeneous implementation.

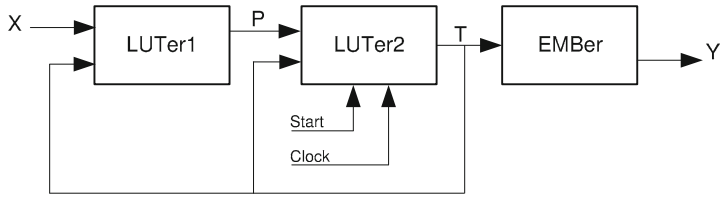


Fig. 2.19 Structural diagram of MP Moore FSM

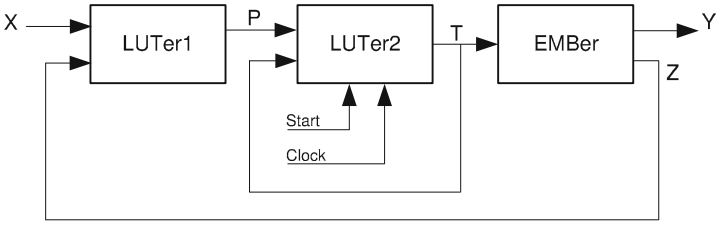


Fig. 2.20 Structural diagram of $M_P P$ Moore FSM

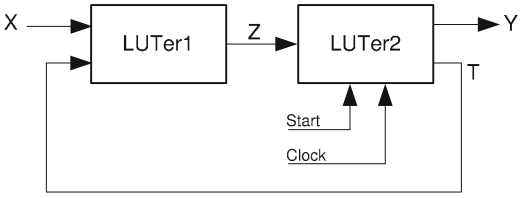


Fig. 2.21 Homogeneous structural diagram of PH Mealy FSM

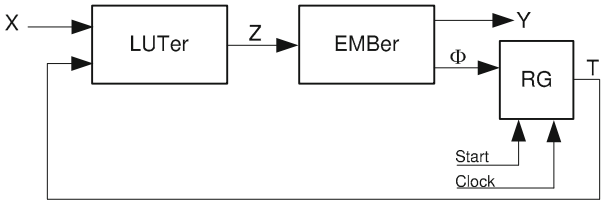


Fig. 2.22 Heterogeneous structural diagram of PH Mealy FSM

The second approach is based on the encoding of the rows of ST. It can be used only for Mealy FSMs. It is reduced to encoding of each function F_h by a binary code $C(F_h)$ having R_H bits ($h = \overline{1, H_0}$). The value of R_H is determined by (1.38). It is possible either homogeneous (Fig. 2.21) or heterogeneous (Fig. 2.22) structures of PH Mealy FSMs.

In both cases, the first block corresponds to BIMF (Fig. 1.18d), whereas the second to BMO (Fig. 1.18d). For example, the design method for the homogeneous variant includes the following steps:

1. Constructing the set of states for a GSA Γ.
2. Executing the state assignment.
3. Constructing the structure table of P Mealy FSM.
4. Encoding the rows of ST.
5. Constructing the table of LUTer1.
6. Constructing the table of LUTer2.
7. Implementing the FSM logic circuit.

Table 2.13 Structure table of Mealy FSM $P(\Gamma_4)$

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Y_h	Φ_h	h
a_1	00	a_2	01	x_1	$y_1 y_2$	D_2	1
		a_2	01	$\bar{x_1} x_2$	$y_3 y_5$	D_2	2
		a_2	01	$\bar{x_1} \bar{x_2}$	y_4	D_2	3
a_2	01	a_3	10	$x_3 x_4$	$y_1 y_2$	D_1	4
		a_3	10	$x_3 \bar{x_4}$	$y_1 y_3$	D_1	5
		a_1	00	$\bar{x_3} x_4$	$y_3 y_5$	–	6
		a_1	00	$\bar{x_3} \bar{x_4}$	$y_1 y_2$	–	7
a_3	10	a_2	01	1	$y_1 y_2$	D_2	8

Let us discuss an example of design for Mealy FSM $PH(\Gamma_4)$. There is the GSA Γ_4 shown in Fig. 2.16. As in the previous case, the following sets can be found for $P(\Gamma_4)$: $X = \{x_1, \ldots, x_4\}$, $Y = \{y_1, \ldots, y_5\}$, $A = \{a_1, a_2, a_3\}$, $\Phi = \{D_1, D_2\}$, $T = \{T_1, T_2\}$. Let us encode the states in the following way: $K(a_1) = 00$, $K(a_2) = 01$, $K(a_3) = 10$. There are $H_0 = 8$ rows in the structure table of FSM $P(\Gamma_4)$ (Table 2.13).

Let the following condition take place:

$$R_H \leq S. \tag{2.26}$$

In this case, there is no influence of codes $C(F_h)$ on the hardware amount in LUTer2. It is implemented using $R + N$ of LUTs. If this condition is violated, then the outcome of row encoding can influence the hardware amount in LUTer2.

In the discussed case, there are $H_0 = 8$ and $R_H = 3$. So, there is the set $Z = \{z_1, z_2, z_3\}$. Let us use LUTs having $S = 2$. In this case, the condition (2.26) is violated. There are the following system of Boolean functions derived from Table 2.13:

$$\begin{aligned} D_1 &= F_4 \vee F_5; \\ D_2 &= F_1 \vee F_2 \vee F_3 \vee F_8; \\ y_1 &= F_1 \vee F_4 \vee F_5 \vee F_7 \vee F_8; \\ y_2 &= F_1 \vee F_4 \vee F_7 \vee F_8; \\ y_3 &= F_2 \vee F_5 \vee F_6; \\ y_4 &= F_3; \\ y_5 &= F_2 \vee F_6. \end{aligned} \tag{2.27}$$

In the worst case, each equation of (2.27) requires two LUTs to be implemented. Let us encode the rows as it is shown in Fig. 2.23.

Fig. 2.23 Codes of rows for FSM $PH(\Gamma_4)$

Z_1 \ Z_2Z_3	00	01	11	10
0	F_1	F_4	F_7	F_8
1	F_3	F_5	F_6	F_2

The following equations can be obtained from the Karnaugh map (Fig. 2.23):

$$\begin{aligned} D_1 &= \bar{z_2} z_3; \\ D_2 &= \bar{z_3}; \\ y_1 &= \bar{z_1} \vee \bar{z_2} z_3; \\ y_2 &= \bar{z_1}; \\ y_3 &= z_1 z_3 \vee z_1 z_2; \\ y_4 &= z_1 \bar{z_2} \bar{z_3}; \\ y_5 &= z_1 z_2. \end{aligned} \tag{2.28}$$

The system (2.28) corresponds to the following circuit of LUTer2 (Fig. 2.24).

In the worst case, it is necessary 14 of LUTs, whereas there are only 10 of LUTs in Fig. 2.24. This number can be decreased if there are the following codes of rows (Fig. 2.25).

Using this Karnaugh map, we can find the following system of functions:

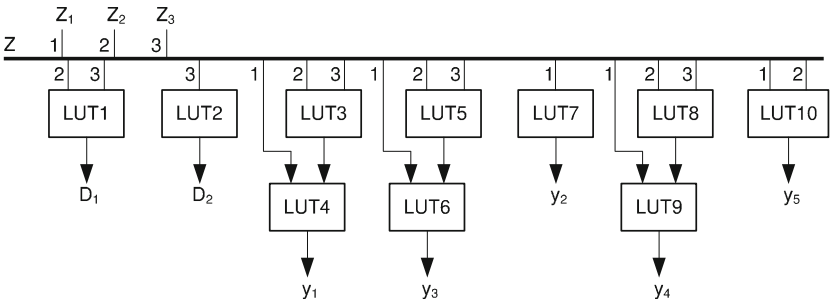


Fig. 2.24 Logic circuit of LUTer2 for codes from Fig. 2.23

Fig. 2.25 Outcome of row encoding for $PH(\Gamma_1)$

z_1 \ z_2z_3	00	01	11	10
0	F_5	F_3	F_2	F_6
1	F_4	F_1	F_8	F_7

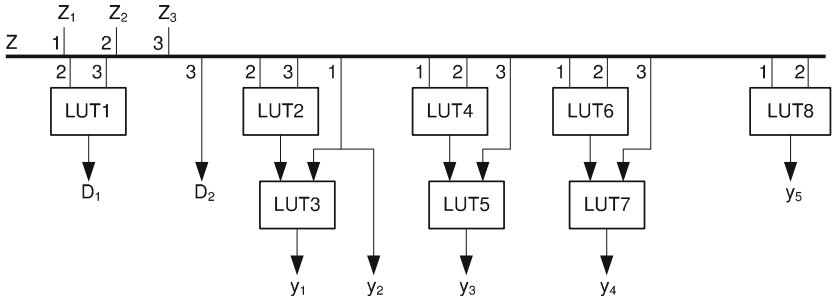


Fig. 2.26 Logic circuit of LUTer2 corresponding to codes from Fig. 2.25

$$
\begin{aligned}
D_1 &= \bar{z_2}\bar{z_3};\\
D_2 &= z_3;\\
y_1 &= z_1 \vee \bar{z_2}\bar{z_3};\\
y_2 &= z_1;\\
y_3 &= \bar{z_1}\bar{z_3} \vee \bar{z_1}z_2;\\
y_4 &= z_1\bar{z_2}z_3;\\
y_5 &= \bar{z_1}z_2.
\end{aligned}
\tag{2.29}
$$

There is the circuit (Fig. 2.26) corresponding to the system (2.29). It requires only 8 of LUTs having $S = 2$.

The table of LUTer1 is constructed on the base of the structure table. The columns a_s, $K(a_s)$, Y_h and Φ_h are deleted from the ST. They are replaced by columns $C(F_h)$, Z_h. There are the variables $z_r \in Z$ in the row h of the table, if there is 1 in the position number r of $K(F_h)$. In the discussed case, it is the Table 2.14.

The table of LUTer1 is constructed using the codes from Fig. 2.25. It is used for deriving the system

$$Z = Z(T, X). \tag{2.30}$$

Table 2.14 Table of LUTer1 for Mealy FSM $PH(\Gamma_4)$

a_m	$K(a_m)$	X_h	$C(F_h)$	Z_h	h
a_1	00	x_1	101	z_1z_2	1
		$\bar{x_1}x_2$	011	z_2z_3	2
		$\bar{x_1}\bar{x_2}$	001	z_3	3
a_2	01	x_3x_4	100	z_1	4
		$x_3\bar{x_4}$	000	–	5
		$\bar{x_3}x_4$	010	z_2	6
		$\bar{x_3}\bar{x_4}$	110	z_1z_2	7
a_3	10	1	111	$z_1z_2z_3$	8

Table 2.15 Table of LUTer2 for Mealy FSM $PH(\Gamma_4)$

$C(F_h)$	Y_h	Φ_h	h
$z_1z_2z_3$	$y_1y_2y_3y_4y_5$	D_1D_2	
000	10100	10	5
001	00010	01	3
010	00101	00	6
011	00101	01	2
100	11000	10	4
101	11000	01	1
110	11000	00	7
111	11000	01	8

It is the following system in the discussed case:

$$\begin{aligned} z_1 &= \bar{T}_1\bar{T}_2x_1 \vee \bar{T}_1T_2x_3x_4 \vee \bar{T}_1T_2\bar{x}_3\bar{x}_4 \vee T_1\bar{T}_2; \\ z_2 &= \bar{T}_1\bar{T}_2x_1 \vee \bar{T}_1\bar{T}_2x_2 \vee \bar{T}_1T_2\bar{x}_3 \vee T_1\bar{T}_2; \\ z_3 &= \bar{T}_1\bar{T}_2\bar{x}_1 \vee T_1\bar{T}_2. \end{aligned} \tag{2.31}$$

The table of LUTer2 is constructed on the base of the ST. The columns a_m, $K(a_m)$, X_h are replaced by columns $C(F_h)$. In the discussed case, it is Table 2.15.

This table is used to find the systems (1.39) and (1.40). They are represented by the system (2.28) in the discussed case.

To design this circuit, we used optimal codes for variables F_h. This example shows the importance of this step and its influence on the hardware amount in LUT-based design.

There are three groups of methods discussed in the next chapters of the book:

1. EMB-based FSMs with RLC and other methods of structural decomposition. There are two main goals in our approach. First, decreasing for the number of EMBs in the FSM logic circuit. Second, decreasing the number of LUTs in the circuit replacing logical conditions.

2. EMB-based FSMs with transformation of object codes. These models are based on transformation of state codes into codes of collections of microoperations and vice versa.

3. EMB-based FSMs with more than one source of codes of classes of pseudoequivalent states. These methods concern only Moore FSMs.

References

1. M. Adamski, A. Barkalov, *Architectural and Sequential Synthesis of Digital Devices* (University of Zielona Góra Press, Zielona Góra, 2006)
2. Altera: http://www.altera.com. Accessed Jan 2015

3. P. Bacchetta, L. Daldos, D. Sciuto, C. Silvano, Low-power state assignment techniques for finite state machines, in *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems (ISCAS'2000)*, vol. 2, Geneva. IEEE (2000), pp. 641–644
4. S. Baranov, *Logic Synthesis of Control Automata* (Kluwer Academic Publishers, Dordrecht, 1994)
5. A. Barkalov, Principles of logic optimization for Moore microprogram automaton. Cybern. Syst. Anal. **34**(1), 54–60 (1998)
6. A. Barkalov, A. Barkalov Jr., Synthesis of finite-state machines with transformation of the object's codes, in *Proceedings of the International Conference TCSET'2004*, Lviv, Ukraina (2004), pp. 61–64. Lviv Polytechnic National University, Lviv, Publishing House of Lviv Polytechnic
7. A. Barkalov, A. Barkalov Jr, Design of mealy finite-state machines with the transformation of object codes. Int. J. Appl. Math. Comput. Sci. **15**(1), 151–158 (2005)
8. A. Barkalov, L. Titarenko, *Logic Synthesis for FSM-Based Control Units*. Lecture Notes in Electrical Engineering, vol. 53 (Springer, Berlin, 2009)
9. A. Barkalov, L. Titarenko, S. Chmielewski, Optimization of logic circuit of Moore FSM on CPLD. Pomiary Automatyka Kontrola **53**(5), 18–20 (2007)
10. A. Barkalov, L. Titarenko, S. Chmielewski, Optimization of Moore FSM on CPLD, in *Proceedings of the Sixth International Conference CAD DD'07*, vol. 2, Minsk (2007), pp. 39–45
11. A. Barkalov, L. Titarenko, S. Chmielewski, Optimization of Moore FSM on system-on chip, in *Proceedings of IEEE East-West Design and Test Symposium—EWDTS'07*, Yerevan, Armenia, Kharkov (2007), pp. 105–109
12. A. Barkalov, L. Titarenko, S. Chmielewski, Reduction in the number of PAL macrocells int the circuit of a Moore FSM. Int. J. Appl. Math. Comput. Sci. **17**(4), 565–675 (2007)
13. A. Barkalov, L. Titarenko, S. Chmielewski, Decrease of hardware amount in logic circuit of Moore FSM. Przegląd Telekomunikacyjny i Wiadomości Telokomunikacyjne (6), 750–752 (2008)
14. A. Barkalov, L. Titarenko, S. Chmielewski, Optimization of Moore FSM on System-on-chip using PAL technology, in *Proceedings of the International Conference TCSET 2008*, Lviv-Slavsko, Ukraina (2008), pp. 314–317. Ministry of Education and Science of Ukraine, Lviv Polytechnic National University, Lviv, Publishing House of Lviv Polytechnic
15. A. Barkalov, I. Zelenjova, Optimization of replacement of logical conditions for an automaton with bidirectional transitions. Autom. Control Comput. Sci. **34**(5), 48–53. Allerton Press Inc
16. L. Benini, G. De Micheli, State assignment for low power dissipation. IEEE J. Solid-State Circuits **30**(3), 258–268 (1995)
17. M. Bolton, *Digital System Design with Programmable Logic* (Addison-Wesley, Boston, 1990)
18. G. Borowik, Synthesis of sequential devices into FPGA with embedded memory blocks. Ph.D. thesis, Warszawa: WUT (2007). Ph.D. Thesis
19. R. Brayton, G. Hatchel, C. McMullen, A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis* (Kluwer Academic Publishers, Boston, 1984)
20. R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, MIS: a multi-level logic optimization system. IEEE Trans. Comput. -Aided Des. **6**(11), 1062–1081 (1987)
21. S. Brown, Z. Vernesic, *Fundamentals of Digital Logic with VHDL Design* (McGraw–Hill, New York, 2000)
22. S. Chattopadhyay, P. Chaudhuri, Genetic algorithm based approach for integrated state assignment and flipflop selection in finite state machines synthesis, in *Proceedings of the IEEE International Conference on VLSI Design*, Los Alamitos. IEEE Computer Society (1998), pp. 522–527
23. C. Chen, J. Zhao, M. Ahmadi, A semi-gray encoding algorithm for low-power state assignment, in *Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS'03*, vol. 5. IEEE (2003), pp. 389–392
24. R. Czerwiński, D. Kania, State assignment method for high speed FSM, in *Proceedings of Programmable Devices and Systems* (2004), pp. 216–221

25. R. Czerwiński, D. Kania, State assignment for PAL-based CPLDs, in *Proceedings of 8th Euromicro Symposium on Digital System Design* (2005), pp. 127–134
26. R. Czerwinski, D. Kania, *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*. Lecture Notes in Electrical Engineering, vol. 23 (Springer, Berlin, 2013)
27. D. Debnath, T. Sasao, Doutput phase optimization for AND-OR-EXOR PLAs with decoders and its application to design of adders. IFICE Trans. Inf. Syst. **E88-D**(7), 1492–1500 (2005)
28. S. Deniziak, K. Sapiecha, 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)* (1998), pp. 47–53
29. S. Devadas, H. Ma, A. Newton, A. Sangiovanni-Vincentelli, MUSTANG: state assignment of finite state machines targeting multilevel logic implementation. IEEE Trans. Comput.-Aided Des. **7**(12), 1290–1300 (1988)
30. S. Devadas, A. Newton, Exact algorithms for output encoding, state assignment, and four-level boolean minimization. IEEE Trans. Comput.-Aided Des. **10**(1), 143–154 (1991)
31. X. Du, G. Hachtel, B. Lin, A. Newton, MUSE: a multilevel symbolic encoding algorithm for state assignment. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **10**(1), 28–38 (1991)
32. A. El-Maleh, S. Sait, F. Khan, Finite state machine state assignment for area and power minimization, in *2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006. Proceedings*. IEEE (2006), pp. 5303–5306
33. B. Escherman, State assignment for hardwired VLSI control units. ACM Comput. Surv. **25**(4), 415–436 (1993)
34. I. Garcia-Vargas, R. Senhadji-Navarro, G. Jiménez-Moreno, A. Civit-Balcells, P. Guerra-Gutierrez, ROM-based finite state machine implementation in low cost FPGAs, in *IEEE International Symposium on Industrial Electronics ISIE 2007*. IEEE (2007), pp. 2342–2347
35. I. Grout, *Digital Systems Design with FPGAs and CPLDs* (Elsevier Science, Oxford, 2008)
36. S. Hassoun, T. Sasao, *Logic Synthesis and Verification* (Kluwer Academic Publishers, Boston, 2002)
37. G. Hatchel, F. Somenzi, *Logic synthesis and verification algorithms* (Kluwer Academic Publishers, Boston, 2000)
38. H. Hu, H. Xue, J. Bian, A heuristic state assignment algorithm targeting area, in *Proceedings of 5th International Conference on ASIC*, vol. 1 (2003), pp. 93–96
39. J. Huang, J. Jou, W. Shen, ALTO: an iterative area/performance algorithms for LUT-based FPGA technology mapping. IEEE Trans. VLSI Syst. **18**(4), 392–400 (2000)
40. A. Iranli, P. Rezvani, M. Pedram, Low power synthesis of finite state machines with mixed D and T flip-flops, in *Proceedings of the Asia and South Pacific—DAC* (2003), pp. 803–808
41. J. Jenkins, *Design with FPGAs and CPLDs* (Prentice Hall, New York, 1995)
42. A. Kahng, *VLSI Physical Design: From Graph Partitioning to Timing Closure* (Springer, Berlin, 2011)
43. T. Kam, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli, *A Synthesis of Finie State Machines: Functional Optimization* (Kluwer Academic Publishers, Boston, 1998)
44. D. Kania, Two-level logic synthesis on PAL-based CPLD and FPGA using decomposition, in *Proceedings of 25th Euromicro Conference* (1999), pp. 278–281
45. D. Kania, Two-level logic synthesis on PALs. Electron. Lett. (17), 879–880 (1999)
46. D. Kania, Coding capacity of PAL-based logic blocks included in CPLDs and FPGAs, in *Proceedings of IFAC Workshop on Programmable Devices and Sysytems (PDS'2000)*. Elsevier Science (2000), pp. 164–169
47. D. Kania, Decomposition-based synthesis and its application in PAL-oriented technology mapping, in *Proceedings of 26th Euromicro Conference*. Maastricht: IEEE Compuetr Society Press (2000), pp. 138–145
48. D. Kania, An efficient algorithm for output coding in PAL-based CPLDs. Int. J. Eng. **15**(4), 325–328 (2002)
49. D. Kania, Logic synthesis of multi–output functions for PAL-based CPLDs, in *Proceedings of IEEE International Conference on Field-Programmable Technology* (2002), pp. 429–432

50. D. Kania, An efficient approach to synthesis of multi-output boolean functions on PAL-based devices. IEEE Proc.—Comput. Digital Tech. **150**, 143–149 (2003)
51. H. Kubatova, *Design of Embedded Control Systems*, chapter Finie State Machine Implementation in FPGAs (Springer, New York, 2005), pp. 177–187
52. Lattice: http://www.lattice.com. Accessed Feb 2015
53. C. Maxfield, *The Design Warrior's Guide to FPGAs* (Academic Press Inc., Orlando, 2004)
54. C. Maxfield, *FPGAs: Instant Access* (Newnes, 2008)
55. E. McCluskey, *Logic Design Principles* (Prentice Hall, Englewood Cliffs, 1986)
56. G. De Micheli, Symbolic design of combinational and sequential logic implemented by two-level macros. IEEE Trans. Comput.-Aided Des. **5**(9), 597–616 (1986)
57. G. De Micheli, *Synthesis and Optimization of Digital Circuits* (McGraw–Hill, New York, 1994)
58. Microsemi: http://www.microsemi.com. Accessed Jan 2015
59. Z. Navabi, *Embedded Core Design with FPGAs* (McGraw–Hill, New York, 2007)
60. W. Nöth, R. Kolla, Spanning tree based state encoding for low power dissipation, in *Proceedings of the Conference on Design, Automation and Test in Europe*. Association for Computing Machinery (1999), p. 37
61. S. Park, S. Cho, S. Yang, M. Ciesielski, A new state assignment technique for testing and low power, in *Proceedings of the 41st annual Design Automation Conference*. Association for Computing Machinery (2004), pp. 510–513
62. C. Pedram, A. Despain, Low-power state assignment targeting two- and multilevel logic implementations. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **17**(12), 1281–1291 (1998)
63. I. Pomerancz, K. Cheng, STOIC: state assignment based on output/input functions. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **12**(8), 1123–1131 (1993)
64. QuickLogic: http://www.quicklogic.com. Accessed Jan 2015
65. M. Rawski, T. Łuba, Z. Jachna, P. Tomaszewicz, *Design of Embedded Control Systems*, chapter The influence of functional decomposition on modern digital design process (Springer, Boston, 2005), pp. 193–203
66. M. Rawski, H. Selvaraj, T. Łuba, An application of functional decomposition in ROM-based FSM implementation in FPGA devices. J. Syst. Architect. **51**(6–7), 423–434 (2005)
67. K. Roy, S. Prasad, SYCLOP: synthesis of CMOS logic for low power applications, in *Proceedings, IEEE 1992 International Conference on Computer Design: VLSI in Computers and Processors, ICCD'92*. IEEE (1992), pp. 464–467
68. R. Rudell, A. Sangiovanni-Vincentelli, Multiple-valued minimization for PLA optimization. IEEE Trans. Comput.-Aided Des. **6**(5), 727–750 (1987)
69. G. Saucier, M. Depaulet, P. Sicard, ASYL: a rule-based system for controller synthesis. IEEE Trans. Comput.-Aided Des. **6**(11), 1088–1098 (1987)
70. G. Saucier, P. Sicard, L. Bouchet, Multi-level synthesis on programmable devices in the ASYL system, in *Proceedings of Euro ASIC* (1990), pp. 136–141
71. C. Scholl, *Functional Decomposition with Application to FPGA Synthesis* (Kluwer Academic Publishers, Boston, 2001)
72. R. Senhadji-Navarro, I. Garcia-Vargas, G. Jiménez-Moreno, A. Civit-Balcells, P. Guerra-Gutierrez, ROM-based FSM implementation using input multiplexing in FPGA devices. Electron. Lett. **40**(20), 1249–1251 (2004)
73. E. Sentowich, K. Singh, L. Lavango, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Bryton, A. Sangiovanni-Vincentelli, SIS: a system for sequential circuit synthesis. Technical Report, University of California, Berkely (1992)
74. E. Sentowich, K. Singh, L. Lavango, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Bryton, A. Sangiovanni-Vincentelli, SIS: a system for sequential circuit synthesis, in *Proceedings of the International Conference of Computer Design (ICCD'92)* (1992), pp. 328–333
75. I. Skliarova, V. Sklyarov, A. Sudnitson, *Design of FPGA-Based Circuits Using Hierarchical Finite State Machines* (TUT Press, Tallinn, 2012)
76. V. Sklyarov, Synthesis and implementation of RAM-based finite state machines in FPGAs, in *Proceedings of Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, Villach. Springer (2000), pp. 718–728

77. V. Sklyarov, I. Skliarova, A. Barkalov, L. Titarenko, *Synthesis and Optimization of FPGA-Based Systems*. Lecture Notes in Electrical Engineering, vol. 294 (Springer, Berlin, 2014)
78. V. Solovjev, M. Czyzy, Refined CPLD macrocells architecture for effective FSM implementation, in *Proceedings of the 25th EUROMICRO Conference*, vol. 1, Milan, Italy (1999), pp. 102–109
79. V. Solovjev, M. Czyzy, The universal algorithm for fitting targeted unit to complex programmable logic devices, in *Proceedings of the 25th EUROMICRO Conference*, vol. 1, Milan, Italy (1999), pp. 286–289
80. V. Solovjev, 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* (2001), pp. 170–173
81. G. Sutter, E. Todorovich, S. López-Buedo, E. Boemo, Low-power FSMs in FPGA: encoding alternatives, in *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*. Springer (2002), pp. 363–370
82. E. Tatalov, Synthesis of compositional microprogram control units for programmable devices. Master's thesis, Donetsk National Technical University, Donetsk (2011)
83. A. Tiwari, K. Tomko, Saving power by mapping finite-state machines into embedded memory blocks in FPGAs, in *Proceedings of the Conference on Design, Automation and Test in Europe—Volume 2*. IEEE Computer Society (2004), pp. 916–921
84. C. Tsui, M. Pedram, A. Despain, Exact and approximate methods for calculating signal and transition probabilities in FSMs, in *31st Conference on Design Automation*. IEEE (1994), pp. 18–23
85. G. Venkatamaran, S. Reddy, I. Pomerancz, GALLOP: genetic algorithm based low power FSM synthesis by simultaneous partitioning and state assignment, in *Proceedings of 16th International Conference on VLSI Design* (2003), pp. 533–538
86. T. Villa, T. Saldachna, R. Brayton, A. Sangiovanni-Vincentelli, Symbolic two-level minimization. IEEE Trans. Comput.-Aided Des. **16**(7), 692–708 (1997)
87. T. Villa, A. Sangiovanni-Vincentelli, NOVA: state assignment of finite state machines for optimal two-level logic implememntation. IEEE Trans. Comput.-Aided Des. **9**(9), 905–924 (1990)
88. Y. Xia, A. Almani, Genetic algorithm based state assignment for power and area optimization. IEEE Proc. Comput. Digital Tech. **149**, 128–133 (2002)
89. Xilinx: http://www.xilinx.com. Accessed Jan 2015
90. Xilinx: XST User Guide. V. 11.3, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf. Accessed Jan 2015
91. S. Yang, Logic synthesis and optimization benchmarks user guide. Technical report, Microelectronic Center of North Carolina (1991)
92. B. Zeidman, *Designing with FPGAs and CPLDs* (CMP Books, Lawrence, 2002)

http://www.springer.com/978-3-319-24200-2

Logic Synthesis for FPGA-Based Finite State Machines

Barkalov, A.; Titarenko, L.; Kolopienczyk, M.; Mielcarek, K.; Bazydlo, G.

2016, XIV, 280 p. 205 illus. in color., Hardcover
ISBN: 978-3-319-24200-2