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 $5 \times 10^6$ 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.
Table 2.1 Characteristics of Spartan-3 family

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

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.
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

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

Fig. 2.4  Karnaugh map for system Y

![Karnaugh map](image)

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

$$V = \lceil V_o / t_F \rceil.$$  (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:

$$y_1 = T_1 \vee \overline{T_2}T_3;$$
$$y_2 = \overline{T_1}T_3 \vee T_1 \overline{T_3};$$
$$y_3 = T_2;$$
$$y_4 = T_1T_3.$$  (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 $T_r$ ($r = 1, 3$) is connected only with a single input of EMB. All functions $y_n$ ($n = 1, 4$) have the same propagation time. This example is very simple. But a simple EMB having the configuration $512 \times 64$ 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.
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 \( \Gamma_1 \) 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:

\[
S \geq \max(L(D_1), \ldots, L(D_R), L(y_1), \ldots, L(y_N)).
\] (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. 2.6 Trivial structural diagrams of LUT-based Mealy (a) and Moore (b) FSMs

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

$$y_1 = \bar{T}_1 \bar{T}_2 \lor T_2 \bar{x}_1 \bar{x}_2 \lor T_1. \quad (2.4)$$

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:

- $\bar{T}_1 \bar{T}_2(x_1 \lor \bar{x}_1)(x_2 \lor \bar{x}_2)$;
- $T_2 \bar{x}_1 \bar{x}_2(T_1 \lor \bar{T}_1)$;
- $T_1(T_2 \lor \bar{T}_2)(x_1 \lor \bar{x}_1)(x_2 \lor \bar{x}_2)$.

After transformation (2.4), the table of LUT1 can be constructed (Table 2.3).

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:

$$D_1 = T_1 \bar{T}_2 T_3 x_1 \bar{x}_2 \lor T_1 T_2 T_3 x_3 x_4 \lor \bar{T}_1 \bar{T}_2 \bar{T}_3 \bar{x}_1 x_3. \quad (2.5)$$

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:

$$D_1 = T_1(\bar{T}_2 T_3 x_1 \bar{x}_2 \lor T_2 T_3 x_3 x_4) \lor \bar{T}_1 \bar{T}_2 T_3 \bar{x}_1 x_3 = T_1 A \lor B. \quad (2.6)$$

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.
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:

\[
F(X) = H(X_0, G_1(X_2), \ldots, G_I(X_I)).
\]  

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 $\Gamma_1$ 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:

$$
\begin{align*}
D_1 &= \overline{T_1} \overline{T_2} T_3 \bar{x}_1 \bar{x}_2 \lor \overline{T_1} T_2 \lor T_1 \overline{T_2} T_3; \\
D_2 &= \overline{T_1} T_2 \overline{T_3} x_1 \lor \overline{T_1} \overline{T_2} T_3 x_2; \\
D_3 &= T_2 T_3 \lor \overline{T_1} T_2 T_3 \bar{x}_1 x_2 \lor \overline{T_1} T_2.
\end{align*}
$$

(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:

$$
\begin{align*}
y_1 &= A_2 \lor A_5 \lor A_6; \\
y_2 &= A_2 \lor A_4 \lor A_5; \\
y_3 &= A_3 \lor A_4; \\
y_4 &= A_6.
\end{align*}
$$

(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):

$$
\begin{align*}
y_1 &= T_1; \\
y_2 &= T_2; \\
y_3 &= \overline{T_1} T_3; \\
y_4 &= T_1 \overline{T_2}.
\end{align*}
$$

(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.

\begin{center}
\textbf{Fig. 2.10} State codes of Moore FSM $P(\Gamma_1)$
\end{center}
The dynamic power dissipated in CMOS circuits can be expressed by the well-known formula [81]:

\[ P = \sum_{n=1}^{N} C_n f_n V_{DD}^2. \]  

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 \( T, X, Y, \Phi, q \). Here sets \( T \) and \( X \) determine addresses of cells, the sets \( Y \) and \( \Phi \) 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).
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:

$$2^{R+L} \leq V_0.$$  \hspace{1cm} (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:

$$n_{EMB} = \left\lceil \frac{N + R}{t_F} \right\rceil.$$  \hspace{1cm} (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).
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:

$$2^{R_1} \cdot N \leq V_0.$$  \hspace{1cm} (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:

$$R = M.$$  \hspace{1cm} (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 \(\Pi_A\), an initial GSA \(\Gamma\) can be transformed into a block GSA \(B(\Gamma)\).
<table>
<thead>
<tr>
<th>FSM</th>
<th>Auto</th>
<th>One-hot</th>
<th>Compact</th>
<th>Sequential</th>
<th>Gray</th>
<th>Johnson</th>
<th>Speed1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUT</td>
<td>MHz</td>
<td>LUT</td>
<td>MHz</td>
<td>LUT</td>
<td>MHz</td>
<td>LUT</td>
</tr>
<tr>
<td>bbara</td>
<td>11</td>
<td>639</td>
<td>9</td>
<td>966</td>
<td>13</td>
<td>635</td>
<td>13</td>
</tr>
<tr>
<td>bbsse</td>
<td>29</td>
<td>559</td>
<td>29</td>
<td>559</td>
<td>29</td>
<td>582</td>
<td>29</td>
</tr>
<tr>
<td>bbtas</td>
<td>5</td>
<td>962</td>
<td>8</td>
<td>966</td>
<td>5</td>
<td>966</td>
<td>5</td>
</tr>
<tr>
<td>beecount</td>
<td>7</td>
<td>952</td>
<td>19</td>
<td>639</td>
<td>7</td>
<td>952</td>
<td>7</td>
</tr>
<tr>
<td>ese</td>
<td>49</td>
<td>480</td>
<td>52</td>
<td>477</td>
<td>46</td>
<td>463</td>
<td>50</td>
</tr>
<tr>
<td>dk14</td>
<td>8</td>
<td>945</td>
<td>29</td>
<td>522</td>
<td>8</td>
<td>945</td>
<td>8</td>
</tr>
<tr>
<td>dk15</td>
<td>7</td>
<td>1062</td>
<td>19</td>
<td>737</td>
<td>7</td>
<td>1062</td>
<td>7</td>
</tr>
<tr>
<td>dk16</td>
<td>46</td>
<td>556</td>
<td>46</td>
<td>556</td>
<td>15</td>
<td>625</td>
<td>19</td>
</tr>
<tr>
<td>dk17</td>
<td>6</td>
<td>952</td>
<td>14</td>
<td>669</td>
<td>6</td>
<td>952</td>
<td>6</td>
</tr>
<tr>
<td>dk27</td>
<td>5</td>
<td>900</td>
<td>8</td>
<td>906</td>
<td>5</td>
<td>897</td>
<td>5</td>
</tr>
<tr>
<td>dk512</td>
<td>17</td>
<td>730</td>
<td>17</td>
<td>730</td>
<td>7</td>
<td>899</td>
<td>7</td>
</tr>
<tr>
<td>ex1</td>
<td>64</td>
<td>586</td>
<td>64</td>
<td>586</td>
<td>74</td>
<td>447</td>
<td>67</td>
</tr>
<tr>
<td>ex4</td>
<td>15</td>
<td>962</td>
<td>15</td>
<td>962</td>
<td>16</td>
<td>626</td>
<td>15</td>
</tr>
<tr>
<td>ex6</td>
<td>29</td>
<td>553</td>
<td>30</td>
<td>580</td>
<td>20</td>
<td>621</td>
<td>23</td>
</tr>
<tr>
<td>keyb</td>
<td>56</td>
<td>384</td>
<td>56</td>
<td>384</td>
<td>65</td>
<td>358</td>
<td>71</td>
</tr>
<tr>
<td>kirkman</td>
<td>51</td>
<td>874</td>
<td>84</td>
<td>1058</td>
<td>53</td>
<td>569</td>
<td>48</td>
</tr>
<tr>
<td>lion</td>
<td>3</td>
<td>1084</td>
<td>5</td>
<td>962</td>
<td>3</td>
<td>1080</td>
<td>3</td>
</tr>
<tr>
<td>mark1</td>
<td>27</td>
<td>726</td>
<td>27</td>
<td>726</td>
<td>19</td>
<td>708</td>
<td>22</td>
</tr>
<tr>
<td>me</td>
<td>5</td>
<td>1071</td>
<td>8</td>
<td>1071</td>
<td>5</td>
<td>1071</td>
<td>6</td>
</tr>
<tr>
<td>opus</td>
<td>22</td>
<td>596</td>
<td>22</td>
<td>754</td>
<td>21</td>
<td>628</td>
<td>26</td>
</tr>
<tr>
<td>planet</td>
<td>100</td>
<td>888</td>
<td>100</td>
<td>888</td>
<td>138</td>
<td>389</td>
<td>145</td>
</tr>
<tr>
<td>planet1</td>
<td>100</td>
<td>888</td>
<td>100</td>
<td>888</td>
<td>138</td>
<td>389</td>
<td>145</td>
</tr>
</tbody>
</table>

(continued)
### Table 2.5 (continued)

<table>
<thead>
<tr>
<th>FSM</th>
<th>Auto</th>
<th>One-hot</th>
<th>Compact</th>
<th>Sequential</th>
<th>Gray</th>
<th>Johnson</th>
<th>Speed1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUT</td>
<td>MHz</td>
<td>LUT</td>
<td>MHz</td>
<td>LUT</td>
<td>MHz</td>
<td>LUT</td>
</tr>
<tr>
<td>pma</td>
<td>73</td>
<td>554</td>
<td>73</td>
<td>554</td>
<td>115</td>
<td>438</td>
<td>108</td>
</tr>
<tr>
<td>s1</td>
<td>77</td>
<td>550</td>
<td>77</td>
<td>550</td>
<td>75</td>
<td>447</td>
<td>89</td>
</tr>
<tr>
<td>s1488</td>
<td>140</td>
<td>425</td>
<td>140</td>
<td>425</td>
<td>141</td>
<td>432</td>
<td>130</td>
</tr>
<tr>
<td>s1494</td>
<td>124</td>
<td>412</td>
<td>124</td>
<td>412</td>
<td>143</td>
<td>442</td>
<td>135</td>
</tr>
<tr>
<td>s208</td>
<td>28</td>
<td>559</td>
<td>28</td>
<td>559</td>
<td>13</td>
<td>669</td>
<td>12</td>
</tr>
<tr>
<td>s27</td>
<td>4</td>
<td>962</td>
<td>21</td>
<td>636</td>
<td>4</td>
<td>962</td>
<td>7</td>
</tr>
<tr>
<td>s298</td>
<td>362</td>
<td>406</td>
<td>362</td>
<td>406</td>
<td>330</td>
<td>313</td>
<td>264</td>
</tr>
<tr>
<td>s386</td>
<td>26</td>
<td>577</td>
<td>31</td>
<td>586</td>
<td>28</td>
<td>581</td>
<td>28</td>
</tr>
<tr>
<td>s420</td>
<td>28</td>
<td>559</td>
<td>28</td>
<td>559</td>
<td>14</td>
<td>629</td>
<td>12</td>
</tr>
<tr>
<td>s510</td>
<td>42</td>
<td>900</td>
<td>42</td>
<td>900</td>
<td>39</td>
<td>448</td>
<td>53</td>
</tr>
<tr>
<td>s820</td>
<td>63</td>
<td>429</td>
<td>63</td>
<td>429</td>
<td>85</td>
<td>395</td>
<td>92</td>
</tr>
<tr>
<td>s832</td>
<td>63</td>
<td>429</td>
<td>63</td>
<td>429</td>
<td>73</td>
<td>412</td>
<td>77</td>
</tr>
<tr>
<td>sand</td>
<td>99</td>
<td>569</td>
<td>99</td>
<td>569</td>
<td>121</td>
<td>426</td>
<td>125</td>
</tr>
<tr>
<td>scf</td>
<td>179</td>
<td>676</td>
<td>179</td>
<td>676</td>
<td>202</td>
<td>338</td>
<td>205</td>
</tr>
<tr>
<td>shiftreg</td>
<td>0</td>
<td>1584</td>
<td>9</td>
<td>1080</td>
<td>0</td>
<td>1584</td>
<td>4</td>
</tr>
<tr>
<td>sse</td>
<td>29</td>
<td>559</td>
<td>29</td>
<td>559</td>
<td>28</td>
<td>543</td>
<td>37</td>
</tr>
<tr>
<td>styr</td>
<td>118</td>
<td>430</td>
<td>118</td>
<td>430</td>
<td>127</td>
<td>369</td>
<td>138</td>
</tr>
<tr>
<td>tav</td>
<td>6</td>
<td>1556</td>
<td>6</td>
<td>1556</td>
<td>911</td>
<td>6</td>
<td>911</td>
</tr>
<tr>
<td>tbk</td>
<td>55</td>
<td>406</td>
<td>179</td>
<td>360</td>
<td>71</td>
<td>465</td>
<td>129</td>
</tr>
</tbody>
</table>
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 \( \Gamma_1 \) 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 \( \Gamma_j \) 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 \( D_r = 1 \) (\( r = 1, R_1 \)) 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:

\[
\begin{align*}
F_1 &= \bar{T}_2\bar{T}_3; \\
F_2 &= T_2\bar{T}_3x_1; \\
F_3 &= T_2\bar{T}_3\bar{x}_1x_2; \\
F_4 &= T_2\bar{T}_3\bar{x}_1\bar{x}_2; \\
F_5 &= T_3.
\end{align*}
\]

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

\[
\begin{align*}
D_1 &= T_2\bar{T}_3\bar{x}_1\bar{x}_2 \lor T_3; \\
D_2 &= \bar{T}_2\bar{T}_3 \lor T_2\bar{T}_3\bar{x}_1; \\
D_3 &= T_2\bar{T}_3.
\end{align*}
\]

Each function of the system (2.16) is implemented using only a single LUT having \( S \leq 4 \).
In the common case, the terms $F_h$ are represented as:

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

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, $T_r^0 = \overline{T_r}$, $T_r^1 = T_r$, $T_r^* = 1$.

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
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 $\Gamma$.
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 $\Gamma_4$ is shown in Fig. 2.16. There are marks of states on GSA $\Gamma_4$. 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.

$$
\begin{align*}
p_1 &= \overline{T_1} \overline{T_2} x_1 \lor \overline{T_1} T_2 x_3; \\
p_2 &= \overline{T_1} T_2 x_2 \lor \overline{T_1} T_2 x_4.
\end{align*}
$$

\tag{2.18}
Table 2.7 Table of RLC for Mealy FSM $MP(\Gamma_4)$

<table>
<thead>
<tr>
<th>$a_m$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K(a_m)$</td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$x_1$</td>
<td>$x_3$</td>
<td>–</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$x_2$</td>
<td>$x_4$</td>
<td>–</td>
</tr>
</tbody>
</table>

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 $p_1p_2$,
Table 2.8 Table of LUTer1 for function $p_1$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$X(p_1)$</th>
<th>$p_1$</th>
<th>$T$</th>
<th>$X(p_1)$</th>
<th>$p_1$</th>
<th>$T$</th>
<th>$X(p_1)$</th>
<th>$p_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1T_2$</td>
<td>$x_1x_3$</td>
<td></td>
<td>$T_1T_2$</td>
<td>$x_1x_3$</td>
<td></td>
<td>$T_1T_2$</td>
<td>$x_1x_3$</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>0</td>
<td>01</td>
<td>00</td>
<td>0</td>
<td>10</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>0</td>
<td>01</td>
<td>01</td>
<td>1</td>
<td>10</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>1</td>
<td>01</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>1</td>
<td>01</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.9 Table of LUTer1 for function $p_2$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$X(p_2)$</th>
<th>$p_2$</th>
<th>$T$</th>
<th>$X(p_2)$</th>
<th>$p_2$</th>
<th>$T$</th>
<th>$X(p_2)$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1T_2$</td>
<td>$x_2x_4$</td>
<td></td>
<td>$T_1T_2$</td>
<td>$x_2x_4$</td>
<td></td>
<td>$T_1T_2$</td>
<td>$x_2x_4$</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>0</td>
<td>01</td>
<td>00</td>
<td>0</td>
<td>10</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>0</td>
<td>01</td>
<td>01</td>
<td>1</td>
<td>10</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>1</td>
<td>01</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>1</td>
<td>01</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.10 Structure table of Mealy FSM $MP(\Gamma_3)$

<table>
<thead>
<tr>
<th>$a_m$</th>
<th>$K(a_m)$</th>
<th>$a_s$</th>
<th>$K(a_s)$</th>
<th>$P_h$</th>
<th>$Y_h$</th>
<th>$\Phi_h$</th>
<th>$h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>00</td>
<td>$a_2$</td>
<td>01</td>
<td>$p_1$</td>
<td>$y_1y_2$</td>
<td>$D_2$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_2$</td>
<td>01</td>
<td>$\bar{p}_1p_2$</td>
<td>$y_3y_5$</td>
<td>$D_2$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_2$</td>
<td>01</td>
<td>$\bar{p}_1p_2$</td>
<td>$y_4$</td>
<td>$D_2$</td>
<td>3</td>
</tr>
<tr>
<td>$a_2$</td>
<td>10</td>
<td>$a_3$</td>
<td>10</td>
<td>$p_1p_2$</td>
<td>$y_1y_2$</td>
<td>$D_1$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_3$</td>
<td>10</td>
<td>$p_1\bar{p}_2$</td>
<td>$y_1y_3$</td>
<td>$D_1$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_1$</td>
<td>00</td>
<td>$\bar{p}_1p_2$</td>
<td>$y_3y_5$</td>
<td>–</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_1$</td>
<td>00</td>
<td>$\bar{p}_1p_2$</td>
<td>$y_1y_2$</td>
<td>–</td>
<td>7</td>
</tr>
<tr>
<td>$a_3$</td>
<td>10</td>
<td>$a_2$</td>
<td>01</td>
<td>1</td>
<td>$y_1y_2$</td>
<td>$D_2$</td>
<td>8</td>
</tr>
</tbody>
</table>

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.
Let the following condition take place:

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

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: $(x_1, x_2)$ and $(x_3, x_4)$. Let us denote this number as $n_P$. Let us encode each pair using $R_P$ variables, where:

$$R_P = \lceil \log_2 n_P \rceil. \quad (2.20)$$

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:

$$P = P(Z, X). \quad (2.21)$$

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:

$$Z = Z(T, P). \quad (2.22)$$
There is the following table of RLC in the case of $MP P(I_4)$ (Table 2.12).

Now, there are the following functions implemented by the LUTer:

\[
\begin{align*}
    p_1 &= \bar{z}_1 x_1 \vee z_1 x_3; \\
    p_2 &= \bar{z}_1 x_2 \vee z_1 x_4.
\end{align*}
\]  

(2.23)

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

\[
R_P + |X(p_g)| \leq S.
\]  

(2.24)

Of course, this approach has sense if there is

\[
R_P < R_0.
\]  

(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 $MP P$ 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.
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 ($h = 1, H_0$). 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 $\Gamma$.
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 $P(Γ_4)$

<table>
<thead>
<tr>
<th>$a_m$</th>
<th>$K(a_m)$</th>
<th>$a_s$</th>
<th>$K(a_s)$</th>
<th>$X_h$</th>
<th>$Y_h$</th>
<th>$Φ_h$</th>
<th>$h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>00</td>
<td>$a_2$</td>
<td>01</td>
<td>$x_1$</td>
<td>$y_1y_2$</td>
<td>$D_2$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_1x_2$</td>
<td>$y_3y_5$</td>
<td>$D_2$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_1x_2$</td>
<td>$y_4$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>$a_2$</td>
<td>01</td>
<td>$a_3$</td>
<td>10</td>
<td>$x_3x_4$</td>
<td>$y_1y_2$</td>
<td>$D_1$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_3x_4$</td>
<td>$y_1y_3$</td>
<td>$D_1$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_3x_4$</td>
<td>$y_3y_5$</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_3x_4$</td>
<td>$y_1y_2$</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$a_3$</td>
<td>10</td>
<td>$a_2$</td>
<td>01</td>
<td>1</td>
<td>$y_1y_2$</td>
<td>$D_2$</td>
<td>8</td>
</tr>
</tbody>
</table>

Let us discuss an example of design for Mealy FSM $PH(Γ_4)$. There is the GSA $Γ_4$ shown in Fig. 2.16. As in the previous case, the following sets can be found for $P(Γ_4)$: $X = \{x_1, \ldots, x_4\}$, $Y = \{y_1, \ldots, y_5\}$, $A = \{a_1, a_2, a_3\}$, $Φ = \{D_1, D_2\}$, $T = \{T_1, T_2\}$. 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 $P(Γ_4)$ (Table 2.13).

Let the following condition take place:

$$R_H \leq S.$$  \hfill (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:

$$D_1 = F_4 \lor F_5;$$
$$D_2 = F_1 \lor F_2 \lor F_3 \lor F_8;$$
$$y_1 = F_1 \lor F_4 \lor F_5 \lor F_7 \lor F_8;$$
$$y_2 = F_1 \lor F_4 \lor F_7 \lor F_8;$$
$$y_3 = F_2 \lor F_5 \lor F_6;$$
$$y_4 = F_3;$$
$$y_5 = F_2 \lor F_6.$$  \hfill (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.
The following equations can be obtained from the Karnaugh map (Fig. 2.23):

\[
\begin{align*}
D_1 &= \bar{z}_2 z_3; \\
D_2 &= \bar{z}_3; \\
y_1 &= \bar{z}_1 \vee \bar{z}_2 z_3; \\
y_2 &= \bar{z}_1; \\
y_3 &= z_1 z_3 \vee z_1 z_2; \\
y_4 &= z_1 \bar{z}_2 \bar{z}_3; \\
y_5 &= z_1 z_2.
\end{align*}
\]

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:
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 $\Phi_h$ 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>
<thead>
<tr>
<th>$a_m$</th>
<th>$K(a_m)$</th>
<th>$X_h$</th>
<th>$C(F_h)$</th>
<th>$Z_h$</th>
<th>$h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>00</td>
<td>$x_1$</td>
<td>101</td>
<td>$z_1z_2$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_1x_2$</td>
<td>011</td>
<td>$z_2z_3$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_1\bar{x}_2$</td>
<td>001</td>
<td>$z_3$</td>
<td>3</td>
</tr>
<tr>
<td>$a_2$</td>
<td>01</td>
<td>$x_3x_4$</td>
<td>100</td>
<td>$z_1$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_3\bar{x}_4$</td>
<td>000</td>
<td>$-$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\bar{x}_3x_4$</td>
<td>010</td>
<td>$z_2$</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\bar{x}_3\bar{x}_4$</td>
<td>110</td>
<td>$z_1z_2$</td>
<td>7</td>
</tr>
<tr>
<td>$a_3$</td>
<td>10</td>
<td>1</td>
<td>111</td>
<td>$z_1z_2z_3$</td>
<td>8</td>
</tr>
</tbody>
</table>
It is the following system in the discussed case:

\[
\begin{align*}
    z_1 &= \bar{T}_1 \bar{T}_2 x_1 \lor \bar{T}_1 T_2 x_3 x_4 \lor \bar{T}_1 T_2 \bar{x}_3 \bar{x}_4 \lor T_1 \bar{T}_2; \\
    z_2 &= \bar{T}_1 \bar{T}_2 x_1 \lor \bar{T}_1 \bar{T}_2 x_2 \lor \bar{T}_1 T_2 \bar{x}_3 \lor T_1 \bar{T}_2; \\
    z_3 &= \bar{T}_1 \bar{T}_2 \bar{x}_1 \lor T_1 \bar{T}_2.
\end{align*}
\]  

(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

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)
54. C. Maxfield, FPGAs: Instant Access (Newnes, 2008)
59. Z. Navabi, Embedded Core Design with FPGAs (McGraw–Hill, New York, 2007)
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
75. I. Skliarova, V. Sklyarov, A. Sudmitson, Design of FPGA-Based Circuits Using Hierarchical Finite State Machines (TUT Press, Tallinn, 2012)
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
92. B. Zeidman, *Designing with FPGAs and CPLDs* (CMP Books, Lawrence, 2002)
Logic Synthesis for FPGA-Based Finite State Machines
2016, XIV, 280 p. 205 illus. in color., Hardcover
ISBN: 978-3-319-24200-2