Chapter 8
FSM-Based Logic Controller Synthesis in Programmable Devices with Embedded Memory Blocks

Grzegorz Borowik, Grzegorz Łabiak, and Arkadiusz Bukowiec

Abstract. For a typical digital system, the design process consists of compilation, translation, synthesis, logic optimization, and technology mapping. Although the final result of that process is a structure built of standard cells, logic cells, macroblocks, and similar components; the characteristics of the system (the silicon area, speed, power, etc.) depend considerably on the logic model of the digital system. Therefore, the synthesis and logic optimization has a significant impact on the quality of the implementation. In this chapter, we describe methods of designing and synthesis for logic controllers in novel reprogrammable structures with embedded memory blocks. This chapter is generally based on the ideas published in [5], however, a number of issues were extended and provide detailed information about the methods and algorithms used in the problem, including [6, 13, 14, 38]. The method starts with the formal specification of a logic controller behavior. To specify the complex nature of a logic controller we have chosen statechart diagrams [21]. The main advantage of this specification is the possibility of detecting all reachable deadlocks [26]. It is particularly important in the case of safety-critical systems since any failure of such system may cause injury or death to human beings. Having graphically specified the behavior, it is subsequently converted into a mathematical model [30]. Next, the mathematical model of the statechart is transformed into an equivalent finite state machine (FSM) [29]. Thus, the logic controller in FSM form can be synthesized by applying ROM-based decomposition method [6] or architectural decomposition method [13], and finally implemented in embedded memory block equipped architectures [45, 48]. Such architectures offer ability to update

Grzegorz Borowik
Institute of Telecommunications, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: G.Borowik@tele.pw.edu.pl

Grzegorz Łabiak · Arkadiusz Bukowiec
Computer Engineering & Electronics Department, University of Zielona Góra,
Licealna 9, 65-417 Zielona Góra, Poland
e-mail: {G.Labiak,A.Bukowiec}@iie.uz.zgora.pl

R. Klempous and J. Nikodem (eds.), *Innovative Technologies in Management and Science*,
Topics in Intelligent Engineering and Informatics 10, DOI: 10.1007/978-3-319-12652-4_8

the functionality, partial reconfiguration, and low non-recurring engineering costs relative to an FPGA design.

8.1 Preliminaries

Logic controller is an electronic digital device used for automation of electromechanical processes, such as control of machinery on factory assembly line, lighting fixtures, traffic control systems, household appliances, and even automation of systems whose role is to maintain an ongoing interactions with their environment, i.e. controlling mechanical devices, such as a train, a plane, or ongoing processes, e.g. processes of a biochemical reactor.

Logic controller receives signals both from controlled object and optional operator, and repeatedly produces outputs to controlled object. If controller operates on binary values, as opposed to continuous values, it is called *binary control system* (binary controller). Therefore, binary controller can easily be implemented as a digital circuit. The general idea of binary control system is presented in Fig. 8.1.

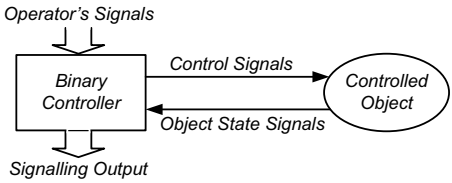


Fig. 8.1 Binary control system

An important application of binary controllers is embedded systems design [19, 20]. In particular, it is advisable for data processing systems. Thus, such system can be realized according to application-specific architecture known as a control unit with datapath (Fig. 8.2). Datapath module receives data and process them according to implemented algorithm and sends status signal to control unit. Control unit signals steers the data among the units and registers of datapath. If data are of discrete

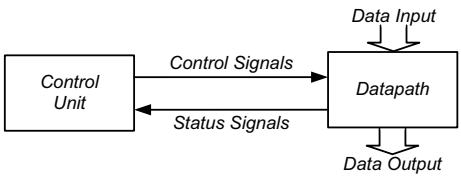


Fig. 8.2 Control unit with datapath architecture

values the whole system can be implemented in Field Programmable Gate Arrays (FPGA); the basic architecture for General-purpose processors [44].

Logic controllers can find their practical application in different areas of biotechnology. In [40] a technology of smart hand prosthesis control based on myoelectric signals is presented. The key elements of this design are arithmetic and control units. Bioreactor controller is the focus of the paper [23]. It provides an overview of current and emerging bioreactor control strategies based on unstructured dynamic models of cell growth and product formation. Nevertheless, process control plays a limited role in the biotechnology industry as compared to the petroleum and chemical industries. This demand for process modeling and control is increasing, however, due to the expiration of pharmaceutical patents and the continuing development of global competition in biochemical manufacturing. The lack of online sensors that allow real-time monitoring of the process state has been an obstruction to biochemical process control. Recent advances in biochemical measurement technology, however, have enabled the development of advanced process control systems [23].

To specify a complex nature of a controller we have chosen statechart diagrams. A statechart diagram is a state-based graphical scheme enhanced with concurrency, hierarchy and broadcasting mechanism. It may describe a complex system, but requires the system is composed of a finite number of states [21].

To synthesize statechart-base logic controller it is necessary to precisely define its behavior, however, the issue of hardware synthesis of statecharts is not solved ultimately. There are many implementation schemes depending on target technology. First, published in [17], consists of transformation of the statechart into the set of hierarchically linked FSMs traditionally implemented. In [18], a special encoding of the statechart configurations targeted at PLA structures is presented. The drawback of this method is that diagram expresses transitions between simple states only. In [36], Ramesh enhanced the coding scheme by introducing a prefix-encoding. However, the common drawbacks of the presented methods is the lack of support for history attributes and broadcast mechanism. Other implementation methods using HDL and based on ASIP are presented in [4, 24] and [10], respectively. In [16] statechart diagram is used as a graphic formalism for program specification for PLC controller, where UML language heavily support the design process.

In this chapter, we describe methods of designing and synthesis for logic controllers in novel reprogrammable structures with embedded memory blocks. It is generally based on the ideas published in [5], however, a number of issues were extended and provide detailed information about the methods and algorithms used in the problem, including [6, 13, 14, 38]. In subsection 8.2, we start with the example of a chemical reactor and its formal specification using the statechart diagram, as well as provide assumptions of hardware implementation. Having graphically specified the behavior, it is subsequently converted into a mathematical model [30]. Next, the mathematical model of the statechart is transformed into an equivalent finite state machine (FSM) [29] that is described in subsection 8.3. Finally, the logic controller in FSM form can be synthesized by applying ROM-based decomposition method [6] (see subsection 8.4.2) or architectural decomposition method [13] (see subsection 8.4.3), and finally implemented in embedded memory block equipped

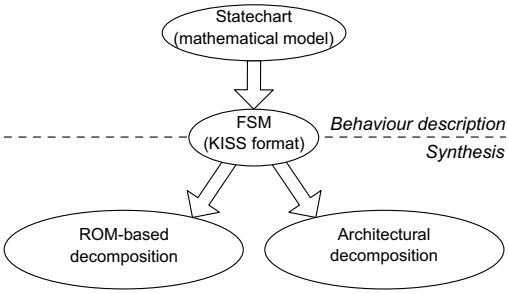


Fig. 8.3 Transformation and synthesis of logic controller

architectures [45, 48]. Such architectures offer ability to update the functionality, partial reconfiguration, and low non-recurring engineering costs relative to an FPGA design (subsection 8.4.1). The idea of transformation and synthesis of logic controller is provided in Fig. 8.3.

8.2 Example and Assumptions of Hardware Implementation

The statechart diagram can certainly specify reactive system behavior. As an example of practical application, the schematic diagram of a chemical reactor and appropriate statechart diagram of its logic controller are presented in Fig. 8.4 and 8.5, respectively.

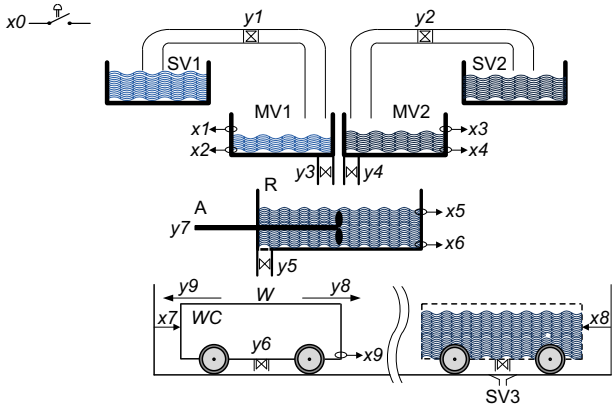


Fig. 8.4 Schematic diagram of chemical plant with wagon

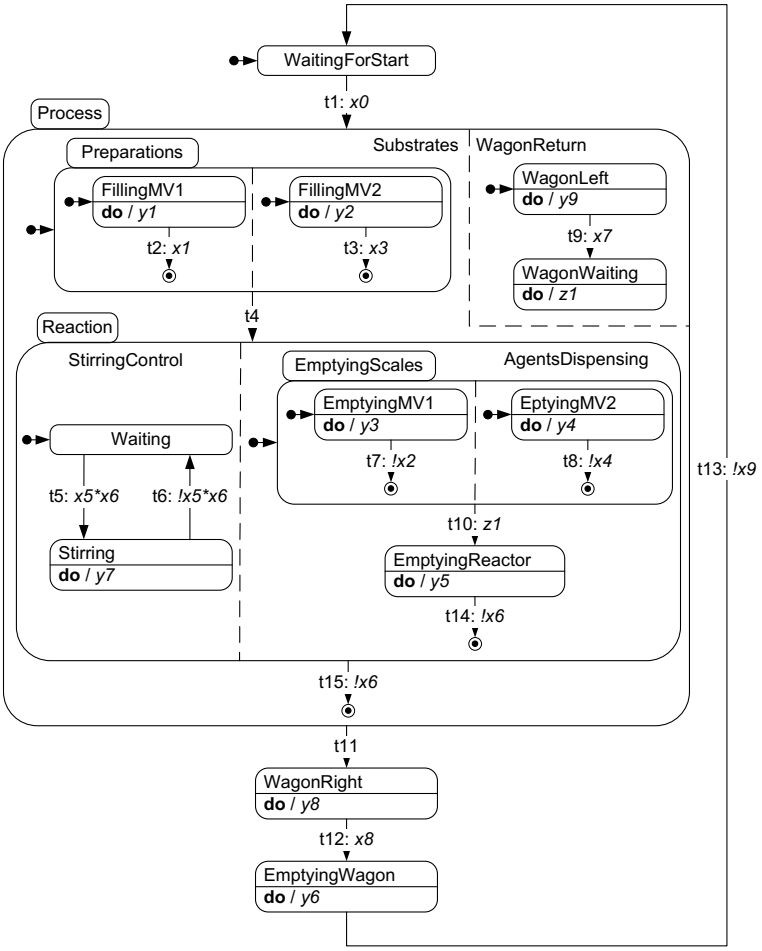


Fig. 8.5 Statechart diagram of chemical reactor controller

The working of the reactor is as follows. Initially, the reacting substances are kept in containers *SV1* and *SV2* (Fig. 8.4), and the emptied wagon waits in its initial position on the far right site (Fig. 8.5, state *WaitingForStart*). Then, the operator starts the proces with the signal *x0*. The pump *y1* and the pump *y2* make that liquid substrates from containers *SV1*, *SV2* are being measured out in scales *MV1* and *MV2*, respectively (state *Preparations*). During this, the wagon is coming back to its far left position. After the substrates are measured out, the main reaction starts (state *Reaction*). Next, scales fill main container *R* with agents (state *AgentDispensing*) and agitator *A* starts rotating (state *StirringControl*). After filling up the main container, the product of the reactor is poured to the wagon (state *EmptyingReactor*). Then, the wagon goes to empty (states *WagonRight* and *EmptyingWagon*).

Rounded rectangles in the statechart diagram, called states, correspond to activities in the controlled object (in this case chemical reactor). In general, states can be in sequential relationship (*OR* state), or in concurrent relationship (*AND* states). Then, these states make sequential or parallel automaton. States can be simple or compound. The latter state can be nested with other compound or simple states. In the diagram (Fig. 8.5), the *AND* states are separated with a dashed line. States are connected with transitions superimposed by predicates. Predicates must be met to transform activity between states connected with an arrow.

To synthesize statechart-based logic controller it is necessary to precisely define its behavior in terms of logic values. In Fig. 8.6 a simple diagram and its waveform illustrate the main dynamic features. Logic value **1** means activity of a state or presence of an event, and value **0** means their absence. When transition $t1$ is fired ($T = 350$) event $t1$ is broadcast and becomes available to the system at next instant of discrete time ($T = 450$). The activity moves from state *START* to state *ACTION*, where entry action (keyword *entry*) and do-activity (ongoing activity, keyword *do*) are performed (events *entr* and *d* are broadcast). Now, transition $t2$ becomes enabled. Its source state is active and predicate superimposed on it (event $t1$) is met. So, at the instant of time $T = 450$, the system transforms activity to the state *STOP*,

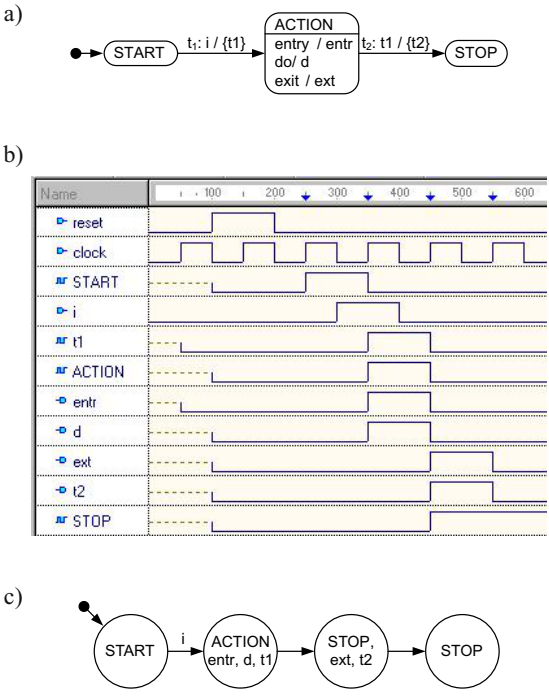


Fig. 8.6 Simple diagram with main features (a), its waveform (b), and equivalent FSM (c)

performs exit action (keyword *exit*, event *ext*) and triggers event $t2$, which do not affect any other transition. The step is finished.

Summarizing, dynamic characteristics of hardware implementation are as follows:

- system is synchronous,
- system reacts to the set of available events through transition executions,
- generated events are accessible to the system during next tick of the clock.

Basing on the assumptions, hardware implementation involves mapping syntax structure of statechart into hardware elements, namely, every state and action (transition actions, *entry* and *exit* actions) are assigned flip-flops and every event is a signal. Both flip-flops and signals have their functions (flip-flops have excitation functions) which are created based on rule of transitions firing and on presented hardware assumption [27, 28].

8.3 Transformation to FSM Model

The transformation of statechart diagram into FSM model [22] involves building equivalent FSM Mealy automaton using statechart elements which for external observer behaves just the way statechart does. Final Mealy automaton is described in KISS format [42]. The main notion the transformation revolves around is a global state of the statechart [29, 30]. The general idea is to create an equivalent target Mealy automaton whose states corresponds to the global states of a statechart. The global state of a statechart is defined as a superposition of local activities, namely, local states, transition events, *entry* and *exit* actions. In hardware implementation these activities are implemented by means of flip-flops and their excitation functions (see subsection 8.2). The whole process is explained on the example diagram in Fig. 8.7.

The diagram in Fig. 8.7 features both concurrency and hierarchy. It consists of five simple states $\{s_1, s_3, s_4, s_5, s_6\}$, one compound state s_2, and two *do* actions $\{x, y\}$ assigned to states, s_4 and s_6, respectively. The diagram has three input signals $\{a, b, c\}$ and two output signals $\{x, y\}$. There is no transition action nor *entry* and *exit* actions.

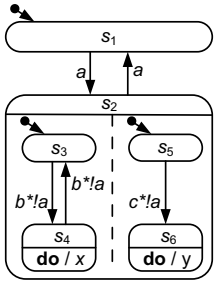


Fig. 8.7 Simple concurrent and hierarchical diagram

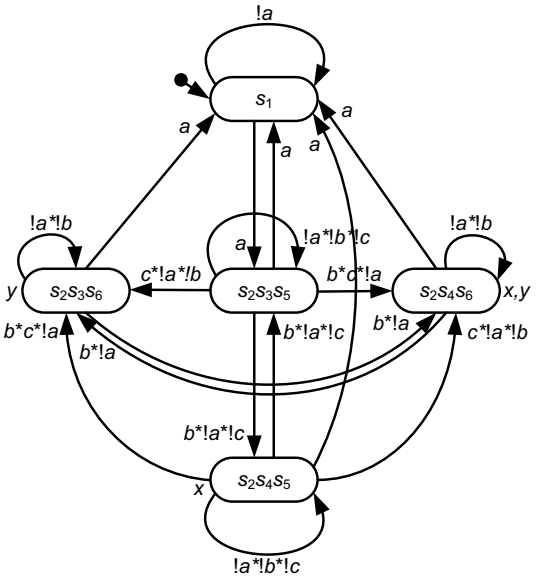


Fig. 8.8 Equivalent FSM Moore automaton of diagram in Fig. 8.7

Having defined a global state of a statechart as a vector of local activities, it is possible to construct hypothetical equivalent Moore type automaton. Fig. 8.8 presents such an automaton. Names of the states correspond to local activities of the original diagram, e.g. state $s_2s_4s_6$ corresponds to the activity of the statechart states $\{s_2, s_4, s_6\}$, respectively, and broadcast events $\{x, y\}$. Transition labels are Boolean expressions which must be met to fire transitions. It is necessary to emphasis that both models statechart and FSM are deterministic and their transition predicates must be orthogonal. In general, due to concurrency and hierarchy, equivalent FSMs have more states and transitions, which is a big drawback of the transformation.

The vector of local activities with their excitation functions allows us to symbolically express a global state as well as the characteristic function for every global state of a statechart. In terms of Boolean algebra, global state is defined as a product of signals representing local activities, e.g. initial global state of the statechart which corresponds to state s_1 of FSM is $G_0 = s_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6$. Characteristic function χ_G of a set of elements $G \subseteq U$ is a Boolean function which yields true value ($\chi_G(g) = 1$) when $g \in G$, otherwise yields false value. It means that a characteristic function represents a set, and Boolean operations on the characteristic functions (e.g. $+$, $*$, negation) correspond to the operations on sets which they precisely represent (e.g. $\cup$, $\cap$, complement). Coding every global state of a statechart, just like G_0,

and summing them we obtain characteristic function representing every global state of a statechart. For the diagram in Fig. 8.7 the characteristic function is as follows:

$$\chi_{[G_0\rangle} = s_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6 + \overline{s}_1 s_2 s_3\overline{s}_4\overline{s}_5 s_6 + \overline{s}_1 s_2 s_3\overline{s}_4 s_5\overline{s}_6 + \\ + \overline{s}_1 s_2\overline{s}_3 s_4\overline{s}_5 s_6 + \overline{s}_1 s_2\overline{s}_3 s_4 s_5\overline{s}_6 \tag{8.1}$$

Computation of the characteristic function is carried out according to the algorithm 8.1 performing symbolic traversal.

Listing 8.1 Symbolic traversal of state space

```
symb_trav_of_Statechart(Z, init_state) {
    χ[G0⟩ = curr_states = init_state;
3   while (curr_states != 0) {
        next_states = Image_computation(Z, curr_states);
5       curr_states = next_states * ¬χ[G0⟩;
        χ[G0⟩ = curr_states + χ[G0⟩;
7   }
 }
```

Sets of global states are represented by their characteristic functions (*put in italic*). The operations on the sets are represented by Boolean operations on corresponding characteristic functions. The algorithm starts with *init_state* (e.g. G_0) and in breath-first manner generates the set of all reachable global next states in one formal step (line no. 4). In line no. 5, only new global states are calculated (*curr_states*), and in line no. 6, new states are added to the so far generated global states ($\chi_{[G_0\rangle}$). The algorithm stops when there is no new *curr_states* (line no. 3).

The formal step from line no. 5 is performed by the *Image_computation* procedure. For the *curr_states* it generates their image in the transformation with the functional vector (δ_S) of excitation functions of local activities in the statechart diagram. Hence, it is possible, in one formal step to compute the set of all reachable global next states for the set of global states (*curr_states*). The procedure is as follows:

$$next\_states = \exists_s \exists_x (curr\_states * \prod_{i=1}^{n} [s'_i \odot (curr\_states * \delta_{Si}(s,x))]), \tag{8.2}$$

$$next\_states = next\_states\langle s' \leftarrow s\rangle, \tag{8.3}$$

where s, s', x denote the present state, next state and input signals, respectively; $\exists_s$ and $\exists_x$ represent the existential quantification of the present state and signal variables; n is a number of state variables; $\odot$ and $*$ represent logic operators XNOR and AND, respectively; equation 8.3 means swapping variables in expression.

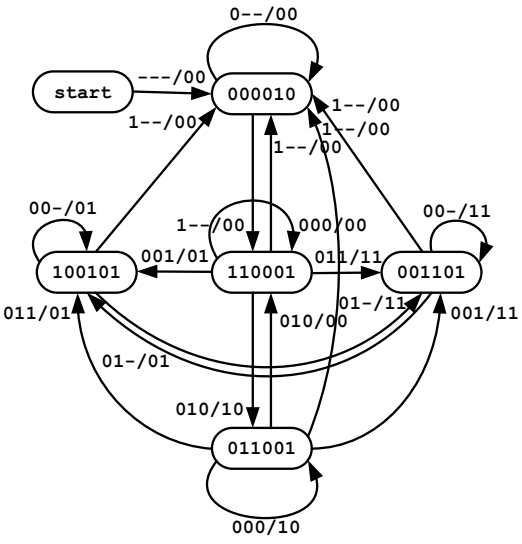
Characteristic function does not contain information on transitions. This information can be obtained by the calculation of the transition relation under input (χ_{TRI}) using the following equation:

$$\chi_{TRI}(s', x, s) = \prod_{i=1}^{n} [s'_i \odot \delta_{Si}(x, s)], \tag{8.4}$$

where the symbol $\odot$ represents the logic XNOR operator and n is the number of state variables. The relation $\chi_{TRI}(s',x,s) = 1$ implies that in state s, there exists a transition to state s' on input x.

The transformation of Moore automaton (Fig. 8.8) into Mealy automaton (Fig. 8.9) is very simple [22], but we must add one extra state (*start*) to include output for initial state. The two characteristic functions ($\chi_{[G_0\rangle}$ and χ_{TRI}) provide enough information to generate equivalent FSM in KISS format. The algorithm 8.2 starts with characteristic function of global states space $\chi_{[G_0\rangle}$ and with characteristic function χ_{TRI}. The transition regarding input signals is represented by the product t, which is a relation between current (G_i) and next (G'_j) states, represented as conjunction formulae (line no. 4). For every pair of states: current state and next state (G_i, G'_j), it is being checked whether there is a transition between them (line no. 5). In line no. 6 state variables (s,s') are removed from transition product t, hence t_x represents the only part of the expression which solely depends on input variables x. Thus, current and next states are put into 4-tuple KISS line (lines 7 and 8) and sent into KISS file (line no. 21). Between lines 15 and 20 the input vector is being computed and dependency on input variables is being checked for each minterm in t_x expression. It is formally computed according the following formula:

$$\frac{\partial f}{\partial x_i} = f_{x_i} \oplus f_{\overline{x}_i}, \tag{8.5}$$




```
.start_kiss
.i 3
.o 2
.p 19
.s 6
.r start
 ---  start 000010 00
 0-- 000010 000010 00
 1-- 001101 000010 00
 1-- 011001 000010 00
 1-- 100101 000010 00
 1-- 110001 000010 00
 00- 001101 001101 11
 001 011001 001101 11
 01- 100101 001101 11
 011 110001 001101 11
 000 011001 011001 10
 010 110001 011001 10
 01- 001101 100101 01
 011 011001 100101 01
 00- 100101 100101 01
 001 110001 100101 01
 1-- 000010 110001 00
 010 011001 110001 00
 000 110001 110001 00
.end_kiss
```

Fig. 8.9 KISS of the FSM in Fig. 8.8

Listing 8.2 Transitions generation algorithm

```
Transitions_Generation(Z, χ[G0⟩, χTRI) {
2 for each global state Gi χ[G0⟩(Gi) = 1;
    for each global state G'j χ[G0⟩(G'j) = 1; {
4     t = χTRI(s',x,s) * Gi * G'j;
      if t = 0 then continue;
6     tx = ∃s' ∃s t
      current−st = Gi;
8     next−st = G'j⟨s' ← s⟩;
      for each output yi ∈ Y {
10      if G'j⟨s' ← s⟩ * λMi ≠ 0 then out[i] = 1
        else out[i] = 0;
12    }
      for each minterm mi in tx {
14      for each input xj ∈ X {
          if ∂mi/∂xj ≠ 0 then { // deps on xj
16          if mi * xj ≠ 0 then in[j] = 1
            else in[j] = 0;
18        }
          else in[j] = −;
20      }
        KISS << <in, current−st, next−st, out>
22    }
  }
24}
```

where f_{x_i}, $f_{\overline{x}_i}$ are positive and negative algebraic cofactor, respectively, and the symbol $\oplus$ represents XOR operator. In lines 16, 17 and 19 we determine the impact of signal x_j on the transition. Subsequently, in lines from 9 to 11, we compute the output vector, where y_i is an i-th output variable, and λ_i is its signal function. Although this is not presented in the algorithm, it is enough to execute these four lines only once per the next state s'.

Figure 8.9 presents final Mealy automaton and its KISS file of the diagram from Fig. 8.7. Binary labels both, states and transitions, correspond to signals and states activities from statechart diagram according to the following order: input $[a,b,c]$, output $[x,y]$, state $[s_3,s_5,s_4,s_6,s_1,s_2]$. A single line in KISS file, e.g. `1-- 000010 110001 00`, defines transition from state s_1 (`000010`) to state $s_2s_3s_5$ (`110001`) under the input a without active outputs.

The transformation has been successfully implemented in academic *HiCoS* system [47] and all Boolean transformations have been performed by means of Binary Decision Diagrams [33, 35].

8.4 Synthesis

In modern logic synthesis, regardless of the implementation technology (programmable devices, Gate Array or Standard Cell structures), the problem of finite state machine synthesis (in particular – the problem of internal state encoding) is an issue of significant practical importance.

Many methods for structural synthesis of FSMs have been reported in the literature. Their diversity is a consequence of different assumptions taken to simplify calculations, as well as different types of intended target components. Thus, different methods of FSM synthesis have been developed for PLA structures [32, 34, 41], for ROM memories [6, 37, 38], and for PLD modules [15].

A distinctive feature of traditional methods of FSM synthesis is the application of logical minimization before the process of state encoding. This minimization is possible when the inputs and outputs of the combinational part of the sequential circuit is represented with multi-valued symbolic variables. Unfortunately, such methods are limited to two-level structures. For other implementation styles different methods are needed. The research in this area goes into two directions: one concerns the implementation with multilevel gate structures, while the other embraces implementations with cellular FPGA and CPLD structures.

In the first case, like for two-level structures, the starting point of the synthesis process is a structure in which the combinational circuit is connected to the inputs of a register operating as state memory (Fig. 8.10a), whereas in the other case, the combinational circuit is connected to the outputs of such a register (Fig. 8.10b).

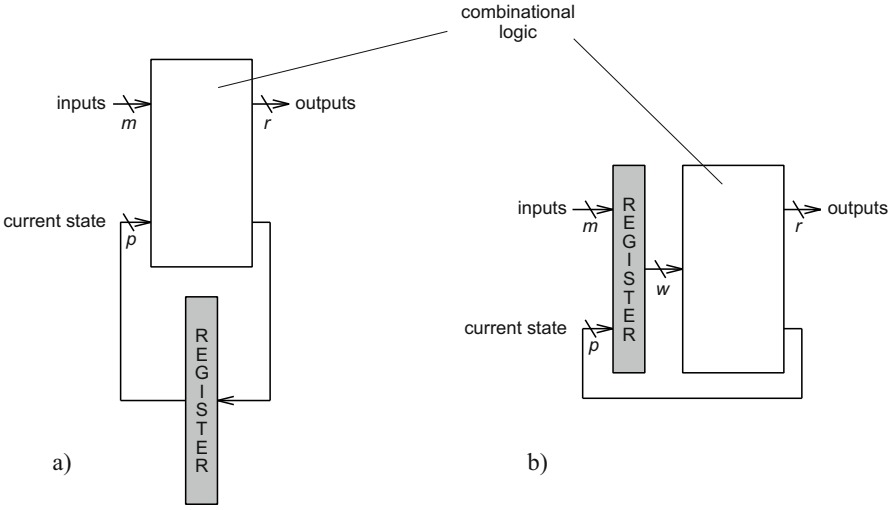


Fig. 8.10 Two models of a sequential circuit: classical (a), with microprogramming capability (b)

Until recently, mainly the first model (Fig. 8.10a) was used in synthesis of sequential machines. The optimization of the selection of state encoding was done for two-level or multilevel gate structures and was aimed at the reduction of hardware resources (silicon area).

The second model (Fig. 8.10b) was used in microprogrammed control circuits, with the combinational circuit implemented with ROM memory [1]. In the microprogrammed version of the sequential circuit, the fixed ROM memory was a separate element – separated from the rest of the circuit. The advantage of this structure was an ability to program the microcode memory, which was the only possible way to reconfigure the circuit at that time. These advantages made the capacity of the memory to be a non-critical factor, although the reduction of this capacity was a common optimization criterion.

Microprogrammed control has been a very popular alternative implementation technique for control units. However, as systems have become more complex and new programmable technologies have appeared, the concept of classical microprogramming has become less attractive for control unit implementations. But the main idea of Microprogrammed Control Units, i.e. implementation of combinational part of the sequential circuit with a ROM, has gained new motivation after the appearance of programmable logic devices [3]. In particular, the growing interest in ROM-based synthesis of finite state machines has been caused by the inclusion of Embedded Memory Blocks in modern FPGAs.

8.4.1 Modern Technologies of Controller Manufacture

Field-programmable devices are very often used for the implementation of logic controllers. Since these devices can be programmed by the user during the design process, they are a good platform for dedicated control algorithms. There are many different types of such devices – from simple Programmable Logic Devices (PLDs) through Complex PLDs (CPLDs) to advanced Field Programmable Gate Arrays (FPGAs) [25].

The research presented in this chapter is oriented towards FPGA devices with Embedded Memory Blocks.

FPGAs are built with a matrix of small configurable logic blocks (CLBs), which are connected using internal programmable interconnections and surrounded by programmable input/output blocks (IOBs) for communication with the environment (Fig. 8.11) [25]. An FPGA contains from several to tens of thousands of CLBs. Each logic block is built of look-up tables (LUTs), D type flip-flops, and some additional control logic. One LUT has typically four inputs, however up to six or more [39], and can implement any Boolean function of this number of variables, i.e. working of four-input LUT can be perceived as 16×1 ROM.

The new FPGAs have also memory blocks [45, 48]. They have different names depending on vendor, for e.g. Embedded Memory Block (EMB) in Altera devices or

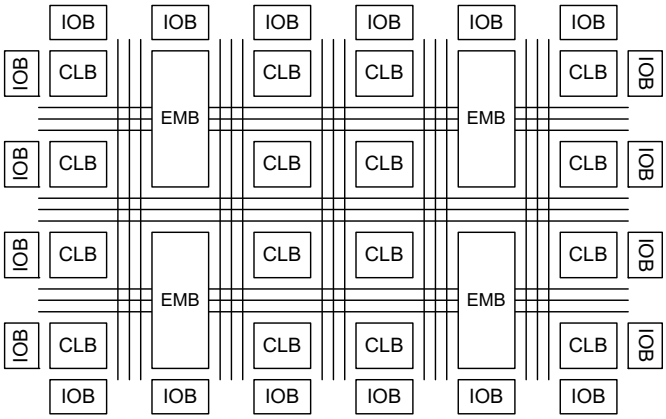


Fig. 8.11 Structure of an FPGA device

Block RAM in Xilinx devices. These blocks are placed in columns typically in outer areas of device (Fig. 8.11). These memories are from 512 bits up to 36 Kbits in size. The most popular size of the memory block of cheaper FPGAs is 4 Kbits. The very big advantage of such block is existence of feature that allows them to be set to one of several modes of data width (Tab. 8.1). They can also work in one of modes, like single-port RAM, a dual-port RAM or ROM. When an embedded memory block works in the ROM mode, it is initiated with the content during the programming process of an FPGA device. In this mode, it can be used for the implementation of combinational functions.

Table 8.1 Typical configuration modes of 4-Kbit embedded memory block

Mode	Number of words	Width of the word [bits]
4K×1	4096	1
2K×2	2048	2
1K×4	1024	4
512×8	512	8
256×16	256	16

8.4.2 Functional Decomposition and ROM-Based Synthesis

One of the main goals of the synthesis is not only technological implementation of logic controller but also optimization of hardware resources consumption. It is particularly important when the design is intended for novel programmable structure containing LUT-based cells and embedded memory blocks. The other factor of vital importance is Boolean minimization strategy. Authors' proposition is to apply the idea of functional decomposition, i.e. a structure with address modifier (Fig. 8.12b), which is best suited for the FSMs in KISS format from previous section [6, 37, 38].

A limited size of embedded memory blocks available in FPGAs is the main reason behind the application of this structure. The implementation of an FSM shown in Fig. 8.12b can be seen as a serial decomposition of the memory block included in the structure of Fig. 8.12a into two parts: an address modifier and a memory block of smaller capacity than required for the realization of the structure in Fig. 8.12a. In the considered FSM implementation both parts are implemented in embedded memory blocks which are configured as ROM memory, with its content determined at the time of the programming. The address modifier (first block) is used here to reduce the number of memory address lines of the second block.

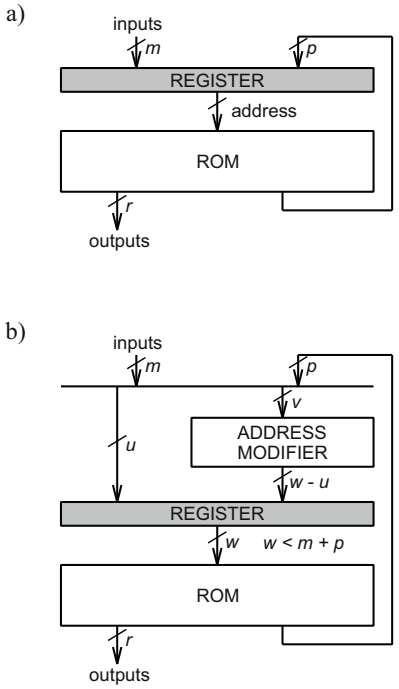


Fig. 8.12 FSM implementation: using ROM memory (a), with the addition of an address modifier (b)

As a result, sequential circuits requiring large-capacity ROM memories (and thus not implementable in the architecture of Fig. 8.12a) can be implemented using memory blocks with a smaller number of inputs. Then, the size of the required memory [6] is equal

$$M = 2^v \cdot (w-u) + 2^w \cdot (r+p). \tag{8.6}$$

For explaining the work of functional decomposition with address modifier partition description and partition algebra are applied [22]. They describe logic dependencies in considered FSM.

Let $\mathbb{T}$ be an isomorphic function between the domain D_δ of the transition function and the set $T = 1,\ldots,t$, where $t = |D_\delta|$. Set T represents the ROM cells needed to store the next state pair $\delta(v,s)$ for each pair (v,s). A *characteristic partition* P_c of the FSM is defined in the following way: each block B_{P_c} of the characteristic partition includes these elements from the set T which correspond to these pairs (v,s) from the domain D_δ which the transition function $\delta(v,s) = s'$ maps onto the same next state s'.

A partition P on the set T is related to a partition π on the states set S if for any inputs v_a, v_b the condition that s_i, s_j belong to one block of the partition π implies that the elements from T corresponding to pairs (v_a,s_i) and (v_b,s_j) belong to one block of the partition P. A partition P on set T is related to a partition θ on the input symbols set V if for any state s_a, s_b the condition that v_i, v_j belong to one block of the partition implies that the elements from T corresponding to pairs (v_i,s_a) and (v_j,s_b) belong to one block of the partition P. In particular, a partition P on set T is related to the set $\{\pi,\theta\}$ if it is related to both π and θ.

Let P_a and P_b be partitions on the set T, and $P_a \geq P_b$. Then a partition $P_a|P_b$, whose elements are blocks of P_b and whose blocks are those of P_a, is a *quotient partition* of P_a over P_b.

For a partition $P_a \geq P_b$ let $P_a|P_b$ denote the quotient partition and let $\varepsilon(P_a|P_b)$ be the number of elements in the largest block of partition $P_a|P_b$. Let $e(P_a|P_b)$ be the smallest integer equal to or greater than $\log_2(\varepsilon(P_a|P_b))$ (i.e., $e(P_a|P_b) = \lceil \log_2(\varepsilon(P_a|P_b)) \rceil$). Then, the notion of *r-admissibility* of the two-block partitions' set $\{P_1,\ldots,P_k\}$ on S in relation to the partition P on S, is defined as $r = k + e(\sigma|\rho)$, where σ is the product of $\{P_1,\ldots,P_k\}$ and ρ is the product of σ and P.

Let $\Pi = \{\pi_1,\ldots,\pi_p\}$ is the set of two-block partitions on S and $\Theta = \{\theta_1,\ldots,\theta_m\}$ is the set of two-block partitions on V, while P_k is a partition on the set T which is related to either π_i or θ_j. Then, $\mathbb{p} = \{P_1,\ldots,P_{m+p}\}$ is the set of all partitions related to partitions $\{\pi_1,\ldots,\pi_p,\theta_1,\ldots,\theta_m\}$. Partitions in Π correspond to the state variables and partitions in Θ correspond to the input variables.

Fact. To achieve unambiguous encoding of address variables and at the same time maintain the consistency relation $\mathbb{T}$ with the transition function, two-block partitions $\mathbb{P} = \{P_1,\ldots,P_w\}$ have to be found, such that:

$$P_1 \cdot P_2 \cdot \ldots \cdot P_w \leq P_c. \tag{8.7}$$

This is a necessary and sufficient condition for $\{P_1, \ldots, P_w\}$ to determine the address variables. This is because each memory cell is associated with a single block of P_c, i.e., with those elements from T which map the corresponding (v, s) pairs onto the same next state.

Although some of the partitions for the $\mathbb{P}$ set can be selected from the $\mathbb{p}$ set, the selection is made in such a way that the simplest addressing unit (address modifier) is produced. Such a selection is possible thanks to the method [9], based on the notion of r-admissibility.

Assume that u partitions $\{\pi_1, \ldots, \pi_l\}$ and $\{\theta_1, \ldots, \theta_{u-l}\}$ were chosen. These partitions correspond to the address lines driven by a single variable, either a state variable q or an external variable x. The result is the state and input symbol partial encoding; i.e.,

$$a_1 = q_1, \ldots, a_l = q_l, a_{l+1} = \theta_1, \ldots, a_u = \theta_{u-l}.$$

This encoding of state variables is possible thanks to the method of construction and coloring weighted graphs.

Then, inequality (8.7) can be written as:

$$P_{i_1} \cdot P_{i_2} \cdot \ldots \cdot P_{i_u} \cdot P_{i_{u+1}} \cdot \ldots \cdot P_{i_w} \leq P_c, \tag{8.8}$$

where $P_U = P_{i_1} \cdot P_{i_2} \cdot \ldots \cdot P_{i_u}$ is related to the partitions $\{\pi_1, \pi_2, \ldots, \pi_l, \theta_1, \theta_2, \ldots, \theta_{u-l}\}$.

The encoding of the part of the state variables remaining after the partial encoding (input variables, in general) can be obtained from the following rules:

$$\begin{aligned} \pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_l \cdot \pi &= \pi(\mathbf{0}), \\ \theta_1 \cdot \theta_2 \cdot \ldots \cdot \theta_{u-l} \cdot \theta &= \theta(\mathbf{0}), \end{aligned}$$

where π and θ represent partitions corresponding to these remaining state variables. $\pi(\mathbf{0})$ as well as $\theta(\mathbf{0})$ are partitions whose blocks are equal to their elements.

Since the design process may be considered as a decomposition of the memory block into two blocks: a combinational address modifier and a smaller memory block, we need to find function G which will determine the second part of the memory address bits.

Inequality (8.8) can be transformed into:

$$P_U \cdot P_G \leq P_c. \tag{8.9}$$

Now, a partition P_G has to be constructed, such that:

$$P_G \geq P_V, \tag{8.10}$$

where $P_G = P_{i_{u+1}} \cdot \ldots \cdot P_{i_w}$ and P_V is related to the partition set $\{\pi, \theta\}$. Let us assume that input variables are encoded.

Theorem. Partition P_V can be constructed in the following way:

$$P_V = P_S \cdot P_{V_\theta}, \tag{8.11}$$

where P_S is the partition related to $\pi(\mathbf{0})$ on the set of states S, and P_{V_θ} is the partition related to θ.

Proof. Let us assume that $P_V = P_{V_\pi} \cdot P_{V_\theta}$, where P_{V_π} is related to π, and P_{V_θ} is related to θ. Since P_U and P_V satisfy $P_U \cdot P_V \leq P_c$, we have $P_U \cdot P_{V_\pi} \cdot P_{V_\theta} \leq P_c$. As a result, $P_U \cdot P_S \cdot P_{V_\theta} \leq P_c$.

Let $\langle V, R, E, P\rangle$ be a quadruple where: V – set of elements, R – an equivalence relation on the set V, E – set of pairs in relation P on the set V, P – two-element relation. A triple $M(V|R, E, P)$ is a *multi-graph*, where $V|R$ – is an equivalence class for an equivalence relation on the set V. Since there exists an isomorphism $V|R \leftrightarrow V'$, we can construct a natural mapping from the multi-graph $M(V|R, E, P)$ to the graph $G(V', E', P)$. This mapping $\psi: M \rightarrow G$ allows for calculation of a chromatic number $\chi(G) = \chi(M)$.

Inequality (8.9) allows us to construct a quotient partition $P_U|P_c$. Then, the triple $\langle P_V, E_1, P_1\rangle$, where: P_V is a partition given by equation (8.11), P_1 is a relation which represents incompatibilities in quotient partition $P_U|P_c$ on the set T (relation of incompatibility in quotient partition $P_U|P_c$ is a relation among all elements in each block of the partition separately) and E_1 is the set of pairs in the relation P_1; is a multi-graph $M_1(P_V, E_1, P_1)$.

After mapping $\psi_1: M_1 \rightarrow G_1$ we calculate a chromatic number $\chi(G_1)$ which is equal to $\chi(M_1)$. The coloring of the graph G_1 determines the P_G partition. The value of

$$\mu = |U| + \lceil \log_2(\chi(M_1)) \rceil \tag{8.12}$$

determines memory size required.

In case of $\mu > w$, a new partition P'_V has to be constructed. Then, P_V has to be multiplied by appropriately chosen two-block partitions related to those which are generated by input variables from the set U. In that case, we obtain a non-disjoint decomposition [8].

In the next step we calculate the remaining state variables. The triple $\langle P_S, E_2, P_2\rangle$, where: P_S is the partition related to $\pi(\mathbf{0})$ on the states set S, P_2 is a relation which represents incompatibilities in quotient partition $P_{V_\theta}|P_G$ and E_2 is the set of pairs in the relation P_2; is a multi-graph $M_2(P_S, E_2, P_2)$.

Similarly to the case discussed above, by coloring an image graph G_2 for the multi-graph M_2, we obtain a new partition on the set S. We encode this partition with the minimal binary code. Value $\lceil \log_2(\chi(M_2)) \rceil$ determines the number of bits needed to encode the remaining state variables and value

$$\nu = |V_\theta| + \lceil \log_2(\chi(M_2)) \rceil \tag{8.13}$$

determines the number of inputs to address modifier.

The issue of finite state machine functional decomposition for FPGA structures with embedded memory blocks has been successfully implemented in university software FSM*dec* [46].

Applying the software to the chemical reactor from Fig. 8.4 transformed to KISS format (Fig. 8.13) we have obtained the content of the address modifier and ROM, as well as state code assignment (Fig. 8.14).

```
.i 10
.o 9
.p 263
.s 33
.r strst
(...)
---1---0-- s7   s6 000001000
-------0-1 s8   s6 000001000
---1---0-1 s9   s6 000001000
-------1-- s6  s10 000000000
---1---1-- s7  s10 000000000
-------1-1 s8  s10 000000000
---------1 s11 s10 000000000
(...)
```

Fig. 8.13 Chemical reactor from Fig. 8.4 transformed to KISS format

a)

```
.i 13
.o 2
.p 57
.type fr
(...)
1-------01111 00
-----0-000111 00
01----0--0000 00
01------10001 00
-0----0-100-- 01
-0------100-1 01
-0------1001- 01
(...)
```

b)

```
.i 9
.o 19
.p 87
.type fr
(...)
-00001101 0001100101000011000
-00110101 0001100101000011000
--0000110 0011010011000000000
--0001001 0011010011000000000
-1000-110 0011010011000000000
-10010101 0011010011000000000
-10110111 0011010011000000000
(...)
```

c)

```
(...)
s7  := 0001100101
s8  := 0000100110
s9  := 0110100111
s10 := 0011010011
s11 := 0001001000
s12 := 0010101000
s13 := 0101001001
(...)
```

Fig. 8.14 Content of address modifier (a), ROM (b), and state code assignment (c); after functional decomposition of chemical reactor

In Fig. 8.15 we provide results for FSM from Fig. 8.9.

a)

```
.i 5
.o 2
.p 14
.type fr
--000 00
0-0-0 00
0101- 00
11101 00
--001 01
000-1 01
10-01 01
0-100 10
0010- 10
10011 10
1-010 11
1-100 11
1101- 11
01101 11
.e
```

b)

```
.i 4
.o 6
.p 11
.type fr
11-- 000100
1-1- 000100
-000 000100
000- 000100
1001 110100
0110 110100
0100 101011
0011 101011
0101 101110
0111 010001
0010 010001
.e
```

c)

```
start  := 0000
000010 := 0001
001101 := 1010
011001 := 1011
100101 := 0100
110001 := 1101
```

Fig. 8.15 Content of address modifier (a), ROM (b), state code reassignment (c)

8.4.3 Synthesis Based on Architectural Decomposition

Other method which cut down the hardware implementation resources for FSMs is architectural decomposition [1, 3]. In this method, the FSM circuit is implemented in a double- or multi-level structure. The circuit of single-level is a combinational circuit that implements Boolean functions of the decomposed FSM. In comparison to the single-level circuit the gain on this circuit is that it implements less Boolean functions and typically requires less look-up tables for its implementation. The second-level circuit works as a decoder. Functions describing its behavior have a regular structure. It means that it can be implemented into new FPGA devices with the use of embedded memory blocks. Overall, such a circuit requires less logic elements but additional memory resources, although memories in FPGAs are very often not used for any other purpose.

Special methods of encoding [3] and modifications of a logic circuit structure are applied. Presented method is based on multiple encoding of internal states [12] and microinstructions [11]. The encodings are performed independently to each other and they can be applied both together [13] or separately [11, 12]. In the former solution the current state is used as partitioning set. In this case, the code of a

microinstruction $K(Y_t)$ is represented by concatenation of the multiple code of the microinstruction $K_s(Y_t)$ and the code of the current state $K(s)$:

$$K(Y_t) = K_s(Y_t) * K(s). \tag{8.14}$$

The code of the internal state $K(s')$ is represented by concatenation of the multiple code of the internal state $K_s(s')$ and the code of the current state $K(s)$:

$$K(s') = K_s(s') * K(s). \tag{8.15}$$

A digital circuit of a FSM with such encodings can be implemented as a double-level structure (Fig. 8.16) called PAY_0 [13]. In this structure, the combinational circuit implements logic functions that encode microinstructions:

$$\lambda_1 : X \times Q \to \Psi, \tag{8.16}$$

and internal states:

$$\delta_1 : X \times Q \to \text{T}, \tag{8.17}$$

where Q is the set of variables storing the code of current state, and $|Q| = p = \lceil \log_2 |S| \rceil$. T is the set of variables storing the code of internal state, and $|\text{T}| = p_1$. Ψ is the set of variables storing the code of microinstruction, and $|\Psi| = r_1$. The instruction decoder implements a function of a decoder of microinstructions:

$$\lambda_2 : \Psi \times Q \to Y \tag{8.18}$$

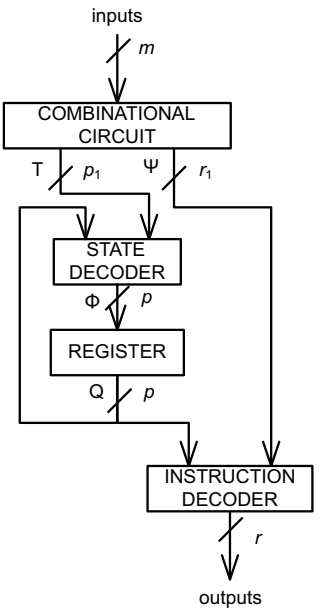


Fig. 8.16 The structural diagram of PAY_0 Mealy FSM

and functions λ_1 and λ_2 yield the function λ of mathematical transformations. The state decoder decodes internal states and generates excitation function:

$$\delta_2 : \mathrm{T} \times Q \rightarrow \Phi \tag{8.19}$$

and, in similar way, functions δ_1 and δ_2 yield the function δ of mathematical transformations. These mathematical transformations proof that application of architectural decomposition do not change the behavior of FSM. The register is build with D-type flip-flops.

The starting point for synthesis process with architectural decomposition is the KISS file obtained from transformation of the statechart, and it consists of the following steps:

- multiple encoding of microinstructions and internal states,
- formation of the transformed table,
- formation of the system of Boolean functions,
- formation of the instruction decoder table,
- formation of the state decoder table,
- implementation of the logic circuit of the FSM.

The multiple encoding of microinstructions and internal states is based on binary encoding of microinstructions and internal states in each subset.

In the case of chemical reactor example (Fig. 8.4, Fig. 8.13), there are 33 states. It means that there are 33 subsets of microinstructions and 33 subsets of internal states. There are maximum 16 microinstructions in one subset. It means that for encoding only 4 bits are required. For example, for the subset based on the state $s8$, the following encoding is obtained:

$K_{s8}(000001000) = 0000,$
$K_{s8}(000000000) = 0001,$
$K_{s8}(000001100) = 0010,$
$K_{s8}(000000100) = 0011.$

Similar situation is obtained for internal states.

The formation of the transformed table is the base for forming system of Boolean functions. It is created from the original table (described in the KISS file) by replacing the state, the microinstruction and the internal state with their codes. Part of transformed KISS table is shown in Fig. 8.17.

The formation of the system of Boolean functions is the base for obtaining Boolean functions. These systems are obtained from the transformed table, in a classical way [1]. For example:

$$\psi_1 = x_1x_2x_3x_4\bar{x_9}\bar{Q_1}Q_2\bar{Q_3}\bar{Q_4}Q_5\bar{Q_6} + x_1\bar{x_2}\bar{x_3}x_4\bar{x_9}\bar{Q_1}Q_2\bar{Q_3}Q_4Q_5\bar{Q_6}$$
$$+ x_1x_2\bar{x_3}x_4\bar{x_9}\bar{Q_1}Q_2\bar{Q_3}Q_4Q_5\bar{Q_6} + x_1x_2x_3x_4\bar{x_9}\bar{Q_1}Q_2\bar{Q_3}Q_4Q_5\bar{Q_6}$$
$$+ \bar{x_1}x_2x_3x_4\bar{x_9}\bar{Q_1}Q_2\bar{Q_3}\bar{Q_4}Q_5\bar{Q_6} + \bar{x_1}\bar{x_2}\bar{x_3}x_4\bar{x_9}\bar{Q_1}Q_2\bar{Q_3}Q_4Q_5\bar{Q_6}$$
$$+ \ldots$$

```
.i 10
.o 9
.p 263
.s 33
.r strst
(...)
 ---1---0-- 000110 0000 0000
 -------0-1 000111 0001 0000
 ---1---0-1 001000 0000 0000
 -------1-- 000101 0001 0001
 ---1---1-- 001001 0001 0001
 -------1-1 000111 0001 0001
 ---------1 001010 0000 0001
(...)
```

Fig. 8.17 Example of transformed KISS description

The formation of the instruction decoder table. This step forms the table that describes the behavior of instruction decoder. This table has four columns:

- binary code of the current state;
- binary code of the microinstruction (from adequate subset);
- binary representation of the microinstruction (from adequate subset);
- number of the line.

Table 8.2 shows part of such table for our example.

Table 8.2 The part of the instruction decoder table

$K(s)$	$K_s(Y_t)$	Y_t	h
000000	0000	000000000	1
000001	0000	010000000	2
000001	0001	000000000	3
000010	0000	010000000	4
000010	0001	000100000	5

The formation of the state decoder table. This step forms the table that describe behavior of the state decoder. This table has four columns:

- binary code of the current state;
- binary code of the internal state (from adequate subset);
- binary representation of excitation functions that switches the memory of the FSM;
- number of the line.

Table 8.3 The part of the state decoder table

$K(s)$	$K_s(s')$	D	h
000000	0000	000011	1
000001	0000	000001	2
000001	0001	000011	3
000010	0000	000001	4
000010	0001	000010	5

Table 8.3 shows part of such table for the presented example.

The implementation of the logic circuit of the FSM. The combinational circuit and the register are implemented with CLBs – LUTs and D-type flip-flops. The instruction decoder is implemented using embedded memory blocks with $2^{(p+r_1)}$ words of r bits. The content of the memory is described by the instruction decoder table, where concatenation of the binary code of the current state and the binary code of a microinstruction (Fig. 8.14) is the memory address, and the binary representation of a microinstruction is the value of the memory word. The state decoder is also implemented in an embedded memory block with $2^{(p+p_1)}$ words of p bits. The content of the memory is described by the state decoder table, where concatenation of the binary code of the current state and the binary code of the internal state (Fig. 8.15) is the memory address, and the binary representation of excitation functions is the value of the memory word. Any (*don't care*) value can be assigned for addresses missing in both tables, since such concatenations of codes for both memories are never used. Both decoders are implemented by memory blocks of an FPGA device working in ROM mode. Schematic diagram for an FPGA technology of multi-level structures is presented in Fig. 8.18.

This diagram is based on Xilinx Spartan and Virtex FPGAs but they can be easy adopted to other FPGAs vendors, like Altera Cyclone and Stratix, since all logic elements, especially memory blocks and their connections, are very similar.

It should be mentioned, that memory blocks in popular FPGAs are synchronous [45, 48]. The clock signal for memory blocks is the same as for the register but memory blocks are trigged by opposite edge (in this case falling edge). It cause that data are ready to read after one cycle and there is no need to wait one clock cycle more until data are stable. It is especially important when an internal state is encoded. It also means that memory blocks can work as an output register in case when microoperations are encoded. Such registers are needed in each digital system with Mealy's outputs to stabilize its operation [2, 25]. Other input signals of memory blocks are connected to logic 1 or logic 0, according to specification of Xilinx Block RAM [48], to satisfy read-only mode.

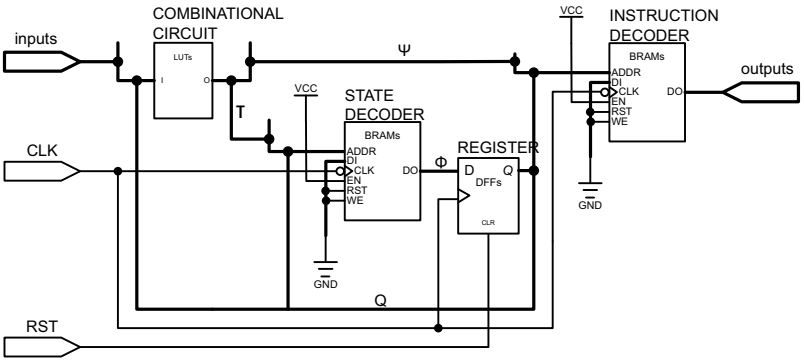


Fig. 8.18 The schematic diagram of PAY_0 Mealy FSM

This is only one possible architecture that could be obtained after architectural decomposition with the application of multiple encoding. The architecture depends on which parameter(s) is(are) encoded and which parameter is used as partition set [14]. The presented architecture yields very good results [13], but the gain in hardware resources consumption is strongly dependent on the characteristics of the control algorithm [3]. It means that architecture and method of encoding should be chosen individually for each algorithm.

8.5 Summary

The increasing complexity of the digital systems and novel advanced technologies have become essential for application of efficient logic synthetic methods as well as digital design and mapping tools. However, the commercial tools usually take into account some trade-off between computational complexity and the quality of the physical implementation of the projects [45, 48]. The diversity of implementations requires the use of advanced methods of logic synthesis, however most existing systems on digital market does not implement such solutions using the outdated synthesis methods. The advancement of logic synthesis methods, the complexity of procedures, and novel digital structures leads to the situation that only computer-aided systems developed mainly at academic research centers are able to support digital design [6, 9, 13, 14, 16, 27, 29, 31, 38, 39, 46, 47].

The idea presented in the chapter is targeted at complex concurrent behavior specified with statechart, which is finally implemented in modern programmable devices equipped with memory blocks and configurable logic. The transformation from statechart diagram into FSM model is carried with symbolic formal methods efficiently implemented by means of Binary Decision Diagrams. Both synthesis strategies (ROM-based scheme with address modifier and architectural decomposition) consume hardware resources to different extent.

ROM-based method uses only memory blocks. Table 8.4 presents the gain in memory bits obtained with functional decomposition scheme and implementation

in the architecture with address modifier and memory (Fig. 8.12b) in comparison to the implementation without address modifier. Decreasing ratio is of the order of tens and the method is especially efficient for more complex behavior. However, the address modifier, as well as the memory content, can easily be implemented in logic cells [38] making the method universal for heterogeneous programmable structures.

Table 8.4 FSM synthesis results before and after introduction of address modifier

name	*before*					*after*	*gain*	*dec.*
	#in	*#out*	*#q*	*#cube*	*#bit*	*#bit*	*%*	*ratio*
Garage	6	3	4	49	7168	1664	77	4.3
TVRemoteControl	8	5	4	55	36864	6912	81	5.3
SimpleReactor	10	15	8	986	6029312	294912	95	20.4
ReactorWithWagon (Fig. 8.4)	10	9	6	263	983040	26112	97	37.6

$$gain = \frac{\#bit_{before} - \#bit_{after}}{\#bit_{before}} \cdot 100\% \qquad decreasing\ ratio = \frac{\#bit_{before}}{\#bit_{after}}$$

Architectural decomposition uses both, memory and configurable logic. According to Figure 8.16, blocks CC and Y are mapped into memory, and blocks P and RG are implemented in LUTs. Table 8.5 presents the gain in LUTs obtained with the application of architectural decomposition in comparison to standard FSM synthesis method which is well known VHDL template utilizing only LUTs [43]. The results were obtained using Xilinx tool with default settings.

Table 8.5 FSM synthesis results with standard method and as PAY_0 Mealy FSM

name	*Standard method*		PAY_0			*LUT gain*	*dec.*
	#LUT	*#FF*	*#LUT*	*#FF*	*#BRAM*	*%*	*ratio*
Garage	82	14	10	4	2	88	8.2
TVRemoteControl	102	12	28	4	2	73	3.6
SimpleReactor	1965	163	472	13	23	77	4.2
ReactorWithWagon (Fig. 8.4)	423	33	46	6	5	89	9.2

$$gain = \frac{\#LUT_{std\_method} - \#LUT_{PAY_0}}{\#LUT_{std\_method}} \cdot 100\% \qquad decreasing\ ratio = \frac{\#LUT_{std\_method}}{\#LUT_{PAY_0}}$$

The idea of functional decomposition which is the base for address modifier concept can be applied to any function [31]. Its application to functions implemented in blocks P, Y and CC can bring further reductions in hardware resources, not only memory bits, but also in configurable logic.

Decomposition methods proved their efficiency for the latest programmable devices. It seems that combining architectural synthesis with functional ROM-based decomposition is very promising for logic controller design, especially for structures equipped with embedded memory blocks.

Acknowledgements. The research was partly financed from budget resources intended for science in 2010 – 2013 as an own research project No. N N516 513939.

References

1. Adamski, M., Barkalov, A.: Architectural and sequential synthesis of digital devices. University of Zielona Góra, Zielona Góra (2006)
2. Baranov, S.I.: Logic synthesis for control automat. Kluwer Academic Publishers, Boston (1994)
3. Barkalov, A., Titarenko, L.: Logic synthesis for FSM-based control units, Lecture Notes in Electrical Engineering, vol. LNEE, vol. 53. Springer, Heidelberg (2009)
4. Bazydło, G., Adamski, M.: Logic controllers design from UML state machine diagrams. Czasopismo Techniczne: Informatyka z. 24, 3–18 (2008) (in Polish)
5. Borowik, G., Rawski, M., Łabiak, G., Bukowiec, A., Selvaraj, H.: Efficient logic controller design. In: Fifth International Conference on Broadband and Biomedical Communications (IB2Com), pp. 1–6 (December 2010), doi:10.1109/IB2COM.2010.5723633
6. Borowik, G.: Improved state encoding for FSM implementation in FPGA structures with embedded memory blocks. Electronics and Telecommunications Quarterly 54(1), 9–28 (2008)
7. Borowik, G., Kraśniewski, A.: Trading-off error detection efficiency with implementation cost for sequential circuits implemented with FPGAs. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST 2011, Part II. LNCS, vol. 6928, pp. 327–334. Springer, Heidelberg (2012)
8. Borowik, G., Łuba, T., Tomaszewicz, P.: On memory capacity to implement logic functions. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST 2011, Part II. LNCS, vol. 6928, pp. 343–350. Springer, Heidelberg (2012)
9. Brzozowski, J.A., Łuba, T.: Decomposition of boolean functions specified by cubes. Journal of Multi-Valued Logic and Soft Computing 9, 377–417 (2003)
10. Buchenrieder, K., Pyttel, A., Veith, C.: Mapping statechart models onto an FPGA-based ASIP architecture. In: EURO-DAC 1996, pp. 184–189 (September 1996), doi:10.1109/EURDAC.1996.558203
11. Bukowiec, A.: Synthesis of FSMs based on architectural decomposition with joined multiple encoding. International Journal of Electronics and Telecommunications 58(1), 35–41 (2012), doi:10.2478/v10177-012-0005-7
12. Bukowiec, A., Barkalov, A., Titarenko, L.: Encoding of internal states in synthesis and implementation process of automata into FPGAs. In: Proceedings of the Xth International Conference on the Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2009, pp. 199–201. Ministry of Education and Science of Ukraine and Lviv Polytechnic National University, Lviv, Publishing House Vezha & Co., Polyana, Ukraine (2009) ISBN: 978-966-2191-05-9
13. Bukowiec, A.: Synthesis of finite state machines for FPGA devices based on architectural decomposition. Lecture Notes in Control and Computer Science, vol. 13 (2009)

14. Bukowiec, A., Barkalov, A.: Structural decomposition of Finite State Machines. Electronics and Telecommunications Quarterly 55(2), 243–267 (2009)
15. Czerwiński, R., Kania, D.: State assignment for PAL-based CPLDs. In: Wolinski, C. (ed.) Proc. of 8th Euromicro Conference on Digital Systems Design, Architectures, Methods and Tools, Porto, Portugal, August 30-September 3, pp. 127–134. IEEE Computer Society (2005), doi:10.1109/DSD.2005.71
16. Doligalski, M.: Behavioral specification diversification for logic controllers implemented in FPGA devices. In: Proceedings of the 9th Annual FPGA Conference - FPGAworld 2012, Stockholm, Sweden, pp. 6:1–6:5. ACM (2012), doi:10.1145/2451636.2451642
17. Drusinsky, D., Harel, D.: Using statecharts for hardware description and synthesis. IEEE Transaction on Coputer-Aided Design 8(7), 798–807 (1989), doi:10.1109/43.31537
18. Drusinsky-Yoresh, D.: A state assignment procedure for single-block implementation of state chart. IEEE Transaction on Coputer-Aided Design 10(12), 1569–1576 (1991), doi:10.1109/43.103506
19. Gajski, D.D., Vahid, F., Narayan, S., Gong, J.: Specification and design of embedded systems. Prentice-Hall, Inc., Upper Saddle River (1994)
20. Gomes, L., Costa, A.: From use cases to system implementation: statechart based codesign. In: Proceedings of 1st ACM & IEEE Conference on Formal Methods and Programming Models for Codesign, MEMOCODE 2003, Mont Saint-Michel, France, pp. 24–33. IEEE Computer Society Press (2003), doi:10.1109/MEMCOD.2003.1210083
21. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987), doi:10.1016/0167-6423(87)90035-9
22. Hartmanis, J., Stearns, R.E.: Algebraic structure theory of sequential machines. Prentice-Hall, New York (1966)
23. Henson, M.A.: Biochemical reactor modeling and control. IEEE Control Systems Magazine 26(4), 54–62 (2006), doi:10.1109/MCS.2006.1657876
24. I-Logix Inc., 3 Riverside Drive, Andover, MA 01810 U.S.A.: STATEMATE Magnum Code Generation Guide (2001)
25. Jenkins, J.H.: Designing with FPGAs and CPLDs. Prentice Hall, Upper Saddle River (1994)
26. Karatkevich, A.: Deadlock analysis in statecharts. In: Forum on Specification on Design Languages (2003)
27. Łabiak, G.: From UML statecharts to FPGA - the HiCoS approach. In: Proceedings of Forum on Specification & Design Languages, FDL 2003, pp. 354–363. Frankfurt am Main (September 2003)
28. Łabiak, G.: The use of hierarchical model of concurrent automaton in digital controller design. University of Zielona Góra Press, Poland, Zielona Góra (April 2005) (in Polish)
29. Łabiak, G.: From statecharts to FSM-description – transformation by means of symbolic methods. In: 3rd IFAC Workshop Discrete-Event System Design, DESDes 2006, Poland, pp. 161–166 (2006), doi:10.3182/20060926-3-PL-4904.00027
30. Łabiak, G., Borowik, G.: Statechart-based controllers synthesis in FPGA structures with embedded array blocks. Intl Journal of Electronic and Telecommunications 56(1), 11–22 (2010), doi:10.2478/v10177-010-0002-7
31. Łuba, T., Borowik, G., Kraśniewski, A.: Synthesis of Finite State Machines for implementation with programmable structures. Electronics and Telecommunications Quarterly 55(2) (2009)
32. de Micheli, G.: Symbolic design of combinational and sequentional logic circuits implemented by low-level logic macros. IEEE Transactions on CAD CAD-5(4), 597–616 (1986), doi:10.1109/TCAD.1986.1270230

33. de Micheli, G.: Synthesis and optimization of digital circuits. McGraw-Hill Higher Education (1994)
34. de Micheli, G., Brayton, R.K., Sangiovanni-Vincentelli, A.: Optimal state assignment for finite state machines. IEEE Transactions on CAD CAD-4(3), 269–284 (1985), doi:10.1109/TCAD.1985.1270123
35. Minato, S.: Binary decision diagrams and applications for VLSI CAD. Kluwer Academic Publishers, Boston (1996)
36. Ramesh, S.: Efficient translation of statecharts to hardware circuits. In: Twelfth International Conference on VLSI Design, pp. 384–389 (January 1999), doi:10.1109/ICVD.1999.745186
37. Rawski, M., Selvaraj, H., Łuba, T.: An application of functional decomposition in ROM-based FSM implementation in FPGA devices. Journal of Systems Architecture 51, 424–434 (2005), doi:10.1016/j.sysarc.2004.07.004
38. Rawski, M., Tomaszewicz, P., Borowik, G., Łuba, T.: 5 Logic synthesis method of digital circuits designed for implementation with embedded memory blocks of FPGAs. In: Adamski, M., Barkalov, A., Węgrzyn, M. (eds.) Design of Digital Systems and Devices. LNEE, vol. 79, pp. 121–144. Springer, Heidelberg (2011)
39. Sasao, T.: On the number of LUTs to realize sparse logic functions. In: Proc. of the 18th International Workshop on Logic and Synthesis, Berkeley, CA, U.S.A., July 31-August 2, pp. 64–71 (2009)
40. Szecówka, P.M., Pedzińska-Rżany, J., Wolczowski, A.R.: Hardware approach to artificial hand control based on selected DFT points of myopotential signals. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST 2009. LNCS, vol. 5717, pp. 571–578. Springer, Heidelberg (2009)
41. Villa, T., Sangiovanni-Vincentelli, A.: NOVA: state assignment of finite state machines for optimal two-level logic implementation. IEEE Transactions on CAD 9(9), 905–924 (1990), doi:10.1109/43.59068
42. Yang, S.: Logic synthesis and optimization benchmarks User Guide Version 3.0. Tech. rep., Microelectronics Center of North Carolina, P.O. Box 12889, Research Triangle Park, NC 27709 (1991)
43. Zwolinski, M.: Digital system design with VHDL. Prentice Hall (2004)
44. Zydek, D., Selvaraj, H., Borowik, G., Luba, T.: Energy characteristic of processor allocator and network-on-chip. Journal of Applied Mathematics and Computer Science 21(2), 385–399 (2011), doi:10.2478/v10006-011-0029-7
45. Altera: Embedded Memory in Altera FPGAs (2010), `http://www.altera.com/technology/memory/embedded/mem-embedded.html`
46. FSMdec: FSMdec Homepage (2012), `http://gborowik.zpt.tele.pw.edu.pl/node/58`
47. HiCoS: HiCoS Homepage (2012), `http://www.uz.zgora.pl/%7eglabiak`
48. Xilinx: Block RAM (BRAM) Block (v1.00a), San Jose (August 2004), `http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf`