State Machines Synthesis and Implementation into FPGAs with Multiple Encoding of States

Arkadiusz Bukowiec, Alexander Barkalov, and Larysa Titarenko

***Abstract* — The method of synthesis and implementation into FPGAs (Field Programmable Gate Arrays) of Mealy FSMs (Finite State Machines) is proposed. Synthesis is based on the architectural decomposition and the multiple encoding. A set of states is divided into subsets based on a current state or a executed microinstruction. Then, states are encoded separately in each subset. The state is decoded in the second-level circuit based on the multiple code and the code of a current state or the code of a executed microinstruction. It leads to implementation of an FSM in double-level structure where utilization of both, LUTs (Look-Up Tables) and embedded memory blocks, is applied. It leads to balanced usage of hardware resources of an FPGA device.**

***Index Terms* — Circuit synthesis, Field programmable gate arrays, Finite state machines, Logic design**

I. Introduction

FINITE state machines (FSMs) with Mealy's outputs [1] are one of the most popular model used in designing control units (CUs) of digital systems. Nowadays field programmable gate arrays (FPGAs) are used very often for implementation of such digital systems [6]. One of the main features of FPGAs is existence of logic elements with restricted number of inputs that are named look-up tables (LUTs) [4]. From another side, logic functions of FSMs (called p-functions) have much more arguments (up to 200) than typical LUTs have inputs (up to 6). This imbalance leads to need of a functional decomposition of Boolean functions describing the behavior of an FSM [5]. The negative result of functional decomposition is increasing a number of levels of the logic circuit of an FSM and increasing a number of required LUTs for a implementation.

On the other hand, new FPGAs are equipped in embedded memory blocks [9]. These blocks can be also used for realization of combinational circuits. The problem is that implementation only with memory blocks also utilize a big number of such blocks and very often exceed the number of available blocks in an FPGA device.

One of methods of decreasing a number of p-functions depending on a big number of arguments is an architectural decomposition of an FSM [2]. Such methods apply encoding of some parameters of an FSM. It leads to implementation of an FSM in a double-level structure where a reduced number of p-functions is realized in the circuit of first level, this circuit is implemented with LUTs, and the circuit of second level operates as a decoder and it is implemented with memory blocks.

The proposed in this article method of synthesis is based on the encoding of internal states divided into subsets based on a current state or a currently executed microinstruction [3]. This encoding allows to decrease a number of p-functions implemented by the combinational circuit of an FSM. The state is decoded in the second level circuit based on the multiple code and the code of a current state or the code of a currently executed microinstruction. Because this system is regular it can be implemented with embedded memory blocks. It leads to decrease a number of LUT elements required for implementation of a logic circuit of an FSM and balanced usage of different resources of an FPGA device.

II. Finite State Machine Definition

A finite state machine is a mathematical model of behavior composed of a finite set of input symbols, a finite nonempty set of states, a finite set of output symbols, transitions and actions [1], [2]. This model can be represented as six tuple:

$$S = \langle X, Y, A, a_1, \delta, \omega \rangle, \tag{1}$$

where:

- X is a finite set of input Boolean variables, $X = \{x_1, \dots, x_L\}$;
- Y is a finite set of output Boolean variables, called microoperations (μO), $Y = \{y_1, \dots, y_N\}$;
- A is a finite, nonempty set of states, $A = \{a_1, \dots, a_M\}$;
- a_1 is the initial state of the FSM, $a_1 \in A$;
- δ is a transition function, defined as a function of a state and affirmation or negation of some input variables:
$$\delta : A \times X \to A; \tag{2}$$

Manuscript received December 22, 2008.

A. Bukowiec is with the Institute of Computer Engineering and Electronics, University of Zielona Góra, Podgórna 50, 65-246 Zielona Góra, Poland (phone: +48 68 328 2304; fax: +48 68 324 4733; e-mail: a.bukowiec@iie.uz.zgora.pl).

A. Barkalov is with the Institute of Computer Engineering and Electronics, University of Zielona Góra, Podgórna 50, 65-246 Zielona Góra, Poland (e-mail: a.barkalov@ iie.uz.zgora.pl).

L. Titarenko is with the Institute of Computer Engineering and Electronics, University of Zielona Góra, Podgórna 50, 65-246 Zielona Góra, Poland (e-mail: l.titarenko@ iie.uz.zgora.pl).

- ω is a output function, and in case of Mealy model it is defined as a function of a state and affirmation or negation of some input variables:
  $$\omega : A \times X \to Y. \quad (3)$$
  In case of Moore model it is defined only as a function of a state:
  $$\omega : A \to Y. \quad (4)$$

Such defined a Mealy FSM can be set up by a direct structural table (DST) [1] with columns: a_m, $K(a_m)$, a_s, $K(a_s)$, X_h, Y_h, Φ_h, h. Here a_m is a current state of an FSM, $a_m \in A$; $K(a_m)$ is a binary code of the state a_m with $R = \lceil \log_2 M \rceil$ bits, the internal Boolean variables $q_r \in Q = \{q_1,\ldots,q_R\}$ are used to encode states a_m; a_s is the next state, $a_s \in A$; $K(a_s)$ is a code of the state a_s, $K(a_s) = K(a_m)$ for $s = m$; X_h is a condition of transition $\langle a_m, a_s \rangle$, it consists from conjunction of affirmation or negation of some logic elements from the set X; Y_h is the microinstruction (μI) which is formed during the transition $\langle a_m, a_s \rangle$, $Y_h \subseteq Y$; Φ_h is the set of memory excitation functions that are equal to 1 to switch an FSM from $K(a_m)$ to $K(a_s)$, $\Phi_h \subseteq \Phi = \{D_1,\ldots,D_R\}$ as a rule D flip-flops are used to form a memory; h is a number of the DST line, $h = 1,\ldots,H$.

III. Base Structures of FSM Logic Circuit

The DST table is used as the base to form the system of functions:
$$\begin{aligned} Y &= Y(Q,X), \\ \Phi &= \Phi(Q,X). \end{aligned} \quad (5)$$
This systems corresponds to functions (3) and (2) and it describes a single-level circuit of Mealy FSM (Fig. 1). This structure is called P Mealy FSM. Here the circuit P implements system of functions (5), the register RG represents the memory of FSM. One of the drawbacks of the structure P is a big number of p-functions:
$$n_p(\mathrm{P}) = N + R. \quad (6)$$

One of the known methods of decreasing this parameter is an encoding of microinstructions [2]. Let DST contain T different microinstructions $Y_t \subseteq Y$. Assign to each set Y_t the binary code $K(Y_t)$ with $R_1 = \lceil \log_2 T \rceil$ bits ($t = 1,\ldots,T$). Use variables $z_r \in Z = \{z_1,\ldots,z_{R_1}\}$ for representation of these codes. In this case a Mealy FSM can be implemented as double-level circuit (Fig. 2) named as PY Mealy FSM [2]. The register RG is exactly the same as in previous structure. The circuit Y implements the system of functions:
$$Y = Y(Z) \quad (7)$$
and transforms the code $K(Y_t)$, represented by variables z_r, into the microinstruction Y_t, built from microoperations y_n. This circuit can be implemented using embedded memory blocks. Now the circuit P implements systems:
$$\begin{aligned} Z &= Z(Q,X), \\ \Phi &= \Phi(Q,X), \end{aligned} \quad (8)$$
and the number of p-functions is decreased to:
$$n_p(\mathrm{PY}) = N + R_1. \quad (9)$$
But this number is still relatively big. It means that such a structure needs still relatively big number of LUTs for implementation of the circuit P. It makes that application of this structure in a process of an FPGA implementation is not grateful.

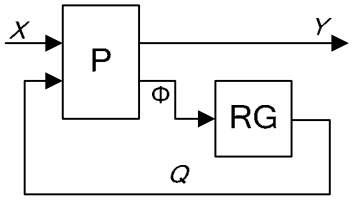

Fig. 1. Structural diagram of P Mealy FSM

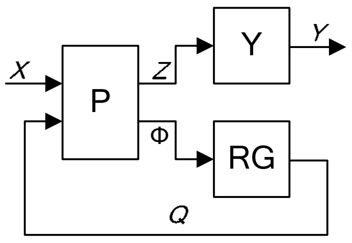

Fig. 2. Structural diagram of PY Mealy FSM

IV. Main Idea of Methods

The idea of further improvement is to encode also the next state (internal state) using the code of a microinstruction or the code of a current state as partial code [3].

A. Multiple Encoding with use of Microinstruction Code

Let divide set of internal states into subsets based on a currently executed microinstruction Y_t. It leads into existence of T subsets $A(Y_t) \subseteq A$ and state $a_s \in A(Y_t)$ iff it is the state of a transition when the microinstruction Y_t is executed. Let $B_t = |A(Y_t)|$ and $B_0 = \max(B_1,\ldots,B_T)$. Encode internal states a_s from each subset $A(Y_t)$ separately by the binary code $K_t(a_s)$ with $R_2 = \lceil \log_2 B_0 \rceil$ bits. This code is represented by variables $\tau_r \in \mathrm{T} = \{\tau_1,\ldots,\tau_{R_2}\}$. In this case the code of the internal state $K(a_s)$ is represented by the concatenation of multiple code of the internal state $K_t(a_s)$ and the code of the microinstruction $K(Y_t)$:
$$K(a_s) = K_t(a_s) * K(Y_t). \quad (10)$$

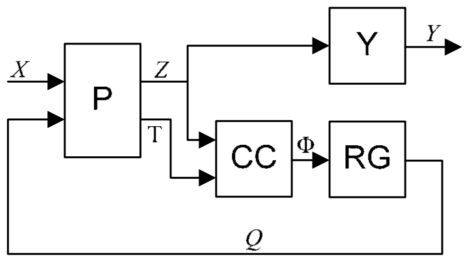

Fig. 3. Structural diagram of PYY Mealy FSM

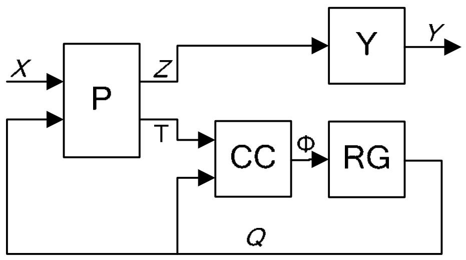

Fig. 4. Structural diagram of PAY Mealy FSM

A digital circuit of an FSM with such an encoding can be implemented as a double-level circuit named as PYY Mealy FSM (Fig. 3). The circuit Y implements the same system (7) like for PY Mealy FSM. The circuit P implements system:

$$Z = Z(Q, X), \quad \mathrm{T} = \mathrm{T}(Q, X), \tag{11}$$

in this case. There is used additional circuit CC in this structure. It is used for decoding internal state and it implements the system:

$$\Phi = \Phi(Z, \mathrm{T}). \tag{12}$$

Because this circuit has regular structure it can be implemented using embedded memory blocks.

This structure permits further reduction of the number of p-functions to:

$$n_p(\mathrm{PYY}) = R_2 + R_1 \tag{13}$$

in comparison with the PY Mealy FSM. It makes that also a number of LUTs required for implementation of the circuit P is reduced and both decoders Y and CC can be implemented with memory blocks what makes that FPGA resources are used in balanced way.

B. *Multiple Encoding with use of State Code*

The method where the code of a current state is used as the partial code of a internal state is very similar to the previous one. In this case the set of internal states is divided into subsets based on a current state a_m. It leads into existence of M subsets $A(a_m) \subseteq A$ and the state $a_s \in A(a_m)$ iff it is the state of the transition from the state a_m. Now, by analogy to the previous method, $C_m = |A(a_m)|$ and $C_0 = \max(C_1, \ldots, C_M)$ and internal states are encoded by the binary code $K_m(a_s)$ with $R_3 = \lceil \log_2 C_0 \rceil$ bits. In this case the code is represented by variables $\tau_r \in \mathrm{T} = \{\tau_1, \ldots, \tau_{R_3}\}$ and the code of the internal state $K(a_s)$ is represented be the concatenation of the multiple code of the internal state $K_m(a_s)$ and the code of the current state $K(a_m)$:

$$K(a_s) = K_m(a_s) * K(a_m). \tag{14}$$

Digital circuit of FSM with this encoding can be implemented as a double-level circuit named as PAY Mealy FSM (Fig. 4). In this structure only the circuit CC implements the different system:

$$\Phi = \Phi(Q, \mathrm{T}) \tag{15}$$

in comparison with PYY Mealy FSM. The circuit P implements system (11), like for the structure PYY, and the circuit Y implements system (7), like for structures PY and PYY.

This structure also permits reduction of the number of p-functions to:

$$n_p(\mathrm{PAY}) = R_3 + R_1 \tag{16}$$

in comparison with the PY Mealy FSM. The rule of realization in FPGA structure is the same as for the structure PYY – the combinational circuit P is implemented with LUTs and both decoders CC and Y are implemented with use of embedded memory blocks.

It is very hard to calculate relation between $n_p(\mathrm{PYY})$ and $n_p(\mathrm{PAY})$ because values of these parameters are strongly connected with FSM parameters (like number of states, number of microinstructions, etc.) of implemented control algorithm and it means that the structure should be selected individually for each case.

V. Method of Synthesis and Implementation

The special method of synthesis for designed structures (Figs. 3 and 4) is proposed. This method includes following steps:

1) Creation and encoding of microinstructions. This step is based on a trivial way of binary encoding. Each microinstruction is assigned a binary code with a value corresponding to the value of its index decreased by 1. So, values of codes are from 0 to $T-1$.
2) Division of the set of internal states. The set of internal states is divided into T or M subsets. Each subset contains only states that are states of a transition during executing the *t*-th microinstruction or from the *m*-th state.
3) Multiple encoding of internal states. Internal states are binary encoded separately in each subset. So, values of codes are from 0 to $B_0 - 1$ or from 0 to $C_0 - 1$. Each state a_s can be assigned several different codes, one in each subset $A(Y_t)$ or $A(a_m)$.
4) Formation of DST of PAY Mealy FSM or PYY Mealy FSM. This table is formed from the original DST by replacing the column Y_h with the column Z_h and

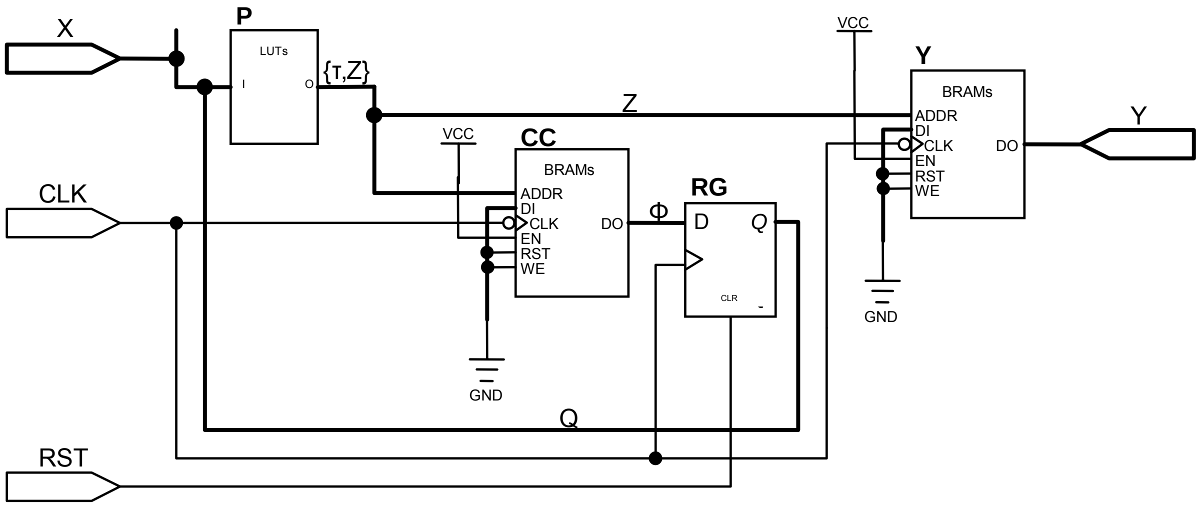

Fig. 5. Schematic diagram of PYY Mealy FSM

columns $K(a_s)$ and Φ_h with columns $K_t(a_s)$ ($K_m(a_s)$) and T_h. The column Z_h contains variables z_r that are equal to 1 in the code $K(Y_t)$. The column $K_t(a_s)$ ($K_m(a_s)$) contains the multiple code of the internal state. The column T_h contains variables τ_r that are equal to 1 in the code $K_t(a_s)$ ($K_m(a_s)$).

5) Formation of microoperations decoder table. This table contains columns $K(Y_t)$, Y_t, t. The column $K(Y_t)$ contains the binary code of the microinstruction from the column Y_t. The column Y_t should be written in a binary format. The column t is a number of the line, $t = 1,\ldots,T$.

6) Formation of internal state code converter table. This table contains columns $K_t(a_s)$ ($K_m(a_s)$), $K(Y_t)$ ($K(a_m)$), $K(a_s)$, i. The column $K_t(a_s)$ ($K_m(a_s)$) contains the multiple code of the internal state a_s for the t-th microinstruction (the m-th state). The t-th microinstruction (the m-th state) is represented by the code from the column $K(Y_t)$ ($K(a_m)$) and the internal state a_s is represented by the code from column $K(a_s)$. The column i is a number of the line, $i = 1,\ldots,\sum_{t=1}^{T} B_t$ ($i = 1,\ldots,\sum_{m=1}^{M} C_m$).

7) Formation of logic equations of the circuit P. These equations form systems Z and T. They are formed basing on the DST of PAY Mealy FSM or PYY Mealy FSM.

8) Implementation of the logic circuit of PAY Mealy FSM or PYY Mealy FSM. The combinational circuit P and the register RG are implemented with CLBs of an FPGA – the circuit P with LUTs and the register RG with D flip-flops.
The circuit Y is implemented with memory blocks, where $K(Y_t)$ is an address and Y_t is a word from this address. The contents of this memory is described by the microoperations decoder table.
The circuit CC is also implemented with memory blocks

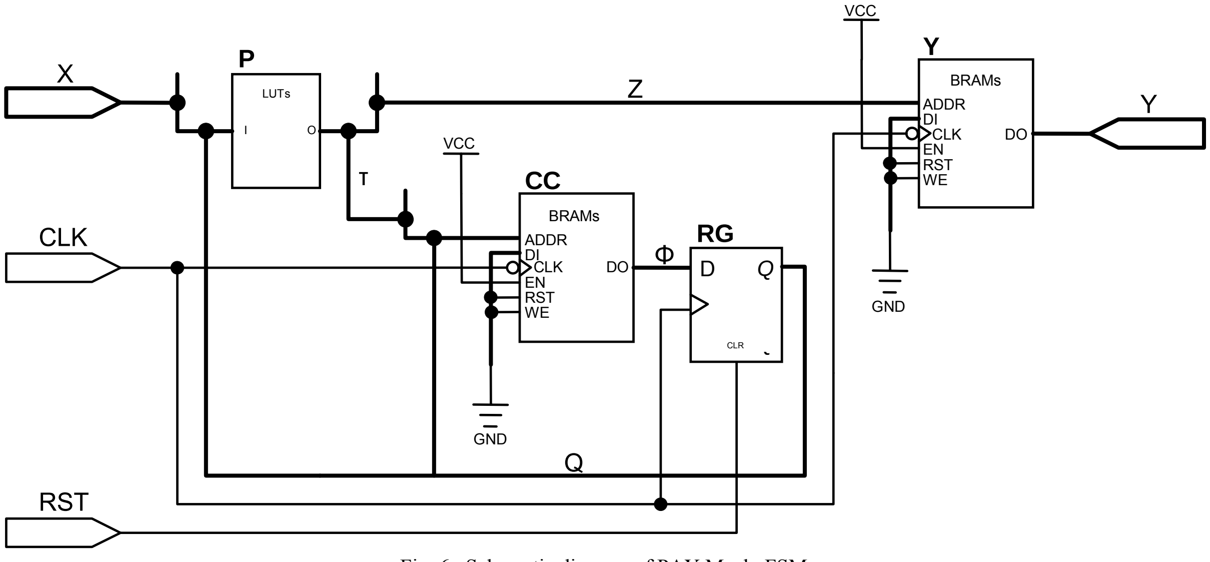

Fig. 6. Schematic diagram of PAY Mealy FSM

where an address is formed as concatenation of $K_t(a_s)$ and $K(Y_t)$ (or $K_m(a_s)$ and $K(a_m)$) and the value of a word for this address is $K(a_s)$. The contents of this memory is described by the internal state code converter table.

Schematic diagrams of the PYY Mealy FSM logic circuit (Fig. 5) and the PAY Mealy FSM logic circuit (Fig. 6) are based on the architecture of Xilinx Virtex FPGAs [9] but they can be easy adopted to FPGAs of other vendors because all logic elements, especially LUTs and memory blocks, and their connections are very similar.

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 until data are stable. It is especially important when an internal state is encoded. It also means that memory blocks also works as an output register in case when microoperations are encoded.

VI. Automata Synthesis System

There was created the Automata Synthesis (A♠S) System [12] to perform the logic synthesis of FSMs with use of designed structures. The software was created in Borland C++ Builder and it works in batch mode under Windows XP operating system.

The input for the A♠S System is an FSM described in a KISS2 format [8]. As output there is generated the set of files. These files represent the structural description of a selected type of an FSM in Verilog HDL [7]. The combinational circuit is described by the set of logic equations using the `assign` statement. The content of memories is described using the `case` statement. Because it should by synthesized as synchronous ROM memory this statement is placed in the `always` block with the falling edge of the CLK signal on the sensitivity list. The address is placed as a selector of the case statement and the content of the memory is described by choices of the case statement. To ensure that such described module will be synthesized as a memory block there is required to set a value of special synthesis attribute `bram_map` to "`YES`" [10]. This is synthesis attribute of Xilinx devices and it is ignored in case of synthesis into FPGA devices from other vendors, But each vendor supplies similar attributes or directives, e.g. the `romstyle` synthesis attribute for Altera FPGAs [11]. Then these files can be the entry point for further synthesis and implementation into selected FPGA device (Fig. 7).

The system also generate a report file where the number of logic functions and the size of memory is calculated. The macro to run third party synthesis (at this version of A♠S only Xilinix XST is supported) can be also generated.

The implemented methods of synthesis were tested using benchmarks from the *LGSynth91* library [8]. The results of synthesis for selected FSMs are presented in Table 1. It can be saw that one of proposed methods of synthesis (PAY or PYY)

TABLE I
Synthesis Results

Benchmark	Type of resources	Structure			
		P	PY	PAY	PYY
ex4	Slices	20	16	**9**	10
	LUTs	36	29	**16**	18
	FFs	4	4	**4**	4
	BRAMs	0	1	**2**	2
ex6	Slices	29	19	24	**15**
	LUTs	52	34	43	**27**
	FFs	3	3	3	**3**
	BRAMs	0	1	2	**2**
keyb	Slices	51	56	**37**	49
	LUTs	90	99	**65**	86
	FFs	5	5	**5**	5
	BRAMs	0	1	**2**	2
opus	Slices	22	16	14	**9**
	LUTs	39	29	24	**16**
	FFs	4	4	4	**4**
	BRAMs	0	1	2	**2**
planet	Slices	141	90	**64**	75
	LUTs	248	155	**113**	131
	FFs	6	6	**6**	6
	BRAMs	0	2	**3**	5
s298	Slices	529	628	**238**	398
	LUTs	951	1143	**433**	719
	FFs	19	26	**13**	19
	BRAMs	0	1	**5**	3
sand	Slices	113	115	89	**84**
	LUTs	199	205	156	**148**
	FFs	5	5	5	**5**
	BRAMs	0	1	2	**2**
styr	Slices	112	109	**87**	90
	LUTs	199	192	**155**	160
	FFs	5	5	**5**	5
	BRAMs	0	1	**2**	2
tma	Slices	55	46	**26**	**26**
	LUTs	97	80	**45**	**45**
	FFs	5	5	**5**	**5**
	BRAMs	0	1	**2**	**2**

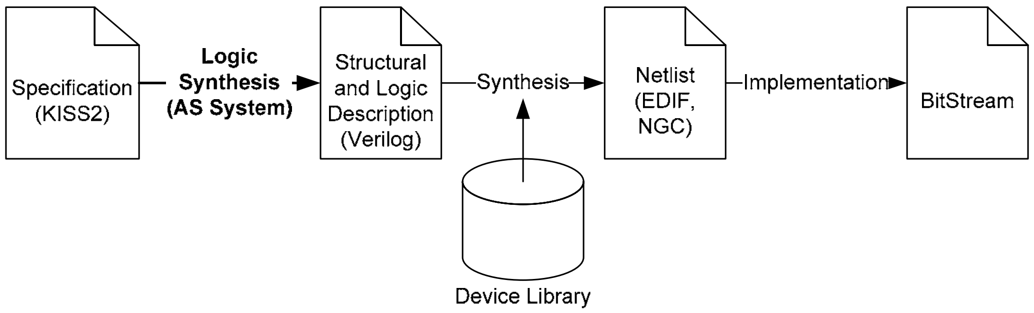

Fig. 7. Design flow for FPGAs with A♠S

always gives better results (in bold) that standard ones (P and PY). The analysis of all benchmarks shown that the application of the structure PAY reduce the number of required LUTs by 33% in comparison with the structure P and by 30% in comparison with the structure PY in average. The gain for the structure PYY is 19 and 16% respectively. Obtained results show that PAY Mealy FSM gives rather better results than PYY Mealy FSM but in some cases implementation of PYY structure gives more benefits. Which method is better depends only on characteristic of currently implemented control algorithm.

Proposed methods required more memory bits than PY Mealy FSM. But new FPGA devices has embedded memory blocks with huge capacity and these blocks can be used for implementation of Y and CC circuits.

VII. Conclusion

The proposed in this article methods of synthesis and implementation of Mealy FSMs with the multiple encoding of internal states based on a current state or a currently executed microinstruction permit to decrease the number of logic elements required for implementation of the combinational circuit of an FSM. The realization of decoders with use of memory blocks allows to utilize different kind of resources that are available in new FPGA devices. It leads to balanced utilization of device resources in a synthesis process of a control units.

There was created the Automata Synthesis System for verification of proposed methods of synthesis. The obtained results shows that designed methods reduce the number of LUTs required for implementation of the combinational circuit of an FSM. Analyzed benchmarks shown that these methods are better than the standard ones. The gain very strong depends on parameters of considered control algorithm and the selection of a structure should be made individually for each control algorithm.

References

[1] S. Baranov, *Logic Synthesis for Control Automat*. Boston: Kluwer Academic Publisher, 1994.

[2] A. Barkalov, and M. Węgrzyn, *Design of Control Units with Programmable Logic*. Zielona Góra: University of Zielona Góra Press, 2006.

[3] A. Bukowiec, A. Barkalov, and L. Titarenko, "FSMs implementation into FPGAs with multiple encoding of states" in *Proc. IEEE East-West Design & Test Symposium*, Lviv, Ukraine, 2008, pp. 72-75.

[4] J. Jenkins, *Designing with FPGAs and CPLDs*. New Jersy: Prentice Hall, 1994.

[5] T. Łuba, M. Rawski, and Z. Jachna "Functional decomposition as a universal method of logic synthesis for digital circuits" in *Proc. 9th Int. Conf. Mixed Design of Integrated Circuits and Systems*,Wrocław, Poland, 2002, pp. 285-290.

[6] Z. Salcic, *VHDL and FPLDs in Digital Systems Design, Prototyping and Customization*. Boston: Kluwer Academic Publishers, 1998.

[7] D. Thomas, and P. Moorby, *The Verilog Hardware Description Language*. Norwell: Kluwer Academic Publishers, 2002.

[8] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide. version 3.0," Microelectronics Center of North Carolina, Tech. Rep. 1991-IWLS-UG-Saeyang, Jan. 1991.

[9] *Virtex 2.5V Filed Programmable Gate Arrays Data Sheet*, Xilinx, San Jose, 2001.

[10] *XST User Guide (8.1i)*, Xilinx, San Jose, 2005.

[11] *Design and Synthesis* vol. 1 of *Quartus II Development Software Handbook (v8.0)*, Altera, San Jose, 2008.

[12] A. Bukowiec. (2008, May). Automata Synthesis System [Online]. Available: http://willow.iie.uz.zgora.pl/~abukowie/AS/as.htm