

# Synthesis of Reconfigurable Control Unit for Combinatorial Processor\*

I. Skliarova  
Department of Electronics  
and Telecommunications  
Aveiro University  
3810-193 Aveiro, Portugal  
iouliia@ua.pt

A. B. Ferrari  
Department of Electronics  
and Telecommunications  
Aveiro University  
3810-193 Aveiro, Portugal  
ferrari@ieeta.pt

**Abstract** *This paper describes the synthesis and implementation of Reconfigurable Control Units (RCU) on the base of FPGAs. The mathematical model of RCU is a traditional Finite State Machine (FSM). In order to be able to modify the behavior of RCU after it has been designed and implemented in hardware we propose to construct RCUs on the base of RAM cores. Synthesis of RCUs is divided in some steps, such as its decomposition in reprogrammable blocks, state encoding, etc. The paper presents algorithms and software tools for performing all these steps, which finally allow to synthesize automatically the circuit of an RCU from a given behavioral specification and to implement it in FPGA on the basis of a pre-designed structure with some predefined constraints. RCUs of such type have been proposed and designed for the use in a reconfigurable combinatorial processor.*

## 1. INTRODUCTION

There are many practical applications, which require the solution of some combinatorial problems. It is known that the majority of these problems are NP-hard and as a result they are time and resource consuming. Because of that it is worthwhile to consider and to implement the respective accelerators such as coprocessors for general-purpose computers. It should be noted that the heterogeneous nature of combinatorial tasks makes it difficult to construct an effective specialized device, i.e. an ASIC. On the other hand, we can analyze different combinatorial tasks and select subsets of unique and common operations. As a result we can design a device including a fixed part (for common operations) and a reconfigurable part (for unique operations). Such device might be constructed on the basis of reconfigurable hardware, such as commercially available FPGAs.

The papers [1,2] present a technique that might be used for such purposes. One part of the proposed device has been fixed and the other has been built on the base of reprogrammable components. This device is intended to be used as a reprogrammable combinatorial processor (RCP). The RCP was designed in order to solve different problems formulated over logic matrices. The reprogrammable part is used to implement such subset of operations that are unique for a given task. Since this subset has to be changed from one application to another we can realize this by modifying the functionality with the aid of a reprogramming technique. In fact there are two components, which have to be reprogrammed. The first component modifies the operations themselves. The second component changes the control algorithm that forces the required sequence of operations to be executed and it can be seen as a RCU. The paper considers a design technique for RCU, which is going to be utilized in RCP, realizing different combinatorial algorithms on the base of the same hardware.

The paper is organized in 6 sections. The first section is this introduction. Section 2 presents the structure of RCU and shows its implementation in FPGA. Section 3 describes the process of RCU synthesis. Section 4 discusses the software tools designed for the synthesis. Section 5 demonstrates the practical use of RCU for solving combinatorial problems. The conclusion is in section 6.

## 2. THE STRUCTURE OF THE RCU AND ITS IMPLEMENTATION IN FPGA

We consider an implementation of the RCU on the base of FPGAs of XC4000 family from Xilinx [3]. In fact this approach is not technology-oriented and can also be used for other families of FPGAs. The model of RCU is a Finite State Machine (FSM) with modifiable behavior. The latter might be presented at structural level as a composition of a combinatorial circuit, which calculates the next states and outputs, and a register that keeps the current state. We are considering so called RAM-based FSMs, i.e. such FSMs for which the combinatorial circuit is constructed from RAM-based cells (see fig. 1). The FSM register stores the current state. Any state code from the register is combined with input variables from the set  $X$  and is considered to be an address of the FSM RAM. Each address activates the word of the FSM RAM that specifies the code of the respective next state. An additional RAM (Y RAM) is used in order to produce outputs on the base of state codes (we are considering the Moore FSM model). It is very easy to reprogram this device. For such purposes it is sufficient to reload the contents of two RAMs.

Let  $K$  be the minimum number of bits needed for state codes. Thus the size of Y RAM is  $2^K \times N$  and the size of FSM RAM is  $2^{(K+L)} \times K$ , where  $L$  is the number of inputs from the set  $X$ , and  $N$  is the number of outputs from the set  $Y$ . However if we increase  $L$  such implementation would be very resource consuming (i.e. the size of the FSM RAM becomes quite large). If we consider a particular technology it is relatively easy to calculate the number of configurable logic blocks (CLBs) of FPGA that have to be used. For example, the CLBs of the XC4000 FPGAs can be configured to

---

\* This work was sponsored by the grant FCT-PRAXIS XXI/BD/21353/99

implement a pair of 16×1 RAMs, a 32×1 RAM, or a 16×1 dual port RAM [3]. Using this information, the amount of CLBs needed in order to implement RAM-based FSMs can be estimated.

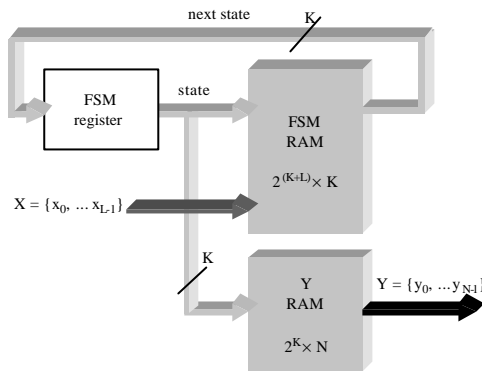


Fig. 1. A trivial structure of RAM-based FSM

The structure in fig 1 can be improved in such a way that the size of RAM-blocks will be essentially decreased (see fig. 2). The new block P is used in order to mix input signals with state codes. In this case the size of FSM codes is fuzzy [4] in a sense that different states have codes with different number of bits. This technique allows to reduce significantly the size of address bus for the FSM RAM. Let  $K'$  be the size of the FSM RAM address bus. In this case the following expression is valid:  $K \leq K' \ll (K+L)$  and for many practical applications  $K' \rightarrow K$ . As a result the size of the FSM RAM becomes  $2^{K'} \times K'$ .

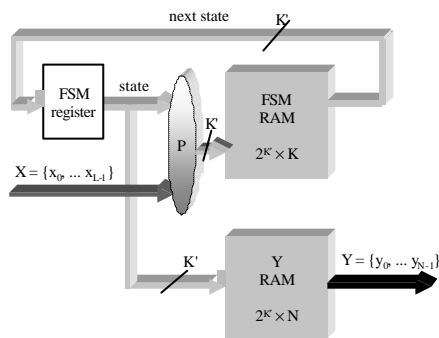


Fig. 2. FSM based on fuzzy state encoding technique

In order to provide the circuit shown in fig. 2 with the property of reprogrammability let us decompose the block P into new components as it is depicted in fig. 3. The block A RAM provides conditional state transitions, i.e. such transitions that are caused by some input variables. For all unconditional transitions the output vector  $\mathbf{a}$  is identically equal to 0. The block P calculates output vector  $\mathbf{p}$  that depends on inputs from the set X and on the vector  $\mathbf{a}$ . Thus by reloading all the RAMs, such as A RAM, Y RAM, P RAM and FSM RAM, we are able to implement different algorithms of logic control on the basis of the same hardware. Let us calculate the required size of memory. Let  $S = \{s_0, \dots, s_{M-1}\}$  is a set of FSM states, where M is the number of states,  $S^x \subseteq S$  the set of states from which there exist conditional transitions, and let  $G = \lceil \log_2(|S^x| + 1) \rceil$ . Thus the size of the Y RAM is  $2^{K'} \times N$ , the size of the FSM RAM is  $2^{K'} \times K'$ , the size of the A RAM is  $2^{K'} \times G$  and the size of the P RAM is  $2^{G+L} \times K'$ .

For implementation of RCU we have used the cheap and simple FPGA XC4010XL of Xilinx, which has very limited number of CLBs (400 CLBs). The following constraints for RCU have been chosen:  $K'=7$ ,  $N=32$ ,  $L=5$ ,  $|S^x|=7$ . Such an implementation has required 215 CLBs, i.e. approximately 55% of the FPGA resources. With the continuous increase in gate count and with the advances in the fabrication process, we can expect that area limitation will not be a problem for such kind of devices in future.

The effectiveness of the proposed technique was estimated with the aid of different experiments that have been performed. They show that for the majority of practical applications  $K \leq K' \leq K+1$ , where K is a minimum number of bits for encoding of the states and  $K'$  is an actual size that was obtained after applying the proposed state encoding method. Thus we can estimate the area reduction that can be achieved if the proposed structure (see fig. 3) is used. This area was compared with the control unit depicted in fig. 1. For the considered above constraints the size of the required memory in fig. 3 is equal to  $2^7 \times 32 + 2^7 \times 3 + 2^{3+5} \times 7 + 2^7 \times 7 = 7.168$  bits and for the structure shown in fig. 1 the size of memory is  $2^{6+5} \times 6 + 2^6 \times 32 = 14.336$  for the best case (i.e. when  $K'=K+1$ ) and  $2^{7+5} \times 7 + 2^7 \times 32 = 32.768$  bits for the worst case (i.e. when  $K'=K$ ). As a result the proposed solution allows to reduce the size of memory by the factor of F, where  $2 \leq F \leq 4.6$ . In fact the factor F will be a little bit less, because the proposed structure requires additional circuits that provide support for reconfiguration.

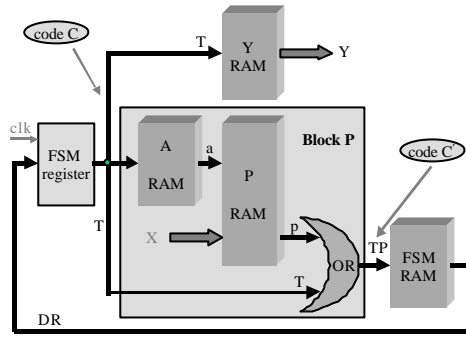


Fig. 3. The structure of RCU

For all the experiments we have used PC board XS40 from XESS containing one FPGA XC4010XL, communicating with PC through the standard parallel interface. The programming of the RCU can be carried out through the parallel port and the command instruction has the following format.

7	6	5	4	3	2	1	0
addr_data(3 : 0)			ram(1 : 0)		prog	clk	

Where

- *clk* – is the signal of synchronization;
- *prog* – distinguishes working and programming modes;
- *ram* – selects the appropriate RAM for the programming;
- *addr\_data* – specifies an address/data for the use in the respective RAM.

Note that programming through the parallel port does not allow to utilize the proposed technique in its full power. However we have used such approach just for the purpose of experimental test. The final RCU will be implemented together with the complete RCP on the base of ALPHA ADM-XRC board [5] linked to PCI bus, which is based on the Virtex FPGA XCV812E [3].

### 3. LOGIC SYNTHESIS OF RCU

In our particular case the synthesis is considered to be a sequence of steps, which allow to translate a given behavioral specification of the RCU into RAM contents assuming that the RCU is based on the predefined structure (on the predefined template) depicted in fig. 3. It is important that the results of synthesis are presented in a set of bitstreams that have to be loaded to FPGA RAM. It means that it is not allowed to change the structure of the circuit shown in fig. 3 and all the connections between its components. Thus we do not need to repeat the process of the design of FPGA-based circuit, which might lead to some unpredictable situations during the stages of mapping, placement and routing. Besides, since the circuit itself has been preliminary tested we can guarantee that it works correctly and it either does not require at all or requires only very simplified procedures for the verification.

We assume that the behavior of RCU has been specified in the form of graph-schemes (GSs) [6]. The first step of synthesis translates a given GS into a structural table, such as that was considered in [4]. After that the following steps have to be performed:

1. Applying the method for fuzzy state encoding technique [4].
2. Replacement of input variables [6] and mixing input variables and state codes [4].
3. Synthesis of bitstreams for each RAM shown in fig. 3.

As it was shown before, the size of the circuit in an FPGA strongly depends on the width of the address line for the FSM RAM. This width might be essentially decreased [4] by the use of a special state encoding. In the proposed algorithm we are trying to minimize the number of address bits in such a way that  $K' \rightarrow \lceil \log_2 M \rceil$ , where  $M$  is the number of states in the FSM.

Let  $S = \{s_0, \dots, s_{M-1}\}$  is a set of FSM states,  $M$  is the number of FSM states,  $X = \{x_0, \dots, x_{L-1}\}$  is the set of FSM input signals,  $Y = \{y_0, \dots, y_{N-1}\}$  is the set of FSM output signals,  $T = \{t_0, \dots, t_{R-1}\}$  is the set of all state transitions. Each state transition  $t_i$  can be presented in the following form:  $t_i = \{s_{from}, s_{to}, X(s_{from}, s_{to})\}$ ,  $i=0, \dots, R-1$ , where  $s_{from}$  is an initial state in the state transition and  $s_{to}$  is the next state,  $X(s_{from}, s_{to})$  is the function of input variables that causes transition from  $s_{from}$  to  $s_{to}$ . When there are two state transitions  $t_i = \{s_{from_i}, s_{to_i}, X(s_{from_i}, s_{to_i})\}$  and  $t_j = \{s_{from_j}, s_{to_j}, X(s_{from_j}, s_{to_j})\}$ , and if  $s_{from_i} = s_{from_j}$  then  $t_i$  and  $t_j$  have a common fanin state and if  $s_{to_i} = s_{to_j}$  then  $t_i$  and  $t_j$  have a common fanout state [7]. Let us divide the set  $T$  into  $M$  subsets:  $T = \{TFI_0, \dots, TFI_{M-1}\}$ ,  $TFI_0 \cap \dots \cap TFI_{M-1} = \emptyset$  and  $TFI_0 \cup \dots \cup TFI_{M-1} = T$ , in such a way that each subset  $TFI_i$ ,  $i=1, \dots, M-1$ , is composed of state transitions with common fanin state. Thus all the transition from the state  $s_0$  form the set  $TFI_0$ , all the transitions from the state  $s_1$  form the set  $TFI_1$ , and so on. Now let us consider the set  $T$  as a union of the

following subsets:  $T=\{TFIO_0, \dots, TFIO_{Q-1}\}$ ,  $0 \leq Q \leq M$ ,  $TFIO_0 \cap \dots \cap TFIO_{Q-1} = \emptyset$  and  $TFIO_0 \cup \dots \cup TFIO_{Q-1} = T$ , and each element  $TFIO_i$ ,  $i=0, \dots, Q-1$ , is composed of such subsets TFI that include state transitions with a common fanout state. Obviously, if the control algorithm does not have conditional transitions then  $TFIO_i = t_i$ ,  $i=0, \dots, R-1$ .

Let us consider an example. Suppose that a control algorithm has been described by a GS depicted in fig. 4a). It can be formally converted to a structural table of the respective FSM [6]. For our example we have  $M=9$ ,  $L=2$ ,  $N=8$ ,  $R=14$ ,  $Q=7$  and  $T=\{t_0, \dots, t_{13}\}=\{TFI_0, \dots, TFI_8\}$ ,  $TFI_0=\{s_0, s_1, 1\}$ ,  $TFI_1=\{s_1, s_2, x_0\}, \{s_1, s_7, x_0, x_1\}, \{s_1, s_8, x_0, x_1\}\}$ , and so on. All the sets  $TFI_i$ ,  $i=0, \dots, M-1$ , are shown in fig. 4b). Fig. 4c) depicts the sets  $T= \{TFIO_0, \dots, TFIO_6\}$ .

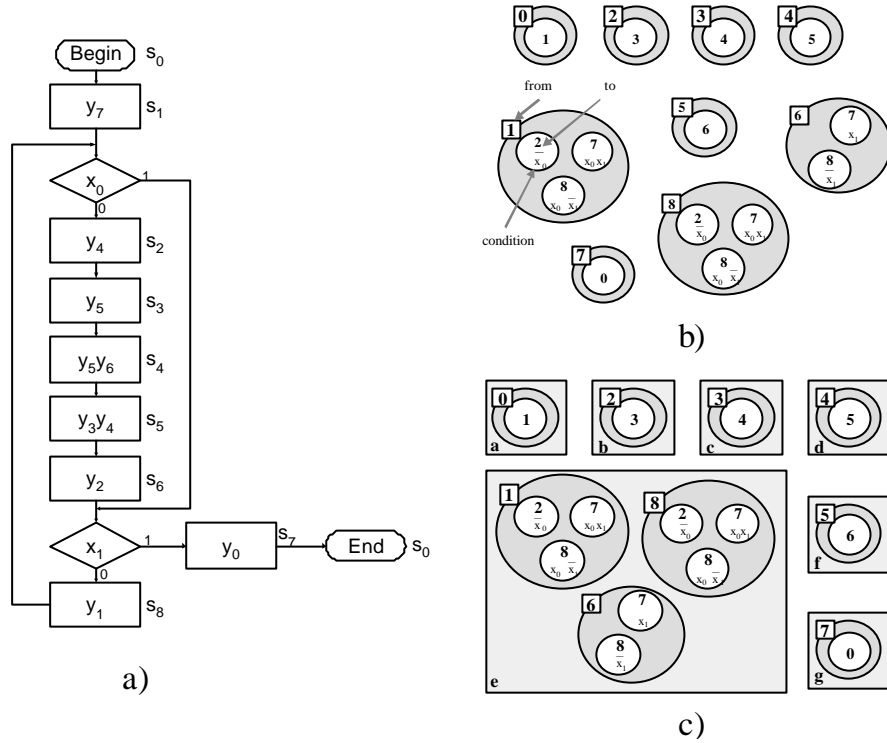


Fig. 4. a) A GS that describes a given control algorithm; b) The sets  $TFI_i$ ,  $i=0, \dots, 8$ ; c) The sets  $TFIO_i$ ,  $i=0, \dots, 6$

According to the method [4] each state transition  $t_i$ ,  $i=0, \dots, R-1$ , has to be assigned a code  $\dot{C}(t_i)$ , and each set  $TFI_i$ ,  $i=0, \dots, M-1$ , has to be assigned a code  $C(TFI_i)$  (note that  $C(TFI_i) = C(s_i)$ ), with the following requirements:

- $C(TFI_0)$  ort ... ort  $C(TFI_{M-1})$ , i.e. the codes of all FSM states must be mutually orthogonal;
- $\dot{C}(t_0)$  ort ... ort  $\dot{C}(t_{R-1})$ , i.e. the codes of all state transitions must be mutually orthogonal except if they have a common fanout state;
- $C(TFI_i) = \dot{C}(t_0) \wedge \dots \wedge \dot{C}(t_{|TFI_i|-1})$ ,  $t_0, \dots, t_{|TFI_i|-1} \in TFI_i$ .

In order to provide this we select such a set  $TFIO_{max}$  that has the maximum number of elements (in our example we have to choose the set  $e$  shown in fig. 4c). Then in this set we have to find a component  $TFI_{max}$  with the maximum number of elements. For our example in fig. 4c) we select the set 1 (since the sets 1 and 8 have the same number of state transitions we consider the first of them). Then we have to choose in the set  $TFI_{max}$  such a state transition  $t_{max}$  that has a common fanout state with the maximum number of elements of  $TFIO_{max}$ . In our example this is a transition to the state  $s_7$ . Note that the transitions to  $s_7$  and to  $s_8$  have a common fanout state with 2 elements (so we take the first of them), while the transition to the state  $s_2$  has a common fanout state with one element. The selected transition is assigned the appropriate code (as it will be demonstrated below) and we choose the next state transition following the same rules. When we assign the codes we have to satisfy the following requirements:

- All the codes  $C(TFI_i)$ ,  $i=0, \dots, M-1$ , must remain mutually orthogonal;
- The codes of state transitions belonging to the same set  $TFI$  must differ by a minimum possible number of bits  $B$ .

The less the value  $B$  the greater number of state transitions can be coded by a given number of bits. The final algorithm of state encoding can be presented in the following sequence of steps:

1. Calculate the value  $K' = \lceil \log_2 |M| \rceil$ .
2. From the sets  $TFIO_i$ ,  $i=0, \dots, Q-1$ , that have not been yet considered select the set  $TFIO_{max}$  with maximum number of elements:  $|TFIO_{max}| \geq |TFIO_i|$ ,  $i=0, \dots, Q-1$ .

3. From the sets  $TFI_i$ ,  $i=0, \dots, M-1$ , that have not been yet considered select the set  $TFI_{max}$  with maximum number of elements:  $|TFI_{max}| \geq |TFI_i|$ ,  $TFI_{max} \subseteq TFIO_{max}$ ,  $TFI_i \subseteq TFIO_{max}$ .
4. Calculate minimum number of bits  $B$ , by which the codes of state transitions that belong to  $TFI_{max}$  must differ:  $B = \lceil \log_2 |TFI_{max}| \rceil$ .
5. Select in the set  $TFI_{max}$  a transition  $t_{max}$ , which has a common fanout state with a maximum number of elements of  $TFIO_{max}$ .
6. Verify if we can assign to the selected transition  $t_{max}$  the code of any state transition  $t_{exist}$  that has been already considered and treated. It is possible if and only if the following two conditions are satisfied:
  - The state transitions  $t_{max}$  and  $t_{exist}$  have a common fanout state;
  - $C(TFI_{max})$  ort  $C(TFI_i)$ ,  $i=0, \dots, M-1$ ,  $i \neq max$ , i.e. the codes of all sets  $TFI_i$ ,  $i=0, \dots, M-1$ ,  $i \neq max$ , are still mutually orthogonal.
7. If we are not able to use any code, which has been already assigned, then it is necessary to take the first assigned code in the set  $TFI_{max}$  (for the first considered transition always use the code "1..1", where the notation "1..1" specifies the code with all ones). Store this code somewhere. Then decrement it and verify if it can be used, i.e. the number of bits that distinguish different codes from the set  $TFI_{max}$  is less than or equal to  $B$ . If this condition is satisfied, verify if the codes in all the sets  $TFI_i$ ,  $i=0, \dots, M-1$ , are still mutually orthogonal, i.e.  $C(TFI_{max})$  ort  $C(TFI_i)$ ,  $i=0, \dots, M-1$ ,  $i \neq max$ . If this is true then use this code. In opposite case decrement the code again and continue this process until the code will not be equal to "0..0". If we are not capable to find a solution then increment the initial code and test it until it will not be equal to "1..1". Note that the number of tests for orthogonality in the worst case will not exceed the value  $\frac{K!}{B!(K-B)!}$ . If again it is impossible to find out the solution increment  $B$  and repeat point 7.
8. If we are not able to code all the states by  $K'$  bits, increment  $K'$  and repeat all the steps starting from point 2.

Applying the proposed algorithm to the GS in fig. 4a) enables us to find out the solution presented in table 1. The column  $s_{from}$  contains all the states, the columns  $C(s_{from})$  and  $C(s_{to})$  keep the codes of the respective states, the column  $X(s_{from}, s_{to})$  presents the function of input variables that cause transitions from  $s_{from}$  to  $s_{to}$ , the column  $Y$  indicates the respective microinstruction that have to be generated. The column  $FSM RAM$  shows the addresses of the FSM RAM where the respective codes of the next states  $C(s_{to})$  will be recorded. In each code  $C(s_{from})$  the characters  $i$  point to those bits, which distinguish addresses of the states with a common fanin state. Let us enumerate the positions of  $i$  in each code from right to left, starting from zero and associate with them a vector  $\mathbf{p}$ . The column  $\mathbf{x} \rightarrow \mathbf{p}$  shows the values of  $\mathbf{p}$  for each transition. For example, in the transition  $t$  from  $s_1$  to  $s_8$ , we have  $C(s_{from})=C(s_1)=1i1i$  and  $C(t)=1110$ , thus  $\mathbf{p}=0100$ .

TABLE 1.  
STRUCTURAL TABLE THAT KEEPS INITIAL AND FINAL DATA FOR SYNTHESIS

$s_{from}$	$C(s_{from}) = C(TFI_{from})$	$X(s_{from}, s_{to})$	$Y$	$\mathbf{x} \rightarrow \mathbf{p}$	$s_{to}$	$C(s_{to})$	$FSM RAM = C^c(t)$
$s_0$	0000	1	-	0000	$s_1$	1010	0000
$s_1$	1i1i = 1010	$\bar{x}_0$ $x_0x_1$ $x_0\bar{x}_1$	$y_7$	0000 0101 0100	$s_2$ $s_7$ $s_8$	0111 1101 1100	1010 1111 1110
$s_2$	0111	1	$y_4$	0000	$s_3$	1000	0111
$s_3$	1000	1	$y_5$	0000	$s_4$	1001	1000
$s_4$	1001	1	$y_5y_6$	0000	$s_5$	1011	1001
$s_5$	1011	1	$y_3y_4$	0000	$s_6$	1110	1011
$s_6$	111i = 1110	$x_1$ $\bar{x}_1$	$y_2$	0001 0000	$s_7$ $s_8$	1101 1100	1111 1110
$s_7$	1101	1	$y_0$	0000	$s_0$	0000	1101
$s_8$	11ii = 1100	$\bar{x}_0$ $x_0x_1$ $x_0\bar{x}_1$	$y_1$	0000 0011 0010	$s_2$ $s_7$ $s_8$	0111 1101 1100	1100 1111 1110

#### 4. SOFTWARE TOOLS

The described algorithm has been implemented in a program written in C++. The input to the program is the name of a text file specifying a given control algorithm in the following form:

*The number of the initial state <condition for the respective transition> The number of the next state <the output values>*,

where angle braces keep optional parameters. For example, for the control algorithm in fig. 4a) we have:

```
0 1
1 nx0 2 y7
1 x0x1 7 y7
1 x0nx1 8 y7
...
```

The program performs all the required calculations and generates the file to be used for programming the RCU, which will realize the given algorithm. In order to program the FSM RAM and the Y RAM (see fig. 3) the respective codes from the columns *FSM RAM* and *C(s<sub>from</sub>)* of table 1 are used. In order to program the A RAM all the states with branches (i.e. the states *s*<sub>1</sub>, *s*<sub>6</sub> and *s*<sub>8</sub> shown in fig. 4a) are sequentially coded by binary codes with minimum size. All the states with unconditional transitions get the code "0...0". In order to program the P RAM we use the contents of the column *x* → *p*, i.e. in the address that combines *x* and *a*, we have to write the code from the respective row of the column *x* → *p*.

Fig. 5 demonstrates how to program all RAMs for the considered example. Dashed lines mean that the respective memory positions can be of any value. In fig. 5 the current state is *s*<sub>1</sub> with the code 1010. In the A RAM the code *a*="01" corresponds to this state (*s*<sub>1</sub>), which in combination with logic condition *x* = "01" activates in P RAM vector *p*="0100". Then the operation (*p* ∨ *state*) is being performed and we obtain the code of the next state "1100". In table 1 this code corresponds to the state *s*<sub>8</sub> so we can see that the constructed circuit functions correctly (see fig. 4a).

If it is necessary to realize any other control algorithm we have to describe it in the considered above format. The developed software tools will construct the respective structural table and all relevant positions of the RAMs will be automatically programmed. In future we will create a library of basic functions for combinatorial algorithms. When required the assisting software tools will extract the corresponding configuration from the library and download it to the RCU. Note that basic structure of RCU is loaded from the beginning and then it is necessary to reprogram only some RAM's positions. This reduces essentially the configuration overhead.

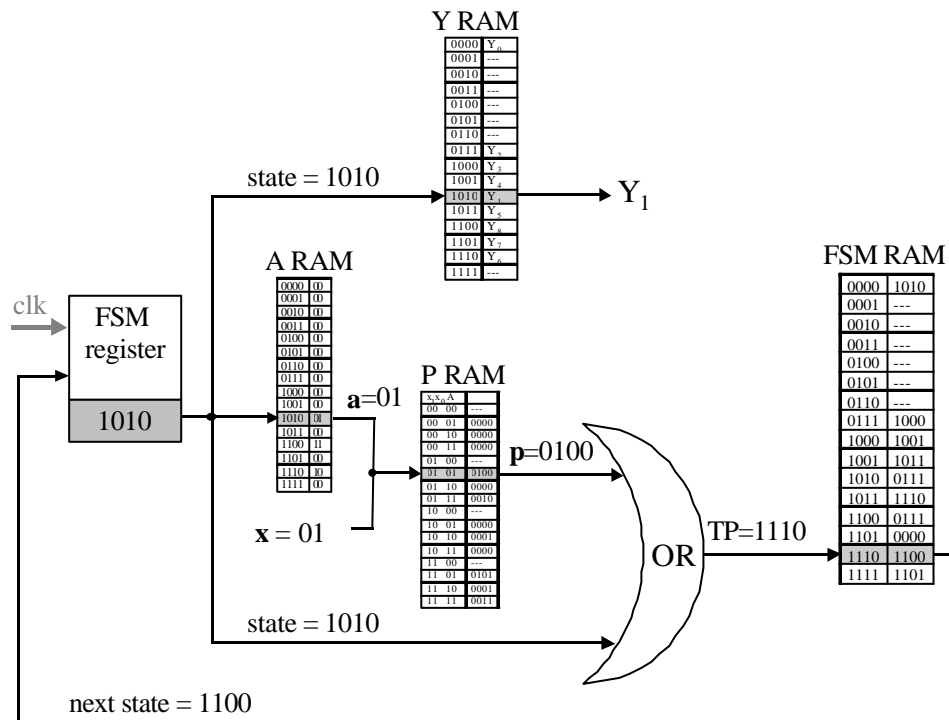


Fig. 5. Programming of RCU RAMs for the control algorithm shown in fig. 4a)

The timing characteristics of the proposed algorithm have been estimated in a number of experiments that have been performed on PC-III, 800 MHz, 256 MB RAM running under Windows 2000. The results are presented in table 2 and they show that the considered technique can be used for practical applications.

TABLE 2.  
THE RESULTS OF EXPERIMENTS

Example	Number M of FSM states	Number L of logic conditions	Number N of control signals	Minimum number of bits K required for state codes	Actually used number of bits $K^c$ for state codes	Synthesis time (in sec.)
min_row	9	2	5	4	4	0.058
ex_t_m2	13	10	32	4	5	0.24
min_cover	29	5	24	5	5	0.15
gr_al	123	20	67	7	8	10.16

## 5. PRACTICAL APPLICATION OF THE PROPOSED TECHNIQUE FOR COMBINATORIAL PROBLEMS

The combinatorial processor proposed and considered in [1,2], enables us to realize a set of operations over logic matrices. With the aid of this processor we can solve a number of combinatorial problems. Logic matrices are also very well suited for keeping them in FPGA RAM-blocks. The proposed technique of synthesis and implementation of RCUs has been tested for solving a number of combinatorial tasks, such as matrix covering, verifying if a given Boolean function  $f(\mathbf{x})$  is a tautology (i.e. it is identically equal to 1 independently on values of vector  $\mathbf{x}$ ), etc. First each problem has to be formulated on logic matrices and then it can be solved with the aid of the designed processor. For instance, for the last problem we have used the following technique. It is known that any Boolean function might be represented in DNF (Disjunctive Normal Form). Then the function can be described by a Boolean or ternary matrix in such a way that columns of the matrix correspond to function arguments and rows of the matrix represent elementary conjunctions [8]. For example, the function

$$f(\mathbf{x}) = x_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 = x_1x_2 \vee \bar{x}_2x_3$$

can be easily converted to the following matrices:

$$\mathbf{B} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & - \\ - & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Thus this problem is equivalent to finding out a ternary vector that is orthogonal to each row of the matrix  $\mathbf{T}$ . If it cannot be found we conclude that the respective function is a tautology [9].

The proposed approach was also compared with the techniques used in Xilinx Foundation Software (Series 3.1i). For all the examples considered the method allows to reduce the number of CLBs needed for control unit [4]. However, if the design is based on the developed hardware template, it assumes a feasibility for implementing any control algorithms that matches the predefined constraints. Since the majority of the considered algorithms do not use all the available resources of the template our method does not allow to reduce the number of CLBs. However, it is very important that the suggested structure provides logic support for reconfiguration while FSMs, synthesized by any other commercially available tools, are unique and can be used just for the implementation of a particular algorithm. If we need to realize a new algorithm it is necessary to repeat all the steps including verification of the circuit and its debugging. Since the execution unit of our processor is also based on a predefined hardware template [1,2] it becomes possible to construct a reprogrammable device for solving a number of combinatorial problems utilizing the same hardware.

Nevertheless in assumption that the constraints of given template match parameters of control algorithm it is feasible to examine an area reduction of the suggested method comparing with Xilinx Foundation Software (XFS). Let us consider the example "min\_cover" from table 2. The results that were obtained for different synthesis and implementation techniques are presented in table 3. In the first case the behavior of the control device was described in VHDL and XFS and FPGA Express software were used for the synthesis. If binary encoding technique was selected the respective circuit required 47 CLBs and if one-hot encoding was selected it required 57 CLBs. In the second case the synthesis was performed with the aid of the developed software tools and the implementation was done on the base of template shown in fig. 2. The block P was constructed from gates with the aid of information generated by the synthesis tools. An example of the constructed block P for table 1 is presented in fig. 6. The resulting circuit cannot be reconfigured and it requires less number of CLBs (37 CLBs), than the circuit built by XFS. In the third case the control device was constructed on the base of structure shown in fig. 1. The sizes of FSM RAM and Y RAM are equal to  $2^{5+5} \times 5$  and  $2^5 \times 24$  accordingly. The resulting circuit required 245 CLBs. In the last case we synthesized control device with the structure depicted in fig. 3. The sizes of FSM RAM, Y RAM, A RAM and P RAM are equal to  $2^5 \times 5$ ,  $2^5 \times 24$ ,  $2^5 \times 4$ ,  $2^{5+4} \times 5$ , respectively. The resulting circuit requires 159 CLBs. Note that the last two devices are reconfigurable and can be used for realizing the considered algorithms and also for implementing any other algorithm whose parameters do not exceed the pre-defined constraints. It can be provided by reloading the respective RAM blocks. The additional resources, which are needed for reconfiguration, were included in the presented amount of CLBs (see table 3).

TABLE 3.  
THE EXPERIMENTS WITH "MIN\_COVER" EXAMPLE

1. Implementation in XFS (not reconfigurable)		2. Applying the proposed method (not reconfigurable)	3. Design based on the structure shown in fig. 1 (reconfigurable)	4. Design based on the structure shown in fig. 3 (reconfigurable)
binary	one-hot			
47 CLBs	57 CLBs	37 CLBs	245 CLBs	159 CLBs

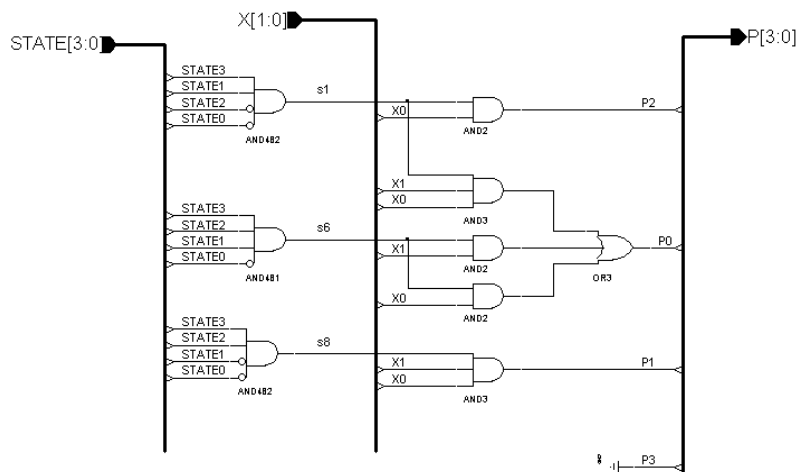


Fig.6. Logic scheme of block P for table 1

## 6. CONCLUSION

The paper presents a technique for synthesis and implementation of reprogrammable control units (RCU) on the base of FPGAs. These units have been designed for the use in a reconfigurable combinatorial processor dealing with logic matrices. Implementation of different control algorithms has been provided on the same hardware template by reprogramming the predefined RAM core. Algorithms for RCU have been specified in a text format. These algorithms can be automatically translated to bitstreams for FPGA RAM-cells with the aid of the developed software tools. Because of the limited size of the XC4010XL FPGA, which has been used for the experiments, the realized algorithms have been constrained by the numbers of inputs, outputs, conditional transitions, etc. However, the proposed hardware template can be used to implement any given algorithm under these constraints. The results of experiments have shown that the proposed realization of RCU requires less area comparing with circuits generated by Xilinx Foundation Software from the same specification. Note that many combinatorial computations over logic matrices can be executed in parallel. Since the proposed RCU does not support parallelism yet, we are going to work on this in the future.

## ACKNOWLEDGMENT

Authors are thankful to the corresponding member of Academy of Science of Belarus, Prof. A.D.Zakrevski for fruitful discussions concerning this work.

## REFERENCES

- [1] I.Sklyarova, A.B.Ferrari, "Exploiting FPGA-based Architectures and Design Tools for Problems of Reconfigurable Computations", Proceedings of the SBCCI 2000 XIII Symposium on Integrated Circuits and System Design, Manaus, Brazil, September 2000, pp. 347-352.
- [2] I.Sklyarova, A.B.Ferrari, "Development tools for problems of combinatorial optimization", Proceedings of the 4th Portuguese Conference on Automatic Control (CONTROLO'2000), Guimarães, Portugal, October 2000, pp. 552-557.
- [3] Xilinx, "The programmable Logic Data Book", Xilinx, San Jose, 2000.
- [4] V.Sklyarov, "Synthesis and Implementation of RAM-based Finite State Machines in FPGAs", Proceedings of FPL'2000, Villach, Austria, August, 2000, pp. 718-728.
- [5] <http://www.alphadata.co.uk>
- [6] S.Baranov, "Logic Synthesis for Control Automata", Kluwer Academic Publishers, 1994.
- [7] G.D.Hachtel, F.Somenzi, "Logic Synthesis and Verification Algorithms", Kluwer Academic Publishers, 1996.
- [8] I.Sklyarova, A.B.Ferrari, "Modelos matemáticos e problemas de optimização combinatoria", Electrónica e
- [9] A.D.Zakrevski, "Logical Synthesis of Cascade Networks", Moscow: Nauka, 1981.