Encoding Algorithms for Logic Synthesis

Valery Sklyarov, Iouliia Skliarova
University of Aveiro, Department of Electronics, Telecommunications and Informatics/IEETA
3810-193 Aveiro, Portugal
skl@det.ua.pt, iouliia@det.ua.pt

Abstract

This paper presents an encoding algorithm that is very efficient for many different logic synthesis problems. The algorithm is based on the use of special tables and includes two basic steps: searching for predefined graphical shapes in the tables, and swapping coded variables in the tables taking into account some constraints. The latter are specified with the aid of an auxiliary graph that reflects the overlap between coded variables in different subsets that have to be accommodated in the tables. The examples in the paper and the results of experiments have shown that the use of the proposed algorithm for state encoding allows the number of logic elements for combinational circuits of finite state machines to be decreased.

1. Introduction

Encoding algorithms (such as [1-4]) are widely used for synthesis and optimization of digital circuits. We will consider such a technique that enables us to construct systems of Boolean functions, and then to decompose these systems into sub-systems for which we are able to reduce dependency of the functions, included into each sub-system, on respective arguments [5]. Using this technique permits many logic synthesis problems to be simplified [6,7]. For example, it enables us to decompose a finite state machine (FSM) into blocks, such as that shown in fig. 1, where variables $\tau_1,...,\tau_R$ and $D_1,...,D_R$ provide interactions between the FSM memory and the combinational circuit. Let us first consider some of the advantages of this structure. The blocks $S_{ax}^1,\dots,S_{ax}^T$ can be constructed in such a way that $X_1\cap\dots\cap X_T = \varnothing$ [7] and $X = \{x_1,\dots,x_L\} = X_1\cup\dots\cup X_T$, where $x_1,\dots,x_L$ are input variables of the FSM. As a result, we can distribute input variables between different blocks. This allows many topological problems to be simplified. Besides, the structure in fig. 1 is very well suited for matrix implementation [6]. Note that the output variables from the set $Y = \{y_1,\dots,y_N\} = Y_1\cup\dots\cup Y_T$ can be distributed in the same manner [7].

The considered approach to state encoding can be applied directly to the structure in fig. 1. Indeed we can determine such sub-functions that do not depend on the input variables from the set X and implement these sub-functions in an autonomous circuit, such as S_a^{T+1}. This circuit can be constructed using two possible methods. The first method incorporates the autonomous circuit into blocks $S_{ax}^1,\dots,S_{ax}^T$ in such a way that we will be able to distribute outputs $D_1,\dots,D_R$ between different groups of blocks $S_{ax}^1,\dots,S_{ax}^T$. The main advantage of this method is that the block S_a^{T+1} becomes unnecessary. The second method assumes that all the variables $D_1,\dots,D_R$ have been distributed between two sets, D_{ax} and D_a, in such a way that $D_{ax}\cap D_a=\varnothing$, and the active values of variables from D_{ax} and D_a will be obtained as outputs from blocks $S_{ax}^1,\dots,S_{ax}^T$ and S_a^{T+1} accordingly. The main idea is to extract the essential part of the logic from $S_{ax}^1,\dots,S_{ax}^T$ and then to implement the extracted logic in S_a^{T+1} as simply as possible.

A similar approach can be followed in order to solve optimization problems in many practical applications, such as microinstruction encoding in microprogramming, matrix-based implementation of digital circuits, etc.

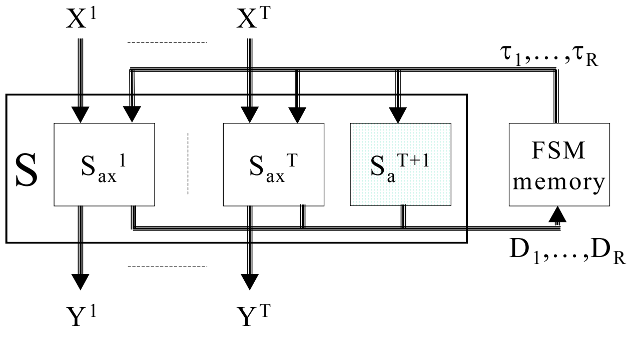


Figure 1. Block decomposition of FSM

2. Example

Let us consider an example from [8]. An FSM can be described as shown in Table 1 (at the beginning let us ignore all symbols enclosed in parentheses). Here, a_{from} - is an initial state, a_{to} - is the next state, $K(a_{from})$ and $K(a_{to})$ - are the codes of the states a_{from} and a_{to}, respectively, $X(a_{from},a_{to})$ - is a product of inputs that forces a corresponding state transition. We assume that FSM memory is built from D flip-flops.

Let us consider various transitions from the same state. We can see that for all conditional transitions (i.e. for all transitions except from the state a_6), all the sub-functions of $D_1,...,D_3$ that must be activated on transitions from a state, depend on both the state and inputs. We will say that a sub-function is active if it has to be assigned to 1. Since all the sub-functions depend on states and inputs, the relevant Boolean expressions, that are used to calculate the values $D_1,...,D_3$, contain variables from the full set $\{x_1,...,x_L,\tau_1,...,\tau_R\}$, where L - is the number of external inputs (in our example L=5) and R - is the size of the FSM memory (in our example R=3).

Consider all sub-functions of $D_1,...,D_3$ that are generated for proper transitions from a state. For example, sub-functions $D_1^3,...,D_3^3$, that have to be activated in transitions from the state a_3 (later we will also mark such sub-functions with a corresponding superscript) are the following: $D_1^3 = a_3x_1$; $D_2^3 = a_3$ (not_$x_1 \vee$ not_x_2); $D_3^3 = a_3$not_x_1 (these expressions can easily be obtained from Table 1). Note that since there exist 3 transitions from a_3, they can be distinguished with the aid of just two Boolean variables, such as $D_1^3,...,D_3^3$. As a result, inputs such as x_1 and x_2 can affect (and change) just two variables from the set $\{D_1^3,...,D_3^3\}$ and the remaining variables (in our example one variable from the set $\{D_1^3,...,D_3^3\}$) can be independent of external inputs from the set $X = \{x_1,...,x_L\}$, i.e. they will only depend on the current state (in our example on the state a_3). If the number of different (non coinciding) next states in state transitions from a_m is equal to q_m, then (R-intlog$_2q_m$) variables from the subset $D_1^m,...,D_R^m$ can be independent of the input variables from the set X.

Let us suppose now that the states for our example have been coded as shown in parentheses in Table 1. The values of the sub-functions $D_1^r,...,D_3^r$ (r=1,2,...,M, M - is the number of FSM states and for our example M=7) that do not depend on input variables, are marked with bold and italic bold fonts. Note that the bold font has been used for passive values, and italic bold font for active values of the sub-functions.

Now the combinational circuit S of the FSM can be decomposed into two sub-circuits in such a way that the first sub-circuit S_{ax} implements all active values of $D_1^1,...,D_3^M$ that are not bold. The functions of S_{ax} depend on both FSM inputs and states. The second sub-circuit S_a implements all active values of $D_1^1,...,D_3^M$ that are bold. The functions of S_a depend only on the states and for our example they are:

$D_1 = a_3 \vee a_6$; $D_2 = a_6 \vee a_7$; $D_3 = a_2 \vee a_6 \vee a_7$;

Such functions are well suited for minimization and are usually very simple.

Table 1. An example of FSM

a_{from}	$K(a_{from})$	$X(a_{from},a_{to})$	a_{to}	$K(a_{to})$	$D(a_{from},a_{to})$
a_1	000 (000)	x_1x_2	a_1	000 (**0**00)	- (-)
		not_x_1 not_x_2	a_2	001 (**0**11)	D_3 (D_2,D_3)
		not_x_1x_2	a_3	010 (**0**01)	D_2 (D_3)
		x_1 not_x_2	a_6	101 (**0**10)	D_1,D_3 (D_2)
a_2	001 (011)	x_3	a_3	010 (0**0*1***)	D_2 (D_3)
		not_x_3	a_5	100 (1**0*1***)	D_1 (D_1, D_3)
a_3	010 (001)	not_x_1	a_4	011 (***1***00)	D_2,D_3 (D_1)
		x_1x_2	a_5	100 (***1***01)	D_1 (D_1, D_3)
		x_1 not_x_2	a_7	110 (***1***11)	D_1,D_2 (D_1, D_2, D_3)
a_4	011 (100)	not_x_4	a_1	000 (0**00**)	- (-)
		x_4	a_4	011 (1**00**)	D_2,D_3 (D_1)
a_5	100 (101)	x_1	a_1	000 (**000**)	- (-)
		not_x_1	a_6	101 (**010**)	D_1,D_3 (D_2)
a_6	101 (010)	1	a_7	***111***	D_1,D_2,D_3
a_7	110 (111)	x_5	a_2	001 (0***11***)	D_3 (D_2, D_3)
		not_x_5	a_7	110 (1***11***)	D_1,D_2 (D_1, D_2, D_3)

3. Primary encoding algorithm

The objective of the algorithm we will look at now is to reduce the functional dependency of outputs on inputs. This is a typical combinatorial algorithm [9]. It is based on the use of special tables that look like Karnaugh maps, but are actually different. Let us designate $A(a_{from})$ the set of states that can be reached through direct transitions from the state a_{from}. Let us assume that for any set $A(a_{from})$ the following requirement has been satisfied:

$$\bigvee_{a_{to}\in A(a_{from})} X(a_{from}, a_{to}) \equiv 1$$

where $X(a_{from},a_{to})$ is a product of $x_1,\dots,x_L$ that forces a corresponding transition. Suppose $K(a_{from},a_{to})$ is a set of codes, for the states a_{to} and $a_{to}\in A(a_{from})$. Since the expression above is valid, if for given a_{from} bit k in all codes $K(a_{from},a_{to})$ has values either 0 and don't care (-), or 1 and don't care, then the sub-function D_k^{from} does not depend on input variables from the set X. Here D_k^{from} is the sub-function of the function D_k that forms the proper value of the function D_k in any transition to the states $a_{to}\in A(a_{from})$. This leads us to the following method of encoding. For each set $K(a_{from},a_{to})$, $a_{from}=1,\dots,M$, it is necessary to find partitioning $\pi_D^{from}=\{D_a^{from},D_{ax}^{from}\}$, for which $D_a^{from}\cup D_{ax}^{from}=\{D_1^{from},\dots,D_R^{from}\}$, $D_a^{from}\cap D_{ax}^{from}=\varnothing$, and $D_1^{from},\dots,D_R^{from}$ - correspond to bits 1,...,R of codes of $K(a_{from},a_{to})$, $|D_a^{from}|\Rightarrow\max$ (we want to find the set D_a^{from} that has the maximum possible number of elements). If $D_r^{from}\in D_a^{from}$ then D_r^{from} satisfies the requirement considered above, i.e. it depends only on a_{from}.

The encoding technique assumes special graphical shapes for different sets, such as $K(a_{from},a_{to})$, that have to be accommodated in the encoding table. These graphical shapes are the same as for Karnaugh maps. The required solution will be found after proper accommodation of appropriate shapes in the encoding table. Note that any set $A(a_{from})$ has to be accommodated in any cell within the relevant shape, such as rectangle. If the shape has more cells than required, the remaining cells might be occupied by other states, which do not have any relationship with the considered set $A(a_{from})$.

Fig. 2 demonstrates this technique for the following FSM: $A(a_1) = \{a_1,a_2,a_5,a_6\}$; $A(a_2) = \{a_1,a_3,a_{20}\}$; $A(a_3) = \{a_8,a_{10}\}$; $A(a_4) = \{a_{18},a_{19}\}$; $A(a_5) = \{a_2,a_6,a_7\}$; $A(a_6) = \{a_5,a_{11}\}$; $A(a_7) = \{a_2,a_3\}$; $A(a_8) = \{a_9,a_{10},a_{11}\}$; $A(a_9) = \{a_{15},a_{17}\}$; $A(a_{10}) = \{a_{12},a_{13},a_{14}\}$; $A(a_{11}) = \{a_3,a_{12}\}$; $A(a_{12}) = \{a_7,a_{14}\}$; $A(a_{20}) = \{a_4,a_{15},a_{16},a_{17}\}$. Here all sub-sets $A(a_{from})$ containing just one element (i.e. representing unconditional state transitions) were skipped. It is obvious that all components of $K(a_{from},a_{to})$ for such sub-sets do not depend on the input variables.

The encoding algorithm involves the following two major steps that are repeated sequentially:

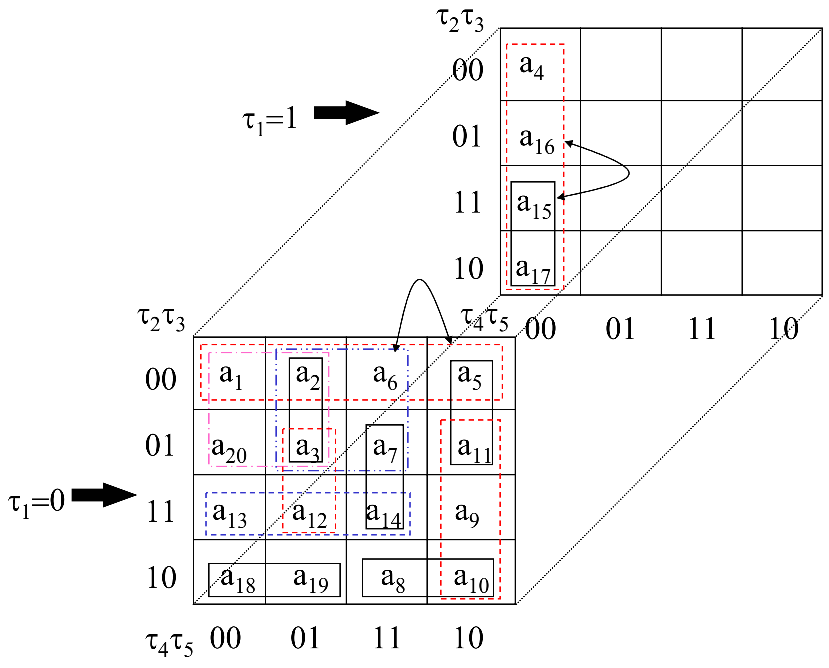


Figure 2. Mapping and swapping processes for encoding tables

- Search for the appropriate place in the encoding table for one of the acceptable graphical shapes. Note that some states can appear in the table more than once. However, we must minimize the number of different codes for the same state, For instance, the two codes 00000 and 00001 (i.e. 0000-) for the state a_m are better than two codes such as 00010 and 00001. In the last case we must repeat all the required transitions from the state a_m twice (the first time for the code 00010 and the second time for the code 00001);
- Swap states within any shape if required, i.e. when we are not able to accommodate a new subset $A(a_{from})$. There are some constraints on the swapping process and they will be examined below.

Consider the graph G_ξ, which reflects the following relationships:

$$(a_m \xi a_s) \Leftrightarrow A(a_m) \cap A(a_s) \neq \varnothing.$$

Vertices of G_ξ correspond to symbols from the set $A=\{a_1,\ldots,a_M\}$. Two vertices a_m and a_s are connected with an edge if and only if the relationship $(a_m \xi a_s)$ is satisfied. Each vertex a_m has been appended to the set $A(a_m)$ and all vertices for which $|A(a_m)|<2$ have been deleted. Each edge has been assigned the set $A(a_m,a_s)$ which is determined as follows: $A(a_m,a_s)=A(a_m) \cap A(a_s)$. Graph G_ξ, for the example considered above is shown in fig. 3.

The first step of our algorithm is divided into the following sub-steps:

1.1. Considering all subsets $A(a_1),\ldots,A(a_M)$. They have been attached to the respective vertices in fig. 3.
1.2. For each subset $A(a_m)$ (m=1,…,M) finding the place in the encoding table that satisfies our target requirement. The subsets are selected starting with one that has the largest number of elements. If the subset, that has been chosen, contains elements in common with some other subsets, then the latter will receive preference in the selection process with the same criteria as has been considered above. The subsets with common elements can easily be found with the aid of graph G_ξ. We will demonstrate this procedure below on an example.
1.3. Accommodate the chosen subset if the proper position has been found.

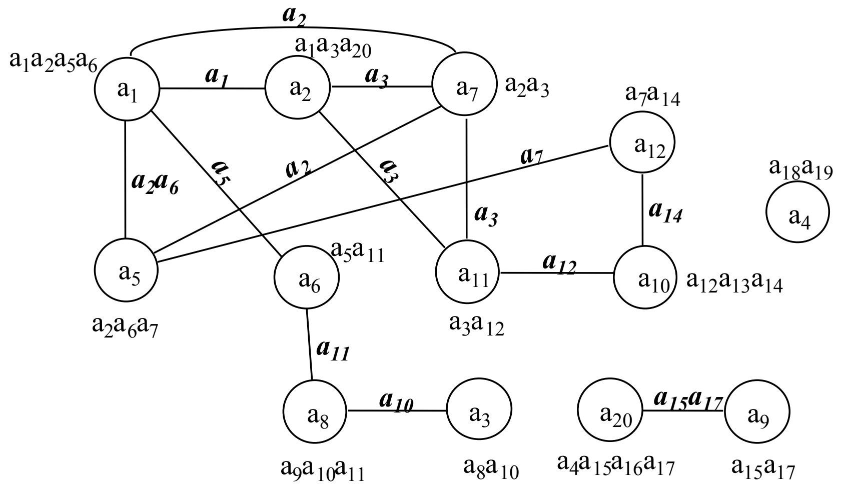


Figure 3. Graph G_ξ

If the results of the previous point are negative then it is necessary to jump to the swapping process, which will be performed at the second step.

The swapping process is performed taking into account all the constraints that can be obtained from the graph G_ξ. After that, the first step will be repeated. If the results of the first step are still negative then a new element is added to the set D_{ax}^{from} and the first step is repeated. Since we can extend D_{ax}^{from} up to $\{D_1,\ldots,D_R\}$, the problem of encoding will ultimately be solved.

The subset $A(a_1)=\{a_1,a_2,a_5,a_6\}$ can be trivially accommodated (see fig. 2). Suppose that initially $K(a_1)=00000$, $K(a_2)=00001$, $K(a_5)=00011$, $K(a_6)=00010$. Then we can accommodate the subset $A(a_2)=\{a_1,a_3,a_{20}\}$ as follows: $K(a_1)=00000$, $K(a_3)=00101$, $K(a_{20})=00100$. Note that when we look for a place we have to use information of the graph G_ξ. Let say when we accommodate a_2 we have to examine all edges connected to the vertex a_2 in order to reserve some space in the encoding table for future steps. In our example we should accommodate a_3 in neighboring cells with a_2 and it has been done in fig. 2. Now we have to accommodate the next intercepting subset with the maximum number of elements and this is $A(a_5)=\{a_2,a_6,a_7\}$. Unfortunately the set $A(a_5)$ cannot be accommodated in the table because there is no graphical shape containing four elements, which would allow us to place the state a_7. Note that the states a_2 and a_6 have already been recorded in the table. Thus, we have to perform the swapping process. There are many possible options for swapping and we will exchange a_5 and a_6 (this is shown in fig. 2 with the aid of double arrow curve). If we change the placement of an element that is attached to an edge in the graph G_ξ, then we also have to carry out some related changes (to check the other vertex to be linked to the same edge). For our example this is not necessary. Applying the same technique makes it possible to fill in the table in fig. 2. This gives us the following result: D_a^1={ D_1^1, D_2^1, D_3^1}; D_a^2={ D_1^2, D_2^2, D_4^2}; D_a^3={ D_1^3, D_2^3, D_3^3, D_4^3}; D_a^4={ D_1^4, D_2^4, D_3^4, D_4^4}; D_a^5={ D_1^5, D_2^5, D_5^5}; D_a^6={ D_1^6, D_2^6, D_4^6, D_5^6}; D_a^7={ D_1^7, D_2^7, D_4^7, D_5^7}; D_a^8={ D_1^8, D_4^8, D_5^8}; D_a^9={ D_1^9, D_2^9, D_4^9, D_5^9}; D_a^{10}={ D_1^{10}, D_2^{10}, D_3^{10} }; D_a^{11}={ D_1^{11}, D_3^{11}, D_4^{11}, D_5^{11}}; D_a^{12}={ D_1^{12}, D_3^{12}, D_4^{12}, D_5^{12}}; D_a^{20}={ D_1^{20}, D_4^{20}, D_5^{20}}; $K(a_1)$=00000; $K(a_2)$=00001; $K(a_3)$=00101; $K(a_4)$=10000; $K(a_5)$=00010; $K(a_6)$=00011; $K(a_7)$=00111; $K(a_8)$=01011; $K(a_9)$=01110; $K(a_{10})$=01010; $K(a_{11})$=00110; $K(a_{12})$=01101; $K(a_{13})$=01100; $K(a_{14})$=01111; $K(a_{15})$=11100; $K(a_{16})$=10100; $K(a_{17})$=11000; $K(a_{18})$=01000; $K(a_{19})$=01001; $K(a_{20})$=00100.

If the value of M is increased, the problem seems to be more complicated. However for the majority of practical applications the graph G_ξ can be decomposed into non-linked sub-graphs. The encoding procedure for any sub-graph, can be performed independently of the other sub-graphs using an individual encoding table that matches this sub-graph, i.e. which has a size corresponding to this sub-graph. For instance, our graph can be decomposed into three sub-graphs G_ξ^1, G_ξ^2, G_ξ^3 as follows: G_ξ^1={a_1, a_2, a_3, a_5, a_6, a_7, a_8, a_{10}, a_{11}, a_{12}}; G_ξ^2={a_9, a_{20}}; G_ξ^3={a_4}. For more complicated tasks the results of decomposition are usually better, i.e. the number of sub-graphs is greater than for our example and each sub-graph contains less vertices.

So, the process of state encoding is iterative and it is based on information that has been extracted from the auxiliary graph G_ξ. The latter might be decomposed in isolated sub-graphs. Since there is no overlapping between states belonging to different sub-graph, the problem of encoding can be solved independently for each sub-graph. For the majority of practical applications the size of encoding table for each sub-graph does not exceed 32-64 cells.

4. Encoding algorithm for block-based decomposition

Let's consider now how we can apply the encoding technique to the block-based decomposition, shown in fig. 1. We will start with the first method, which allows us to eliminate the block S_a^{T+1}. In this case the state transition table will be decomposed into sub-tables W_1,...,W_T [7]. Each of W_t contains information for constructing the respective block S_{ax}^t (see fig. 1). Now the problem can be formulated as follows. For each set $K(A_{from}(W_t),a_{to})$ ($A_{from}(W_t)$ is a set of states written in the column a_{from} of the sub-table W_t) it is necessary to find partitioning {D_a^t, D_{ax}^t}, for which $D_a^t \cup D_{ax}^t$={D_1^t,...,D_R^t}, $D_a^t \cap D_{ax}^t = \varnothing$, and D_1^t,...,D_R^t - correspond to bits 1,...,R of codes of $K(A_{from}(W_t),a_{to})$, $|D_1^t| \Rightarrow \max$ (we want to find the set D_1^t that has the maximum possible number of elements). If $D_r^t \in D_a^t$ then D_r^t satisfies the requirement considered above, i.e. it depends only on states.

In order to perform state assignment according to the rules considered above, we can use special tables μ_t (t = 1,...,T), shown in fig. 4 (table μ_t is used for the state transition table W_t). The rows of each table μ_t correspond to various possible codes of D_{ax}^t; the columns of each table μ_t correspond to various possible codes of D_a^t; all states of W_t from each set $A(a_{from}) \subseteq K(A_{from}(W_t),a_{to})$ must be accommodated in the same column of μ_t.

Suppose two blocks of our FSM have been described by sub-tables W_1 and W_2 that contain the following data: $A_{from}(W_1)$ = {a_1,a_5,a_7,a_8,a_{10}}; $K(a_1,a_{to})$ = {a_1,a_2,a_5,a_6}; $K(a_5,a_{to})$ = {a_2,a_6,a_7}; $K(a_7,a_{to})$ = {a_2,a_3}; $K(a_8,a_{to})$ = {a_9,a_{10},a_{11}}; $K(a_{10},a_{to})$ = {a_{12},a_{13},a_{14}}; $A_{from}(W_2)$ = {a_2,a_3,a_4,a_6,a_9,a_{11},a_{12}}; $K(a_2,a_{to})$ = {a_1,a_3,a_4,a_{15},a_{16},a_{17}}; $K(a_3,a_{to})$ = {a_8,a_{10}}; $K(a_4,a_{to})$ = {a_{18},a_{19}}; $K(a_6,a_{to})$ = {a_5,a_{11}}; $K(a_9,a_{to})$ = {a_{15},a_{17}}; $K(a_{11},a_{to})$ = {a_3,a_{12}}; $K(a_{12},a_{to})$ = {a_7,a_{14}}. The states a_{13},a_{14},a_{15},a_{16},a_{17},a_{18},a_{19} have not been included into $A_{from}(W_1)$ and $A_{from}(W_2)$ because there are only unconditional transitions from these states. Fig. 4 demonstrates state encoding for our example. All subsets $A(a_{to})$ are highlighted either with gray shading or with a line with a double arrow head pointing to the corresponding states. Now the resulting codes can be easily obtained. The tables μ_1 and μ_2 have been filled with the states according to the algorithm considered in the previous section. The final partitions are: {D_a^1, D_{ax}^1} = { {D_3^1,D_4^1,D_5^1}, {D_1^1,D_2^1} }; {D_a^2, D_{ax}^2} = { {D_1^2,D_2^2}, {D_3^2,D_4^2,D_5^2} }.

Let's consider now the second method, which assumes the autonomous block S_a^{T+1} and enables us to simplify the blocks S_{ax}^1,..., S_{ax}^T (see fig. 1). In this case the state transition table will also be decomposed into sub-tables W_1,...,W_T that contain information for constructing the respective block S_{ax}^t (see fig. 1). However the partitioning {D_a, D_{ax}} is considered to be common for all the sub-tables and the sub-functions from D_a have to be implemented in an autonomous block S_a^{T+1}. The simplification of the partitioning makes it possible to simplify the whole problem, which nevertheless has many features in common with the previously considered problem. Suppose that for given sub-tables $W_1,...,W_T$ we have built the graph G_ξ. For example, the graph G_ξ shown in fig. 3 was built for sub-tables W_1, W_2 and $A_{from}(W_1)=\{a_1,a_5,a_7,a_8,a_{10}\}$; $A_{from}(W_2)=\{a_2,a_3,a_4,a_6,a_9,a_{11},a_{12},a_{20}\}$.

Let us build a new compressed graph $\widetilde{G}_\xi$, which contains the joined vertices of the G_ξ. The vertices a_m, $a_s,...,a_k$ can be joined if and only if $|A(a_m)\cup A(a_s)\cup...\cup A(a_k)|\le 2^r$ where r – is a predefined constraint [7] (this is the number of outputs of S_{ax}). If a_m, $a_s,...,a_k$ are joined, the new common vertex corresponds to the set $A(a_m)\cup A(a_s)\cup...\cup A(a_k)$. Each edge of $\widetilde{G}_\xi$ connecting vertices v_m and v_s has the weight $\rho(v_m,v_s)$ which is calculated as follows:

$$\rho(v_m,v_s) = \sum_{a_k \in A(v_m,v_s)} |A(a_k)|,$$

where $A(v_m,v_s)$ is the set of common states in vertices v_m and v_s.

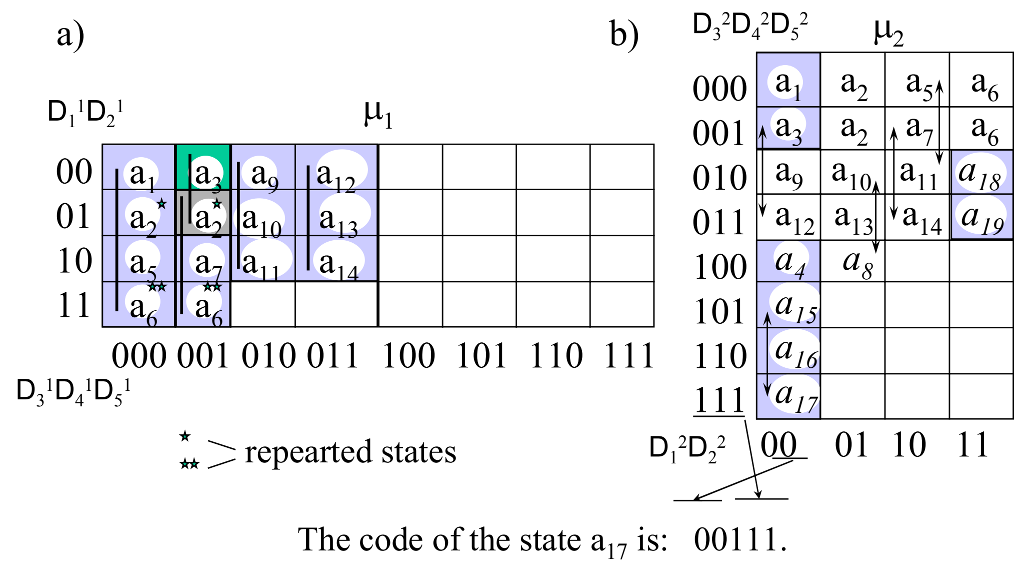

a)
$D_1^1D_2^1$
μ_1
00
01
10
11
a_1 a_3 a_9 a_{12}
a_2^* a_2^* a_{10} a_{13}
a_5^{**} a_7^{**} a_{11} a_{14}
a_6 a_6
000 001 010 011 100 101 110 111
$D_3^1D_4^1D_5^1$
repearted states
b)
$D_3^2D_4^2D_5^2$
μ_2
000 a_1 a_2 a_5 a_6
001 a_3 a_2 a_7 a_6
010 a_9 a_{10} a_{11} a_{18}
011 a_{12} a_{13} a_{14} a_{19}
100 a_4 a_8
101 a_{15}
110 a_{16}
111 a_{17}
$D_1^2D_2^2$
00 01 10 11
The code of the state a_{17} is: 00111.

Figure 4. Encoding tables for block-based decomposition: a) for the sub-table W_1; b) for the sub-table W_2

The final graph $\widetilde{G}_\xi$ is shown in fig. 5. Let us mark the number of vertices of $\widetilde{G}_\xi$ with δ. Our task can be solved if $\delta\le 2^{R-r}$. In most circumstances we can satisfy this constraint. Now the problem of state encoding can be transformed into the mapping of graph $\widetilde{G}_\xi$ onto an R-r dimensional cube C_μ (see fig. 6a). If we can solve this problem we can directly fill in the single encoding table μ (see fig. 6b).

Some states, such as a_1, a_2, a_5, a_6, a_7, a_{10}, a_{12} are repeated twice in table μ. They are located in neighboring corners of C_μ and therefore they are assigned codes having don't care components (we have already mentioned this possibility earlier). Just one (underlined) state a_3 is located in the diagonal of C_μ, and a_3 was assigned two different binary codes that are 10001 and 00010. This is allowed, but all the transitions from the state a_3 are repeated twice (from two states 10001 and 00010). That is why when we map graph $\widetilde{G}_\xi$ onto the C_μ we try to minimize the total weight of the diagonal (non-neighboring) edges.

The results of state encoding can be easily obtained from fig. 6b.

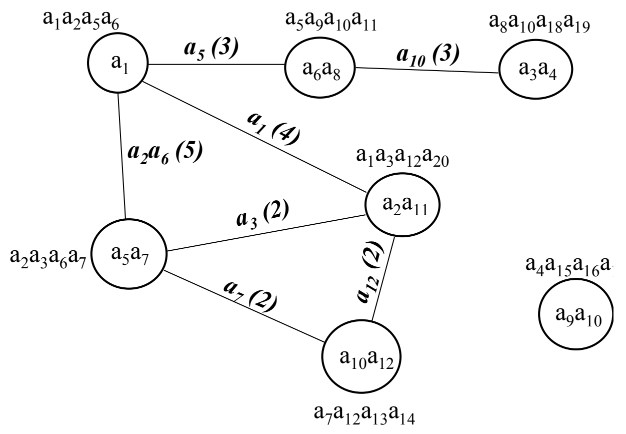
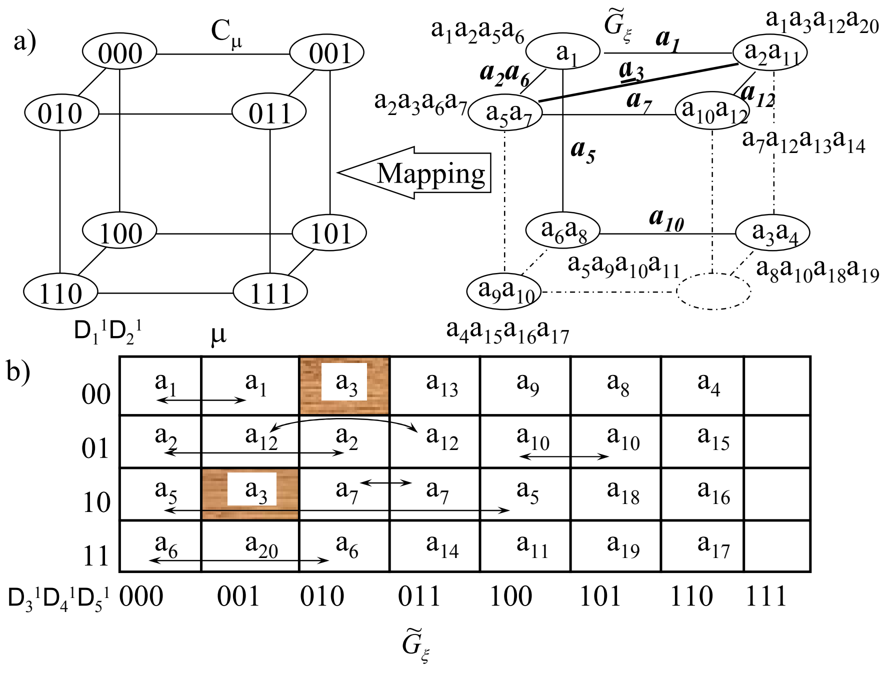


Figure 5. Compressed graph



$D_1^1D_2^1$ \ $D_3^1D_4^1D_5^1$	000	001	010	011	100	101	110	111
00	a_1	a_1	a_3	a_{13}	a_9	a_8	a_4	
01	a_2	a_{12}	a_2	a_{12}	a_{10}	a_{10}	a_{15}	
10	a_5	a_3	a_7	a_7	a_5	a_{18}	a_{16}	
11	a_6	a_{20}	a_6	a_{14}	a_{11}	a_{19}	a_{17}	

Figure 6. Mapping of the graph $\tilde{G}_\xi$ onto the cube C_μ (a) and the table μ (b)

5. Experimental results

The proposed encoding algorithms were applied to more than 100 FSMs. Fig. 7 shows the results obtained for 25 FSMs. We considered block-based decomposition of FSMs and R=intlog$_2$M, R_{av} is an average number of outputs for the blocks. So, the considered technique makes possible the number of outputs required for block-based FSMs to be decreased in average by 1.8.

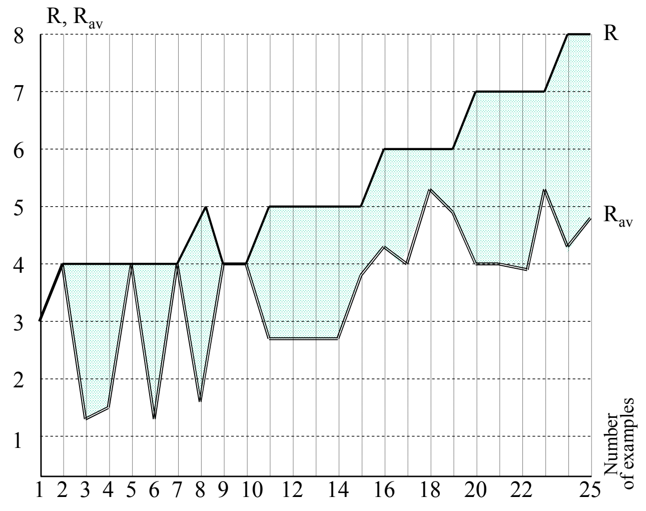


Figure 7. The results of experiments

Table 2 contains the results of the design of RAM-based FSMs [9] before (see the column RAM_{before}, where it is shown the size of the required RAM in bits) and after (see the column RAM_{after}) applying of the considered encoding technique. Here the R_{min} is the minimal possible size of FSM memory, R_{real} is the size of FSM memory after applying the considered encoding technique.

We checked also applicability of the considered method for RAM-based FSM implemented in FPGAs [10]. For such purposes we used dual port embedded memory blocks available for FPGAs of Spartan/Virtex families. Thus, the first port took part in the FSM state transitions whilst the second port was used to reprogram RAM from the PC via the USB port.

Table 2. The results os experiments with RAM-based FSMs

Ex.	M/L/N	R_{min}	R_{real}	RAM_{before}	RAM_{after}
1	5/2/4	3	3	224	56
2	8/4/3	3	4	768	112
3	10/8/5	4	4	36864	288
4	15/2/4	4	5	512	288
5	19/7/10	5	5	61440	480
6	24/11/26	5	5	2031616	992
7	27/6/27	5	6	65536	2112
8	22/18/34	5	6	327155712	10240
9	29/18/35	5	6	335544320	10496
10	30/14/27	5	6	16777216	4224

Note that the circuit S_a^{T+1} can easily be implemented on the basis of RAM-blocks. Thus, the remaining circuits ($S_{ax}^1,\ldots,S_{ax}^T$) can be simplified (because some of their functionality has already been implemented in S_a^{T+1}). Finally, it allows simplifying circuits described in [10]. Besides it makes possible to construct FSMs with more regular structure.

For all considered examples the total number of CLBs (configurable logic blocks) needed for the FSM circuits was reduced on average by 20%.

6. Conclusion

In the previous discussion we have presented encoding algorithms that allow Boolean functions to be decomposed in such a way that the dependency of sub-functions, obtained as a result of the decomposition, on the arguments has been reduced. These algorithms can be efficiently used for many different logic synthesis problems. The examples in the paper and the results of experiments have shown that using these algorithms has allowed the number of logic circuits required for an FSM to be decreased.

7. References

[1] A.J.Menezes, P.C. van Oorschot, "Coding Theory and Cryptology", Discrete and Combinatorial Mathematics, editor-in-chief K.H. Rosen, CRC Press, 2000, pp. 889-954.

[2] A. Villa, A.Sangiovanni-Vincentelly, "NOVA: State Assignment of Finite State Machines for Optimal Two Levels Logic Implementations", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 9, 1990, pp. 905-924.

[3] X. Du, G. Hatchel, B. Lin, A.R. Newton, "MUSE: A Multilevel Symbolic Encoding Algorithm for State Assignment", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 10, 1991, pp. 28-38.

[4] T. Villa, T. Kam, R.K. Brayton, A. Sangiovanni-Vincentelli, Synthesis of Finite State Machines: Logic Optimization, Kluwer Academic Publishers, 1997

[5] V. Sklyarov, "Synthesis and Implementation of RAM-based Finite State Machines in FPGAs", Proceedings of FPL'2000, Villach, Austria, August, 2000, pp. 718-728.

[6] S. Baranov, Logic Synthesis for Control Automata, Kluwer Academic Publishers, 1994.

[7] V. Sklyarov, A. Rocha, A. Ferrari, "Synthesis of Reconfigurable Control Devices Based on Object-Oriented Specification", Advanced Technique for Embedded System Design and Test, Kluwer Academic Publishers, pp. 151-177, 1998.

[8] V. Sklyarov, I. Skliarova, "Evolutionary Algorithm for State Encoding", International Federation for Information Processing, vol. 217, 1st IFIP International Conference on Artificial Intelligence in Theory and Practice - AI'2006, ed. M. Bremer, 19th IFIP World Computer Congress - WCC'2006, Santiago de Chile, Chile, August 2006, pp. 227-236.

[9] D.L. Kreher, D.R. Stinson, Combinatorial Algorithms. CRC Press, 1999, 329 p.

[10] V. Sklyarov, "Reconfigurable models of finite state machines and their implementation in FPGAs", Journal of Systems Architecture, 47, 2002, pp. 1043-1064.