Finite State Machines With Input Multiplexing: A Performance Study

Ignacio Garcia-Vargas and Raouf Senhadji-Navarro

***Abstract*—Finite state machines with input multiplexing (FSMIMs) have been proposed in previous works as a technique for efficient mapping FSMs into ROM memory. In this paper, we propose a new architecture for implementing FSMIMs, called FSMIM with state-based input selection, whose goal is to achieve a further reduction in memory usage. This paper also describes in detail the algorithms for generating FSMIMs used by the tool FSMIM-Gen, which has been developed and made available on the Internet for free public use. A comparative study in terms of speed and area between FSMIM approaches and other field programmable gate array-based techniques is presented. The results show that the FSMIM approaches obtain huge reductions in the look-up table (LUT) usage by using a small number of embedded memory blocks. In addition, speed improvements over conventional LUT-based implementations have been obtained in many cases.**

***Index Terms*—Embedded memory blocks (EMBs), finite state machine (FSM), field programmable gate array (FPGA), logic synthesis, ROM.**

I. Introduction

In the last decade, the number of embedded memory blocks (EMBs) available in field programmable gate arrays (FPGAs) has increased greatly. The development of efficient techniques for implementing finite state machines (FSMs) using EMBs is a great challenge [1]–[6]. The reported advantages of ROM-based FSM implementations make it an interesting alternative to the conventional LUT-based implementations. Firstly, the use of EMBs frees look-up tables (LUTs) that can be used for other general purposes [5]. Secondly, speed improvements have been obtained by the fact that EMBs have a fixed access memory time independently of its content [6]. Finally, a considerable power consumption reduction can be achieved by disabling EMBs during the idle states [4].

Most of the approaches for enhancing the performance of ROM-based FSM implementations rely on a functional decomposition of the memory component into two elements: 1) a combinational address modifier and 2) a smaller memory component [1], [5], [7]. Senhadji-Navarro *et al.* [8] presented the fundamentals of a new approach called FSM with input multiplexing (FSMIM) whose main goal is to reduce the ROM memory depth. This approach includes optimization techniques and an architecture, hereinafter called FSMIM with transition-based input selection (FSMIM-T), which uses a multiplexer bank as address modifier. In [6], significant speed improvements and area reductions have been obtained by FSMIM implementations on FPGAs due to the following two facts: 1) typically, a state transition involves many don't care inputs [9] and 2) current FPGAs allow very efficient implementations of wide multiplexers by using dedicated multiplexers [10].

In this paper, we have made several new contributions with respect to those presented in [6] and [8]. Firstly, we describe in detail the optimization process involved in the FSMIM implementations. Secondly, we propose a new architecture to implement FSMIMs, called FSMIM with state-based input selection (FSMIM-S), with the aim of further reducing the size of the ROM memory. Finally, we present a comparative study between FSMIM and others techniques. A tool called FSMIM-Gen for generating FSMIMs from FSMs have been developed and distributed as open-source [11].

The authors are with the Departamento de Arquitectura y Tecnología de Computadores, Universidad de Sevilla, E.T.S. Ingeniería Informática, Sevilla 41012, Spain (e-mail: raouf@us.es).

II. Conventional ROM-Based Implementation

The transition and output functions of an FSM can be implemented using memory [12]. In this paper, we assume Mealy machines with synchronous outputs because the EMBs available in current FPGAs are synchronous. Fig. 1(a) shows the reference architecture for ROM-based implementations of Mealy machines. The ROM stores the FSM outputs and the next state of each FSM transition. The ROM size in bits is

$$C_{\text{ROM}} = 2^m|S|(n+p) \leq 2^{m+p}(n+p) \tag{1}$$

where S is the set of states, m is the number of inputs, n is the number of outputs, and $p = \lceil \log_2|S| \rceil$ is the number of state encoding bits. The memory usage grows exponentially with the number of inputs and the number of state encoding bits [12]. The speed is degraded due to routing overhead when a large number of EMBs is required. Moreover, if the memory depth exceeds the maximum depth of the EMBs, then the speed is further reduced because some LUTs are used for implementing the multiplexers required to join the EMBs.

III. FSMIM

The FSMIM approach tries to reduce the depth of the ROM memory by reducing both the m and $|S|$ values in (1). Let us define the effective inputs of a state s as the subset of FSM inputs that are relevant to determine the state transitions of s. For any given state, the ROM could be addressed using only the effective inputs of the state instead of all FSM inputs. If the maximum number of effective inputs per state (denoted by m') is less than m, then the ROM depth can be reduced from $2^m|S|$ to $2^{m'}|S|$. For this purpose, a combinational component called input selector bank (ISB) is used to select the effective inputs of each state. The ISB is composed by m' elements called input selectors. For each state, each input selector selects one different effective input of the state from a subset of the FSM inputs. The ISB selects m' inputs no matter how many effective inputs the present state has. For those states that have less effective inputs than m', only part of the selected inputs are effective; the rest are don't care values for the state. We will refer to them as don't care selected inputs (DCSIs). DCSIs can be exploited to form groups of states that can be encoded with the same code. So, if all states are grouped into N groups, then the ROM depth can be reduced to $2^{m'}N$ with $N \leq |S|$. This allows to reduce the depth even in FSMs with states sensitive to all inputs (i.e., $m' = m$).

An optimization process is used to transform FSMs into FSMIMs which can be implemented efficiently using both the FSMIM-S and FSMIM-T architectures.

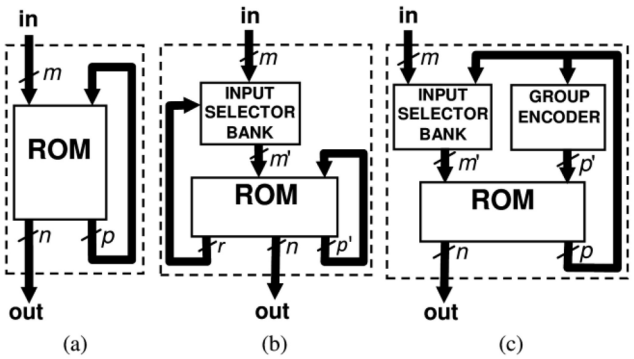


Fig. 1. ROM-based FSM architectures. (a) Conventional. (b) FSMIM-T. (c) FSMIM-S.

$$A = \begin{bmatrix} x_6 & x_5 & x_1 & s_0 \\ x_5 & x_1 & - & s_1 \\ x_1 & - & - & s_2 \\ - & - & x_2 & s_3 \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{bmatrix} \quad A^{ISS} = \begin{bmatrix} x_1 & x_5 & x_6 & s_0 \\ x_1 & x_5 & - & s_1 \\ x_1 & - & - & s_2 \\ x_2 & - & - & s_3 \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{bmatrix}$$

(a)

State	$S^2S^1S^0$
S_0	000
S_1	001
S_2	010
S_3	011
S_4	100
S_5	101

(b)

$$A_1^{SG} = \begin{bmatrix} x_1 & x_5 & x_6 & s_0 \\ x_1 & x_5 & - & s_1 \\ x_1 & 0 & - & g_{23} \\ x_2 & 1 & - & g_{23} \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{bmatrix} \quad A_2^{SG} = \begin{bmatrix} x_1 & x_5 & x_6 & s_0 \\ x_1 & x_5 & 0 & g_{123} \\ x_1 & 0 & 1 & g_{123} \\ x_2 & 1 & 1 & g_{123} \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{bmatrix} \quad A_3^{SG} = \begin{bmatrix} x_1 & x_5 & x_6 & s_0 \equiv g_0 \\ x_1 & x_5 & 0 & g_{123} \\ x_1 & 0 & 1 & g_{123} \\ x_2 & 1 & 1 & g_{123} \\ x_2 & x_4 & 0 & g_{45} \\ x_3 & x_4 & 1 & g_{45} \end{bmatrix}$$

(c)

Group	G^1G^0
g_0	00
g_{123}	01
g_{45}	10

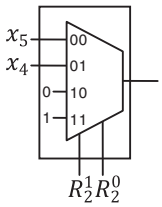
(d)

$S^2S^1S^0$	G^1G^0
000	00
001	01
010	01
011	01
100	10
101	10
110	–
111	–

(e)

$S^2S^1S^0$	Output
000	x_5
001	x_5
010	0
011	1
100	x_4
101	x_4
110	--
111	--

(f)



(g)

Fig. 2. Example of FSMIM generation. (a) ISS. (b) State encoding. (c) SG. (d) Group encoding. (e) Truth table of the GE of FSMIM-S. (f) Truth table of the second input selector of A_3^{SG} for FSMIM-S. (g) Second input selector of A_3^{SG} for FSMIM-T (R_2^1 and R_2^0 are selection bits).

A. Optimization Process

Before describing the optimization process, we introduce the input selection matrix (ISM), which represents the relationship between states and effective inputs. Each row of the ISM contains the effective inputs of a state and each column contains the FSM inputs connected to an input selector. Let $S = \{s_1, s_2, \ldots, s_q\}$ and $X = \{x_1, x_2, \ldots, x_m\}$ be the sets of states and inputs, respectively. Let us define an ISM as a matrix $A = (a_{ij}) \in \mathcal{M}_{q \times m'}$, where $a_{ij} \in X \cup \{0, 1, -\}$ represents the input selected by the jth input selector for the state s_i. The values 0 and 1 indicate that the input selector selects the constant value 0 and 1, respectively, instead of an FSM input. For clarity, the ISM matrix includes an additional column for representing either the state s_i or the group to which s_i belongs. Fig. 2(a) shows the ISM of a FSMIM with six states and three input selectors (see the matrix A).

The optimization process starts with the input selector simplification (ISS) procedure, which reduces the complexity of the ISB. In general terms, input selectors with less number of inputs have better performance in terms of speed and area. As the ISB is in the critical path, the speed of the FSMIM implementation can be enhanced by the ISS procedure. The proposed algorithm is based on the classical dynamic-programming-based solution of the Knapsack problem [13]. We have modified this algorithm in order to include conflicts between items so that two items with conflicts cannot be stored in the Knapsack. We will refer to it as Knapsack with conflicts (KC) problem. Unlike the classical algorithm, the KC problem does not guaranty an optimal solution. Initially, each item represents an FSM input. Two inputs have a conflict if they are effective inputs of the same state. For example, in Fig. 2(a), the inputs x_1, x_5, and x_6 have a conflict because they are effective inputs of the state s_0. The profit of each item (i.e., each FSM input) is equal to its weight. The weight represents the number of states for which the input is effective. A different Knapsack is required for each input selector (or ISM column), so the ISS procedure requires m' instances of the KC problem. The subset of inputs assigned to each input selector is determined by solving its corresponding KC problem. In each step of the algorithm, one instance of the KC problem is solved. In the ith step, the items not stored in the previous steps $(0, 1, \ldots, i-1)$ are used for the current KC problem. If it is not possible to include all of the items in m' Knapsacks, then an item is selected to be partitioned into two different items which are treated as different inputs although represent the same original input. So, the weights of the new items are reduced and some conflicts could be eliminated. For example, in Fig. 2(a), the input x_1 can be partitioned into the items x_1^1 (the effective input x_1 of the state s_0) and x_1^2 (the effective input x_1 of states s_1 and s_2). So, x_1^1 and x_1^2 have a weight of 1 and 2, respectively; and there are no conflict between x_6 and x_1^2. The algorithm selects the partitioning that removes the greatest number of conflicts, updates the weights and the conflict relation, and tries again to solve the m' instances of the KC problem.

Fig. 2(a) shows an example of the ISS procedure. The matrix A is the initial ISM, which has six states and three input selectors (with five, three, and two input signals). The matrix A^{ISS} is the ISM obtained after the ISS procedure. The algorithm for the KC problem has been applied three times (one for each input selector). In the first round, the inputs x_1, x_2, and x_3 have been assigned to the first input selector (first column). The weights of x_1, x_2, and x_3 are 3, 2, and 1, respectively; this makes a total weight of 6 for the Knapsack related to the first column (note that 6 is the maximum value). In addition, x_1, x_2, and x_3 do not have conflicts between them. Similarly, the algorithm tries to allocate the rest of the items (x_4, x_5, and x_6) in the rest of the columns (one by one). In this example, no partition has been required. As a result of the ISS procedure, the input selectors have three, two, and one input signals.

After the ISS procedure, the state grouping (SG) procedure is applied to further reduce the memory depth which can also result in speed improvements [12]. A group of states can be encoded with the same encoding bits if exists an assignment of values 0 and 1 to the DCSIs that allows to allocate the transitions of these states in different words of the ROM. We will refer to these groups of states simply as groups. The constant values 0 and 1 must be selected by the ISB in the same way that the FSM inputs (note that the ISM include these values). For example, the states s_2 and s_3 of A^{ISS} [see Fig. 2(a)] have been identified by the same group g_{23} due to the second input selector selects 0 for s_2 and 1 for s_3 [see A_1^{SG} in Fig. 2(c)]. Let us say that the states s_2 and s_3 have been grouped into the group g_{23}. So, in the FSM transitions, the next states s_2 and s_3 are replaced by the next group g_{23} with the input selections $(x_1, 0, -)$ and $(x_2, 1, -)$, respectively.

Given an ISM, the SG procedure can be summarized as follows. The algorithm processes the different columns of the ISM one by one. The DCSIs of each column are set to 0 or 1 in order to create new groups. The ISM is updated before the next column is processed. Both the order in which the columns are processed and the order in which the groups are created have influence in the final number of groups obtained. FSMIM-Gen uses different sorting strategies and selects the best solution. The distribution of the DCSIs in the ISM determines the minimum number of groups that can be obtained. However, the distribution of the DCSIs is obtained by the ISS procedure and it is not changed by the SG procedure to avoid increasing the complexity of the ISB.

Fig. 2(c) shows an example of the SG procedure applied to A^{ISS} [see Fig. 2(a)]. Initially, the number of groups is equal to the number of FSM states ($G = \{s_0, s_1, s_2, s_3, s_4, s_5\}$). In the first step, s_2 and s_3 are grouped into g_{23} by fixing the DCSIs of the second column (see A_1^{SG}). Next, g_{23} and s_1 are grouped into g_{123} by fixing the related DCSIs to 0 and 1 (see A_2^{SG}). Finally, s_4 and s_5 are grouped into g_{45} (see A_3^{SG}). The SG procedure reduces the number of groups from 6 to 3 ($G = \{s_0, g_{123}, g_{45}\}$). So, the number of encoding bits is reduced from 3 to 2.

B. FSMIM-T Architecture

The FSMIM-T architecture is shown in Fig. 1(b). In this architecture, the ISB is a multiplexer bank. The bits used to control the multiplexers, called selection bits, are stored in the ROM memory. Fig. 2(g) shows the input selector of the second column of A_3^{SG} [see Fig. 2(c)]. For each FSM transition, the ROM contains the FSM outputs, the next group, and the selection bits for the next state. The ROM size in bits is

$$C_{\text{FSMIM-T}} = 2^{m'}|G|\left(n + p' + r\right) \leq 2^{m'+p'}\left(n + p' + r\right) \qquad (2)$$

where G is the set of groups, $p' = \lceil \log_2 |G| \rceil$ is the number of group encoding bits, and r is the number of selection bits. In Fig. 2(c), $r = 6$ for A_3^{SG}. Each state encoding bit saved by grouping states requires the values 0 and 1 to be assigned to at least one different ISM column. Moreover, a column with constants needs at least two selection bits [e.g., in Fig. 2(c), the third column of A_3^{SG} requires two selection bits to select 0, 1, or x_6]. So, $p' + r > p$ (i.e., the ROM width increases). However, the width increment has less influence on the speed and size of the ROM than the depth reduction [12]. The goal of the FSMIM-T architecture is to use a very efficient implementation of the ISB in terms of speed and area. This is achieved by using multiplexers, which can be implemented efficiently in current FPGAs [10]. The inclusion of the selection bits in the ROM allows to reach the objective at the expense of increasing the ROM width with respect to the conventional ROM-based implementations (CONV-ROMs).

C. FSMIM-S Architecture

The FSMIM-S architecture is shown in Fig. 1(c). This architecture has been proposed to allow the implementation of FSMIM with less memory resources than those required by FSMIM-T. The address modifier is composed by two combinational components: the ISB and the group encoder (GE). Unlike the ISB of FSMIM-T, the ISB of FSMIM-S is controlled by the present state encoding bits. GE is a combinational component with p inputs and p' outputs that generate the encoding bits of the group to which the present state belongs. The performance of ISB and GE depends on the state and group encoding. Fig. 2 shows the truth tables of the GE [see Fig. 2(e)] and of the second input selector [see Fig. 2(f)]. Both are obtained from A_3^{SG} [see Fig. 2(c)] by using the encoding of states and groups shown in Fig. 2(b) and (d), respectively.

TABLE I
PARAMETERS OF THE FSMS USED IN THE EXPERIMENTS

	m	$\frac{m'}{m}$	p	$\frac{p'}{p}$	r	n	$\lvert S\rvert$	$\frac{\lvert G\rvert}{\lvert S\rvert}$	D	$\frac{D'}{D}$	LUT	Frq	EMB
Mean	7.1	0.8	5.0	0.8	7.4	7.3	32.5	0.6	12614	0.35	305	258	10.6
Std	2.0	0.3	1.2	0.2	3.2	4.6	21.5	0.3	34681	0.35	618	95	22.1
Min	4.0	0.2	3.0	0.4	2.0	2.0	6.0	0.1	480	0.001	9	76	1.0
Q1	5.5	0.5	4.0	0.7	6.0	4.0	14.0	0.4	1536	0.04	31	180	1.0
Q2	8.0	0.8	6.0	0.8	7.0	7.0	33.0	0.7	2432	0.12	92	257	2.0
Q3	8.0	1.0	6.0	1.0	9.5	9.5	48.0	0.8	5632	0.71	358	323	7.0
Max	12.0	1.0	7.0	1.0	17.0	19.0	128.0	1.0	188416	0.98	4100	467	109.0

D: ROM depth of CONV-ROM (# words); ***D'***: ROM Depth of FSMIM (# words); **LUT**: # LUTs of LUT-ISE; **Frq**: Speed of LUT-ISE (MHz); **EMB**: # 18Kbits EMBs of CONV-ROM.

Like in the conventional ROM-based architecture, each word of the ROM stores the FSM next state and outputs. So, this architecture can reduce the depth of the ROM with respect to the conventional ROM-based architecture without increasing the width. The ROM size in bits is

$$C_{\text{FSMIM-S}} = 2^{m'}|G|(n + p) \leq 2^{m'+p'}(n + p). \qquad (3)$$

From (1) to (3), it can be seen that $C_{\text{FSMIM-S}} \leq \min\{C_{\text{FSMIM-T}}, C_{\text{ROM}}\}$. This memory reduction is achieved at expense of increasing the number of used LUTs due to the implementation of a more complex address modifier. So, FSMIM-S is usually slower than FSMIM-T.

IV. EXPERIMENTAL RESULTS

In this paper, we have used Xilinx ISE Design Suite 13.4 and the xc6slx75-3 FPGA device (Virtex-6). This device includes 172 EMBs of 18 Kbits which can be configured as two independent EMBs of 9 Kbits. The techniques CONV-ROM, ISE LUT-based implementation (LUT-ISE), ISE EMB-based implementation (EMB-ISE), FSMIM-T implementation, and FSMIM-S implementation have been compared. The FSMIM technique can be applied to a FSM if $m' < m$ and/or if two or more states can be grouped after the ISS procedure (i.e., there is at least one ISM column with at least two DCSIs). We have used MCNC benchmarks [14] and 150 FSMs generated by BenGen tool [15]. We have discarded the cases in which the FSMIM cannot be applied and the cases whose CONV-ROM requires only one 9 Kbits EMB (i.e., no further reduction can be obtained). The final number of FSMs used in the experiments was 73. However, ten of these cases cannot be implemented in the target FPGA device using CONV-ROM and/or EMB-ISE due to their high resource requirements. These cases have been excluded from the tables to allow a proper comparison (we will refer to them as unsuccessful cases). Table I shows the mean, standard deviation, minimum value, quartiles, and maximum value of the architectural parameters and of the speed and area results of the reference techniques (i.e., CONV-ROM and ISE-LUT). When a residual 9 Kbits EMB is used by ISE, it is computed as 0.5 EMB. The parameters of FSMIM are shown normalized with respect to the corresponding parameters of the FSM.

Firstly, we compare the speed increment of each technique with respect to LUT-ISE because it is the standard technique and it does not use EMBs (see Table II). The statistic measures shown in this and other similar tables are: mean, standard deviation, minimum value, quartiles, and maximum value. Both FSMIM-T and CONV-ROM achieve a positive average increment. Although CONV-ROM obtains the best results, the percentage of the cases where each technique is faster than LUT-ISE is slightly greater for FSMIM-T (see "Hit" column). In fact, FSMIM-T achieves the highest operating frequency in the 11% of the cases. As expected, FSMIM-S gets

TABLE II
SPEED INCREMENT WITH RESPECT TO LUT-ISE (%)

	Mean	Std	Min	Q1	Q2	Q3	Max	Hit [a]
CONV-ROM	19.7	51.0	-70.6	-20.6	3.8	64.6	140.6	50.8
FSMIM-T	12.8	34.9	-43.5	-17.3	4.1	41.2	89.5	52.4
FSMIM-S	-4.8	31.0	-61.4	-29.4	-12.4	15.0	72.2	38.1
EMB-ISE	-36.2	17.1	-75.4	-45.4	-38.5	-25.6	17.9	1.6

[a]Percentage of the cases where the technique is faster than LUT-ISE

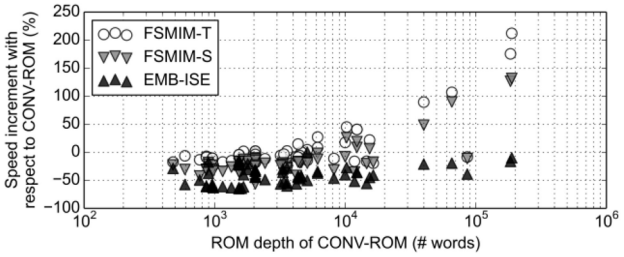


Fig. 3. Speed increment in percentage of the ROM-based techniques with respect to CONV-ROM versus the ROM depth of CONV-ROM.

TABLE III
REDUCTION OF THE NUMBER OF USED EMBS WITH RESPECT TO CONV-ROM (%)

	Mean	Std	Min	Q1	Q2	Q3	Max
EMB-ISE	-147.3	129.9	-550.0	-228.6	-125.0	-50.0	50.0
FSMIM-T	34.8	53.3	-100.0	0.0	50.0	78.4	99.5
FSMIM-S	50.6	36.8	0.0	0.0	50.0	83.8	99.5

TABLE IV
REDUCTION OF THE NUMBER OF USED LUTS WITH RESPECT TO LUT-ISE (%)

	Mean	Std	Min	Q1	Q2	Q3	Max
EMB-ISE	-37.4	88.6	-298.5	-66.1	-23.6	29.3	98.2
FSMIM-S	81.1	21.3	-53.7	74.8	83.6	95.3	99.7
FSMIM-T	92.1	9.8	55.6	90.6	95.1	98.9	≈100.0
CONV-ROM	97.6	10.8	38.9	100.0	100.0	100.0	100.0

TABLE V
NUMBER OF SAVED LUTS PER USED EMB

	Mean	Std	Min	Q1	Q2	Q3	Max
EMB-ISE	-5.6	22.8	-107.7	-7.4	-1.4	5.1	26.7
CONV-ROM	91.5	122.8	0.3	10.6	26.5	135.1	448.0
FSMIM-T	119.2	110.1	10.0	47.5	77.0	154.5	447.0
FSMIM-S	142.5	134.0	-22.0	44.0	91.0	225.8	619.5

TABLE VI
COMPARATIVE OF FSMIM-S RESULTS WITH RESPECT TO FSMIM-T (%)

	Mean	Std	Min	Q1	Q2	Q3	Max
EMB reduction	33.7	20.8	0.0	20.0	33.3	50.0	68.8
LUT increment	337.7	286.8	0.0	120.0	300.0	466.7	1475.0
Saved LUT/EMB inc. [a]	45.3	59.4	-218.9	14.0	45.5	89.9	178.9
Saved LUT reduction	11.7	22.4	0.0	2.7	5.0	12.8	159.5
Speed reduction	13.3	10.8	-19.8	7.1	13.1	19.2	47.2

[a]Increment of the number of saved LUTs per used EMB

worse results than FSMIM-T because the address modifier is more complex. Despite this, FSMIM-S is faster than LUT-ISE in the 38% of the cases (the increment is greater than or equal to 15% for the 25% of the cases). Clearly, the worst results are obtained by EMB-ISE.

Compared to CONV-ROM, the FSMIM-S and FSMIM-T techniques are faster in 13% and 30% of the cases, respectively. In addition, as shown in Fig. 3, the speed increment of the FSMIM-based architectures with respect to CONV-ROM tends to increase with the increasing of the ROM depth of CONV-ROM. This shows the impact of the ROM depth reduction on speed. In fact, for the five cases in which the ROM depth is greater than the maximum EMB depth, the average speed increment of FSMIM-S and FSMIM-T are 115% and 131%, respectively. In the unsuccessful cases, the ROM depth are always greater than 10^5; so, in a larger target device, it is expected further increments.

Let us now present a comparison of area between the different techniques. First, we analyze the EMB reduction of the different ROM-based techniques with respect to CONV-ROM (see Table III). Again, the worst technique is EMB-ISE. FSMIM implementations use much less memory than the other techniques (in the half of the cases, the EMB reduction is greater than or equal to 50% for both FSMIM-based architectures). As expected, FSMIM-S always uses a number of EMBs less than or equal to FSMIM-T. We highlight that, in six of the unsuccessful cases, CONV-ROM requires more EMBs than those available in any device of the Virtex-6 family while FSMIM-S and FSMIM-T require only 2.9 and 5.6 EMBs on average, respectively.

In order to evaluate the number of LUTs saved in the implementation of FSMs by using EMBs, we have calculated the reduction of the number of used LUTs with respect to LUT-ISE (see Table IV). The worst technique is EMB-ISE, which uses more LUTs than LUT-ISE on average despite the fact that it uses a considerable number of EMBs. The average reduction is greater than 80% for CONV-ROM and FSMIM-based techniques. Therefore, in these techniques, the use of EMBs allows a huge reduction of the LUT usage. As expected, CONV-ROM obtains the best results. However, the average reduction of FSMIM-T is very close to that of CONV-ROM despite the fact that CONV-ROM do not include any address modifier. We have also calculated the number of saved LUTs per used EMB for each technique (see Table V). Although both FSMIM implementations achieve better results than CONV-ROM, FSMIM-S is the most effective technique for saving LUTs.

Table VI shows a comparison between FSMIM-S and FSMIM-T. We have used the cases in which FSMIM-T requires more than one 9 Kbits EMB (the unsuccessful cases have also been included). FSMIM-S reduces significantly the number of EMBs at expense of a little speed reduction. Although the relative LUT increment of FSMIM-S is high, the number of used LUTs (22 on average) is not significant compared to the number of saved LUTs (446 on average). The number of saved LUTs of FSMIM-S is only 12% less than FSMIM-T on average; however, it is important to highlight that FSMIM-S saves 45% more LUTs per used EMB.

V. CONCLUSION

Regarding the area results, the studied techniques could be sorted in ascending order of EMB usage and descending order of LUT usage as follows: LUT-ISE, FSMIM-S, FSMIM-T, and CONV-ROM. The possibility of exploiting all kinds of resources available in FPGAs allows to fit the design into a smaller (and cheaper) device. FSMIM-based architectures allow to find an adequate tradeoff between LUT and EMB usage. In fact, they obtain huge reductions in the LUT utilization by using a reasonable number of EMBs. The proposed FSMIM-S architecture extends the number of available alternatives for satisfying the area constraints. FSMIM-S is the best design option when the number of unused EMBs is limited. Otherwise, FSMIM-T is a better option if the speed is critical. Although CONV-ROMs

with small ROM depth are faster on average than FSMIM implementations, this trend is reversed when the ROM depth increases. We think that FSMIM-Gen is a useful tool for integrating the use of EMBs in the conventional design flow.

In future work, we plan to enhance the performance of the FSMIM technique. Recently, we have modeled the ISS optimization problem as a new generalization of the classical hitting set problem, and proposed an integer linear programming formulation [16]. We plan to integrate this formulation into FSMIM-Gen. Moreover, we will study the influence of the state and group encoding on the performance of the ISB and GE of FSMIM-S-based implementations. We plan to integrate an algorithm for finding efficient encoding into FSMIM-Gen.

ACKNOWLEDGMENT

The authors would like to thank Prof. L. Józwiak for providing us a benchmark set generated by the BenGen tool.

REFERENCES

[1] G. Borowik, T. Luba, and B. J. Falkowski, "Logic synthesis method for pattern matching circuits implementation in FPGA with embedded memories," in *Proc. 12th Int. Symp. Design Diagn. Electron. Circuits Syst. (DDECS),* Liberec, Czech Republic, 2009, pp. 230–233.

[2] B. Le Gal, A. Ribon, L. Bossuet, and D. Dallet, "Reducing and smoothing power consumption of ROM-based controller implementations," in *Proc. 23rd ACM Symp. Integr. Circuits Syst. Design,* New York, NY, USA, 2010, pp. 8–13.

[3] J. Cong and K. Yan, "Synthesis for FPGAs with embedded memory blocks," in *Proc. ACM/SIGDA 8th Int. Symp. Field Program. Gate Arrays (FPGA),* New York, NY, USA, 2000, pp. 75–82.

[4] A. Tiwari and K. A. Tomko, "Saving power by mapping finite-state machines into embedded memory blocks in FPGAs," in *Proc. Design Autom. Test Europe Conf. Exhibit.*, vol. 2. Paris, France, 2004, pp. 916–921.

[5] M. Rawski, H. Selvaraj, and T. Luba, "An application of functional decomposition in ROM-based FSM implementation in FPGA devices," in *Proc. Euromicro Symp. Digit. Syst. Design*, Belek-Antalya, Turkey, 2003, pp. 104–110.

[6] I. Garcia-Vargas, R. Senhadji-Navarro, G. Jimenez-Moreno, A. Civit-Balcells, and P. Guerra-Gutierrez, "ROM-based finite state machine implementation in low cost FPGAs," in *Proc. IEEE Int. Symp. Ind. Electron. (ISIE)*, Vigo, Spain, 2007, pp. 2342–2347.

[7] H. Selvaraj, M. Rawski, and T. Luba, "FSM implementation in embedded memory blocks of programmable logic devices using functional decomposition," in *Proc. IEEE Int. Conf. Inf. Technol. Cod. Comput*., Las Vegas, NV, USA, 2002, pp. 355–360.

[8] R. Senhadji-Navarro, I. Garcia-Vargas, G. Jimenez-Moreno, and A. Civit-Ballcels, "ROM-based FSM implementation using input multiplexing in FPGA devices," *Electron. Lett.*, vol. 40, no. 20, pp. 1249–1251, 2004.

[9] M. W. Weiss, S. C. Seth, S. K. Mehta, and K. L. Einspahr, "Design verification and functional testing of finite state machines," in *Proc. 14th Int. Conf. VLSI Design*, Bangalore, India, 2001, pp. 189–195.

[10] *Using Dedicated Multiplexers in Spartan-3 Generation FPGAs, Application Note: Spartan-3 FPGA Series*, Xilinx Inc., San Jose, CA, USA, 2005.

[11] I. Garcia-Vargas. (Mar. 2013). *Home Page—Prof. Ignacio Garcia-Vargas*. [Online]. Available: http://personal.us.es/iggv/en/material.html, accessed Mar. 3, 2015.

[12] R. Senhadji-Navarro, I. Garcia-Vargas, and J. L. Guisado, "Performance evaluation of RAM-based implementation of finite state machines in FPGAs," in *Proc. 19th IEEE Int. Conf. Electron. Circuits Syst. (ICECS)*, Seville, Spain, 2012, pp. 225–228.

[13] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer, 2004.

[14] K. McElvain. (1993). *IWLS 93 Benchmark Set: Version 4.0, Distributed as a Part of the IWLS 93 Benchmark Set*. [Online]. Available: http://ddd.fit.cvut.cz/prj/Benchmarks

[15] L. Jozwiak, D. Gawlowski, and A. Slusarczyk, "An effective solution of benchmarking problem: FSM benchmark generator and its application to analysis of state assignment methods," in *Proc. Euromicro Symp. Digit. Syst. Design (DSD)*, Rennes, France, 2004, pp. 160–167.

[16] I. Garcia-Vargas and R. Senhadji-Navarro, "The minimum maximal k-partial-matching problem," *Optim. Lett.*, vol. 7, no. 8, pp. 1959–1968, 2012.