Design Automation for Embedded Systems (2022) 26:83–103
https://doi.org/10.1007/s10617-022-09259-z

Optimization based on the minimum maximal *k*-partial-matching problem of finite states machines with input multiplexing

Ignacio Garcia-Vargas[1] · Raouf Senhadji-Navarro[1]

Received: 12 July 2021 / Accepted: 29 April 2022 / Published online: 2 June 2022

Abstract

Finite State Machines with Input Multiplexing (FSMIMs) were proposed in previous work as a technique for efficient mapping Finite State Machines (FSMs) into ROM memory. In this paper, we present new contributions to the optimization process involved in the implementation of FSMIMs in Field Programmable Gate Array (FPGA) devices. This process consists of two stages: (1) the simplification of the bank of input selectors of the FSMIM, and (2) the reduction of the depth of the ROM. This has a significant impact both on the number of used Look-Up Tables (LUTs) and on the number of the Embedded Memory Blocks (EMBs) required by the ROM. For the first stage, we present two approaches to optimize FSMIM implementations based on the Minimum Maximal *k*-Partial Matching (MMKPM) problem: one of them applies the greedy algorithm for the MMKPM problem, and the other based on a new multiobjetive variant of the MMKPM and its corresponding Integer Linear Programing formulation. We also propose a modification of the second stage, in which the characteristics of EMBs are taken into account to improve implementation results. The new optimization process significantly reduces the number of used FPGA resources with respect to the previous one. In addition, the proposed approaches achieve an adequate trade-off between the usage of EMBs and LUTs with respect to conventional FSM implementations based on ROM and to those based on LUT.

Keywords Finite state machine · Finite state machine with Input multiplexing · FPGA · ROM · Integer linear programming · Minimum maximal *k*-partial-matching

Ignacio Garcia-Vargas and Raouf Senhadji-Navarro have contributed equally to this work.

✉ Ignacio Garcia-Vargas
iggv@us.es

✉ Raouf Senhadji-Navarro
raouf@us.es

1 Department of Computer Architecture and Technology, University of Seville, E.T.S. Ingeniería Informática, Avda. Reina Mercedes s/n, 41012 Seville, Spain

1 Introduction

In digital design, the implementation of Finite State Machines (FSMs) has received attention from researchers for decades [1–8]. Optimizing FSM implementations in terms of area, speed or power consumption is essential to meet the design constraints demanded by applications. The inclusion of Embedded Memory Blocks (EMBs) in Field Programmable Gate Array (FPGA) devices has sparked a renewed interest in FSM implementations in the last years. Mapping the transition and output functions of a FSM into a ROM memory (the corresponding implementations will be referred to as *conventional ROM-based FSM implementations*) has become an interesting and useful alternative to conventional synthesis methods based on Look-Up Tables (LUTs). The advantages reported in the literature can be summarized as follows. First, exploiting unused EMBs to implement FSMs frees LUTs, which can be used for other purposes [9]. Second, speed improvements can be obtained if the function mapped on the ROM requires a considerable number of logic levels when mapped on LUTs [10, 11]. Third, a significant reduction in power consumption can be obtained by mapping FSMs to EMBs and disabling them during idle states [12].

In recent years, different techniques to reduce the number of required EMBs by using a small number of LUTs have been proposed [4, 9, 11, 13–15]. The EMB-based FSM implementations generated by these techniques make a more efficient use of the available FPGA resources, allowing an adequate trade-off between EMB and LUT usage. Most of these techniques use functional decomposition methods to decompose the transition function of the FSM into two subfunctions: one subfunction is implemented using a combinational element (called address modifier) whereas the other is implemented using a smaller memory component [4, 9, 15]. Other techniques apply structural decomposition methods to divide the logic block that represents the FSM into two or more subblocks [11, 16]. Then, a subblock with fewer inputs than the original logic block is implemented using EMBs; the remaining are implemented using combinational elements.

In the context of EMB-based FSM implementations, FSM with Input Multiplexing (FSMIM) is a technique proposed in [13] to reduce the depth of the memory in which the FSM transitions are mapped. This technique included an optimization process and an architecture called FSMIM with transition-based input selection (FSMIM-T), which uses a multiplexer bank as address modifier. FSMIMs take advantages of the fact that state transitions usually involve many don’t care inputs. The experiments reported in [10, 14] show area and speed improvements with respect to conventional approaches in Xilinx FPGAs. That work was extended in [14] by including a new architecture, called FSMIM with state-based input selection (FSMIM-S), which allows further reductions in the number of EMBs at the expense of reducing the speed and increasing the number of LUTs. Both architectures are based on a combinational circuit, called *Input Selector Bank* (ISB), which selects a different subset of FSM inputs for each state, allowing to reduce the depth of the memory. The optimization process described in [14] consisted of two stages. In the first stage, the ISB is simplified, allowing to reduce the number of LUTs and eventually the delay imposed by it. In the second stage, the depth of the memory is reduced beyond what the ISB would attain by itself. The FSMIM model has been recently used in research work by other authors [17, 18].

In [19], we defined the Minimum Maximal k-Partial-Matching (MMKPM) problem, which allows to model the problem involved in the simplification of the ISB. We probed that MMPKM problem is NP. We also proposed a greedy algorithm and an Integer Linear Programming (ILP) formulation for this problem. A performance study for these approaches using random bipartite graphs were presented; however, they were not applied to the generation of FSMIM implementations, and therefore no implementation detail was considered.

By contrast, in the development of this work, we have applied the MMKPM problem to the optimization of FSMIM implementations in FPGAs for the first time. This has allowed us to study the effect of the MMKPM on the whole implementation process (e.g., the influence of the optimization in the number of LUTs used by the ISB implementation or in the quality of the results of the subsequent stage). On the basis of the analysis of these preliminary results, we have proposed a new multiobjective variant of the MMKPM problem and its corresponding ILP formulation in order to obtain better implementation results.

The main contributions of this paper can be summarized as follows:

- A new multiobjective variant of the MMKPM problem and its corresponding ILP formulation are proposed.
- The greedy algorithm proposed in [19] and the new ILP formulation are applied for the first time to the generation of FSMIM implementations in FPGAs.
- We propose a modification of the second stage, in which, unlike the previous approach, the characteristics of EMBs are taken into account to improve the implementation results.
- We present a comprehensive experimental study in which the new techniques and those proposed in [14] are compared.
- Experimental results obtained by the new techniques are compared with conventional FSM implementations based on ROM and with those based on LUT.
- For the first time, we have used an Intel FPGA as target device (previous works used Xilinx FPGAs [10, 13, 14]). This allows us to validate the FSMIM technique in a device with a different architecture.

The rest of the paper is organized as follows. Section 2 presents a background of the conventional ROM-based implementations, the FSMIM architectures and the optimization process involved in the generation of FSMIM implementations. In Sect. 3, the new optimization process is presented. In Sect. 4, experimental results are discussed. Finally, concluding remarks and future work are presented in Sect. 5.

2 Background

2.1 Conventional ROM-based implementation

Mealy and Moore machines can be mapped into memory using the architecture shown in Fig. 1a (a specific architecture can be used for Moore machines, but here we consider the most general architecture) [11]. As the EMBs available in current FPGAs are synchronous, we assume Mealy machines with synchronous outputs. The input signals and present state encoding bits form the address of the ROM, which stores the FSM outputs and the next state of each FSM transition. The next state is fed back to the address signal as the present state [20]. The number of bits of the ROM is

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

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 exponential increase of the ROM size [see (1)] is a critical problem not only for area consumption but also for speed, which is degraded for the following two reasons: (1) the significant routing overhead when many

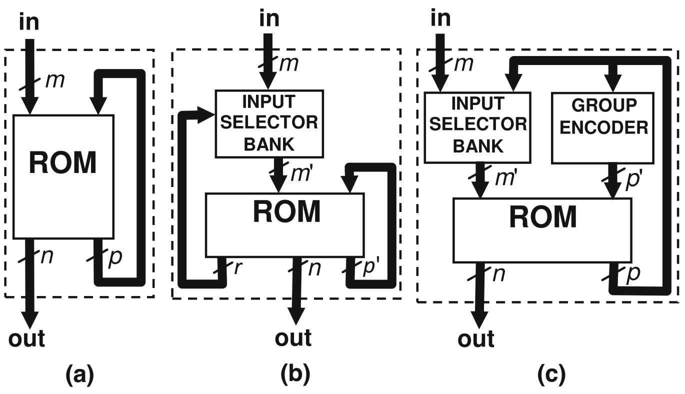


Fig. 1 ROM-based FSM architectures: **a** Conventional, **b** FSMIM-T and **c** FSMIM-S

EMBs are required, and (2) the use of LUTs to join EMBs when the memory depth exceeds the maximum depth of the EMBs [20].

2.2 Finite state machine with input multiplexing

The main purpose of the FSMIM approach is to reduce the number of used EMBs with respect to conventional ROM-based FSM implementations by using a small number of LUTs. For that, the technique tries to reduce the depth of the ROM memory by decreasing both m and $|S|$ in (1). In addition to decreasing the memory requirement, the reduction of the depth can also decrease delay imposed by the memory [20]. However, the speed is not considered as an objective of the proposed technique but only a potential benefit. For each state, the ROM can be addressed using only the subset of FSM inputs that are relevant to determine the transitions of the state (called *effective inputs*). This allows to reduce the ROM depth if the maximum number of effective inputs per state (denoted by m') is less than the number of inputs (m). For this purpose, the ISB selects the effective inputs of each state from FSM inputs (so, an ISB has m inputs and m' outputs). For those states that have less than m' effective inputs, only part of the inputs selected by the ISB are effective; the remaining are don't care inputs (called *Don't Care Selected Inputs* [DCSIs]). The ROM depth can be further reduced by using DCSIs to form groups of states that can be encoded with the same code. Thus, instead of using the code of the present state to address the ROM, the code of the group the present state belongs to (i.e., the code of the present group) is used. If all states are grouped into N groups (with $N < |S|$), the depth can be reduced to $2^{m'}N$ words. This allows to reduce the ROM depth even in FSMs with some states sensitive to all inputs (i.e., with $m' = m$). FSMIM architectures are shown in Fig. 1b and c. The ROM address is composed of m' effective inputs and the p' encoding bits of the present group, where $p' = \lceil \log_2 N \rceil$.

The aim of the FSMIM-T architecture (see Fig. 1b) is to reduce the memory requirement at the same time that the ISB keeps as simple as possible. This allows efficient implementations in terms of LUTs without degrading the speed. In this case, the ISB is implemented as a bank of multiplexers, which is greatly benefited from embedded multiplexers available in current FPGAs [21, 22]. Another advantage of the bank of multiplexers is that it can be optimized in a high-level way because the complexity of a multiplexer is only determined by its number of inputs (i.e., it is not necessary to use the synthesis tool to evaluate its complexity as in the case of general logic functions). The multiplexer bank is composed of m' multiplexers called *input selectors*. For each state, each input selector selects one different effective input of the state from a different subset of FSM inputs. Therefore, each input selector is controlled by a different set of control bits (called *selection bits*), which are stored in the ROM. This allows to simplify the multiplexers at the expense of increasing the ROM width with respect to the conventional ROM-based implementation. For each FSM transition, the ROM stores the FSM outputs, the encoding bits of the next group, and the selection bits. The number of bits of the ROM of a FSMIM-T is

$$2^{m'} N(n + p' + r) \tag{2}$$

where r is the number of selection bits.

The aim of the FSMIM-S architecture (see Fig. 1c) is to implement FSMIMs using the minimum number of EMBs [14]. Each word of the ROM stores the next state and FSM outputs, as in the conventional ROM-based architecture. Therefore, the number of bits of the ROM of a FSMIM-S is

$$2^{m'} N(n + p) \tag{3}$$

In this way, the depth of the ROM can be reduced without increasing its width (as occur in the FSMIM-T architecture). However, the number of used LUTs with respect to the FSMIM-T architecture increases for two reasons. First, input selectors are not multiplexers controlled by selection bits but logic functions that select the FSM inputs from the present state encoding bits; so, the ISB implementation is more complex than the corresponding one in the FSMIM-T architecture, which uses the minimum possible number of selection bits. Second, a new combinational component, called *group encoder* (GE), is required to map the present state (coded by p bits) to the present group (coded by p' bits). FSMIM-S implementations are usually slower than FSMIM-T implementations due to the use of the GE and a more complex ISB [14].

As a conclusion, the FSMIM-T architecture is suitable for designs in which it is critical to reduce the number of LUTs whereas FSMIM-S is more appropriate when the number of EMBs is limited. When the speed must be taken into account, FSMIM-T implementations offer advantages over the FSMIM-S ones.

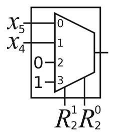
2.3 Generation of FSMIMs from FSMs

The optimization process presented in [14] is used to transform a FSM into a FSMIM whose implementation is efficient in terms of EMB and LUT usage, which can also have a positive impact on speed. This process is independent of the used FSMIM architecture. The optimization problems involved in the process can be described in terms of the *Input Selection Matrix* (ISM) [14], in which rows contain the effective inputs of each state, and columns contain the FSM inputs connected to each input selector. Let $S = \{s_1, s_2, \ldots, s_q\}$ and $X = \{x_1, x_2, \ldots, x_m\}$ be the set of states and the set of FSM 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 (a FSM input, a constant value or a don't care) selected for the state s_i by the j-th input selector. For clarity, the ISM includes an additional column for the state corresponding to each row. Fig. 2-a shows two examples of ISM obtained from a same FSM.

The optimization process consists of the two following stages: (1) *Input Selectors Simplification* (ISS), which simplifies the ISB, and (2) *State Grouping* (SG), which reduces the ROM depth by merging states. First, the ISS stage is applied to the ISM created from the

$$A = \left[\begin{array}{ccc|c} x_6 & x_5 & x_1 & s_0 \\ x_5 & x_1 & - & s_1 \\ x_1 & - & - & s_2 \\ - & - & x_2 & s_3 \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{array}\right] \qquad A^{ISS} = \left[\begin{array}{ccc|c} x_1 & x_5 & x_6 & s_0 \\ x_1 & x_5 & - & s_1 \\ x_1 & - & - & s_2 \\ x_2 & - & - & s_3 \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{array}\right]$$

(a)



(b)

$$A_1^{SG} = \left[\begin{array}{ccc|c} x_1 & x_5 & x_6 & s_0 \\ x_1 & x_5 & - & s_1 \\ x_1 & 0 & - & g_{23} \\ x_2 & 1 & - & g_{23} \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{array}\right] \quad A_2^{SG} = \left[\begin{array}{ccc|c} x_1 & x_5 & x_6 & s_0 \\ x_1 & x_5 & 0 & g_{123} \\ x_1 & 0 & 1 & g_{123} \\ x_2 & 1 & 1 & g_{123} \\ x_2 & x_4 & - & s_4 \\ x_3 & x_4 & - & s_5 \end{array}\right] \quad A_3^{SG} = \left[\begin{array}{ccc|c} x_1 & x_5 & x_6 & g_0 \equiv s_0 \\ x_1 & x_5 & 0 & g_{123} \\ x_1 & 0 & 1 & g_{123} \\ x_2 & 1 & 1 & g_{123} \\ x_2 & x_4 & 0 & g_{45} \\ x_3 & x_4 & 1 & g_{45} \end{array}\right]$$

(c)

Fig. 2 Example of FSMIM generation: **a** Input Selector Simplification, **b** 2nd input selector for the FSMIM-T architecture (R_2^1 and R_2^0 are the selection bits), and **c** State Grouping

given FSM. Then, the SG stage is applied to the resultant ISM, generating the ISM that determines the final FSMIM implementation.

2.3.1 Input selectors simplification

The complexity of the ISB is measured as the sum of the number of inputs of each input selector, that is, the sum of the number of different inputs in each ISM column (we will refer to this measure as *selection cost*). The main goal of the ISS stage is to reduce the selection cost by minimizing the number of inputs of each input selector. In general, input selectors with a smaller number of inputs require a smaller number of LUTs. Since the ISB is usually in the critical path (this is always true for FSMIM-T implementations), the speed of FSMIM implementations can increase if the ISS stage reduces the delay of the ISB.

The complexity of a multiplexor only depends on the number of inputs. Therefore, in the FSMIM-T architecture, there exists a direct relation between the selection cost and the complexity of the ISB. So, the minimization the selection cost has a strong influence on the reduction of the number of LUTs required by the ISB. However, in the FSMIM-S architecture, the complexity of the logic function defined by the ISB depends not only on the number of inputs but also on the complexity of such logic function. Therefore, the minimization the selection cost has less influence on the reduction of the number of used LUTs.

Given an ISM, the ISS stage permutes the elements of each row to reduce the number of different inputs in each column (i.e., the number of inputs of each input selector of the ISB). We refer to each feasible solution of the ISS stage as a *permutation* of the ISM. For example, A^{ISS} is a permutation of A (see Fig. 2a) that reduces the number of inputs of the input selectors from 5 to 3, from 3 to 2, and from 2 to 1.

In [14], we proposed an algorithm to simplify the ISB. This algorithm iteratively solves instances of a variant of the classical Knapsack problem (which will be referred to as Knapsack with Conflicts [KC] problem). The KC problem extends the classical Knapsack problem by introducing disjunctive constraints for pairs of items which are not allowed to be packed together into the knapsack [23, 24]. The algorithm to simplify the ISB, which will be referred to as KC-based ISS algorithm, processes the columns of the ISM iteratively. With the goal of reducing the number of different elements in each column, a different instance of the KC problem is solved to determine what element of each row (either a FSM input or a DCSI) is located in the column. The remaining elements will form the KC instance corresponding to the next column. Therefore, the algorithm processes as many instances of KC as ISM columns, allowing to reduce the selection cost. The classical dynamic-programming-based method for the Knapsack problem [25] has been modified to solve each instance of KC. Obviously, independently of the optimality of the method used to solve each instance of KC, the algorithm does not guarantee an optimal solution in terms of the selection cost because the selection of elements is done in a column without considering how it affects to the rest of columns.

2.3.2 State grouping

The main goal of the SG stage is to reduce the ROM depth by forming groups of states that can be encoded with the same code (we will refer to them simply as *groups*). A set of states can form a group if exists an assignment of constant values to the DCSIs that allows to allocate their transitions in different ROM words. For example, the states s_2 and s_3 of A^{ISS} (see Fig. 2a) can be identified by the symbol g_{23} in matrix A_1^{SG} (see Fig. 2c) because the 2nd input selector allows to address the transitions of either s_2 (if 0 is selected) or s_3 (if 1 is selected). Let us say that these states are merged into the group g_{23}. So, in FSM transitions, the next states s_2 and s_3 are replaced by g_{23}, with $(x_1, 0, -)$ as input selection for s_2, and $(x_2, 1, -)$ for s_3. Fig. 2b shows the multiplexer corresponding to the 2nd input selector for the FSMIM-T architecture.

Given an ISM, the algorithm presented in [14] (we will refer to it as *old SG algorithm*) processes all ISM columns iteratively in a certain order. Initially, each single state forms a different group. For each column, the DCSIs are set to 0 or 1 to merge pairs of the groups obtained after processing the previous columns. Fig. 2c shows an example of the SG procedure applied to A^{ISS} (see Fig. 2a). First, s_2 and s_3 are merged into g_{23} by setting the DCSIs of the second column (see A_1^{SG}). Then, g_{23} and s_1 are merged into g_{123} by setting, in the third column, the DCSI of s_1 to 0 and all DCSIs of g_{23} to 1 (see A_2^{SG}). Note that this is possible because g_{23} and s_1 have a DCSI in the same column for all their rows. Finally, s_4 and s_5 are merged into g_{45} (see A_3^{SG}). SG reduces the number of groups from 6 to 3 ($G = \{g_0, g_{123}, g_{45}\}$); so, the number of encoding bits is reduced from 3 to 2. As the example shows, a low dispersion of the amount of DCSIs in the ISM columns (i.e., a lot of DCSIs in few columns) favours the effectiveness of the algorithm (we will refer to this concept as *DCSI dispersion*). For example, when SG is applied to A (see Fig. 2a), which has higher DCSI dispersion than A^{ISS}, only the pairs of states (s_2, s_3) and (s_4, s_5) can be merged.

3 Proposed optimization process to generate FSMIMs from FSMs

In this paper, we present two new approaches for the ISS stage, which are based on the MMKPM problem [19]. In one of these approaches, the greedy algorithm presented in [19] has been applied for the first time to generate FSMIMs; in the other one, a new multiobjective variant of the MMKPM problem and the corresponding ILP formulation have been proposed. Unlike the greedy algorithm, the ILP formulation is able to obtain optimal solutions. In addition, we present a modification of the SG stage to achieve further improvements.

3.1 Proposed input selectors simplification

In order to improve the results obtained by the ISS stage, we proposed in [19] the MMKPM problem, along with an ILP formulation and a greedy algorithm. This problem, which is NP-complete, allows to model the optimization problem involved in the ISS stage. The objective of the KC-based ISS algorithm and the MMPKM problem is to minimize the complexity of the ISB by minimizing the selection cost. Unlike the solutions of the KC-based ISS algorithm, the optimal solutions of the MMKPM problem do correspond to minimum selection cost. Anyway, both can be not optimum in terms of the number of selection bits. In FSMIM-T implementations, this affects both the ISB (which is controlled by the selection bits) and the ROM (which stores the selection bits). In Fig. 2-a, the selection cost and the number of selection bits for A are $5 + 3 + 2 = 10$ and $\lceil \log_2 5 \rceil + \lceil \log_2 3 \rceil + \lceil \log_2 2 \rceil = 6$, respectively. If the input x_2 of the 4th row is moved from column 3 to 2, the selection cost remains at 10, but the number of selection bits decreases from 6 to 5. This paper presents a multiobjective variant of the MMKPM problem, which, in addition to minimizing the selection cost, allow to further reduce the ROM size by minimizing both the number of selection bits (which reduces the ROM width of the FSMIM-T architecture) and the DCSI dispersion (which improves the results of the SG stage for both FSIMIM architectures [see Sect. 2.3.2]). This variant is called *MMKPM with minimum selection cost, number of selection bits and DCSI dispersion* (MMKPM-SSD).

For simplicity, we restrict the description of the MMKPM-SSD problem to the context of the ISM ([19] presents a general definition of the MMKPM problem). Let $A = (a_{ij}) \in \mathcal{M}_{q\times k}$ be an ISM where $a_{ij} \in X \cup \{-\}$ (A does not contain any constant because ISS is applied before SG). Given A, a bipartite graph $G = (R \cup X, E)$ is constructed, where $R = \{1, \ldots, q\}$ represents the rows of A. We refer to the vertices of R and X as rows and inputs, respectively. The edges of $E \subseteq R \times X$ link the rows of A with their inputs. The degree of a vertex $s \in R \cup X$ is denoted by $\deg_G(s)$, and the set of edges in E that are incident to s, by $E(s)$. G cannot represent permutations of A because it does not contain information about columns. Let us define a *partial-matching* in G as a set of edges without common rows. Each column of A can be represented by a different partial-matching, which selects at most one input from each row (the absence of selected inputs in a row represents a DCSI). Let us define a *maximal k-partial-matching* in G as a collection $P = \{P_1, \ldots, P_k\}$ of k partial-matchings that define a partition of E. A maximal k-partial-matching determines a permutation of A in which the inputs of the i-th column are given by P_i. The selection cost can be calculated from P as $\sum_{i=1}^{k} |X(P_i)|$, where $X(P_i)$ denotes the set of inputs that are incident to any edge of P_i.

The objective of the MMKPM problem is to find a maximal k-partial-matching $P = \{P_1, \ldots, P_k\}$ in G such that $\sum_{i=1}^{k} |X(P_i)|$ is minimum (i.e., a permutation of A with a minimum selection cost). The MMKPM-SSD problem adds two new objectives of lower priority: the minimization of the number of selection bits and the minimization of the DCSI dispersion. The number of selection bits can be calculated from P as $\sum_{i=1}^{k} \lceil \log_2 |X(P_i)| \rceil$. Regarding the DCSI dispersion, let us define the *weighted cardinality* of P as $\sum_{i=1}^{k} i|P_i|$. Given P, if a DCSI of P_i is swapped with an input of P_j of the same row (we say that the DCSI is moved from P_i to P_j), then $|P_i|$ increases by one and $|P_j|$ decreases by one, and so the weighted cardinality decreases by $j - i$ when $i < j$. Hence, minimizing the weighted cardinality of P minimizes the DCSI dispersion in the resultant permutation of A because DCSIs are moved to partial-matchings with higher indices. For example, in Fig. 2a, the weighted cardinality of the maximal k-partial-matchings associated to A and A^{ISS} are $1 \cdot 5 + 2 \cdot 4 + 3 \cdot 2 = 19$ and $1 \cdot 6 + 2 \cdot 4 + 3 \cdot 1 = 17$, respectively.

Given an ISM $A = (a_{ij}) \in \mathcal{M}_{q\times k}$ and the corresponding bipartite graph $G = (R \cup X, E)$, the MMKPM-SSD problem can be formulated as the problem of finding a maximal k-partial-matching $P = \{P_1, \ldots, P_k\}$ in G such that the tuple

$$\left(\sum_{i=1}^{k} |X(P_i)|,\ \sum_{i=1}^{k} \lceil \log_2 |X(P_i)| \rceil,\ \sum_{i=1}^{k} i|P_i|\right) \tag{4}$$

is minimum in lexicographical order. The following is the proposed 0/1 ILP formulation for the MMKPM-SSD problem, which is an extension of that presented in [19]. Let the variables $y_{x,i}, z_{e,i} \in \{0, 1\}$ be defined as

$$y_{x,i} = \begin{cases} 1 & \text{if } x \in X(P_i) \\ 0 & \text{otherwise} \end{cases} \qquad \forall x \in X,\ i = 1, \ldots, k, \tag{5}$$

$$z_{e,i} = \begin{cases} 1 & \text{if } e \in P_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall e \in E,\ i = 1, \ldots, k, \tag{6}$$

and let $d_{j,i} \in \{0, 1\}$ be defined as the bit of weight 2^j corresponding to the binary representation of the value $2^{\lceil \log_2 |X(P_i)| \rceil} - 1$, which is a string of 1s whose length is equal to $\lceil \log_2 |X(P_i)| \rceil$. These variables allow to express (4) as the tuple

$$\left(\sum_{i=1}^{k} \sum_{x \in X} y_{x,i},\ \sum_{i=1}^{k} \sum_{j=0}^{D-1} d_{j,i},\ \sum_{i=1}^{k} i \sum_{e \in E} z_{e,i}\right), \tag{7}$$

where

$$D \geq \max\{\lceil \log_2 |X(P_i)| \rceil : i = 1, \ldots, k\}. \tag{8}$$

Then, the ILP formulation is given by

$$\text{min. } QDk \sum_{i=1}^{k} \sum_{x \in X} y_{x,i} + Q \sum_{i=1}^{k} \sum_{j=0}^{D-1} d_{j,i} + \sum_{i=1}^{k} i \sum_{e \in E} z_{e,i}, \tag{9}$$

$$\text{where } Q = \frac{1}{2}(k^2 + k)\,|R|, \tag{10}$$

$$D = \left\lceil \log_2 \min\{|R|, |X|\} \right\rceil, \tag{11}$$

subject to

$$\sum_{e \in E(r)} z_{e,i} \leq 1 \qquad \forall r \in R,\ i = 1, \ldots, k, \tag{12}$$

$$\sum_{i=1}^{k} z_{e,i} \leq 1 \qquad \forall e \in E, \tag{13}$$

$$\sum_{i=1}^{k} \sum_{e \in E(x)} z_{e,i} = \deg_G(r) \qquad \forall r \in R, \tag{14}$$

$$z_{e,i} \leq y_{x,i} \qquad \forall e \equiv (r, x) \in E,\ i = 1, \ldots, k, \tag{15}$$

$$y_{x,i} \leq \sum_{e \in E(x)} z_{e,i} \qquad \forall x \in X,\ i = 1, \ldots, k, \tag{16}$$

$$2^j d_{j,i} \leq \sum_{x \in X} y_{x,i} - 1 \qquad j = 0, \ldots, D-1,\ i = 1, \ldots, k, \tag{17}$$

$$\sum_{x \in X} y_{x,i} - 1 \leq \sum_{j=0}^{D-1} 2^j d_{j,i} \qquad i = 1, \ldots, k, \tag{18}$$

$$d_{j,i} \geq d_{j+1,i} \qquad j = 0, \ldots, D-2,\ i = 1, \ldots, k. \tag{19}$$

In (9), Q and D [see (10) and (11)] are used to sort lexicographically the solutions. Q is an upper bound of the maximum weighted cardinality because $|R| \geq |P_i| = \sum_{e \in E} z_{e,i}$. D satisfies (8) because $|X(P_i)| \leq \min\{|R|, |X|\}$ (by definition of a partial-matching), and therefore kD is an upper bound of the number of selection bits.

Constraint (12) ensures that each P_i has at most one edge incident to each row, i.e., that P_i is a partial-matching. Constraint (13) ensures that one edge cannot belong to different partial-matchings, i.e., that the k partial-matchings are disjoint. In addition, each edge belongs to a partial-matching due to (14); therefore, the feasible solutions are maximal k-partial-matchings. Constraints (15) and (16) ensure the coherence of $z_{e,i}$ and $y_{x,i}$. Constraint (15) guarantees that an edge $e \equiv (r, x)$ belongs to P_i only if the input $x \in X(P_i)$ (i.e., $z_{e,i}$ is equal to 1 only if $y_{x,i} = 1$), and (16) imposes that an input x belongs to $X(P_i)$ only if there exists at least one edge $e \equiv (r, x) \in P_i$. Regarding the number of selection bits, (17) ensures that $d_{j,i} = 0$ for all j such that $2^j > |X(P_i)| - 1$; (18), that $d_{k,i} = 1$ for the greatest k such that $2^k \leq |X(P_i)| - 1$; and (19), that $d_{j,i} = 1$ for all $j < k$. As a result, given i, the number of $d_{j,i}$ variables whose value is 1 is equal to the number of bits required for the binary representation of $|X(P_i)| - 1$ (i.e., $\lceil \log_2 |X(P_i)| \rceil$).

The ILP formulation of the MMKPM-SSD problem differs from that of the MMKPM problem in the objective [see (9)] and in the new constraints (17), (18), and (19).

3.2 Proposed state grouping

With the goal of reducing as much as possible the depth of the ROM of FSMIM implementations, the old SG algorithm processes the ISM until no more groups can be merged (i.e., until the depth of the ROM cannot be further reduced). However, the depth of the implemented ROM is usually greater than that obtained by the algorithm because of the discrete size of EMBs. For example, the FSMIM of Fig. 2a has $|S| = 6$ states, $m' = 3$ effective inputs and $p' = 3$ state encoding bits. The FSMIM-T corresponding to the ISM A^{ISS} requires $r = \lceil \log_2 3 \rceil + \lceil \log_2 2 \rceil + \lceil \log_2 1 \rceil = 3$ selection bits. Supposing that the number of outputs (n) is 8, the FSMIM-T requires a ROM of $2^{m'} \cdot |S| = 48$ words of $n + p' + r = 14$ bits, that is, a ROM of 672 bits [this value can be obtained with (2)]. If we suppose EMBs of 512 bits with 32 words of 16 bits, then the FSMIM-T implementation requires two EMBs (this is only a toy example to illustrate the concept mentioned above, since EMBs are greater in real devices). After the old SG algorithm reduces the number of groups from 6 to 3 (see A_3^{SG} in Fig. 2c), the depth of the required ROM is reduced from 48 to $2^{m'} \cdot 3 = 24$ words whereas the size of each word increase from 14 to 16 bits due to $r = \lceil \log_2 3 \rceil + \lceil \log_2 4 \rceil + \lceil \log_2 3 \rceil = 6$ and $p' = \lceil \log_2 3 \rceil = 2$. So, the FSMIM-T can be implemented in a unique EMB. However, the FSMIM-T corresponding to the ISM A_2^{SG} (Fig. 2-c), which has 4 groups instead of 3, can also be implemented in one EMB due to it requires a ROM of $2^{m'} \cdot 4 = 32$ words of 16 bits (since $r = \lceil \log_2 3 \rceil + \lceil \log_2 4 \rceil + \lceil \log_2 3 \rceil = 6$ and $p' = \lceil \log_2 4 \rceil = 2$). Therefore, in this example, the old SG algorithm merges more states than necessary because 4 groups are enough to minimize the number of required EMBs.

In the cases in which reaching the minimum number of groups is not necessary to minimize the number of required EMBs, the ISM obtained by the algorithm is more complex than necessary due to the constant values assigned to DCSIs, which has a negative impact on speed and area. In both FSMIM architectures, the number of required LUTs can increase because, in FSMIM-T, the number of selection bits grows, and, in FSMIM-S, the selection function corresponding to the ISB has fewer don't care outputs. This increase in the number of LUTs can cause a decrease of the maximum operating frequency. Regarding EMBs, the width of the ROM of the FSMIM-T architecture grows with the number of selection bits, which can increase the number of required EMBs. However, this effect does not occur in FSMIM-S implementations, in which the ROM width remains constant due to it only depends on the state encoding bits and the number of outputs. To sum up, merging more groups than necessary can increase the number of LUTs required by both FSMIM architectures as well as the number of EMBs required by FSMIM-T implementations (but not by FSMIM-S implementations). Therefore, from the point of view of the implementation results, the optimal number of groups is not the minimum value found by the old SG algorithm but the maximum number of groups that minimizes the number of EMBs required by the implementation.

This paper proposes an algorithm to find the optimal number of groups (called *EMB-granularity-based SG algorithm*), which is a modification of the old SG algorithm. After each pair of groups are merged, the obtained ISM is used to estimate the number of EMBs required by the corresponding FSMIM implementation, and it is considered as the candidate solution only if the number of EMBs decreases. After all columns have been processed, the last ISM chosen as candidate solution determines the best solution (i.e., the one with the optimal number of groups). To estimate the number of EMBs, the algorithm takes into account the available configurations for the depth and width of the EMBs of the target device [21, 22].

Compared with the old SG algorithm, the EMB-granularity-based SG algorithm allows to reduce the ISB complexity of both architectures and the ROM size of the FSMIM-T architecture; however, it cannot further reduce the ROM size of the FSMIM-S architecture.

4 Experimental results

Two different optimization strategies have been implemented, which differ in the technique applied in the ISS stage. One of them (called *ILP strategy*) applies the proposed ILP formulation to solve the MMKPM-SSD problem and the other (called *greedy strategy*) applies the greedy algorithm presented in [19] to solve the MMKPM problem. In the SG stage, both strategies apply the EMB-granularity-based SG algorithm. The main purpose of this section is to compare both strategies with the optimization process published in [14] (hereinafter called OLD-FSMIM), which applies the KC-based ISS algorithm in the ISS stage and the old SG algorithm in the SG stage. In addition, the FSMIM implementations obtained with the presented strategies are compared with conventional FSM implementations based on ROM (CONV-ROM) and with those based on LUT (CONV-LUT).

All designs have been synthesized and implemented in a Intel Max 10 device (10M50DAF484C6GES) using Quartus Prime software version 18.0. Therefore, the presented results include routing overhead. The maximum clock frequency and the resource utilization (EMBs and LUTs) of the implemented FSMs have been obtained using speed and area optimization, respectively. As the main goal of EMB-based FSM implementations is to save LUTs by using EMBs, their efficiency can be measured by calculating the number of saved LUTs with respect to CONV-LUT per each used EMB (this measure will be called *saved LUTs per EMB* [SLPE]).

The target device includes 182 EMBs of 9 Kbits, which can be split into two independent EMBs of 4.5 Kbits (residual 4.5 Kbits EMBs are computed as 0.5 EMBs). The experimental study uses the IWLS93 standard benchmark set [26] (composed by 43 FSMs) and a bechmark set generated by BenGen tool [27] (composed by 150 FSMs) (these FSMs will be referred to as *medium-sized test cases* because the majority have that size). We have discarded the cases in which CONV-ROM only requires half EMB (i.e., the minimum amount of memory that can be instantiated) and the cases in which the FSMIM technique cannot be applied (i.e., in which the number of effective inputs is equal to the number of inputs and, after the ISS stage, there is no ISM column with two or more DCSIs). The final number of used FSMs was 57. In addition, in order to evaluate the proposed technique with larger FSMs, we have generated 72 synthetic test cases (which will be referred to as *large-sized test cases*). The first and third quartiles for the ROM size in bits of the CONV-ROM implementations for medium-sized and large-sized test cases are, in this order, 1.7×10^4, 3.1×10^5, 3.0×10^6 and 2.7×10^7. Thus, the sweep range is wide and the samples are quite homogeneously distributed.

As MMKPM-SSD is an NP-complete problem, the computation time spent by the corresponding ILP formulation to find an optimal solution is expected to be high when the size of the problem instance grows. In 54% of the total cases (including both medium-sized and large-sized test cases), Gurobi reaches the time limit, and so the solution found is not optimal. However, as Sects. 4.1 and 4.2 show, even in these cases, the optimization process obtains better results using the ILP formulation than using the greedy algorithm for the MMKPM problem or the KC-based ISS algorithm. For the cases in which the time limit is reached and the solution found is not optimal, the average gap (which indicates how long is the obtained solution from the optimal solution) is 22%. Therefore, the optimization results could be improved by increasing the time limit. Regarding the other approaches to solve the ISS stage, the execution time for the greedy algorithm and the KC-based ISS algorithm are 0.03 s and 134 s, respectively. The average speed-up obtained by the greedy algorithm is 2,282. In addition to the fact that the greedy algorithm is the fastest, the optimization process obtains better results using it than using the KC-based ISS algorithm (see Sects. 4.1 and 4.2).

4.1 Comparison between the proposed strategies and OLD-FSMIM using medium-sized test cases

The results obtained using the medium-sized test cases are summarized in Table 1, which shows the EMB reduction (EMB red.), the LUT reduction (LUT red.), SLPE increment (SLPE inc.) and speed increment (Speed inc.) obtained by the proposed optimization strategies with respect to OLD-FSIMIM. The table shows the following statistical measures: mean (Mean), standard deviation (Std), minimum value (Min), quartiles (Q1, Q2 and Q3), maximum value (Max), and the percentage of cases in which the proposed technique obtains better results (hit rate [HR]) and worse results (miss rate [MR]) than OLD-FSMIM. One test case has been excluded from the statistical data because the FSMIM-T implementation generated by OLD-FSMIM requires more than 182 EMBs; however, the implementation was possible using the proposed strategies.

Regarding the EMB usage, as explained in Sect 3.2, the EMB-granularity-based SG algorithm cannot further reduce the ROM size of the FSMIM-S architecture compared with the old SG algorithm. In addition, in the FSMIM optimization process, the influence of the ISS stage on the ROM size of the FSMIM-S architecture is not very significant because the ISS

Table 1 Percentage improvement with respect to OLD-FSMIM using medium-sized test cases

	Arch.	Strat.	Mean	Std	Min	Q1	Q2	Q3	Max	HR	MR
EMB red.	FSMIM-S	ILP	0	0	0	0	0	0	0	0	0
		Greedy	0	0	0	0	0	0	0	0	0
	FSMIM-T	ILP	8	14	0	0	0	15	50	30	0
		Greedy	7	12	0	0	0	11	50	29	0
LUT red.	FSMIM-S	ILP	24	23	−17	6	24	40	70	81	14
		Greedy	22	24	−31	2	23	38	70	75	19
	FSMIM-T	ILP	24	22	−17	0	20	43	67	71	2
		Greedy	22	20	0	0	18	40	60	70	0
SLPE inc.	FSMIM-S	ILP	9	12	−9	1	3	15	39	81	14
		Greedy	8	12	−9	1	3	12	39	75	19
	FSMIM-T	ILP	15	25	0	0	3	20	107	71	2
		Greedy	11	20	0	0	1	16	107	70	0
Speed inc.	FSMIM-S	ILP	2	8	−12	−4	2	7	25	56	44
		Greedy	3	8	−14	−3	3	7	25	61	39
	FSMIM-T	ILP	4	6	−10	0	5	7	21	70	21
		Greedy	4	5	−5	0	3	7	20	70	21

stage cannot reduce the ROM width of this architecture (as occur in the FSMIM-T one): this stage can only improve the results of the subsequent SG stage by concentrating the DCSIs in the minimum number of ISM columns. Therefore, the effect of the proposed strategies in the EMB reduction of the FSMIM-S architecture is less likely to be observed in medium-sized FSMs. This is confirmed by the results (the average reduction is 0% for both strategies). However, for FSMIM-T implementations, the ILP strategy reduces the number of EMBs in 44% of the cases in which OLD-FSMIM uses more than 0.5 EMBs (i.e., in which a further reduction is possible); for these cases, which represents the 30% of the whole sample, the average reduction is 27%. The greedy strategy obtains similar results.

Regarding the LUT usage, the average reduction obtained for all architectures and strategies range from 22 to 24% with hit rates from 70 to 85%. The miss rates in FSMIM-T implementations are null or negligible; however, they reach 19% in FSMIM-S ones. The reason is that, unlike the multiplexer bank of the FSMIM-T architecture, the complexity of the ISB of the FSMIM-S one depends not only on the number of inputs but also on the complexity of the logic functions. Therefore, the minimum number of LUTs is not always reached with the minimum selection cost.

Regarding speed, the difference between the hit rate (70%) and the miss rate (21%) for FSMIM-T implementations points out that, in general, in addition to reducing the FPGA resources, the proposed strategies increase the speed (despite the poor average increment); however, this improvement is not so clear in the case of the FSMIM-S architecture because of the reasons mentioned above.

On average, the results of ILP strategies do not show significant improvements with respect to that of greedy strategies. In general, the reason could be that MMKPM problems related to medium-sized test cases are no greater enough. As it is difficult to directly compare both strategies using the results of Table 1, we have calculated the improvements in EMB and LUT usage of the ILP strategy with respect to the greedy one. In FSMIM-T implementations, the hit rate is 12% in EMB reduction and 21% in LUT reduction, with an average reduction for the success cases of 17% and 20%, respectively. In both comparisons, the miss rate is null or negligible. In FSMIM-S implementations, despite of the less influence of the ISS stage on the complexity of the ISB, the hit rate in LUT reduction is 22% whereas the miss rate is only 14%.

4.2 Comparison between the proposed strategies and OLD-FSMIM using large-sized test cases

The results obtained using the large-sized test cases are summarized in Table 2. These results show that the effectiveness of the proposed strategies grows with the size of the FSM. Both the area and speed results for FSMIM-T implementations are significantly improved when the strategies are applied to larger FSMs (compare HR and Mean of Tables 1 and 2). Regarding FSMIM-S implementations, the number of EMBs is reduced with respect to OLD-FSMIM (the hit rate achieved by the ILP strategy is 43% with an average reduction of 23% for these cases). Therefore, the proposed strategies obtain the best results considering the main goal of the FSMIM-S architecture (i.e., the implementation of FSMIMs with a minimum number of EMBs, being secondary the reduction of the number of LUTs as explained in Sect. 2.2). However, in general, the improvement in the EMB reduction obtained by the SG stage negatively affects the ISB complexity and degrades the results of the ISS stage. In FSMIM-S implementations, this degradation along with the less influence of the ISS stage on the complexity of the ISB reduces the improvement in LUT usage, which explains the diminishment of the average LUT reduction in large-sized FSMs. Despite this, in both architectures, the net effect is that the efficiency of the proposed strategies in the use of resources increases significantly with the size of the FSM, as the SLPE results indicate (compare the SLPE of Tables 1 and 2). As a conclusion, these results show that, compared to OLD-FSMIM, the effectiveness of the proposed strategies grows with the size of the FSM for both FSMIM architectures.

Regarding the comparison between greedy and ILP strategies, despite of the fact that the greedy one achieves significant improvements over OLD-FSMIM, the ILP strategy further improves these results (in a significant way for FSMIM-T). To highlight the differences between the proposed strategies, the improvements of the ILP strategy with respect to the greedy one are summarized in Table 3. The hit rate in EMB reduction is 76% in FSMIM-T implementations, and 32% in FSMIM-S ones (they are about 10x larger than the miss rates); in both cases, the average reduction for the success cases is close to 15%. These improvements are due to the lower DCSI dispersion obtained by the ILP formulation in the ISS stage, which allows the SG stage to further reduce the number of groups.

The LUT reductions obtained by the ILP strategy with respect to the greedy one are also significant for FSMIM-T (with an average reduction of 14% and a hit rate of 72%); however, it is negligible for FSMIM-S (although the hit rate is 60%). Despite this, in both cases, the SLPE results show that the ILP strategy is clearly more efficient than the greedy one in the use of resources. The hit rate in SLPE increment is 75% in FSMIM-T implementations; on average, ILP strategy saves 15% more LUTs per used EMB than the greedy strategy, and this value increases to 21% for the success cases. Although the average values are lower for FSMIM-S implementations, in the 75% of the cases, the ILP strategy saves on average 11% more LUTs per used EMB.

Finally, although the improvements in speed are small, the hit rates are at least 68% while the miss rates are at most 32%. These results show that the ILP strategy can improve the

Table 2 Percentage improvement with respect to OLD-FSMIM using large-sized test cases

	Arch.	Strat.	Mean	Std	Min	Q1	Q2	Q3	Max	HR	MR
EMB red.	FSMIM-S	ILP	9	22	– 100	0	0	10	50	43	1
		Greedy	4	20	– 100	0	0	3	50	26	10
	FSMIM-T	ILP	30	15	– 28	22	30	38	66	97	1
		Greedy	21	16	– 39	13	17	27	58	96	1
LUT red.	FSMIM-S	ILP	10	20	– 82	0	11	24	47	74	26
		Greedy	7	25	– 102	1	10	22	51	75	25
	FSMIM-T	ILP	44	27	– 33	28	48	65	86	90	7
		Greedy	36	24	– 36	23	33	55	83	90	4
SLPE red.	FSMIM-S	ILP	23	40	– 10	2	7	16	185	91	9
		Greedy	15	32	– 13	1	4	9	131	84	16
	FSMIM-T	ILP	53	37	0	28	44	63	195	98	2
		Greedy	36	34	0	16	23	40	140	98	0
Speed inc.	FSMIM-S	ILP	5	17	– 26	– 2	3	10	57	60	40
		Greedy	– 1	15	– 32	– 11	0	8	41	53	47
	FSMIM-T	ILP	12	7	– 6	8	12	17	35	96	4
		Greedy	8	5	– 5	4	7	11	21	94	6

Table 3 Percentage improvement of ILP with respect to Greedy using large-sized test cases

	Arch.	Mean	Std	Min	Q1	Q2	Q3	Max	Hit	Miss
EMB red.	FSMIM-S	4	16	– 83	0	0	4	50	32	3
	FSMIM-T	11	16	– 82	4	11	18	55	76	6
LUT red.	FSMIM-S	1	23	– 89	– 5	3	11	60	60	39
	FSMIM-T	14	19	– 29	0	15	23	71	72	18
SLPE inc.	FSMIM-S	7	20	– 29	0	1	5	100	72	26
	FSMIM-T	15	21	– 45	4	12	19	123	75	18
Speed inc.	FSMIM-S	7	15	– 21	– 1	2	14	57	68	32
	FSMIM-T	4	6	– 8	0	3	6	20	75	25

resources utilization without degrading or even increasing the speed respect to the greedy one.

As a conclusion, the greedy strategy, which is faster and does not require any solver, offers a good balance between the quality of results and the computation time; so, it is a suitable candidate if the requirements are not very demanding. However, if the design requirements are not met, the ILP strategy should be used.

4.3 Comparison between the proposed strategies and the conventional FSM implementations

The FSMIM implementations generated by the proposed strategies are compared with the conventional FSM implementations (CONV-ROM and CONV-LUT). This study uses the same benchmark set (the medium-sized test cases) and the same measures as in [14]. However, the logic structure of the target device (an Intel FPGA) is different from those used in previous work about FSMIM (Xilinx FPGAs). We have not used the large-sized test cases because they have been generates exclusively to evaluate the EMB-based FSM implementations, whose results mainly depend on structural properties of the FSM (number of states, transitions, inputs, effective inputs, and outputs). Therefore, we understand that these benchmarks provides less confidence to evaluate CONV-LUT, whose results significantly depend on the complexity of the transition and output functions. The FSMs of CONV-LUT and the ROMs of CONV-ROM have been described according to VHDL templates provided by Intel. All FSMIM approaches achieve an average EMB reduction with respect to CONV-ROM of at least 70%, with a hit rate of 100% (see Table 4). Regarding the number of used LUTs with respect to CONV-LUT (see Table 5), for all FSMIM approaches, the average reduction is at least 85% (with minimum reductions of at least 48%). Although FSMIM approaches use more LUTs than CONV-ROM, they attain greater SLPE values in all test cases, with an average increment of at least 941% (see Table 4). Such high values show that FSMIM approaches are more efficient that CONV-ROM in the use of EMBs, with FSMIM-S approaches being the most efficient, since they significantly reduce the number of LUTs by using a limited number of EMBs. Regarding speed, although the aim of the EMB-based techniques is to save LUTs, FSMIM-T approaches are faster than CONV-LUT in 35% of cases, with significant improvements in 25% of cases (see Q3 in Table 5). On average, CONV-ROM obtains slightly better results than FSMIM-T approaches but at the expense of a significant increase in the number of EMBs. Therefore, FSMIM-T approaches achieve the best balance between speed and resource usage. With a hit rate of at most 33%, FSMIM-S approaches obtain the worst results (this is the price to pay for being the technique that uses the least number of EMBs and saves the greatest number of LUTs per EMB [see Table 5]).

The critical path of CONV-ROM implementations includes one EMB and, eventually, the LUTs required to join EMBs [20]. However, the critical path of FSMIM implementations also includes the LUTs used to implement the ISB or the GE. So, due to the delay imposed by the ISB/GE, FSMIM implementations cannot compete in terms of speed with the CONV-ROM implementations that use a small number of EMBs. However, the number of LUTs required to join EMBs and the routing overhead increase with the number of EMBs, producing a significant increase of the critical path delay. When the number of EMBs of CONV-ROM implementations is big enough, the reduction of the critical path delay due to the reduction in the number of EMBs achieved by FSMIM approaches can compensate the penalty imposed by the ISB/GE. This effect can be observed in Fig. 3, which shows the relationship between the speed increment of FSMIM implementations with respect to CONV-ROM implementations and the ROM size of CONV-ROM. It includes the results obtained in FSMIM-S and FSMIM-T implementations using the ILP strategy for all test cases whose CONV-ROM implementation fits in the target device. The speed increment grows with the ROM size, specially in the right-hand side of the figure (for these cases, FSMIM approaches are the best EMB-based alternative for both speed and EMB usage). As could be expected, this trend is more clear in the case of the FSMIM-T architecture. We must highlight that 11 of the 57 medium-sized test cases and 64 of 72 large-sized test cases have been excluded because the implementation generated by CONV-ROM does not fit in the target device due to it requires more than 182 EMBs; however, these FSMs were successfully implemented by applying the FSMIM approaches.

Table 4 Percentage improvements with respect to CONV-ROM using medium-sized test cases

	Approach	Mean	Std	Min	Q1	Q2	Q3	Max	HR	MR
EMB red.	FSMIM-S-ILP	74	20	31	62	79	93	99	100	0
	FSMIM-S-Greedy	74	20	31	62	79	93	99	100	0
	FSMIM-T-ILP	70	23	10	57	76	88	99	100	0
	FSMIM-T-Greedy	70	24	10	57	76	88	99	100	0
SLPE inc.	FSMIM-S-ILP	1343	3445	26	92	318	1215	20724	100	0
	FSMIM-S-Greedy	1338	3453	26	92	320	1215	20724	100	0
	FSMIM-T-ILP	966	2079	13	114	298	654	11326	100	0
	FSMIM-T-Greedy	941	2072	13	114	298	645	11326	100	0

Table 5 Percentage improvements with respect to CONV-LUT using medium-sized test cases

	Approach	Mean	Std	Min	Q1	Q2	Q3	Max	HR	MR
LUT red.	FSMIM-S-ILP	86	11	48	84	88	92	98	100	0
	FSMIM-S-Greedy	85	11	48	82	88	92	98	100	0
	FSMIM-T-ILP	94	8	67	94	97	99	100	100	0
	FSMIM-T-Greedy	94	8	67	93	97	99	100	100	0
	CONV-ROM	97	4	77	96	99	100	100	100	0
Speed inc.	FSMIM-S-ILP	– 14	54	– 67	– 54	– 36	4	111	30	70
	FSMIM-S-Greedy	– 14	52	– 67	– 54	– 34	5	120	33	67
	FSMIM-T-ILP	4	70	– 63	– 49	– 21	29	180	35	65
	FSMIM-T-Greedy	4	70	– 63	– 49	– 21	26	179	35	65
	CONV-ROM	7	74	– 58	– 44	– 18	27	210	37	63

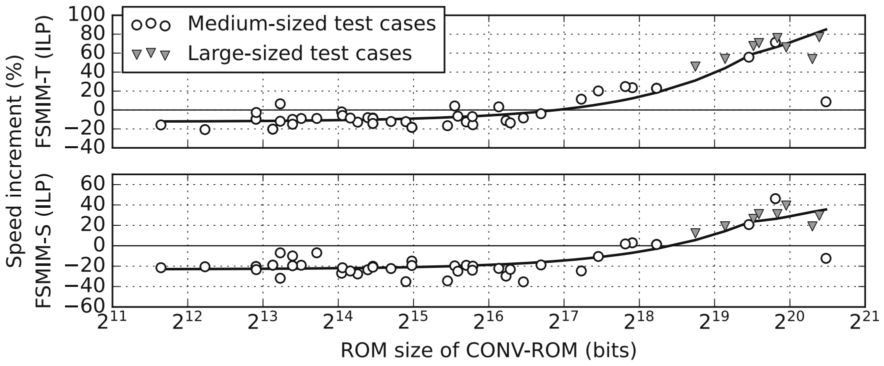


Fig. 3 Percentage speed increment with respect to CONV-ROM versus the ROM depth of CONV-ROM, and the corresponding LOWESS (LOcally WEighted Scatterplot Smoothing) curve

5 Conclusion and future work

In this paper, new techniques for optimizing FSMIM implementations have been proposed. As in previous work, the optimization process is divided into the ISS and SG stages. From these optimization techniques, two strategies that differ in the ISS stage have been established. In one of them, the greedy algorithm for the MMKPM problem (proposed in [19]) has been applied for the first time to the generation of FSMIM implementations. The other strategy is based on the MMKPM-SSD problem and the corresponding ILP formulation, which have been proposed in this paper to improve the optimization results. Finally, the EMB-granularity-based SG algorithm has been proposed to improve the results of the SG stage, which is applied in both strategies.

The proposed strategies have been compared to OLD-FSMIM [14] using both medium and large-sized FSM benchmark sets. Regarding the FSMIM-T architecture, these strategies significantly improve the area results. The ILP strategy reduces the number of EMBs in 44% of medium-sized test cases in which OLD-FSMIM uses more than 0.5 EMBs, and in 97% of large-sized test cases; in both benchmark sets, the average reduction for the success cases is at least 27% (there is only one case in which the number of EMBs increases). In addition, the average LUT reductions are 24% and 44%, with hit rates of 71% and 90%, respectively (and with insignificant miss rates of at most 7%). The ILP strategy save 25% and 53% more LUTs per each used EMB, with hit rates of 71% and 98% in medium-sized and large-sized test cases, respectively. The average speed increment is not significant for medium-sized test cases, although the hit rate is 70%; however, the improvement is more significant for large-sized test cases, with an average speed increment of 12% and a hit rate of 96%.

Regarding the FSMIM-S architecture, the proposed improvements in the optimization process have an indirect influence on the ROM size of this architecture. Due to this and to the smaller ROM size requirements of FSMIM-S implementations, the proposed strategies cannot improve the EMB usage for the medium-sized test cases. However, the ILP strategy reduces the number of EMBs in 43% of the large-sized test cases with an average EMB reduction of 23% for these cases (the miss rate is only 1%). The relationship between the selection cost and the complexity of the ISB of the FSMIM-S architecture is indirect, which reduces the effectiveness of the ISS stage on LUT usage and speed. Nevertheless, the ILP strategy save 9% and 23% more LUTs per each used EMB in medium-sized and large-sized test cases, respectively; in both sets, the hit rates are greater than or equal to 81%. Finally, although the average speed increments are not significant (at most 5%), the hit rates are greater than the miss rates.

Compared with the conventional techniques, the FSMIM approaches are an interesting design alternative. All FSMIM approaches achieve an average reduction of the number of EMBs with respect to CONV-ROM of at least 70% with a hit rate of 100%. Comparing the LUT usage with respect to CONV-LUT, all FSMIM approaches obtain reductions greater than 47%, with average reductions of at least 85%. The significant number of LUTs saved per EMB shows that the FSMIM approaches are more efficient that CONV-ROM in the use of EMBs, with the FSMIM-S approaches being the most efficient. Regarding speed, although the aim of the EMB-based techniques is to save resources, FSMIM-T approaches are faster than CONV-LUT in 35% of cases. On average, CONV-ROM obtains slightly better results than FSMIM-T approaches but at the expense of a significant increase in the number of EMBs. Therefore, FSMIM-T approaches achieve the best balance between speed and resource usage. Moreover, we have proved that there exist a clear trend between the speed increment obtained by FSMIM approaches and the size of the FSM; thus, for large-sized FSMs, FSMIM approaches are the best EMB-based alternative for both speed and EMB usage. The good implementation results obtained by the proposed strategies prove the feasibility of the FSMIM technique in devices with an architecture different to that of Xilinx FPGAs (which have been used in previous work [10, 13, 14]).

The average improvements obtained by the ILP strategy with respect to the greedy one for medium-sized test cases are not significant; however, as the miss rates are null or negligible for the FSMIM-T, the ILP strategy offers the possibility of improving the results with the confidence that they will never worse. For this architecture, in a 12% of cases, the ILP strategy uses, on average, 17% less EMBs than the greedy one; the hit rate for LUT reduction is 21% with with an average reduction for the success cases of 20%. For large-sized test cases, the improvements of the ILP strategy over the greedy one are much more significant, reaching an adequate balance between EMB and LUT usage. For instance, the hit rate in SLPE increment is 75% in FSMIM-T implementations; on average, the ILP strategy saves 15% more LUTs per used EMB than the greedy strategy, and this value increases to 21% for the success cases. Although the average values are lower for FSMIM-S implementations, in the 75% of the cases, the ILP strategy saves on average 11% more LUTs per used EMB.

As a conclusion, we propose some practical recommendations for researchers and designers in order to take advantage of the proposed architectures and strategies. The techniques studied could be sorted in ascending order of EMB usage and descending order of LUT usage as follows: CONV-LUT, FSMIM-S, FSMIM-T, and CONV-ROM. In addition, the ILP strategy requires fewer resources (both EMB and LUT) than the greedy one when both are applied to the same FSMIM architecture. The possibility of exploiting all kinds of resources available in FPGAs allows to fit the design into a smaller (and cheaper) device. FSMIM approaches allow to find an adequate trade-off between LUT and EMB usage. In fact, they obtain huge reductions in the LUT utilization by using a reasonable number of EMBs. The FSMIM-S architecture is the best design option when the number of unused EMBs is limited and the speed is not critical; otherwise, the FSMIM-T architecture is a better option. Regarding the strategies, the greedy one, which is faster and does not require any solver, offers a good balance between the quality of results and the computation time; so, it is a suitable candidate if the design requirements are not very demanding. However, if these requirements are not met, the ILP strategy should be used.

As future work, we plan to extend this work in order to improve the presented results. For each kind of FPGA device, the number of LUTs of a multiplexor can be estimated from the number of inputs. We want to modify the objective function of the MMKPM-SSD problem to quantify the number of LUTs required by the multiplexors of the FSMIM-T implementation. This will allow the ISS stage to find optimal solutions in terms of LUT usage, which will improve the area results. Regarding speed, the input selector with more inputs determines the critical path delay of the ISB; therefore, reducing the maximum number of inputs could reduce the delay. In this direction, we plan to propose another variant of the MMKPM-SSD for speed optimization that minimizes the maximum number of inputs of the input selectors instead of the selection cost. With aim of improving the implementation results for the FSMIM-S architecture, we plan to analyze the influence of the encoding of states and groups on the number of LUTs required by the ISB and the GE. Finally, we will develop a new version of the free FSMIM-Gen tool [28] that includes the proposed optimization process. FSMIM-Gen starts from the specification of the FSM in KISS format [26] and generates a synthesizable VHDL, which can be synthesized and implemented within the design flow of Electronic Design Automation (EDA) tools.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Barkalov A, Titarenko L (2009) Logic synthesis for FSM-based control units, vol 53. Springer, Heidelberg, Berlin
2. Klimowicz AS, Solov'ev VV (2015) Structural models of finite-state machines for their implementation on programmable logic devices and systems on chip. J Comput Syst Sci Int 54(2):230–242. https://doi.org/10.1134/S1064230715010074
3. Kubica M, Kania D (2020) Technology mapping of FSM oriented to LUT-based FPGA. Appl Sci 10(11):3926. https://doi.org/10.3390/app10113926
4. Rawski M, Selvaraj H, Luba T (2003) Digital system design, 2003. In: Proceedings. Euromicro Symposium on (2003), pp 104–110
5. El-Maleh A, Sait S, Nawaz Khan F (2006) Circuits and systems, 2006. In: ISCAS 2006. Proceedings. 2006 IEEE international symposium on (2006), p 4
6. Janarthanan A, Tiwari A, Tomko K (2007) Circuits and systems, 2007. In: MWSCAS 2007. 50th midwest symposium on (2007) pp 502–505
7. Mengibar L, Entrena L, Lorenz M, Millan E (2005) Partitioned state encoding for low power in FPGAs. Electron Lett 41(17):948–949. https://doi.org/10.1049/el:20052307
8. Sklyarov V (2002) Reconfigurable models of finite state machines and their implementation in FPGAs. J Syst Archit 47:1043–1064
9. Borowik G, Falkowski B, Luba T (2007) Design and diagnostics of electronic circuits and systems, 2007. DDECS '07. IEEE, pp 1–6
10. Garcia-Vargas I, Senhadji-Navarro R, Jimenez-Moreno G, Civit-Balcells A, Guerra-Gutierrez P (2007) Industrial electronics, 2007. In: ISIE 2007. IEEE international symposium on (2007), pp 2342–2347
11. Barkalov A, Titarenko L, Kolopienczyk M, Mielcarek K, Bazydlo G (2016) Design of EMB-based Mealy FSMs. Springer, Heidelberg, Berlin, pp 193–237
12. Tiwari A, Tomko K (2004) Design, automation and test in Europe conference and exhibition, 2004. Proceedings, vol 2, pp 916–921
13. Senhadji-Navarro R, Garcia-Vargas I, Jimenez-Moreno G, Civit-Ballcels A (2004) ROM-based FSM implementation using input multiplexing in FPGA devices. Electron Lett 40(20):1249–1251
14. Garcia-Vargas I, Senhadji-Navarro R (2015) Finite state machines with input multiplexing: a performance study. IEEE Trans Comput Aided Des Integr Circuits Syst 34(5):867–871. https://doi.org/10.1109/TCAD.2015.2406859
15. Selvaraj H, Rawski M, Luba T(2002) Proceedings. International conference on information technology: coding and computing, pp 355–360. https://doi.org/10.1109/ITCC.2002.1000415
16. Barkalov A, Titarenko L, Krzywicki K (2021) Structural decomposition in FSM design: roots, evolution, current state-a review. Electronics. https://doi.org/10.3390/electronics10101174
17. Das N, Priya PA (2019) FPGA implementation of an improved reconfigurable FSMIM architecture using logarithmic barrier function based gradient descent approach. Int J Reconfigurable Comput. https://doi.org/10.1155/2019/3727254
18. Mardani Kamali H, Zamiri Azar K, Homayoun H, Sasan A (2020) 2020 IEEE computer society annual symposium on VLSI (ISVLSI)
19. Garcia-Vargas I, Senhadji-Navarro R (2013) The minimum maximal k-partial-matching problem. Optim Lett 7(8):1959–1968. https://doi.org/10.1007/s11590-012-0531-3
20. Senhadji-Navarro R, Garcia-Vargas I, Guisado J (2012) Electronics, circuits and systems (ICECS). In: 2012 19th IEEE international conference on (2012), pp 225–228
21. Xilinx (2016) 7 series FPGAs configurable logic block: user guide
22. Altera (2011) Advanced synthesis cookbook
23. Yamada T, Kataoka S, Watanabe K (2002) Heuristic and exact algorithms for the disjunctively constrained knapsack problem. Inf Process Soc Jpn J 43:2864–2870

24. Pferschy U, Schauer J (2009) The knapsack problem with conflict graphs. J Gr Algorithms Appl 13(2):233–249
25. Kellerer H, Pferschy U, Pisinger D (2004) Knapsack problems. Springer, Berlin
26. Yang S (1991) Logic synthesis and optimization benchmarks user guide. version 3.0
27. Jozwiak L, Gawlowski D, Slusarczyk A (2004) Digital system design, 2004. In: DSD 2004. Euromicro symposium on (2004), pp 160 – 167. https://doi.org/10.1109/DSD.2004.1333272
28. Garcia-Vargas I (2013) FSMIM-Gen. http://personal.us.es/iggv/en/material.html