electronics

MDPI

Article

A New Approach for Implementing Finite State Machines with Input Multiplexing

Ignacio Garcia-Vargas *,† and Raouf Senhadji-Navarro †

Department of Computer Architecture and Technology, University of Seville, 41012 Seville, Spain; raouf@us.es
* Correspondence: iggv@us.es
† These authors contributed equally to this work.

Abstract: The model called Finite State Machine with Input Multiplexing (FSMIM) was proposed as a mechanism for implementing Finite State Machines (FSMs) using ROM memory. This paper presents a novel approach for achieving more efficient FSMIM implementations in Field Programmable Gate Array (FPGA) devices. The aim of the proposed approach is to obtain further reductions in the use of Embedded Memory Blocks (EMBs). Unlike previous works, the proposed approach reduces the depth of the ROM by grouping states before simplifying the input selectors of the FSMIM. For this purpose, a new strategy for grouping states is proposed, and its optimality is proven. In addition, a new variant of the Minimum Maximal *k*-Partial Matching (MMKPM) problem and its corresponding Integer Linear Programming (ILP) formulation are proposed for simplifying input selectors. The proposed approach requires a significantly smaller number of EMBs than the approaches proposed previously.

Keywords: Finite State Machine; Finite State Machine with input multiplexing; FPGA; Embedded Memory Block; ROM; Integer Linear Programming; Minimum Maximal *k*-Partial-Matching

check for updates

Citation: Garcia-Vargas, I.; Senhadji-Navarro, R. A New Approach for Implementing Finite State Machines with Input Multiplexing. *Electronics* **2023**, *12*, 3763. https://doi.org/10.3390/electronics12183763

Academic Editors: Dariusz Kania, Alexander Barkalov, Remigiusz Wiśniewski and Larysa Titarenko

Received: 28 July 2023
Revised: 1 September 2023
Accepted: 4 September 2023
Published: 6 September 2023

1. Introduction

Researchers have devoted significant attention to the optimization of Finite State Machine (FSM) implementations regarding area, speed, or power consumption over decades [1–8]. Current Field Programmable Gate Array (FPGA) devices include Embedded Memory Blocks (EMBs), which allow, among other elements, the implementation of ROM memory. Therefore, the output and transition functions of an FSM can be mapped into EMBs. The advantages of these implementations, called *ROM-based FSM implementations*, are the following: (1) if FSMs are mapped into non-used EMBs, the saved Lookup Tables (LUTs) can be used for implementing other components of the system [9]; (2) when the implementation of the function using LUTs requires a considerable number of logic levels, the speed can be increased by mapping the function on EMBs [10,11]; and (3) the EMBs used to implement the FSM can be disabled during the idle states, achieving a substantial reduction in power consumption [12].

However, when FSMs are mapped directly into ROM memory (we will refer to these implementations as *conventional ROM-based FSM implementations*), the size of the memory increases exponentially with the size of the ROM address, which is composed of the state encoding bits and the FSM inputs. Consequently, small FSMs may require more memory than that available in the FPGA device. Developing techniques that reduce memory usage without considerably increasing LUT usage is already a challenge.

In the literature, various techniques have been proposed to reduce the number of required EMBs [4,9–11,13]. These techniques enable a more efficient utilization of the FPGA resources, striking an appropriate balance between EMB and LUT usage. Most of these techniques employ functional decomposition [4,9,13] or structural decomposition [11,14]. In this scope, the model called FSM with Input Multiplexing (FSMIM) was proposed to reduce the number of ROM words required to map the FSM transitions [10]. Depending on whether the focus is on reducing EMBs or LUTs, two distinct architectures have been proposed: FSMIM with transition-based input selection (FSMIM-T) and FSMIM with state-based input selection (FSMIM-S) [10]. Both types of implementation use a combinational circuit, called *Input Selector Bank* (ISB), to select a different subset of the FSM inputs for each state, allowing for reducing the ROM memory. The optimization process proposed in [15] consists of two stages. In the first one, called *Input Selector Simplification* (ISS), the ISB is simplified with the aim of saving LUTs (which can also allow it to increase its speed). The optimization process employed in the ISS stage is modeled as a variant of the Minimum Maximal k-Partial-Matching (MMKPM) problem [16] and solved using an Integer Linear Programming (ILP) formulation [15] or a greedy algorithm [16]. In the second stage, known as *State Grouping* (SG), the memory depth is further reduced beyond the capabilities of the ISB alone. Other researchers have recently used the FSMIM model in their work [17,18].

To summarize, the key contributions of this paper are as follows:

- A new approach to generate FSMIMs from conventional FSMs is proposed with the aim of obtaining further reductions in EMB usage.
- Unlike previous approaches [10,15], the SG stage is applied before the ISS stage.
- A new strategy for the SG stage that obtains optimal solutions in terms of grouping states is proposed.
- The ILP formulation proposed in [15] for the ISS stage is extended in order to maintain coherence with the results obtained by the SG stage.
- We present a thorough experimental study comparing the new approach with the one proposed in [15].

The remaining sections of the paper are structured as follows: Backgrounds on ROM-based implementations and, particularly, FSMIM implementations are presented in Section 2. In Section 3, a new approach for generating FSMIMs from conventional FSMs is proposed. In Section 4, the experimental results are presented and analyzed. Finally, the main conclusions are highlighted in Section 5.

2. Background

2.1. Conventional ROM-Based Implementation

The conventional architecture for implementing FSMs into ROM memory is shown in Figure 1a, where p, m, and n are the number of state encoding bits, the number of inputs, and the number of outputs, respectively. By supposing that N_s is the number of states, the number of encoding bits is calculated as $p = \lceil \log_2 N_s \rceil$. Although there exists a specific architecture for Moore machines, in this work, we have considered a general architecture that allows to implementation of both Mealy and Moore machines [11]. Moreover, since current FPGAs include synchronous EMBs, FSM outputs are synchronized by the clock. Each word of the ROM block, which is implemented using EMBs, contains the next state and the FSM outputs of an FSM transition. Therefore, the ROM width is equal to $n + p$ bits.The FSM inputs and the present state encoding bits form the ROM address. Thus, the maximum ROM depth is equal to 2^{m+p} words, i.e., the maximum ROM size increases exponentially with m and p. If all the states are encoded with codes in the interval $[0, N_s)$, then the ROM depth is equal to $2^m N_s \leq 2^{m+p}$ words.

The exponential increase in the ROM depth not only has an impact on area but also on speed. The memory access time is degraded as the number of EMBs required to implement the ROM increases. The reasons for this behavior are the following: (1) the routing overhead and (2) the delay imposed by the LUTs used to connect the outputs of the EMBs whenever the ROM depth is greater than the maximum depth of the EMBs.

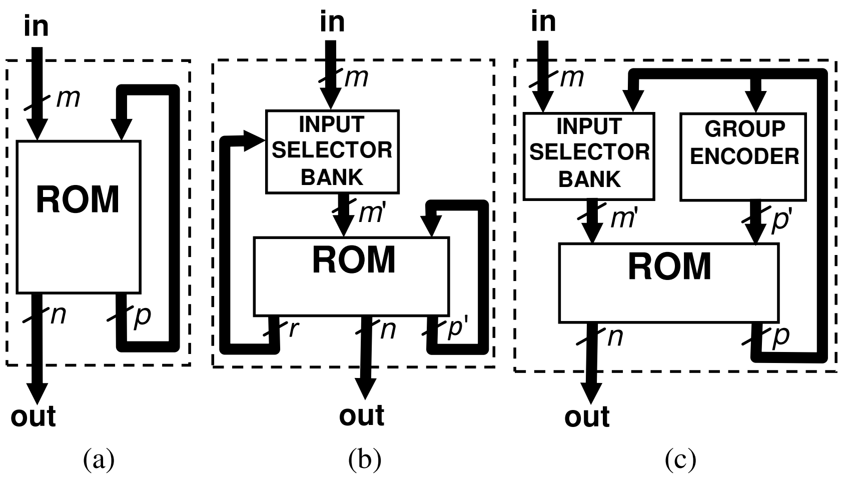


Figure 1. ROM-based FSM architectures: (**a**) conventional, (**b**) FSMIM-T, and (**c**) FSMIM-S.

2.2. Finite State Machine with Input Multiplexing

The FSMIM technique reduces the depth of the memory of conventional ROM-based FSM implementations by decreasing both the number of FSM inputs that address the ROM (m) and the number of states (N_s). The consequent reduction in the number of required EMBs is achieved at the expense of using LUTs. In FSMIM architectures (see Figure 1b,c), the ROM is addressed using, for each state, only the subset of FSM inputs that have an influence on its behavior (which are called *effective inputs*). In both architectures, the combinational circuit ISB selects the effective inputs of the present state. The ISB has m inputs (the FSM inputs) and m' outputs (the selected inputs); then, the ROM depth can be reduced if $m' < m$. Obviously, the value of m' is determined by the state with the greatest number of effective inputs. Given a state with $e < m'$ effective inputs, the $m' - e$ inputs that are not effective can be considered as *don't care* signals (which are referred to as *Don't Selected Inputs* (DCSIs)). In FSMIM architectures, the DCSIs can be exploited to form groups of states that can share the same state code. Therefore, in the ROM address, the code of the present state can be replaced by the code of the group to which it belongs. As a consequence, the ROM depth is reduced to $2^{m'}N_g$ words, where $N_g < N_s$ represents the number of groups. This allows a reduction in the ROM depth even in FSMs in which there exist states that respond to all FSM inputs (i.e., in which $m' = m$).

By supposing that the groups can be codified with $p' = \lceil \log_2 N_g \rceil$ bits, the ROM address requires $m' + p'$ bits; therefore, the ROM depth is reduced with respect to conventional ROM-based implementations by $N_s 2^m - N_g 2^{m'}$ words, which not only reduces the memory requirement but also the memory access time of the ROM (despite the fact that speed is not considered as a goal of the FSMIM technique).

There exist two different architectures for implementing FSMIMs, namely, FSMIM-T (see Figure 1b) and FSMIM-S (see Figure 1c). In both architectures, the selection of the effective inputs is performed by the called *input selectors*. In the FSMIM-T architecture (see Figure 1b), the ISB is composed of multiplexers, so its implementation can greatly benefit from the embedded multiplexers available in many FPGA devices [19,20]. For each transition, the ROM stores the bits that control the multiplexers (called *selection bits*), the FSM outputs, and, instead of the next state, the encoding bits of the next group. By supposing that the number of selection bits is r, the ROM width is $(r + p' + n)$ bits, which is greater than the width of the conventional ROM-based implementations.

In the FSMIM-S architecture (see Figure 1c), each ROM word contains the same information as in the conventional ROM-based implementations (i.e., contains the next state encoding bits and FSM outputs). Therefore, unlike FSMIM-T, FSMIM-S can reduce the depth of the ROM without increasing its width. However, the number of LUTs required by the corresponding implementation increases due to the following reasons. First, the multiplexers of FSMIM-T use the minimum number of selection bits. However, the ISB of

FSMIM-S uses the present-state encoding bits as selection bits, increasing the complexity of the corresponding logic functions. Second, as the ROM of FSMIM-S stores the next state instead of the next group, the architecture requires an extra combinational component (called *Group Encoder*) to generate the present group from the present state, which has p inputs and p' outputs.

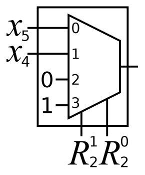
In conclusion, we can say the FSMIM-T architecture should be used when the LUT usage and speed are critical, whereas FSMIM-S should be used when the number of available EMBs is limited [10].

2.3. Generation of FSMIMs from FSMs

An approach to generate FSMIMs from FSMs was presented in [15] with the objective of enabling efficient FSMIM implementations in terms of LUT and EMB usage. The *Input Selection Matrix* (ISM) [10], which can be considered an abstraction of the ISB, is used to describe the optimization problems arising in the generation of FSMIMs. In the ISM, rows and columns represent states and input selectors, respectively. The effective inputs of each state are represented by a different row, and each column represents the set of FSM inputs that can be selected by an input selector. More formally, let $S = \{s_1, s_2, \ldots, s_q\}$ be the set of states, $X = \{x_1, x_2, \ldots, x_m\}$ the set of FSM inputs, and k the maximum number of effective inputs. Then, an ISM is defined as a matrix, $A = (a_{i,j}) \in \mathcal{M}_{q\times k}$, where $a_{i,j} \in X \cup \{0, 1, -\}$ is the input selected by the input selector, j, for the state, s_i. The selected input can be a constant value in addition to an FSM input or a *don't care*. To indicate the correspondence between ISM rows and states, the ISM can include a supplementary column that specifies the state name associated with each row. Different ISMs, which determine different FSMIM implementations, can be obtained from the same FSM. For example, see the two ISMs shown in Figure 2a.

$$A = \left[\begin{array}{ccc|c} x_3 & x_4 & x_1 & s_1 \\ - & x_6 & x_3 & s_2 \\ x_2 & x_5 & - & s_3 \\ - & x_1 & - & s_4 \\ - & x_4 & - & s_5 \\ x_1 & - & - & s_6 \\ x_5 & - & - & s_7 \end{array}\right] \qquad A^{ISS} = \left[\begin{array}{ccc|c} x_1 & x_4 & x_3 & s_1 \\ x_6 & - & x_3 & s_2 \\ x_2 & x_5 & - & s_3 \\ x_1 & - & - & s_4 \\ - & x_4 & - & s_5 \\ x_1 & - & - & s_6 \\ - & x_5 & - & s_7 \end{array}\right]$$

(**a**)



(**b**)

$$A_1^{SG} = \left[\begin{array}{ccc|c} x_1 & x_4 & x_3 & s_1 \\ x_6 & - & x_3 & s_2 \\ x_2 & x_5 & - & s_3 \\ x_1 & - & - & s_4 \\ 0 & x_4 & - & g_{57} \\ x_1 & - & - & s_6 \\ 1 & x_5 & - & g_{57} \end{array}\right] \quad A_2^{SG} = \left[\begin{array}{ccc|c} x_1 & x_4 & x_3 & s_1 \\ x_6 & 0 & x_3 & g_{24} \\ x_2 & x_5 & - & s_3 \\ x_1 & 1 & - & g_{24} \\ 0 & x_4 & - & g_{57} \\ x_1 & - & - & s_6 \\ 1 & x_5 & - & g_{57} \end{array}\right] \quad A_3^{SG} = \left[\begin{array}{ccc|c} x_1 & x_4 & x_3 & g_1 = \{s_1\} \\ x_6 & 0 & x_3 & g_{24} \\ x_2 & x_5 & 0 & g_{357} \\ x_1 & 1 & - & g_{24} \\ 0 & x_4 & 1 & g_{357} \\ x_1 & - & - & g_6 = \{s_6\} \\ 1 & x_5 & 1 & g_{357} \end{array}\right]$$

(**c**)

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

The approach presented in [15] is divided into two stages: the first one is the ISS stage, which is intended to simplify the ISB, and the second one is the SG stage, which groups the states to reduce the ROM depth. Given an FSM, the two stages process the corresponding ISM in sequence: the ISS stage permutes the elements of ISM rows; then, the SG stage assigns constant values. The ISM produced by the SG stage determines the implementation of the generated FSMIM.

2.3.1. ISS Stage

The primary objective of the ISS stage is the reduction in LUT count by simplifying the ISB. Moreover, the critical path often includes the ISB, especially in FSMIM-T implementations; therefore, reducing the complexity of the ISB can also improve the speed of the corresponding implementations.

As the selectors with fewer inputs need generally fewer LUTs, the ISB complexity can be estimated as the total number of inputs of the selectors. This measure, which is called *selection cost*, is calculated as the sum of the number of distinct values in each column of the ISM. Since the complexity of the ISB in FSMIM-T is determined exclusively by the number of inputs of their selectors (i.e., their multiplexers), it correlates directly with the selection cost. However, in FSMIM-S, this correlation is weaker because the complexity of the involved logic functions depends on other factors in addition to input count (e.g., the states encoding).

Given an FSMIM, permuting the elements within each row of the corresponding ISM defines an equivalent FSMIM, the transitions of which are allocated in different ROM words. This operation over an ISM is called a *permutation*. The ISS stage finds the permutation of the ISM that minimizes the selection cost (i.e., the sum of the input count of each selector of the ISB). In the example of Figure 2a, A^{ISS} is a permutation of A in which the input count of the ISB is reduced in the following way: from 4 to 3 for the first selector, from 4 to 2 for the second one, and from 2 to 1 for the last one (see columns 1, 2, and 3, respectively).

In [15], the ISS stage is modeled as a problem called *MMKPM with minimum selection cost, number of selection bits, and DCSI dispersion* (MMKPM-SSD), which is a variant of the MMKPM problem [16]. MMKPM-SSD is described in terms of a bipartite graph obtained from an ISM without constant values (the constant values are added by the SG stage, which is applied after the ISS stage). Given an ISM $A = (a_{i,j}) \in \mathcal{M}_{q\times k}$, where $a_{i,j} \in X \cup \{-\}$, a bipartite graph, $G = (R \cup X, E)$, is defined in which the rows of A are represented by $R = \{1, \ldots, q\}$ and the FSM inputs by X. Each edge of $E \subseteq R \times X$ maps a row of A to its effective inputs. A set of edges in G without common rows is referred to as a *partial-matching*, and a collection of k partial-matchings $P = \{P_1, \ldots, P_k\}$ is referred to as *k-partial-matching*. A k-partial-matching is *maximal* if it defines a partition of E. Each partial-matching of a maximal k-partial-matching represents a different column of A because it selects one input from each row (or none if it is a DCSI). Therefore, any permutation of A can be represented by a maximal k-partial-matching, where P_i determines the inputs of column i. A low dispersion of DCSIs in the ISM columns (i.e., a high concentration of DCSIs in few columns) benefits the SG stage, which is applied after the ISS stage. The DCSI dispersion, in addition to the selection cost and the number of selection bits, can be calculated from a maximal k-partial-matching [15].

The objective of the MMKPM-SSD problem is to find a maximal k-partial-matching with minimum selection cost, minimum number of selection bits, and minimum DCSI dispersion (in that order of priority). An ILP formulation to solve the MMKPM-SSD problem was also proposed in [15].

2.3.2. SG Stage

Given an ISM, $A = (a_{i,j}) \in \mathcal{M}_{q\times k}$, and a state, $s \in S$, the *domain* of s, denoted by $\delta(s) \subseteq \mathbb{B}^k$, is defined as the set of values that the selected inputs of s can take. Obviously, if no constant values are selected for a state, then its domain is equal to $\mathbb{B}^k$. For example, in the ISM A^{ISS} of Figure 2, $\delta(s_4) = \mathbb{B}^3$. However, the assignment of constant values to the DCISs of a state reduces the cardinality of its domain. For example, if both DCSIs of s_4 are assigned to 0, then $\delta(s_4) = \{(0,0,0), (1,0,0)\}$. The cardinality of the domain of a state, s, satisfies that

$$2^{w_s} \le |\delta(s)| \le 2^k \tag{1}$$

where w_s is the number of effective inputs of the state, s.

The ROM address is formed by the present-state encoding bits and k input signals. Hence, each state code is associated with 2^k ROM words. However, the set of ROM addresses in which the state transitions are allocated is determined by the state encoding bits and their domain. Therefore, if the domains of a group of states are disjoint, then the states in the group can be encoded with the same code and share the 2^k addresses associated with such code, which allows for reducing the depth of the ROM (hereinafter, these groups of states will simply be called *groups*).

The SG stage reduces the depth of the ROM by grouping states. For each group, the SG stage finds an assignment of constant values that ensures the domains of its states are disjoint. For example, in Figure 2, the group, g_{24}, in A_2^{SG} can be used to identify the states, s_2 and s_4 (see A_2^{ISS}), because their domains differ in the value of the second column. We will say that s_2 and s_4 are merged into the group $g_{24} = \{s_2, s_4\}$. With this assignment of constant values, whenever the present state is g_{24}, the transitions of s_2 or s_4 can be addressed by selecting in the second input selector the values 0 or 1, respectively. Therefore, in the corresponding state transitions, the next state, s_2, must be substituted by g_{24} with input selection $(x_6, 0, x_3)$, whereas s_3 must be substituted by g_{24} with $(x_1, 1, -)$. For example, Figure 2b shows the multiplexer that implements the second input selector for FSMIM-T.

The assignment of constant values may have a negative effect on the speed and area of FSMIM Implementations. In FSMIM-T, the number of selection bits grows, which increases the number of LUTS (due to the increase in the complexity of the multiplexers) and the number of EMBs (due to the increase in the width of the ROM). In FSMIM-S, the number of EMBs is not affected because the ROM width does not depend on the selection cost; however, the number of LUTs does increase due to the increased complexity of the logic functions that implement the ISB, which have fewer *don't care* outputs. Consequently, the SG stage must stop forming groups when the number of EMBs can no longer be reduced.

The algorithm presented in [15] for the SG stage is called a *EMB-granularity-based SG* algorithm (hereinafter, simply, SG algorithm (SGA)). It starts from an ISM without constant values, in which each state forms a separate group; therefore, the initial number of groups is equal to the number of states. The ISM columns are processed iteratively. At each step of the algorithm, two groups are merged by assigning constant values to the DCSIs of the processed column; then, the number of required EMBs is estimated from the resulting ISM. If the number of EMBs decreases, the ISM is selected as a candidate solution. The processing of the column continues until no more groups can be merged; then, the algorithm proceeds with the next column. After processing all columns, the final candidate solution is considered the best solution. That is, among all the solutions found with the smallest number of EMBs, the best one is the solution with the largest number of groups, as it has the least negative impact on speed and area.

An example of the application of SGA is shown in Figure 2 (where A_l^{SG} represents the result of processing column l by applying SGA to A^{ISS}). First, s_5 and s_7 are merged into $g_{57} = \{s_5, s_7\}$ by setting the DCSIs of the first column (see A_1^{SG}). Then, the algorithm processes the second column and merges s_2 and s_4 into $g_{24} = \{s_2, s_4\}$ (see A_2^{SG}). Finally, during the processing of the third column, s_3 and g_{57} are merged into $g_{357} = \{s_3, s_5, s_7\}$ by fixing the DCSI of s_3 to 0 and all DCSIs of g_{57} to 1 (see A_3^{SG}). This is possible because all rows of g_{57} and s_3 have a DCSI in the third column. In contrast, although g_{24} has a DCSI in the third column, it also has an effective input; therefore, it cannot be merged with s_6. The final set of groups is $G = \{g_1, g_{24}, g_{357}, g_6\} = \{\{s_1\}, \{s_2, s_4\}, \{s_3, s_5, s_7\}, \{s_6\}\}$; then, the number of groups is reduced from 7 to 4.

Note that the set of groups obtained by SGA is always a partition of S.

3. Proposed Approach to Generate FSMIMs from FSMs

By applying the ISS stage before the SG stage, the approach proposed in [15] for generating FSMIM implementations gives priority to the simplification of the input selectors over state grouping. As a consequence, in general, the ROM depth (and, thus, the number of used EMBs) cannot be reduced as much as possible. With the aim of further reducing the number of EMBs, this paper presents a new approach that prioritizes state grouping, which requires, among other things, applying SG before ISS.

Depending on the FSMIM architecture, the new approach consists of two or three stages. In FSMIM-S, the approach applies an initial SG stage followed by an ISS stage. However, FSMIM-T requires that an additional SG stage be applied after ISS.

3.1. Proposed SG Stage

The number of groups obtained by SGA depends on the positions of DCSIs in the ISM. However, the minimum number of groups only depends on the number of effective inputs of each state. Given an ISM, $A = (a_{i,j}) \in \mathcal{M}_{q\times k}$, the set of groups, $G = \{g_1, \ldots, g_t\}$, obtained by SGA verifies that

$$\left\lceil \frac{\sum_{s\in S} 2^{w_s}}{2^k} \right\rceil \leq |G| \tag{2}$$

where w_s is the number of effective inputs of s.

Proof. Let us define the domain of a group, $g \subseteq S$, as $\Delta(g) = \cup_{s\in g}\delta(s) \subseteq \mathbb{B}^k$. By construction, the domains of any pair of states of a group, $g \in G$, are disjoint. Then, $|\Delta(g)| = |\cup_{s\in g}\delta(s)| = \sum_{s\in g}|\delta(s)|$. Therefore, using (1), we have that

$$\sum_{s\in g} 2^{w_s} \leq |\Delta(g)| \leq 2^k \tag{3}$$

Since G is a partition of S and (3) is satisfied, then

$$\left\lceil \frac{\sum_{s\in S} 2^{w_s}}{2^k} \right\rceil = \left\lceil \frac{\sum_{i=1}^{t}\sum_{s\in g_i} 2^{w_s}}{2^k} \right\rceil \leq \left\lceil \frac{\sum_{i=1}^{t}|\Delta(g_i)|}{2^k} \right\rceil \leq t = |G|$$

□

In the approach presented in [15], the SG stage starts from the ISM generated by the previous ISS stage. In general, the placement of the DCSIs in the ISM is not optimal due to the permutations performed by the ISS stage; therefore, the SG stage cannot reach the minimum number of groups. For example, in Figure 2c, the final number of groups obtained by the SG algorithm is 4 (see A_3^{SG}); however, the minimum number of groups is $\left\lceil \frac{2^3+2^2+2^2+2^1+2^1+2^1+2^1}{2^3} \right\rceil = 3$.

Unlike previous work, the proposed method starts with an SG stage, which permutes the elements of each ISM row to find an optimal distribution of the DCSIs. This permutation, called *initial SG permutation*, moves the DCSIs to the last columns of each row. More formally, given an ISM, $A = (a_{i,j}) \in \mathcal{M}_{q\times k}$, the initial SG permutation arranges the elements of each row of A in such a way that all $a_{i,j}, a_{i,l} \in A$ satisfy that $j < l$ if $a_{i,j}$ is not a DCSI and $a_{i,l}$ is a DCSI. It can be proven that SGA reaches the minimum number of groups when it is applied to the resultant ISM of the initial SG permutation. Figure 3 shows the application of the proposed initial SG stage to the FSMIM of Figure 2, where A^π is obtained by applying the initial SG permutation to A.

$$A^\pi = \left[\begin{array}{ccc|c} x_3 & x_4 & x_1 & g_1 \\ x_6 & x_3 & - & g_2 \\ x_2 & x_5 & - & g_3 \\ x_1 & - & - & g_4 \\ x_4 & - & - & g_5 \\ x_1 & - & - & g_6 \\ x_5 & - & - & g_7 \end{array}\right] \qquad \text{where } g_i = \{s_i\}$$

$$A_1^\pi = A^\pi \qquad A_2^\pi = \left[\begin{array}{ccc|c} x_3 & x_4 & x_1 & g_1 \\ x_6 & x_3 & - & g_2 \\ x_2 & x_5 & - & g_3 \\ x_1 & 0 & - & g_{45} \\ x_4 & 1 & - & g_{45} \\ x_1 & 0 & - & g_{67} \\ x_5 & 1 & - & g_{67} \end{array}\right] \qquad A_3^\pi = \left[\begin{array}{ccc|c} x_3 & x_4 & x_1 & g_1 \\ x_6 & x_3 & 0 & g_{23} \\ x_2 & x_5 & 1 & g_{23} \\ x_1 & 0 & 0 & g_{4567} \\ x_4 & 1 & 0 & g_{4567} \\ x_1 & 0 & 1 & g_{4567} \\ x_5 & 1 & 1 & g_{4567} \end{array}\right]$$

Figure 3. Initial SG stage applied to the FSMIM example of Figure 2.

Lemma 1. *Let $A^{\pi} = (a_{i,j}) \in \mathcal{M}_{q \times k}$ be the ISM obtained by the initial SG permutation; $G_0^{\pi} = \{\{s_1\}, \dots \{s_q\}\}$ be the initial set of groups; and G_l^{π} be the set of groups obtained by SGA after the l-th column of A^{π} is processed. For all $1 \le l \le k$, it holds that*

$$|G_l^{\pi}| = \left\lceil \frac{c_l + |G_{l-1}^{\pi}|}{2} \right\rceil \tag{4}$$

where c_l is the number of effective inputs in column l.

Proof. Let $A_{l-1}^{\pi} = (a_{i,j}) \in M_{q \times k}$ be the ISM obtained after the column $l-1$ is processed. G_l^{π} is obtained by assigning constant values to the l-th column of A_{l-1}^{π} and by merging the groups of G_{l-1}^{π}. For example, in Figure 3, G_3^{π} (see the last column of A_3^{π}) is obtained by assigning constant values to column 3 of A_2^{π}.

Let us suppose that $a_{i,l}$ is an effective input (i.e., it is not a DCSI). Then, due to the initial SG permutation, $a_{i,r}$ is also an effective input for all $r < l$. This implies that the group, $g \in G_{l-1}^{\pi}$, related to row i could not be merged with other groups during the processing of previous columns. Thus, it contains exclusively one state (s_i). In addition, it cannot be merged with other groups during the processing of column l. We will refer to these groups as *isolated groups* of G_{l-1}^{π}. For example, in Figure 3, the unique isolated group of G_2^{π} is g_1 because it has the unique effective input in the third column of A_2^{π}; the remaining groups are non-isolated.

The number of isolated groups of G_{l-1}^{π} is equal to the number of effective inputs in column l (i.e., equal to c_l). Then, the number of non-isolated groups is $|G_{l-1}^{\pi}| - c_l$. As the non-isolated groups have a DCSI in column l of each one of their rows, they can be merged by pairs during the processing of column l. Therefore, the final number of groups of G_l^{π} is

$$|G_l^{\pi}| = c_l + \left\lceil \frac{|G_{l-1}^{\pi}| - c_l}{2} \right\rceil = \left\lceil \frac{c_l + |G_{l-1}^{\pi}|}{2} \right\rceil$$

□

Proposition 1. *SGA reaches the minimum number of groups determined by (2) when it is applied to A^{π}.*

Proof. Rewriting (2) in terms of ISM rows instead of states and taking into account that the final number of groups is that obtained after processing the last column (i.e., the k-th column), it must be proven that

$$|G_k^{\pi}| = \left\lceil \frac{\sum_{i=1}^{q} 2^{w_i}}{2^k} \right\rceil \tag{5}$$

where w_i is the number of effective inputs of state s_i.

Let $w_i^{(r)}$ be the number of effective inputs in the first r columns of row i of A^{π}. Due to the initial SG permutation, for all $r > 1$, $w_i^{(r)}$ is equal to either $w_i^{(r-1)}$ (if $a_{i,r}$ is a DCSI) or $w_i^{(r-1)} + 1$ (if $a_{i,r}$ is an effective input). Therefore, the number of effective inputs in column r, with $r > 1$, can be calculated as $c_r = \sum_{i=1}^{q} \left(w_i^{(r)} - w_i^{(r-1)} \right)$.

As $w_i = w_i^{(k)}$, the proposition is proven if

$$|G_r^{\pi}| = \left\lceil \frac{\sum_{i=1}^{q} 2^{w_i^{(r)}}}{2^r} \right\rceil \tag{6}$$

is satisfied for all $r = 1, \dots, k$, which will be proven by induction.

For $r = 1$, as c_1 is the number of effective inputs in column 1, we have c_1 rows with $w_i^{(1)} = 1$ and $q - c_1$ rows with $w_i^{(1)} = 0$. Then, $\sum_{i=1}^{q} 2^{w_i^{(1)}} = 2^1 c_1 + (q - c_1)2^0 = c_1 + q = c_1 + |G_0^\pi|$. Therefore,

$$\left\lceil \frac{\sum_{i=1}^{q} 2^{w_i^{(1)}}}{2} \right\rceil = \left\lceil \frac{c_1 + |G_0^\pi|}{2} \right\rceil$$

which is equal to $|G_1^\pi|$ using (4).

For the inductive step, let us suppose that (6) is true for $|G_r^\pi|$. Using (4),

$$\begin{aligned}
|G_{r+1}^\pi| &= \left\lceil \frac{c_{r+1} + |G_r^\pi|}{2} \right\rceil \\
&= \left\lceil \frac{c_{r+1} + \left\lceil \frac{\sum_{i=1}^{q} 2^{w_i^{(r)}}}{2^r} \right\rceil}{2} \right\rceil \\
&= \left\lceil \frac{\left\lceil \frac{2^r c_{r+1} + \sum_{i=1}^{q} 2^{w_i^{(r)}}}{2^r} \right\rceil}{2} \right\rceil \\
&= \left\lceil \frac{2^r c_{r+1} + \sum_{i=1}^{q} 2^{w_i^{(r)}}}{2^{r+1}} \right\rceil \\
&= \left\lceil \frac{2^r \sum_{i=1}^{q} \left(w_i^{(r+1)} - w_i^{(r)} \right) + \sum_{i=1}^{q} 2^{w_i^{(r)}}}{2^{r+1}} \right\rceil \\
&= \left\lceil \frac{\sum_{i=1}^{q} \left(2^r \left(w_i^{(r+1)} - w_i^{(r)} \right) + 2^{w_i^{(r)}} \right)}{2^{r+1}} \right\rceil
\end{aligned}$$

Using the initial SG permutation, we have two possibilities for each row, i. If $a_{i,r+1}$ is a DCSI, then $w_i^{(r+1)} = w_i^{(r)}$. Thus, $2^r \left(w_i^{(r+1)} - w_i^{(r)} \right) + 2^{w_i^{(r)}} = 2^{w_i^{(r)}} = 2^{w_i^{(r+1)}}$. If $a_{i,r+1}$ is an effective input, then $w_i^{(r)} = r$ and $w_i^{(r+1)} = r + 1$. Hence, $2^r \left(w_i^{(r+1)} - w_i^{(r)} \right) + 2^{w_i^{(r)}} = 2^r + 2^r = 2^{r+1} = 2^{w_i^{(r+1)}}$. In both cases, it holds that $2^r \left(w_i^{(r+1)} - w_i^{(r)} \right) + 2^{w_i^{(r)}} = 2^{w_i^{(r+1)}}$; therefore,

$$|G_{r+1}^\pi| = \left\lceil \frac{\sum_{i=1}^{q} 2^{w_i^{(r+1)}}}{2^{r+1}} \right\rceil$$

Thus, the proposition is proven. □

By applying the initial SG stage to the ISM A of Figure 2, the set of groups obtained is $G_3^\pi = \{g_1, g_{23}, g_{4567}\} = \{\{s_1\}, \{s_2, s_3\}, \{s_4, s_5, s_6, s_7\}\}$ (see the last column of A_3^π in Figure 3); therefore, SGA reaches the minimum number of groups, which is 3, using (2).

The optimization process for the FSMIM-S architecture is composed of two stages: the initial SG stage followed by the new ISS stage. As explained above, the initial SG stage first applies the initial SG permutation and then SGA, which takes into account the number of EMBs required by each candidate solution (see Section 2.3.2). Unlike the old ISS stage, which does not manage ISMs with constant values, the new one takes into account the existence of the previously created groups. Thus, only permutations that do not affect such groups are allowed.

However, in FSMIM-T, the ROM width is overestimated during the initial SG stage due to the fact that the selection cost is not minimized until the subsequent ISS stage. As a consequence, SGA cannot carry out an accurate estimation of the number of EMBs. To solve this problem, the optimization process uses three stages: (1) the initial SG stage, in which SGA is applied with the EMB estimation disabled (thus, the algorithm obtains the minimum number of groups but not the best solution); (2) the new ISS stage; and (3) the final SG stage in which SGA is applied, with the EMB estimation enabled, to the resultant ISM but without any constant assigned (i.e., without any group created). The number of groups is the minimum after the two first stages. Therefore, even though the constant values of the ISM are removed in the third stage, and a minimal solution exists for the distribution of the resultant DCSIs. As a consequence, there is a high probability that SGA will once again obtain an optimal solution, even if such a DCSI distribution does not match the initial SG permutation. The reduction in the ROM width is performed during the ISS stage, allowing SGA to incorporate the EMB estimation in the third stage.

3.2. Proposed ISS Stage

Like in the previous approach [15], the ISS simplification can be carried out by both an ILP formulation and a greedy algorithm. In this section, both approaches are described.

3.2.1. ILP Formulation for ISS Stage

To find solutions when the ISS stage is applied after the SG stage, this paper proposes a variant of the MMKPM-SSD called *MMKPM with the minimum number of selection bits and minimum selection cost with links* (MMKPM-SSL). In order to maintain the relationship between the constants established in the previous SG stage, the concept of *link* is introduced: a link guarantees that a set of DCSIs that have been assigned to merge two groups will always belong to the same column of the ISM regardless of the permutations conducted. The constants involved will be called *linked constants*. For example, the constants assigned to $a_{2,3}$ and $a_{3,3}$ in A_3^π (see Figure 2) are linked because they allow the formation of a group (g_{23}). Therefore, any permutation that locates them in different columns is not allowed. Like the MMPKM-SSD problem, the MMKPM-SSL problem minimizes the selection cost and the number of selection bits; however, the minimization of the DCSI dispersion is not an objective in the new optimization problem due to the fact that its purpose was to favor the subsequent state grouping (note that, in the new approach, the SG stage is performed first, resulting in an optimal solution).

Unlike the MMKPM-SSD problem, the MMKPM-SSL problem prioritizes the minimization of the number of selection bits over the selection cost. The reason is explained as follows. The goal of MMPKM-SSL is to reduce the number of EMBs. In FSMIM-T, the number of selection bits has a direct influence on the ROM width. However, in FSMIM-S, the number of EMBs does not depend on the input selection; therefore, it is not clear which of the two objectives must have more priority. For homogeneity, we have used the same criterion in both architectures.

Let $X = \{x_1, \ldots, x_m\} \cup \{0, 1\}$ be the set of FSMIM inputs, which also includes the constant values 0 and 1 added by the SG stage. After the initial SG procedure, each row of the ISM can contain multiple instances of the same constant value, which will correspond to the same edge of the bipartite graph; therefore, a multigraph (instead of the simple graph used in the MMKPM-SSD problem) is required to represent the relation between rows and inputs.

Let $G = (R \cup X, E, \gamma)$ be a bipartite multigraph, where R and X are the vertex sets, E is the edge set, and $\gamma : E \to R \times X$ is the function that maps edges to pairs of vertices. An edge, $e \in E$, is incident to the vertices, $r \in R$ and $x \in X$ iff $\gamma(e) = (r, x)$. Given an ISM, $A = (a_{i,j}) \in \mathcal{M}_{q \times k}$, where $a_{i,j} \in X \cup \{-\}$, let us define the bipartite multigraph, $G = (R \cup X, E, \gamma)$, with $E = \{(i, j) : a_{i,j} \in X\}$ and $\gamma((i, j)) = (i, a_{i,j})$. That is, for each element $a_{i,j}$ of A, such that $a_{i,j} \equiv x \in X$ (which implies that it is not a DCSI), an edge, $e = (i, j) \in E$, incident to row $i \in R$ and to input $x \in X$ is created (i.e., $\gamma(e) = (i, x)$). In the resultant multigraph, any pair of edges incident to a pair of linked constants are called *linked edges*. All definitions and properties related to the generic MMKPM problem [16] and its variant MMKPM-SSD [15] can be applied to the multigraph G.

Given A and G, the objective of the MMKPM-SSL problem is to find a maximal k-partial-matching $P = \{P_1, \ldots, P_k\}$ in G such that the tuple

$$\left(\sum_{i=1}^{k} \lceil \log_2 |X(P_i)| \rceil,\ \sum_{i=1}^{k} |X(P_i)| \right) \tag{7}$$

is minimum in lexicographical order, where $X(P_i)$ is the set of inputs incident to any edge of P_i, the expression $\sum_{i=1}^{k} |X(P_i)|$ is equal to the selection cost, and $\sum_{i=1}^{k} \lceil \log_2 |X(P_i)| \rceil$ is equal to the number of selection bits.

The MMKPM-SSL problem can be solved by using the ILP formulation described below, which is an extension of the formulation for the MMKPM-SSD proposed in [15].

Let the variables $y_{x,i}, z_{e,i} \in \{0,1\}$ be defined as

$$y_{x,i} = \begin{cases} 1 & \text{if } x \in X(P_i) \\ 0 & \text{otherwise} \end{cases} \qquad \forall x \in X,\ i = 1, \ldots, k, \tag{8}$$

$$z_{e,i} = \begin{cases} 1 & \text{if } e \in P_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall e \in E,\ i = 1, \ldots, k, \tag{9}$$

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 sequence of $\lceil \log_2 |X(P_i)| \rceil$consecutive 1 s.

By supposing that

$$D = \lceil \log_2 \min\{|R|, |X|\} \rceil \tag{10}$$

and

$$Q = k \min\{|R|, |X|\} \tag{11}$$

the following ILP formulation allows us to solve the following MMKPM-SSL problem:

$$\text{min. } Q \sum_{i=1}^{k} \sum_{j=1}^{D} d_{j,i} + \sum_{i=1}^{k} \sum_{x \in X} y_{x,i} \tag{12}$$

subject to

$$\sum_{e \in E(r)} z_{e,i} \leq 1 \qquad \forall r \in R,\ i = 1, \ldots, k, \tag{13}$$

$$\sum_{i=1}^{k} z_{e,i} \leq 1 \qquad \forall e \in E, \tag{14}$$

$$\sum_{i=1}^{k} \sum_{e \in E(x)} z_{e,i} = d_G(r) \qquad \forall r \in R, \tag{15}$$

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

$$y_{x,i} \leq \sum_{e \in E(x)} z_{e,i} \qquad \forall x \in X,\ i = 1, \ldots, k, \tag{17}$$

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

$$\sum_{x \in X} y_{x,i} - 1 \leq \sum_{j=0}^{D-1} 2^j d_{j,i} \qquad i = 1, \ldots, k, \tag{19}$$

$$d_{j,i} \geq d_{j+1,i} \qquad j = 0, \ldots, D-2,\ i = 1, \ldots, k. \tag{20}$$

$$z_{e,i} = z_{e',i} \qquad \forall e, e' \in E \mid e \text{ and } e' \text{ are linked}, i = 1, \ldots, k. \tag{21}$$

where $d_G(s)$ is the degree of the vertex, s, and $E(s)$ represents the set of edges in E that are incident to the vertex, s.

The constraint (13) guarantees that P_i is a partial matching, that is, that each P_i has, at most, one edge incident to each row. The constraint (14) guarantees the k-partial-matchings are disjoint, that is, that one edge cannot belong to different partial matchings. In addition, all edges belong to a partial matching by constraint (15); thus, the feasible solutions are maximal k-partial-matchings. The coherence of $z_{e,i}$ and $y_{x,i}$ is guaranteed by (16) and (17). On the one hand, the constraint (16) ensures that an edge, $e \equiv (r, x)$, belongs to P_i (i.e., $z_{e,i} = 1$), only if the input $x \in X(P_i)$ (i.e., $y_{x,i} = 1$). On the other hand, (17) ensures that an input x belongs to $X(P_i)$ only if there exists at least one edge $e \equiv (r, x) \in P_i$.

The constraints (18), (19) and (20) are related to the calculation of the number of selection bits. The constraint (18) ensures that $d_{j,i} = 0$ for all j, such that $2^j \geq |X(P_i)| - 1$. The constraint (19) guarantees that the variable, $d_{j,i}$, corresponding to the more significant digit of the binary representation of $|X(P_i)| - 1$ has a value of 1. Moreover, all digits less significant than it will also be equal to 1 due to the constraint (20).

The constant, Q, is an upper bound of the selection cost, which is used in (12) to lexicographically sort the solutions. Finally, (21) ensures that any pair of linked edges belong to the same partial matching (i.e., that any pair of linked constants is located at the same ISM column).

The difference between the MMKPM-SSD and the MMKPM-SSL problems can be summarized as follows:

- The MMKPM-SSL problem does not have the minimization of the DCSI dispersion as an objective.
- The MMKPM-SSL problem prioritizes the minimization of the number of selection bits over the selection cost.
- The MMKPM-SSL problem includes constraint (21).
- In the MMKPM-SSL problem, constraint (15) is the result of rewriting the corresponding constraint in terms of a multigraph.

3.2.2. Greedy Algorithm for the ISS Stage

A greedy algorithm to solve the MMKPM problem was proposed in [16], which implements the following intuitive strategy to find good solutions. Given a bipartite graph, $G = (R \cup X, E)$, the algorithm constructs a k-partial-matching, $P = \{P_1, P_2, \ldots, P_k\}$, by selecting, at each step, the largest set of edges incident to the same vertex, $x \in X$, that can be added to some partial matching, P_i, and then adding it to P_i. The process terminates when the obtained k-partial-matching is maximal, that is, when all edges have been processed.

This greedy algorithm was used as an alternative to the ILP formulation in the experimental results of [15] to solve the MMKPM-SSD problem. However, the algorithm cannot be directly used to solve instances of the new MMKPM-SSL problem because such a greedy strategy does not guarantee that linked edges remain in the same partial matching.

To avoid this issue, we propose a slight modification of the algorithm that involves initializing each partial matching, $P_j \in P$, with the edges corresponding to the fixed DCSIs in the column, j, of the ISM. This initialization ensures that the algorithm does not subsequently assign linked edges to different partial matchings (i.e., it does not separate linked constants in different columns). More formally, given a bipartite multigraph, $G = (R \cup X, E, \gamma)$, instead of initializing P with empty sets, the new algorithm initializes each partial matching as $P_j = \{e \equiv (i, j) \in E : \gamma(e) = (i, 0) \vee \gamma(e) = (i, 1)\}$ for all $P_j \in P$.

4. Experimental Results

In this section, the proposed approach (which will be referred to as SG-ISS) is compared with that proposed in [15] (which will be referred to as ISS-SG). For each approach, both the ILP and the greedy strategies are used in the ISS stage, giving rise to the following techniques: SG-ISS-ILP, SG-ISS-Greedy, ISS-ILP-SG, and ISS-Greedy-SG.

The experimental study was carried out using a set of 72 large-sized FSMs, which were also used in [15] (Table 1 shows the statistical measures of the different parameters of these FSMs). The target device was an Intel Max 10 FPGA with 50K logic elements and a speed grade of 6, which includes 182 EMBs of 9 Kbits. The synthesis and implementation of the designs were conducted using Quartus Prime software (ver. 18.0). As the results were obtained after placement and routing, they include routing overhead. The speed (maximum clock frequency) and area results (EMB and LUT usage) were obtained using speed and area optimization, respectively. The efficiency of the FSMIM techniques can be measured by calculating the number of saved LUTs with respect to a conventional LUT-based approach for each used EMB (we will refer to this measure as SPLE (saved LUTs per EMB)). The ILP formulation is solved using Gurobi [21] with a maximum execution time of 60 min.

Table 1. Parameters of the FSMs used in the experiments.

	Mean	**Std**	**Min**	**Q1**	**Q2**	**Q3**	**Max**
# of states	185.1	143.2	35.0	67.5	142.5	253.0	497.0
# of inputs	12.5	1.7	10.0	11.0	12.5	14.0	15.0
# of outputs	4.6	1.4	2.0	4.0	4.0	6.0	7.0
# of transitions	2739.5	3604.5	198.0	622.5	1391.5	3424.5	22,950.0
Maximum number of effective inputs	8.0	0.8	7.0	7.0	8.0	9.0	9.0

Table 2 shows the EMB reduction, the LUT reduction, the SLPE increment, and the speed increment obtained by SG-ISS with respect to ISS-SG (obviously, each strategy is compared with its corresponding one, i.e., SG-ISS-ILP with ISS-ILP-SG as well as SG-ISS-Greedy with ISS-Greedy-SG). Therefore, the results shown in the table corresponding to the techniques that use the ILP strategy cannot be compared with those obtained using the greedy strategy because they were calculated using a different reference. In addition to the usual statistical measures, the table shows the Hit Rate (HR) and the Miss Rate (MR), which represent the percentage of cases in which SG-ISS obtains better and worse results than ISS-SG, respectively.

Table 2. Percentage improvement in the proposed SG-ISS approach with respect to ISS-SG.

	Arch.	**ISS Strat.**	**Mean**	**Std. Dev.**	**Min**	**Q1**	**Q2**	**Q3**	**Max**	**HR**	**MR**
EMB red. (%)	FSMIM-S	Greedy	31	23	0	7	33	50	77	81	0
		ILP	28	23	0	4	33	50	77	78	0
	FSMIM-T	Greedy	24	24	−20	0	32	43	69	72	22
		ILP	24	26	−30	0	33	46	73	71	17
LUT red. (%)	FSMIM-S	Greedy	−8	36	−153	−17	−1	9	64	47	53
		ILP	−18	48	−268	−27	−11	2	65	26	74
	FSMIM-T	Greedy	−204	195	−820	−300	−136	−80	52	6	94
		ILP	−138	164	−670	−215	−78	−33	63	7	93
SLPE inc. (%)	FSMIM-S	Greedy	55	53	−2	8	46	88	215	90	10
		ILP	46	44	−6	1	40	84	183	78	22
	FSMIM-T	Greedy	37	46	−18	−3	33	69	171	67	33
		ILP	42	53	−24	−0	32	81	264	71	29
Speed inc. (%)	FSMIM-S	Greedy	9	20	−29	−2	7	20	80	65	35
		ILP	5	16	−25	−4	1	15	48	56	44
	FSMIM-T	Greedy	−7	7	−24	−11	−7	−3	9	13	87
		ILP	−5	7	−25	−8	−5	−0	13	23	77

We must highlight that some cases (two for the ILP strategy and three for the greedy one) were excluded because the implementation obtained by ISS-SG did not fit in the device. Nevertheless, the FSMs were successfully implemented by means of the proposed SG-ISS approach.

The proposed approach achieves noteworthy reductions in EMB usage. The obtained average reduction is 24% in the FSMIM-T architecture and close to 30% in the FSMIM-S one. The approach is slightly less effective in the FSMIM-T architecture because the initial SG stage reduces the depth of the ROM at the expense of harming the subsequent minimization of the selection cost. This fact only affects the ROM of FSMIM-T, diminishing the overall reduction, as it requires storing the selection bits. Independently of the ISS strategy and the architecture, the hit rate is at least 71%. In FSMIM-S, the proposed approach never uses more EMBs than the old technique (in FSMIM-T, the miss rate does not exceed 22%).

The negative average LUT reduction indicates that, as would be expected, the number of used LUTs increases. The worst results are obtained by FSMIM-T due to the direct correlation between the complexity of the ISB (which is based on multiplexers) and the selection cost. However, despite the high increment ratios, it must be taken into account that the number of used LUTs is low in this architecture (the average number of LUTs per FSM is about 30). In the case of FSMIM-S, the lower average increment in used LUT confirms that the impact of the selection cost on the complexity of the ISB is lower. Despite the increase in LUT usage in both architectures, the SLPE increment keeps high values, indicating that the new approach is more effective as a means of saving LUTs by using EMBs than the ISS-SG approach (this is particularly true for the FSMIM-S architecture).

Regarding speed, the increase in LUTs required by FSMIM-T causes, as is expected, a degradation in the maximum clock frequency, although this is not significant (less than 7%). In FSMIM-S, the use of the ILP strategy increases the speed in 56% of the cases, obtaining an average increment of 5% (the results are even better for the greedy strategy). These good results are explained by the lower dependency between the selection cost and the ISB complexity in this architecture, along with the fact that the reduction in the number of EMBs can improve the speed. In contrast, in the case of FSMIM-T, the positive effect of the EMB reduction on speed does not allow for offsetting the speed degradation due to the increase in the number of LUTs.

It is important to highlight that the proposed SG-ISS approach is not intended to replace but rather to provide an alternative to the ISS-SG approach. The four techniques, along with the two FSMIM architectures, constitute eight design alternatives that can be chosen in accordance with the design requirements. In order to study the efficiency of these design alternatives, for each FSM, we have calculated the EMB reduction, LUT reduction, and speed increment in each design alternative with respect to the worst-case scenario (i.e., the alternative that obtains the worst result for the FSM and the studied measure). The obtained average values (in percentages) are shown in Figures 4–6. On the one hand, it is clear that the SG-ISS approach and the FSMIM-S architecture should be used when reducing memory usage is a priority and the speed is not critical. On the other hand, the ISS-SG approach and the FSMIM-T architecture should be used when saving LUTs is the primary objective. The faster and solver-free greedy approach provides a favorable trade-off between result quality and computation time. This is an adequate option when the design requirements are not very demanding; otherwise, the ILP strategy should be used.

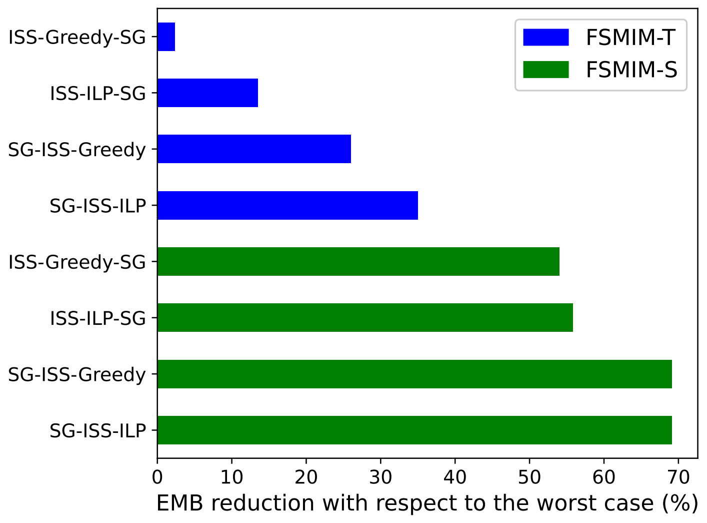


Figure 4. EMB reduction with respect to the worst case for each of the techniques.

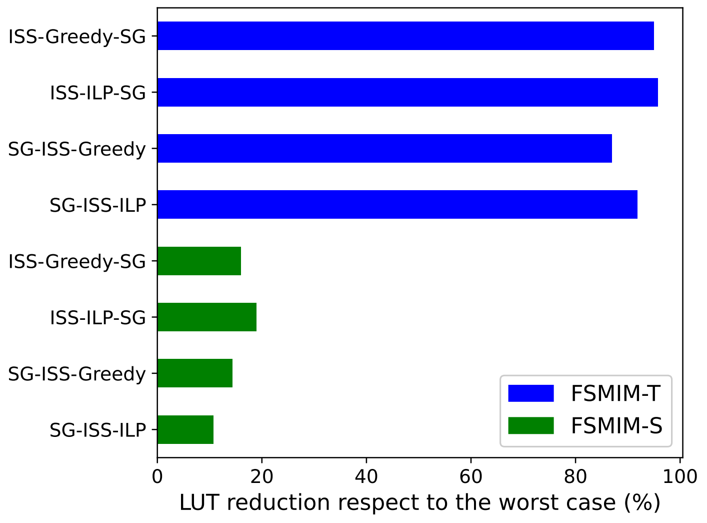


Figure 5. LUT reduction with respect to the worst case for each of the techniques.

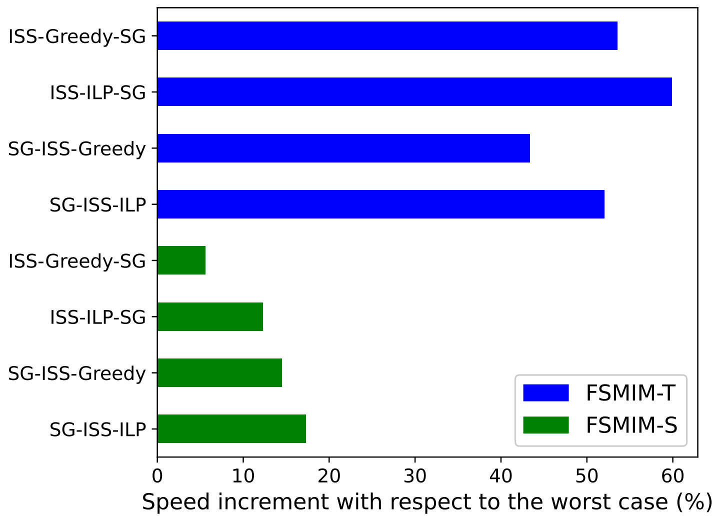


Figure 6. Speed increment with respect to the worst case for each of the techniques.

5. Conclusions

A new approach for generating FSMIMs from conventional FSMs, called SG-ISS, was proposed with the aim of obtaining further reductions in the number of required EMBs. Unlike previous approaches [10,15], the SG stage is applied before the ISS stage. For this, a new strategy for the SG stage that obtains optimal solutions in terms of grouping states was proposed. In addition, the ILP formulation proposed in [15] for the ISS stage was extended in order to maintain coherence with the results obtained in the SG stage. A comparison study between the new approach and that proposed in [15] was presented.

The results show that the SG-ISS significantly reduces the number of EMBs without a considerable increase in the number of LUTs. Thus, the proposed SG-ISS approach can be an alternative to that proposed in [15] (called the ISS-SG approach) when saving EMBs is critical. We can, therefore, conclude that SG-ISS and ISS-SG can be considered complementary approaches, which, along with the two ISS strategies (ILP and Greedy) and the two FSMIM architectures (FSMIM-T and FSMIM-S), provide eight different design alternatives. These alternatives were evaluated by analyzing the average EMB reduction, LUT reduction, and speed increment in each design alternative with respect to the worst case. From this study, we can conclude that the SG-ISS approach and the FSMIM-S architecture should be used when reducing memory usage is a priority and the speed is not critical. On the other hand, the ISS-SG approach and the FSMIM-T architecture should be used when saving LUTs is the primary objective.

With respect to the ISS strategy, the greedy algorithm provides a good trade-off between result quality and computation time (in addition to no solver being required). Therefore, it is a good choice when the design requirements are not very demanding. On the other hand, the ILP formulation could be used when the requirements are not met using the greedy algorithm. An Electronic Design Automation (EDA) tool incorporating the proposed design alternatives could select one of them, considering the design requirements and the effort level specified by the designer.

Author Contributions: Conceptualization, R.S.-N. and I.G.-V.; Methodology, R.S.-N. and I.G.-V.; Software, R.S.-N. and I.G.-V.; Validation, R.S.-N. and I.G.-V.; Formal analysis, R.S.-N. and I.G.-V.; Investigation, R.S.-N. and I.G.-V.; Writing—original draft, R.S.-N. and I.G.-V.; Writing—review & editing, R.S.-N. and I.G.-V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: We would like to thank Gurobi for providing us with a free academic license to carry out this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barkalov, A.; Titarenko, L. *Logic Synthesis for FSM-Based Control Units*; Springer International Publishing: Cham, Switzerland, 2009; Volume 53. [CrossRef]
2. Klimowicz, A.S.; Solov'ev, V.V. Structural Models of Finite-State Machines for Their Implementation on Programmable Logic Devices and Systems on Chip. *J. Comput. Syst. Sci. Int.* **2015**, *54*, 230–242. [CrossRef]
3. Kubica, M.; Kania, D. Technology Mapping of FSM Oriented to LUT-Based FPGA. *Appl. Sci.* **2020**, *10*, 3926. [CrossRef]
4. Rawski, M.; Selvaraj, H.; Luba, T. An application of functional decomposition in ROM-based FSM implementation in FPGA devices. In Proceedings of the Euromicro Symposium on Digital System Design, Belek-Antalya, Turkey, 3–5 September 2003; pp. 104–110.
5. El-Maleh, A.; Sait, S.; Nawaz Khan, F. Finite state machine state assignment for area and power minimization. In Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006, Kos, Greece, 21–24 May 2006.
6. Janarthanan, A.; Tiwari, A.; Tomko, K. Power-efficient FSM mapping in FPGAs through SEMB dormancy control. In Proceedings of the 50th Midwest Symposium on Circuits and Systems, MWSCAS 2007, Montreal, QC, Canada, 5–8 August 2007; pp. 502–505.
7. Mengibar, L.; Entrena, L.; Lorenz, M.; Millan, E. Partitioned state encoding for low power in FPGAs. *Electron. Lett.* **2005**, *41*, 948–949. [CrossRef]
8. Sklyarov, V. Reconfigurable models of finite state machines and their implementation in FPGAs. *J. Syst. Archit.* **2002**, *47*, 1043–1064. [CrossRef]

9. Borowik, G.; Falkowski, B.; Luba, T. Cost-Efficient Synthesis for Sequential Circuits Implemented Using Embedded Memory Blocks of FPGA's. In Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems, DDECS'07, Kraków, Poland, 11–13 April 2007; pp. 1–6.
10. Garcia-Vargas, I.; Senhadji-Navarro, R. Finite State Machines With Input Multiplexing: A Performance Study. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2015**, *34*, 867–871. [CrossRef]
11. Barkalov, A.; Titarenko, L.; Kolopienczyk, M.; Mielcarek, K.; Bazydlo, G. *Design of EMB-Based Mealy FSMs*; Springer International Publishing: Cham, Switzerland, 2016; pp. 193–237.
12. Tiwari, A.; Tomko, K. Saving power by mapping finite-state machines into embedded memory blocks in FPGAs. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Paris, France, 16–20 February 2004; Volume 2, pp. 916–921.
13. Selvaraj, H.; Rawski, M.; Luba, T. FSM implementation in embedded memory blocks of programmable logic devices using functional decomposition. In Proceedings of the International Conference on Information Technology: Coding and Computing, Las Vegas, NV, USA, 8–10 April 2002; pp. 355–360. [CrossRef]
14. Barkalov, A.; Titarenko, L.; Krzywicki, K. Structural Decomposition in FSM Design: Roots, Evolution, Current State—A Review. *Electronics* **2021**, *10*, 1174. [CrossRef]
15. Garcia-Vargas, I.; Senhadji-Navarro, R. Optimization based on the minimum maximal k-partial-matching problem of finite states machines with input multiplexing. *Des. Autom. Embed. Syst.* **2022**, *26*, 83–103. [CrossRef]
16. Garcia-Vargas, I.; Senhadji-Navarro, R. The minimum maximal k-partial-matching problem. *Optim. Lett.* **2013**, *7*, 1959–1968. [CrossRef]
17. Das, N.; Priya, P.A. FPGA Implementation of an Improved Reconfigurable FSMIM Architecture Using Logarithmic Barrier Function Based Gradient Descent Approach. *Int. J. Reconfigurable Comput.* **2019**, *2019*, 3727254. [CrossRef]
18. Mardani Kamali, H.; Zamiri Azar, K.; Homayoun, H.; Sasan, A. SCRAMBLE: The State, Connectivity and Routing Augmentation Model for Building Logic Encryption. In Proceedings of the 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Limassol, Cyprus, 6–8 July 2020; pp. 153–159.
19. Xilinx. *7 Series FPGAs Configurable Logic Block: User Guide*; Xilinx: San Jose, CA, USA , 2016.
20. Altera. *Advanced Synthesis Cookbook*; Altera: San Jose, CA, USA, 2011.
21. Gurobi Optimization. Gurobi Optimizer Reference Manual. 2020. Available online: http://www.gurobi.com (accessed on 1 September 2023).