Research Article

FPGA Implementation of Reconfigurable Finite State Machine with Input Multiplexing Architecture Using Hungarian Method

Nitish Das and P. Aruna Priya

Department of ECE, SRM University, Kattankulathur, Chennai 603203, India

Correspondence should be addressed to P. Aruna Priya; arunapriya.p@ktr.srmuniv.ac.in

Received 25 July 2017; Revised 27 October 2017; Accepted 16 November 2017; Published 10 January 2018

1. Introduction

Designing a complex digital system requires an efficient method that includes modeling a control unit (i.e., a controller). The operational speed of such systems depends on the speed of their controllers. The mathematical model for designing a controller for applications such as microprocessor control units, circuit testing, and digital signal processing (DSP) is a finite state machine (FSM). Consequently, designing such systems requires an efficient synthesis technique for high-speed FSM [1, 2]. Applications such as DSP [3, 4] and built-in self-test (BIST) [5] require specific operations to be performed only in the particular instances. Different control units are required to complete each operation. Hence, to optimally perform these operations, a single control unit is defined which can configure itself depending upon the applied mode of operation; it is also known as reconfigurable FSM [1]. The mode of operation for such FSM is controlled by a counter, timer, or any user-defined control signals based on the application requirements. An example of a reconfigurable FSM is given in [1] as a test chip for wireless sensor network. In this example, Transition-Based Reconfigurable FSM (TR-FSM) [1] is configured into one of the MCNC FSM benchmark circuits (i.e., dk15, s386, or cse) at different instances. Moreover, any application which requires sequential processing can be broken down into a series of instances (i.e., multistage reconfigurable signal processing) where each instance only a particular operation is performed [3]. Hence, for such applications, efficient architectures can be created using reconfigurable FSM. These emerging trends in the research necessitate a framework for optimal synthesis of high-speed reconfigurable FSM.

Conventional LUT-based architectures have been used for FSM implementation on a FPGA platform [6]. Similarly, ROM-based architectures are investigated for FSM implementations. Due to the area and speed advantages, they act as an excellent alternative to their conventional LUT-based counterparts [7]. In such implementations, a considerable reduction in power consumption is obtained by disabling embedded memory blocks (EMBs) during the idle
states [8, 9]. The fundamental framework for FSM with input multiplexing (FSMIM) is made in [7] whose prime objective is to shorten the depth of ROM memory. In their approach, an input selector (which consists of a multiplexer bank) is used. The basic idea that has been implemented is to select only a specific set of inputs for a particular state. FSMIM with state-based input selection (FSMIM-S) is proposed in [10], which further reduces the ROM memory size.

Another approach for implementation of reconfigurable FSM is RAM-based architectures. In literature, there are two underlying RAM-based architectures, that is, variation-based reconfigurable multiplexer bank (VRMUX) and combination-based reconfigurable multiplexer bank (CRMUX) [11]. The RAM-based architectures do not serve as a novel tool for implementation of complicated FSM structures such as parallel hierarchical finite state machines (PHFSM) [12] or reversible FSM [13]. Due to significant advantages of FSMIM-S architecture over other architectures, it is used to create a framework for the high-speed Reconfigurable FSMIM-S architecture.

The Reconfigurable FSMIM-S architecture is constructed by combining the conventional FSMIM-S architecture [10] and an optimized multiplexer bank (which defines the mode of operation). For this, the descriptions of a set of FSMs are taken for a particular application. Hence, the problem is to obtain the optimized multiplexer bank for the given set of FSMs. It can be solved by mapping all the FSMs into one large FSM (called baseckt) in that set. The objective of this process is to perform optimal matching between baseckt and the other FSMs in the set so that a minimum number of bits are changed by changing the mode of operation. This situation (i.e., performing one-to-one mapping) transforms the problem into a weighted bipartite graph matching problem where the objective is to match the description of FSMs in the set to baseckt with minimal cost [14]. As a solution, an iterative greedy heuristic based Hungarian algorithm is proposed. In this algorithm, the weights are assigned based on the input combinations, state code, and the output combinations to form a cost matrix. A cost matrix reduction based technique, that is, Hungarian algorithm [15, 16], is used for matching. A greedy based heuristic (GBH) search technique [17] is combined with the Hungarian algorithm to optimize the augmenting path search. At every iteration, descriptions of two FSMs (i.e., baseckt and one of the FSMs in the set) are taken as inputs. It produces the modified descriptions of the FSMs of the same dimension as outputs. At the end of the algorithm, a mutual XOR operation is performed among the modified descriptions, which provides the required optimized multiplexer bank.

The experimental results from MCNC FSM benchmarks illustrate the advantages of the proposed architecture as compared with VRMUX [11], as operating speed is enhanced at an average of 30.43% and LUT consumption is reduced by an average of 5.16% in FPGA implementation. It also shows that the operating speed is improved at an average of 9.14% in comparison with CRMUX [11] during FPGA implementation. The limitation of the proposed technique is the requirement of higher LUTs, as it requires an average of 88.65% more LUTs in comparison with CRMUX [11] during FPGA implementation.

The rest of the paper is outlined as follows. Section 2 consists of the Reconfigurable FSMIM-S architecture and the proposed iterative greedy heuristic based Hungarian algorithm. The experimental evaluation of the proposed algorithm, implementation of the Reconfigurable FSMIM-S architecture, and comparison with other proposals from the literature are presented in Section 3. The concluding remarks are devised in Section 4.

2. Proposed Method

As most of the FPGA platforms use synchronous EMBs, Mealy machines with synchronous outputs are used in this paper. Let a Mealy FSM be described by the following columns: \( a_{m} \) is a code of current state \( (a_{m} \in A) \), \( A = \{a_{1}, \ldots , a_{M}\} \) is a set of states; \( K(a_{m}) \) is a code of state \( a_{m} \in A; h \) is the number of transitions per state \( (h \in H) \), \( H = \{t_{1}, \ldots , t_{M}\} \) is a set of number of transitions per state corresponding to \( A; a_{1} \) is a state of transition (the next state); \( K(a_{1}) \) is a code of state \( a_{1} \in A; X = \{x_{1}, \ldots , x_{L}\} \) is the set of input variables, \( Y = \{y_{1}, \ldots , y_{N}\} \) is the set of output variables; and \( D = \{d_{1}, \ldots , d_{R}\} \) is defined as excitation functions for the flip-flops, where \( R \) is the number of flip-flops (i.e., the number of bits in internal state codes), \( R \in \left\{ \log_{2}M, J \right\} \).

The descriptions of a set of FSMs are taken for a particular application. The fundamental idea is to obtain the description of a single FSM by mapping all the FSMs into one large FSM (called baseckt) in that set. The objective of this process is to perform optimal matching between baseckt and the other FSMs in the set so that a minimum number of bits are changed by changing the mode of operation. This situation (i.e., performing one-to-one mapping) transforms the problem into a weighted bipartite graph matching problem where the objective is to match the description of FSMs in the set to baseckt with minimal cost [14]. As a solution, an iterative greedy heuristic based Hungarian algorithm is proposed. In this algorithm, the weights are assigned based on the input combinations, state code, and the output combinations to form a cost matrix. A cost matrix reduction based technique, that is, Hungarian algorithm [15, 16], is used for matching. A greedy based heuristic (GBH) search technique [17] is combined with the Hungarian algorithm to optimize the augmenting path search. At every iteration, descriptions of two FSMs (i.e., baseckt and one of the FSMs in the set) are taken as inputs. It produces the modified descriptions of the FSMs of the same dimension as outputs. At the end of the algorithm, a mutual XOR operation is performed among the modified descriptions, which provides the required optimized multiplexer bank.

The experimental results from MCNC FSM benchmarks illustrate the advantages of the proposed architecture as compared with VRMUX [11], as operating speed is enhanced at an average of 30.43% and LUT consumption is reduced by an average of 5.16% in FPGA implementation. It also shows that the operating speed is improved at an average of 9.14% in comparison with CRMUX [11] during FPGA implementation. The limitation of the proposed technique is the requirement of higher LUTs, as it requires an average of 88.65% more LUTs in comparison with CRMUX [11] during FPGA implementation.

The rest of the paper is outlined as follows. Section 2 consists of the Reconfigurable FSMIM-S architecture and the proposed iterative greedy heuristic based Hungarian algorithm. The experimental evaluation of the proposed algorithm, implementation of the Reconfigurable FSMIM-S architecture, and comparison with other proposals from the literature are presented in Section 3. The concluding remarks are devised in Section 4.

2. Proposed Method

As most of the FPGA platforms use synchronous EMBs, Mealy machines with synchronous outputs are used in this paper. Let a Mealy FSM be described by the following columns: \( a_{m} \) is a code of current state \( (a_{m} \in A) \), \( A = \{a_{1}, \ldots , a_{M}\} \) is a set of states; \( K(a_{m}) \) is a code of state \( a_{m} \in A; h \) is the number of transitions per state \( (h \in H) \), \( H = \{t_{1}, \ldots , t_{M}\} \) is a set of number of transitions per state corresponding to \( A; a_{1} \) is a state of transition (the next state); \( K(a_{1}) \) is a code of state \( a_{1} \in A; X = \{x_{1}, \ldots , x_{L}\} \) is the set of input variables, \( Y = \{y_{1}, \ldots , y_{N}\} \) is the set of output variables; and \( D = \{d_{1}, \ldots , d_{R}\} \) is defined as excitation functions for the flip-flops, where \( R \) is the number of flip-flops (i.e., the number of bits in internal state codes), \( R \in \left\{ \log_{2}M, J \right\} \).

The descriptions of a set of FSMs are taken for a particular application. The fundamental idea is to obtain the description of a single FSM by mapping all the FSMs into one large FSM (called baseckt) in that set. The inputs, states, and outputs of an FSM in the set are mapped into baseckt in their respective order. The mode bits are applied through a 2 x 1 multiplexer in those positions where the polarity of bit differs (i.e., 1 in place 0 and vice versa) to perform such mapping. Hence, the resultant FSM operates in two modes, where baseckt mode is the default mode of operation. Similarly, all other FSMs in the set are mapped into baseckt. In this way, a single FSM (i.e., baseckt) combined with a multiplexer bank (which defines the mode of operation) acts as reconfigurable FSM. It can be configured into a particular FSM in the set by applying the specific mode bits. Due to numerous advantages mentioned in the literature, FSMIM-S architecture [10] is chosen to implement the FSM (i.e., baseckt) part. Therefore, the Reconfigurable FSMIM-S architecture is constructed by combining the conventional FSMIM-S architecture [10] and multiplexer bank for mode based reconfiguration as shown in Figure 1.

It encounters the following two major difficulties:

(i) The complexity of the resultant multiplexer bank is very high.

(ii) It becomes difficult to define the dummy states and dummy transitions. Dummy states and dummy transitions are such states and transitions which are not present in baseckt but exist in the other FSMs in the set and vice versa. These states and transitions lead the system to failure.
As a solution, an iterative greedy heuristic based Hungarian algorithm is proposed. In this algorithm, the descriptions of a set of FSMs (i.e., \([h, X, a_m, a_s, Y]\)) are taken as inputs. It provides the optimized multiplexer bank for mode based reconfiguration as output. It also provides the updated description (i.e., description without dummy states and dummy transitions) of base_ckt, which is used to construct the conventional FSMIM-S part of the proposed architecture. Let \((B + 1)\) be the set of FSMs for a particular application. Based on the complexity of the description of FSM, the largest FSM is selected from the set. It is called base_ckt. The rest of the FSMs are called recon_ckt_1, recon_ckt_2,..., recon_ckt_B, respectively.

Each input, state, or output of a recon_ckt_b \(\in\) \{recon_ckt_1, recon_ckt_2,..., recon_ckt_B\} can be mapped into any one of the inputs, states, or outputs, respectively, of base_ckt; that is, there exists a one-to-one mapping. These mappings cannot be performed independently because inputs, states, and outputs of an FSM are interdependent. Consequently, mapping an input or state of recon_ckt_b into base_ckt is transformed into a weighted bipartite graph matching problem or linear assignment problem (LAP) [14] as shown in Figure 2. In this LAP, the weights are assigned based on the input combinations, state code, and the output combinations to form a cost matrix. The objective of this process is to perform matching with a minimal cost so that a minimum number of bits are changed by changing the mode of operation. Therefore, the complexity of the multiplexer bank is reduced.

In the literature, the following approaches are proposed to solve a LAP:

(i) Modified Hungarian algorithm [16]
(ii) Simple greedy heuristic based algorithm [17]
(iii) Evolutionary heuristic algorithm [18].

The maximum number of inputs or states does not exceed 100 in MCNC FSM benchmarks or FSMs used in real-world applications. So, the number of vertices used in the resultant weighted bipartite graph is always low which results in small LAP. But, the number of LAPs formed in this process is enormous because input matching and state matching are performed together as shown in Figure 2. Hence, the primary requirement of the algorithm to solve LAP becomes the fast convergence. Therefore, a cost matrix reduction based technique, that is, Hungarian algorithm [15, 16], is used for matching. A greedy based heuristic (GBH) search technique [17] is combined with the Hungarian algorithm to optimize the augmenting path search. The pseudocode of this technique is summarized in Algorithm 1. (Note: subscripts “base” and “recon” denote the parameters of base_ckt and recon_ckt, respectively, throughout the paper.)

At every iteration \(\in\) \{1,...,B\}, descriptions of two FSMs, that is, base_ckt and recon_ckt_b, are taken as inputs. The major contributing factors for power consumption and LUT requirement in FSM are the number of inputs and the internal states [8, 19]. In any FSM, input variable and states are interdependent. Thus, input and state matching are performed together between base_ckt and recon_ckt.

If \(L_{base} \geq L_{recon}\), then \(E = L_{base} p_{L,base}\) combinations of input lines for base_ckt are generated to match with input lines of recon_ckt_b. \((L_{base} – L_{recon})\) input lines act as don’t cares while the system operates in recon_ckt_b mode. Otherwise, \(E = L_{recon} p_{L,recon}\) combinations of input lines for recon_ckt_b are generated to match with input lines of base_ckt. In this case, \((L_{recon} – L_{base})\) input lines act as don’t cares while the system operates in base_ckt mode.

Now, for each combination of input lines, state matching is performed (Algorithm 2). This situation can be seen as a LAP where the objective is to match the states of recon_ckt_b to the states of base_ckt with minimal cost [14, 17]. For this,
the number of states in both the FSMs is equalized. Thus, if \( M_{\text{base}} \geq M_{\text{recon}} \), then \( (\alpha - M_{\text{recon}} \) where \( \alpha = M_{\text{base}} \)) dummy states are added in recon_ckt.b. Otherwise \( (\alpha - M_{\text{base}} \), where \( \alpha = M_{\text{recon}} \)) dummy states are added in base_ckt.

All LAP solving algorithms require a cost matrix as an input to perform an optimal assignment. So, to form a cost matrix for this problem, a procedure named weight_assignment is proposed.

In this procedure, the combinations of input lines, \( a_m \) and \( h \), for base_ckt and recon_ckt.b are taken as inputs. It provides the cost matrix to map recon_ckt.b states into base_ckt states. An array is created at each transition in both base_ckt and recon_ckt.b by combining \([\text{input\_combination} \in \{x_1, x_2, \ldots, x_L\}, a_m]\).

The basic idea that has been implemented is as follows: (i) replace the recon_ckt.b state with the base_ckt state sequentially in the recon_ckt_array; (ii) evaluate the weight by performing Bitwise-XOR operation (i.e., transition matching) for that particular replacement; (iii) then, construct the cost matrix.

For each transition in recon_ckt_array (i.e., \( h_{\text{recon}} \in \{1, 2, \ldots, t_{m_{\text{recon}}}\} \)), transition matching is performed. This situation can be seen as a LAP where the objective is to match the transition of recon_ckt.b to the transition of base_ckt with minimal cost \([14, 17]\). For this, the number of transitions for the particular state is equalized in both the FSMs. Therefore, if \( t_{m_{\text{base}}} \geq t_{m_{\text{recon}}} \), then \( (\beta - t_{m_{\text{recon}}} \) where \( \beta = t_{m_{\text{base}}} \)) dummy transitions are added in the recon_ckt_array. Otherwise \( (\beta - t_{m_{\text{base}}} \), where \( \beta = t_{m_{\text{base}}} \)) dummy transitions are added in the base_ckt_array. Thus, for each transition in base_ckt_array (i.e., \( h_{\text{base}} \in \{1, 2, \ldots, t_{m_{\text{base}}}\} \)), a Bitwise-XOR operation is performed between the arrays for that particular transition. The total number of 1s in the Bitwise-XOR operations is counted to create a cost matrix for transition matching. Then, optimal assignment of transitions.
Input. The descriptions of the FSMS (i.e., $[h, X, a_m, a_n, Y]$)

Output. The optimized multiplexer bank for mode based reconfiguration

begin

select the largest FSM from the set based on the description;

circuit $\text{ckt} \leftarrow$ largest FSM;

circuit reconckt_1, reconckt_2, ..., reconckt_B \leftarrow$ rest of the FSMS in the set;

for each circuit reconckt_b $\in$ [reconckt_1, reconckt_2, ..., reconckt_B] do

if $(L_{\text{base}} \geq L_{\text{recon}})$ then /* performing the input matching */

generate, $E \leftarrow L_{\text{base}}$ combinations of input lines for circuit $\text{ckt}$ to match with input lines of reconckt_b;

go to state matching; /* calling the function- "state matching" */

else if $(L_{\text{base}} < L_{\text{recon}})$ then

generate, $E \leftarrow L_{\text{recon}}$ combinations of input lines for reconckt_b to match with input lines of base_ckt;

go to state matching; /* calling the function- "state matching" */

select combinations of input lines with min$\{\text{assignment cost}_1, \ldots, \text{assignment cost}_n\}$
& min$\{\text{total cost}_1, \ldots, \text{total cost}_n\}$;

perform binary state assignment in circuit $\text{ckt}$ & reconckt_b i.e. apply $K(a_m)$ & $K(a_n)$;

weight assignment(); /* creating arrays by [selected input combination, $K(a_m), K(a_n)$]*/

go to dummy replacement; /* calling the function- "dummy replacement" */

if $(N_{\text{base}} \geq N_{\text{recon}})$ then /* performing the output matching */

generate, $G \leftarrow N_{\text{base}}$ combinations of output lines for circuit $\text{ckt}$ to match with output lines of reconckt_b;

go to output matching; /* calling the function- "output matching" */

else if $(N_{\text{base}} < N_{\text{recon}})$ then

generate, $G \leftarrow N_{\text{recon}}$ combinations of output lines for reconckt_b to match with output lines of base_ckt;

go to output matching; /* calling the function- "output matching" */

select combinations of output lines with min$\{\text{XOR count}_1, \ldots, \text{XOR count}_n\}$;

update the description of base_ckt;

end

for each circuit reconckt_b $\in$ [reconckt_1, reconckt_2, ..., reconckt_f(B - 1)] do

go to dummy replacement; /* calling the function- "dummy replacement" */

update the description of reconckt_b;

end

perform a mutual (i.e. $\frac{N}{2}$) Bitwise-XOR operations between the updated descriptions of FSMS;

obtain the optimized multiplexer bank for mode based reconfiguration;

end

Algorithm 1: Iterative greedy heuristic based Hungarian algorithm.

is performed by greedy based heuristic Hungarian algorithm (GBH_hungarian_algorithm) between base_ckt_array and reconcka_ckt_array. Let match_count be a variable defined as

\[
\text{match} \_\text{count} = \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij} \lambda_{ij}, \quad (1)
\]

where, $C_{ij} \leftarrow$ cost matrix, $\lambda_{ij} \leftarrow$ decision variable.

In this way, by using match_count (from (1)), the cost matrix formation to map reconckt_b states into base_ckt states is completed. The pseudocode of the procedure, weight_assignment, is summarized in Algorithm 5.

Let $V$ and $U$ represent the set of vertices (i.e., transitions or states) for reconckt and base_ckt, respectively. $\mu = (V \cup U, \xi)$ is defined as a balanced weighted bipartite graph, where $|V| = |U| = n$. $C$ is the cost matrix. A number $C_{ij} \geq 0$ for each edge $[i, j] \in \xi$ is called the cost (or weight) of the edge $[i, j]$. In GBH_hungarian_algorithm, the cost matrix $C$ is taken as input. It provides an optimal assignment between $V$ and $U$ as output. GBH in [17] is an iterative cost matrix reduction based approach to solve the LAP. At each iteration, a single vertex is eliminated from either $V$ or $U$ until the advent of some stopping conditions. Let $k$ be the last iteration (whereas $k$ is a positive integer). Therefore, either $k$ or $(k - 1)$ vertices are eliminated from $\mu$ at the last iteration.

Let $V_k \subseteq V$ and $U_k \subseteq U$ be the subsets of the remaining vertices in $V$ and $U$, respectively, at iteration $k$. At the first iteration, that is, $k = 1$, $V_1 = V$, and $U_1 = U$, respectively, the objective of the LAP is to assign $n$ resources to $n$ tasks in such a way that optimal total cost should be obtained for the
assignment. The LAP can be mathematically formulated as follows:

\[ f_k = \min \sum_{i=1}^{\eta} \sum_{j=1}^{\eta} C_{ij} \lambda_{ij} \]

s.t. \[ \sum_{j=1}^{\eta} \lambda_{ij} = 1, \]

\[ \sum_{i=1}^{\eta} \lambda_{ij} = 1, \]

\[ \lambda_{ij} \in \{0, 1\}, \]

\[ \forall i, j = 1, \ldots, \eta. \]

Equation (2) represents the objective function for LAP. If resource \( i \) is allocated to task \( j \) then the decision variable \( \lambda_{ij} = 1 \) and 0 otherwise as depicted in (5). One-to-one mapping should be practiced between resources and tasks. Equations (3) and (4) ensure these criteria.

At each iteration, there are two options to eliminate a vertex, that is, from either \( V \) or \( U \). For each \( i \in V_k \) and \( j \in U_k \), the following parameters are defined to select one of the above options:

\[ C^{V}_{ik} = \frac{1}{|U_k|} \cdot \sum_{j \in U_k} \sum_{j \in V_k} C_{ij}, \]

\[ C^{U}_{ik} = \frac{1}{|V_k|} \cdot \sum_{i \in V_k} \sum_{j \in U_k} C_{ij}; \]

\[ f^{V}_{k} = \min_{i \in V_k} f(v_k, u_k); \]

\[ f^{U}_{k} = \min_{j \in U_k} f(v_k, u_k). \]

In (6), \( C^{V}_{ik} \) and \( C^{U}_{ik} \) can act as “potential cost contribution” [17] of vertices \( i \in V_k \) and \( j \in U_k \) to \( f_k \) in (2). Thus, the potential cost contribution is evaluated for the vertices, and if it exceeds the corresponding removal cost, then such vertices are eliminated.

If \( C^{V}_{ik} \leq C^{U}_{ik} \), then an attempt is made to remove one of the vertices from \( V_k \) to \( V \). From (7), if \( f^{V}_{k} \leq f^{U}_{k-1} \), that is, the objective function value is improved by eliminating \( i_k \), then \( V_k \) is set to \( V \) and the next iteration is executed.

Otherwise, one of the vertices from \( U_k \) is eliminated. From (8), if \( f^{V}_{k} \leq f^{U}_{k-1} \), that is, the objective function value is improved by eliminating \( j_k \), then \( U_k \) is set to \( U \) and the next iteration is executed.

In this case, when the objective function value is not improved by eliminating either \( i_k \) or \( j_k \), then algorithm halts and the obtained solution is \( f_{k-1} \). Furthermore, if \( C^{V}_{ik} > C^{U}_{ik} \), then the above steps are repeated in the opposite order. The pseudocode of this approach is devised in Algorithm 6.

Therefore, after obtaining the cost matrix from weight_assignment for state matching, GBH_hungarian_algorithm is applied to obtain the following parameters:

\[ \text{assignment_cost}_{e} = \sum_{i=1}^{\alpha} \sum_{j=1}^{\alpha} C_{ij} \cdot \lambda_{ij}, \]

where \( e \in \{1, 2, \ldots, E\}; \)

\[ \text{total_cost}_{e} = \sum_{i=1}^{\alpha} \sum_{j=1}^{\alpha} C_{ij}, \]

where \( e \in \{1, 2, \ldots, E\}. \)
Thus, all the recon\_ckt\_b states are replaced by their assigned base\_ckt states, and all the complete arrays of recon\_ckt\_b are arranged corresponding to \(a_{m,\text{base}}\) order. Hence, from (9), the combinations of input lines are selected with \(\min\{{\text{assignment\_cost}_1, \ldots, \text{assignment\_cost}_g}\} \) & \(\min\{{\text{total\_cost}_1, \ldots, \text{total\_cost}_p}\}\).

Now, binary state codes \(K(a_{m,\text{base}})\) and \(K(a_i)\) are applied in base\_ckt and recon\_ckt\_b. As it changes the weights of cost matrix, weight assignment is again applied to construct a modified cost matrix. In this case, arrays are created by combining \({\text{selected\_input\_combination}}, K(a_i), K(a_{m,\text{base}})\). Dummy states are replaced in matched states of base\_ckt and recon\_ckt\_b by using Propositions 1 and 2. Then, dummy transitions are replaced by using Proposition 1. The dummy replacement algorithm is shown in Algorithm 3.

**Proposition 1.** Dummy transitions in a matched state of base\_ckt or recon\_ckt\_b should be replaced with one of the existing transitions in that particular state with a minimum cost.

**Proof.** For each matched state (or assigned state after matching) \(\in\) recon\_ckt\_b, if \(t_{m,\text{base}} \geq t_{m,\text{recon}}\) then \(t_{m,\text{base}} - t_{m,\text{recon}}\) dummy transitions are present in recon\_ckt\_b state.

Hence, there are \((t_{m,\text{base}} - t_{m,\text{recon}})\) transitions, present in the corresponding state of base\_ckt which are unassigned. These unassigned transitions in base\_ckt will lead the system to failure while operating in recon\_ckt\_b mode. As a solution, these unassigned transitions of base\_ckt are assigned to the existing transitions of recon\_ckt\_b with the least cost by looking at the particular column of the modified cost matrix.

Similarly, for each matched state (or assigned state after matching) \(\in\) recon\_ckt\_b, if \(t_{m,\text{base}} < t_{m,\text{recon}}\) then \((t_{m,\text{recon}} - t_{m,\text{base}})\) dummy transitions are present in base\_ckt state. Hence, there are \((t_{m,\text{recon}} - t_{m,\text{base}})\) transitions, present in the corresponding state of recon\_ckt\_b which are unassigned. These unassigned transitions in recon\_ckt\_b will lead the system to failure while operating in base\_ckt mode. As a solution, these unassigned transitions of recon\_ckt\_b are assigned to the existing transitions of base\_ckt with the least cost by looking at the particular row of the modified cost matrix.

Let \(M_{C_{ij}}\) represent the modified cost matrix for a matched state, where rows \(U_i\) and columns \(V_j\) denote the base\_ckt and recon\_ckt\_b transitions, respectively. Thus, the unassigned transitions in base\_ckt state can be assigned by (10) as follows:

\[
\text{unassigned}_U_i \rightarrow V_j: \min \left( M_{C_{1,ij}}, M_{C_{2,ij}}, M_{C_{3,ij}}, \ldots, M_{C_{ij}} \right). \tag{10}
\]

Similarly, the unassigned transitions in recon\_ckt\_b state can be assigned by (11) as follows:

\[
\text{unassigned}_V_j \rightarrow U_i: \min \left( M_{C_{i1}}, M_{C_{i2}}, M_{C_{i3}}, \ldots, M_{C_{ij}} \right). \tag{11}
\]
Proposition 2. If \( M_{\text{base}} < M_{\text{recon}} \), then dummy states are replaced by splitting the matched state in base\_ckt.

Proof. In FSM, splitting a state with high transitions results in low power consumption [8, 19]. It also improves the operating speed [2, 20]. If \( M_{\text{base}} > M_{\text{recon}} \), then there are \( (M_{\text{base}} - M_{\text{recon}}) \) states, present in base\_ckt which are unassigned. These unassigned states in base\_ckt will lead to failure in the system while operating in recon\_ckt\_b mode. As base\_ckt is the largest FSM in the collection and its transitions per state are greater than recon\_ckt\_b, splitting recon\_ckt\_b states are insignificant for the system performance. So, these unassigned states of base\_ckt are assigned using Proposition 1. \( \square \)

If \( M_{\text{base}} < M_{\text{recon}} \), then \( (M_{\text{recon}} - M_{\text{base}}) \) dummy states are replaced by splitting the matched state in base\_ckt. Let \( \Psi(\alpha_{\text{mbase}}) = Q(\alpha_{\text{mbase}} - \alpha_{\text{mrecon}}) \), where \( Q \) is a positive integer. Only the states for which \( |\Psi(\alpha_{\text{mbase}})| > 1 \) can be split [19]. Each state can be split into nonoverlapping subsets of \( (\alpha_{\text{mbase}} - \alpha_{\text{mrecon}}) \) transitions. Algorithm 7 is proposed to split a base\_ckt state.

At this stage, the states and the input lines of both the FSMs are completely matched and fixed. Hence, the output matching is performed by performing a Bitwise-XOR operation and selecting the combination with the least count of 1’s. If \( N_{\text{base}} \geq N_{\text{recon}} \), then \( G = N_{\text{recon}} P_{N_{\text{base}}} \), combinations of output lines for base\_ckt are generated to match with output lines of recon\_ckt\_b. Otherwise, \( G = N_{\text{recon}} P_{N_{\text{base}}} \) combinations of output lines for recon\_ckt\_b are generated to match with output lines of base\_ckt. Then, for each combination of output lines, Bitwise-XOR operation is performed between corresponding output lines of base\_ckt and recon\_ckt\_b. Let XOR\_count\_g, where \( g \in \{1, 2, \ldots, G\} \) represents the total number of 1’s in the Bitwise-XOR operation for a particular combination of output lines. Therefore, the combinations of output lines with \( \min \{\text{XOR\_count}_1, \ldots, \text{XOR\_count}_G\} \) are selected.

At the end of every iteration, the description of base\_ckt is updated to operate on the next iteration. At the end of 8th iteration, for each recon\_ckt\_b \( \in \{\text{recon\_ckt}_1, \text{recon\_ckt}_2, \ldots, \text{recon\_ckt}_{(B - 1)}\} \), replacement of dummy transitions and states is performed and updated descriptions of recon\_ckt\_1, recon\_ckt\_2, ..., recon\_ckt\_(B - 1) are obtained. In this way, descriptions of all FSMs are optimally matched, having the same dimension. Therefore, a mutual (i.e., \( \frac{G}{C_2} \)) Bitwise-XOR operation between the updated descriptions of FSMs is conducted which provides the optimized multiplexer bank for mode based reconfiguration.

3. Experimental Evaluation

Experiments have been conducted to illustrate the advantages of the proposed architecture using the FSM benchmark circuits from MCNC/LGSynth [21] as shown in Table 1.

<table>
<thead>
<tr>
<th>Benchmark circuits</th>
<th>Number of states</th>
<th>Number of inputs</th>
<th>Number of outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>si494</td>
<td>48</td>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td>sand</td>
<td>32</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>styr</td>
<td>30</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>planet</td>
<td>48</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>s832</td>
<td>25</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>cse</td>
<td>16</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>s386</td>
<td>13</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>ex6</td>
<td>8</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>mc</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>planetl</td>
<td>48</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>si488</td>
<td>48</td>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td>s208</td>
<td>18</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

The proposed iterative greedy heuristic based Hungarian algorithm has been implemented in MATLAB (2016b) environment. MATLAB HDL Coder tool is used to generate the Verilog HDL code for multiplexer bank for mode based reconfiguration. The Reconfigurable FSMIM-S architecture is described in Verilog HDL and implemented on a Xilinx xc6vlx75t Speed Grade-3 device (Virtex-6) by using Xilinx ISE 14.6 [15]. All computations are performed using a computer with an Intel(R) Core(TM) i5, 8 GB RAM, and 2.67 GHz CPU.

Let \( x_1, x_2, x_3, \ldots \) be the input lines, \( y_1, y_2, y_3, \ldots \) be the output lines, and \( S_1, S_2, S_3, \ldots \) be the states of an FSM. In the proposed algorithm, at the first stage, input matching is performed along with the state matching: after that, dummy states and transitions are replaced. Then, output matching is performed (Algorithm 4).

As the number of inputs or outputs exceeds 8, it requires the generation of more than \( 8^6 P_8 = 40320 \) combinations for matching, which exhausts the simulation resources. Hence, the excess input lines are discarded from input matching, which contains the maximum number of don’t cares out of the total number of transitions. Similarly, the excess output lines are discarded from output matching, which contains the minimum number of 1’s out of the total number of transitions. Therefore, the complexities of input selector bank and group encoder are reduced because the information content of these lines is minimum.

The FSM “si494” has been considered as base\_ckt, because it consists of 48 states, 8 inputs, 19 outputs, and 250 transitions which are of higher values as compared with any of the FSMs in the collection. Hence, “si494” is considered as an FSM included in the design at the 0th iteration. In this case, state splitting is never used for dummy state replacement, because base\_ckt contains the highest number of states. All dummy states and transitions are replaced by using Proposition 1. For output matching, \( y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{16}, y_{17}, y_{18} \) are discarded because they contain 62, 12, 13, 8, 6, 6, 38, 4, 16, 46, and 78 instances of 1’s, respectively, out of a total of 250 transitions.

In the 1st iteration, an FSM, “sand,” is included in the design. For input matching, \( x_6, x_7, \) and \( x_9 \) are discarded because they contain 178, 150, and 182 don’t cares, respectively, out of a total of 184 transitions. All states are matched with the
states of base_ckt in respective order. For output matching, $y_8$ is discarded because it contains 3 instances of 1’s, out of a total of 253 transitions.

In the 2nd iteration, an FSM, "styr," is included in the design. For input matching, $x_4$ is discarded because it contains 160 don’t cares, out of a total of 166 transitions. States S3, S4, S15, and S16 are matched with S4, S3, S16, and S15, respectively, of base_ckt. The rest of the states are matched with the states of base_ckt in respective order. For output matching, $y_4$ and $y_9$ are discarded because they contain 5 and 6 instances of 1’s, respectively, out of a total of 254 transitions.

In the 3rd iteration, an FSM, "planet," is included in the design. All states are matched with the states of base_ckt in respective order. For input matching, $y_2$, $y_{10}$, $y_{11}$, $y_{12}$, $y_{13}$, $y_{14}$, $y_{15}$, $y_{16}$, $y_{17}$, $y_{18}$, and $y_{19}$ are discarded because they contain 19, 5, 2, 6, 13, 3, 4, 2, 4, 4, and 6 instances of 1’s, respectively, out of a total of 255 transitions.

In the 4th iteration, an FSM, "s386," is included in the design. For input matching, $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, $x_6$, $x_7$, $x_{11}$, $x_{12}$, $x_{13}$, $x_{14}$, and $x_{15}$ are discarded because they contain 240, 228, 239, 234, 241, 224, 230, 235, 241, and 241 don’t cares, respectively, out of a total of 245 transitions. All states are matched with the states of base_ckt in respective order. For output matching, $y_3$, $y_4$, $y_5$, $y_6$, $y_9$, $y_{10}$, $y_{12}$, $y_{13}$, $y_{14}$, $y_{16}$, $y_{17}$, and $y_{18}$ are discarded because they contain 3, 5, 2, 2, 6, 2, 4, 2, 4, and 6 instances of 1’s, respectively, out of a total of 259 transitions.

In the 5th iteration, an FSM, "mc," is included in the design. All states of "mc" are matched with the states of base_ckt in respective order. In the 6th iteration, an FSM, "s386," is included in the design. All states of "s386" are matched with the states of base_ckt in respective order. In the 7th iteration, an FSM, "ex6," is included in the design. All states of "ex6" are matched with the states of base_ckt in respective order. In the 8th iteration, an FSM, "mc," is included in the design. All states of "mc" are matched with the states of base_ckt in respective order. In the 9th iteration, an FSM, "planet," is included in the design. All states are matched with the states of base_ckt in respective order. For output matching, $y_4$, $y_{10}$, $y_{11}$, $y_{12}$, $y_{13}$, $y_{14}$, $y_{15}$, $y_{16}$, $y_{17}$, $y_{18}$, and $y_{19}$ are discarded because they contain 19, 5, 2, 26, 13, 3, 4, 2, 4, 4, and 23 instances of 1’s, respectively, out of a total of 279 transitions.

In the 10th iteration, an FSM, "s1488," is included in the design. All states are matched with the states of base_ckt in respective order. For output matching, $y_1$, $y_2$, $y_3$, $y_4$, $y_5$, $y_6$, $y_{10}$, $y_{11}$, $y_{14}$, and $y_{17}$ are discarded because they contain 6, 7, 4, 13, 6, 38, 10, 16, 42, 61, and 64 instances of 1’s, respectively, out of a total of 281 transitions.

In the 11th iteration, an FSM, "s208," is included in the design. For input matching, $x_3$, $x_4$, and $x_5$ are discarded because they contain 153, 153, and 153 don’t cares, respectively, out of a total of 153 transitions. All states are matched with the states of base_ckt in respective order. The input matching and output matching among FSMs are shown in Tables 2 and 3, respectively, along with the minimum assignment_cost, total_cost, and XOR_count (as defined in Algorithm 1).

At the end of the proposed algorithm, optimized multiplexer bank for mode-based reconfiguration is formed by performing a mutual Bitwise-XOR operation between the updated descriptions of FSMs. Hence, to evaluate the individual hardware contribution of FSMs in the Reconfigurable FSMIM-S architecture is evaluated as follows: (i) The Bitwise-XOR operation is performed iteratively between the updated description of base_ckt and the FSM included in that particular iteration. (ii) The Verilog HDL code for the required multiplexer bank is generated at every iteration to implement in Xilinx xc6vlx75t-3 device. (iii) Thus, the number of LUTs occupied by the particular FSM is measured by the difference between the numbers of LUTs in the current iteration and previous iteration of the system. The iterative implementation of the Reconfigurable FSMIM-S architecture on Virtex-6 is shown in Table 4. Therefore, at the last iteration, the total number of LUTs required and the average speed of operation are obtained for the proposed architecture.

The experimental results from MCNC FSM benchmarks illustrate the advantages of the proposed architecture as compared with VRMUX [11]. As a result, operating speed is enhanced at an average of 30.43%, and LUT consumption is reduced by an average of 5.16% in FPGA implementation. It also shows that the operating speed is improved at an average of 9.14% in comparison with CRMUX [11] during FPGA implementation. The limitation of the proposed technique is the requirement of higher LUTs, as it requires an average of 88.65% more LUTs in comparison with CRMUX [11] during FPGA implementation. The comparisons of hardware
### Table 2: Input matching among FSMs.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>s1494 base_ckt</th>
<th>sand recon.ckt_1</th>
<th>styr recon.ckt_2</th>
<th>planet recon.ckt_3</th>
<th>s832 recon.ckt_4</th>
<th>cse recon.ckt_5</th>
<th>s386 recon.ckt_6</th>
<th>ex6 recon.ckt_7</th>
<th>mc recon.ckt_8</th>
<th>planet1 recon.ckt_9</th>
<th>s1488 recon.ckt_10</th>
<th>s208 recon.ckt_11</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>x4</td>
<td>x7</td>
<td>x6</td>
<td>x9</td>
<td>x5</td>
<td>x4</td>
<td>x3</td>
<td>-</td>
<td>x2</td>
<td>x1</td>
<td>x3</td>
<td>x1</td>
</tr>
<tr>
<td>x2</td>
<td>x2</td>
<td>x4</td>
<td>x3</td>
<td>x16</td>
<td>-</td>
<td>-</td>
<td>x1</td>
<td>-</td>
<td>x1</td>
<td>x1</td>
<td>x3</td>
<td>x3</td>
</tr>
<tr>
<td>x3</td>
<td>x3</td>
<td>x3</td>
<td>x4</td>
<td>x38</td>
<td>-</td>
<td>-</td>
<td>x1</td>
<td>-</td>
<td>x1</td>
<td>x3</td>
<td>x6</td>
<td>x6</td>
</tr>
<tr>
<td>x4</td>
<td>x10</td>
<td>x3</td>
<td>x2</td>
<td>x17</td>
<td>x3</td>
<td>x7</td>
<td>x1</td>
<td>-</td>
<td>x1</td>
<td>x3</td>
<td>x4</td>
<td>x9</td>
</tr>
<tr>
<td>x5</td>
<td>x3</td>
<td>x3</td>
<td>x5</td>
<td>x8</td>
<td>x4</td>
<td>x1</td>
<td>x2</td>
<td>x2</td>
<td>x2</td>
<td>x2</td>
<td>x2</td>
<td>x11</td>
</tr>
<tr>
<td>x6</td>
<td>x5</td>
<td>x5</td>
<td>x7</td>
<td>x7</td>
<td>x1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x2</td>
<td>x7</td>
<td>x2</td>
</tr>
<tr>
<td>x7</td>
<td>x9</td>
<td>x1</td>
<td>x1</td>
<td>x5</td>
<td>x6</td>
<td>x3</td>
<td>-</td>
<td>x7</td>
<td>-</td>
<td>x7</td>
<td>x2</td>
<td>x10</td>
</tr>
<tr>
<td>x8</td>
<td>x11</td>
<td>x2</td>
<td>-</td>
<td>x10</td>
<td>x7</td>
<td>x3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x8</td>
<td>x7</td>
</tr>
</tbody>
</table>

Minimum assignment cost: 0 0 0 0 0 0 0 0 0 2 0

Minimum total cost: 11 50 14 22 19 31 6 1 17 42 33
Table 3: Output matching among FSMs.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>s1494</th>
<th>sand</th>
<th>styr</th>
<th>planet</th>
<th>s832</th>
<th>cse</th>
<th>s386</th>
<th>ex6</th>
<th>mc</th>
<th>planetl</th>
<th>s1488</th>
<th>s208</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>base_ckt</td>
<td>recon_ckt_1</td>
<td>recon_ckt_2</td>
<td>recon_ckt_3</td>
<td>recon_ckt_4</td>
<td>recon_ckt_5</td>
<td>recon_ckt_6</td>
<td>recon_ckt_7</td>
<td>recon_ckt_8</td>
<td>recon_ckt_9</td>
<td>recon_ckt_10</td>
<td>recon_ckt_11</td>
</tr>
<tr>
<td>$y_8$</td>
<td>$y_4$</td>
<td>$y_5$</td>
<td>$y_2$</td>
<td>$y_{19}$</td>
<td>$-$</td>
<td>$y_6$</td>
<td>$y_5$</td>
<td>$y_4$</td>
<td>$y_6$</td>
<td>$y_7$</td>
<td>$-$</td>
<td>$y_3$</td>
</tr>
<tr>
<td>$y_{11}$</td>
<td>$y_6$</td>
<td>$y_6$</td>
<td>$y_6$</td>
<td>$y_{15}$</td>
<td>$y_7$</td>
<td>$y_1$</td>
<td>$y_8$</td>
<td>$y_2$</td>
<td>$y_7$</td>
<td>$y_7$</td>
<td>$y_7$</td>
<td>$y_1$</td>
</tr>
<tr>
<td>$y_{12}$</td>
<td>$y_3$</td>
<td>$y_7$</td>
<td>$y_8$</td>
<td>$y_2$</td>
<td>$y_3$</td>
<td>$y_3$</td>
<td>$y_4$</td>
<td>$y_5$</td>
<td>$y_1$</td>
<td>$y_1$</td>
<td>$y_4$</td>
<td>$-$</td>
</tr>
<tr>
<td>$y_{13}$</td>
<td>$y_9$</td>
<td>$y_8$</td>
<td>$y_9$</td>
<td>$y_8$</td>
<td>$y_5$</td>
<td>$y_4$</td>
<td>$y_1$</td>
<td>$y_1$</td>
<td>$y_4$</td>
<td>$y_9$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$y_{14}$</td>
<td>$y_7$</td>
<td>$y_3$</td>
<td>$y_5$</td>
<td>$y_7$</td>
<td>$y_1$</td>
<td>$-$</td>
<td>$y_6$</td>
<td>$y_3$</td>
<td>$y_2$</td>
<td>$y_9$</td>
<td>$-$</td>
<td>$y_2$</td>
</tr>
<tr>
<td>$y_{15}$</td>
<td>$y_3$</td>
<td>$y_2$</td>
<td>$y_3$</td>
<td>$y_1$</td>
<td>$y_4$</td>
<td>$y_7$</td>
<td>$y_7$</td>
<td>$-$</td>
<td>$y_5$</td>
<td>$y_{13}$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$y_{17}$</td>
<td>$y_5$</td>
<td>$y_1$</td>
<td>$y_1$</td>
<td>$y_{11}$</td>
<td>$y_6$</td>
<td>$y_3$</td>
<td>$y_2$</td>
<td>$-$</td>
<td>$y_3$</td>
<td>$y_{12}$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$y_{18}$</td>
<td>$y_1$</td>
<td>$y_{10}$</td>
<td>$y_7$</td>
<td>$y_5$</td>
<td>$y_2$</td>
<td>$y_3$</td>
<td>$-$</td>
<td>$y_6$</td>
<td>$y_{16}$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>XOR_count</td>
<td>95</td>
<td>163</td>
<td>106</td>
<td>76</td>
<td>134</td>
<td>154</td>
<td>86</td>
<td>72</td>
<td>142</td>
<td>103</td>
<td>136</td>
<td></td>
</tr>
</tbody>
</table>
**Input.** Combination of input lines for base ckt, \( a_{m_{\text{base}}}, h_{\text{base}} \), combination of input lines for recon ckt, \( a_{m_{\text{recon}}}, h_{\text{recon}} \);

**Output.** Cost matrix \( C \) /* cost matrix formation to map recon ckt states into base ckt states */

begin

base ckt array ← create an array at each transition in base ckt by combining
\[ \text{input combination} \in \{ x_{1_{\text{base}}}, \ldots, x_{L_{\text{base}}} \}, a_{m_{\text{base}}}; \]

recon ckt array ← create an array at each transition in recon ckt by combining
\[ \text{input combination} \in \{ x_{1_{\text{recon}}}, \ldots, x_{L_{\text{recon}}} \}, a_{m_{\text{recon}}}; \]

for each (state, \( a_{m_{\text{recon}}} \) ∈ recon ckt array) do

replace \( a_{1_{\text{base}}} \leftarrow a_{1_{\text{recon}}}; \)

* go to transition matching /* calling the function - “transition_matching” */

\[ C[1, 1] \leftarrow \text{match_count}; \]

replace \( a_{L_{\text{base}}} \leftarrow a_{1_{\text{recon}}}; \)

* go to transition matching /* calling the function - “transition_matching” */

\[ C[1, 2] \leftarrow \text{match_count}; \]

---

replace \( a_{1_{\text{base}}} \leftarrow a_{1_{\text{recon}}}; \)

* go to transition matching /* calling the function - “transition_matching” */

\[ C[1, M_{\text{base}}] \leftarrow \text{match_count}; \]

---

 *** transition_matching ***

for each (transition in recon ckt array, \( h_{\text{recon}} \in \{ 1, 2, \ldots, t_{m_{\text{recon}}} \} \) do

if \( t_{m_{\text{base}}} \geq t_{m_{\text{recon}}} \) then

add \( t_{m_{\text{base}}} - t_{m_{\text{recon}}} \) dummy transitions in the recon ckt array;

\[ \beta \leftarrow t_{m_{\text{base}}}; \]

else if \( t_{m_{\text{base}}} < t_{m_{\text{recon}}} \) then

add \( t_{m_{\text{recon}}} - t_{m_{\text{base}}} \) dummy transitions in the base ckt array;

\[ \beta \leftarrow t_{m_{\text{recon}}}; \]

end

for each (transition in base ckt array, \( h_{\text{base}} \in \{ 1, 2, \ldots, t_{m_{\text{base}}} \} \) do

perform Bitwise-XOR operation between the arrays for that particular transition;

\( C[h_{\text{recon}}, h_{\text{base}}] \) ← the total number of 1’s in the Bitwise-XOR operation;

end

end

GBH hungarian_algorithm(); /* calling the procedure - “GBH_hungarian_algorithm” */

arrange the recon ckt arrays based on assignment obtained from

GBH_hungarian_algorithm;

[match_count ← \( \sum_{i=1}^{t_{m_{\text{base}}}} \sum_{j=1}^{t_{m_{\text{recon}}}} C_{ij} \lambda_{ij}; \]

end

Algorithm 5: Weight assignment.

requirement and operating speed are presented in Figures 3 and 4, respectively.

The operating speed of the proposed system is maximum (i.e., 810.17 MHz) and its LUT requirement is minimum (i.e., 42 LUTs) in the 0th iteration. The operating speed is reduced, and the LUT requirement is increased successively by adding an FSM at each iteration as shown in Table 4. Therefore, the proposed architecture acts as an ideal candidate for such applications where the similarity between the sets of FSMs is high (i.e., fewer differences in their descriptions). Many
Input. Cost matrix C
Output. Optimal assignment between V and U
begin
  \( v_i \leftarrow V, u_i \leftarrow U \), \( f_2 \leftarrow \phi(v_i, u_i) \) and \( k \leftarrow 1 \) /* Initialization */
while \( ((f^V_k > f^U_k) \&\& (f^U_k > f^V_{k-1})) \) do
  \( C^V_k = \frac{1}{|u_k|} \sum_{i \in V_k} \sum_{j \in u_k} C_{ij}; \)
  \( C^U_k = \frac{1}{|V_k|} \sum_{i \in V_k} \sum_{j \in u_k} C_{ij}; \)
  \( f^V_k = \min_{u_k} \phi(v_k, u_k); \)
  \( f^U_k = \min_{v_k} \phi(v_k, u_k); \)
if \( C^V_k \leq C^U_k \) then /* performing vertex elimination from set V */
  if \( f^V_k \leq f^V_{k-1} \) then /* identify whether vertex elimination from V is “profitable” */
    \( f_k \leftarrow f^V_k \), \( v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k; \)
  end
else if \( f^V_k \leq f^V_{k-1} \) then /* identify whether vertex elimination from U is “profitable” */
  if \( f^V_k \leq f^V_{k-1} \) then /* identify whether vertex elimination from V is “profitable” */
    \( f_k \leftarrow f^V_k \), \( v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k; \)
  end
end
else if \( C^V_k > C^U_k \) then /* performing vertex elimination from set U */
  if \( f^U_k \leq f^U_{k-1} \) then /* identify whether vertex elimination from U is “profitable” */
    \( f_k \leftarrow f^U_k \), \( v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k; \)
  end
else if \( f^V_k \leq f^V_{k-1} \) then /* identify whether vertex elimination from V is “profitable” */
  if \( f^V_k \leq f^V_{k-1} \) then /* identify whether vertex elimination from V is “profitable” */
    \( f_k \leftarrow f^V_k \), \( v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k; \)
  end
end
end
end
define the GBH Hungarian algorithm.

Algorithm 6: GBH Hungarian algorithm.

Input. \( \alpha_{m,\text{base}}, M_{\text{base}}, M_{\text{recon}}, h \)
Output. Resultant states to replace dummy states in base.ckt
begin
  while \( ((\text{matched state, } \alpha_{m,\text{base}} \in \text{base.ckt}) \&\& (M_{\text{base}} \geq M_{\text{recon}})) \) do
    for each (transition in base.ckt, \( h_{\text{base}} \in \{1, 2, \ldots, t_{m,\text{base}}\} \)) do
      if \( (t_{m,\text{base}} - t_{m,\text{recon}}) \geq 1 \) then
        split the state, [\( \Psi(\alpha_{m,\text{base}}) = Q > 1 \)];
      end
    end
  end
  \( m_{\text{base}} \leftarrow m_{\text{base}} + 1; \)
end
if \( M_{\text{base}} > M_{\text{recon}} \) then
  replace dummy states in recon.ckt.b by Proposition 1;
end
end
Algorithm 7: Base.ckt state splitting.

FPGA families such as Altera stratix-IV, stratix-V, or MAX- II do not contain RAM blocks, and hence CRMUX cannot be used. The proposed architecture is preferred in such cases.

Moreover, in the proposed algorithm, the next state function is partially included in matching, and binary state encoding is used. The experimental results from [22–24] show that the evolutionary state encoding algorithms such as [23] or [24] outperform the binary or random state encoding techniques by an average of 59.72% and 64.06%, respectively. Therefore, the LUT requirement for the proposed architecture can be further reduced by 20 to 30% by using the evolutionary state encoding techniques.

4. Concluding Remarks
This paper presents a high-speed reconfigurable FSM with input multiplexing and state-based input selection (Reconfigurable FSM1M-S) architecture. The creation of such
Table 4: Iterative implementation of the Reconfigurable FSMIM-S architecture on Virtex-6.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>FSM included in the particular iteration</th>
<th>#LUTs occupied in the particular iteration</th>
<th>Maximum operating frequency</th>
<th>Maximum path delay</th>
<th>#LUTs occupied by the FSM (#LUTs in the current iteration - #LUTs in the previous iteration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0th</td>
<td>s1494</td>
<td>42</td>
<td>810.17 MHz</td>
<td>4.571 ns</td>
<td>42</td>
</tr>
<tr>
<td>1st</td>
<td>sand</td>
<td>79</td>
<td>779.271 MHz</td>
<td>4.778 ns</td>
<td>37</td>
</tr>
<tr>
<td>2nd</td>
<td>styr</td>
<td>105</td>
<td>775.164 MHz</td>
<td>4.455 ns</td>
<td>26</td>
</tr>
<tr>
<td>3rd</td>
<td>planet</td>
<td>137</td>
<td>728.704 MHz</td>
<td>4.609 ns</td>
<td>32</td>
</tr>
<tr>
<td>4th</td>
<td>s832</td>
<td>199</td>
<td>725.005 MHz</td>
<td>5.381 ns</td>
<td>62</td>
</tr>
<tr>
<td>5th</td>
<td>cse</td>
<td>230</td>
<td>722.335 MHz</td>
<td>5.140 ns</td>
<td>31</td>
</tr>
<tr>
<td>6th</td>
<td>s386</td>
<td>249</td>
<td>720.643 MHz</td>
<td>5.662 ns</td>
<td>19</td>
</tr>
<tr>
<td>7th</td>
<td>ex6</td>
<td>255</td>
<td>715.231 MHz</td>
<td>4.967 ns</td>
<td>6</td>
</tr>
<tr>
<td>8th</td>
<td>mc</td>
<td>269</td>
<td>706.889 MHz</td>
<td>4.526 ns</td>
<td>14</td>
</tr>
<tr>
<td>9th</td>
<td>planet1</td>
<td>303</td>
<td>676.338 MHz</td>
<td>5.098 ns</td>
<td>34</td>
</tr>
<tr>
<td>10th</td>
<td>s1488</td>
<td>330</td>
<td>671.760 MHz</td>
<td>4.486 ns</td>
<td>27</td>
</tr>
<tr>
<td>11th</td>
<td>s208</td>
<td>349</td>
<td>665.181 MHz</td>
<td>3.014 ns</td>
<td>19</td>
</tr>
</tbody>
</table>

Note: #LUTs denotes the number of LUTs in ISE.

architecture leads to a problem of defining the optimized multiplexer bank for mode based reconfiguration for the set of FSMs in a particular application. This situation transforms the problem into a weighted bipartite graph matching problem where the objective is to match the description of FSMs in the set with minimal cost. As a solution, an iterative greedy heuristic based Hungarian algorithm is proposed, which provides the required optimized multiplexer bank. By using the proposed architecture, operating speed is enhanced at an average of 30.43% and LUT consumption is reduced by an average of 5.16% in FPGA implementation in comparison with VRMUX [11]. It has also been shown that the operating speed is improved at an average of 9.14% as compared with CRMUX [11]. The only trade-off of the proposed technique is that it requires 88.65% more LUTs compared with CRMUX [11] during FPGA implementation.

Further, the improvement on this work is focused on reducing the LUT requirement to implement the proposed architecture. In this study, a binary state encoding is used, and next state function is partially included in matching. However, evolutionary state encoding algorithms such as [23] or [24] can be used to reduce the increased LUT requirement.

Conflicts of Interest

The authors declare that they have no conflicts of interest.
References


[21] https://people.engr.ncsu.edu/brglez/CBLbenchmarks/LGSynth89/fsmexamples/.


