Hindawi
International Journal of Reconfigurable Computing
Volume 2018, Article ID 6831901, 15 pages
https://doi.org/10.1155/2018/6831901

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

Academic Editor: Michael Hübner

The mathematical model for designing a complex digital system is a finite state machine (FSM). Applications such as digital signal processing (DSP) and built-in self-test (BIST) require specific operations to be performed only in the particular instances. Hence, the optimal synthesis of such systems requires a reconfigurable FSM. The objective of this paper is to create a framework for a reconfigurable FSM with input multiplexing and state-based input selection (Reconfigurable FSMIM-S) architecture. The Reconfigurable FSMIM-S architecture is constructed by combining the conventional FSMIM-S architecture 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. The problem of obtaining the required optimized multiplexer bank is transformed into a weighted bipartite graph matching problem where the objective is to iteratively match the description of FSMs in the set with minimal cost. As a solution, an iterative greedy heuristic based Hungarian algorithm is proposed. The experimental results from MCNC FSM benchmarks demonstrate a significant speed improvement by 30.43% as compared with variation-based reconfigurable multiplexer bank (VRMUX) and by 9.14% in comparison with combination-based reconfigurable multiplexer bank (CRMUX) during field programmable gate array (FPGA) implementation.

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 at 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 base_ckt) in that set. The objective of this process is to perform optimal matching between base_ckt 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 base_ckt 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., base_ckt 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 ($a_m \in A$, where $A = \{a_1, \ldots, a_M\}$ 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 ($h \in H$, where $H = \{t_1, \ldots, t_M\}$ is a set of number of transitions per state corresponding to A); a_s is a state of transition (the next state); $K(a_s)$ is a code of state $a_s \in A$; $X = \{x_1, \ldots, x_L\}$ 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), $R \in \{\lfloor \log_2 M \rfloor, M\}$.

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 base_ckt) in that set. The inputs, states, and outputs of an FSM in the set are mapped into base_ckt in their respective order. The mode bits are applied through a 2×1 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 base_ckt mode is the default mode of operation. Similarly, all other FSMs in the set are mapped into base_ckt. In this way, a single FSM (i.e., base_ckt) 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., base_ckt) 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 base_ckt but exist in the other FSMs in the set and vice versa. These states and transitions lead the system to failure.

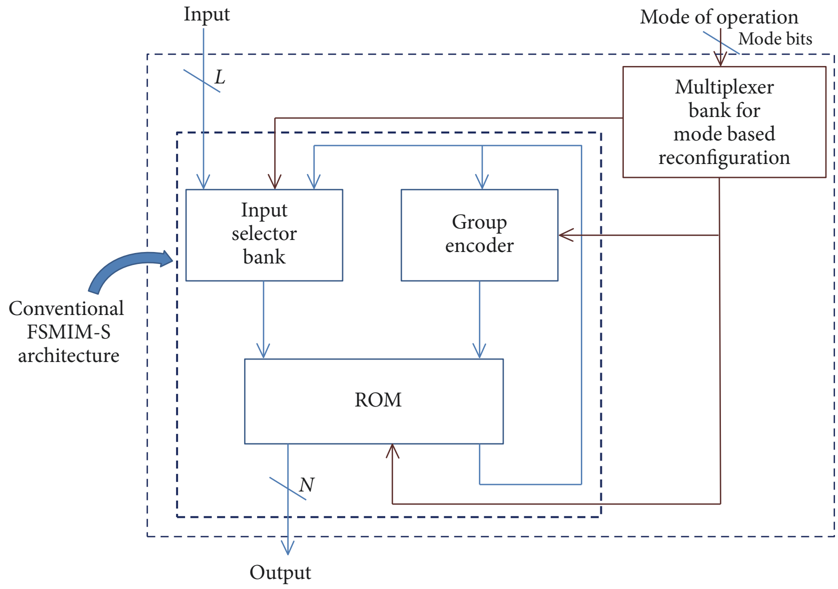

Figure 1: Reconfigurable FSMIM-S architecture.

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 $\in$ $\{1, \ldots, B\}$, 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 $L\_\text{base} \geq L\_\text{recon}$, then $E = {}^{L\_\text{base}}P_{L\_\text{recon}}$ combinations of input lines for base_ckt are generated to match with input lines of recon_ckt_b. $(L\_\text{base} - L\_\text{recon})$ input lines act as don't cares while the system operates in recon_ckt_b mode. Otherwise, $E = {}^{L\_\text{recon}}P_{L\_\text{base}}$ combinations of input lines for recon_ckt_b are generated to match with input lines of base_ckt. In this case, $(L\_\text{recon} - L\_\text{base})$ 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,

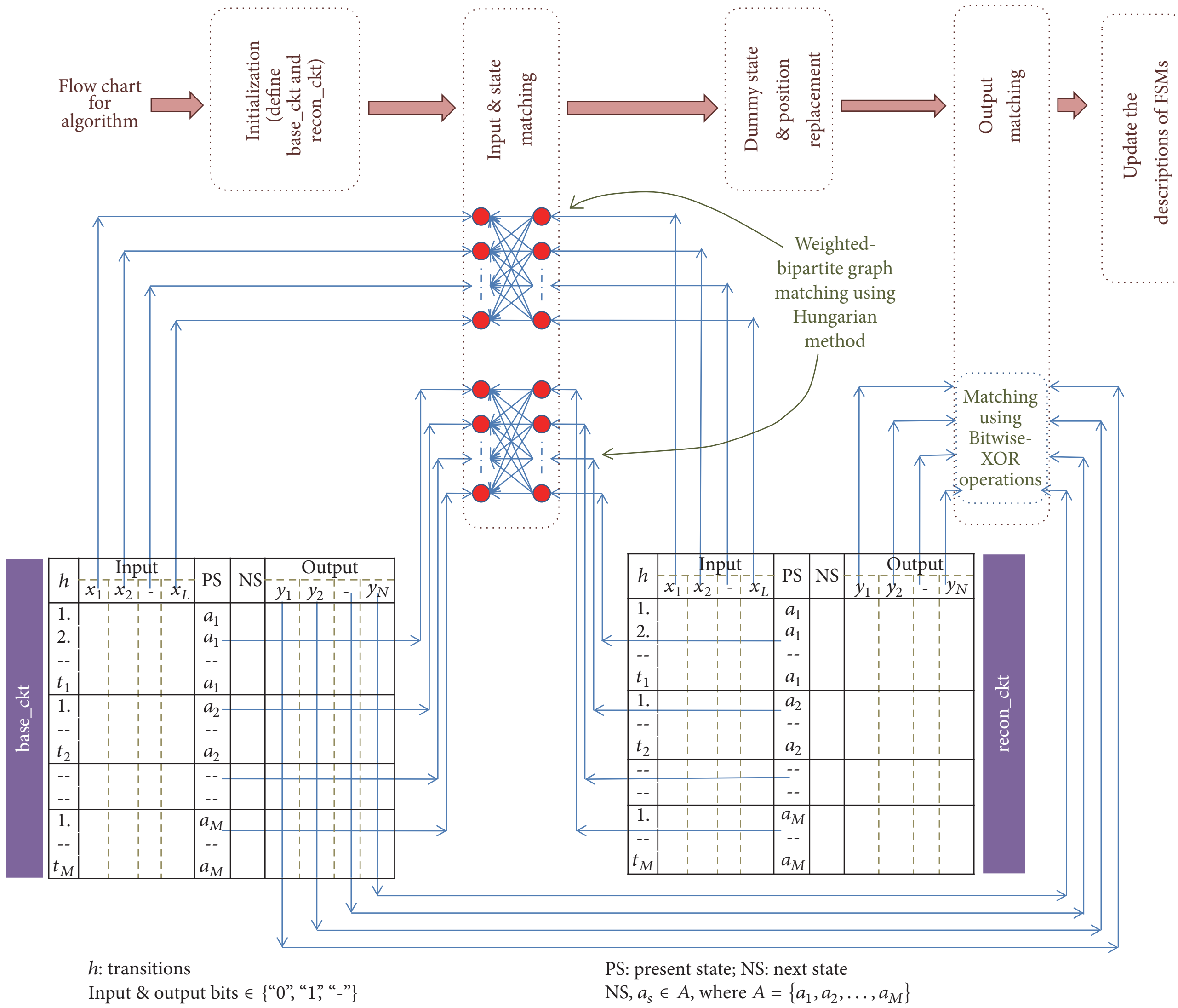

FIGURE 2: Flow chart for iterative greedy heuristic based Hungarian algorithm.

the number of states in both the FSMs is equalized. Thus, if M_base $\geq$ M_recon, then (α − M_recon, where α = M_base) dummy states are added in recon_ckt_b. Otherwise (α − M_base, where α = M_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 [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 1's 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_s, Y])
Output. The optimized multiplexer bank for mode based reconfiguration
begin
select the largest FSM from the set based on the description;
base_ckt ← largest FSM;
recon_ckt_1, recon_ckt_2, ..., recon_ckt_B ← rest of the FSMs in the set;
for each recon_ckt_b ∈ {recon_ckt_1, recon_ckt_2, ..., recon_ckt_B} do
        if (L_base ≥ L_recon) then /∗ performing the input matching ∗/
                generate, E ← ^{L_base}P_{L_recon} combinations_of_input_lines for base_ckt to match with
                input lines of recon_ckt_b;
                go to state_matching; /∗ calling the function- "state_matching" ∗/
        else if (L_base < L_recon) then
                generate, E ← ^{L_recon}P_{L_base} combinations_of_input_lines for recon_ckt_b to match
                with input lines of base_ckt;
                go to state_matching; /∗ calling the function- "state_matching" ∗/
        end
        select combinations_of_input_lines with min{assignment_cost_1, ..., assignment_cost_E}
        & min{total_cost_1, total_cost_2, ..., total_cost_E};
        perform binary state assignment in base_ckt & recon_ckt_b i.e. apply K(a_m) & K(a_s);
        weight_assignment( ); /∗creating arrays by [selected_input_combination, K(a_s), K(a_m)]∗/
        go to dummy_replacement; /∗ calling the function- "dummy_replacement" ∗/
        if (N_base ≥ N_recon) then /∗ performing the output matching ∗/
                generate, G ← ^{N_base}P_{N_recon} combinations_of_output_lines for base_ckt to match
                with output lines of recon_ckt_b;
                go to output_matching; /∗ calling the function- "output_matching" ∗/
        else if (N_base < N_recon) then
                generate, G ← ^{N_recon}P_{N_base} combinations_of_output_lines for recon_ckt_b to match
                with output lines of base_ckt;
                go to output_matching; /∗ calling the function- "output_matching" ∗/
        end
        select combinations_of_output_lines with min{XOR_count_1, ..., XOR_count_G};
        update the description of base_ckt;
end
for each recon_ckt_b ∈ {recon_ckt_1, recon_ckt_2, ..., recon_ckt_(B − 1)} do
        go to dummy_replacement; /∗ calling the function- "dummy_replacement" ∗/
        update the description of recon_ckt_b;
end
perform a mutual (i.e. ^B C_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 recon_ckt_array. Let match_count be a variable defined as

$$\text{match\_count} = \sum_{i=1}^{\beta}\sum_{j=1}^{\beta} C_{ij}\lambda_{ij}, \tag{1}$$

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

In this way, by using match_count (from (1)), the cost matrix formation to map recon_ckt_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 recon_ckt and base_ckt, respectively. $\mu = (V \cup U, \xi)$ is defined as a balanced weighted bipartite graph, where $|V| = |U| = \eta$. 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 μ 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 η resources to η tasks in such a way that optimal total cost should be obtained for the

```
Input. Combinations_of_input_lines, the descriptions of base_ckt & recon_ckt_b (i.e. [h, X, a_m, a_s, Y])
Output. Assignment_cost_e, total_cost_e, modified description of recon_ckt_b
begin
for all (combinations_of_input_lines) do
    if (M_base ≥ M_recon) then /* equating the number of states in both the FSMs */
        add (M_base − M_recon) dummy states in recon_ckt_b;
        α ← M_base;
    else if (M_base < M_recon) then
        add (M_recon − M_base) dummy states in base_ckt;
        α ← M_recon;
    end
    weight_assignment( ); /* calling the procedure- "weight_assignment" */
    GBH_hungarian_algorithm( ); /* performing present state matching */
        assignment_cost_e ← Σ_{i=1}^{α} Σ_{j=1}^{α} C_ij · λ_ij; where e ∈ {1, 2, ..., E};
        total_cost_e ← Σ_{i=1}^{α} Σ_{j=1}^{α} C_ij; where e ∈ {1, 2, ..., E};
    replace all states of recon_ckt_b by their corresponding states of base_ckt obtained
    from GBH_hungarian_algorithm;
    corresponding to a_{m_base} order, arrange all the complete arrays of recon_ckt_b;
end
end
```

Algorithm 2: State_matching.

assignment. The LAP can be mathematically formulated as follows:

$$f_k \equiv \varphi(v_k, u_k) = \min \quad \sum_{i=1}^{\eta}\sum_{j=1}^{\eta} C_{ij}\lambda_{ij}; \tag{2}$$

$$\text{s.t.} \quad \sum_{j=1}^{\eta} \lambda_{ij} = 1, \quad \text{where } \forall i = 1, \ldots, \eta; \tag{3}$$

$$\sum_{i=1}^{\eta} \lambda_{ij} = 1, \quad \text{where } \forall j = 1, \ldots, \eta; \tag{4}$$

$$\lambda_{ij} \in \{0, 1\}, \quad \text{where } \forall i, j = 1, \ldots, \eta. \tag{5}$$

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:

$$C_k^V = \frac{1}{|u_k|} \cdot \sum_{i\in v_k}\sum_{j\in u_k} C_{ij}, \qquad C_k^U = \frac{1}{|v_k|} \cdot \sum_{i\in v_k}\sum_{j\in u_k} C_{ij}; \tag{6}$$

$$f_k^V = \min_{i\in v_k} \varphi(v_k, u_k); \tag{7}$$

$$f_k^U = \min_{j\in u_k} \varphi(v_k, u_k). \tag{8}$$

In (6), C_k^V and C_k^U 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 $C_k^V \le C_k^U$, then an attempt is made to remove one of the vertices from $v_k \subseteq V$. From (7), if $f_k^V \le f_{k-1}$, that is, the objective function value is improved by eliminating i_k, then v_{k+1} is set to v_k and the next iteration is executed.

Otherwise, one of the vertices from $u_k \subseteq U$ is eliminated. From (8), if $f_k^U \le f_{k-1}$, that is, the objective function value is improved by eliminating j_k, then u_{k+1} is set to u_k 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 $C_k^V > C_k^U$, 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:

$$\text{assignment\_cost}_e \longleftarrow \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha} C_{ij} \cdot \lambda_{ij}, \quad \text{where } e \in \{1, 2, \ldots, E\}; \qquad \text{total\_cost}_e \longleftarrow \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha} C_{ij}, \quad \text{where } e \in \{1, 2, \ldots, E\}. \tag{9}$$

```
Input. The descriptions of base_ckt & recon_ckt_b (i.e. [h, X, a_m, a_s, Y])
Output. The descriptions base_ckt & recon_ckt_b without dummy states and dummy transitions.
begin
if (M_base ≥ M_recon) then /* replacing the dummy states */
        replace dummy states in recon_ckt_b by Proposition 1; /* by considering states in place of
        transitions in the modified cost matrix */
else if (M_base < M_recon) then
        replace dummy states in base_ckt by by Proposition 2;
end
for each (matched state, a_m_recon ∈ recon_ckt_b) do /* replacing the dummy transitions */
        for each (transition in recon_ckt_b, h_recon ∈ {1, 2, ..., t_m_recon}) do
                if (t_m_base ≥ t_m_recon) then
                        replace dummy transitions in recon_ckt_b by Proposition 1;
                else if (t_m_base < t_m_recon) then
                        replace dummy transitions in base_ckt by Proposition 1;
                end
        end
end
end
```

Algorithm 3: Dummy_replacement.

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_base} order. Hence, from (9), the combinations of input lines are selected with $\min\{\text{assignment\_cost}_1, \ldots, \text{assignment\_cost}_E\}$ & $\min\{\text{total\_cost}_1, \text{total\_cost}_2, \ldots, \text{total\_cost}_E\}$.

Now, binary state codes $K(a_m)$ and $K(a_s)$ 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 [selected_input_combination, $K(a_s)$, $K(a_m)$]. 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 $M\_C_{i\times j}$ 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:

$$\text{unassigned\_}U_i \longrightarrow V_j\text{: } \min\left(M\_C_{1j}, M\_C_{2j}, M\_C_{3j}, \ldots, M\_C_{ij}\right). \tag{10}$$

Similarly, the unassigned transitions in recon_ckt_b state can be assigned by (11) as follows:

$$\text{unassigned\_}V_j \longrightarrow U_i\text{: } \min\left(M\_C_{i1}, M\_C_{i2}, M\_C_{i3}, \ldots, M\_C_{ij}\right). \tag{11}$$

Proposition 2. *If $M_$base < $M_$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 $M_$base > $M_$recon, then there are ($M_$base − $M_$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. □

If $M_$base < $M_$recon, then ($M_$recon − $M_$base) dummy states are replaced by splitting the matched state in base_ckt. Let $\Psi(a_{m\_\text{base}}) = Q(t_{m\_\text{base}} - t_{m\_\text{recon}})$, where Q is a positive integer. Only the states for which $|\Psi(a_{m\_\text{base}}) > 1|$ can be split [19]. Each state can be split into nonoverlapping subsets of $(t_{m\_\text{base}} - t_{m\_\text{recon}})$ 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 $N_$base ≥ $N_$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{XOR_count$_1, \ldots,$ 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 Bth iteration, for each recon_ckt_b ∈ {recon_ckt_1, recon_ckt_2, ..., 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., ${}^{B}C_2$) 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 1: Benchmark circuits from MCNC/LGSynth [21].

Benchmark circuits	Number of states	Number of inputs	Number of outputs
s1494	48	8	19
sand	32	11	9
styr	30	9	10
planet	48	7	19
s832	25	18	19
cse	16	7	7
s386	13	7	7
ex6	8	5	8
mc	4	3	5
planet1	48	7	19
s1488	48	8	19
s208	18	11	2

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 $S1, S2, S3, \ldots$ 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 ${}^{8}P_8 = 40320$ 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 "s1494" 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, "s1494" 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, $y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_9, y_{10}, y_{16}$, and 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

Input. Combinations_of_output_lines
Output. XOR_count_g
begin
for all (combinations_of_output_lines) do
    perform Bitwise-XOR operation between corresponding output lines of base_ckt & recon_ckt_b;
    $\text{XOR\_count}_g \leftarrow$ the total number of 1's in the Bitwise-XOR operation;
        where, $g \in \{1, 2, \ldots, G\}$;
end
end

Algorithm 4: Output_matching.

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_9 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 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 255 transitions.

In the 4th iteration, an FSM, "s832," is included in the design. For input matching, $x_1, x_2, x_3, x_4, x_6, x_6, x_{11}, x_{12}, x_{13}, x_{14}$, 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, $y_3, y_4, y_6, y_9, y_{10}, y_{12}, y_{13}, y_{14}, y_{16}, y_{17}$, and y_{18} are discarded because they contain 3, 5, 2, 2, 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, "cse," is included in the design. All states of "cse" 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, "planet1," 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, $y_1, y_2, y_3, y_4, y_5, y_6, y_8, y_{10}, y_{11}, y_{14}$, 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.

	Circuits										
s1494 base_ckt	sand recon_ckt_1	styr recon_ckt_2	planet recon_ckt_3	s832 recon_ckt_4	cse recon_ckt_5	s386 recon_ckt_6	ex6 recon_ckt_7	mc recon_ckt_8	planet1 recon_ckt_9	s1488 recon_ckt_10	s208 recon_ckt_11
x_1	x_4	x_7	x_6	x_9	x_5	x_4	x_3	-	x_5	x_1	x_1
x_2	x_2	x_4	x_3	x_{16}	x_2	-	x_4	x_3	x_1	x_5	x_8
x_3	x_1	x_5	x_4	x_{18}	-	x_2	-	x_1	x_6	x_3	x_6
x_4	x_{10}	x_3	x_2	x_{17}	x_3	x_7	x_1	-	x_3	x_4	x_9
x_5	x_5	x_8	x_5	x_8	x_4	x_1	x_2	x_2	x_4	x_8	x_{11}
x_6	x_3	x_6	x_7	x_7	x_1	x_5	-	-	x_2	x_7	x_2
x_7	x_8	x_1	x_1	x_5	x_6	x_6	x_5	-	x_7	x_2	x_{10}
x_8	x_{11}	x_2	-	x_{10}	x_7	x_3	-	-	-	x_6	x_7
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.

	Circuits										
s1494 base_ckt	sand recon_ckt_1	styr recon_ckt_2	planet recon_ckt_3	s832 recon_ckt_4	cse recon_ckt_5	s386 recon_ckt_6	ex6 recon_ckt_7	mc recon_ckt_8	planet1 recon_ckt_9	s1488 recon_ckt_10	s208 recon_ckt_11
y_8	y_4	y_5	y_2	y_{19}	-	y_6	y_5	y_4	y_6	y_{18}	-
y_{11}	y_6	y_6	y_6	y_{15}	y_7	y_1	y_8	y_2	y_7	y_7	y_1
y_{12}	y_2	y_7	y_8	y_2	y_3	y_3	y_4	y_5	y_1	y_{13}	-
y_{13}	y_9	y_8	y_9	y_8	y_5	y_4	y_1	y_1	y_8	y_{19}	-
y_{14}	y_7	y_3	y_5	y_7	y_1	-	y_6	y_3	y_2	y_9	y_2
y_{15}	y_3	y_2	y_3	y_1	y_4	y_7	y_7	-	y_5	y_{15}	-
y_{17}	y_5	y_1	y_1	y_{11}	y_6	y_5	y_2	-	y_3	y_{12}	-
y_{19}	y_1	y_{10}	y_7	y_5	y_2	y_2	y_3	-	y_9	y_{16}	-
XOR_count	95	163	106	76	134	154	86	72	142	103	136

```
Input. Combination_of_input_lines for base_ckt, a_m_base, h_base,
       combination_of_input_lines for recon_ckt_b, a_m_recon, h_recon;
Output. Cost matrix C /* cost matrix formation to map recon_ckt_b states into base_ckt states */
begin
base_ckt_array ← create an array at each transition in base_ckt by combining
                 [input_combination ∈ {x_1_base, ..., x_L_base}, a_m_base];
recon_ckt_array ← create an array at each transition in recon_ckt_b by combining
                 [input_combination ∈ {x_1_recon, ..., x_L_recon}, a_m_recon];
for each (state, a_m_recon ∈ recon_ckt_array) do
    replace a_1_base ← a_1_recon;
    go to transition_matching; /*calling the function- "transition_matching"*/
    C[1, 1] ← match_count;

    replace a_2_base ← a_1_recon;
    go to transition_matching; /*calling the function- "transition_matching"*/
    C[1, 2] ← match_count;
        -----------------
        -----------------
        -----------------
    replace a_M_base ← a_1_recon;
    go to transition_matching; /*calling the function- "transition_matching"*/
    C[1, M_base] ← match_count;

                    ***transition_matching***
    for each (transition in recon_ckt_array, h_recon ∈ {1, 2, ..., t_m_recon}) do
        if (t_m_base ≥ t_m_recon) then
            add (t_m_base − t_m_recon) dummy transitions in the recon_ckt_array;
            β ← t_m_base;
        else if (t_m_base < t_m_recon) then
            add (t_m_recon − t_m_base) dummy transitions in the base_ckt_array;
            β ← t_m_recon;
        end
        for each (transition in base_ckt_array, h_base ∈ {1, 2, ..., t_m_base}) do
            perform Bitwise-XOR operation between the arrays for that particular
            transition;
            C[h_recon, h_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 ← Σ_{i=1}^{β} Σ_{j=1}^{β} C_ij λ_ij;
end
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
$v_1 \leftarrow V, u_1 \leftarrow U, f_0 \leftarrow \varphi(v_1, u_1)$ and $k \leftarrow 1$ /∗ Initialization ∗/
while $((f_k^V > f_{k-1})$ && $(f_k^U > f_{k-1}))$ do
    $C_k^V = (1/|u_k|) \cdot \sum_{i \in v_k} \sum_{j \in u_k} C_{ij}$;
    $C_k^U = (1/|v_k|) \cdot \sum_{i \in v_k} \sum_{j \in u_k} C_{ij}$;
    $f_k^V = \min_{i \in v_k} \varphi(v_k, u_k)$;
    $f_k^U = \min_{j \in u_k} \varphi(v_k, u_k)$;
    if $C_k^V \leq C_k^U$ then /∗ performing vertex elimination from set V ∗/
        if $f_k^V \leq f_{k-1}$ then /∗ identify whether vertex elimination from V is "profitable" ∗/
            $f_k \leftarrow f_k^V, v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k$;
        else if $f_k^U \leq f_{k-1}$ then /∗ identify whether vertex elimination from U is "profitable" ∗/
            $f_k \leftarrow f_k^U, v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k$;
        end
    else if $C_k^V > C_k^U$ then /∗ performing vertex elimination from set U ∗/
        if $f_k^U \leq f_{k-1}$ then /∗ identify whether vertex elimination from U is "profitable" ∗/
            $f_k \leftarrow f_k^U, v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k$;
        else if $f_k^V \leq f_{k-1}$ then /∗ identify whether vertex elimination from V is "profitable" ∗/
            $f_k \leftarrow f_k^V, v_{k+1} \leftarrow v_k, u_{k+1} \leftarrow u_k$;
        end
    end
    $k \leftarrow k + 1$;
end
end

Algorithm 6: GBH_hungarian_algorithm.

Input. a_{m_base}, M_base, M_recon, h
Output. Resultant states to replace dummy states in base_ckt
begin
while ((matched state, $a_{m\_\text{base}} \in$ base_ckt) && (M_base $\geq$ M_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, $|\Psi(a_{m\_\text{base}}) = Q > 1|$;
        end
    end
    m_base $\leftarrow$ m_base + 1;
end
if (M_base > M_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 FSMIM-S) architecture. The creation of such

Table 4: Iterative implementation of the Reconfigurable FSMIM-S architecture on Virtex-6.

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

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

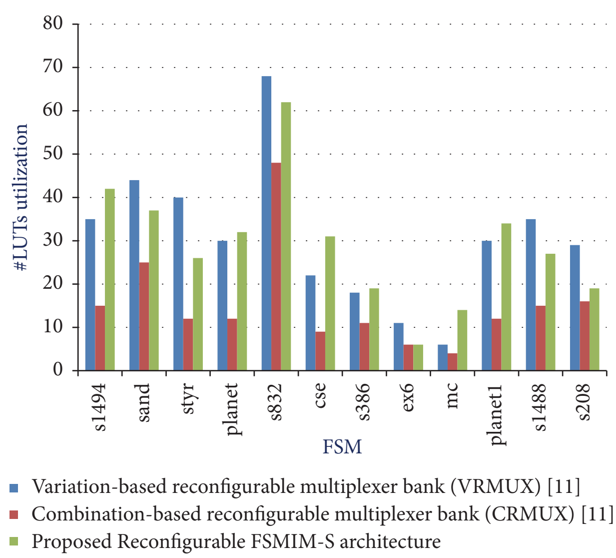

Figure 3: Comparison of hardware requirements during FPGA implementation.

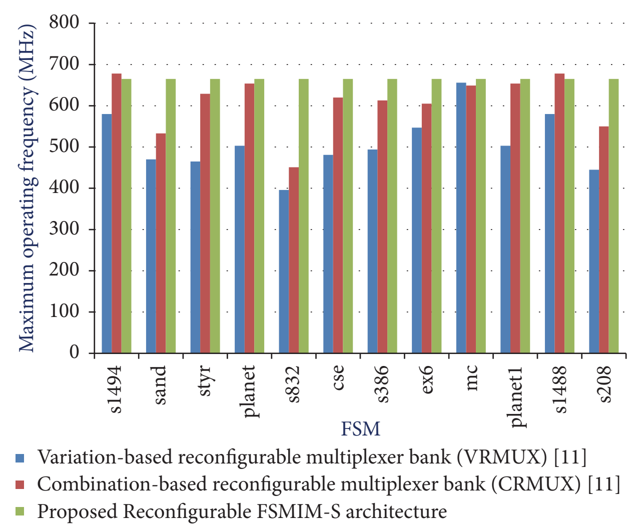

Figure 4: Comparison of operating speeds during FPGA implementation.

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

[1] J. Glaser, M. Damm, J. Haase, and C. Grimm, "TR-FSM: Transition-based Reconfigurable finite state machine," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 3, article no. 23, 2011.

[2] A. Karatkevich, *Design of Reconfigurable Logic Controllers*, vol. 45, springer, Berlin, Germany, 2016.

[3] J. Wu, D. Yang, and Z. Chen, "Design and application of multi-stage reconfigurable signal processing flow on FPGA," *Computers and Electrical Engineering*, vol. 42, pp. 1–11, 2015.

[4] E. De Lucas, M. Sanchez-Elez, and I. Pardines, "DSPONE48: A methodology for automatically synthesize HDL focus on the reuse of DSP slices," *Journal of Parallel and Distributed Computing*, vol. 106, pp. 132–142, 2017.

[5] M. Nandini Priya and R. Brindha, "An enhanced architecture for high performance BIST TPG," in *Proceedings of the 2nd IEEE International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2015*, pp. 1–6, Coimbatore, India, March 2015.

[6] K. Mielcarek, A. Barkalov, and L. Titarenko, "Designing LUT-based mealy FSM with transformation of collections of output functions," in *Proceedings of the 5th International Conference on Modern Circuits and Systems Technologies, MOCAST 2016*, pp. 1–4, Thessaloniki, Greece, May 2016.

[7] R. Senhadji-Navarro, I. García-Vargas, G. Jiménez-Moreno, and A. Civit-Ballcels, "ROM-based FSM implementation using input multiplexing in FPGA devices," *IEEE Electronics Letters*, vol. 40, no. 20, pp. 1249–1251, 2004.

[8] A. Klimowicz, V. Solov'Ev, and T. Grzes, "Minimization method of finite state machines for low power design," in *Proceedings of the 18th Euromicro Conference on Digital System Design, DSD 2015*, pp. 259–262, Funchal, Portugal, August 2015.

[9] S. N. Pradhan and P. Choudhury, "Low power and high testable Finite State Machine synthesis," in *Proceedings of the International Conference and Workshop on Computing and Communication, IEMCON 2015*, pp. 1–5, Canada, October 2015.

[10] I. Garcia-Vargas and R. Senhadji-Navarro, "Finite state machines with input multiplexing: A performance Study," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 867–871, 2015.

[11] R. Senhadji-Navaro and I. Garcia-Vargas, "high-speed and area-efficient reconfigurable multiplexer bank for RAM-based finite state machine implementations," *Journal of Circuits, Systems and Computers*, vol. 24, no. 7, Article ID 1550101, 2015.

[12] V. Sklyarov and I. Skliarova, "Synthesis of parallel hierarchical finite state machines," in *Proceedings of the 2013 21st Iranian Conference on Electrical Engineering, ICEE 2013*, Iran, May 2013.

[13] S. Gupta, V. Pareek, S. C. Jain, and D. Jain, "Realization of sequential reversible circuit from finite state machine," in *Proceedings of the International Computer Science and Engineering Conference, ICSEC 2014*, pp. 458–463, Khon Kaen, Thailand, August 2014.

[14] M. Barketau, E. Pesch, and Y. Shafransky, "Minimizing maximum weight of subsets of a maximum matching in a bipartite graph," *Discrete Applied Mathematics*, vol. 196, pp. 4–19, 2015.

[15] K. Date and R. Nagi, "GPU-accelerated Hungarian algorithms for the linear assignment problem," *Parallel Computing*, vol. 57, pp. 52–72, 2016.

[16] J. Dutta and S. C. Pal, "A note on Hungarian method for solving assignment problem," *Journal of Information and Optimization Sciences*, vol. 36, no. 5, pp. 451–459, 2015.

[17] V. Stozhkov, V. Boginski, O. . Prokopyev, and E. L. Pasiliao, "A simple greedy heuristic for linear assignment interdiction," *Annals of Operations Research*, vol. 249, no. 1-2, pp. 39–53, 2017.

[18] S. K. Ramadoss, A. P. Singh, and I. K. G. Mohiddin, "An evolutionary heuristic algorithm for the assignment problem," *OPSEARCH*, vol. 51, no. 4, pp. 589–602, 2014.

[19] T. N. Grzes and V. V. Solov'ev, "Minimization of power consumption of finite state machines by splitting their internal states," *Journal of Computer and Systems Sciences International*, vol. 54, no. 3, pp. 367–374, 2015.

[20] V. Salauyou, "Synthesis of high-speed finite state machines in FPGAs by state splitting," in *Computer Information Systems and Industrial Management: 15th IFIP TC8 International Conference, CISIM 2016*, K. Saeed and W. Homenda, Eds., vol. 9842 of *Lecture Notes in Computer Science*, pp. 741–751, Springer International Publishing, Vilnius, Lithuania, 2016.

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

[22] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementations," in *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 327–332, June 1989.

[23] A. H. El-Maleh, "Majority-based evolution state assignment algorithm for area and power optimisation of sequential circuits," *IET Computers & Digital Techniques*, vol. 10, no. 1, pp. 30–36, 2016.

[24] A. H. El-Maleh, "A probabilistic pairwise swap search state assignment algorithm for sequential circuit optimization," *Integration, the VLSI Journal*, vol. 56, pp. 32–43, 2017.