Research Article

FPGA Implementation of an Improved Reconfigurable FSMIM Architecture Using Logarithmic Barrier Function Based Gradient Descent Approach

Nitish Das and Aruna Priya P

Department of ECE, SRM Institute of Science & Technology, Kattankulathur-603203, Chennai, India

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

Received 31 October 2018; Revised 2 January 2019; Accepted 10 January 2019; Published 1 April 2019

Academic Editor: John Kalomiros

Copyright © 2019 Nitish Das and Aruna Priya P. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recently, the Reconfigurable FSM has drawn the attention of the researchers for multistage signal processing applications. The optimal synthesis of Reconfigurable finite state machine with input multiplexing (Reconfigurable FSMIM) architecture is done by the iterative greedy heuristic based Hungarian algorithm (IGHA). The major problem concerning IGHA is the disintegration of a state encoding technique. This paper proposes the integration of IGHA with the state assignment using logarithmic barrier function based gradient descent approach to reduce the hardware consumption of Reconfigurable FSMIM. Experiments have been performed using MCNC FSM benchmarks which illustrate a significant area and speed improvement over other architectures during field programmable gate array (FPGA) implementation.

1. Introduction

Digital signal processing (DSP) [1–3], pattern matching [4], and circuit testing [5] are the primary applications for most of the digital systems. These applications require a hardware-oriented as well as high-speed control unit. A finite state machine (FSM) is an integral part of any complex digital system. Its inputs are multiplexed to make it hardware oriented, which is known as the finite state machine with input multiplexing (FSMIM). It serves as a control unit, and its operating speed determines the processing speed of the system. The applications as mentioned earlier can be observed as cascaded stages (i.e., multistage) of operations [2], where each stage requires a specific FSM. Hence, a Reconfigurable FSM is investigated in the literature for optimal performance in such applications [6, 7]. A Reconfigurable FSM is defined as a single FSM, which acts as one of the FSMs from the set (i.e., set of FSMs for a specific application) by applying particular mode bits. Its implementation is performed on field programmable gate array (FPGA) platforms [6].

The Reconfigurable FSMIM architecture is created by joining (A) Conventional FSMIM architecture [8] and (B) multiplexer bank (which defines the mode based reconfiguration). The optimal synthesis of both the constituting elements is done by Iterative greedy heuristic based Hungarian algorithm (IGHA) [6]. An efficient state encoding technique for an FSM serves as a vital tool to optimize the hardware utilization while implementing on an FPGA platform [9, 10]. In the case of Reconfigurable FSMIM, the state encoding of the constituent FSMs altogether affects the look-up table (LUT) requirement of the Reconfigurable FSMIM [6].

The major problem concerning IGHA is the disintegration of a state encoding technique. It uses binary state encoding as a default state assignment technique for operation. The state assignment method for the Reconfigurable FSMIM architecture leads to an optimization problem [6]. To the best of the authors’ knowledge, all the state assignment techniques proposed in the literature provide state codes only for a single FSM. Therefore, the objective of this work is the integration of IGHA with an optimal state encoding technique to reduce the hardware consumption of Reconfigurable FSMIM on an FPGA platform.

In the literature, another direction in the implementation of an FSM is RAM-based architectures. The following three
types of RAM-based FSM architectures are studied [11]: (a) basic RAM-based FSM architecture, (b) RAM-based FSM architecture with transition-controlled multiplexers, and (c) RAM-based FSM architecture with state-controlled multiplexers. In the basic RAM-based FSM architecture, bits are stored in the form of words. For each transition (i.e., present state combined with the external inputs), the outputs and the state assignment bits for next state are stored in the RAM word memory [12, 13]. The RAM size required for basic RAM-based FSM implementation is enormous. Hence, to reduce the RAM depth, RAM-based FSM architecture with transition-controlled multiplexers is used. It consists of an input selector bank, which provides active inputs from the external inputs for selecting a particular state [11]. RAM-based FSM architecture with state-controlled multiplexers is used to reduce the RAM size further. It consists of two separate RAM blocks, out of which the smaller RAM block is assigned to operate the input selector bank [11]. Thus, designing such architecture is very complicated.

In this paper, the Improved Reconfigurable FSMIM architecture is proposed, which surmounts the issue of high LUT consumption during FPGA implementation. The proposed architecture is formed using the improved iterative greedy heuristic based Hungarian algorithm (Improved-IGHA). The Improved-IGHA is the integration of IGHA with the state assignment using logarithmic barrier function based gradient descent approach.

To validate the proposed approach, experiments have been performed using MCNC FSM benchmarks [14]. Experimental results for the proposed architecture illustrate a significant area reduction by an average of 20.38% and speed improvement by an average of 32.73% over VRMUX [11] during FPGA implementation. It also demonstrates an adequate area reduction by an average of 16.05% and speed improvement by an average of 1.77% over Reconfigurable FSMIM-S architecture [6] during FPGA implementation. When these results are compared with CRMUX [11], a speed improvement by an average of 11.06% is obtained. The proposed architecture requires an average of 58.38% more LUTs as compared with CRMUX [11] during FPGA implementation. It is the only trade-off for the proposed design.

The remainder of this article is formed as follows. The research problem formulation is made in Section 2. Section 3 consists of state assignment using logarithmic barrier function based gradient descent approach and an illustrative example. Experimental setup and comparative analysis of this work with the literature are devised in Section 4. In the end, concluding remarks are drawn in Section 5.

2. Problem Formulation

Recently, the Reconfigurable FSM has drawn the attention of the researchers for multistage signal processing applications. A novel framework for the creation of Reconfigurable FSMIM is given in [6].

A Mealy FSM is represented in a vector form, such as $(S, X, Y, \delta, \pi, S_0)$ where

$S = (S_0, \ldots, S(M)) \leftarrow$ set of states;

$X = (x_1, \ldots, x_L) \leftarrow$ set of input variables;

$Y = (y_1, \ldots, y_N) \leftarrow$ set of output variables;

$\delta \Rightarrow S * X \rightarrow S \leftarrow$ transition function;

$\pi \Rightarrow S * X \rightarrow Y \leftarrow$ output function;

$S_0 \leftarrow$ initial state.

Moreover, the following variables are defined to illustrate the complete functionality of an FSM:

$S(m) \leftarrow$ any instantaneous state $S(m) \in S$ where $m \in (0, 1, \ldots, M)$;

$K(S(m))$ $\leftarrow$ binary state code for the state $S(m) \in S$;

$H = (t_1, \ldots, t_M)$ $\leftarrow$ set of number of transitions per state corresponding to $S$;

$h \leftarrow$ number of transitions per state where $h \in (1, 2, \ldots, H)$;

$R \leftarrow$ the minimum length of a binary-state code, $R = \lceil \log_2 M \rceil$.

The Reconfigurable FSMIM is defined as a single FSM, which acts as any one of the FSM from the set (i.e., set of FSMs for a specific application) by applying particular mode bits. A set of FSM for a specific application is chosen, where $base_ckt \leftarrow$ the largest FSM (i.e., the FSM with the highest total number of transitions, states, and inputs) in the set and $recon_ckt_1, recon_ckt_2, \ldots, recon_ckt_B$ $\leftarrow$ rest of the FSMs in the set. $base_ckt$-mode is the default mode of operation for the Reconfigurable FSMIM [6].

The Reconfigurable FSMIM architecture is created by joining the following two parts: (A) Conventional FSMIM architecture [8], & (B) Multiplexer bank (which defines the mode based reconfiguration). The optimal synthesis of the Multiplexer bank is done by iterative greedy heuristic based Hungarian algorithm (IGHA) [6]. At the last phase of IGHA, state transitions of each constituent FSM of the Reconfigurable FSMIM architecture are presented in Figure 1. Therefore, the state encoding of the constituent FSMs altogether affects the LUT requirement of the Reconfigurable FSMIM architecture. At the end of IGHA, a modified description of a single FSM (i.e., $base_ckt$) is obtained which is used to create the Conventional FSMIM part [6].

In FSM implementation on an FPGA platform, state encoding technique acts as a tool for minimizing the hardware consumption [9, 10]. For example, an MCNC FSM benchmark $tbk$ requires 82 LUTs when implemented on a Xilinx xc6vlx75t-3 device (Virtex-6) using the Grey encoding technique. But it needs only 41 LUTs on the same platform using the binary encoding technique.

The major problem concerning IGHA is the disintegration of a state encoding technique. It uses binary state encoding as a default state assignment technique for operation [6]. The state assignment method for the Reconfigurable FSMIM architecture leads to an optimization problem as evident from Figure 1. To the best of the authors’ knowledge, all the state assignment techniques proposed in the literature provide state codes only for a single FSM.
3. Methodology

This work is an extension of work presented in [6]. Hence, all the variables from [6] are used in the same context throughout the article. An improved version of IGHA (Improved-IGHA) is proposed. It addresses the issue of optimal state encoding.

A recent body of literature has investigated the performance of three fundamental types of state encoding techniques on an FPGA platform [9]. The studied methods are as follows: (a) structural methods, (b) heuristic approaches, and (c) pragmatic approaches. Out of these three approaches, structural state encoding technique outperforms on an FPGA platform [9,10]. It uses the knowledge of internal structure (i.e., state transition) of the FSM to generate optimal state codes. Therefore, structural information of FSMs is considered to develop the proposed state encoding technique for the Reconfigurable FSMIM.

The structural information of the Reconfigurable FSMIM (i.e., state transition) is obtained from Figure 1. Hence, a unified weight matrix is defined by adding the weight of all component FSMs for the same corresponding states. It is given in (1).

The mathematical formulation of the cost function for an FSM is given in [15]. It uses the structural information (i.e., state transitions) of the particular FSM. Let \( \omega_{ij} \) be the element of weight matrix and \( \text{Dist}_{\text{Matrix}}_{ij} \) be the hamming distance between two particular state codes. \( \text{Dist}_{\text{Matrix}}_{ij} \) is obtained by counting the number of 1’s after an exclusive-OR operation between the binary state codes as shown in Figure 2. Therefore, from the literature [15], the cost associated with a particular set of state codes (i.e., \( \mu \)) is defined by (2).

\[
\omega_{11} = \omega_{11}\text{base}_\text{ckt} + \omega_{11}\text{recon}_\text{ckt}_1 + \omega_{11}\text{recon}_\text{ckt}_B + \cdots \\
+ \omega_{11}\text{recon}_\text{ckt}_B \\
\vdots \\
\omega_{M(M-1)} = \omega_{M(M-1)}\text{base}_\text{ckt} + \omega_{M(M-1)}\text{recon}_\text{ckt}_1 + \omega_{M(M-1)}\text{recon}_\text{ckt}_B + \cdots + \omega_{M(M-1)}\text{recon}_\text{ckt}_B \\
\vdots \\
\omega_{M1} = \omega_{M1}\text{base}_\text{ckt} + \omega_{M1}\text{recon}_\text{ckt}_1 + \omega_{M1}\text{recon}_\text{ckt}_2 + \cdots + \omega_{M1}\text{recon}_\text{ckt}_B \\
\vdots \\
\omega_{MM} = \omega_{MM}\text{base}_\text{ckt} + \omega_{MM}\text{recon}_\text{ckt}_1 + \omega_{MM}\text{recon}_\text{ckt}_2 + \cdots + \omega_{MM}\text{recon}_\text{ckt}_B \\
\text{cost} (\mu) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} (\omega_{ij} \times \text{Dist}_{\text{Matrix}}_{ij}) 
\]
where $E_{\text{map}}$ (i.e., $\omega_{ij}$) indicates the edge weights between the nodes & $V_{\text{map}}$ (i.e., columns of $\mu$) represents the set of nodes. Hence, each node corresponds to a particular binary state code because $\mu_{ij}$ opts only the binary labels. $M$ symbolizes the total number of nodes in the graph $G_{\text{map}}$.

Let a hypercube be characterized as $\chi_\eta = (V_\chi, E_\chi)$, where $\eta$ is the dimension, $E_\chi$ is the set of edges, and $V_\chi$ is the set of vertices of the hypercube [16]. The cardinality of $E_\chi$ and $V_\chi$ is given in (3) and (4), respectively.

$$|E_\chi| = |V_\chi| \times \frac{\eta}{2} = 2^{\eta-1} \times \eta$$  \hspace{1cm} (3) \\
$$|V_\chi| = 2^\eta$$  \hspace{1cm} (4)

Now, the concept of hypercube embedding is used to reduce (2). An embedding is performed from graph $G_{\text{map}}$ onto a hypercube $\chi_\eta$ as described earlier [16, 17]. It is defined as $\mu : V_{\text{map}} \rightarrow V_\chi$ which is a one-to-one mapping function. Consequently, $M$-binary $\eta$-vectors are defined as in (5). Thus, if a node of graph $G_{\text{map}}$ (i.e., $i$) is expressed by a binary state code, the corresponding vertex of the hypercube (i.e., $k_i$) is represented by the same binary state code.

$$k_i \in \{k : k \in \{0, 1\}^\eta\}$$

where $i \in V_{\text{map}}$ (i.e., columns of $\mu$) 

In a hypercube, $\text{Dist}_\chi_{ij}$ represent the hamming distance between $k_i$ and $k_j$. It is shown in

![Diagram](image-url)
(6), where \( r_{ij} \) is the instantaneous value of \( k_{ij} \). The value of \( r_{ij} \) varies between \(-1\) and \(1\). Therefore, the cost function is reduced to (7) using hypercube embedding.

\[
\text{DistMatr}_{x_j} = \sum_{k=1}^{n} \left( r_{ik} - r_{jk} \right)^2
\]

where, \( k_{ij} = \begin{cases} 
1 & \text{if } r_{ij} \geq 0 \\
0 & \text{if } r_{ij} < 0
\end{cases} \)

\[
\text{cost}(\mu) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \sum_{k=1}^{n} \left( \omega_{ij} \times (r_{ik} - r_{jk})^2 \right)
\]

The objective is thus confined to minimize the cost function given in (7). Evidently, it is a discrete optimization problem, where each state can opt only a particular binary state code.

The convergence of Improved-IGHA depends on the convergence of its constituent algorithms, i.e., IGHA and the applied state assignment technique. Therefore, an algorithm with a high convergence speed is preferred to construct the state assignment technique for Improved-IGHA.

The evolutionary technique, such as genetic algorithm (GA), presents a significant shortcoming as its convergence speed slows down near the global optimum [18, 19]. Similarly, particle swarm optimization (PSO) and differential evolution (DE) operate with a higher convergence rate but suffer premature convergence, which is a critical drawback of these methods [20, 21]. In the literature, penalty-based approaches, such as Lagrangian technique and logarithmic-barrier function [22, 23], have proven their potentials to obtain the optimum solution with a high convergence speed [22, 23]. These methods are advantageous in solving a discrete or combinatorial optimization problem [24, 25].

Therefore, the LBF-based Gradient descent approach is adopted to construct the state assignment technique for Improved-IGHA. It is a full-fledged method that assures the feasible solution. The mathematical formulation of the cost minimization function is performed by LBF. Then, it is reduced iteratively by the gradient-projection approach. The flow chart for the Improved-IGHA is presented in Figure 3.

In LBF technique, the search operation is performed in a continuous space domain to deduce the optimal points. Then, these points are discretized to obtain the optimal solution [26, 27].

In LBF method, an objective function subject to inequality constraints is given in

\[
\text{min } f(\tau)
\]

s.t. \( \text{constraint}_i(\tau) \geq 0, \quad i = 1, \ldots, u \)

(8)

The logarithmic barrier function to minimize the cost function (as in (7)) is given in (9). In LBF search, for any move which omits the constraints, the second term serves as a barrier [28] as shown in

\[
\text{min } \psi(\tau, \phi) = f(\tau) + \sum_{i=1}^{u} \log_e (\text{constraint}_i(\tau))
\]

(9)

At the iteration \( \text{iter} \), (9) is defined as shown in

\[
\text{min } \psi(\tau, \phi^{\text{iter}}) = f(\tau) + \sum_{i=1}^{u} \log_e (\text{constraint}_i(\tau))
\]

(10)

Initially, LBF selects a feasible \( \tau^0 \) and \( \phi^0 > 0 \). Then, it chooses \( \phi^{\text{iter}+1} = \sigma \cdot \phi^{\text{iter}}, \) where \( \sigma < 1 \). This iterative process goes on until \( \phi^{\text{iter}} \) reaches an adequately small value.

A full-fledged method is required to solve (10) with respect to \( \tau \). A first-order gradient-projection approach [29] is well-suited for iteratively minimizing (10). In this approach, the model parameters (a.k.a. weight vectors) are evaluated to minimize the objective function when an analytical calculation is not possible [30, 31]. In this approach, the underlying representation of the objective function of the problem is given in

\[
\text{min } \psi(\tau, \phi)
\]

s.t. \( \theta(\tau) = 0 \)

(11)

An iteration of this projection method is defined by (12).

\[
\tau \leftarrow \tau - \rho (\nabla \psi(\tau, \phi))
\]

(12)

Thus, small steps (i.e., \( \rho \)) are taken in the negative gradient direction of the objective function as illustrated in (12). Then, (13) is used to outline the value of \( \tau \) on the constraint surface at the next iteration (i.e., \( \tau^{\text{iter}+1} \)).

\[
\tau = \lfloor \tau \rfloor^{\theta(\tau)=0}
\]

(13)
The convergence criterion for this iterative process is defined by (14), where \( \theta \in [0, 1] \).

\[
\left| r^{(iter^{j+1})} - r^{(iter^{j})} \right| < \theta \quad (14)
\]

In this way, embedding problem is reduced to the determination of \( M \)-binary \( \eta \)-vectors (as shown in (15)) which optimizes the cost function (i.e., (7)).

\[
\tau_i \in \left\{ \tau : \tau \in \mathbb{R}^M \& \| \tau \|^2 = 1 \right\}; \quad i \in V_{map} \text{ where } \| \tau \|^2 \text{ denotes the norm of } \tau
\]

Hence, the cost function (from (7)) is defined in terms of Hamming distance as shown in (16).

\[
\text{cost} (\mu) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \left( \omega_{ij} \times \| \tau_i - \tau_j \|^2 \right)
\]

where, \( \tau_i = \begin{bmatrix} \tau_{i1}, \tau_{i2}, \ldots, \tau_{iM} \end{bmatrix}^T \| \tau_i - \tau_j \|^2 = \sum_{k=1}^{M} (\tau_{ik} - \tau_{jk})^2 \quad (16)
\]

The constraint (i.e., boundary condition) for this problem is formed, such as any two vertices on hypercube should not contain the same binary state code (i.e., \( \tau_i - \tau_j \neq 0 \)). Hence, the mathematical representation of the constraint is presented in (17).

\[
\text{constraint} (\tau) = \| \tau_i - \tau_j \|^2 > 0
\]

By applying (16) and (17) on (9), the objective function for LBF is reduced to

\[
\min \psi (\tau, \phi) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \left( \omega_{ij} \times \| \tau_i - \tau_j \|^2 \right) + \phi \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \log \left( \| \tau_i - \tau_j \|^2 \right)
\]

Therefore, the entity \( \Theta (\tau) \) (from (13)) is defined by

\[
\Theta (\tau) = \left[ \| \tau_1 \|^2 - 1, \| \tau_2 \|^2 - 1, \ldots, \| \tau_M \|^2 - 1 \right]^T
\]

s.t. \( \tau = \begin{bmatrix} \tau_{11}, \tau_{12}, \ldots, \tau_{1M} \end{bmatrix}^T \)

\[
\tau_i = \begin{bmatrix} \tau_{i1}, \tau_{i2}, \ldots, \tau_{iM} \end{bmatrix}^T
\]

where each \( \tau_i \) (1 \( \leq \) \( i \) \( \leq \) \( M \))

The evaluation of the derivative term (i.e., \( \nabla \psi (\tau, \phi) \)) is required to move in the gradient descent direction as shown in (12). The needed derivative term is obtained by putting (20), (21), (22), and (23) into (18). Hence, \( \nabla \psi (\tau, \phi) \) is defined by (24).

\[
\frac{\partial}{\partial \tau_i} \left( \| \tau_i - \tau_j \|^2 \right) = 2 \left( \tau_i - \tau_j \right) \quad (20)
\]

\[
\frac{\partial}{\partial \tau_j} \left( \| \tau_i - \tau_j \|^2 \right) = 2 \left( \tau_i - \tau_j \right) \quad (21)
\]

\[
\frac{\partial}{\partial \tau_i} \left( \log \left( \| \tau_i - \tau_j \|^2 \right) \right) = \frac{2 \left( \tau_i - \tau_j \right)}{\| \tau_i - \tau_j \|^2} \quad (22)
\]

\[
\nabla \psi (\tau, \phi) = 2 \begin{bmatrix} \sum_{j=1}^{M} \left( \omega_{1j} \times (\tau_1 - \tau_j) \right) + \phi \sum_{j=1}^{M} \left( \frac{(\tau_1 - \tau_j)}{\| \tau_1 - \tau_j \|^2} \right) \\
\sum_{j=1}^{M} \left( \omega_{2j} \times (\tau_2 - \tau_j) \right) + \phi \sum_{j=1}^{M} \left( \frac{(\tau_2 - \tau_j)}{\| \tau_2 - \tau_j \|^2} \right) \\
\vdots \\
\sum_{j=1}^{M} \left( \omega_{Mj} \times (\tau_M - \tau_j) \right) + \phi \sum_{j=1}^{M} \left( \frac{(\tau_M - \tau_j)}{\| \tau_M - \tau_j \|^2} \right) \end{bmatrix}
\]

(24)

By applying (19) into (13), the normalized vector \( \tau \) is defined as shown in

\[
[\tau]_{\Theta (\tau) = 0} = \begin{bmatrix} \tau_1 \\
\tau_2 \\
\vdots \\
\tau_M \end{bmatrix}
\]

If (14) is satisfied, a solution vector which is defined as \( \tilde{\tau}_{ij} \) is obtained at the end of the iteration. Therefore, the required set of state codes (i.e., \( \tilde{k}_{ij} \)) is deduced by discretizing \( \tilde{\tau}_{ij} \) using

\[
\tilde{k}_{ij} = \begin{cases} 1 & \text{if } \tilde{r}_{ij} \geq 0 \\
0 & \text{if } \tilde{r}_{ij} < 0 \end{cases} \quad (26)
\]

The pseudocode for the proposed state assignment approach is presented in Algorithm 1.

3.2. An Illustrative Example for the Improved Reconfigurable FSMIM Architecture

The following MCNC FSM benchmarks [14] are considered to demonstrate the steps involved in the creation of the Improved Reconfigurable FSMIM architecture:
| Input: the objective function defined by Equation (7) |
| Output: \( \mu^* \) (i.e., the final state code vector) |

\[
\begin{align*}
\text{begin} & \\
\text{Initialization:} & \quad \mu \leftarrow \text{Binary state codes}; \\
& \quad \phi \leftarrow \text{initial } \phi \text{ (s.t. } \phi^0 > 0); \\
\text{while} & \quad (\phi > \text{final } \phi) \quad \text{do} \\
\text{repeat} & \\
& \quad \text{for } \text{iter} \leftarrow 1 \text{ to } \theta \\
& \quad \quad \tau \leftarrow \tau^{(\text{iter}-1)} \\
& \quad \quad -\rho [\nabla \psi(\tau, \phi)]; \\
& \quad \quad / \ast \text{ by Equation (12)} \\
& \quad \quad \& \text{ Equation (24)}/ \ast \\
\text{end} & \\
\text{return} & \quad \tilde{\tau}_j \left( \text{i.e. the value of } \tau \text{ at the iteration } \theta \right); \\
\text{evaluate} & \quad \tilde{k}_j = \{1, \text{ if } \tilde{\tau}_j \geq 0; 0, \text{ if } \tilde{\tau}_j < 0\}; \\
& \quad / \ast \text{ by Equation (26)}/ \ast \\
\text{Compute cost for the new value of } \mu \text{ using Equation (7)}; \\
\text{if} & \quad (\text{cost(old } \mu) \geq \text{cost(new } \mu)) \text{ then} \\
& \quad \text{update, } \mu \leftarrow \text{new } \mu; \\
\text{else if} & \quad (\text{cost(old } \mu) < \text{cost(new } \mu)) \text{ then} \\
& \quad \text{update, } \mu \leftarrow \text{old } \mu; \\
\text{until} & \quad \text{the algorithm converges} \\
\phi & \leftarrow \sigma \cdot \phi; \\
\text{end} & \\
\text{return} & \quad \mu^*; \\
\end{align*}
\]

**Algorithm 1:** State assignment using logarithmic barrier-function based gradient descent approach for the Reconfigurable FSM.

(1) train11 (description is provided in Table 1)
(2) lion9 (description is provided in Table 2)

The improved Reconfigurable FSMIM architecture is created by joining (A) Conventional FSMIM architecture and (B) Multiplexer bank (which defines the mode based reconfiguration). The optimal synthesis of the Multiplexer bank is done by the proposed Improved-IGHA. At the end of the proposed algorithm, a modified description of a single FSM (i.e., base_skt) is obtained which is used to create the Conventional FSMIM part [6]. The Improved-IGHA consists of the following steps:

(i) **Initialization** (Define base_skt and recon_skt): train11 is selected as base_skt because its complexity is greater than lion9 as observed from their descriptions. Consequently, lion9 acts as recon_skt.

(ii) **Input and State Matching using Hungarian Algorithm:** Input and state matchings are performed together using Algorithms 1, 2, 5, and 6 from [6]. Combinations of input lines of base_skt (i.e., \( 2^2 \times 2 = 2 \)) are generated. For the first combination \((x_{1\text{[train11]}}, x_{2\text{[train11]}}), \) states are matched as \( S_0_{\text{lion9}} \rightarrow S_0_{\text{train11}}, S_1_{\text{lion9}} \rightarrow S_1_{\text{train11}}, S_2_{\text{lion9}} \rightarrow S_2_{\text{train11}}, S_3_{\text{lion9}} \rightarrow S_3_{\text{train11}}, S_4_{\text{lion9}} \rightarrow S_4_{\text{train11}}, S_5_{\text{lion9}} \rightarrow S_5_{\text{train11}}, S_6_{\text{lion9}} \rightarrow S_6_{\text{train11}}, S_7_{\text{lion9}} \rightarrow S_7_{\text{train11}}, S_8_{\text{lion9}} \rightarrow S_8_{\text{train11}}, S_9_{\text{lion9}} \rightarrow S_9_{\text{train11}}, \) and Dummy state \( \rightarrow S_{10_{\text{train11}}}. \) It offers zero assignment_cost and total_cost. For the second combination \((x_{3\text{[train11]}, x_{4\text{[train11]}}}, \) states are matched as \( S_0_{\text{lion9}} \rightarrow S_0_{\text{train11}}, S_3_{\text{lion9}} \rightarrow S_1_{\text{train11}}, S_1_{\text{lion9}} \rightarrow S_2_{\text{train11}}, S_4_{\text{lion9}} \rightarrow S_3_{\text{train11}}, S_5_{\text{lion9}} \rightarrow S_4_{\text{train11}}, S_6_{\text{lion9}} \rightarrow S_5_{\text{train11}}, S_7_{\text{lion9}} \rightarrow S_6_{\text{train11}}, S_8_{\text{lion9}} \rightarrow S_7_{\text{train11}}, S_9_{\text{lion9}} \rightarrow S_8_{\text{train11}}, S_{10_{\text{train11}}} \rightarrow S_{11_{\text{train11}}}, \) and Dummy state \( \rightarrow S_{10_{\text{train11}}}. \) It also offers zero assignment_cost and total_cost. Therefore, the first combination \((x_{1\text{[train11]}, x_{2\text{[train11]}}}, \) is finalized to match with \((x_{3\text{[train11]}, x_{4\text{[train11]}}}. \)

(iii) **Dummy State and Position Replacement:** The replacements of the dummy states and positions in base_skt and recon_skt are performed using Algorithm 3 from [6]. The replaced dummy states (highlighted in “bold italic font”) and dummy positions (highlighted in “bold font”) are presented in Tables 3 and 4.

(iv) **Output Matching using Bitwise-XOR Operations:** Output Matching is not required in this case, as...
Table 1: Description of `train1` from MCNC FSM Benchmarks [14].

| Input | PS | NS | O/P | y₁ |
|-------|----|----|-----|----|
| x₁    | x₂ |    |     |    |
| 0     | 0  | S₀ | S₀  | 0  |
| 1     | 0  | S₀ | S₁  | -  |
| 0     | 1  | S₀ | S₂  | -  |
| 1     | 0  | S₁ | S₁  | 1  |
| 0     | 0  | S₁ | S₃  | 1  |
| 1     | 1  | S₁ | S₅  | 1  |
| 0     | 1  | S₂ | S₂  | 1  |
| 0     | 0  | S₂ | S₇  | 1  |
| 1     | 1  | S₂ | S₉  | 1  |
| 0     | 0  | S₃ | S₃  | 1  |
| 0     | 1  | S₃ | S₄  | 1  |
| 0     | 0  | S₄ | S₄  | 1  |
| 1     | 1  | S₅ | S₅  | 1  |
| 0     | 1  | S₅ | S₆  | 1  |
| 0     | 1  | S₆ | S₆  | 1  |
| 0     | 0  | S₆ | S₀  | -  |
| 0     | 0  | S₇ | S₇  | 1  |
| 1     | 0  | S₇ | S₈  | 1  |
| 1     | 0  | S₈ | S₈  | 1  |
| 0     | 0  | S₈ | S₀  | -  |
| 1     | 1  | S₉ | S₉  | 1  |
| 0     | 1  | S₉ | S₁₀ | 1  |
| 1     | 0  | S₁₀| S₁₀| 1  |
| 0     | 0  | S₁₀| S₀  | -  |

There is a single output line in `base.ckt` as well as in `recon.ckt`.

(v) **Update the descriptions of FSMs:** The updated descriptions of `base.ckt` and `recon.ckt` are presented in Tables 3 and 4, respectively.

(vi) **State assignment using logarithmic barrier function based gradient descent approach for the Reconfigurable FSM:** The pictorial representation of state transitions for `base.ckt` and `recon.ckt` (from Tables 3 and 4) is given in Figure 4. Therefore, the weight matrix \( \omega \) is formed using (1). It is given in

\[
\omega = \begin{bmatrix}
2 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 1 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]  \quad (27)

The proposed state assignment algorithm starts by considering the binary state codes as an initial solution. It offers the cost as 62 (from (2)).

At the 100th iteration, the instantaneous value \( \tau \) (from previous iteration, \( \tau^{(i-99)} \)) is obtained as defined by

\[
\tau^{(iter-99)} = \begin{bmatrix}
0.51 & 0.85 & -0.85 & 0.84 \\
-0.52 & -0.85 & -0.51 & 0.85 \\
0.85 & -0.52 & -0.51 & -0.85 \\
0.84 & -0.51 & 0.51 & -0.52 \\
-0.51 & 0.51 & 0.85 & -0.52 \\
-0.85 & -0.52 & -0.85 & -0.51 \\
-0.52 & 0.84 & -0.51 & -0.85 \\
0.84 & 0.51 & -0.52 & -0.85 \\
0.51 & -0.85 & -0.85 & 0.51 \\
0.85 & 0.84 & 0.51 & -0.52 \\
-0.51 & 0.51 & -0.52 & 0.85
\end{bmatrix}^{T}
\]

Table 2: Description of `lion9` from MCNC FSM Benchmarks [14].

| Input | PS | NS | O/P | y₁ |
|-------|----|----|-----|----|
| x₁    | x₂ |    |     |    |
| 0     | 0  | S₀ | S₀  | 0  |
| 0     | 0  | S₀ | S₀  | 0  |
| 1     | 0  | S₁ | S₀  | 0  |
| 0     | 0  | S₁ | S₀  | 0  |
| 1     | 0  | S₁ | S₂  | 1  |
| 0     | 0  | S₁ | S₇  | 1  |
| 1     | 1  | S₂ | S₂  | 1  |
| 1     | 1  | S₂ | S₃  | 0  |
| 0     | 1  | S₃ | S₃  | 1  |
| 0     | 0  | S₃ | S₄  | 1  |
| 1     | 0  | S₄ | S₄  | 1  |
| 0     | 0  | S₄ | S₄  | 1  |
| 0     | 0  | S₅ | S₅  | 1  |
| 0     | 0  | S₅ | S₆  | 1  |
| 0     | 0  | S₆ | S₆  | 1  |
| 0     | 0  | S₆ | S₆  | 1  |
| 1     | 0  | S₇ | S₇  | 1  |
| 0     | 0  | S₇ | S₇  | 1  |
| 1     | 0  | S₇ | S₇  | 1  |
| 0     | 0  | S₇ | S₇  | 1  |
| 0     | 0  | S₇ | S₇  | 1  |
| 0     | 0  | S₈ | S₈  | 1  |
| 0     | 0  | S₈ | S₈  | 1  |
| 1     | 0  | S₉ | S₉  | 1  |
| 0     | 0  | S₉ | S₉  | 1  |
| 1     | 0  | S₉ | S₁₀ | 1  |
| 0     | 0  | S₁₀| S₁₀| 1  |
| 0     | 0  | S₁₀| S₁₀| 1  |
| 0     | 0  | S₁₀| S₁₀| 1  |
Table 3: Updated description of train11.

| Input | PS | NS | O/P |
|-------|----|----|-----|
| x₁  | x₂ | x₁ | x₂ | y₁ |
| 0    | 0  | S₀ | S₀ | 0 |
| 1    | 0  | S₀ | S₁ | - |
| 0    | 1  | S₀ | S₂ | - |
| 1    | 0  | S₁ | S₁ | 1 |
| 0    | 0  | S₁ | S₃ | 1 |
| 1    | 1  | S₁ | S₅ | 1 |
| 0    | 1  | S₂ | S₂ | 1 |
| 0    | 0  | S₂ | S₇ | 1 |
| 1    | 1  | S₂ | S₉ | 1 |
| 0    | 0  | S₃ | S₃ | 1 |
| 0    | 1  | S₃ | S₄ | 1 |
| 0    | 0  | S₃ | S₃ | 1 |
| 0    | 1  | S₄ | S₄ | 1 |
| 0    | 0  | S₄ | S₀ | - |
| 0    | 1  | S₄ | S₄ | 1 |
| 1    | 1  | S₅ | S₅ | 1 |
| 0    | 1  | S₆ | S₆ | 1 |
| 0    | 0  | S₆ | S₀ | - |
| 0    | 0  | S₇ | S₇ | 1 |
| 1    | 0  | S₇ | S₈ | 1 |
| 1    | 0  | S₇ | S₈ | 1 |
| 1    | 0  | S₈ | S₈ | 1 |
| 0    | 0  | S₈ | S₈ | 1 |
| 0    | 0  | S₈ | S₀ | - |
| 0    | 1  | S₉ | S₉ | 1 |
| 1    | 0  | S₉ | S₁₀ | 1 |
| 1    | 1  | S₉ | S₉ | 1 |
| 1    | 0  | S₁₀ | S₁₀ | 1 |
| 0    | 0  | S₁₀ | S₀ | - |
| 0    | 0  | S₁₀ | S₀ | - |

The derivative (from (24)) is evaluated as defined by

\[ \nabla \psi (\tau, \phi) = \begin{bmatrix} -1517 & -1224 & -2118 & 2126 \\ -1739 & 1258 & -2622 & 2960 \\ 2135 & 701 & 1558 & 402 \\ -1223 & 1598 & 1793 & 736 \\ 722 & -739 & 2886 & 760 \\ 1506 & -1209 & -1775 & 401 \\ -398 & 1754 & 763 & -2889 \\ -1549 & -2118 & 418 & -1579 \\ 2945 & 2953 & 1761 & -1736 \\ -2583 & -1504 & 1506 & 2079 \end{bmatrix}^T \tag{29} \]

So, the current value of \( \tau \) (i.e., \( \tau^{(\text{iter}100)} \)) is obtained from (12). It is given in (30) by choosing \( \rho = 10^{-3} \) (a very small value).

\[ \begin{bmatrix} 2.027 & 2.074 & 1.268 & -1.286 \\ 1.219 & -2.108 & 2.112 & -2.11 \\ -1.285 & -1.221 & -2.068 & -1.252 \\ 2.063 & -2.108 & -1.283 & -1.256 \\ -1.232 & 1.249 & -2.036 & -1.28 \\ 2.038 & -2.06 & 2.039 & 1.216 \\ 2.059 & 1.268 & -1.268 & 2.089 \\ -2.095 & -2.113 & -1.251 & 1.216 \\ 2.073 & 2.014 & -2.026 & -1.229 \end{bmatrix}^T \tag{30} \]
Then, $\tau$ is directed towards the unity radius hypersphere. It is given in

\[
\begin{bmatrix}
0.85 & 0.84 & 0.51 & -0.52 \\
0.51 & -0.85 & 0.85 & -0.85 \\
-0.52 & -0.51 & -0.85 & -0.51 \\
0.84 & -0.85 & -0.52 & -0.51 \\
-0.51 & 0.51 & -0.85 & -0.52 \\
-0.85 & -0.52 & 0.84 & -0.85 \\
0.85 & 0.85 & 0.51 & -0.51 \\
0.51 & -0.51 & -0.52 & 0.85 \\
0.85 & 0.51 & -0.51 & 0.85 \\
-0.85 & -0.85 & -0.51 & 0.51 \\
0.84 & 0.84 & -0.85 & -0.51 \\
\end{bmatrix}^T \rightarrow S0
\]

\[
\begin{bmatrix}
1110, S1 \rightarrow 1010, S2 \rightarrow 0000, S3 \rightarrow 1000, S4 \rightarrow 0100, S5 \rightarrow 0010, S6 \rightarrow 0110, S7 \rightarrow 1001, S8 \rightarrow 1101, S9 \rightarrow 0001, S10 \rightarrow 1100
\end{bmatrix}
\]

The required set of state codes is deduced as $S0 \rightarrow 1110, S1 \rightarrow 1010, S2 \rightarrow 0000, S3 \rightarrow 1000, S4 \rightarrow 0100, S5 \rightarrow 0010, S6 \rightarrow 0110, S7 \rightarrow 1001, S8 \rightarrow 1101, S9 \rightarrow 0001, and S10 \rightarrow 1100$ by discretizing the current value of $\tau$ using (26). Hence, the cost is reduced to 48 (from (2)).

In the end, a Bitwise-XOR operation is performed between the updated descriptions of train11 and lion9. It provides the Multiplexer bank (i.e., part-B). The updated descriptions of train11 are used to construct the Conventional FSMIM part (i.e., part-A).

4. Numerical Results and Discussions

To validate the proposed approach, experiments have been performed using MCNC FSM benchmarks [14]. MATLAB (2016b) environment is used to implement the proposed Improved-IGHA. It produces the optimized description for the constituting parts of the Improved Reconfigurable FSMIM architecture. The obtained description is then converted into the Verilog HDL code using MATLAB HDL Coder tool-box. The implementation of the Improved Reconfigurable FSMIM architecture is performed on the Virtex-6 speed-3 device as in [6, 11]. The configuration of the workstation to execute computations is as follows: Intel(R) Core i7 (6th Gen), 16 GB RAM, and 3.5 GHz CPU.

In Improved-IGHA, combinations of input lines, states, and output lines are generated using permutation to perform input, state, and output matching, respectively. The number of input and output lines used for matching is restricted to 7 (i.e., $7^P_7 = 5040$ combinations) to utilize the resources efficiently. Hence, the information content of an input/output line becomes the criteria for selection. An input/output line with high information content is preferred.

The following MCNC FSM benchmarks [14] are selected to illustrate the implementation of the Improved Reconfigurable FSMIM architecture and present its comparative analysis with the existing literature: s1494, s832, s208, planet, s386, sand, mc, sty, cse, ex6, planet1, and s1488.

$s1494$ is chosen as base_skt (i.e., the circuit added at the $0^{th}$ iteration of Improved-IGHA), as it is more complex (i.e., the total number of transitions is high) as compared with the other FSMs in the set. The other FSMs in the set are added iteratively in the design in their respective order.

In an FSM, a specific state is chosen only if a particular set of input bits (i.e., 1’s or 0’s) are present. Hence, the percentage of 1’s and 0’s together in an input line acts as information content as shown in Table 5 (the selected input lines to match with base_skt are highlighted). Similarly, the output is always defined by “1.” Hence, the percentage of 1’s in an output line serves as information content as shown in Tables 6 and 7 (the selected output lines to match with base_skt are highlighted).

At the first phase of Improved-IGHA, input and state matching are performed together, and optimal assignments (with respect to base_skt) are made. It is presented in Table 5. All the recon_skt states are mapped onto base_skt states in their respective order. Output matching (with respect to base_skt) is performed iteratively by Bitwise-XOR operations. It is presented in Tables 6 and 7. Then, after updating the descriptions of constituting FSMs, the state assignment using logarithmic barrier function based gradient descent approach is performed.

To present a comparative analysis of the total computation time required by IGHA [6] and Improved-IGHA, an inbuilt feature in MATLAB named “stopwatch timer” is used. It evaluates the elapsed time (i.e., the execution time between the starting and stopping of a function). As evident from the literature [6], linear assignment problems (LAPs) are solved several times by IGHA to perform matchings among all generated combinations to add recon_skt_B $\in \{recon_skt_1, \ldots, recon_skt_B\}$ iteratively. The convergence period of IGHA to solve a single LAP ranges from 0.03 ms to 0.6 ms. Hence, the total elapsed time taken by IGHA (i.e., $t_{IG}$) is given in (32). The convergence time for the state assignment using LBF-based gradient descent approach (i.e., $t_{SE}$) to add recon_skt_B $\in \{recon_skt_1, \ldots, recon_skt_B\}$ iteratively is given in Table 8. Therefore, the total elapsed time taken by Improved-IGHA (i.e., $t_{Proposed}$) is an addition of $t_{IG}$ and $t_{SE}$ (from Figure 3). It is presented in Table 8.

\[
\text{Total elapsed time for IGHA} = \sum_{\text{recon_skt}_B} \left(1 + E\right) \left(M_{base} \times M_{recon}\right)
\]

\[
\times \text{Elapsed time for IGHA to solve a single LAP}
\]

\[
\text{s.t. } E = \begin{cases} L_{base}^P_{recon} \text{ if } L_{base} \geq L_{recon}; \\ L_{recon}^P_{base} \text{ if } L_{base} < L_{recon}; \end{cases}
\]

The experimental results presented in Table 8 illustrate that the total computation time required by IGHA is far higher than the convergence time for the proposed state assignment technique (i.e., $t_{IG} \gg t_{SE}$). Therefore, the total computation time required by Improved-IGHA is equivalent to the total computation time needed by IGHA (i.e., $t_{Proposed} \equiv t_{IG}$).
| FSM | No. of I/P | Input lines with their information content                                                                 | Matched with base \( \mathcal{X} \) | No. of state |
|-----|-----------|----------------------------------------------------------------------------------------------------------------|--------------------------------------|--------------|
| 1494| 8         | \( x_1(99.6\%) \), \( x_2(10\%) \), \( x_3(25.6\%) \), \( x_4(24.8\%) \), \( x_5(12.4\%) \), \( x_6(66\%) \), \( x_7(41.6\%) \), \( x_8(20.4\%) \) | \( x_1, x_3 \), \( x_4, x_5 \), \( x_6, x_7 \), \( x_8 \) | 48           |
| 832 | 18        | \( x_1(2.04\%) \), \( x_2(6.93\%) \), \( x_3(2.44\%) \), \( x_4(4.48\%) \), \( x_5(70.61\%) \), \( x_6(1.63\%) \), \( x_7(14.28\%) \), \( x_8(9.79\%) \), \( x_9(8.97\%) \), \( x_{10}(8.16\%) \), \( x_{11}(8.57\%) \), \( x_{12}(6.12\%) \), \( x_{13}(4.08\%) \), \( x_{14}(4.08\%) \), \( x_{15}(1.63\%) \), \( x_{16}(1.63\%) \), \( x_{17}(59.18\%) \), \( x_{18}(81.63\%) \), \( x_{19}(96.73\%) \), \( x_{20}(91.5\%) \), \( x_{21}(0\%) \), \( x_{22}(0\%) \), \( x_{23}(2.61\%) \), \( x_{24}(5.22\%) \), \( x_{25}(12.41\%) \), \( x_{26}(49.01\%) \), \( x_{27}(77.12\%) \) | \( x_9, x_{11} \), \( x_{15}, x_{17}, x_{20} \), \( x_{19}, x_{22}, x_{24} \) | 25           |
| 208 | 11        | \( x_1, x_2, x_3, x_4, x_5, x_6, x_7 \)                                                                                   | \( x_1, x_3 \), \( x_4, x_5 \), \( x_6, x_7 \), \( x_8 \) | 18           |
| 386 | 7         | \( x_1, x_2, x_3, x_4, x_5, x_6, x_7 \)                                                                                   | \( x_1, x_3 \), \( x_4, x_5 \), \( x_6, x_7 \), \( x_8 \) | 13           |
| sand| 11        | \( x_1(52.17\%) \), \( x_2(52.17\%) \), \( x_3(52.17\%) \), \( x_4(24.45\%) \), \( x_5(3.26\%) \), \( x_6(18.47\%) \), \( x_7(26.63\%) \), \( x_8(1.08\%) \), \( x_{10}(79.89\%) \), \( x_{11}(19.56\%) \) | \( x_4, x_5 \), \( x_9, x_{10} \), \( x_{11}, x_{12} \), \( x_{13} \) | 32           |
| mc  | 3         | \( x_1, x_2, x_3 \)                                                                                                         | \( \neg x_1, \neg x_3 \), \( \neg x_2 \) | 4            |
| styx| 9         | \( x_1(92.16\%) \), \( x_2(4.81\%) \), \( x_3(48.79\%) \), \( x_4(69.87\%) \), \( x_5(68.67\%) \), \( x_6(39.15\%) \), \( x_7(4.81\%) \), \( x_8(5.42\%) \), \( x_9(3.61\%) \) | \( x_{21}, x_{24} \), \( x_{51}, x_{53} \), \( x_{81}, x_{83}, x_{1} \) | 30           |
| cse | 7         | \( x_1, x_2, x_3, x_4, x_5, x_7 \)                                                                                             | \( x_{51}, x_{21} \), \( x_{71}, x_{31} \), \( x_{41}, x_{1} \), \( x_5 \) | 16           |
| ex6 | 5         | \( x_1, x_2, x_3, x_4, x_5 \)                                                                                                 | \( \neg x_1, \neg x_2 \), \( \neg x_2 \) | 8            |
Table 5: Continued.

| FSM     | No. of I/P | Input lines with their information content | Matched with base_ckt | No. of state |
|---------|------------|---------------------------------------------|------------------------|--------------|
| planet1 | 7          | $x_1, x_2, x_3, x_4, x_5, x_6, x_7$         | $x_5, x_1, x_3, x_4, x_2$ | 48           |
| s1488   | 8          | $x_1(98.8\%), x_2(12.74\%), x_3(23.5\%), x_4(23.9\%), x_5(16.33\%), x_6(65.73\%), x_7(40.23\%), x_8(18.32\%)$ | $x_1, x_3, x_4, x_5, x_7, x_8$ | 48           |

![Figure 4: The pictorial representation of state transitions for base_ckt and recon_ckt.](image)

Convergence plot for the state assignment using logarithmic barrier function based gradient descent approach after adding the last constituting FSM in the proposed architecture is presented in Figure 5. It starts by taking binary state codes as an initial code. The cost offered to the proposed architecture is calculated by (2). It converges to 200 iterations. The cost is reduced from 1028 to 923 as shown in Figure 5. Consequently, at 200th iteration, the following state codes are obtained: $S0 \rightarrow 010111, S1 \rightarrow 000000, S2 \rightarrow 000111, S3 \rightarrow 110000, S4 \rightarrow 010100, S5 \rightarrow 110101, S6 \rightarrow 000110, S7 \rightarrow 011101, S8 \rightarrow 001100, S9 \rightarrow 011010, S10 \rightarrow 011110, S11 \rightarrow 001110, S12 \rightarrow 010110, S13 \rightarrow 111011, S14 \rightarrow 000011, S15 \rightarrow 100110, S16 \rightarrow 110111, S17 \rightarrow 001010, S18 \rightarrow 011100, S19 \rightarrow 100001, S20 \rightarrow 101001, S21 \rightarrow 110010, S22 \rightarrow 100000, S23 \rightarrow 001000, S24 \rightarrow 000111, S25 \rightarrow 101101, S26 \rightarrow 010011, S27 \rightarrow 101010, S28 \rightarrow 110001, S29 \rightarrow 001011, S30 \rightarrow 111010, S31 \rightarrow 011111, S32 \rightarrow 000101, S33 \rightarrow 000100, S34 \rightarrow 111101, S35 \rightarrow 000001, S36 \rightarrow 001101, S37 \rightarrow 101000, S38 \rightarrow 000010, S39 \rightarrow 100110, S40 \rightarrow 100010, S41 \rightarrow 011001, S42 \rightarrow 100010, S43 \rightarrow 001001, S44 \rightarrow 010001, S45 \rightarrow 100101, S46 \rightarrow 101111, and S47 \rightarrow 010010.
| FSM | No. of O/P | Output lines with their information content | Matched with base.skt |
|-----|-----------|--------------------------------------------|----------------------|
| s1494 | 19 | \(y_1(24.8\%) \), \(y_2(4.8\%) \), \(y_3(5.2\%) \), \(y_4(3.2\%) \), \(y_5(2.4\%) \), \(y_6(2.4\%) \), \(y_7(15.2\%) \), \(y_8(25.2\%) \), \(y_9(1.6\%) \), \(y_{10}(6.4\%) \), \(y_{11}(87.2\%) \), \(y_{12}(40.4\%) \), \(y_{13}(32.8\%) \), \(y_{14}(70.4\%) \), \(y_{15}(38.4\%) \), \(y_{16}(18.4\%) \), \(y_{17}(70\%) \), \(y_{18}(31.2\%) \), \(y_{19}(49.2\%) \) | \(y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{16}, y_{17}, y_{19} \) |
| s832 | 19 | \(y_1(5.71\%) \), \(y_2(2.44\%) \), \(y_3(1.22\%) \), \(y_4(1.63\%) \), \(y_5(2.44\%) \), \(y_6(0.81\%) \), \(y_7(2.44\%) \), \(y_8(73.06\%) \), \(y_9(0.81\%) \), \(y_{10}(0.81\%) \), \(y_{11}(5.3\%) \), \(y_{12}(2.44\%) \), \(y_{13}(0.81\%) \), \(y_{14}(1.63\%) \), \(y_{15}(6.12\%) \), \(y_{16}(0.81\%) \), \(y_{17}(1.63\%) \), \(y_{18}(2.44\%) \), \(y_{19}(41.22\%) \) | \(y_{10}, y_{14}, y_{9}, y_{7} \) |
| s208 | 2 | \(y_1, y_2 \) | \(y_1, y_2, y_3 \) |
| planet | 19 | \(y_1(54.78\%) \), \(y_2(23.47\%) \), \(y_3(69.56\%) \), \(y_4(16.52\%) \), \(y_5(32.17\%) \), \(y_6(73.91\%) \), \(y_7(26.08\%) \), \(y_8(28.69\%) \), \(y_9(91.3\%) \), \(y_{10}(4.34\%) \), \(y_{11}(1.73\%) \), \(y_{12}(22.6\%) \), \(y_{13}(11.3\%) \), \(y_{14}(2.6\%) \), \(y_{15}(3.47\%) \), \(y_{16}(1.73\%) \), \(y_{17}(3.47\%) \), \(y_{18}(3.47\%) \), \(y_{19}(20\%) \) | \(y_9, y_{13}, y_7 \) |
| s386 | 7 | \(y_1, y_2, y_3 \), \(y_4, y_5, y_6, y_7 \) | \(y_9, y_{12}, y_3 \) |
| sand | 9 | \(y_1(22.28\%) \), \(y_2(36.41\%) \), \(y_3(16.84\%) \), \(y_4(62.5\%) \), \(y_5(15.76\%) \), \(y_6(8.15\%) \), \(y_7(17.39\%) \), \(y_8(1.63\%) \), \(y_9(3.26\%) \) | \(y_9, y_{13}, y_7 \) |
| mc | 5 | \(y_1, y_2, y_3 \), \(y_4, y_5 \) | \(y_9, y_{13} \) |
| styr | 10 | \(y_1(15.66\%) \), \(y_2(33.73\%) \), \(y_3(25.9\%) \), \(y_4(3.01\%) \), \(y_5(8.43\%) \), \(y_6(7.22\%) \), \(y_7(5.42\%) \), \(y_8(1.145\%) \), \(y_9(3.614\%) \), \(y_{10}(4.819\%) \) | \(y_9, y_{13} \) |
| cse | 7 | \(y_1, y_2, y_3 \), \(y_4, y_5, y_6, y_7 \) | \(y_9, y_{13} \) |
| ex6 | 8 | \(y_1(58.82\%) \), \(y_2(29.41\%) \), \(y_3(55.88\%) \), \(y_4(29.41\%) \), \(y_5(50\%) \), \(y_6(26.47\%) \), \(y_7(5.88\%) \), \(y_8(23.52\%) \) | \(y_9, y_{13} \) |
Table 7: The information content for output lines of MCNC FSM Benchmarks and their matching with output lines of base.skl.

| FSM      | No. of O/P | Output lines with their information content                                      | Matched with base.skl |
|----------|------------|----------------------------------------------------------------------------------|-----------------------|
| planet1  | 19         | \(y_1(54.78\%), \ y_2(23.47\%), \ y_3(69.56\%), \ y_4(16.52\%), \ y_5(32.17\%), \ y_6(73.91\%),\)  |                       |
| s1488    | 19         | \(y_1(2.39\%), \ y_2(2.78\%), \ y_3(1.59\%), \ y_4(15.13\%), \ y_5(71.31\%), \ y_6(3.98\%), \ y_7(51.39\%),\)  |                       |
|          |            | \(y_8(6.3\%), \ y_9(16.7\%), \ y_{10}(37.1\%), \ y_{11}(7.05\%), \ y_{12}(24.3\%), \ y_{13}(87.6\%),\)      | \(y_{14}(31.1\%), \ y_{15}(25.4\%), \ y_{16}(31.4\%),\)  |
|          |            | \(y_{17}(1.73\%), \ y_{18}(3.47\%), \ y_{19}(3.47\%),\)                                      |                       |

Table 8: Comparative analysis of the total computation time required by IGHA [6] and Improved-IGHA.

| Iteration No. | FSM included in the specific iteration | Total elapsed time for IGHA [6] (Hrs) \(t_{IG} \) | Total elapsed time for state encoding tech. (ms) \(t_{SE} \) | Total elapsed time for Improved-IGHA (Hrs) \(t_{Proposed} = t_{IG} + t_{SE} \) |
|---------------|----------------------------------------|---------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| 0\(^{th}\)    | s1494                                  | 0                                           | 296.529                                         | 0                                               |
| 1\(^{st}\)    | s832                                   | 25.34                                       | 214.468                                         | 25.34                                           |
| 2\(^{nd}\)    | s208                                   | 18.57                                       | 156.062                                         | 18.57                                           |
| 3\(^{rd}\)    | planet                                 | 48.65                                       | 338.560                                         | 48.65                                           |
| 4\(^{th}\)    | s386                                   | 13.17                                       | 182.808                                         | 13.17                                           |
| 5\(^{th}\)    | sand                                   | 32.43                                       | 293.894                                         | 32.43                                           |
| 6\(^{th}\)    | mc                                     | 0.182                                       | 148.784                                         | 0.182                                           |
| 7\(^{th}\)    | sty                                    | 30.409                                      | 249.509                                         | 30.409                                          |
| 8\(^{th}\)    | cse                                    | 16.21                                       | 387.923                                         | 16.21                                           |
| 9\(^{th}\)    | ex6                                    | 4.38                                        | 167.144                                         | 4.38                                            |
| 10\(^{th}\)   | planet1                                | 48.544                                      | 312.809                                         | 48.544                                          |
| 11\(^{th}\)   | s1488                                  | 48.654                                      | 326.406                                         | 48.654                                          |

At the last phase of Improved-IGHA, a mutual Bitwise-XOR operation is conducted between the updated descriptions of FSMs. Therefore, the constituting parts of the proposed architecture are created. The individual share of constituent FSMs in the Improved-Reconfigurable FSMIM architecture is determined by the difference between the occupied LUTs in the recent and its previous iteration. After adding all the constituting FSMs in the proposed design (i.e., at the last iteration), the total LUT consumption and operating frequency are obtained. It is presented in Table 9.

Experimental results for the proposed architecture illustrates a significant area reduction by an average of 20.38% and speed improvement by an average of 32.73% over VRMUX [11] during FPGA implementation. It also demonstrates an adequate area reduction by an average of 16.05% and speed improvement by an average of 1.77% over Reconfigurable FSMIM-S architecture [6] during FPGA implementation. When these results are compared with CRMUX [11], a speed improvement by an average of 11.06% is obtained. The proposed architecture requires an average of 58.38% more LUTs as compared with CRMUX [11] during FPGA implementation. When these results are compared respectively.

5. Concluding Remarks

This article furnishes the framework for the Improved-Reconfigurable FSMIM architecture. The Improved-Reconfigurable FSMIM architecture is created by joining the following two parts: (A) Conventional FSMIM architecture and (B) Multiplexer bank (which defines the mode based reconfiguration). An improved version of iterative greedy
Table 9: Implementation of the Improved Reconfigurable FSMIM architecture on the Virtex-6 speed-3 device in an iterative manner.

| Iteration No. | FSM included in the specific iteration | LUTs occupied in the specific iteration | Maximum Operating Frequency (MHz) | Maximum Path Delay (ns) | LUTs occupied by the FSM in the previous iteration |
|---------------|---------------------------------------|----------------------------------------|----------------------------------|------------------------|--------------------------------------------------|
| 0th           | s1494                                 | 40                                     | 831.693                          | 3.898                  | 40                                               |
| 1st           | s832                                  | 97                                     | 803.44                           | 4.288                  | 57                                               |
| 2nd           | s208                                  | 114                                    | 793.583                          | 5.252                  | 17                                               |
| 3rd           | planet                                | 142                                    | 785.326                          | 4.219                  | 28                                               |
| 4th           | s386                                  | 157                                    | 776.863                          | 4.534                  | 15                                               |
| 5th           | sand                                  | 187                                    | 760.88                           | 4.117                  | 30                                               |
| 6th           | mc                                     | 198                                    | 757.237                          | 3.854                  | 11                                               |
| 7th           | styr                                   | 217                                    | 743.431                          | 4.204                  | 19                                               |
| 8th           | cse                                    | 240                                    | 713.929                          | 4.401                  | 23                                               |
| 9th           | ex6                                    | 249                                    | 704.892                          | 4.649                  | 9                                                |
| 10th          | planet1                                | 274                                    | 690.83                           | 4.977                  | 25                                               |
| 11th          | s1488                                 | 293                                    | 676.928                          | 5.151                  | 19                                               |

#LUTs → number of LUTs occupied in ISE

Figure 5: Convergence plot for the state assignment using logarithmic barrier function based gradient descent approach after adding the last constituting FSM in the proposed architecture.

The proposed architecture illustrates a significant area reduction by an average of 20.38% and speed improvement by an average of 32.73% over VRMUX [11] during FPGA implementation. It also demonstrates an adequate area reduction by an average of 16.05% and speed improvement by an average of 1.77% over Reconfigurable FSMIM-S architecture [6] during FPGA implementation. When these results are compared with CRMUX [11], a speed improvement by an average of 11.06% is obtained. The proposed architecture requires an average of 58.38% more LUTs as compared with CRMUX [11] during FPGA implementation. It is the only trade-off for the proposed design.
Further, the proposed architecture will be investigated to develop an efficient architecture for multistage signal processing [1, 2] and circuit testing [5] based applications.

Data Availability
The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest
The authors declare that they have no conflicts of interest.

Acknowledgments
The datasets generated during and/or analyzed during the current study are available in [6] repository [DOI: 10.1155/2018/6831901]. This work is conducted in the Department of ECE, SRM Institute of Science and Technology, Kattankulathur-603203, Chennai, India.

References
[1] 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.
[2] 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.
[3] J. Zheng, W. Gao, D. Wu, and D. Xie, "An efficient VLSI architecture for CBAC of AVS HDTV decoder," Signal Processing: Image Communication, vol. 24, no. 4, pp. 324–332, 2009.
[4] N. I. Rafla and I. Gauba, "A reconfigurable pattern matching hardware implementation using on-chip RAM-based FSM," in Proceedings of the 53rd IEEE International Midwest Symposium on Circuits and Systems, MWSCAS 2010, pp. 49–52, IEEE, Seattle, Wash, USA, August 2010.
[5] Z. Ling, K. Ji-Shun, and Y. Zhi-Qiang, "Virtual scan chains reordering using a RAM-based module for high test compression," Microelectronics Journal, vol. 43, no. 11, pp. 869–872, 2012.
[6] N. Das and P. A. Priya, "FPGA implementation of reconfigurable finite state machine with input multiplexing architecture using hungarian method," International Journal of Reconfigurable Computing, vol. 2018, Article ID 6831901, 15 pages, 2018.
[7] 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, pp. 1–14, 2011.
[8] 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.
[9] L. Jozwiak, A. Slusarczyk, and D. Gawlowski, "Multi-objective optimal FSM state assignment," in Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD’06), pp. 385–396, IEEE, Dubrovnik, Croatia, 2006.
[10] S. Denziak and M. Wisniewski, "FPGA-based state encoding using symbolic functional decomposition," IEEE Electronics Letters, vol. 46, no. 19, pp. 1316–1318, 2010.
[11] R. Senhadji-Navarro 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, pp. 1–15, 2015.
[12] M. Kołopieńczyk, L. Titarenko, and A. Barkalov, "Design of EMB-based moore FSMs," Journal of Circuits, Systems, and Computers, vol. 26, no. 7, pp. 1–23, 2017.
[13] M. Kołopieńczyk, A. Barkalov, and L. Titarenko, "Hardware reduction for RAM-based moore FSMs," in Proceedings of the 7th International Conference on Human System Interactions, HSI 2014, pp. 255–260, IEEE, Costa da Caparica, Portugal, June 2014.
[14] https://people.engr.ncsu.edu/brglez/CBL/benchmark/.
[15] S. Devadas, H.-K. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "Mustang: state assignment of finite state machines targeting multilevel logic implementations," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 7, no. 12, pp. 1290–1300, 1988.
[16] K. Kabył, A. Berrachedi, and É. Sopena, "A note on the cubical dimension of new classes of binary trees," Czechoslovak Mathematical Journal, vol. 65, no. 1, pp. 151–160, 2015.
[17] M. Liu and H.-M. Liu, "Vertex-fault-tolerant cycles embedding on enhanced hypercube networks," Acta Mathematicae Applicatae Sinica, vol. 32, no. 1, pp. 187–198, 2016.
[18] G. Pavai and T. V. Geetha, "New crossover operators using dominance and co-dominance principles for faster convergence of genetic algorithms," Soft Computing, pp. 1–26, 2018.
[19] S. Ganjefar and M. Tofighi, “Optimization of quantum-inspired neural network using memetic algorithm for function approximation and chaotic time series prediction,” *Neurocomputing*, vol. 291, pp. 175–186, 2018.

[20] H. Huang, L. Lv, S. Ye, and Z. Hao, "Particle swarm optimization with convergence speed controller for large-scale numerical optimization," *Soft Computing*, pp. 1–17, 2018.

[21] R. Knobloch, J. Mlynek, and R. Srb, "The classic differential evolution algorithm and its convergence properties," *Applications of Mathematics*, vol. 62, no. 2, pp. 197–208, 2017.

[22] F. E. Curtis, "A penalty-interior-point algorithm for nonlinear constrained optimization," *Mathematical Programming Computation*, vol. 4, no. 2, pp. 181–209, 2012.

[23] P. Armand and R. Omheni, "A mixed logarithmic barrier-augmented Lagrangian method for nonlinear optimization," *Journal of Optimization Theory and Applications*, vol. 173, no. 2, pp. 523–547, 2017.

[24] W. Murray and K.-M. Ng, "An algorithm for nonlinear optimization problems with binary variables," *Computational Optimization and Applications*, vol. 47, no. 2, pp. 257–288, 2010.

[25] E. M. Soler, V. A. De Sousa, and G. R. M. Da Costa, "A modified primal-dual logarithmic-barrier method for solving the optimal power flow problem with discrete and continuous control variables," *European Journal of Operational Research*, vol. 222, no. 3, pp. 616–622, 2012.

[26] R. García Otero and A. Iusem, "A proximal method with logarithmic barrier for nonlinear complementarity problems," *Journal of Global Optimization*, vol. 64, no. 4, pp. 663–678, 2016.

[27] L. Menniche and D. Benterki, "A logarithmic barrier approach for linear programming," *Journal of Computational and Applied Mathematics*, vol. 312, pp. 267–275, 2016.

[28] R. Shen, Z. Meng, C. Dang, and M. Jiang, "Algorithm of barrier objective penalty function," *Numerical Functional Analysis and Optimization*, vol. 38, no. 11, pp. 1–17, 2017.

[29] I. Chakroun, T. Haber, and T. J. Ashby, "SW-SGD: the sliding window stochastic gradient descent algorithm," *Procedia Computer Science*, vol. 108, pp. 2318–2322, 2017.

[30] A. Senov and O. Granichin, "Projective approximation based gradient descent modification," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3899–3904, 2017.

[31] B. Mu, J. Ren, and S. Yuan, "An efficient approach based on the gradient definition for solving conditional nonlinear optimal perturbation," *Mathematical Problems in Engineering*, vol. 2017, Article ID 3208431, 10 pages, 2017.
