electronics

Article

Mapping Outputs and States Encoding Bits to Outputs Using Multiplexers in Finite State Machine Implementations

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

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

Abstract: This paper proposes a new technique for implementing Finite State Machines (FSMs) in Field Programmable Gate Arrays (FPGAs). The proposed approach extends the called column compaction in two ways. First, it is applied to the state-encoding bits in addition to the outputs, allowing a reduction in the number of logic functions required both by the state transition function and by the output function. Second, the technique exploits the dedicated multiplexers usually included in FPGAs to increase the number of columns that can be compacted. Unlike conventional state-encoding techniques, the proposed approach reduces the number of logic functions instead of their complexity. An Integer Linear Programming (ILP) formulation that maximizes the number of compacted columns has been proposed. In order to evaluate the effectiveness of the proposed approach, experimental results using standard benchmarks are presented. In most cases, the proposed approach reduces the number of used Look-Up Tables (LUTs) with respect to the conventional FSM implementation.

Keywords: finite state machine; column compaction; state encoding; dedicated multiplexer; synthesis; FPGA

Citation: Senhadji-Navarro, R.; Garcia-Vargas, I. Mapping Outputs and States Encoding Bits to Outputs Using Multiplexers in Finite State Machine Implementations. *Electronics* **2023**, *12*, 502. https://doi.org/10.3390/electronics12030502

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

Received: 14 December 2022
Revised: 12 January 2023
Accepted: 14 January 2023
Published: 18 January 2023

1. Introduction

Finite State Machines (FSMs) are probably the most widely used component in digital design. They allow one to model any sequential circuit, particularly control units. Optimizing FSM implementations in terms of area, speed or power consumption is essential to meet the design constraints demanded by applications. For this reason, the synthesis of FSMs has received the attention of researchers, designers and EDA tool developers for decades [1–4]. One of the most active fields has been the encoding of the FSM states to achieve optimal implementations of the state transition and output functions [5–20].

Other techniques are focused on the optimal assignment of don't care outputs, which are very usual because the values of some outputs do not matter in some states. The technique called column compaction exploits these don't cares to reduce the number of logic functions that must be implemented [21–25]. The aim of this technique is to assign values of zero or one to the don't cares in order to achieve that two or more outputs are equal to each other, and so only one logic function must be implemented for each group of equivalent outputs.

Our technique extends column compaction in two ways. First, it extends the concept of column compaction to the state-encoding bits. To the best knowledge of the authors' knowledge, column compaction has never been applied to the state transition function. In addition to assign values to don't cares to find equivalent outputs, our technique also encodes the states to achieve that some encoding bits are equal to some outputs. Therefore, it can be viewed as an extension of column compaction that is applied to the state transition function, in addition to the output function. This allows one to reduce the number of logic functions required to implement both functions. We will refer to the one-to-one correspondence between state-encoding bits and outputs as direct mapping of state-encoding bits. For analogy, column compaction will be referred to as direct mapping of outputs.

Second, the proposed technique exploits the dedicated 2:1 multiplexers usually included in Field Programmable Gate Arrays (FPGAs), which allow one to combine the outputs of two Look-Up Tables (LUTs) [26]. In order to reduce the number of LUTs used, the technique searches pairs of outputs that can be combined by means of dedicated multiplexers to generate the next state-encoding bits or other outputs. Therefore, the proposed technique also extends the concept of compaction to that of mapping via multiplexers. Each multiplexer is controlled by a bit of the present state. Due to the fact that the state-encoding bits are control inputs of the multiplexers and outputs of some of them, state encoding has a great influence on the results that can be obtained. Unlike conventional state-encoding techniques, the proposed approach reduces the number of logic functions instead of their complexity. Our technique is based on an Integer Linear Programming (ILP) formulation, which allows a solver to find the optimal mapping (or suboptimal, when the execution time is limited).

As a conclusion, the proposed technique uses three mechanisms to reduce the number of logic functions: direct mapping of outputs (i.e., column compaction), direct mapping of state-encoding bits and mapping via multiplexers, which are each modeled by a unique ILP formulation. Unlike the other two mechanisms, which are presented for the first time, column compaction is a classical technique. However, the novelty lies in the fact that it is based on a ILP formulation instead of heuristics [22], which allows one to obtain the optimal value, or in the worst case, to find a bound of the error corresponding to a non-optimal solution.

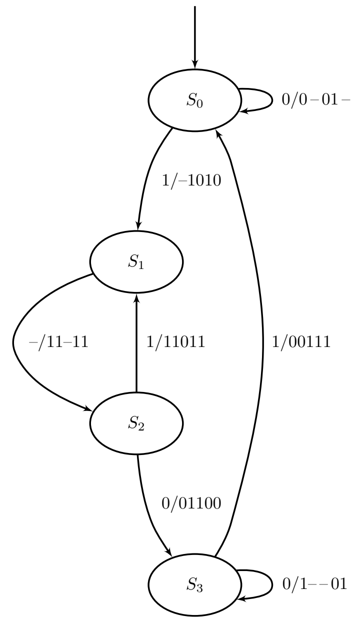
The remainder of this article is organized as follows. In Section 2, brief background about FSMs and FPGA devices is presented. Section 3 describes the fundamentals of the proposed technique and its architecture. Section 4 presents the proposed ILP formulation. Section 5 shows the obtained experimental results. Finally, conclusions are drawn in Section 6.

2. Background

Let $\mathcal{F} = (S, t, o, S_0)$ be a FSM, where S represents the set of states; $t : S \times \{0,1\}^r \rightarrow S$, the state transition function; $o : S \times \{0,1\}^r \rightarrow \{0,1,-\}^m$, the output function; and S_0, the reset state. The parameters r and m represent the numbers of inputs and outputs, respectively.

The State Transition Diagram (STD) is commonly used to show the behavior of a FSM. Figure 1 shows an example of FSM, whose STD is shown in Figure 1a. For a proper description of the proposed technique, it is more convenient to represent the FSM in the form of a State Transition Table (STT). As Figure 1b,c show, a STT is a table whose rows represent the transitions of the FSM as the 4-tuple (in, ps, ns, out), where in represents the inputs; ps, the present state; ns, the next state; and out, the outputs. The symbol "–" represents a don't care value. A symbolic STT shows the states as symbols (see Figure 1b), whereas a codified STT shows the states once they have been codified (see Figure 1c).

FPGAs are semiconductor devices that are based around a matrix of Configurable Logic Blocks (CLBs) connected via programmable interconnects. In current Xilinx FPGA devices, each CLB is composed by two slices, and each one of them includes four LUTs. In addition, each slice includes three dedicated 2:1 multiplexers: two called F7 and one called F8. Each F7 combines the outputs of two LUTs whereas F8 combines the outputs of two F7s. Dedicated multiplexers can be used to create general-purpose functions of up to 13 inputs in 2 LUTs or 27 inputs in 4 LUTs (i.e., in one slice) [26].

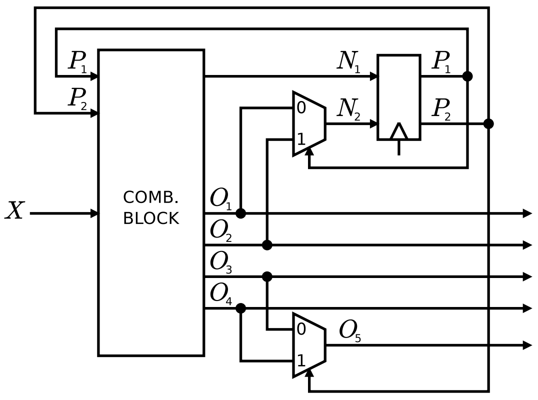


Input	**Present state**	**Next state**	**Outputs**				
X			O_1	O_2	O_3	O_4	O_5
0	S_0	S_0	0	–	0	1	–
1	S_0	S_1	–	1	0	1	0
–	S_1	S_2	1	1	–	1	1
0	S_2	S_3	0	1	1	0	0
1	S_2	S_1	1	1	0	1	1
0	S_3	S_3	1	–	–	0	1
1	S_3	S_0	0	0	1	1	1

(**b**)

Input	**Present state**		**Next state**		**Outputs**				
X	P_1	P_2	N_1	N_2	O_1	O_2	O_3	O_4	O_5
0	0	0	0	0	0	–	0	1	**0**
1	0	0	1	1	**1**	1	0	1	0
–	1	1	0	1	1	1	–	1	1
0	0	1	1	0	0	1	1	0	0
1	0	1	1	1	1	1	0	1	1
0	1	0	1	0	1	**0**	**1**	0	1
1	1	0	0	0	0	0	1	1	1

(**a**) (**c**)



(**d**)

Figure 1. Example of the mapping of state-encoding bits and outputs to outputs: (**a**) STD, (**b**) symbolic SST, (**c**) encoded STT and (**d**) implementation.

3. Mapping Outputs and States Encoding Bits to Outputs by Means of Multiplexers

The aim of the proposed technique is to generate FSM outputs or the next state-encoding bits from FSM outputs without using extra LUTs. Both direct mapping and mapping via multiplexers are applied for this purpose. Mapping via multiplexers benefits from dedicated multiplexers, avoiding the use of extra LUTs. The advantage of direct mapping is that it allows one to reduce the number logic levels (i.e., to remove the logic levels corresponding to the dedicated multiplexers) and to improve the routing; therefore, this mapping can increase the speed. To sum up, including mapping via multiplexer allows one to find solutions with a more mapped signals. However, the proposed ILP formulation gives priority to direct mapping.

By exploiting the usual strong relationship between the transition and output functions, the proposed technique codifies the states to achieve that the maximum number of signals (whether outputs or state-encoding bits) are mapped. In addition to encode the states, the proposed technique assigns values to the don't care outputs to achieve its objective. There is a key difference between the proposed technique and the conventional state-encoding approaches [5–20] that justifies its novelty. State-encoding algorithms assign a different code to each FSM state with the purpose of simplifying the complexity of the transition and output functions. On the other hand, the proposed approach reduces the number of logic functions instead of its complexity. Compared with the conventional state-encoding approaches, assigning don't care outputs and codifying states in a non-optimal way in terms of function complexity can result in an increase in the number of LUTs required by each logic function. However, a reduction in the number of logic functions achieved by the proposed technique can compensate for such an increase, resulting in a lesser overall LUT count.

Figure 2 shows the general architecture to implement the FSMs obtained from the proposed technique. For simplicity, the signals generated from a single output (i.e., by applying direct mapping) have been excluded from the figure. The combinational block implements the logic corresponding to the unmapped bits of the next state and the unmapped outputs, so it partially implements the state transition and output functions. The mapped signals are generated from unmapped outputs by means of multiplexers controlled by a bit of the present state (which is not necessarily different from those of other multiplexers). The larger the number of mapped signals, the simpler the combinational block. This is due to the fact that the number of logic functions is reduced.

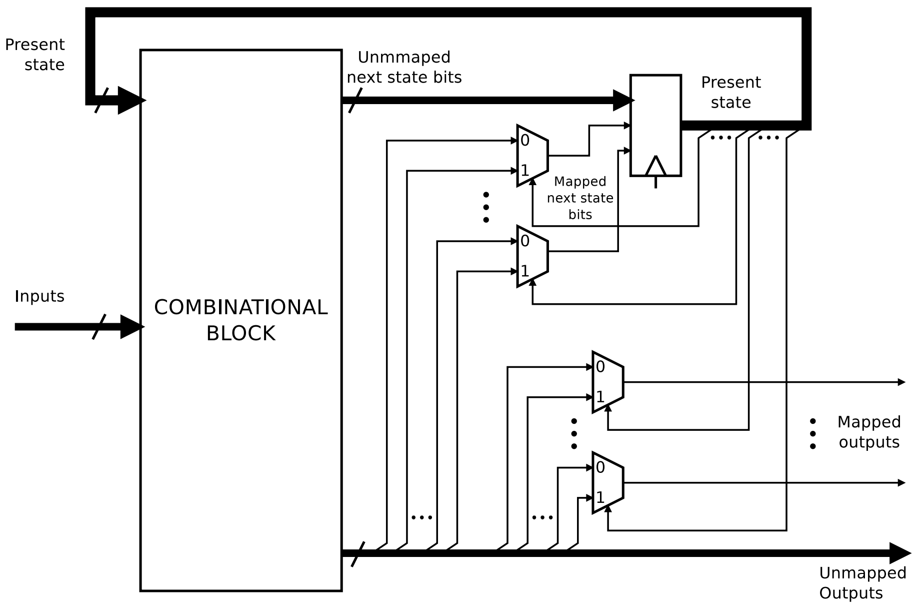


Figure 2. General architecture for mapping state-encoding bits and outputs to outputs.

Figure 1c shows the STT of the FSM example (Figure 1b) after applying a mapping of state-encoding bits and outputs to outputs. After codifying the states and assigning some don't care outputs, the next state-encoding bit N_2 can be mapped to outputs O_1 and O_2 by means of a multiplexer controlled by the present state-encoding bit P_1, which selects O_1 if $P_1 = 0$ and O_2 if $P_1 = 1$ (see Figure 1b,c). Note that Figure 1c shows in bold the don't care outputs that have been assigned. Similarly, O_5 can be mapped to O_3 and O_4 by means of a multiplexer controlled by P_2, which selects O_3 if $P_2 = 0$ and O_4 if $P_2 = 1$. This mapping allows one to reduce the number of logic functions that must be implemented in LUTs from seven to five (see Figure 1d). By supposing LUTs of four inputs, the implementation obtained by the proposed technique requires five LUTs, whereas the conventional FSM implementation requires seven LUTs, independently of the state encoding used.

We define optimal mapping of outputs and state-encoding bits to outputs (OMOSEBO) as the problem of finding the mapping that maximizes the number of mapped signals (i.e., that minimizes the number of logic functions that are required to implement the output and state transition functions).

We have proposed an ILP formulation for this problem, which prioritizes direct mapping over mapping via multiplexers. Some constraints are imposed by the Xilinx FPGA architecture itself. As the output of each LUT is physically connected to a unique F7 multiplexer, routing the output of the LUT to a different multiplexer requires an additional LUT (called route-thru). F8 multiplexers present similar problems because the output of each F7 is physically connected to a unique F8 multiplexer. To prevent these situations, the ILP formulation must ensure that a FSM output cannot be used to generate more than one signal via multiplexer. However, although dedicated multiplexers F7 and F8 can be chained, allowing the multiplexing of up to four signals, we have limited our approach to a unique number of multiplexers to avoid increasing the complexity of the proposed ILP formulation. Therefore, the solver must exclude candidate solutions in which outputs mapped via multiplexers are inputs of other multiplexers.

4. Integer Linear Programming Formulation

Let $S = \{S_1, S_2, \ldots, S_e\}$ be the set of states of a FSM. Let us suppose that O_i for $i = 1, \ldots, m$ represents each output of the FSM. Let us consider the tuple $(T_1, T_2, \ldots, T_q)$, where $T_i \equiv (S_j, S_k) \in S \times S$, and S_j and S_k represent the present and next state, respectively, of the transition corresponding to i-th row of the STT. Let us consider that P_i and N_i represent the i-th encoding bit of the present and the next state, respectively. Let us suppose that $O^i_j \in \{0, 1, -\}$ represents the value of the output O_j for the transition corresponding to the i-th row of the STT.

The ILP formulation requires the following binary variables:

- $f_{i,j,k,r}$ for all i, j, k, r, which is equal to one if and only if N_i can be generated from O_j and O_k via a multiplexer controlled by P_r so that $N_i = O_j$ for all transitions in which $P_r = 0$, and $N_i = O_k$ for the remaining (N_i is therefore mapped to O_j and O_k).
- $g_{i,j,k,r}$ for all i, j, k, r, which is equal to one if and only if O_i can be generated from O_j and O_k via a multiplexer controlled by P_r so that $O_i = O_j$ for all transitions in which $P_r = 0$, and $O_i = O_k$ for the remaining (O_i is therefore mapped to O_j and O_k).
- $h_{i,j}$ for all i, j, which represents the value of the O_j for the transition corresponding to the i-th row of the STT. Note that $h_{i,j}$ must be equal to O^i_j for all i, j such that $O^i_j \in \{0, 1\}$ (i.e., O^i_j is not a don't care); however, the remainder values of $h_{i,j}$ will be determined by the solver.
- $c_{i,j}$ for all i, j, which represents the j-th bit of the encoding of S_i.
- $d_{i,j,k}$ for all i, j, k such that $j < k$, which is equal to one if and only if i-th bit of the encoding of S_j and that of S_k are different.

Note that if $f_{i,j,j,r} = 1$, then there exits a direct mapping of N_i to O_j (so the value of r is not relevant). Similarly, if $g_{i,j,j,r} = 1$, then there exists a direct mapping of O_i to O_j.

The objective of the OMOSEBO problem is to find a mapping of outputs and state-encoding bits that maximize the following tuple in lexicographical order:

$$\Big(\sum_{u,v,t,r} f_{u,v,t,r} + \sum_{\{w,v,t,r|v\neq t\}} g_{w,v,t,r} + \sum_{\{w,v,r|v>w\}} g_{w,v,v,r}, \sum_{u,v,r} f_{u,v,v,r} + \sum_{\{w,v,r|v>w\}} g_{w,v,v,r}\Big) \quad (1)$$

The first element allows one to maximize the number of mapped signals; the second one prioritizes the direct mapping over the mapping via multiplexer. The terms $g_{w,v,v,r}$ include only the cases in which $v > w$ to avoid counting twice one output directly mapped to another (e.g., if $g_{w,v,v,r} = 1$ and $g_{v,w,w,r'} = 1$, then only O_v or O_w can be removed in the corresponding implementation).

Encoding e states requires $d = \lceil \log_2 e \rceil$ bits; therefore, $z = d + m$ is the total number of signals (either outputs or state-encoding bits). As $\sum_{u,v,t,r} f_{u,v,t,r} + \sum_{\{w,v,t,r|v\neq t\}} g_{w,v,t,r} + \sum_{\{w,v,r|v>w\}} g_{w,v,v,r}$ represent the number of mapped signals, it is obvious that $\sum_{u,v,t,r} f_{u,v,t,r} + \sum_{\{w,v,t,r|v\neq t\}} g_{w,v,t,r} + \sum_{\{w,v,r|v>w\}} g_{w,v,v,r} < z$. We use this property to define the weight $\alpha = 1 + \frac{1}{z}$ for the terms corresponding to direct mapping in order to model the multiobjective function. The ILP formulation of the OMOSEBO problem can be expressed as follows:

Maximize

$$\sum_{\{u,v,t,r|v\neq t\}} f_{u,v,t,r} + \alpha \sum_{u,v,r} f_{u,v,v,r} + \sum_{\{w,v,t,r|v\neq t\}} g_{w,v,t,r} + \alpha \sum_{\{w,v,r|v>w\}} g_{w,v,v,r} \tag{2}$$

subject to

$$-f_{u,v,t,r} + c_{i,r} - h_{j,v} + c_{k,u} \geq -1 \quad \forall u,v,t,r,T_j \equiv (S_i,S_k) \tag{3}$$
$$f_{u,v,t,r} - c_{i,r} - h_{j,v} + c_{k,u} \leq 1 \quad \forall u,v,t,r,T_j \equiv (S_i,S_k) \tag{4}$$
$$-f_{u,v,t,r} - c_{i,r} - h_{j,t} + c_{k,u} \geq -2 \quad \forall u,v,t,r,T_j \equiv (S_i,S_k) \tag{5}$$
$$f_{u,v,t,r} + c_{i,r} - h_{j,t} + c_{k,u} \leq 2 \quad \forall u,v,t,r,T_j \equiv (S_i,S_k) \tag{6}$$
$$-g_{w,v,t,r} + c_{i,r} - h_{j,v} + h_{i,j,w} \geq -1 \quad \forall w,v,t,r,T_j \equiv (S_i,S_k) \tag{7}$$
$$g_{w,v,t,r} - c_{i,r} - h_{j,v} + h_{i,j,w} \leq 1 \quad \forall w,v,t,r,T_j \equiv (S_i,S_k) \tag{8}$$
$$-g_{w,v,t,r} - c_{i,r} - h_{j,t} + h_{i,j,w} \geq -2 \quad \forall w,v,t,r,T_j \equiv (S_i,S_k) \tag{9}$$
$$g_{w,v,t,r} + c_{i,r} - h_{j,t} + h_{i,j,w} \leq 2 \quad \forall w,v,t,r,T_j \equiv (S_i,S_k) \tag{10}$$
$$\sum_{v,t,r} f_{u,v,t,r} \leq 1 \quad \forall u \tag{11}$$
$$\sum_{v,t,r} g_{w,v,t,r} \leq 1 \quad \forall w \tag{12}$$
$$\sum_{\{u,t,r|t\neq v\}} f_{u,v,t,r} + \sum_{\{u,t,r|t\neq v\}} f_{u,t,v,r} + \sum_{\{w,t,r|t\neq v\}} g_{w,v,t,r} + \sum_{\{w,t,r|t\neq v\}} g_{w,t,v,r} \leq 1 \quad \forall vs. \tag{13}$$
$$g_{w,w,t,r} = 0 \quad \forall w,t,r \tag{14}$$
$$g_{w,t,w,r} = 0 \quad \forall w,t,r \tag{15}$$
$$(g_{w,v,t,r} - 1)m + \sum_{\{v',t',r'|t'\neq v'\}} g_{v,v',t',r'} \leq 0 \quad \forall w,v,t \neq v,r \tag{16}$$
$$(g_{w,t,v,r} - 1)m + \sum_{\{v',t',r'|t'\neq v'\}} g_{v,v',t',r'} \leq 0 \quad \forall w,v,t \neq v,r \tag{17}$$
$$(f_{u,v,t,r} - 1)m + \sum_{\{v',t',r'|t'\neq v'\}} g_{v,v',t',r'} \leq 0 \quad \forall u,v,t \neq v,r \tag{18}$$
$$(f_{u,t,v,r} - 1)m + \sum_{\{v',t',r'|t'\neq v'\}} g_{v,v',t',r'} \leq 0 \quad \forall u,v,t \neq v,r \tag{19}$$
$$(g_{w,v,v,r} - 1)z + \sum_{w',v',r'} g_{w',w,v',r'} + g_{w',v',w,r'} + \sum_{u,v',r'} f_{u,w,v',r'} + f_{u,v',w,r'} \leq 0 \quad \forall w,v,r \tag{20}$$
$$-d_{i,j,k} + c_{j,i} + c_{k,i} \geq 0 \quad \forall i,j,k \tag{21}$$
$$d_{i,j,k} + c_{j,i} + c_{k,i} \leq 2 \quad \forall i,j,k \tag{22}$$
$$\sum_i d_{i,j,k} \geq 1 \quad \forall j,k \tag{23}$$
$$h_{i,j} = O^i_j \quad \forall i,j|O^i_j \in \{0,1\} \tag{24}$$

Constraints (3)–(6) ensure that the mapping is coherent with the transition function. Constraints (3) and (4) are the result of the linearization of the following constraint:

$$f_{u,v,t,r} = 1 \wedge c_{i,r} = 0 \implies c_{k,u} = h_{j,v} \quad \forall u,v,t,r,T_j \equiv (S_i,S_k) \tag{25}$$

For each transition $T_j \equiv (S_i, S_k)$, if $P_r = 0$ (i.e., the r-th bit of the encoding of the present state, S_i, is equal to 0), then the multiplexer connects O_v to N_u; therefore, the u-th bit of the coding of the next state, S_k, must be equal to the value of the output O_v. Similarly, for the cases in which $P_r = 1$, (5) and (6) are obtained from the following constraint:

$$f_{u,v,t,r} = 1 \wedge c_{i,r} = 1 \implies c_{k,u} = h_{j,t} \quad \forall u,v,t,r,T_j \equiv (S_i, S_k) \tag{26}$$

Constraints (7)–(10) guarantee the coherence with the output function. Constraints (7) and (8) are obtained by the linearization of the following constraint:

$$g_{w,v,t,r} = 1 \wedge c_{i,r} = 0 \implies h_{j,w} = h_{j,v} \quad \forall w,v,t,r,T_j \equiv (S_i, S_k) \tag{27}$$

If $P_r = 0$, then the multiplexer connects O_v to O_w; therefore, they must be equal to each other. Similarly, for the cases in which $P_r = 1$, (9) and (10) are obtained from the following constraint:

$$g_{w,v,t,r} = 1 \wedge c_{i,r} = 1 \implies h_{j,w} = h_{j,t} \quad \forall w,v,t,r,T_j \equiv (S_i, S_k) \tag{28}$$

Constraints (11) and (12) ensure that the state-encoding bits and the outputs, respectively, are not mapped more than once (see Figure 3a,b). Constraint (13) guarantees that any output cannot be used to generate more than one signal by means of a multiplexer, whether a state-encoding bit or an output (see Figure 3c). Constraints (14) and (15) prevent that an output is mapped to itself (see Figure 3d).

Constraints (16) and (17) are the results of the linearization of the constraints

$$g_{w,v,t,r} = 1 \implies \sum_{v',t' \neq v',r'} g_{v,v',t',r'} = 0 \quad \forall w,v,t \neq v,r \tag{29}$$

and

$$g_{w,t,v,r} = 1 \implies \sum_{v',t' \neq v',r'} g_{v,v',t',r'} = 0 \quad \forall w,v,t \neq v,r \tag{30}$$

respectively. They prevent that an output generated from others via a multiplexer is used to map an output using a multiplexer (see Figure 4a). Similarly, (18) and (19) are obtained by linearization of the constraints

$$f_{u,v,t,r} = 1 \implies \sum_{v',t' \neq v',r'} g_{v,v',t',r'} = 0 \quad \forall u,v,t \neq v,r \tag{31}$$

and

$$f_{u,t,v,r} = 1 \implies \sum_{v',t' \neq v',r'} g_{v,v',t',r'} = 0 \quad \forall u,v,t \neq v,r \tag{32}$$

respectively. They prevent the mapping of a state-encoding bit in the circumstances described above (see Figure 4b).

Constraint (20) is obtained from constraint

$$g_{w,v,v,r} = 1 \implies \sum_{w',v',r'} g_{w',w,v',r'} + g_{w',v',w,r'} + \sum_{u,v',r'} f_{u,w,v',r'} + f_{u,v',w,r'} = 0 \quad \forall w,v,r \tag{33}$$

which prevents an output directly mapped to another being used to generate a third signal (either an output or a state-encoding bit). Figure 4c,d show the cases of outputs and state-encoding bits, respectively. The aim of these constraints is to prevent that a direct mapping leads to the situations shown in Figure 4e, despite constraints from (16) to (19). It might be thought that (20) prevents some valid solutions. For example, supposing that O_v is not a mapped output, the implementation shown on the left of Figure 4c could be valid if it is not prevented by (20), allowing that O_w and $O_{w'}$ are mapped outputs. However, the solver will find the following equivalent solution (which do is allowed): O_v and O'_v generate O'_w via the multiplexer, and O_v also generates O_w.

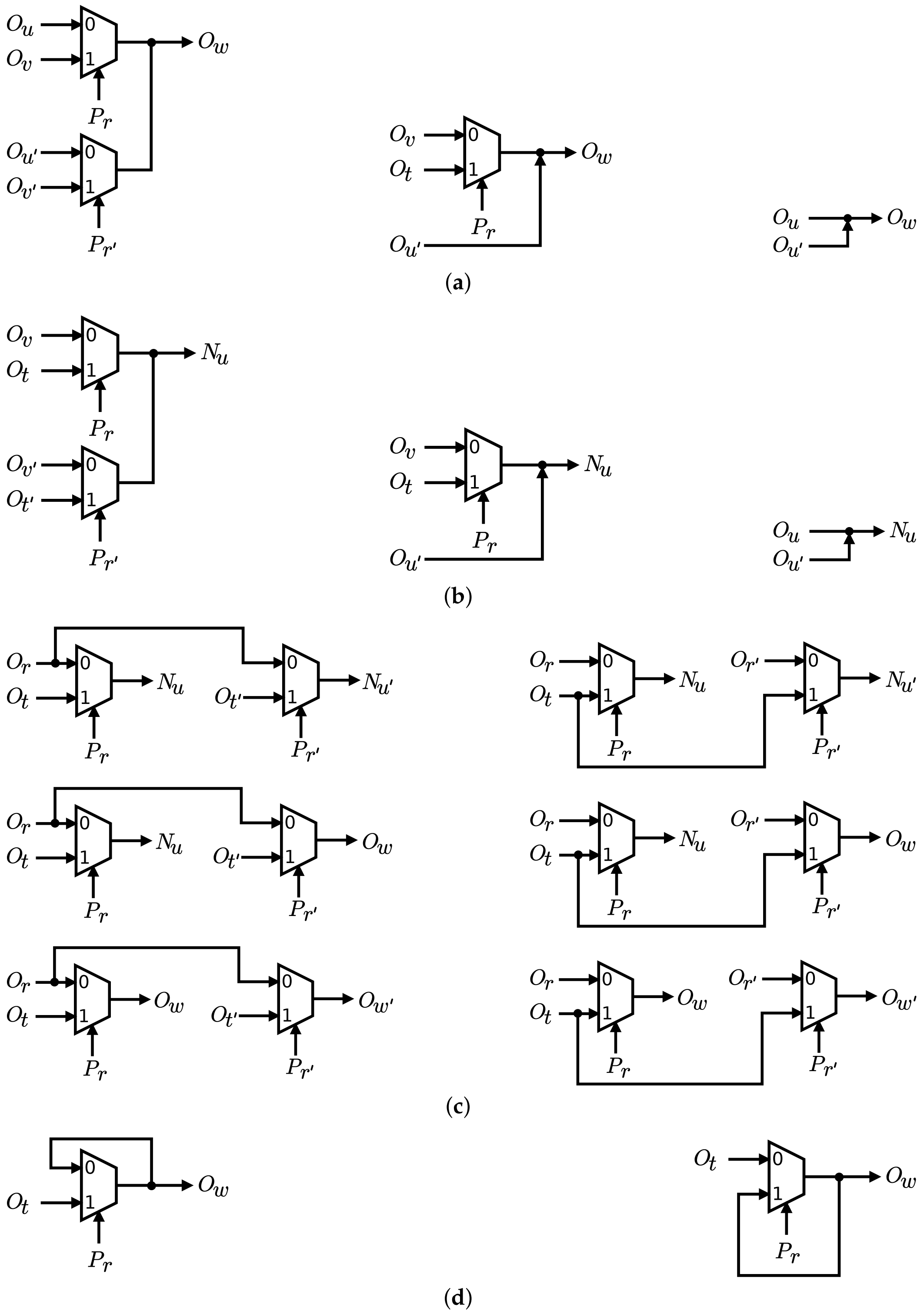


Figure 3. Solutions that ILP constraints from (11) to (15) prevent: (**a**) constraint (11), (**b**) constraint (12), (**c**) constraint (13) and (**d**) constraints (14) and (15).

Finally, (21)–(23) guarantee the code assigned to each state is different from the other ones. Constraints (21) and (22) ensure that the i-th bit of the encoding of S_j is different from the i-th bit of the encoding of S_k if $d_{i,j,k} = 1$; that is,

$$d_{i,j,k} = 1 \implies c_{j,i} \neq c_{k,i} \quad \forall i, j, k \tag{34}$$

Constraint (23) guaranties that any pair of states, S_j and S_k, differ at least in one bit.

Constraint (24) assigns the values of the outputs that are not don't care values, and thus they cannot be determined by the solver.

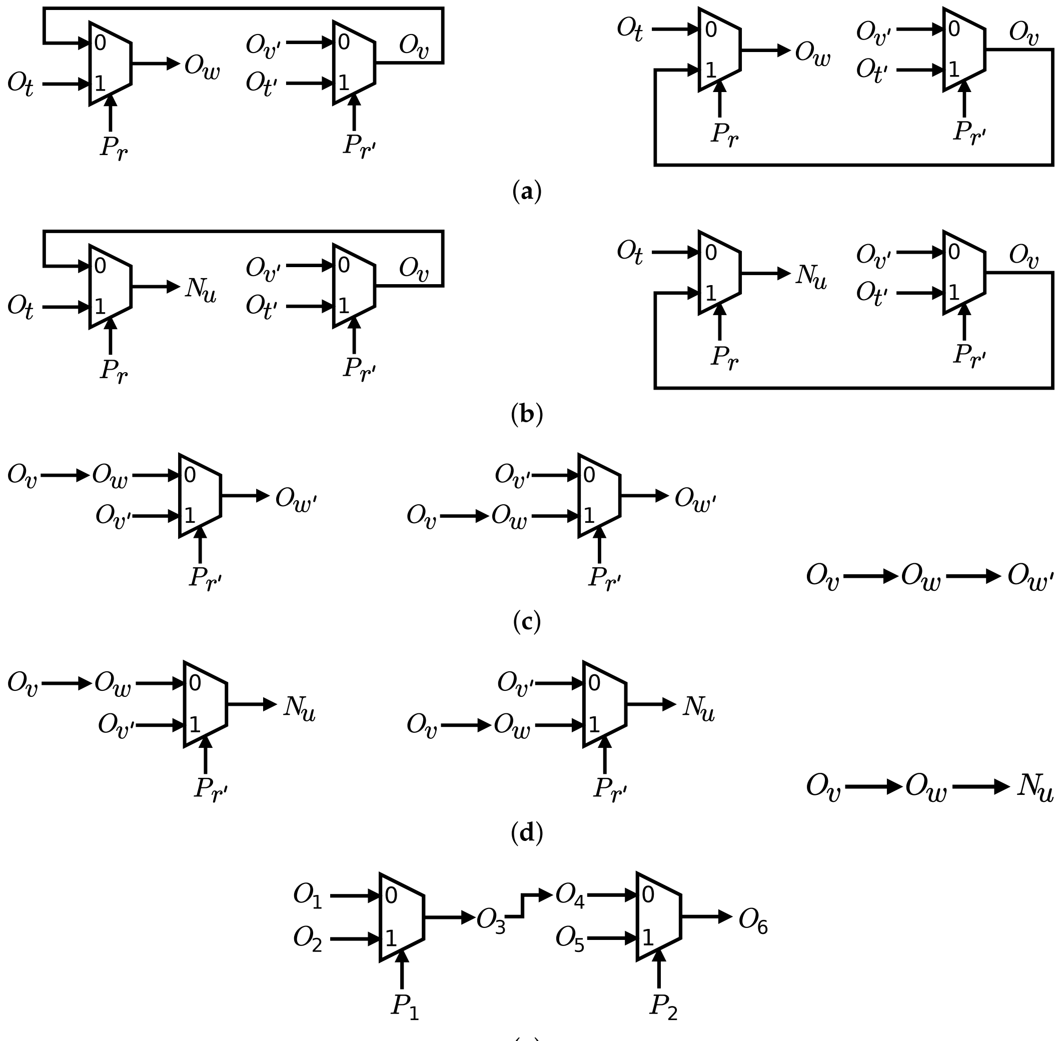


Figure 4. Solutions that ILP constraints from (16) to (20) prevent: (**a**) constraints (16) and (17), (**b**) constraints (18) and (19), (**c**) constraint (20) for outputs, (**d**) constraint (20) for state encoding bits and (**e**) an example of the situation that must be prevented.

5. Experimental Results

OMOSEBO has been evaluated using FSMs of the MCNC benchmark set [27]. The number of states of these FSMs were previously minimized by STAMINA [28]. All designs were synthesized and implemented using Xilinx Vivado Design Suite 2022.1 in a Spartan-7 FPGA (xc7s6cpga196-2). Therefore, the results include the delay of the placement-and-routing stage. In order to evaluate the effectiveness of OMOSEBO, the implementations obtained using this technique were compared to the corresponding conventional FSM implementations (which will be called CONV). CONV implementations were generated using the template for FSMs of Vivado (see Figure 5, which shows the VHDL code for implementing the FSM example using the FSM template). The proposed ILP formulation was implemented using the API of Gurobi [29] for Python and solved with a time limit of 4 hours. The optimization was carried out on an Intel i7-10700K at 3.80 GHz with 16 GB of RAM. The HDL description generated after the optimization process uses a modified version of the FSM template that included a standard description of multiplexors (see Figure 6, which shows the VHDL code for implementing the FSM example by applying OMOSEBO). As the aim of OMOSEBO is to improve the area, Vivado was configured for area optimization using the preconfigured strategies Flow_AreaOptimized_high and Area_Explore for synthesis and implementation, respectively.

```
entity fsm_example is
    port ( clk: in std_logic;
           input : in std_logic_vector( 1 to 1 );
           output : out std_logic_vector( 1 to 5 ));
end fsm_example;

architecture conv of fsm_example is
    type state_type is (S0, S1, S2, S3);
    signal state, next_state : state_type;
    signal output_i : std_logic_vector( 1 to 5 );
    signal input_i : std_logic_vector( 1 to 1 );
begin
    SYNC_PROC: process (clk)
    begin
        if (clk'event and clk = '1') then
            input_i <= input;
            output <= output_i;
            state <= next_state;
        end if;
    end process;
    TRANSITIONS: process( state, input_i )
    begin
        case (state) is
            when S0 =>
                if input_i(1)='0' then
                    next_state <= S0;
                    output_i <= "0-01-";
                else
                    next_state <= S1;
                    output_i <= "-1010";
                end if;
            when S1 =>
                next_state <= S2;
                output_i <= "11-11";
            when S2 =>
                if input_i(1)='0' then
                    next_state <= S3;
                    output_i <= "01100";
                else
                    next_state <= S1;
                    output_i <= "11011";
                end if;
            when S3 =>
                if input_i(1)='0' then
                    next_state <= S3;
                    output_i <= "1--01";
                else
                    next_state <= S0;
                    output_i <= "00111";
                end if;
        end case;
    end process;
end conv;
```

Figure 5. VHDL code for implementing the FSM example using the FSM template.

```
entity fsm_example is
    port ( clk: in std_logic;
           input : in std_logic_vector( 1 to 1 );
           output : out std_logic_vector( 1 to 5 ));
end fsm_example;

architecture omosebo of fsm_example is
    constant S0: std_logic_vector(1 to 2) := "00";
    constant S1: std_logic_vector(1 to 2) := "11";
    constant S2: std_logic_vector(1 to 2) := "01";
    constant S3: std_logic_vector(1 to 2) := "10";
    signal state, next_state : std_logic_vector(1 to 2);
    signal output_i : std_logic_vector( 1 to 5);
    signal output_i_src : std_logic_vector( 1 to 4);
    signal input_i : std_logic_vector( 1 to 1 );
begin
    output_i(1) <= output_i_src(1);
    output_i(2) <= output_i_src(2);
    output_i(3) <= output_i_src(3);
    output_i(4) <= output_i_src(4);
    output_i(5) <= output_i_src(3) when state(2) = '0' else output_i_src(4);
    next_state(2) <= output_i_src(1) when state(1) = '0' else output_i_src(2);
    input_i <= input;
    output <= output_i;
    SYNC_PROC: process (clk)
    begin
        if (clk'event and clk = '1') then
            state <= next_state;
        end if;
    end process;
    TRANSITIONS: process( state, input_i )
    begin
        case (state) is
            when S0 => -- 00
                if input_i(1)='0' then
                    next_state(1) <= S0(1);
                    output_i_src <= "0-01";
                else
                    next_state(1) <= S1(1);
                    output_i_src <= "1101";
                end if;
            when S1 => -- 11
                next_state(1) <= S2(1);
                output_i_src <= "11-1";
            when S2 => -- 01
                if input_i(1)='0' then
                    next_state(1) <= S3(1);
                    output_i_src <= "0110";
                else
                    next_state(1) <= S1(1);
                    output_i_src <= "1101";
                end if;
            when S3 => -- 10
                if input_i(1)='0' then
                    next_state(1) <= S3(1);
                    output_i_src <= "1010";
                else
                    next_state(1) <= S0(1);
                    output_i_src <= "0011";
                end if;
            when others =>
                next_state <= (others => '-');
                output_i_src <= (others => '-');
        end case;
    end process;
end omosebo;
```

Figure 6. VHDL code for implementing the FSM example by applying OMOSEBO.

Table 1 shows the number of LUTs and the maximum clock frequency (in MHz) obtained by CONV and OMOSEBO. In addition, the columns "LUT Red." and "Freq. Inc."

show the area reduction and the speed increment, respectively, obtained by OMOSEBO with respect to CONV; thus, positive values correspond to the cases in which the proposed approach was successful (in column "LUT Red.", these values are shown in bold).

Table 1. Implementation results.

	CONV		**OMOSEBO**		**Comparison**	
FSM	**# LUTs**	**Freq. (MHz)**	**# LUTs**	**Freq. (MHz)**	**LUT Red. (%)**	**Freq. Inc. (%)**
s1	78	358	32	430	**59**	20
mark1	26	443	15	592	**42**	34
s27	7	649	5	693	**29**	7
planet	104	406	85	438	**18**	8
opus	17	567	14	665	**18**	17
s510	58	461	50	440	**14**	−5
s820	77	389	68	396	**12**	2
ex1	70	415	63	407	**10**	−2
ex6	20	643	18	627	**10**	−2
styr	103	376	95	348	**8**	−7
ex4	15	742	14	716	**7**	−4
cse	41	426	41	472	0	11
s832	76	379	85	382	−12	1
keyb	40	357	46	348	−15	−3
mc	3	907	4	851	−33	−6
Mean	49	501	42	520	11	5

In 53% of cases, the time limit was reached. In half of these cases, the gap (i.e., the measure that indicates how long the obtained solution is from the optimal) was more than 50%. Therefore, the results could be improved using more time.

OMOSEBO required a lesser number of LUTs than CONV in 73% of cases. The average LUT reduction was 11%; however, if only the successful cases are taken into account, this value increases to 21%. Although OMOSEBO reduced the number of logic functions in all cases, there were three FSMs (20% of the sample) in which the number of used LUTs was greater than that obtained by CONV. This is due to the fact that OMOSEBO encodes the states to reduce the number of logic functions, whereas CONV encodes the states to simplify the logic functions, and so there may be cases in which the increase in complexity of the logic functions does not compensate for fewer functions. In addition, the don't care outputs assigned by OMOSEBO may make the simplification process performed by Vivado more difficult.

Regarding the maximum clock frequency, the results do not show a clear correlation between LUT reduction and speed increment (the correlation coefficient is 0.69). Nevertheless, although the goal of the proposed technique is to reduce the area, in the 55% of cases in which the OMOSEBO reduced the area, it also increased the speed, obtaining an average speed increment of 15% for such cases. In order to study the relationship between speed increment and area reduction, we obtained the regression line (see Figure 7). Although area and speed usually are conflicting goals, in general terms, the area reduction obtained by OMOSEBO was achieved without degrading the speed; in fact, the regression line shows that the area reduction has a positive impact on the increase in speed.

In order to show the improvement that OMOSEBO represents with respect to conventional column compaction [22], we applied that technique to all studied FSMs. For that, we added the following constraints to the proposed ILP formulation:

$$f_{u,v,t,r} = 0 \quad \forall u, v, t, r \tag{35}$$

$$g_{w,v,t,r} = 0 \quad \forall w, v, t, r | t \neq vs. \tag{36}$$

The constraint (35) prevents the mapping of state bits to outputs. Similarly, (36) prevents the mapping of outputs to outputs via multiplexers. Therefore, only direct mapping of outputs (i.e., column compaction) is allowed.

The impact of column compaction with respect to the solutions found by OMOSEBO, which also includes direct mapping of next state bits and mapping via multiplexers, is very limited. Although the optimal value was reached in all cases, column compaction reduced the number logic functions only in four cases. In the remaining 11 cases, the reduction was due exclusively to direct mapping of next state bits and/or to mapping via multiplexers. Even in those four cases, the number of logic functions obtained by OMOSEBO was less than that of column compaction for two of them (a 17% and a 27% less for keyb and mark1, respectively), and equal for the two others (s820 and s832).

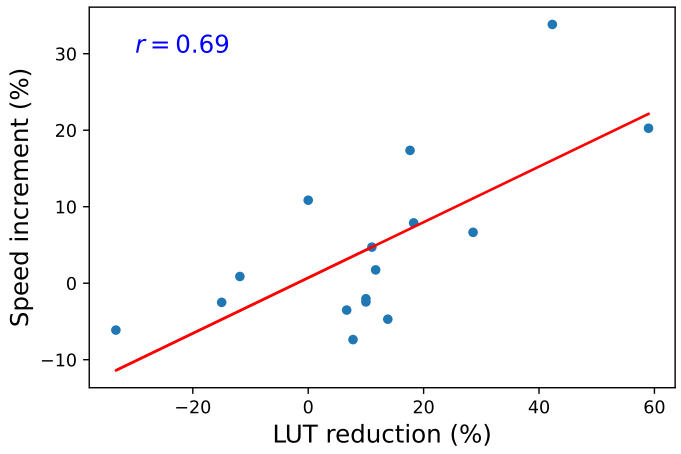


Figure 7. Speed increment vs. LUT reduction (*r* represents the correlation coefficient).

6. Conclusions

In this paper, OMOSEBO, a new technique for implementing FSMs in FPGAs, has been presented. It can be viewed as an extension of column compaction that is applied to the state transition function in addition to the output function. Moreover, OMOSEBO also extends the concept of compaction to that of mapping via multiplexers. Therefore, the proposed technique uses three mechanisms to reduce the number of logic functions: direct mapping of outputs (i.e., column compaction), direct mapping of state-encoding bits and mapping via multiplexers, which are modeled by a unique ILP formulation. Unlike conventional state-encoding techniques, in which states are encoded to simplify the state transition and output functions, OMOSEBO reduces the number of required logic functions.

In order to evaluate the effectiveness of OMOSEBO, experimental results using standard benchmarks have been presented. Regarding column compaction, unlike other similar approaches, the proposed ILP formulation allows one to find optimal solutions. Despite that, in 11 of the 15 studied cases, the reduction in the number of logic functions is due exclusively to direct mapping of next state bits and/or mapping via multiplexers, whereas only in 2 of the studied cases, the reduction is due exclusively to column compaction. In conclusion, OMOSEBO represents a significant improvement over column compaction.

The FSM implementations obtained by applying OMOSEBO have been compared to conventional FSM implementations. The results show that OMOSEBO reduces the number of used LUTs with respect to the conventional implementation in 73% of cases.

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

Funding: This research received no external funding.

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

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

References

1. Katz, R.H. *Contemporary Logic Design*; Benjamin/Cummings: Redwood City, CA, USA, 1994.
2. Barkalov, A.; Titarenko, L. *Logic Synthesis for FSM-Based Control Units*; Lecture Notes in Electrical Engineering; Springer: Berlin/Heidelberg, Germany, 2009.
3. Barkalov, A.; Titarenko, L.; Kolopienczyk, M.; Mielcarek, K.; Bazydlo, G. Design of EMB-Based Mealy FSMs. In *Logic Synthesis for FPGA-Based Finite State Machines*; Springer International Publishing: Berlin/Heidelberg, Germany, 2016; pp. 193–237. [CrossRef]
4. Baranov, S. *Logic and System Design of Digital Systems*; TUT Press: Tallin, Estonia, 2008.
5. Barkalov, A.; Titarenko, L.; Mielcarek, K.; Chmielewski, S. Twofold State Assignment for Mealy FSMs. In *Logic Synthesis for FPGA-Based Control Units*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 61–90.
6. Barkalov, A.; Titarenko, L.; Mielcarek, K.; Chmielewski, S. Twofold State Assignment for Moore FSMs. In *Logic Synthesis for FPGA-Based Control Units*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 91–116.
7. El-Maleh, A. A Probabilistic Tabu Search State Assignment Algorithm for Area and Power Optimization of Sequential Circuits. *Arab. J. Sci. Eng.* **2020**, *45*, 6273–6285. [CrossRef]
8. da Silva Ribeiro, R.; de Carvalho, R.L.; da Silva Almeida, T. Optimization of state assignment in a finite state machine. *Acad. J. Comput. Eng. Appl. Math.* **2022**, *3*, 9–16. [CrossRef]
9. Solov'ev, V. Minimization of mealy finite-state machines by using the values of the output variables for state assignment. *J. Comput. Syst. Sci. Int.* **2017**, *56*, 96–104. [CrossRef]
10. Salauyou, V.; Ostapczuk, M. State Assignment of Finite-State Machines by Using the Values of Output Variables. In *Theory and Applications of Dependable Computer Systems. Advances in Intelligent Systems and Computing*; Springer: Berlin/Heidelberg, Germany, 2020; Volume 1173, pp. 91–116.
11. Das, N.; Priya P, A. Reset: A Reconfigurable state encoding technique for FSM to achieve security and hardware optimality. *Microprocess. Microsyst.* **2020**, *77*, 103196. [CrossRef]
12. Barkalov, A.A.; Titarenko, L.; Mielcarek, K. Reducing LUT Count for Mealy FSMs With Transformation of States. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2022**, *41*, 1400–1411. [CrossRef]
13. Aly, W.M. Solving the State Assignment Problem Using Stochastic Search Aided with Simulated Annealing. *Am. J. Eng. Appl. Sci.* **2009**, *2*, 703–707. [CrossRef]
14. De Micheli, G.; Brayton, R.; Sangiovanni-Vincentelli, A. Optimal State Assignment for Finite State Machines. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **1985**, *4*, 269–285. [CrossRef]
15. Villa, T.; Sangiovanni-Vincentelli, A. NOVA: State assignment of finite state machines for optimal two-level logic implementation. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **1990**, *9*, 905–924. [CrossRef]
16. Mengibar, L.; Entrena, L.; Lorenz, M.; Millan, E. Partitioned state encoding for low power in FPGAs. *Electron. Lett.* **2005**, *41*, 948–949. [CrossRef]
17. Chen, D.S.; Sarrafzadeh, M.; Yeap, G. State encoding of finite state machines for low power design. In Proceedings of the 1995 IEEE International Symposium on Circuits and Systems (ISCAS), Seattle, DC, USA, 30 April–3 May 1995; Volume 3, pp. 2309–2312. [CrossRef]
18. Avedillo, M.; Quintana, J.; Huertas, J. State merging and state splitting via state assignment: A new FSM synthesis algorithm. *IEEE Proc. Comput. Digit. Tech.* **1994**, *141*, 229–237. [CrossRef]
19. Almaini, A.; Miller, J.; Thomson, P.; Billina, S. State assignment of finite state machines using a genetic algorithm. *IEEE Proc. Comput. Digit. Tech.* **1995**, *142*, 279–286. [CrossRef]
20. El-Maleh, A.; Sait, S.; Nawaz Khan, F. Finite state machine state assignment for area and power minimization. In Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, Kos, Greece, 21–24 May 2006; p. 4.
21. Wei, R.S.; Tseng, C.J. Column compaction and its application to the control path synthesis. In Proceedings of the 1987 IEEE International Conference on Computer-Aided Design Digest of Technical Papers, Santa Clara, CA, USA, 9–12 November 1987; Volume 87, pp. 320–323.
22. Binger, D.; Knapp, D. Encoding multiple outputs for improved column compaction. In Proceedings of the 1991 IEEE International Conference on Computer-Aided Design Digest of Technical Papers, Santa Clara, CA, USA, 11–14 November 1991; pp. 230–233. [CrossRef]
23. Mitra, S.; Avra, L.; McCluskey, E. An output encoding problem and a solution technique. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **1999**, *18*, 761–768. [CrossRef]
24. Le Gal, B.; Ribon, A.; Bossuet, L.; Dallet, D. Area optimization of ROM-based controllers dedicated to digital signal processing applications. In Proceedings of the 2010 18th European Signal Processing Conference, Aalborg, Denmark, 23–27 August 2010; pp. 547–551.
25. Lee, S.; Choi, K. Critical-Path-Aware High-Level Synthesis with Distributed Controller for Fast Timing Closure. *ACM Trans. Des. Autom. Electron. Syst.* **2014**, *19*. [CrossRef]

26. Xilinx. 7 Series FPGAs Configurable Logic Block: User Guide, 2016. Available online: https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB (accessed on 17 January 2023).
27. Yang, S. *Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0*; Microelectronic Center of North Carolina: Research Triangle, NC, USA, 1991.
28. Rho, J.K.; Hachtel, G.D.; Somenzi, F.; Jacoby, R.M. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **1994**, *13*, 167–177. [CrossRef]
29. Gurobi Optimization. Gurobi Optimizer Reference Manual. 2020. Available online: http://www.gurobi.com (accessed on 17 January 2023).