Methodology for Distributed-ROM-based Implementation of Finite State Machines

Raouf Senhadji-Navarro and Ignacio Garcia-Vargas

***Abstract*—This brief explores the optimization of distributed-ROM-based Finite State Machine (FSM) implementations as an alternative to conventional implementations based on Look-Up Tables (LUTs). In distributed-ROM implementations, LUTs with constant output value (called constant LUTs) and LUTs with the same content (called equivalent LUTs) can be saved. We propose a methodology to implement FSMs using distributed ROM that includes: (1) a greedy state encoding algorithm, (2) an algorithm to find the way of interconnecting the address signals to the ROM that maximize the number of constant or equivalent LUTs, and (3) a set of architectures to implement the columns of the ROM. The results obtained have been compared with conventional LUT-based implementations using standard benchmarks. The proposed technique reduces the number of LUTs in a 91% of cases and increases the speed in all cases.**

***Index Terms*—distributed ROM, ROM-based implementation, finite state machine, FSM, FPGA.**

I. Introduction

A Finite State Machine (FSM) is probably the most widely used component of digital systems because it allows to model any sequential circuit, and control units in particular. Optimizing FSM implementations in terms of area, speed or power consumption is essential to meet the design constraints demanded by applications. The implementation process of FSMs in FPGAs can be divided into two main stages: the search for an optimal state encoding that allows an efficient implementation of the output and state transition functions, and the mapping of the resulting logic functions into FPGA resources. The problem of FSM state encoding has been receiving attention of researchers, designers and EDA tool developers for decades [1]. Regarding the logic function mapping, the conventional logic synthesis methods consist in the decomposition of logic functions into subfunctions of q-inputs that are mapped into q-input Look-Up Tables (LUTs) [2], [3]. Conventional techniques to implement FSMs in FPGAs map the output and state transition into LUTs by applying conventional logic synthesis methods from a behavioural HDL description of the FSM (i.e., these functions are treated as general functions without using a specific architecture).

As an alternative to conventional techniques, ROM memory can be used to implement any logic functions or, particularly, the output and state transition functions of an FSM. In FPGAs, there exist two different kind of ROM memory: block ROM (built from embedded memory blocks) and distributed ROM (built from LUTs configured as ROM) [4]. Distributed ROM is used to store small amount of data whereas block ROM is used to implement larger memories. In the literature, different approaches that implement FSMs using block ROM have been reported [5]. However, to the best knowledge of the authors, the problem of optimizing FSM implementations based on distributed ROM has not been addressed yet. In this brief, we propose a novel methodology to implement FSMs using distributed ROM instead of block ROM; thus, as distributed ROM is built from LUTs, the proposed technique can also be seen as an alternative method to map logic functions into LUTs (in this work, we focus on the output and state transition functions of FSMs).

Each column of a distributed-ROM is built from LUTs configured as ROM whose outputs are connected to multiplexers. Current FPGA devices include dedicated multiplexers in order to combine the output of the LUTs. Dedicated multiplexers can improve the performance and density of general functions, but they are especially suitable for efficient implementation of small distributed ROMs. In comparison with conventional techniques, an advantage of the implementation of logic functions as distributed ROM is that it can greatly benefit from embedded multiplexers [4]. In distributed ROM implementations, LUTs with a constant output value (called constant LUTs) can be replaced by fixed logic values. In addition, all LUTs with the same content (called equivalent LUTs) can be substituted by a unique LUT. The number of constant or equivalent LUTs (called redundant LUTs) depends on the state encoding and on the way of connecting the address signals to the ROM (called address assignment); therefore, an optimization process can be applied to save LUTs. Don't care inputs and outputs, usually present in real FSMs, can be exploited by the optimization process to increase the number of redundant LUTs.

The proposed methodology can be divided into the following three steps: (1) a greedy algorithm finds a state encoding that increases the potential number of redundant LUTs, (2) an algorithm finds the address assignment that maximizes the number of redundant LUTs, and (3), for each column of the distributed ROM, different architectures are evaluated and the best one (taking into account the optimization goal) is selected to be part of the final implementation.

As distributed ROM is built from LUTs, both the conventional techniques and the proposed one use the same kind of FPGA resources. However, since the architectures included in the proposed methodology are based on distributed ROM, the implementations can greatly benefit from embedded multiplexers. In addition, these architectures restrict the search space of solutions with respect to conventional techniques. The

Raouf Senhadji-Navarro and Ignacio Garcia-Vargas are with the Department of Computer Architecture and Technology, University of Seville, Spain, e-mail: raouf@us.es

Manuscript received March 11, 2020; revised March 11, 2020.

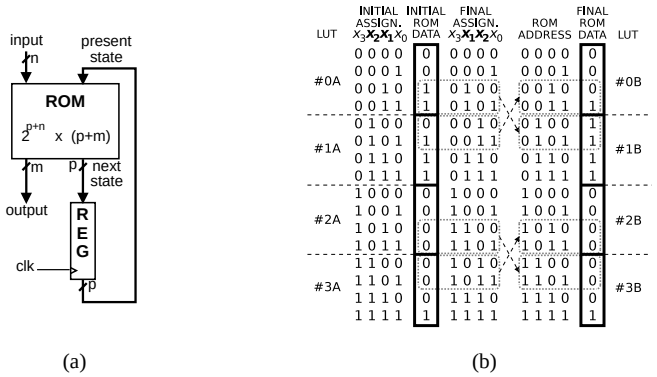

Fig. 1. ROM-based FSM implementation: (a) architecture and (b) two different address assignments for a distributed ROM based on 2-input LUTs.

proposed optimization process, which is specific for distributed ROM, benefits from the restricted search space, making easier the finding good solutions.

II. Distributed-ROM-based FSM implementation

Fig. 1a shows a ROM-based FSM implementation. Each ROM word contains a transition of the FSM (i.e., the FSM output and the next state encoding bits of the transition). The address is composed of the input signal and the present state encoding bits. The next state is stored in the register and is fed back to the address signal as the present state [6].

In a FPGA based on q-input LUTs, a distributed ROM of $2^u \times v$ is composed by v columns, each of which contains 2^{u-q} LUTs configured as ROM (we will refer to them as data LUTs when necessary to avoid ambiguity) and a multiplexer of 2^{u-q} inputs. The input of all LUTs are connected to the q least significant bits of the address. For each column, the outputs of the LUTs are connected to a multiplexer controlled by the $u-q$ most significant bits of the address [6].

Dedicated multiplexers improve the performance and density of general functions. However, the implementation of multiplexers takes advantages of these resources in a particular way. For example, in a Spartan-7 FPGA, the three dedicated multiplexers of one slice allow the implementation of generic functions with up to 8 inputs whereas they can be used to implement up to 16-input multiplexers (which are 20-input logic functions) [4]. Thus, ROMs whose size is not too big are efficiently implemented as distributed ROM.

Mapping logic functions to distributed ROMs restricts the set of possible implementations that can be inferred; for example, all data LUTs must be connected to the same subset of inputs. As a consequence, the search space of solutions is smaller than that of conventional techniques. The proposed optimization process benefits from this restricted search space, making easier the finding good solutions.

III. Optimization of distributed-ROM-based FSM implementations

The number of data LUTs can be reduced by finding an adequate address assignment. For example, Fig. 1b shows the effect of two different address assignments on a distributed ROM based on 2-input LUTs (we will refer to these assignments as initial and final assignments). The final assignment (in which the address signals x_1 and x_2 have been interchanged) allows to increase the number of constant LUTs from 1 (see LUT #2A) to 2 (see LUTs #1B and #2B) and the number of equivalent LUTs from 0 to 2 (see LUTs #0B and #3B); so, the total number of LUTs is reduced from 3 to 1.

In distributed-ROM-based FSM implementations, don't care inputs and outputs have a positive influence on the number of redundant LUTs. Don't care outputs can be set to 0 or to 1 as needed to obtain redundant LUTs. Don't care inputs have the effect of replicating ROM words. In general terms, this effect can be exploited to increase either the number of constant LUTs (by assigning don't cares to the address signals connected to data LUTs) or the number of equivalent LUTs (by assigning don't cares to the address signals not connected to data LUTs). For example, let us suppose that 2-input LUTs are used to implement a distributed-ROM-based FSM with 4 states coded as (s_1, s_0) and 4 inputs (x_3, x_2, x_1, and x_0), in which the state coded as $(0,0)$ is not sensitive to x_2 and x_3. If the assignment is $(s_1, s_0, x_1, x_0, x_3, x_2)$, the first four addresses (determined by $0000x_3x_2$) contain the same value; therefore, the first LUT is constant. Similarly, the next three LUTs (related to the addresses $0001x_3x_2$, $0010x_3x_2$ and $0011x_3x_2$) are also constant. However, if the assignment is $(s_1, s_0, x_3, x_2, x_1, x_0)$, the four addresses determined by $00x_3x_200$ (which address the first word of each LUT) contain the same value. Since it is also true for the 2nd, 3rd and 4th word, the first four LUTs are equivalent.

Let us suppose an FSM with n inputs and p state encoding bits that is implemented in a FPGA based on q-input LUTs. The number of bits of the ROM address is $n+p$, where the q least significant bits correspond to the signals connected to LUTs. Permutations of the q least significant bits can only rearrange the content of each LUT, but cannot transform non-redundant LUTs into redundant. This can be done only by means of permutations that interchange one or more bits between the q least significant bits and the rest. Hence, the number of address assignments that must be evaluated is given by $\binom{n+p}{q}$. In a distributed ROM, the columns can have different input assignments because the implementation of each column has its own multiplexer and its own set of LUTs. To take advantage of this property, the procedure of searching the best address assignment (i.e., the assignment that maximizes the number of redundant LUTs) is applied independently to each column. Thus, the set of the best address assignments of each column forms the best address assignment for the ROM (hereinafter simply referred to as best address assignment).

In order to find the the best address assignment, the proposed methodology iteratively applies a brute-force algorithm to each column of the ROM. A quick analysis of the time complexity of the algorithm shows that the number of performed operations increases with the number of different assignments and the number of ROM words; thus, the time complexity is $\mathcal{O}((n+p)q2^{n+p})$, where q is 6 in current FPGAs. Regarding space complexity, the amount of memory required increases with the number of ROM words; thus, the space complexity is $\mathcal{O}(2^{n+p})$. Although the time and space complexity are non-

polynomial, almost all test cases used in the experiments are tractable. Anyway, the algorithm can be easily accelerated using parallel architectures such as multicore processors, computer clusters or GPUs. Since columns are independently to each other, each one of them can be processed by a different task (i.e., thread or process). In a computer cluster, each node could process a different column of the ROM by executing the brute-force algorithm; in this way, the total computation time would be reduced to that corresponding of a single column. In addition, the inherent parallelism of the algorithm can be exploited to achieve further performance improvements. Each task can process a different subset of address assignments, which allow to reduce the time of processing of each column. In addition, each column can be partitioned into blocks that are processed by different tasks (i.e., each task generates the rearranged content of its block and counts the number of redundant LUTs); this allow to reduce the time of processing of each possible address assignment. As a conclusion, since the tasks can be performed independently of each other, we can say that the algorithm is embarrassingly parallel, and therefore it can greatly benefit from parallelization.

Before searching the best address assignment, the ROM content must be generated from the FSM using an appropriate state encoding. We propose an heuristic to find a state encoding that increases the number of constant LUTs. Whenever the state encoding bits are connected to the most significant bits of the ROM address, the transitions of any state, and therefore the encoding bits of its next states, are stored in contiguous words. If these next states are encoded using consecutive binary values, the probability that their most significant bits have the same values increases, and so does the probability of contiguous identical values in the corresponding columns. Hence, such state encoding can help the subsequent optimization process to find better address assignments for the columns that store the most significant encoding bits; however, the rest of columns are not benefited.

The proposed greedy algorithm (see Algorithm 1) processes all states with more than one next state in increasing order of the number of next states. All ROM words related to a state with a unique next state contain always the same next state code, hence the encoding used is irrelevant for the purpose of the algorithm. For each state, the algorithm assigns consecutive binary codes to all its next states except to those that have been previously codified. So, the encoding of the next states imposes constraints for the states that have not been processed yet. The number of constraints increases with the number of codified next states; for this reason, the states that have a less number of different next states are firstly processed. After all states with more than one next state have been processed (see lines from 5 to 8), the algorithm assigns binary codes to those states which remain uncoded (see lines from 9 to 11). The reset state is coded with zero because the FPGA registers can be reset to zero without using additional logic.

IV. Distributed-ROM architectures

This section presents different architectures for implementing a single column of a distributed ROM. All of them benefit from the existence of redundant LUTs, or only constant LUTs in a particular architecture. The optimization process described in Section III is a high-level process that determines and rearranges the content of each column of the ROM to increase the number of redundant LUTs (or constant LUTs in the particular architecture) independently of the architecture. After the optimization process is applied to the FSM, the next stage of the proposed methodology consists in the selection of the best architecture for each column of the corresponding distributed ROM. To achieve this, all architectures are evaluated for each column by synthesizing and implementing each architecture separately. The result of this stage is a VHDL description of the distributed ROM in which each column is implemented with the best architecture. As we have described in Section II, a distributed ROM is implemented using a set of data LUTs and a multiplexer. For example, Fig. 2a shows the implementation of a ROM 512×1 in a Spartan-7 FPGA. In this device, each logic cell includes three dedicated 2:1 multiplexers: two called F7 and one called F8 [4]. Each F7 combines the output of two LUTs whereas F8 combines the output of two F7s. In the example, the 8:1 multiplexer is implemented using four F7s and one LUT that implements a 4:1 multiplexer (in this case, F8 is not used because it does not allow to remove the LUT). Let us suppose that the ROM of Fig. 2a includes 2 constant LUTs (shown with dotted line and shaded background) and 2 equivalent LUTs (shown with shaded background). In this architecture, no LUT can be saved because each embedded multiplexer is physically connected to a specific pair of LUTs. In general terms, this architecture does not take advantage of the existence of redundant LUTs;

Algorithm 1 State encoding

Input: Set of states $S = \{s_0, s_1, \ldots, s_{t-1}\}$ where s_0 is the initial state, state transition function $f : S \times X \rightarrow S$ where X is the set of inputs
Output: Set of state codes $\{c_0, c_1, \ldots, c_{t-1}\}$ where c_i represents the code of s_i

1: $k \leftarrow 1$; $c_0 \leftarrow 0$; $c_i \leftarrow -1$ for $i = 1, 2, \ldots, t-1$
2: $D_i \leftarrow \{f(s_i, x) | x \in X\}$ for $i = 0, 1, 2, \ldots, t-1$
3: **for each** D_i in increasing order of $|D_i|$ **do**
4: **for each** $s_j \in D_i$ such that $c_j < 0$ and $|D_i| > 1$ **do**
5: $c_j \leftarrow k$; $k \leftarrow k + 1$
6: **for each** $s_j \in S$ such that $c_j < 0$ **do**
7: $c_j \leftarrow k$; $k \leftarrow k + 1$

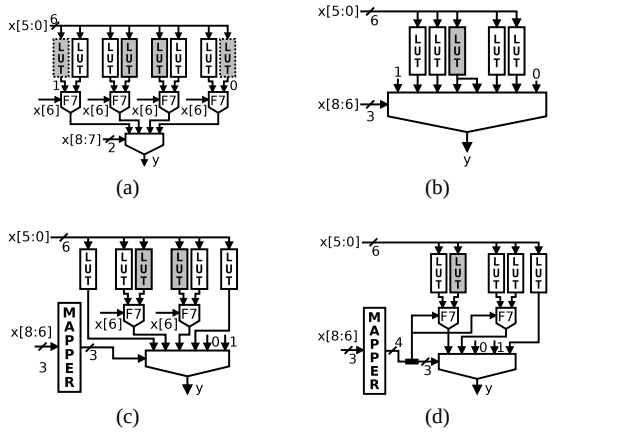

Fig. 2. Example of implementations of a distributed ROM 512×1 in a Spartan-7: (a) complete architecture, (b) architecture based on a unique multiplexer, (c) architecture with partially-mapped address signals, and (d) architecture with fully-mapped address signals.

therefore, it does not benefit from the proposed optimization technique and so it is not included in the set of evaluated architectures. The remaining of this section describes the included architectures.

The architecture shown in Fig. 2b uses a unique multiplexer that can be viewed as a particular selection function of 8 inputs (i.e., 5 data inputs and 3 control inputs). The number of data LUTs is reduced from 8 to 4 respect to the architecture shown in Fig. 2a. The synthesis tool can simplify the function; however, the implementation does not benefit from dedicated multiplexers in the same way as a regular multiplexer.

We propose an alternative architecture that removes the constant LUTs from the architecture shown in Fig. 2a and that uses embedded multiplexers only for non-constant LUTs (see Fig. 2c). A combinational circuit (called mapper) maps the most significant bits of the address signal to the control bits of the multiplexer. The mapper allows to reduce the number of inputs of the multiplexer, and therefore its complexity, because all constant inputs are reduced to only two. This architecture is useful when the number of LUTs saved by the simplification of the multiplexer pays off the extra logic required by the mapper. However, since embedded multiplexers are controlled directly by address signals, only blocks of 128 bits that start at addresses multiple of 128 can be mapped to two LUTs connected by one F7 (similarly, blocks of 256 bits must start at addresses multiple of 256). In addition, equivalent LUTs can not be removed when they belong to different blocks.

To overcome these problems, the mapper can also generate the control bits of the embedded multiplexers (see Fig. 2d), and therefore any block of 64 bits can be mapped to any LUT. This architecture has two advantages: (1) equivalent LUTs can be removed and (2) it allows to reduce the number of inputs of the multiplexer (in the example, from 4 to 3). However, the disadvantage is that the number of outputs of the mapper can increase (in the example, from 3 to 4).

Finally, the synthesis tools inference distributed ROMs and apply their own optimization techniques when ROMs are described using the proper VHDL instantiation template. As the implementation results depend largely on the ROM data, the methodology also includes the implementations generated from these descriptions in the set of evaluated architectures.

V. Experimental results

MCNC benchmark FSMs [7] have been used to evaluate the proposed technique. The test cases whose number of words is not greater than 128 (i.e., each column requires at most 2 LUTs) have been discarded because no optimization can be attained. The FSMs used in all experiments have been previously minimized: the number of states have been minimized by STAMINA [8], and the resulting logic have been simplified by ESPRESSO [9]. Once FSMs are minimized, states are codified using Algorithm 1 and then the best address assignment is determined by the search procedure. Two different assignments are selected: one that maximizes only the number of constant LUTs for the architecture with partially mapped address signals (see Fig. 2d), and one that maximizes the number of redundant LUTs for the rest of architectures. The results obtained by the search procedure for each FSM are summarized in Table I, which shows the some statistical measures of the number of different possible address assignments per column (ApC), the execution time per column (TpC), the total execution time (Time), the ratio between the maximum and minimum theoretical number of required data LUTs (Max/Min), and the number of words of the ROM (Depth).

TABLE I
Best-address-assignment search procedure: experimental data

	Mean	Std. Dev.	Min	Q1	Q2	Q3	Max
ApC	1476.3	1812.7	28	273	693	1716	8008
TpC (s)	52.7	213.3	0.0	0.3	1.9	8.3	1006.5
Time (s)	776.2	2983.9	0.1	2.4	19.4	157.2	14091.1
Max/Min	7.3	7.2	2.0	3.2	4.8	7.6	29.0
Depth	8634.2	13980.7	256	1280	3072	8192	65536

The search procedure has been carried out using a brute-force algorithm implemented in Python and executed on an Intel i7-8700K at 3.70 GHz with 16 GB of RAM using only a single processor. The original purpose of the brute-force algorithm was to study the influence of the address assignment on the number of LUTs that can be saved in order to justify a future development of an efficient algorithm. However, the resulting execution times show that the algorithm can be used in most practical applications, in which the number of assignments to evaluate is tractable (see ApC and TpC in Table I). In fact, only four test cases have been excluded because they require too much computation time. For the studied cases, the total execution time is negligible in most of them and it is greater than 13 minutes in only one case.

Regarding the theoretical number of data LUTs required by each address assignment, the maximum is always at least double the minimum. The maximum value is in average 7.3 times the minimum value; so, the theoretical average number of LUTs that could be saved using the best assignment is 86%. Therefore, finding a good address assignment is critical to obtain efficient distributed ROM-based FSM implementations.

In order to evaluate the impact of Algorithm 1, the process of searching the best address assignment has been applied to ROMs generated from the FSMs using a random state encoding, and the results have been compared with those obtained by Algorithm 1. In 55% of the cases, the number of LUTs obtained with the encoding of Algorithm 1 is less than that obtained with the random encoding. In addition, the opposite is only true in a 32% of the cases.

FSMs have been synthesized and implemented using Vivado Design Suite 2019.2 in a Spartan-7 FPGA (xc7s6cpga196-2); therefore, the presented results include the routing overhead. Different Vivado preconfigured strategies for synthesis and implementation have been used, and, for each FSM, the strategy that achieves the best result according to the optimization goal (i.e., the least number of LUTs for area optimization and the highest frequency for speed optimization) have been selected. Each column of the optimized distributed-ROM-based FSM has been implemented using the architectures described in Section IV. For each column, the architecture that has obtained the best result is selected to form part of

TABLE II
EXPERIMENTAL RESULTS

Area optimation						Speed optimation					
FSM	DIST-ROM		VIV-LUT		Red.	FSM	DIST-ROM		VIV-LUT		Inc.
	LUT	MHz	LUT	MHz	(%)		LUT	MHz	LUT	MHz	(%)
ex4	11	668	27	457	**59.3**	ex4	15	720	27	457	**57.7**
cse	37	505	74	385	**50.0**	pma	56	480	112	329	**46.1**
opus	16	566	28	503	**42.9**	bbsse	27	608	43	453	**34.4**
ex6	22	629	38	502	**42.1**	opus	19	644	42	507	**27.1**
pma	46	398	78	288	**41.0**	styr	113	414	151	329	**25.7**
s298	112	353	184	329	**39.1**	sse	30	557	34	447	**24.5**
keyb	39	424	58	298	**32.8**	bbrts[a]	11	658	15	532	**23.6**
styr	105	346	151	329	**30.5**	ex6	22	629	42	510	**23.2**
mark1	14	669	20	569	**30.0**	keyb	39	424	69	351	**20.8**
sand	117	376	162	344	**27.8**	cse	37	505	75	430	**17.4**
bbrts[a]	11	658	15	532	**26.7**	planet	103	440	174	376	**16.9**
planet	98	421	133	331	**26.3**	s1	85	427	72	372	**14.9**
sse	26	544	34	447	**23.5**	ex1	71	456	98	404	**12.8**
bbsse	27	608	35	451	**22.9**	mark1	14	669	25	596	**12.2**
s208	11	629	14	566	**21.4**	s208	11	629	14	566	**11.0**
s386	30	514	38	446	**21.1**	s1494	126	416	131	380	**9.5**
ex1	68	443	86	397	**20.9**	sand	125	376	194	351	**7.2**
tbk	47	401	55	384	**14.5**	s1488	121	415	128	394	**5.3**
s1488	121	415	128	394	**5.5**	s298	129	360	196	343	**4.9**
s1494	124	412	131	380	**5.3**	s420	15	552	16	531	**4.0**
s420	15	552	14	498	-7.1	s386	30	514	41	504	**2.1**
s1	75	386	64	345	-17.2	tbk	55	429	59	427	**0.5**

[a]bbara_bbtas FSM

the final implementation (called DIST-ROM). To evaluate the effectiveness of the proposed technique, conventional LUT-based FSM implementations have been generated according to Vivado instantiation template (called VIV-LUT).

Table II shows the maximum clock frequency (in MHz) and the number of LUTs for both area and speed optimization. The columns "Red." and "Inc." show the area reduction and the speed increase, respectively, obtained by DIST-ROM respect to VIV-LUT; thus, a positive value represents that DIST-ROM is better than VIV-LUT (these values are shown in bold). Regarding area optimization, in 91% of the cases, DIST-ROM reduces the number of LUTs with an average reduction of 29%. Note that this reduction is not achieved at expense of decreasing the speed; in fact, it increases in all cases. The average speed increment is 20% for the cases in which the area is reduced. Regarding speed optimization, DIST-ROM is faster in all cases (with an average speed increment of 18%). In all cases except in *s1*, the number of LUTs is also reduced. As a conclusion, DIST-ROM can be considered as a real alternative to VIV-LUT for speed and area optimization.

Finally, in order to study the influence of the proposed optimization technique regardless of the effect of the architectures, the implementation results of DIST-ROM have been compared with those obtained by the proposed architectures but using a random state encoding and the worst address assignment (called NOPT-ROM). In area optimization, DIST-ROM uses less number of LUTs than NOPT-ROM in 68% of the cases and increases the speed in the same proportion. On average, the proposed optimization technique reduces the number of LUTs from 63 to 53 and increases the speed from 470 to 496 MHz. In speed optimization, DIST-ROM is faster than NOPT-ROM in 95% of the cases and reduces the number of LUTs in all cases. On average, the proposed optimization technique increases the speed from 447 to 515 MHz and reduces the number of LUTs from 85 to 55. Therefore, the implementations benefit from the proposed optimization. However, the results could be improved if the best-address-assignment search procedure also took into account the number of LUTs of the multiplexer and the mapper, or the low-level optimizations done by Vivado (e.g., this tool exploits don't cares to reduce the number of inputs of the data LUTs and to join pairs of 5-input LUTs into 6-input LUTs).

VI. CONCLUSION AND FUTURE WORKS

This brief explores the optimization of ROM-based FSM implementations as an alternative to conventional LUT-based implementations. The presented study shows that the theoretical number of LUTs that can be saved if the best assignment is used instead of the worst one is 86% on average. Compared with conventional LUT-based implementation, both in area and speed optimization, the proposed technique increases the speed in all cases; in addition, the number of LUTs is reduced in 91% of the cases in area optimization, and in 95% of the cases in speed optimization. These results lead the authors to conclude that the proposed distributed-ROM-based FSM implementations can be considered as a feasible alternative to conventional LUT-based implementations in practical applications.

As future work, we plan to study new methods to find a better approximation to the optimal address assignment. The current approximation can be improved by including the cost of the multiplexer and the mapper in the estimation of the number of LUTs and by taking into account low-level optimizations performed by the synthesis tool. Moreover, with the purpose of simplifying the mapper, the optimal assignment from address signals to multiplexer inputs will be studied. In order to apply the methodology to larger FSMs, we will try to develop algorithms to obtain the best address assignment without using brute force. Finally, we plan to study the characteristics of an FSM that make that its implementation is more efficient using one or another architecture.

REFERENCES

[1] L. Jozwiak, A. Slusarczyk, and A. Chojnacki, "Fast and compact sequential circuits for the FPGA-based reconfigurable systems," *Journal of Systems Architecture*, vol. 49, pp. 227–246, 09 2003.
[2] R. Rudell, "Tutorial: design of a logic synthesis system," in *33rd Design Automation Conference Proceedings, 1996*, 1996, pp. 191–196.
[3] J. Cong and Y. Ding, "Combinational logic synthesis for lut based field programmable gate arrays," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 1, no. 2, p. 145204, Apr. 1996.
[4] Xilinx, "7 series FPGAs configurable logic block: User guide," 2016.
[5] A. Barkalov, L. Titarenko, M. Kolopienczyk, K. Mielcarek, and G. Bazydlo, *Design of EMB-Based Mealy FSMs*. Springer International Publishing, 2016, pp. 193–237.
[6] R. Senhadji-Navarro, I. Garcia-Vargas, and J. Guisado, "Performance evaluation of RAM-based implementation of finite state machines in FPGAs," in *IEEE International Conference on Electronics, Circuits and Systems*, 2012, pp. 225–228.
[7] S. Yang, "Logic synthesis and optimization benchmarks user guide. version 3.0," 1991.
[8] J.-K. Rho, G. D. Hachtel, F. Somenzi, and R. M. Jacoby, "Exact and heuristic algorithms for the minimization of incompletely specified state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 2, pp. 167–177, Feb 1994.
[9] M. Theobald, S. M. Nowick, and T. Wu, "Espresso-hf: a heuristic hazard-free minimizer for two-level logic," in *33rd Design Automation Conference Proceedings, 1996*, June 1996, pp. 71–76.