Synthesis of Finite State Machines on Memristor Crossbars

Umberto Ferrandino[1], Marcello Traiola[2], Mario Barbareschi[1], Antonino Mazzeo[1], Petr Fišer[3], Alberto Bosio[2]
[1]DIETI - University of Naples Federico II - Italy
Email: u.ferrandino@studenti.unina.it, {mario.barbareschi, mazzeo}@unina.it
[2]LIRMM - University of Montpellier / CNRS - France
Email: {firstname.lastname}@lirmm.fr
[3]Dept. of Digital Design, Czech Technical University of Prague - Czech Republic
Email: fiserp@fit.cvut.cz

***Abstract*—Memristor device represents one of the most relevant technologies to deal with CMOS technological issues. In the scientific literature, a relevant amount of works have discussed the memristor device, with a particular emphasis on memristor-based crossbar architectures. However, while the synthesis of combinational logic circuits is widely discussed, the same cannot be said for sequential logic circuits. In this work, we propose a new approach for synthesizing sequential circuits based on memristor crossbar, by enhancing an existing architecture. This approach only exploits memristors within the crossbar for implementing the state feedback mechanism, with the aim of advancing the integration process of memristor-based circuits. Moreover, to provide an automated synthesis process of memristor-based sequential circuits, we extend a pre-existing automated synthesis framework so it can be integrated with widely used tools and formats as register-transfer level (RTL) or Berkeley Logic Interchange Format (BLIF) files. We performed several experiments on publicly available benchmarks in order to compare the proposed architecture against its predecessor in terms of circuit integration and efficiency. Obtained results highlight acceptable overheads (up to a maximum of 24%) compared with the opportunity of integration offered by the proposed architecture.**

I. Introduction

Nowadays, in counterflow with the well-known Moore's law, the CMOS technology is starting to reach its physical limits. Among the issues characterizing the latest CMOS manufacturing process, we can cite performance reduction, increasing leakage current, reduced production yield, costly manufacturing and testing processes [1].

Many solutions have been studied for dealing with these problems and, among the emerging technologies, the memristor is one of the most promising candidates due to high integration factor, scalability, CMOS compatibility and non-volatileness capability. The memristor device is inherently both a computational and a memory element, and for these reasons it is a subject of many scientific works in the literature: Computing-in-Memory (CiM) architectures [2], non-volatile memories [3], nanoscale and neuromorphic computing [4], data-intensive applications [5].

Some memristor-based circuit architectures have been proposed in the literature so far: Snider Boolean Logic Circuit (SBLC) [6], Fast Boolean Logic Circuit (FBLC) [7], Implication Logic circuit [8], Threshold Logic Circuit [9] and others. In particular, the FBLC enhances computing performance of the crossbar connection used in SBLC for Boolean functions computation with parallel evaluation of minterms in a constant number of steps, regardless of the implemented Boolean function [7].

Moreover, research efforts are focusing on synthesis approaches targeting the optimization of area, performance and power consumption. Some frameworks have already been proposed for memristor-based crossbars [10], [11]. However, while a relevant number of works target the synthesis of combinational logic circuits, only few works have been proposed about the synthesis of sequential circuits on memristor crossbars, so far. Moreover, such propositions are based on combining memristor-based crossbars and intermediate registers [10], [12]. In this paper, we propose an enhancement of the FBLC architecture to extend its capabilities to implement Finite State Machines (FSMs). In particular, instead of using CMOS latches as storage element, we propose a novel technique to exploit memristor devices within the crossbar itself to realize a feedback mechanism. We called such new architecture "Stateful FBLC" (SFBLC). Moreover, we enhance the XbarGen framework by Traiola et al. [11] in order to extend the automated synthesis process to FSMs and increase its integration with commonly used formats in digital circuit design, as register transfer level (RTL) and Berkeley logic interchange format (BLIF) [13]. We carried out some experiments for comparing the classical FBLC approach with the proposed architecture, by analyzing occupied area, dynamic power consumption and delay.

The remainder of the paper is organized as follows. Section II provides an overview of the FBLC and discusses the differences introduced by our stateful implementation. Section III explains the performed experiments and comparisons between FBLC architecture and our stateful FBLC-based approach. Section IV concludes the paper suggesting some reflections and perspectives for future works.

II. FBLC-based Finite State Machine implementation

In this section, we first describe the classical FBLC architecture, then we present our FBLC extension in order to implement a FSM as a memristor crossbar.

A. FBLC overview

The Fast Boolean Logic Circuit (FBLC) architecture was originally proposed by Xie et al. in [7] to compute any Boolean functions on memristor-based crossbars in a constant number of steps by exploiting the SoP expression of minterms obtained by means of De Morgan's laws:

$$f = M_1 + M_2 + ... + M_n = \overline{\overline{M_1} \cdot \overline{M_2} \cdots \overline{M_n}} \quad (1)$$

In Equation 1, M_i (where $i \in \{1, .., n\}$) represents the i-th minterm of the Boolean function f. To allow the parallel computation of the Boolean function minterms, a set of primitive operations implemented in the crossbar is exploited (i.e., COPY, NAND, AND, and NOT operations). A CMOS Control Unit has to opportunely drive the control signals in order to execute the primitive operations through the following steps:

- INA: initializes all memristors to logical 1 (Reset);
- RIN: receives input values;
- CFM: provides configuration for all minterms, copying the values stored on input memristors in parallel (COPY);
- EVM: evaluates all minterms in parallel (NAND);
- EVR: the results of minterms are used to generate $\bar{f}$ (AND);
- INR: inverts $\bar{f}$ to obtain f (NOT);
- SOU: sends out the obtained result.

Thanks to the crossbar configuration, all minterms of the Boolean function can be computed in parallel, then only 7 steps are needed to evaluate the result of the Boolean function. Clearly, each Boolean function has to be mapped in an ad-hoc crossbar, placing memristors depending on the minterms to be computed. While this approach perfectly fits a Boolean function computation (i.e., a stateless logic), it requires some additional components to implement a sequential circuit, such as a mechanism for managing and computing the state. Currently, only off-crossbar registers have been proposed as storage mechanism for the state. To the best of our knowledge, no memristor-based crossbar circuits provide an internal state-propagation logic neither a state storage element.

In the next subsection, we present how we enhance the FBLC architecture for implementing sequential circuits.

B. Stateful FBLC (SFBLC)

In order to implement a sequential circuit by means of a memristor-based crossbar, we extend the FBLC architecture introducing the Stateful FBLC (SFBLC). The main idea is that we can compute the next-state as a classic output, since we can express it as a function of inputs and current state. Consequently, we need a feedback mechanism which allows to use the calculated state as an input for the subsequent computation. Figure 1-a) shows the SFBLC architecture. The *Input Latch* (IL), the *Logic Block* (LB), and the *Output Latch* (OL) are inherited from the classic FBLC. In order to implement the *next-state* function, the so-called *Feedback Latch* (FBL) has been introduced. Basically, such block performs two additional actions, exploiting existing operations on which the FBLC is based.

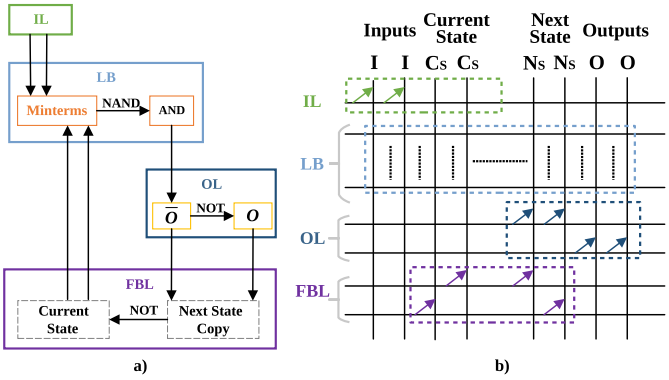

Fig. 1. State propagation logic between crossbar blocks. A 2-bit state is supposed, as example.

- *Next State Copy*: uses the COPY operation to move the computed next-state (treated as output) from the OL block to the FBL block.
- *Current State*: leverages the NOT operation for correctly set input memristors.

As explained in [7], for each input/output of the function, the FBLC architecture requires two memristors: one for the direct value and one for the inverse value of the function input/output. Therefore, the *next state copy* operation saves both the direct and inverse values of the next-state and the *Current State* operation can correctly set the two state-input memristors, which have been accordingly placed in the right rows. In order to support the FBL, we modified the crossbar topology, as shown in Figure 1-b. More in detail, for each state variable - computed as an output - we add two columns and one row with relative memristors, as required by FBLC architecture [7]. Furthermore, two additional memristors are placed within the FBL block for each state variable. State-related input memristors are strategically placed to the FBL - instead of being within the IL block like other inputs - for simplifying the feedback mechanism of the state value. The user inputs are normally received by memristors in the IL. To summarize, for each state variable, two additional rows and two memristors (one per row) are introduced within the FBL.

Accordingly, the control unit of the crossbar has to be extended in order to manage the FBL. Figure 1-a traces the path followed by state-related information. The associated control unit is specified In Figure 2. Compared to the classical

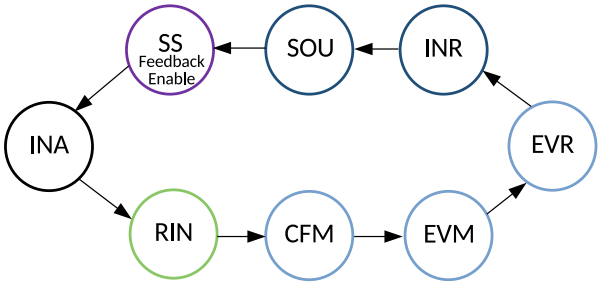

Fig. 2. Modified CMOS Control Logic with the additional state SS.

FBLC control unit, some steps are enriched by new operations

and a new one has been introduced: the *Saving Step* (**SS**). Below, we detail the modifications. In the *INR*, the next-state is computed as a normal FBLC output. Then, in the *SS*, it is stored with a vertical copy, as previously described. If the feedback mechanism is enabled, the *RIN* step transfers the saved state value to the *current-state input* memristors. This operation takes place on the FBL rows. In the *INA*, all memristors of the crossbar are reset except for the FBL ones, when feedback is required. State-related memristors are then reset in the CFM and EVR for letting them receive new state values computed in the INR. Reset operations do not interfere with normal controller operations (minterms configuration and results evaluation).

C. Case Study

To show a significant case study, we take into consideration a Mealy machine that implements a simple *2-bit binary counter*. Such machine exhibits one-bit input, two-bits output and four internal states. The transition graph is shown in Figure 3. For the sake of simplicity, a two-bits encoding scheme was used for state variables.

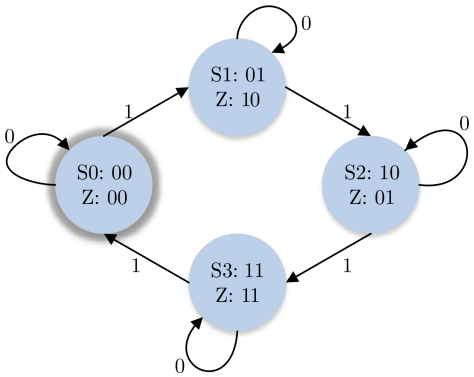

Fig. 3. Transition Graph for considered Mealy Machine with states encoding and per-state output values (Z).

Equations 2 and 3 describe circuit *output* and *next-state*, respectively.

$$\begin{cases} Z_0 = Y_1\overline{X} + \overline{Y_1}X = Y_1 \oplus X \\ Z_1 = Y_0\overline{X} + Y_0\overline{Y_1} + \overline{Y_0}Y_1X \end{cases} \quad (2)$$

$$\begin{cases} Y_0' = Z_1 \\ Y_1' = Z_0 \end{cases} \quad (3)$$

Figure 4 depicts the FBLC architecture with feedback CMOS registers, while Figure 5 reports the memristor crossbar of the above mentioned circuit (i.e., Mealy Machine) implemented by using the FBLC architecture.

On the other hand, Figure 6 shows the crossbar obtained by exploiting the proposed SFBLC architecture.

As can be remarked in Figures 5 and 6, resulting memristor crossbars have different dimensions. The FBLC implementation exploits a 10 rows 14 columns crossbar. On the other hand, the SFBLC uses a 14 rows 14 columns crossbar. Since both FBLC and SFBLC compute the next-state as a primary output, both have 4 columns and 2 rows dedicated to the

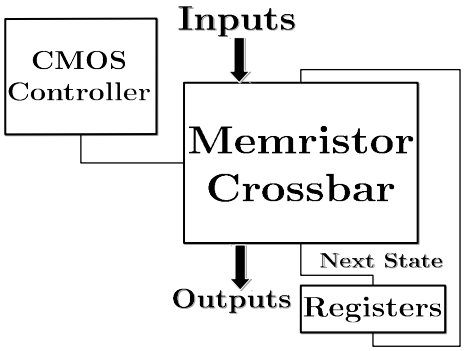

Fig. 4. FBLC with feedback registers.

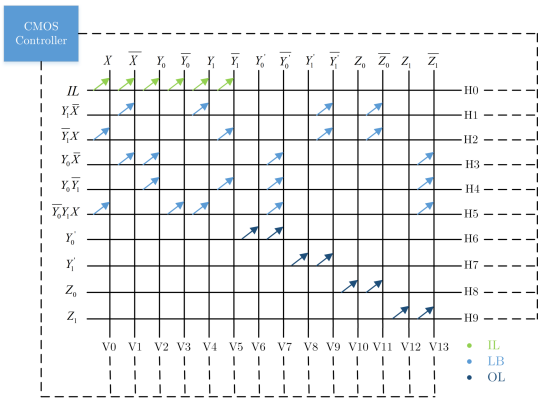

Fig. 5. FBLC memristor-based crossbar for considered Mealy Machine.

state. In addition, for the FBLC architecture, obtained next-state values are copied to classic registers (whether CMOS or memristor-based) and, for the SFBLC, to the FBL block which adds 4 rows and 4 memristors. This is in line with what has been previously described: for each state variable, the SFBLC approach requires two additional rows and two memristors within the crossbar. Finally, the 4 state-related input memristors - that we find in the IL block of the FBLC - are moved to the FBL block, in the SFBLC.

In order to compare the SFBLC architecture against the classic FBLC architecture, leveraging an automated approach allows a rapid and significant experiment campaign. In the next section, we introduce the framework that we extended in order to achieve such goal: *XbarGen*.

D. Extending the XbarGen framework

XbarGen is an automated framework by Traiola et al. [11] for memristor-based crossbars synthesis. It executes a translation of a Boolean function into a simulable digital circuit, targeting the FBLC architecture. XbarGen takes as input the high level description of a Boolean function (i.e., its Boolean expression), and then it executes the mapping to one (or more, if needed) memristor-based crossbars.

Nevertheless, XbarGen is only capable of dealing with combinational circuits. In order to meet the requirements for synthesizing sequential circuits, we enhanced XbarGen. It has remained backward-compatible being able to target both

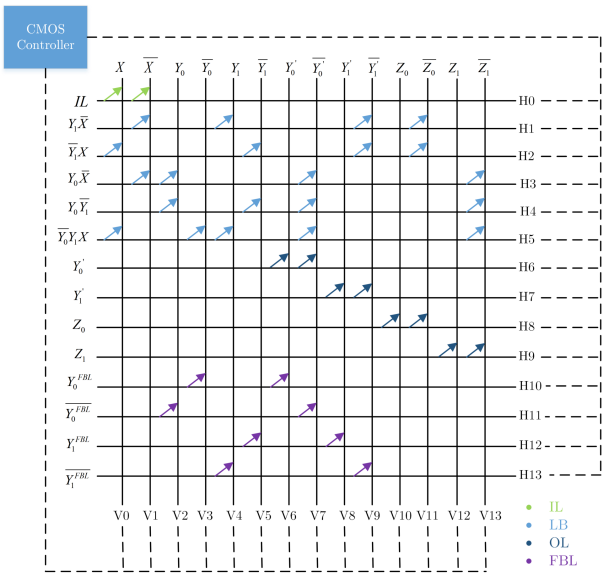

Fig. 6. SFBLC memristor-based crossbar for considered Mealy Machine.

FBLC and SFBLC architectures, and to estimate the related parameters.

More in detail, we embedded in XbarGen the support for the *Berkeley logic interchange format (BLIF) [13]*, which has the required semantic power for stateful elements representation. Several tools, such as Yosys by Clifford Wolf [14], can be used to obtain a BLIF representation of finite state machines, given *RTL Verilog* files. In this way, XbarGen can be easily integrated in the classic digital circuit design flow and adds a significant contribution to the automated design of memristor-based circuits.

III. Evaluation

This section presents the experimental evaluation of the proposed SFBLC architecture against the classical FBLC. In order to investigate challenges and opportunities of using the internal feedback mechanism or external registers, we used the same crossbar structure for both architectures. All circuits were synthesized as a single-crossbar. For achieving an optimized 2-levels form we exploited the ABC tool from Berkeley [15]. Specifically, we used the ABC command *collapse*. A set of nine different sequential circuits from the *ITC'99 Benchmarks* [16] was used for mapping both architectures (i.e., SFBLC and FBLC with feedback registers). In particular, we implemented the feedback registers for the FBLC architecture with the CMOS technology. The *XbarGen* framework [11] was exploited to automatically generate both architectures. Investigated circuit attributes include occupied area, circuit delay (for the computation of a single state of the sequential circuit, e.g., transition from state S0 to state S1 in Figure 3) and power consumption. For studying the latter precisely, the two architectures were compared across a 1000 ns-long simulation, and stimulated by using the same randomly-generated inputs, for a given circuit. The input patterns were randomly generated by means of *ModelSim* ensuring a 100% state coverage and about 50% transition coverage.

A. Experimental setup

The above described configurations (i.e., FBLC and SFBLC) are compared in terms of area, power consumption and delay. To extract these characteristics, we examined the control unit and the memristor crossbar in different manners. The former one (i.e., the control unit) is a standard CMOS circuit. Thus, we performed a classical synthesis using a *65nm CMOS technology library* and we extracted the desired attributes.

On the other hand, we resorted to the Equations proposed in [12] to characterize the memristor-based crossbars.

Power consumption:

$$\begin{cases} P_{xbar} = \sum_{all\ devices} P_R \\ P_R = \frac{V_R^2}{R} \end{cases} \tag{4}$$

where V_R is the voltage applied to the memristor terminals and R depends on the actual resistance value of the memristor (R_{ON} or R_{OFF}). In this study, we focus on the dynamic power consumption (memristor switching activity) which is very fast and simple to obtain exploiting the VHDL description provided by the XbarGen framework.

Occupied area:

$$\begin{cases} A_{xbar} = (N_R + 1) \cdot (N_C + 1) \cdot A_m \\ A_m = 4F^2 \end{cases} \tag{5}$$

where N_R and N_C respectively are the number of rows and columns in the crossbar, A_m is the memristor area and F is the feature size of the technology.

Delay for each computing step (a single step of figure Figure 2):

$$D_{xbar} = T_{sw} + D_{nw} \tag{6}$$

where: T_{sw} is the memristor switching time and D_{nw} is the Resistive-capacitive delay due to the propagation of the signal from the voltage driver to the n^{th} memristor.

By leveraging XbarGen, we obtained information on the memristor-based crossbars such as N_R and N_C. Moreover, we retrieved the VHDL descriptions of the circuits. By simulating the VHDL models, we were able to calculate the total number of switching memristors during the 1000ns, in order to compute the dynamic power consumption. Finally, by integrating technological parameters (i.e., T_{sw}, A_m, V_R, R_{ON} and R_{OFF}), circuit attributes were properly determined. Combining results for both CMOS and memristor portions, we compared the two different architectures for each circuit.

B. Results

In Table I, we report data obtained exploiting XbarGen (rows are sorted by crossbar-size). The columns report in order: number of rows (N_R) and columns (N_C) in the crossbar, number of memristors(N_m), number of latches (N_{Latch}) used in circuit description. The Table II shows the switching activity of the crossbar for the given input patterns. Data was obtained by simulating the VHDL models of the circuits. In detail, $Sw_{Off \to On}$ and $Sw_{On \to Off}$ are the number of memristors switching from R_{OFF} to R_{ON} and vice-versa, respectively.

Furthermore, in Table III, we list the adopted memristor technological parameters. Such parameters refer to those used

TABLE I
MEMRISTOR-BASED CROSSBAR ATTRIBUTES

Exp	N_R		N_C		N_m		N_{Latch}
	FBLC	**SFBLC**	**FBLC**	**SFBLC**	**FBLC**	**SFBLC**	**-**
B02	16	22	16	16	57	63	3
B01	30	36	20	20	136	142	3
B06	37	43	28	28	132	138	3
B08	125	167	110	110	564	606	21
B10	182	216	102	102	856	890	17
B03	309	369	136	136	1712	1772	30
B13	357	461	248	248	1768	1872	52
B09	1537	1593	116	116	25633	25689	28
B07	8377	8475	214	214	112812	112910	49

TABLE II
CROSSBAR SWITCHING ACTIVITY

Exp	$Sw_{off\to on}$		$Sw_{on\to off}$	
	FBLC	**SFBLC**	**FBLC**	**SFBLC**
B02	900	1008	874	998
B01	4026	4256	3966	4246
B06	3647	3841	3592	3819
B08	24801	27349	24573	27270
B10	36587	39288	36212	39225
B03	69722	71054	68869	70979
B13	67402	74702	66742	74554
B09	526058	543686	513427	543621
B07	2312480	2513730	2252030	2513603

by authors of [12]. In Table IV and Table V the obtained results for occupied Area and Dynamic Power Consumption of both CMOS and Crossbar circuits are reported. Finally, in Table VI we report the overhead percentage of the SFBLC compared to the FBLC. Reported values were obtained by using above discussed equations (4, 5 and 6) combined with results in Table I and technological parameters in Table III. Moreover, data in Table V were obtained by combining the switching activity of Table II with equation 4 and *power-related* technological parameter in Table III (i.e., R_{on}, $\frac{R_{off}}{R_{on}}$ and V_R).

The experimental data show that having an integrated mechanism for state propagation (i.e., SFBLC approach) implies some costs in terms of crossbar size, memristors number and power consumption. Some interesting observations can be made. Having an increased crossbar dimension implies an increased area, both for the crossbar and the CMOS Controller, which has to drive a bigger circuit. The number of additional FBL lines is totally predictable, as well as the number of memristors, since the number of required latches is known. These SFBLC parameters can be expressed as:

$$N_R^{(SFBLC)} = N_R^{(FBLC)} + 2 \cdot N_{Latch}$$
$$N_M^{(SFBLC)} = N_M^{(FBLC)} + 2 \cdot N_{Latch}$$

The controller area overhead can be simply computed by means of synthesis tools. Concerning the FBLC, it always has a smaller memristor crossbar but it needs off-crossbar state-latches that lead the CMOS part of the circuit to increase as the number of latches grows. For this reason, concerning the area consumption, the SFBLC approach rarely results in a smaller area. This could happen only if the FBLC external registers were so many, so that their overhead exceeds the SFBLC overhead. Dynamic power consumption follows a similar trend: both a bigger crossbar and a bigger CMOS-Controller for the SFBLC architecture result in higher consumption, as expected. While the CMOS Controller power consumption overhead is caused by the additional voltage drivers, the crossbar overhead is due to the memristors used for the next-state copy within the FBL. The other crossbar memristors exhibit the same activity for both FBLC and SFBLC. Another important factor is the time delay. Both FBLC and SFBLC need a constant number of computation steps, but, due to the additional state (SS), the SFBLC architecture has an overhead of about 14.2% compared to FBLC, as shown in Table VII. Contrary to area and power consumption, there is no compromise: the SFBLC always has a higher delay. Please note that for the delay comparison we only report data for the crossbars.

Reported experiments only target circuit with simple/medium logical complexity because of the limited scalability of the single-crossbar approach. In fact, for bigger circuits (i.e., b07 and b09) the crossbar begins to be remarkably large for both the architectures. It is worth to remark that this phenomenon is due to the SOP implementation and not to the proposed approach. Besides, such issue can be mitigated by using decomposition.

TABLE III
TECHNOLOGICAL PARAMETERS

$F(nm)$	$A_m(\mu m^2)$	$R_{on}(M\Omega)$	$\frac{R_{off}}{R_{on}}$	$V_R(V)$	$T_{sw}(ns)$	D_{nw}
65	**0.0169**	**100**	**7k**	**2.1**	**1.71**	**NEGL**

TABLE IV
AREA COMPARISON (μm^2)

Exp	FBLC			SFBLC		
	CMOS	**XBAR**	**TOTAL**	**CMOS**	**XBAR**	**TOTAL**
B02	577.19	4.88	582.08	635.96	6.61	642.57
B01	892.32	11	903.32	970.84	13.13	983.97
B06	1024.92	18.62	1043.54	1179.36	21.56	1200.92
B08	4115.28	236.36	4351.64	4296.76	315.15	4611.91
B10	5181.28	318.55	5499.83	5374.72	377.73	5752.45
B03	8158.8	717.74	8876.54	8368.88	856.66	9225.54
B13	10794.16	1506.5	12300.66	11040.64	1944.14	12984.78
B09	32773.52	3041.09	35814.6	32924.32	3151.82	36076.13
B07	171488.19	30441.46	201929.66	171846.99	30797.55	202644.54

TABLE V
DYNAMIC POWER CONSUMPTION COMPARISON(μW)

Exp	FBLC			SFBLC		
	CMOS	**XBAR**	**TOTAL**	**CMOS**	**XBAR**	**TOTAL**
B02	346.19	0.04	346.23	429.26	0.04	429.31
B01	592.55	0.17	592.72	673.25	0.19	673.44
B06	684.61	0.16	684.76	817.78	0.17	817.95
B08	2537.73	1.08	2538.81	2952.40	1.20	2953.60
B10	2900.57	1.60	2902.17	3441.20	1.73	3442.93
B03	4484.68	3.04	4487.72	5337.00	3.13	5340.13
B13	5821.45	2.94	5824.39	7068.60	3.29	7071.89
B09	19409.70	22.65	19432.35	21739.90	23.98	21763.88
B07	107645.13	99.33	107744.46	118182.00	110.87	118292.87

IV. CONCLUSION AND FUTURE DIRECTION

Several memristor-based architectures have been proposed in the literature. None of these were focused on the realization of sequential circuits with an integrated state feedback mechanism entirely based on memristor-based crossbars. Therefore,

TABLE VI
OVERHEAD PERCENTAGE FOR AREA AND POWER CONSUMPTION

Exp	Area Overhead			Power Overhead		
	CMOS	XBAR	TOTAL	CMOS	XBAR	TOTAL
B02	10.18%	35.29%	10.39%	24.00%	14.19%	23.99%
B01	8.80%	19.35%	8.93%	13.62%	7.06%	13.62%
B06	15.07%	15.79%	15.08%	19.45%	6.32%	19.45%
B08	4.41%	33.33%	5.98%	16.34%	10.98%	16.34%
B10	3.73%	18.58%	4.59%	18.64%	8.32%	18.63%
B03	2.57%	19.35%	3.93%	19.01%	3.06%	18.99%
B13	2.28%	29.05%	5.56%	21.42%	11.70%	21.42%
B09	0.46%	3.64%	0.73%	12.01%	5.88%	12.00%
B07	0.21%	1.17%	0.35%	9.79%	11.61%	9.79%

TABLE VII
DELAY COMPARISON (NS)

	FBLC	SFBLC	Overhead
Xbar	11.97	13.68	14.29%

in this work, we proposed the Stateful Fast Boolean Logic Circuit (SFBLC), as an extension of the FBLC memristor-based crossbar architecture [7], to implement Finite State Machines. We extended the XbarGen framework [11], by adding sequential circuits synthesis capability and BLIF file management, for obtaining an automated and integrated framework which adds a significant contribution to the design of memristor-based circuits. We performed experiments on publicly available benchmarks for comparing the proposed architecture against its predecessor, the FBLC. Experiments show acceptable overheads (up to 15% for the area and up to 24% for the dynamic power consumption and 14% for the delay) compared to the opportunity of advancing the migration process towards new integrated memristive architectures with the aim of exploiting such promising technology to cope with CMOS related issues. Finally, we remarked that complex logic circuits turn out to be not manageable with a single-crossbar-based circuit. Therefore, as a future work, we want to extend the presented methodology by investigating possible hierarchical synthesis solutions and identifying benefits and challenges.

REFERENCES

[1] S. Lloyd, "Ultimate physical limits to computation," *Nature*, vol. 406, no. 6799, pp. 1047–1054, Aug 2000. [Online]. Available: http://dx.doi.org/10.1038/35023282

[2] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nat Nano*, vol. 8, no. 1, pp. 13–24, Jan 2013. [Online]. Available: http://dx.doi.org/10.1038/nnano.2012.240

[3] S. S. Sarwar, S. A. N. Saqueb, F. Quaiyum, and A. B. M. H. U. Rashid, "Memristor-based nonvolatile random access memory: Hybrid architecture for low power compact memory design," *IEEE Access*, vol. 1, pp. 29–34, 2013.

[4] G. Indiveri, B. Linares-Barranco, R. A. Legenstein, G. Deligeorgis, and T. Prodromakis, "Integration of nanoscale memristor synapses in neuromorphic computing architectures," *CoRR*, vol. abs/1302.7007, 2013. [Online]. Available: http://arxiv.org/abs/1302.7007

[5] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, and J. van Lunteren, "Memristor based computation-in-memory architecture for data-intensive applications," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1718–1725.

[6] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, no. 6, pp. 1165–1172, Mar 2005. [Online]. Available: https://doi.org/10.1007/s00339-004-3149-1

[7] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[8] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *2009 IEEE/ACM International Symposium on Nanoscale Architectures*, July 2009, pp. 33–36.

[9] L. Gao, F. Alibart, and D. B. Strukov, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 115–119, March 2013.

[10] H. A. D. Nguyen, L. Xie, M. Taouil, S. Hamdioui, and K. Bertels, "Synthesizing hdl to memristor technology: A generic framework," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2016, pp. 43–48.

[11] M. Traiola, M. Barbareschi, A. Mazzeo, and A. Bosio, "Xbargen: A memristor based boolean logic synthesis tool," in *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Sept 2016, pp. 1–6.

[12] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "A mapping methodology of boolean logic circuits on memristor crossbar," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[13] U. of California Berkeley, "Berkeley logic interchange format (blif)," https://www.cse.iitb.ac.in/ supratik/courses/cs226/spr16/blif.pdf.

[14] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[15] *Abc User Guide*. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/

[16] F. Corno, M. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *Design Test of Computers, IEEE*, vol. 17, no. 3, pp. 44–53, Jul 2000.