# Formal Design Space Exploration for memristor-based crossbar architecture

Marcello Traiola, Mario Barbareschi, Alberto Bosio

# Formal Design Space Exploration for Memristor-based Crossbar Architecture

Marcello Traiola[1], Mario Barbareschi[2], Alberto Bosio[1]
[1]LIRMM - University of Montpellier / CNRS - France - Email: {firstname.lastname}@lirmm.fr.
[2] DIETI - University of Naples Federico II - Italy - Email: mario.barbareschi@unina.it

***Abstract*—The unceasing shrinking process of CMOS technology is leading to its physical limits, impacting several aspects, such as performances, power consumption and many others. Alternative solutions are under investigation in order to overcome CMOS limitations. Among them, the memristor is one of promising technologies. Several works have been proposed so far, describing how to synthesize boolean logic functions on memristors-based crossbar architecture. However, depending on the synthesis parameters, different architectures can be obtained. Design Space Exploration (DSE) is therefore mandatory to help and guide the designer in order to select the best crossbar configuration. In this paper, we present a formal DSE approach. The main advantage is that it does not require any simulation and thus it avoids any runtime overheads. Preliminary results show the huge gain in runtime compared to simulation-based DSE.**

**Index terms - Memristor crossbar, Design Space Exploration, Boolean Functions, Circuit Synthesis**

## I. Introduction

Today's computing devices are based on the CMOS technology, that is the subject of the famous Moore's Law [1], predicting that the number of transistors in an integrated circuit will be doubled every two years. Despite the advantages of the technology shrinking, we are facing the physical limits of CMOS. Among the multiple challenges arising from technology nodes lower than 20 nm, we can highlight the high leakage current (i.e., high static power consumption), reduced performance gain, reduced reliability, complex manufacturing process leading to low yield, complex testing process, and extremely costly masks [2], [3], [4], [5].

Additionally, the expected never-ending increasing of performances is indeed no longer true. Looking in more detail, the classical computer architectures, either Von Neumann or Harvard, divide the computational unit (i.e., CPU) from the storage element (i.e., memory). Therefore, data have to be transferred inside the computational element in order to be processed and then transferred back to be stored. The main problem of this paradigm is the bottleneck due to the data transfer time limited by the bandwidth. For instance, transferring one TeraByte at the rate of 1Gbit/second requires more than two hours.

Many new technologies are under investigation, among them the memristor is a promising one [6]. Indeed, being a non-volatile device able to act as both storage and information processing unit, the memristor presents many advantages: CMOS process compatibility, lower cost, zero standby power, nanosecond switching speed, great scalability, high density and non-volatile capability [7], [8]. Thanks to its nature (i.e., computational as well as storage element), the memristor is exploited in different kind of applications, such as neuromorphic systems [9], non-volatile memories [10], computing architectures for data-intensive applications [11].

A fundamental component of any kind of computing architecture is the implementation of boolean logic functions. In [12], the authors proposed a methodology for the synthesis of boolean logic functions on a memristor-based crossbar. Their work showed that is possible to implement any kind of boolean function on a memristor-based crossbar. In our previous work, we illustrated a methodology to automatically map an arbitrary boolean function to a memristor-based crossbar implementation [13]. By applying different minimization tools and different synthesis parameters, we also showed that each obtained architecture is strongly dependent on them. Design Space Exploration (DSE) is therefore mandatory to help and guide the designer in order to select the best architecture.

Bearing in mind such consideration, in this paper, we present a formal DSE approach that aims to calculate interesting circuits attributes avoiding simulation campaigns. We propose an algorithmic method to estimate both workload independent attributes (e.g. performance, area, etc.) and workload dependent ones. In particular, we estimate the power consumption of a given memristor-based crossbar architecture (the Fast Boolean Logic Circuit [12]) providing both a lower and an upper bound for the power consumption and an error estimation.

The remainder of the paper is structured as following. Section II presents the state of the art and provides the required background about the memristor-based computation. Section III presents the synthesis flow and the design space exploration framework, while the Section IV gives the experimental results. Finally, the Section V draws the conclusions.

## II. Background and state-of-the-art

In this section we provide the basics about the memristor modeling, as well as the way the memristor can be exploited to implement an arbitrary boolean function.

### A. Memristor model

The memristor is a non-linear electrical component characterized by a variable electrical resistance, which value depends on the history of the charge flowed through the device itself. As we aim to implement a digital circuit, we

refer to the memristor Voltage-Current relation depicted in Figure 1, detailed in [14], as the best solution for modeling the memristor's behavior (i.e., thanks to the ideal response to a pulse-wave). Such a model considers that the voltage applied to the memristor's terminals does not change the device resistance until it crosses one of the two thresholds $V_{th}$. In the adopted ideal model, they are symmetrically defined.

We resort to the Snider Boolean Logic (SBL) [14] convention whereby a lower resistance (steeper curve denoted as $R_{ON}$) represents a 0-logic while an higher resistance (lower slope curve denoted as $R_{OFF}$) represents a 1-logic.

Two basic operations can be performed, defined as SET and RESET. The former allows to program the memristor to $R_{ON}$, hence at 0-logic, while the latter performs the memristor switching to $R_{OFF}$, that corresponds to 1-logic. The Figure 1 depicts SET and RESET operations as described by Xie et al. in [12].

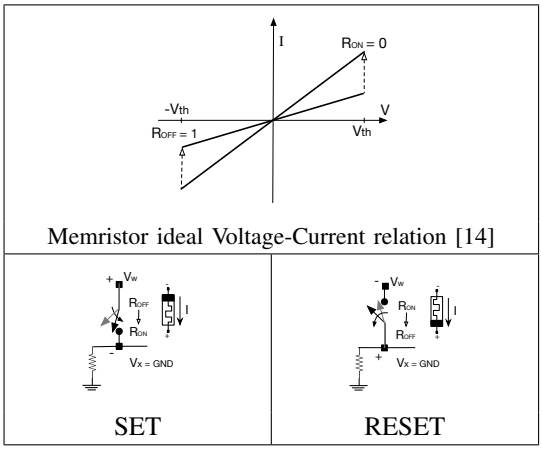



Fig. 1: Set and Reset operations [12].

### B. Fast Boolean Logic Circuits

Snider proposed in [14] a design methodology to implement boolean functions on memristor-based crossbar. The proposed approach was then improved by Xie et al. in [12]. Let us briefly recall their proposition referring to it as Fast Boolean Logic Circuit (***FBLC***). First, the logic circuit requires that the Boolean function is expressed as sum of products:

$$f = M_1 + M_2 + ... + M_n = \overline{\overline{M_1} \cdot \overline{M_2} \cdot \cdots \cdot \overline{M_n}}$$

NAND, AND, NOT

Then, as Figure 2-a shows, FBLC is divided in blocks, useful to accomplish FSM's steps (Figure 2-b) which are:

INA: **IN**itialize **A**ll the memristors to $R_{OFF}$;
RI: the input block **R**eceives the **I**nputs;
CFM: **C**on**F**igure all **M**interms simultaneously, in parallel;
EVM: **EV**aluate all **M**interms simultaneously (NAND);
EVR: **EV**aluate **R**esults: $\overline{F}$ is calculated (AND);
INR: **IN**vert **R**esults: $\overline{F}$ need to be inverted to achieve $f$
SO: **S**end **O**utputs: the result captured in OL is sent out.

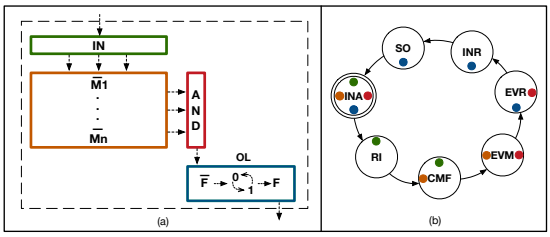



Fig. 2: Fast Boolean Logic Circuit.

Below, the description of the blocks:

- Input box: where inputs are stored during the RI step;
- NAND box: where minterms are configured during CFM and evaluated during EVM;
- AND box: where results of EVM are stored and AND operation is performed during EVR;
- Output box: where results of EVR are stored and inversion operation is performed during INR;

For the purpose of realizing each step of the FSM, the authors proposed some *primitive operations* [12]

Each of these operations can be performed using as many input and output memristors as desired.

By driving the crossbar's nano-wires with the right voltages during each step, it is possible to evaluate a boolean function in a constant number of steps.

## III. Synthesis Flow and DSE

As described in [12], FBLC approach implements a boolean function as a Sum-of-Product (SoP). Thus, the resulting crossbar has to be configured accordingly to the function's minterms.

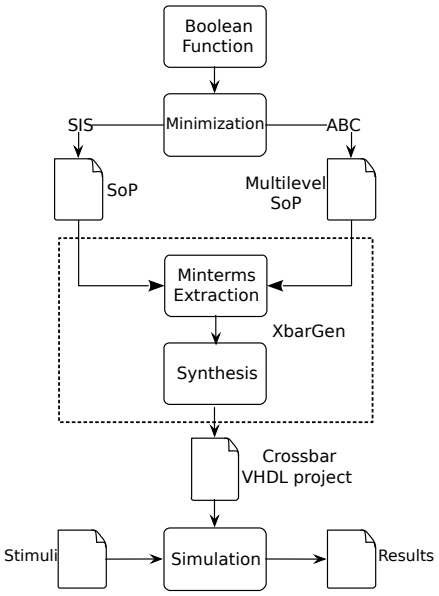



Fig. 3: Synthesis flow.

The proposed synthesis flow is depicted in the Figure 3. The input of the flow is the target boolean function that is

minimized by using two different synthesis tools (i.e., SIS [16] and ABC [15] by Berkeley). We exploited two different tools in order to estimate the impact of different synthesis parameters and algorithms on the circuit characteristics (i.e., performance, area, power consumption, etc.). More in detail, SIS is exploited for generating 2-levels logical networks while ABC is exploited for generating multi-levels logical networks. The result is the boolean function minimized and described as a set of minterms. As described above, different descriptions can be obtained.

The subsequent step is the mapping of minimized boolean function onto a crossbar-based memristor circuit. The tool XbarGen [13] is able to extract the function minterms from the generated representation in order to analyze them and build the corresponding FBLC circuit. The result is a set of VHDL files modeling the crossbar circuit. Finally, the crossbar VHDL model can be simulated by using any available logic simulator.

During the mapping process, XbarGen extracts the crossbar attributes that will be exploited by the proposed formal DSE approach. Let us first detail those attributes before moving to the DSE description. They can be divided in two main categories, namely the workload independent and workload dependent. Next subsections describe both of them and last subsection details the formal DSE.

### A. Workload independent attributes

The workload independent attributes do not need any simulation (i.e., we do not have to simulate the crossbar VHDL model) to be evaluated. They are extracted by XbarGen during the mapping process and they are formalized as follows:

**Number of memristors in the circuit** defined by the following equation:

$$N_m = \sum_j \left[ 2 * N_{in}(l_j) + \sum_i (N_{occ}(m_i, l_j)) + \sum_i (N_{lit}(m_i) * p_{ij}) + 2 * N_{out}(l_j) \right] \quad (1)$$

**Total area of the circuit** defined by the following equation:

$$Area = \sum_j \left\{ [2 * N_{in}(l_j) + 2 * N_{out}(l_j)] * \left[ 1 + \sum_i (p_{ij}) + N_{out}(l_j) \right] \right\} \quad (2)$$

**Number of crossbars**, i.e. $N_C$

**Response time of the circuit** defined by the following equation:

$$RespTime = T_C * N_C \quad (3)$$

given that:

- Indexes $i$ and $j$ run on minterms and crossbars respectively
- $N_{in}$ and $N_{out}$ are the number of inputs and outputs respectively;
- $N_{occ}(m_i, l_j)$ is the number of occurrence of i-th minterm in j-th crossbar;
- $N_{lit}(m_i)$ is the number of literals of i-th minterm;
- $p_{ij}$ is equal to 1 if the *i-th* minterm is present in the *j-th* crossbar, otherwise it is equal to 0;
- $T_C$ is the 'Latency' of a Crossbar;
- $N_C$ is the Number of Crossbars in the circuit.

### B. Workload dependent attributes

The workload dependent attributes require the simulation of the generated VHDL circuits to be evaluated. In this work, we consider the **power consumption** as workload dependent attributes formalized as:

$$P = \sum_j \left[ N_{up_j} \cdot C_{up} + N_{down_j} \cdot C_{down} \right] \quad (4)$$

given that:

- index $j$ runs on crossbars;
- $N_{up_j}$ and $N_{down_j}$ are the number of memristors in the *j-th* crossbar that switch from '0' to '1' and from '1' to '0' respectively;
- $C_{up}$ and $C_{down}$ are the power consumption of a memristor switching from '0' to '1' and from '1' to '0' respectively.

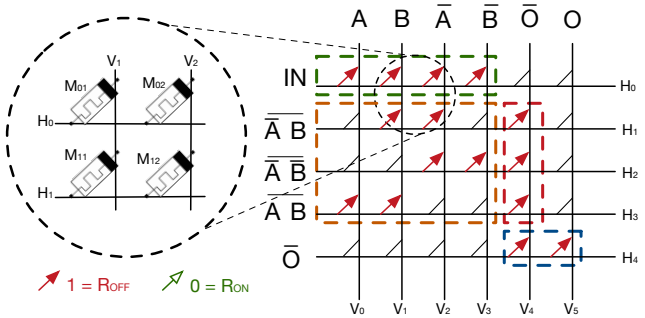
It is worth to note that $N_{up_j}$ and $N_{down_j}$ depend on the applied workload.

### C. Formal DSE

The main goal of the proposed DSE is the characterization of the synthesized crossbars w.r.t. the above identified attributes. The idea is to avoid any simulation to speed up the DSE. Clearly, for the workload independent attributes the formal DSE is straightforward since it is enough to exploit the equations (1),(2) and (3).

The challenging issue is determining the actual power consumption. Even if the power consumption is a workload dependent attribute, we will show how to compute two bounds that cannot be exceeded by the actual power consumption: a ***worst*** case bound and a ***best*** case bound. It is worth to emphasize that such bounds will be computed without any simulation.

Referring to the equation (4), the idea that we exploit is to identify within the crossbar the elements that do not depend on actual inputs and manage those which are dependent on actual inputs. Thus, we can observe - as Figure 4 shows - that the architecture has a first RESET stage (INA) in which all the memristor in the circuit are set to '1'.




$$O = AB + \overline{A}B + \overline{AB} = \overline{\overline{AB} \cdot \overline{\overline{A}B} \cdot \overline{A\overline{B}}}$$

Fig. 4: Reset stage.

Therefore, during this stage, we have only the contribution of $N_{up_j} \cdot C_{up}$ while, during the rest of the computation, only $N_{down_j} \cdot C_{down}$ contributes to the power consumption. Moreover, in both worst and best case scenarios, we consider a concatenation of executions providing inputs which trigger the worst and the best case respectively. Bearing in mind this, we can observe that, whether $N_{down_j}$ memristors switch from '1' to '0' during the computation, the RESET phase has to switch the same number of memristors from '0' to '1'. Therefore, considering both worst and best case, we can assume that the two contributions are equal:

$$N_{up_j} = N_{down_j} \tag{5}$$

In order to estimate this contribution let us consider that a crossbar can be divided in 4 parts, as depicted in the Figure 2-a. Hence we can assume the following:

- Concerning the green IN box, it is clear that **half** of the input memristors are going to switch during each execution of the circuit. It is worth noting that for each input $x_i$ of the function we have 2 memristors: $x_i$ and $\overline{x_i}$;
- The same consideration is true for the output memristors, in the blue OL box.

Therefore such two blocks of memristors can be evaluated, in terms of switching memristors, independently from the actual input values.
Thus, we are able to rewrite the equation 5 as follows:

$$N_{up_j} = N_{down_j} = N_{in}(l_j) + N_{out}(l_j) + N_{int_j} \tag{6}$$

where:

- $N_{int_j}$ is the number of memristors that belong to the minterm boxes NAND and AND within the *j-th* crossbar that switches during the computation:

$$N_{int_j} = (N_{m_{NAND}} + N_{m_{AND}})_j \tag{7}$$

Let us now discuss about the remaining orange NAND and the red AND boxes. The memristors of these parts switch, accordingly with the actual input values. For them, the goal is to find the two bounds.

It is worth to highlight that the number of switching memristors in the AND box depends on which memristors switch in the NAND box. Therefore, the best and worst cases are computed by considering only the NAND box.

Bearing in mind that half of the input memristors will eventually switch $1 \rightarrow 0$ during each execution of the circuit, in order to find the best and worst input vectors we count, for each vertical nanowire, the number of memristors in the NAND box. For each couple of literals $x_i$ and $\overline{x_i}$ (vertical nanowires) we consider, as for the best case, the one that leads to the minimum number of memristors and, as for the worst case, the one that leads to the maximum.

Finally, we compute how many AND memristors will switch using the selected input vectors: since the minterms box is performing a NAND operation, if a minterm has at least one literal among those in the selected input vectors, the corresponding memristor in the AND box will not switch, otherwise it will.

It is worth to note that, in order to compute $(N_{m_{NAND}}, N_{m_{AND}})_{worst/best}$ we do not need the truth table of the function; we only count the number of memristors in the NAND box in order to find the best and the worst combinations of inputs and then we verify if each minterm will be whether '0' or '1'. Therefore the algorithm has a *linear* complexity.

This is a great improvement compared to doing a simulation with all the combinations of inputs that would lead to a complexity $\theta(2^n)$, with n the number of inputs

The algorithm 1 realizes what we explained so far.

**Data:** Literals, Minterms
**Result:** $(N_{m_{NAND}}, N_{m_{AND}})_{worst/best}$
**for** *each crossbar j* **do**
  **for** *each input* ($x_i$, $\overline{x_i}$) **do**
    **if** $\overline{x_i}$*.NandOccurrence* $>$ $x_i$*.NandOccurrence* **then**
      vectorWorstCase $\leftarrow \overline{x_i}$;
      vectorBestCase $\leftarrow x_i$;
      $(N_{m_{NAND}})_{worst}$ += $\overline{x_i}.NandOccurrence$;
      $(N_{m_{NAND}})_{best}$ += $x_i.NandOccurrence$;
    **else**
      vectorWorstCase $\leftarrow x_i$;
      vectorBestCase $\leftarrow \overline{x_i}$;
      $(N_{m_{NAND}})_{worst}$ += $x_i.NandOccurrence$;
      $(N_{m_{NAND}})_{best}$ += $\overline{x_i}.NandOccurrence$;
    **end**
  **end**
  **for** *each minterm m* **do**
    **if** *m does not contain any element of vectorWorstCase* **then**
      $(N_{m_{AND}})_{worst} = (N_{m_{AND}})_{worst} + 1$;
    **end**
    **if** *m does not contain any element of vectorBestCase* **then**
      $(N_{m_{AND}})_{best} = (N_{m_{AND}})_{best} + 1$;
    **end**
  **end**
**end**

**Algorithm 1:** Get best and worst case.

Let us resort to the example of Figure 5 to illustrate the above considerations. It is easy to see that, depending on

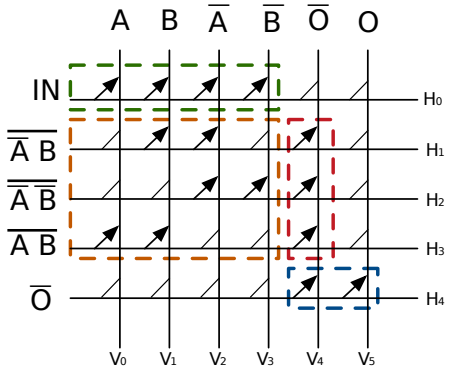



Fig. 5: FBLC example.

the chosen input vector, the number of switching memristors changes. The table I shows the different scenarios depending

on the applied input vectors.
- From $1^{st}$ to $4^{th}$ column we have input vectors;
- In the *"NAND box"* column the nanowire coordinates of the switching NAND memristors are reported in the form vX-hY;
- In the *"AND box"* column the nanowire coordinates of the switching AND memristors are reported in the form vX-hY;
- In the *"Tot"* column the sum of the two previous columns is reported.

| A | B | $\overline{A}$ | $\overline{B}$ | NAND box | AND box | Tot |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | v0-h3, v1-h1, v1-h3 | v4-h2 | 4 |
| 0 | 1 | 1 | 0 | v0-h3, v3-h2 | v4-h1 | 3 |
| 1 | 0 | 0 | 1 | v1-h1, v1-h3, v2-h1, v2-h2 | // | 4 |
| 1 | 1 | 0 | 0 | v2-h1, v2-h2, v3-h2 | v4-h3 | 4 |

TABLE I: Switching memristors.

The worst case is estimated by choosing the input vector that makes switch the maximum number of memristors in the NAND box, like explained so far. In the example, the tool selects the inputs "1001" that make $N_{m_{NAND}}$ be equal to 4. Then $N_{m_{AND}}$ is also calculated *for that particular case* (0 in the example).

The best case is estimated by choosing the input vector that makes switch the minimum number of memristors in the NAND box. In the example above, the tool selects the inputs "0110" that make $N_{m_{NAND}}$ be equal to 2. Then $N_{m_{AND}}$ is also calculated *for that particular case* (1 in the example).

*Error estimation:* Since without a simulation it is not possible to know how many AND memristors would have switched if the tool had chosen another input vector, we perform an error estimation. Observing the example in Figure 6 and the relative table II, the importance of estimating the error become clear.

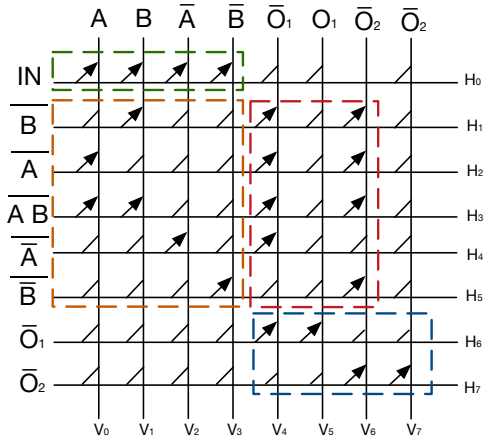



Fig. 6: FBLC estimating error.

In the worst case, the tool would choose the input vector "0011" that makes switch 4 NAND memristor and 2 AND ones; however the real worst case is given by the input vector "1100". Indeed, despite it would make switch only 2 NAND memristor, it would make switch 6 AND memristors.

| A | B | $\overline{A}$ | $\overline{B}$ | NAND box | AND box | Tot |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | v0-h2, v0-h3, v1-h1, v1-h3 | v4-h4, v6-h5 | 6 |
| 0 | 1 | 1 | 0 | v0-h2, v0-h3, v3-h5 | v4-h1, v4-h4, v6-h1 | 6 |
| 1 | 0 | 0 | 1 | v1-h1, v1-h3, v2-h4 | v4-h2, v6-h2, v6-h5 | 6 |
| 1 | 1 | 0 | 0 | v2-h4, v3-h5 | v4-h1, v4-h2, v4-h3, v6-h1,v6-h2, v6-h3 | 8 |

TABLE II: Switching memristors.

In the best case, the tool would choose the input vector "1100" that make switch only 2 NAND memristors but 6 AND ones; however, this is not the real best case. Indeed, despite the other input vectors would make switch more NAND memristors, they would make switch less AND memristors producing better results.

Therefore we estimate the error in a conservative manner as follows:

- in the worst case is the difference between total number of AND memristors and the actual number of switched AND memristors:

$$E_{worst} = N_{m_{AND_{max}}} - N_{m_{AND}} \tag{8}$$

- in the best case is just $N_{m_{AND}}$:

$$E_{best} = N_{m_{AND}} \tag{9}$$

In both best and worst cases *the error is overestimated*. Referring to the above example (Figure 6) we note that:

- $E_{worst}$ is actually equal to $8 - 6 = 2$ but the tool will overestimate it:

$$E_{worst} = N_{m_{AND_{max}}} - N_{m_{AND}} = 8 - 2 = 6.$$

- $E_{best}$ is actually equal to 2 but the tool will overestimate it:

$$E_{best} = N_{m_{AND}} = 6.$$

Moreover, we must make another observation about the estimation of the power consumption in presence of more crossbars: in both best and worst cases we assume that each crossbar of the circuit gets as input the vector that triggers the best (worst) case. Nevertheless, each crossbar's input depends on the output of the previous one which has a low probability to produce exactly the output we estimated.

In conclusion we estimate the power consumption bounds in both best and worst cases as follows:

$$P = (C_{up} + C_{down}) \cdot \sum_j \left[ N_{in}(l_j) + N_{out}(l_j) + N_{int_j} \right] \tag{10}$$

where:

- $N_{int_j}$ depends on the input vector and can be estimated in both best and worst cases.

## IV. Experimental Results

This section provides experimental results achieved by the flow discussed in section III. A bunch of combinatorial circuits are used as benchmarks, details about circuits characteristics are available in [17]. Tables III and IV report the achieved results for the single crossbar and for the multiple crossbars respectivley.

For each circuit, Tables report the number of inputs (*IN*), minterms, memristors ($N_m$), the area, the estimated power

| Exp. | IN | Single Crossbar | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Minterms | $N_m$ | Area | Time | Xbars | $P_{worst}$ | $E_{worst}$ | $P_{best}$ | $E_{best}$ | DSE Time (ms) | SimTime Overhead |
| xor5 | 5 | 16 | 108 | 216 | 7 | 1 | 46 | 16 | 47 | 1 | 2,18945 | 1286,08% |
| squar5 | 5 | 30 | 261 | 1014 | 7 | 1 | 92 | 85 | 88 | 4 | 8,25324 | 669,46% |
| rd53 | 5 | 32 | 192 | 576 | 7 | 1 | 90 | 32 | 76 | 6 | 3,66841 | 1159,77% |
| con1 | 7 | 9 | 50 | 216 | 7 | 1 | 23 | 8 | 21 | 2 | 1,93385 | 992,99% |
| 5xp1 | 7 | 70 | 385 | 2754 | 7 | 1 | 168 | 74 | 159 | 16 | 8,86976 | 919,19% |
| Z5xp1 | 7 | 128 | 1506 | 4726 | 7 | 1 | 466 | 575 | 471 | 6 | 183,385 | 210,45% |
| rd73 | 7 | 141 | 1001 | 2900 | 7 | 1 | 486 | 141 | 417 | 43 | 19,3369 | 1260,28% |
| misex1 | 8 | 18 | 132 | 780 | 7 | 1 | 60 | 32 | 43 | 3 | 4,22439 | 751,92% |
| rd84 | 8 | 255 | 2475 | 6240 | 7 | 1 | 1036 | 411 | 1029 | 1 | 120,229 | 543,06% |
| ex5 | 8 | 256 | 9810 | 45440 | 7 | 1 | 1129 | 7586 | 1125 | 30 | 32241,5 | 11,66% |
| 9sym | 9 | 87 | 629 | 1780 | 7 | 1 | 277 | 86 | 268 | 2 | 10,8552 | 1347,78% |
| clip | 9 | 166 | 1078 | 4816 | 7 | 1 | 506 | 164 | 417 | 9 | 24,1155 | 1098,00% |
| Z9sym | 9 | 420 | 4220 | 8440 | 7 | 1 | 1900 | 420 | 1900 | 0 | 143,374 | 790,09% |
| apex4 | 9 | 438 | 5489 | 24678 | 7 | 1 | 2002 | 1716 | 1775 | 4 | 1823,29 | 82,04% |
| sao2 | 10 | 58 | 529 | 1764 | 7 | 1 | 302 | 78 | 151 | 2 | 10,1846 | 1170,25% |
| table3 | 14 | 175 | 2702 | 10640 | 7 | 1 | 1252 | 645 | 805 | 0 | 366,519 | 190,27% |
| alu4 | 14 | 954 | 8306 | 42372 | 7 | 1 | 3933 | 973 | 3428 | 28 | 588,95 | 441,86% |
| misex3 | 14 | 1426 | 15559 | 80696 | 7 | 1 | 8801 | 1828 | 4932 | 2 | 2618,36 | 260,61% |
| b12 | 15 | 76 | 412 | 4128 | 7 | 1 | 245 | 75 | 118 | 26 | 11,4098 | 780,89% |
| table5 | 17 | 158 | 2566 | 11136 | 7 | 1 | 1220 | 606 | 740 | 0 | 353,965 | 185,95% |
| duke2 | 22 | 87 | 1103 | 11934 | 7 | 1 | 618 | 242 | 247 | 4 | 63,8902 | 426,18% |
| cordic | 23 | 1206 | 19625 | 60450 | 7 | 1 | 9789 | 1204 | 8634 | 2 | 1797,42 | 591,21% |
| misex2 | 25 | 29 | 303 | 4128 | 7 | 1 | 169 | 27 | 110 | 3 | 11,1904 | 562,40% |
| vg2 | 25 | 110 | 980 | 7854 | 7 | 1 | 534 | 110 | 350 | 14 | 23,0245 | 1030,28% |
| apex2 | 39 | 1035 | 15610 | 85198 | 7 | 1 | 12719 | 1075 | 2068 | 252 | 1292,45 | 591,65% |
| seq | 41 | 1066 | 14502 | 167504 | 7 | 1 | 9831 | 1457 | 3222 | 8 | 2022,78 | 327,43% |
| apex1 | 45 | 206 | 3018 | 44000 | 7 | 1 | 1175 | 1103 | 776 | 36 | 824,237 | 97,92% |
| apex3 | 54 | 280 | 3498 | 68848 | 7 | 1 | 1393 | 1019 | 1090 | 4 | 707,142 | 132,90% |
| e64 | 65 | 65 | 2470 | 34060 | 7 | 1 | 2212 | 65 | 195 | 2 | 81,5855 | 802,18% |
| apex5 | 117 | 1160 | 8004 | 495908 | 7 | 1 | 5150 | 1184 | 1844 | 174 | 940,781 | 274,71% |
| o64 | 130 | 65 | 457 | 17554 | 7 | 1 | 261 | 65 | 196 | 65 | 69,1192 | 151,54% |

TABLE III: Experiments Single Crossbar

| Exp. | IN | Multiple Crossbars | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Minterms | $N_m$ | Area | Time | Xbars | $P_{worst}$ | $E_{worst}$ | $P_{best}$ | $E_{best}$ | DSE Time (ms) | SimTime Overhead |
| xor5 | 5 | 22 | 156 | 894 | 42 | 6 | 80 | 20 | 70 | 14 | 7,20985 | 536,45% |
| rd53 | 5 | 58 | 414 | 4790 | 56 | 8 | 215 | 56 | 183 | 40 | 20,3347 | 468,59% |
| squar5 | 5 | 71 | 515 | 6546 | 56 | 8 | 274 | 69 | 228 | 56 | 25,9582 | 476,89% |
| con1 | 7 | 19 | 139 | 804 | 35 | 5 | 71 | 19 | 62 | 13 | 6,58962 | 524,34% |
| 5xp1 | 7 | 140 | 985 | 23218 | 70 | 10 | 512 | 133 | 438 | 98 | 61,8654 | 418,40% |
| rd73 | 7 | 155 | 1135 | 27178 | 91 | 13 | 601 | 146 | 502 | 114 | 75,499 | 398,81% |
| Z5xp1 | 7 | 199 | 1430 | 38432 | 84 | 12 | 751 | 191 | 630 | 142 | 96,445 | 399,95% |
| misex1 | 8 | 72 | 504 | 7314 | 49 | 7 | 264 | 68 | 226 | 54 | 23,0975 | 511,97% |
| rd84 | 8 | 374 | 2756 | 138236 | 119 | 17 | 1459 | 354 | 1214 | 271 | 268,565 | 261,77% |
| ex5 | 8 | 1224 | 8648 | 1228506 | 105 | 15 | 4544 | 1188 | 3776 | 860 | 1761,63 | 141,35% |
| clip | 9 | 202 | 1514 | 42884 | 77 | 11 | 802 | 196 | 667 | 151 | 108,394 | 377,48% |
| 9sym | 9 | 239 | 1773 | 66596 | 98 | 14 | 936 | 231 | 776 | 170 | 140,665 | 342,63% |
| Z9sym | 9 | 266 | 1986 | 66928 | 112 | 16 | 1052 | 258 | 869 | 193 | 156,091 | 340,54% |
| apex4 | 9 | 3151 | 22589 | 8125002 | 133 | 19 | 11913 | 3045 | 9839 | 2208 | 11209,4 | 129,68% |
| sao2 | 10 | 168 | 1280 | 32520 | 98 | 14 | 684 | 157 | 567 | 128 | 84,4688 | 405,69% |
| alu4 | 14 | 1464 | 10734 | 1734366 | 154 | 22 | 5695 | 1415 | 4692 | 1068 | 2775,75 | 123,68% |
| misex3 | 14 | 1561 | 11429 | 1757416 | 161 | 23 | 6072 | 1510 | 4997 | 1150 | 2933,87 | 123,98% |
| table3 | 14 | 2152 | 15562 | 3153604 | 161 | 23 | 8294 | 2094 | 6777 | 1603 | 4743,15 | 142,25% |
| b12 | 15 | 89 | 649 | 12954 | 56 | 8 | 341 | 85 | 289 | 66 | 37,3648 | 436,55% |
| table5 | 17 | 1980 | 14467 | 2415332 | 175 | 25 | 7774 | 1941 | 6296 | 1544 | 3939,18 | 143,80% |
| duke2 | 22 | 673 | 5015 | 381946 | 147 | 21 | 2686 | 651 | 2199 | 521 | 692,15 | 205,72% |
| cordic | 23 | 111 | 865 | 16618 | 119 | 17 | 459 | 100 | 390 | 84 | 54,124 | 417,33% |
| misex2 | 25 | 121 | 921 | 19698 | 77 | 11 | 494 | 118 | 405 | 96 | 49,9148 | 479,33% |
| vg2 | 25 | 208 | 1564 | 43002 | 140 | 20 | 821 | 198 | 694 | 149 | 113,663 | 373,55% |
| apex2 | 39 | 445 | 3627 | 134188 | 203 | 29 | 1986 | 439 | 1597 | 395 | 354,094 | 287,99% |
| seq | 41 | 2413 | 17877 | 3580692 | 203 | 29 | 9599 | 2357 | 7790 | 1869 | 5833,47 | 155,45% |
| apex1 | 45 | 2626 | 19093 | 4823786 | 189 | 27 | 10151 | 2566 | 8315 | 1939 | 7179,4 | 146,71% |
| apex3 | 54 | 2240 | 16127 | 4110536 | 140 | 20 | 8510 | 2178 | 7023 | 1584 | 5738,37 | 130,22% |
| e64 | 65 | 1437 | 12134 | 416608 | 448 | 64 | 6754 | 1437 | 5349 | 1406 | 1535,65 | 262,27% |
| apex5 | 117 | 1293 | 9630 | 1239620 | 147 | 21 | 5346 | 1276 | 4197 | 1189 | 2412,64 | 120,04% |
| o64 | 130 | 130 | 1162 | 67872 | 56 | 8 | 645 | 130 | 517 | 130 | 159,803 | 180,74% |

TABLE IV: Experiments Multiple Crossbars

consumption from $P_{worst}$ downto $P_{best}$ including the error estimations ($E_{worst}$ and $E_{best}$). Moreover, in columns *DSE Time* and *SimTime Overhead* we report respectively the execution time of the proposed formal DSE and the overhead of a single simulation of the synthesized VHDL circuit compared to DSE. Simulations were carried out using ModelSim 10.3. As shown in the tables, the cost of performing a full simulation to determine the power consumption is in average very high: about 617% for the single crossbar and about 300% for the multiple crossbar. This clearly prove the benefits of using our approach instead of a full simulation. The difference between single and multiple crossbars can be explained by the fact that multiple crossbars are more complex to manage for our tool, so that the execution time is slightly higher than for the single crossbar.

It is worth emphasizing that, on a given circuit, the simulation is performed on a single workload while the formal DSE execution is actually *independent* of any specific workload. Thus, the formal DSE takes only one execution in order to perform the estimation of both best and worst cases of power consumption, along with an error estimation. The simulation approach, on the other hand, require a full simulation per each workload to find actual best and worst cases of power consumption.

## V. Conclusion

In this paper, we presented a formal DSE approach avoiding simulation campaigns. We proposed an algorithmic method to estimate both workload independent attributes and workload dependent ones. In particular, we estimate the power consumption of a given memristor-based crossbar architecture providing both a lower and an upper bounds and the related error estimation. As part of future work, we aim to compare memristor architectures against "classic" CMOS ones.

## References


[1] ITRS 2013 report. [Online]. Available: http://www.itrs.net/

[2] B. Hoefflinger, "Chips 2020: A Guide to the Future of Nanoelectronics", The Frontiers Collection, Springer Berlin Heidelberg, 2012, pp. 421–427

[3] J. McPherson, "Reliability trends with advanced CMOS scaling and the implications for design", in IEEE Custom Integrated Circuits Conference, 2007, pp. 405–412

[4] S. Borkar, "Design perspectives on 22Nm CMOS and beyond", in Proceedings of the 46th Annual Design Automation Conference, 2009, pp. 93-94

[5] G. Gielen, et al., "Emerging yield and reliability challenges in nanometer CMOS technologies", in Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1322-1327, 2008

[6] L. Chua, 'Memristor-the missing circuit element', IEEE Transactions on Circuit Theory, vol. 18, no. 5, pp. 507–519, 1971

[7] R. Waser et al., "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," Advanced Materials, vol. 21, pp. 2632–2663, 2009

[8] J. J. Yang et al., "Memristive devices for computing," Nature nanotechnology, vol. 8, pp. 13–24, 2013

[9] J. R. Burger et al., "Variation-tolerant computing with memristive reservoirs," IEEE/ACM International Symposium in Nanoscale Architectures (NANOARCH), 2013, pp. 1–6.

[10] K.-H. Kim et al., "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," Nano letters, vol. 12, pp. 389–395, 2011.

[11] S. Hamdioui, et al., 'Memristor based computation-in-memory architecture for data-intensive applications', in Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1718-1725, 2015

[12] Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels, "Fast Boolean Logic Mapped on Memristor Crossbar", IEEE International Conference on Computer Design (ICCD), pp. 335-342, 2015.

[13] M. Traiola, M. Barbareschi, A. Mazzeo and A. Bosio, "XbarGen: A memristor based boolean logic synthesis tool," 2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Tallinn, Estonia, 2016, pp. 1-6.

[14] G. Snider, "Computing with hysteretic resistor crossbars," Applied Physics A, vol. 80, pp. 1165–1172, 2005.

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

[16] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, Robert K. Brayton and Alberto L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis", EECS Department University of California, Berkeley Technical Report No. UCB/ERL M92/41 1992

[17] Saeyang Yang, "Logic Synthesis and Optimization Benchmarks User Guide". [Online]. Available: http://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth91.pdf