Estimating Dynamic Power Consumption for Memristor-based CiM Architecture

Marcello Traiola[a,1], Mario Barbareschi[b,2], Alberto Bosio[a,1]

[a] *LIRMM - University of Montpellier / CNRS - France - Email: {lastname}@lirmm.fr.*
[b] *DIETI - University of Naples Federico II - Italy - Email: mario.barbareschi@unina.it*

Abstract

Nowadays, Computing-in-Memory (CiM) represents one of the most relevant solutions to deal with CMOS technological issues and several works have been proposed so far targeting front and back-end synthesis. However, a given CiM architecture can be synthesized depending on different parameters, leading to different implementations w.r.t. area, power consumption and performance. It is thus mandatory to have an evaluation framework to characterize the actual implementation depending on the above terms. This is even more important during the Design Exploration phase, in which many different implementations are explored to identify the best candidate w.r.t. the user requirements. In this work, we focus on the dynamic power consumption estimation of a given CiM implementation. Instead of resorting to a simulation-based power estimation, we propose an analytical approach that will dramatically speed up the estimation since no simulations are required. By comparing the proposed approach against the simulation-based method over a massive experimental campaign, we show that the accuracy of the estimation turns out to be very high.

Keywords: Computing-in-Memory Architecture, Power Estimation, Memristor, Design Space Exploration, Boolean Functions, Circuit Synthesis

1. Introduction

Energy-efficient computation is one of the most challenging problem faced today by the research community. Two main walls have to be broken in order to go beyond state-of-the-art solutions: architectural and technology walls. The most advanced computing architectures are still based on the Von Neumann or Harvard paradigm, in which data has to be moved from the storage element (typically a Random Access Memory, RAM) to the computational element (namely the CPU) and then moved back. This is nowadays becoming no more affordable due to the fact that most of the time and energy required to complete any task is spent for data transfer only. The second wall is the CMOS technology that is reaching its physical limits and violating the famous Moore's law. New technology nodes will not double the performances as was in the past. Moreover, the leakage current increasing leads to higher static power consumption and lower reliability [1, 2, 3, 4].

*Corresponding author
Email address: traiola@lirmm.fr (Marcello Traiola)
[1] 161, rue Ada - 34095 Montpellier cedex 5, France
[2] Via Claudio, 21 - 80125 Naples, Italy

One of the emerging solutions to enable energy-efficient computation is the so-called Computation-in-Memory (CiM) paradigm: by merging together computation and storage, it is possible to avoid data transfers from and back to memory. This is possible thanks to the memristor [5], a non-volatile device able to act as both storage and information processing unit. Moreover, the memristor presents many advantages: CMOS process compatibility, lower cost, zero standby power, nanosecond switching speed, great scalability, high density and non-volatile capability [6, 7].

Thanks to its nature (i.e., computational as well as storage element), the memristor is exploited in different kinds of applications, such as neuromorphic systems [8], non-volatile memories [9] and computing architectures for data-intensive applications. This paper focuses on the latter ones, and more in particular on the CiM architecture proposed in [10]. For this architecture, some works have been proposed so far targeting both front and back-end synthesis [11, 12]. However, a given CiM architecture can be synthesized depending on many parameters [11] leading to different implementations w.r.t. area, power consumption and performances. In order to analyze and compare different solutions varying on these parameters, an evaluation framework needs to be properly defined. This is even more important during the Design Exploration phase during which many different implementations are explored to identify the best candidate(s) w.r.t. the user requirements. In this work, we focus on the dynamic power consumption estimation of a given CiM implementation, proposing a front-end evaluation framework. Instead of resorting to a simulation-based power estimation, we propose an analytical approach that will dramatically speed up the estimation time since no simulations are required. Comparing the proposed approach against the simulation-based method, experimental results show that the dynamic power consumption estimation turns out significantly accurate. The proposed methodology can be helpful to designers in the early stages of the circuit design (i.e., front-end) for obtaining a preliminary estimation of the dynamic power consumption. In successive phases, back-end analyses can further refine estimations by including technological information.

The remainder of the paper is structured as follows. Section 2 provides the required background about the memristor-based CiM and the state-of-the-art power estimation approaches. Section 3 presents the synthesis flow and the design space exploration framework, while the Section 4 gives the experimental results. Finally, the Section 5 draws the conclusions.

2. Background and state-of-the-art

In this section we provide basic concepts about the memristor-based CiM architecture as well as about power estimation approaches.

2.1. Memristor-based CiM

The memristor is a two-terminal non-linear passive electrical component characterized by a variable electrical resistance, which value depends on the history of the current flowed through the device itself. Since we exploit memristors to implement digital circuits, we refer to the memristor Voltage-Current relation depicted in Figure 1, detailed in [13], as the best solution for modeling the device behavior (i.e., taking into account the ideal response to a pulse-wave). Such model considers that the voltage applied to memristor terminals does not change the device

resistance until one of the two thresholds $\pm V_{th}$ is crossed. In the adopted ideal model, they are symmetrically defined.

We resort to the Snider Boolean Logic (SBL) [13] convention, whereby a lower resistance (steeper curve denoted as R_{ON}) represents a logic 0 while an higher resistance (lower slope curve denoted as R_{OFF}) represents a logic 1. Two basic operations can be performed, defined as SET and RESET. The former allows to program the memristor to R_{ON}, hence at logic 0, while the latter performs the memristor switching to R_{OFF}, that corresponds to logic 1.

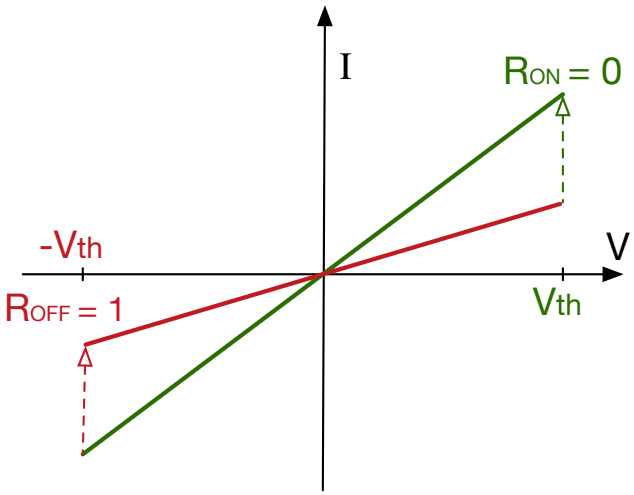


Figure 1: Memristor Ideal model.

2.1.1. Fast Boolean Logic Circuits

Snider proposed in [13] a design methodology to implement boolean functions on a memristor-based crossbar. The proposed approach was then improved by Xie et al. in [14]. 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 as shown in Equation 1.

$$AB + \overline{A}B + \overline{AB} = \overline{\overline{AB} \cdot \overline{\overline{A}B} \cdot \overline{\overline{AB}}} \tag{1}$$

The left member of the Equation 1 can be easily manipulated through transformation rules (i.e., De Morgan's laws). The obtained form (right member of the Equation 1) can be computed exploiting three boolean operations: NAND, AND and NOT.

Then, as Figure 2-a shows, FBLC is divided into blocks. The control logic is realized by a Finite State Machine (FSM) through several 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.

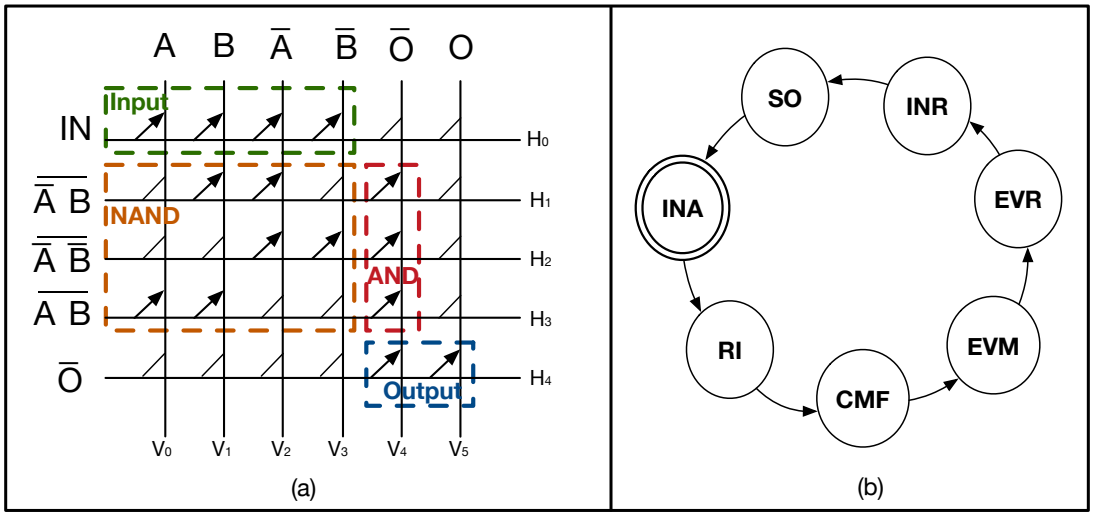


Figure 2: Fast Boolean Logic Circuit.

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 of [14] proposed some *primitive operations*. Each of these operations can be performed using as many input and output memristors as desired. By driving the crossbar's nano-wires with proper voltages during each step, it is possible to achieve boolean function calculations in a constant number of steps.

2.2. Power Estimation

Power estimation methodologies resort to power models, which are dependent on the target abstraction level. Clearly, the higher estimation precision one desires, the lower abstraction level one has to resort to, at the cost of an increasing estimation time. More in detail, power estimation is based on a macro-modeling approach, in which a power model is created using pre-characterized power values [15]. This power pre-characterization can be computed at different abstraction levels [16, 17]. The accuracy and the simulation run-time to obtain the characterization

point will depend on the available information of the circuit structure and activity at the considered level (i.e., gate or transistor level). Several works have been proposed at gate-level to assess power consumption per component, i.e. dynamic (switching, short-circuit) and static or leakage power [18, 19]. This kind of works models the gate current as a triangular shape taking into account different operational conditions (supply and ground voltages) and input switching conditions (rise, fall, static). Their results show good accuracy with transistor level simulations.

State-of-the-art methodologies need to be adapted for the memristor-based computing architecture. First of all, the static or leakage power does not need to be taken into account anymore, since the memristor is a 0-leakage device [6, 7]. Secondly, the fact that CiM architecture is implemented as a memristor-crossbar requires a new approach for computing the dynamic power consumption (i.e., power consumed during memristor switching). In [12], the authors proposed to compute the power consumption of a given CiM architecture as expressed by Equation 2. It corresponds to the sum of the power consumed by memristor-crossbar (P_{xbar}), by the CMOS voltage drivers ($P_{vd,all}$) and by the crossbar controller (P_{ctrl}) for all steps N_{step} to be executed.

$$\begin{cases} P_{CiM} & = \sum_{n=1}^{N_{step}} (P_{xbar} + P_{vd,all} + P_{ctrl})_n \\ P_{xbar} & = \sum_{\text{memristors}} P_R \\ P_R = \frac{V_R^2}{R} \end{cases} \tag{2}$$

For each execution step, P_{xbar} is the total power consumed by all the devices within the crossbar, which consists of the active and disabled memristors. $P_R = \frac{V^2}{R}$ is the power consumed by each memristor, where V_R is the voltage across the memristors and R is its resistance. $P_{vd,all}$ is the power consumed by all the voltage drivers and, we assume that the voltage drivers are almost ideal voltage sources and their power consumption is very small as compared with that consumed by the crossbar (which constitutes the load of the voltage drivers). The value of P_{ctrl} is provided by a commercial tool (i.e., Synopsys® PrimePower).

In order to properly calculate the value of P_{xbar}, we exploit the knowledge about of the state of each memristor in the crossbar (i.e., either R_{ON} or R_{OFF}) and of the actual V_R. As previously mentioned, depending on desired abstraction level specific information are available. At front-end level, information about chosen technology (i.e., operating voltages, switching time, resistance values) are not available yet. Therefore, at such level, only the dynamic power consumption characterization can be addressed: indeed, studying the switching activity of memristors allows such characterization. Thus, the problem can be reduced to the counting the switches for each memristor of the crossbar under a given workload. In the back-end phase, further information can be added for refining the estimations (e.g., sneak paths, delay), as performed in [12].

The main problem of the above approach is the computational time that can be very high. Moreover, as the approach is workload-dependent, another problem is the identification of a significant workload for power estimation purposes (i.e., a workload that avoid over- and under-stress the circuit). This paper aims at targeting the above two issues by proposing an analytical structural approach able to estimate a lower and upper bound of dynamic power consumption without resorting to any simulations. This leads to overcome the two issues above identified: (i) dramatically run-time reduction and (ii) workload independence. The main drawback of this approach is a slight

accuracy loss due to the estimation required by the methodology. Later on, we prove that the estimation stands under the 3%.

3. Analytical Dynamic Power Estimation

As stated in the previous section, power consumption estimation is a workload dependent task. Thus, it is necessary to simulate the circuit to perform the estimation, incurring in high computational time. We propose an analytical method to estimate the dynamic power consumption in order to overcome the above mentioned issues (i.e., high computational time and workload dependency).

We resort to the P_{xbar} term of Equation 2 rewriting it as follows:

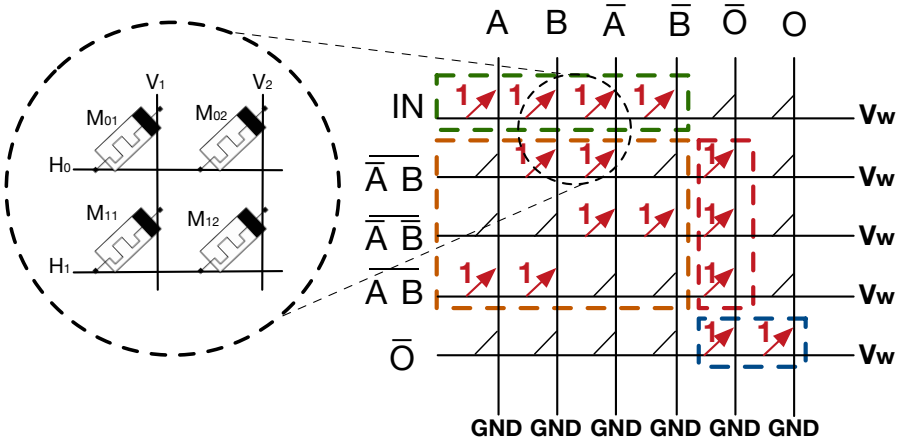
$$P_{xbar} = N_{up} \cdot C_{up} + N_{down} \cdot C_{down} \tag{3}$$

where:

- N_{up} and N_{down} represent the number of memristors in the crossbar that switch from '0' to '1' and from '1' to '0' respectively; they actually depend on the applied workload
- C_{up} and C_{down} represent the cost in terms of power consumption of a memristor switching from '0' to '1' and from '1' to '0' respectively.

For achieving the workload independence, we propose a structural estimation of a power interval that bounds the dynamic power consumption of the circuit, for any input vectors. For achieving the run-time reduction, such interval is actually computed without involving any simulation process.

Referring to the Equation (3), we want to characterize N_{up} and N_{down} relying on the structure and the control logic of the circuit. First, we can observe - as Figure 3 shows - that during the first stage of the computation (i.e., reset stage) all the memristors in the circuit are set to logic 1.



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

Figure 3: Reset stage.

Therefore, during this stage, we have only the contribution of $N_{up} \cdot C_{up}$ while, during the rest of the computation, only $N_{down} \cdot C_{down}$ contributes to the power consumption. Besides, considering a concatenation of executions (i.e., more successive input vectors), we can observe that, whether N_{down} memristors switch from '1' to '0' during the computation, the successive RESET phase has to switch the same number of memristors from '0' to '1'. Therefore, we can assume that the two contributions are equal:

$$N_{up} = N_{down} \tag{4}$$

We can also identify parts of the circuit that do not depend on actual input vectors (i.e., the workload). We resort to the Figure 2-a for explaining the basic idea.

- Concerning the Input box, half of the input memristors will switch during the execution of the circuit. This happens because we have 2 memristors for each literal x_i of the function: x_i and $\overline{x_i}$;
- The same consideration is true for the output memristors, in the Output box.

Therefore, such two parts of the circuit can be evaluated independently of the workload.
Thus, we are able to rewrite the equation 4 as follows:

$$N_{up} = N_{down} = N_{in} + N_{out} + N_{int} \tag{5}$$

where:

- N_{in} is the number of inputs (i.e., literals)
- N_{out} is the number of outputs
- N_{int} is the number of switching memristors that belong to the NAND and AND boxes:

$$N_{int} = (N_{m_{NAND}} + N_{m_{AND}}) \tag{6}$$

Finally, the memristors of these two parts switch accordingly to the actual input values. Thus, the goal is to estimate the power interval considering only NAND and AND boxes. Therefore, we will obtain two values of N_{int}: a best case value and a worst case value. The term $N_{m_{AND}}$ of Equation 6 depends on $N_{m_{NAND}}$. More in detail, the switching memristors in the AND box depend on those in the NAND box. Since we want to keep the computation simple and efficient, we will estimate only $N_{m_{NAND}}$ (i.e., the biggest part of the circuit) and, consequently, we will obtain $N_{m_{AND}}$ leveraging the dependency on $N_{m_{NAND}}$.

In order to estimate $N_{m_{NAND}}$, we count the number of memristors in the NAND box for each couple of vertical nano-wires that corresponds to the same input (i.e., x_i and $\overline{x_i}$). The input term - either x_i or $\overline{x_i}$ - that leads to have less memristors in the NAND box will be part of the best case. Otherwise it will be part of the worst case. Iterating this process for each input, we will obtain both worst and best case input vectors.

Finally, considering the above obtained vectors, we are able to compute how many memristors will switch in the AND box. Specifically, since we are performing a NAND operation, if a minterm has at least one literal among

those in the considered input vector, the corresponding memristor in the AND box will not switch, otherwise it will.

To summarize, we count the number of memristors in the NAND box to find the best case and worst case input vectors, then we verify if each minterm will be whether '0' or '1'. In order to do so, we do not need the truth table of the function. 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 $\theta(2^n)$, where n is the number of inputs.

The algorithm 1 realizes what we explained so far.

Algorithm 1: Get best and worst case.

Data: Literals, Minterms
Result: $(N_{m_{NAND}}, N_{m_{AND}})_{worst/best}$

```
for each crossbar j do
    for each input (x_i, ¬x_i) do
        if ¬x_i.NandOccurrence > x_i.NandOccurrence then
            vectorWorstCase ← ¬x_i;
            vectorBestCase ← x_i;
            (N_mNAND)worst += ¬x_i.NandOccurrence;
            (N_mNAND)best += x_i.NandOccurrence;
        else
            vectorWorstCase ← x_i;
            vectorBestCase ← ¬x_i;
            (N_mNAND)worst += x_i.NandOccurrence;
            (N_mNAND)best += ¬x_i.NandOccurrence;
        end
    end
    for each minterm m do
        if m does not contain any element of vectorWorstCase
         then
            (N_mAND)worst = (N_mAND)worst + 1;
        end
        if m does not contain any element of vectorBestCase then
            (N_mAND)best = (N_mAND)best + 1;
        end
    end
end
```

Let us resort again to the example of Figure 2-a to illustrate the above considerations. It is easy to see that, depending on the chosen input vector, the number of switching memristors changes. To show the different scenarios, we resort to Table 1 which reports:

- From 1^{st} to 4^{th} column, the input vectors;
- In the *"NAND box"* column, the number of switching memristor within the NAND box;
- In the *"AND box"* column, the number of switching memristor within the AND box;
- In the *"Tot"* column, the sum of the two previous columns.

A	B	$\overline{A}$	$\overline{B}$	NAND box	AND box	Tot
0	0	1	1	3	1	4
0	1	1	0	2	1	3
1	0	0	1	4	0	4
1	1	0	0	3	1	4

Table 1: Switching memristors.

Applying the proposed algorithm to the example, the input vector "1001" is chosen as the worst case one since it makes $N_{m_{NAND}}$ be equal to 4. Consequently, $N_{m_{AND}}$ is calculated as 0, considering the selected input vector.

Conversely, the input vector "0110" is chosen as the best case one since it makes $N_{m_{NAND}}$ be equal to 2. Accordingly, $N_{m_{AND}}$ is calculated as 1, considering the selected input vector.

Finally, the two selected vectors lead to obtain the lower and the upper bounds of the power interval (i.e., [3,4])

3.1. Extended Power Interval

For some particular configurations of the FBLC, it can happen that the obtained bounds are not the actual ones. Resorting to the example in Figure 4 and the relative Table 2, such scenario become clear.

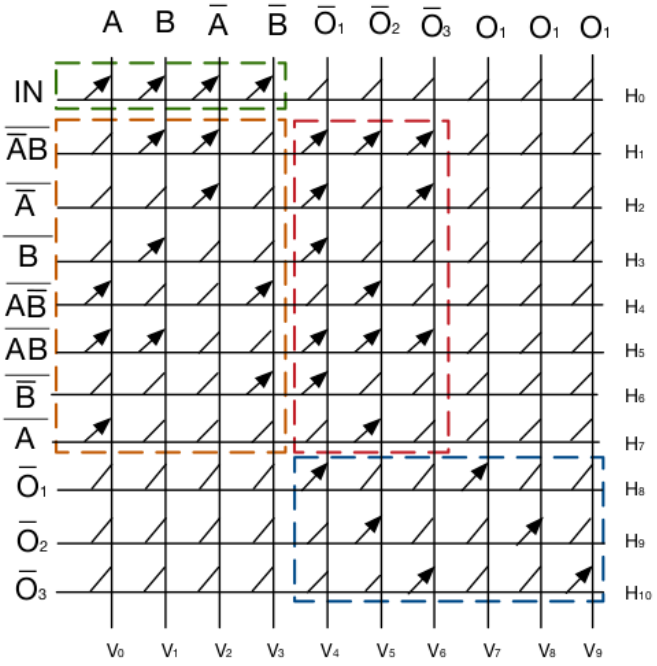


Figure 4: FBLC estimating error.

A	B	$\overline{A}$	$\overline{B}$	NAND box	AND box	Tot
0	0	1	1	6	3	9
0	1	1	0	5	6	11
1	0	0	1	5	3	8
1	1	0	0	4	5	9

Table 2: Switching memristors.

Applying the proposed Analytical Dynamic Power Estimation, the worst case is estimated as the input vector "0011" which makes switch 6 NAND memristors. On the other side, the best case is estimated as the input vector "1100" which makes switch only 4 NAND memristors. In both cases the estimation does not give the expected results. Indeed, the worst case is actually the input vector "0110". It makes switch fewer memristors in the NAND box - compared to the estimated worst case - but more in the AND box. On the other side, the best case is actually the input vector "1001" which makes switch more memristors in the NAND box than the estimated best case but fewer in the AND box. Indeed, resorting to Table 2 it can be remarked that the estimated power interval (i.e., [9,9]) is not the correct one. In fact, the actual best and worst cases provide the power interval [8,11].

Therefore it is worth to characterize an *Extended Power Interval* that will cover these scenarios. In order to be very conservative we chose the two extended boundaries as follows.

$$P_{ext_{worst}} = P_{worst} + E_{worst} \tag{7}$$

$$P_{ext_{best}} = P_{best} - E_{best} \tag{8}$$

where:

- E_{worst} is the difference between the total number of memristors in the AND box and the estimated number of switched ones.
- E_{best} is just the estimated number of switched memristors in the AND box.

Equations 9 and 10 formalize those terms.

$$E_{worst} = N_{m_{AND_{tot}}} - N_{m_{AND}} \tag{9}$$

$$E_{best} = N_{m_{AND}} \tag{10}$$

As above stated, we are very conservative estimating such Extended Interval. Indeed, referring to the above example (Figure 4):

$$E_{worst} = 12 - 3 = 9$$

$$E_{best} = 5$$

In this way, the estimated Extended Interval is [4,18] which contains the actual one.

Despite the Extended Interval is such a really conservative estimation, it turns to be necessary really rarely. Indeed, in presence of more crossbars, we estimate for each one the vectors that lead to the local best and worst cases. Nevertheless, since each crossbar inputs depend on the outputs of the previous one, the probability of producing output values exactly equal to the ones we estimated is really low.

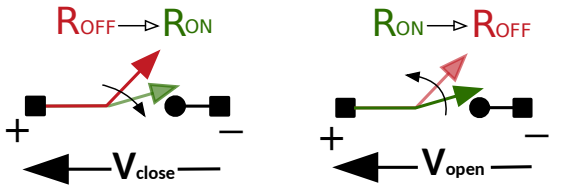
Finally, the above introduced Extended Interval has been introduced in order to cover all the possible cases.

In conclusion, we estimate the dynamic power consumption interval as follows:

$$P_{interval} = [(C_{up} + C_{down}) \cdot (N_{in} + N_{out} + N_{int_{best}}), (C_{up} + C_{down}) \cdot (N_{in} + N_{out} + N_{int_{worst}})] \quad (11)$$

4. Experimental Results

This section provides experimental results. We carried out experiments with the proposed method and we compared it against a behavioral simulation-based one. In order to perform simulations, we realized a behavioral VHDL model of the memristor. Such model is able to keep track of the number of the commutations of memristors. In figure 5 we depict the basic concept of the model.



(a) Closing operation (b) Opening operation

Figure 5: Memristor Behavioral Model

Basically, the memristor model acts like an ideal switch. It can be opened and closed by driving its terminals with different levels of voltage. By means of a counter, we kept track of both the number of commutation $R_{OFF} \rightarrow R_{ON}$ and the number of commutation $R_{ON} \rightarrow R_{OFF}$ of each memristor in the circuit. We were able, then, to exploit such model along with the VHDL code produced by XbarGen [11] for carrying out simulations.

Next subsections explain how we set up the tool-chain for carrying out experiments and discuss the results.

4.1. Experimental Flow

For 19 different combinational circuits, we applied the following flow. We exploited two different kinds of automatic tools: two logic synthesis tools (SIS [20] and ABC [21] by Berkeley) and an architecture-aware High Level Synthesis tool (XbarGen [11], developed by us). Given a combinational circuit (i.e., boolean function), we obtained different boolean expressions by exploiting the former tools. Then, levering XbarGen, we performed the mapping of the obtained boolean expressions onto the target architecture (i.e., FBLC). Specifically, in Table 3 we report the different syntheses we performed. For each configuration (1st column) we report the specific command we used for logic synthesis and we describe the target architecture we obtained by using XbarGen mapping (2nd column). Within the table, the configurations are sorted by the number of crossbars that they generated. The *Multiple Crossbar* configuration generated a lot of crossbars with a couple of inputs each, *3-LUT* generated fewer crossbars with 3 inputs each and so on. Finally, the *Single Crossbar* configuration generated only one big crossbar. In the end, we obtained 133 circuits split into 7 groups. By setting up such experimental flow, we were able to measure the precision of the proposed estimation approach for different configurations of memristor-based crossbars (i.e., Single Crossbar, Multiple Crossbar, 3LUTS up to 7LUTS mappings).

Table 3: Performed Syntheses

Configuration	The circuit is synthesized to obtain
Multiple Crossbars	More crossbars without any optimization ("strash" command of ABC [21])
3-LUT	More crossbars with 3 inputs each ("if -K 3" command of ABC [21])
4-LUT	More crossbars with 4 inputs each ("if -K 4" command of ABC [21])
5-LUT	More crossbars with 5 inputs each ("if -K 5" command of ABC [21])
6-LUT	More crossbars with 6 inputs each ("if -K 6" command of ABC [21])
7-LUT	More crossbars with 7 inputs each ("if -K 7" command of ABC [21])
Single Crossbar	A single crossbar ("collapse" command of SIS [20])

By exploiting XbarGen [11], we were able to analyze the Boolean expressions to obtain both the dynamic power estimation of the memristor-based crossbar circuits and their VHDL implementations. Furthermore, as an output of XbarGen, we got the two best/worst case input vectors which lead the circuit consuming the estimated power boundary values (i.e, P_{worst} and P_{best}). In order to validate the power estimation analytical method and compare it with the simulation-based approach, we simulated each VHDL circuit with up to 4096 random input vectors (for the big circuits) and with those generated by XbarGen. Next, leveraging the memristor model described in Section 4, we collected the number of switching memristor within the crossbar(s). Finally, we calculated the average dynamic power for each circuit, along with the power interval (i.e, the minimum and the maximum values). Equations 12 and 13 describe how we calculated the average dynamic power $P_{sim_{avg}}$ and the power interval $[P_{sim_{best}}, P_{sim_{worst}}]$ respectively:

$$P_{sim_{avg}} = \frac{\sum_{i=1}^{N} P_i}{N} \tag{12}$$

$$P_{sim_{best}} = \min_i(P_i) \quad i \in [1, N] \qquad P_{sim_{worst}} = \max_i(P_i) \quad i \in [1, N] \tag{13}$$

where P_i is the dynamic power consumption for the i-th input vector and N the number of observations (i.e., the number of input vectors).

Furthermore, we wanted to measure how much the obtained sample mean (i.e.,$P_{sim_{avg}}$) was a good estimator of the population mean (i.e., the real Average dynamic Power). To do so, we resorted to the Relative Standard Error (RSE). The RSE of a sample mean is the Standard Error (SE) of the mean divided by the mean and expressed as a percentage. The SE is a sort of standard deviation of the error in the sample mean w.r.t. the real mean. Equations 14 and 15 describe the expressions of SE and RSE respectively.

$$SE_{\bar{x}} = \frac{s}{\sqrt{N}} \tag{14}$$

$$RSE = \frac{SE_{\bar{x}}}{\bar{x}} \tag{15}$$

where $\bar{x}$ is the sample mean (i.e.,$P_{sim_{avg}}$), s the *sample standard deviation* and N the number of observations of the sample (i.e., the number ofs input vectors).

So, we calculated the Relative Standard Error (RSE) of the mean, for each circuit. Then, for each configuration, we

calculated the average, minimum and maximum RSE. It is worth to remember that each configuration contains 19 circuits. In table 4 we report the above mentioned values (2nd-4th columns), for each configuration (1st column).

Table 4: Relative Standard Error (RSE)

Configuration	Avg. RSE	Min. RSE	Max. RSE
Multiple Crossbars	0.17%	0.0001%	0.43%
3-LUT	0.25%	0.0002%	0.68%
4-LUT	0.35%	0.0004%	0.87%
5-LUT	0.60%	0.0005%	2.10%
6-LUT	0.66%	0.0005%	2.05%
7-LUT	0.70%	0.0017%	1.24%
Single Crossbar	1.18%	0.000003%	3.46%

For each configuration, the average RSE was not greater than 1.20%. Even in the worst cases, the RSE was not greater than 3.46%. Thus, we can state that, for each circuit, the simulation-based estimation of the mean of the dynamic power consumption was very close to the real one (i.e., the mean that we would be able to calculate if we performed an exhaustive simulation).

It is worth to emphasize that, despite we applied the input vectors successively for each experiment, we considered each one as independent. This is due to the reset phase of the FBLC which take back all the memristors to value 1.

Finally, details about the characteristics of the synthesized circuits are available in [22].

4.2. Experimental results and discussion

In this subsection, we detail experimental results to demonstrate the effectiveness of our approach. More in detail, we highlight and discuss the accuracy of the proposed *Analytical Dynamic Power Estimation* comparing it with Dynamic Power Estimation obtained by means of simulations.

As previously discussed, the power consumption is a workload dependent attribute. In order to perform a quick estimation of such attribute, XbarGen [11] is able to estimate an interval that contains the actual dynamic power consumption of the memristor-based crossbar(s). This is made by means of a structural analysis.

Taking into account previous considerations, first of all we wanted to investigate the accuracy of the interval estimation. Table 5 shows, for each configuration (1st column), the percentage of correct predictions (i.e., the simulated interval corresponds) (2nd column), the average (3rd column), minimum (4th column), maximum (5th column) errors of prediction. It is worth to remind that, for each circuit, we performed the estimation of two bounds. Therefore, errors were estimated against two values for each circuit: the upper bound (i.e., worst case) and the lower bound (i.e., best case). Furthermore, since XbarGen enables us to find the worst/best-case input vectors, estimation errors concern only cases where actual values are lower than the estimated lower bound or greater than the estimated upper bound.

As the table shows, the percentage of estimation error is very low. Moreover, when an error occurs, often it concerns only one of the two bounds (i.e., the upper or the lower).

It is worth noting that, as we moved from Multiple towards Single Crossbar, the predictions quality dropped. This can be explained by the phenomenon that we discussed in Subsection 3.1. It turns out that such phenomenon mostly impacts big crossbars and, hence, the estimation. With relatively small crossbars, we do not observe it.

Table 5: Interval Estimation

Configuration	in Range (%)	Avg. error (%)	Max. error (%)
Multiple Crossbars	100.00%	0.00%	0.00%
3-LUT	100.00%	0.00%	0.00%
4-LUT	100.00%	0.00%	0.00%
5-LUT	89.47%	3.11%	11.11%
6-LUT	84.21%	2.91%	8.70%
7-LUT	78.95%	2.05%	4.76%
Single Crossbar	57.89%	1.92%	8.70%
Total	**87.22%**	**1.43%**	**4.75%**

Therefore, a designer could resort to such analytic estimations taking into account some band-guards. In Subsection 3.1 we discussed an extended interval. The considered interval is $[(P_{best} - E_{best}), (P_{worst} + E_{worst})]$. P_{best} and P_{worst} are the analytically estimated lower bound and upper bound for the dynamic power consumption, respectively. E_{best} and E_{worst} are the estimated extensions w.r.t. P_{best} and w.r.t. P_{worst}, respectively. These values, estimated by XbarGen [11], are really conservative. Indeed, it is actually impossible to generate vectors which lead the circuit to consume such values of dynamic power. Nevertheless, those values delimit the ranges for the guard-bands that a designer can choose. Indeed, such ranges are $[(P_{best} - E_{best}), P_{best}]$ and $[P_{worst}, (P_{worst} + E_{worst})]$

Now, let us discuss about the average dynamic power consumption estimation ($P_{est_{avg}}$). We estimated it as the midpoint of the analytically evaluated interval, as reported in the following Equation 16:

$$P_{est_{avg}} = \frac{P_{best} + P_{worst}}{2} \tag{16}$$

Table 6: Average Dynamic Power Estimation Accuracy

Configuration	Avg. Est. error %	Est. error range %	
Multiple Crossbars	-3.72%	-6.34%	0.51%
3-LUT	-3.34%	-8.13%	0.90%
4-LUT	-2.80%	-8.48%	1.72%
5-LUT	-3.04%	-10.52%	1.52%
6-LUT	-2.54%	-9.80%	1.26%
7-LUT	-2.37%	-8.33%	3.06%
Single Crossbar	-0.91%	-7.26%	1.02%

Let us discuss the results by means of the *estimation error*. We evaluated the estimation error as follows:

$$Err_{est} = \frac{P_{sim_{avg}} - P_{est_{avg}}}{P_{sim_{avg}}} \cdot 100 \tag{17}$$

where $P_{sim_{avg}}$ and $P_{est_{avg}}$ are described in Equations 12 and 16 respectively. A positive value states that we underestimate the average power. Conversely, a negative value states that we overestimate the average power. Table 6 shows the results. Given a configuration (1st column), column 2 shows the average error of estimation (as a percentage) for all the circuits synthesized with that configuration. Column 3 reports the related error range. Results clearly show that the Analytical Dynamic Power Estimation that we propose is accurate.

Finally, we consider another significant parameter, that is the execution time of our tool. In tables 7 to 9 we report the time required to get the output. Since the execution time depends on the actual load of the machine on which the experiments run, the average time can be heavily affected by outliers. For this reason, for each configuration in the 1st column, Table 7 highlights the median values of the execution time of our approach (2nd column) and the execution time of the simulation-based approach (3rd column). In the 4th column, it shows the percentage of speedup of our method compared to the simulation-based one.

Moreover, in tables 8 and 9 we report the same comparison for minimum and maximum execution time values respectively.

We calculated the Speedup percentage as reported in Equation 18

$$SpeedUp = \frac{SimulationTime}{AnalyticTime} \cdot 100 \tag{18}$$

Table 7: Median Execution Time Comparison

Configuration	**Median Analytic Time (ms)**	**Median Simulation Time (sec)**	**Speedup %**
Multiple Crossbars	96.45	9	9331.74%
3-LUT	50.47	6	11887.24%
4-LUT	37.44	6	16026.50%
5-LUT	27.71	5	18044.68%
6-LUT	27.96	5	17880.96%
7-LUT	29.96	5	16689.70%
Single Crossbar	23.02	5	21716.00%

Table 8: Minimum Execution Time Comparison

Configuration	**Minimum Analytic Time (ms)**	**Minimum Simulation Time (sec)**	**Speedup %**
Multiple Crossbars	6.59	4	60701.53%
3-LUT	4.33	3	69251.92%
4-LUT	3.12	2	64014.95%
5-LUT	2.51	2	79643.83%
6-LUT	1.95	4	204873.95%
7-LUT	1.90	2	105502.48%
Single Crossbar	1.93	4	206841.28%

As tables show, the speedup gain is remarkable. Indeed, Table 7 shows a speedup median value of 16x on average, Table 8 shows a speedup average value of 113x for the simplest circuits and, finally, Table 9 shows a speedup average value of 10x for the biggest ones.

5. Conclusion

In this work, we proposed an analytical approach to the Dynamic Power Estimation for a given Computing-in-Memory implementation. The goals were to achieve workload independence and speed up the estimation time

Table 9: Maximum Execution Time Comparison

Configuration	Maximum Analytic Time (ms)	Maximum Simulation Time (sec)	Speedup %
Multiple Crossbars	1535.65	414	26959.27%
3-LUT	665.57	87	13071.50%
4-LUT	454.04	48	10571.73%
5-LUT	355.64	24	6748.45%
6-LUT	319.22	18	5638.83%
7-LUT	1241.85	16	1288.40%
Single Crossbar	1797.42	73	4061.38%

compared to a simulation-based approach. Experiments were carried out to compare the proposed method against the simulation-based one. Results highlighted a remarkable speedup (16x on average) at the cost of a slight accuracy reduction (3% on average).

References

[1] B. Hoefflinger, Chips 2020: A Guide to the Future of Nanoelectronics, Frontiers Collection, Springer Customer Service Center Gmbh, 2016.
URL https://books.google.fr/books?id=SbCzDAEACAAJ

[2] J. W. McPherson, Reliability trends with advanced cmos scaling and the implications for design, in: 2007 IEEE Custom Integrated Circuits Conference, 2007, pp. 405–412. doi:10.1109/CICC.2007.4405763.

[3] S. Borkar, Design perspectives on 22nm cmos and beyond, in: 2009 46th ACM/IEEE Design Automation Conference, 2009, pp. 93–94. doi:10.1145/1629911.1629940.

[4] G. Gielen, P. D. Wit, E. Maricau, J. Loeckx, J. Martin-Martinez, B. Kaczer, G. Groeseneken, R. Rodriguez, M. Nafria, Emerging yield and reliability challenges in nanometer cmos technologies, in: 2008 Design, Automation and Test in Europe, 2008, pp. 1322–1327. doi:10.1109/DATE.2008.4484862.

[5] L. Chua, Memristor-the missing circuit element, IEEE Transactions on Circuit Theory 18 (5) (1971) 507–519. doi:10.1109/TCT.1971.1083337.

[6] R. Waser, R. Dittmann, G. Staikov, K. Szot, Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges, Advanced Materials 21 (25-26) (2009) 2632–2663.

[7] J. J. Yang, D. B. Strukov, D. R. Stewart, Memristive devices for computing, Nature nanotechnology 8 (1) (2013) 13–24.

[8] J. Bürger, C. Teuscher, Variation-tolerant computing with memristive reservoirs, in: Proceedings of the 2013 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH '13, IEEE Press, Piscataway, NJ, USA, 2013, pp. 1–6.
URL http://dl.acm.org/citation.cfm?id=2769681.2769683

[9] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, W. Lu, A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications, Nano Letters 12 (1) (2012) 389–395. doi:10.1021/nl203687n.

[10] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, J. van Lunteren, Memristor based computation-in-memory architecture for data-intensive applications, in: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE '15, EDA Consortium, San Jose, CA, USA, 2015, pp. 1718–1725.
URL http://dl.acm.org/citation.cfm?id=2757012.2757210

[11] M. Traiola, M. Barbareschi, A. Mazzeo, A. Bosio, Xbargen: A memristor based boolean logic synthesis tool, in: 2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2016, pp. 1–6. doi:10.1109/VLSI-SoC.2016.7753567.

[12] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, K. Bertels, A mapping methodology of boolean logic circuits on memristor crossbar, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems PP (99) (2017) 1–1. doi:10.1109/TCAD.2017.2695880.

[13] G. Snider, Computing with hysteretic resistor crossbars, Applied Physics A 80 (6) (2005) 1165–1172. doi:10.1007/s00339-004-3149-1.

[14] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, K. Bertels, Fast boolean logic mapped on memristor crossbar, in: 2015 33rd IEEE International Conference on Computer Design (ICCD), 2015, pp. 335–342. doi:10.1109/ICCD.2015.7357122.

[15] N. Bansal, K. Lahiri, A. Raghunathan, Automatic power modeling of infrastructure ip for system-on-chip power analysis, in: 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07), 2007, pp. 513–520. doi:10.1109/VLSID.2007.46.

[16] X. Liu, M. C. Papaefthymiou, Hype: hybrid power estimation for ip-based systems-on-chip, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24 (7) (2005) 1089–1103. doi:10.1109/TCAD.2005.850891.

[17] F. Klein, R. Leao, G. Araujo, L. Santos, R. Azevedo, A multi-model engine for high-level power estimation accuracy optimization, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 17 (5) (2009) 660–673. doi:10.1109/TVLSI.2009.2013627.

[18] C. Knoth, H. Jedda, U. Schlichtmann, Current source modeling for power and timing analysis at different supply voltages, in: 2012 Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 923–928. doi:10.1109/DATE.2012.6176629.

[19] S. Gupta, S. S. Sapatnekar, Compact current source models for timing analysis under temperature and body bias variations, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 20 (11) (2012) 2104–2117. doi:10.1109/TVLSI.2011.2169686.

[20] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, A. Sangiovanni-vincentelli, Sis: A system for sequential circuit synthesis, Tech. rep. (1992).

[21] Abc user guide - [online].
URL `http://www.eecs.berkeley.edu/ alanmi/abc/`

[22] B. Lisanke, Logic synthesis and optimization benchmarks, Microelectronics Center of North Carolina, Tech. Rep.
URL `http://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth91.pdf`