Multidisciplinary Workshop on Innovative Systems

Report part 2

Group:
Baldo Martino 250688
Riccardo Massa 251880
Maurizio Spada 249329

2018/2019

Contents

STEP 5

Introduction

In this step a model of the LiM array (namely CLIMA) has to be provided. This model will be used to compute the following parameters:

- delay;
- area;
- power consumption.

In order to compute these results, a Matlab model for the basic LiM cell and for the entire CLIMA array has been realized. Unfortunately when a specific algorithm, like the AES one, has to be executed, there are not a lot of parameters which can be chosen. In this specific case the only input signals that our hardware implementation can receive are the 128 bits input Key and the 128 bits input plain text. For each given plain text, the AES algorithm which has been implemented, encrypts it with the input key. It is clear that since only one encryption per time can be executed there is not any parameter that the user can choose. If one wants to encrypt more data he has to repeat again the algorithm.
To be noted that also the size of the CLIMA is fixed because the algorithm needs a certain number of LiM cells to be executed: if there are more of them they will not be used, unless a parallel computation is performed (but also in this case the algorithm should be repeated n times, so at the end it is sufficient to multiply the parameters given by the one-execution case for n).
For this reason the implemented model will compute the delay, the area and the power consumption independently on external parameters.
For what concerns the basic cell of the CLIMA, its structure is represented in figure 1.1.

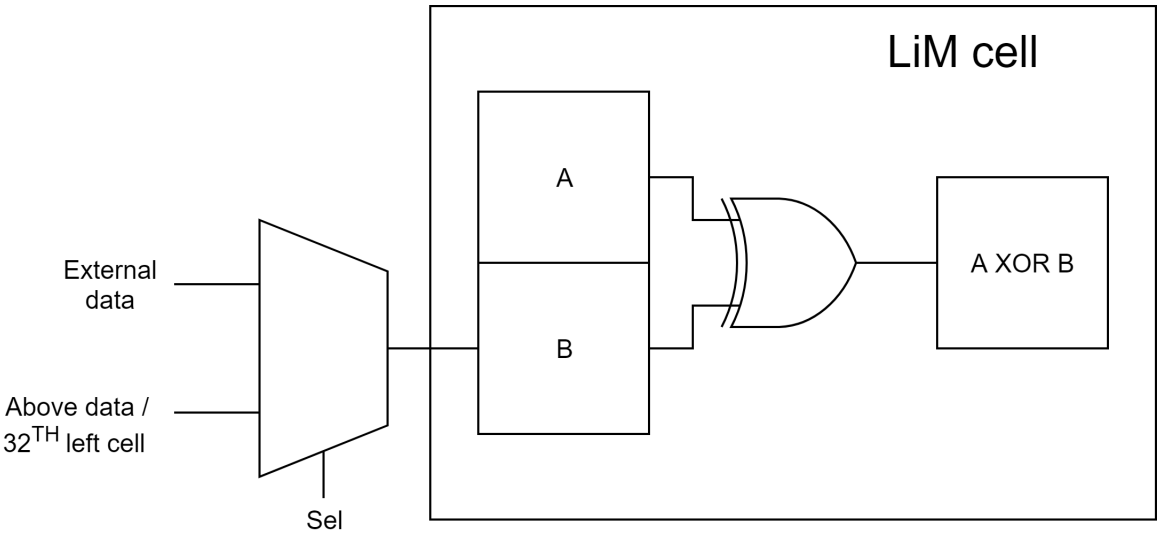

Figure 1.1: CLIMA basic cell

In order to compute the three required parameters of the CLIMA basic cell, a hierarchical model is used and it is based on:

- A first script which computes the area, delay and power consumption of the multiplexer at the input.

- A second script, provided by the technological group, which gives the area, the delay and the power consumption of the LiM cell.

Multiplexer

To model all the parameters related to a multiplexer, a NAND2 based multiplexer has been used (figure 1.2).

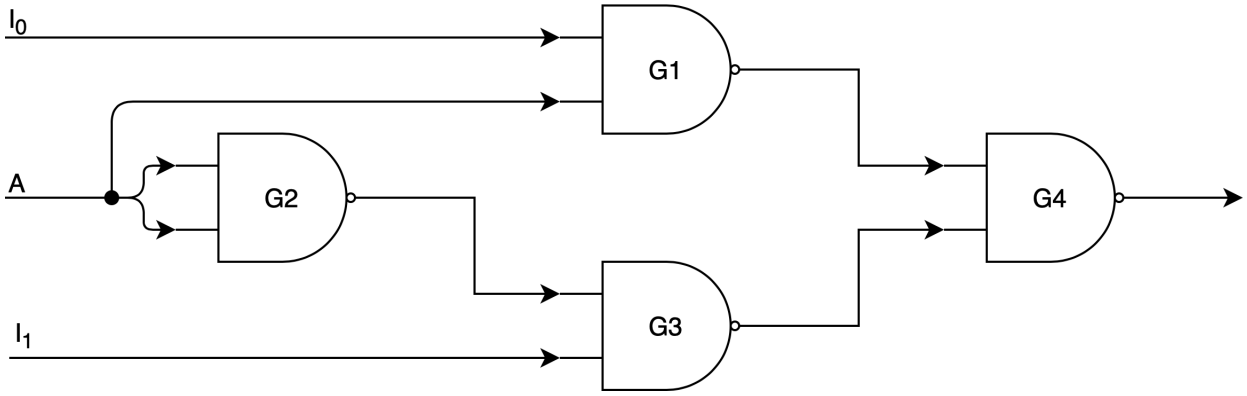

Figure 1.2: NAND2 based multiplexer

In this way it is sufficient to compute all the parameters needed for a NAND2 and then combine properly them in order to get power, delay and area of each multiplexer.
A Matlab script for the NAND2 was provided and it computes the power (both dynamic and static), the delay and the area.
The idea is that by setting properly the inputs probabilities of each NAND2 gate, it should be sufficient to call the NAND2 script multiple times and combine together all the parameters that it provides at each call.
For the input A, I_0 and I_1, it is reasonable to assume 50% probability that the input is in the logic state 1:

$$P_A = 0.5$$
$$P_{I_0} = 0.5$$
$$P_{I_1} = 0.5$$

It is clear that when the input of a NAND2 gate is the output of a previous gate, it is mandatory to compute the probability of the output and to use this probability as the input one. For each gate (except the last one, namely G4) the output probability has been computed as follows:

$$P_{G_1} = 1 - P_{I_0} \cdot P_A$$
$$P_{G_2} = 1 - P_A$$
$$P_{G_3} = 1 - P_{I_1} \cdot P_{G_2}$$

It is straightforward to see that also the output probability can be computed in a similar way:

$$P_{G_4} = 1 - P_{G_1} \cdot P_{G_3}$$

Now since each NAND2 gate takes into account the probability of each node and computes in the proper way their switching activity, it is reasonable to assume that the total dynamic power of a multiplexer is the sum of each NAND2 gate dynamic power:

$$P_{dyn}|_{TOT} = P_{dyn}|_{G_1} + P_{dyn}|_{G_2} + P_{dyn}|_{G_3} + P_{dyn}|_{G_4} \tag{1.1}$$

Also the total static power has been obtained by adding all the NAND2 contributions, because each one of them takes into account the inputs probabilities:

$$P_{stat}|_{TOT} = P_{stat}|_{G_1} + P_{stat}|_{G_2} + P_{stat}|_{G_3} + P_{stat}|_{G_4} \tag{1.2}$$

To compute the total delay it is mandatory to analyse the critical path of the multiplexer in order to know how many NAND2 contributions have to be added. By looking at the figure 1.2 it is straightforward to see that the critical path is composed by the following gates:

$$G2 + G3 + G4$$

It is sufficient to consider these three contributions to compute the total delay. Now since the delay of a gate does not depend on the input values and the wires delay is not considered in this analysis, the total delay can be actually computed as follows:

$$\tau_{PD} = 3 \cdot \tau_{PD}|_{NAND2\_GATE} \tag{1.3}$$

The total area is clearly the sum of each NAND2 gate area, so it can be computed as:

$$Area_{TOT} = 4 \cdot Area_{NAND2} \tag{1.4}$$

It is important to highlight that the NAND2 script which was provided, had several errors in terms of measurement units, and some capacitance contributions were not computed in the right way. So if one had used it, non-reasonable results would have been obtained.
This script has been modified and corrected and the following results have been obtained:

Quantity	Value	Measurement unit
P_{dyn}	26.75	nW
P_{static}	36.33	nW
P_{tot}	63.08	nW
$Delay$	42.45	ps
$Area$	0.05	μm^2

To be noted that these results are based on the LOP-2005 technology, with gate length equals to $L_{gate} = 45.1\,\text{nm}$ and a clock frequency of $F_{CLK} = 100\,\text{MHz}$.

LiM cell interface

As discussed in the previous report ([1]), each basic cell is able to do operations which are not required in our algorithm implementation (A and B read operation and a separate command to initialize the XOR operation). To decrease the number of signals which is necessary to deal with and to avoid that a future implementation of the LiM cell (more signals or a different combination of them to perform a certain operation) forces to modify all the VHDL files up to the control unit with an high probability of errors (which are very difficult to find for this type of memory), it has been created an interface for the LiM cell 1.3.

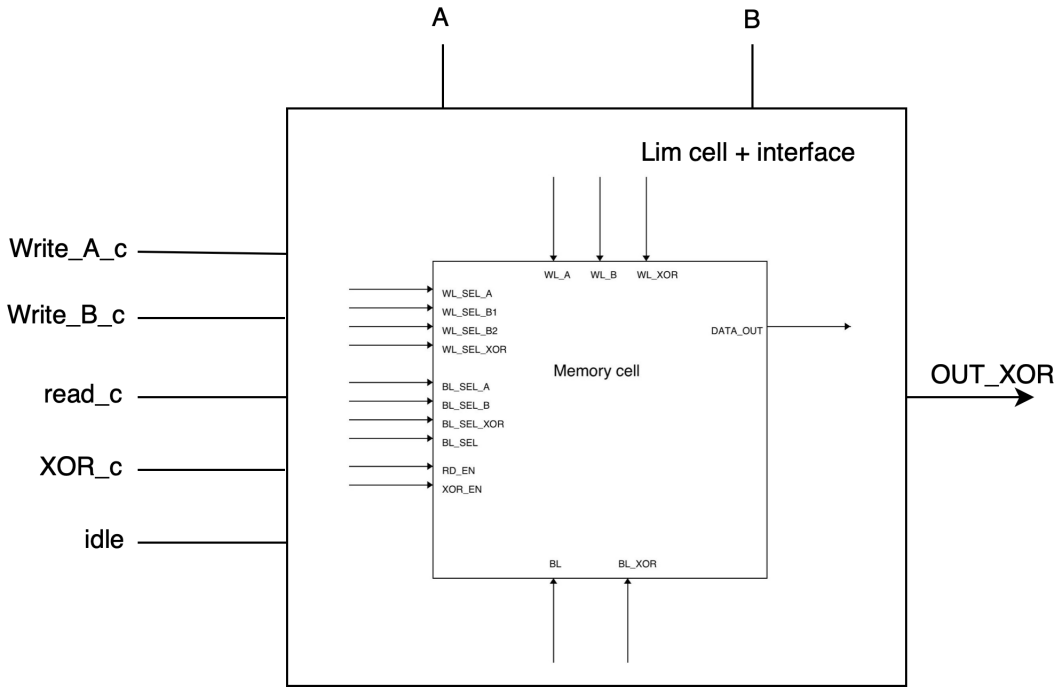

Figure 1.3: LiM cell + interface

In order to model the interface on Matlab, it is important to retrieve and analyze a possible implementation of it. By using a software called Quartus and its built-in RTL viewer tool, the implementation in the figure 1.4 has been obtained.
As it is possible to see from this implementation, there are 48 2-to-1 1-bit multiplexers. Since it is very difficult to know the worst case for the dynamic power, it has been assumed that all 48 multiplexers work simultaneously and so the following formulas have been used to compute the required parameters:

$$P_{dyn} = 48 \cdot P_{dyn}|_{mux2to1}$$

$$P_{stat} = 48 \cdot P_{stat}|_{mux2to1}$$

$$A = 48 \cdot Area|_{mux2to1}$$

The interface delay has not been computed since there is not an easy way to understand the critical path and even if a tool like Synopsys were used to synthesize the interface and compute the critical path, it would be impossible to ensure that the critical path of the interface has to be taken into account to compute the critical path of the whole architecture (this because it is not feasible to synthetize the CLIMA array with Synopsys).
However in the delay of the whole architecture an overhead has been taken into account (section 2.11), in order not to underestimate the critical path and to take into account also the setup time, the clock to output delay and the skew.
Once the model has been written, its results have been compared with the Synopsys ones. From this comparison it turned out that the static power computed by the Matlab model was too high and too pessimistic than the Synopsys one (approximately there was 1 order of magnitude of difference between the two). For this reason another Matlab script, which contains the Synopsys values for power (both static and dynamic) and area, has been written.
The choice of writing two different scripts for the interface has been made mainly because each cell of the CLIMA array has an interface: if its static power were overestimated so much, the CLIMA overall static power would be certainly far from the simulated result.

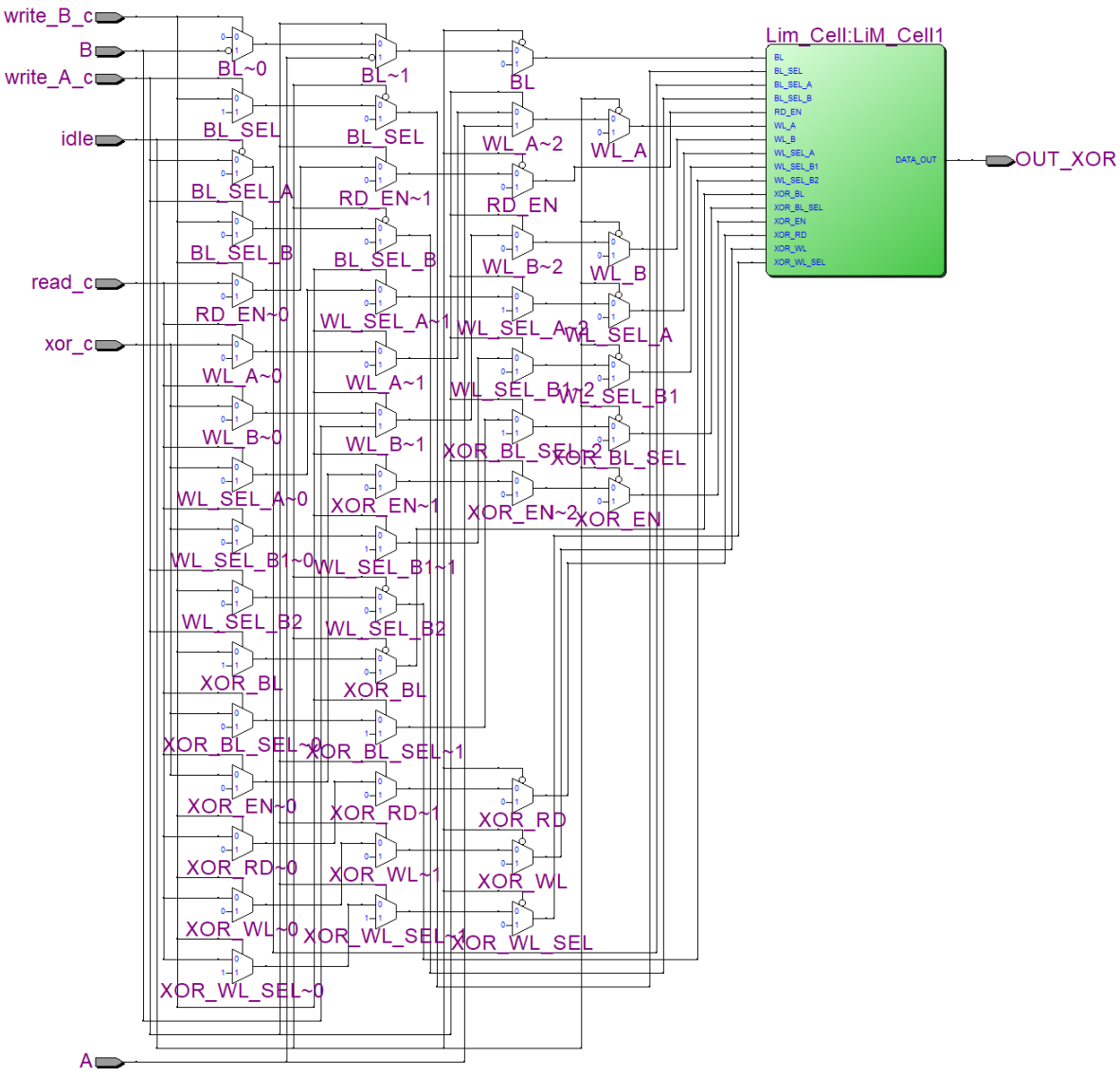

Figure 1.4: Quartus interface implementation

Moreover also the whole CLIMA array has two models: one takes into account the interface modelled by 48 multiplexers, while the other uses the Synopsys values for the interface.

CLIMA analysis

The CLIMA structure is shown in the figure 1.5. As one can see, there are three arrays of 128

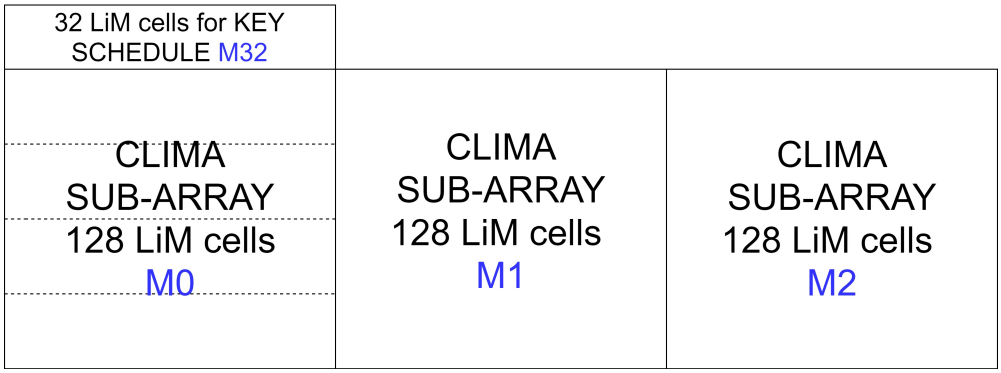

Figure 1.5: CLIMA array

bits each, and a row of 32 bits, for a total of 416 basic cells (from now on, we will call basic cell or Lim cell, the cell itself plus its interface). However it has to be remarked that, only `M1` and `M2` have a multiplexer for each cell (see previous report ([1]). Therefore, the contribution of power and area of the multiplexers have not to be considered for `M0` and `M32`.
Area, delay and power consumption of the whole CLIMA array have been obtained as follows:

- **Area**: in order to compute the total area of the CLIMA, the area of each cell has to be summed. So one can use the following equation to compute the total area:

$$A_{CLIMA} = (A_{cell} + A_{interface}) \cdot (416) + A_{mux} \cdot 256$$

- **Delay**: to compute the overall delay, the worst case has to be analyzed: since each cell of `M1` and `M2` is composed by a multiplexer and a LiM cell, the delay of the CLIMA can be computed as the delay of the multiplexer plus the delay of the basic cell:

$$\tau_{CLIMA} = \tau_{cell} + \tau_{mux}$$

- **Power**: the average power consumption of the CLIMA has been computed by considering both dynamic power and static power:
  - *Dynamic Power*: first of all, the dynamic power has to be computed when a XOR operation, of the memory array, is executed. Moreover, the implemented algorithm works in such a way that the three 128-bit arrays do not work simultaneously, except for the operations of writing and reading. Now the worst case is when the maximum number of cells work in the same clock cycle. This happens in the state 10 of the FSM (see report [1]): in this case the whole sub-array M1 and the first row of the sub-array M0 perform the XOR operation: for this reason, in order to compute the dynamic power, the following equation has been used:

$$P_{Dyn_{CLIMA}} = (P_{Dyn_{cell}} + P_{Dyn_{interface}}) \cdot (128 + 32) + P_{Dyn_{mux}} \cdot 128$$

  - *Static Power*: regarding the static power contribution, one has to take into consideration the whole CLIMA array (all three sub-arrays). Therefore the static power can be computed by considering all the LiM cells present in the whole CLIMA array and the multiplexers present in `M0` and `M1`:

$$P_{Stat_{CLIMA}} = (P_{Stat_{cell}} + P_{Stat_{interface}}) \cdot 416 + P_{Stat_{mux}} \cdot 256$$

Final remarks

In this step the following scripts have been written:

- mux2to1 1-bit;
- interface modelled by 48 multiplexers;
- interface modelled by using Synopsys parameters;
- CLIMA array, which contains the interface modelled by multiplexers;
- CLIMA array, which contains the interface modelled by using Synopsys parameters.

Each one of them provides an evaluation of the power consumption, area and delay (except for the interface scripts which do not compute the delay) and thanks to the two versions of the interface and of the CLIMA array, it is possible to compare the two different models and decide which one is better in terms of what one needs to analyze.

STEP 6

Introduction

The purpose of this step is to model the CLIMA surrounding logic. This model will be used to compute the same parameters analyzed in the previous step:

- delay;
- area;
- power consumption.

The external logic contains a lot of registers and S-Boxes because, as discussed in the previous report [1], not all the operations required by the AES algorithm can be executed inside the CLIMA array.
In order to estimate the value of power, delay and area, all the main building blocks have been modelled using Matlab and then they have been combined together in order to evaluate the worst case scenario. Therefore, for each block a Matlab script has been realized.
For sake of clarity the whole datapath (including the control unit) is reported here:

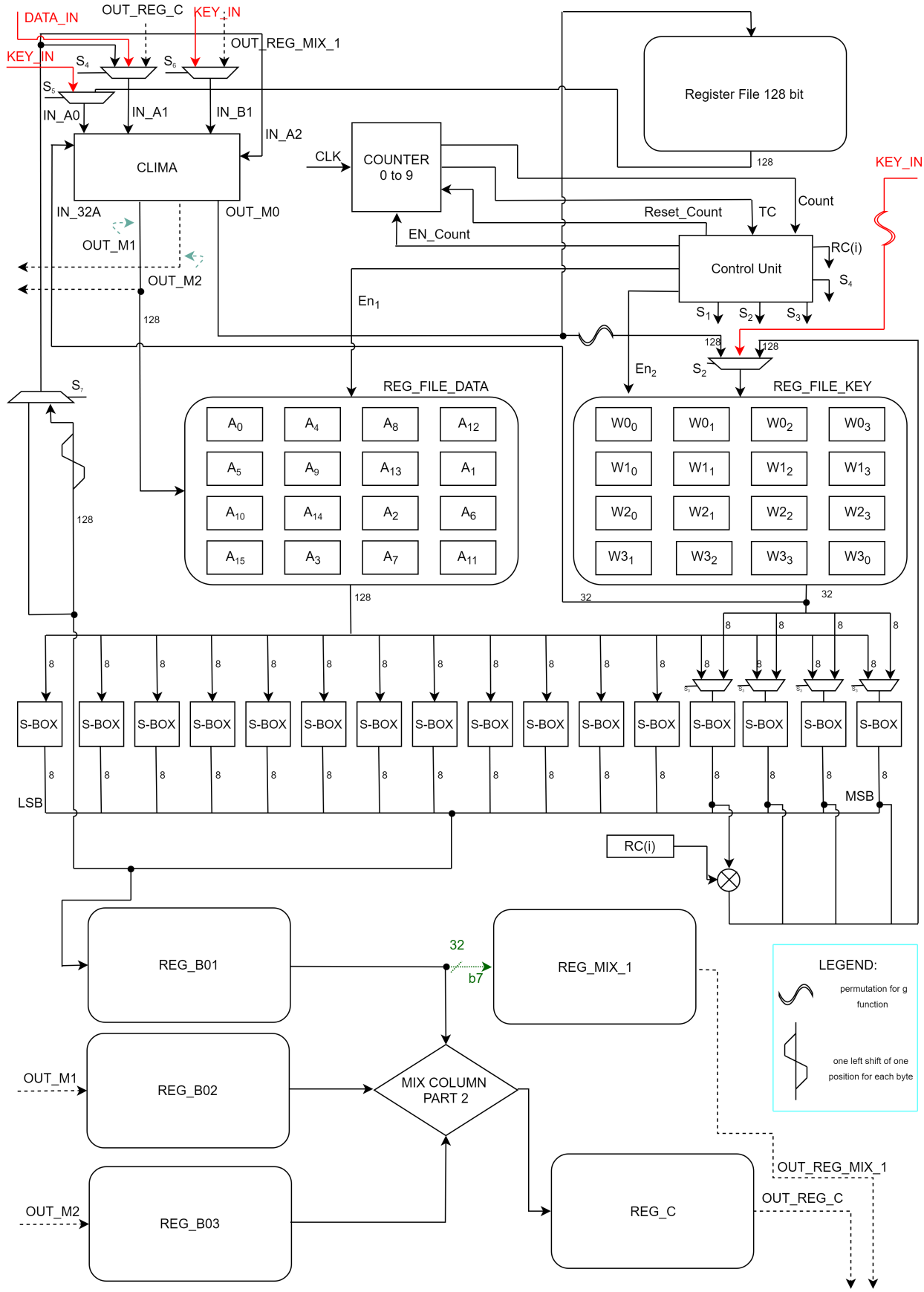

Figure 2.1: Complete architecture structure

Register Files

First, a Matlab model of the flip-flop has been realized. A flip-flop can be modelled using 2 consecutive D-Latches and an Inverter, as it is possible to see in the figure 2.2.

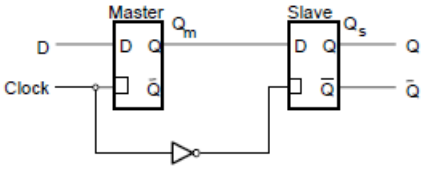

Figure 2.2: flip-flop

The structure of a D-Latch is reported below (figure 2.3).

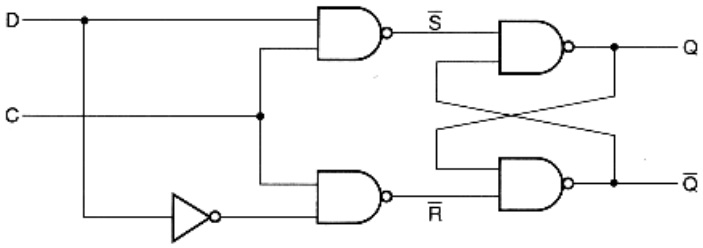

Figure 2.3: D-Latch

The inverter can be modelled by a single NAND2 gate. Therefore, the flip-flop model has a total of 11 NAND2 gates. In order to compute area, power and delay, the provided NAND2 script has been used. For what concerns the delay, it can be computed as the delay of 8 consecutive NAND2 gates. An overhead has to be added in order to take into account the clock to output delay, the setup time and the skew.
Once the model of the flip-flop has been realized, it is possible to analize all 8 register files in the datapath. It should be noted that all the 128-bit register files are read and written simultaneously, therefore there is no decoder inside these register files.
Consequently, each 128-bit register file has been modelled as 128 flip-flops.

Multiplexers

Inside the datapath there are:

- 3 2-to-1 128-bit multiplexers: each one of them can be modelled as 128 2-to-1 1-bit multiplexers;
- 2 3-to-1 128-bit multiplexers: every 3-to-1 128-bit multiplexer can be modelled as a sequence of two 2-to-1 128-bit multiplexers;
- 4 2-to-1 8-bit multiplexers: each 2-to-1 8-bit multiplexer can be modelled as 8 2-to-1 1-bit multiplexer.

It is clear that every multiplexer in the datapath can be represented as a combination of 2-to-1 1-bit multiplexers. So the model retrieved in the previous step (`Step 5` section 1.2) has been used as base model for all the multiplexers.

S-Boxes

The S-Box is a 128-bit Look-Up-Table, which can be modelled taking into account:

- the 128-bit array of cells;
- the 256-to-1 8-bit output multiplexer.
- the 8-bit input decoder.

The 128-bit array of cells can be modelled as 128 flip-flops, and the previous flip-flop model has been used for this purpose. For what concerns the output multiplexer, it is possible to note that it can be decomposed into a total of 255 2-to-1 8-bit multiplexers. Each one of them is modelled as 8 2-to-1 1-bit multiplexers. Therefore the output mux is made up of 2040 2-to-1 1-bit multiplexers.
It is clear that in practice an output multiplexer (specially one of this dimension) is not implemented in this way. Nevertheless this was the simplest way to model it.
Synopsys has been used to synthesize the output multiplexer, in order to have an idea on how it is actually implemented by means of basic logic gates, but the circuit was too complex with a lot of different logic gates, so the wide multiplexer model has been kept.
The 8-bit input decoder, instead, has not been modelled by means of basic logic gates because it has a very big and complex structure. Therefore power, area and delay parameters have been computed by using Synopsys and then they have been inserted as given parameters in the Matlab script.
For what concerns the power consumption of the overall S-Box, the following approximations have been used:

- Dynamic Power: the 128-bit array of cells contribution is modelled as the one of 128 flip-flops. On the other hand, the dynamic power of the output multiplexer has been computed taking into account that only 8 2-to-1 8-bit multiplexers over the total of 255 are active simultaneously. As already discussed the input decoder contribution has been derived from the Synopsys synthesis;
- Static Power: the 128-bit array of cells contribution is modelled as the one of 128 flip-flops. The input decoder contribution, instead, is the one provided by Synopsys synthesis. For what concerns the output multiplexer static power, the I_{gate} and the I_{off} contributions have been separated as follows:
  - the I_{gate} of only 8 2-to-1 8-bit active multiplexers have been taken into account;
  - the I_{off} of the remaining 247 2-to-1 8-bit inactive multiplexers is considered.

The values of dynamic power and static power of the Matlab model of the S-Box are compared with the ones provided by the Synopsys synthesis of the S-Box: the dynamic power of the Matlab model is actually pretty close to the Synopsys one while the static power given by Matlab is approximately 7 times higher than the Synopsys one. Therefore, it has been decided to realize another Matlab script for the S-Box which contains the parameters given by the Synopsys synthesis. For this reason, two different scripts of the whole datapath have been also realized: the first one uses the S-Box modelled with Matlab (except for the input decoder) while the latter uses the S-Box parameters given by the Synopsys synthesis.

Counter

The 4-bit counter can be modelled as 4 flip-flops and 2 AND2 gates. Moreover, one AND2 gate can be realized using 1 NAND2 gate and 1 INVERTER (which can be modelled as another NAND2 gate). So at the end the dynamic power, as well as the static one, can be computed by summing up the power of 4 flip-flops and 4 NAND2 gates.
For the area value, it is sufficient to sum the area of all the flip-flops and all the NAND2 gates.

Mix Column Part 2

The structure of the Mix Column Part 2 block is made up of a series of XOR2 gates. In particular, there are 48 8-bit XOR2 gates in this block. Each 8-bit XOR2 gate can be modelled as 8 1-bit XOR2 gates. So at the end this block has been modelled with 384 1-bit XOR2 gates. The 1-bit XOR2 gate model has been provided by the technological group, so it was sufficient to multiply the value of area and power of the single XOR gate for 384, to obtain the parameters needed to model the entire Mix Column Part 2 block.

Key schedule

The key schedule block is shown in the following figure:

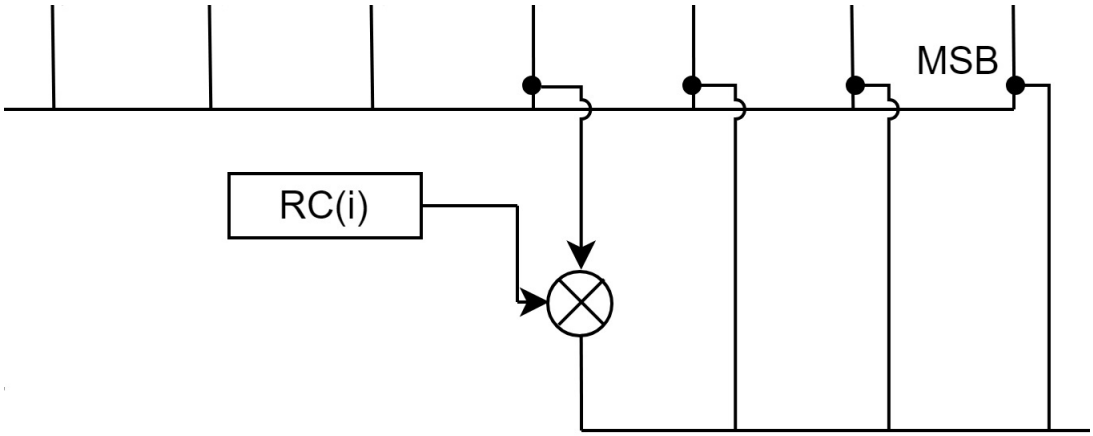

Figure 2.4: Key schedule block

It is made up of a RC(i) register, which is simply a 8-bit register, and a 8-bit XOR2 gate, which has been modelled as 8 1-bit XOR2 gates. The 8-bit register has been modelled with 8 flip-flops as discussed in the `Register files` section (2.2).
To compute the power and the area of this block, it is sufficient to take into account all the contributions of the 8 flip-flops and of the 8 1-bit XOR2 gates.

Control Unit

The control unit is too complex to be modelled by means of basic logic gates, because there is an high number of states as one can see in the figure 2.5.
The only way to obtain an estimation of power, area and delay is to use Synopsys to synthesize the control unit and to compute these parameters.

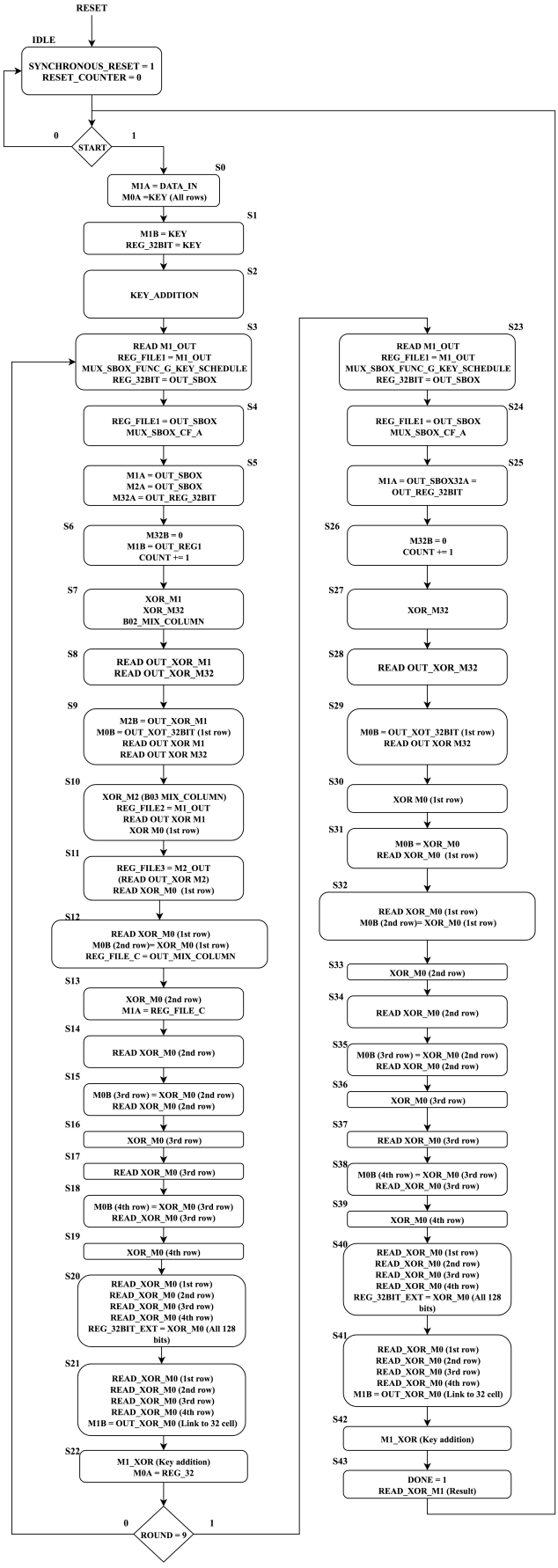

Figure 2.5: Control Unit

The following results have been obtained:

	Control Unit	**Measurement unit**
Pdyn	12.17	μW
Pstatic	5.35	μW
Ptot	17.52	μW
Delay	1.01	ns
Area	212.8	μm^2

It has to be remarked that all the power values and the area have been obtained with a frequency of $F_{CLK} = 100\,\text{MHz}$ and a technological node of 45 nm. The delay instead has been obtained as the minimum clock period for which the slack is zero.

Complete model of the CLIMA surrounding logic

Once all the external blocks have been modelled, it is possible to realize a Matlab script that computes the power consumption, the area and the delay of the whole datapath including the control unit but without the CLIMA array. This script uses the output values coming from the models of all the previous blocks.
As described in section 2.4, 2 different datapath scripts for the CLIMA surrounding logic have been realized: the first one uses the S-Box modelled by using Matlab, while the latter uses the S-Box parameters given by the Synopsys synthesis. For both of them the overall area, delay and power consumption have been computed as follows:

- **Area**: sum of the area of each block, including the area of the Control Unit.
- **Power**: the average power consumption of the CLIMA surrounding logic has been computed. In order to do so, both dynamic power and static power have to be considered:
  - *Dynamic Power*: in order to compute this value, the Finite State Machine of the architecture has been analyzed in order to find the state in which the highest number of blocks are used. It is possible to see that this happens in the S4 state, where one register file, 4 2-to-1 8-bit multiplexers and all the 16 S-Boxes are used. Therefore, the dynamic power is the sum of the contributions of these blocks plus the control unit contribution.
  - *Static Power*: sum of the static power of each block (including the static power of the control unit).
- **Delay**: The critical path of the surrounding logic (figure 2.6) has been computed thanks to the Synopsys synthesis. More specifically, the critical path goes through:
  - the control unit: Synopsys analysis shows that the critical path goes through the following gates in the control unit:
    - 1 flip-flop;
    - 1 AND3X2 gate which can be modelled as 3 consecutive NAND2 gates with fan-out equals to 1 and 1 NAND2 gate with fan-out equals to 2;
    - 2 NAND2 gates with fan-out equals to 1.
  - 1 2-to-1 8-bit multiplexer, which has a delay equal to the one of a 2-to-1 1-bit multiplexer;

- 1 S-Box. The delay of this block can be seen as the sum of the delay of 1 flip-flop (due to the array of memory of the LUT) plus the delay of the input decoder and the one of the output multiplexer. By taking into account the model of the output multiplexer exploited previously in this step (section 2.4), its delay is equal to 8 times the delay of 1 2-to-1 1-bit multiplexer. The delay of the input decoder is instead the one coming from the Synopsys analysis.
- 1 2-to-1 128-bit multiplexer, which has a delay equal to the one of 1 2-to-1 1-bit multiplexer.
- 1 3-to-1 128-bit multiplexer which, as it has been previously described (section 2.3), can be modelled as 2 consecutive 2-to-1 128-bit multiplexers. Therefore it has a total delay which is 2 times the one of a 2-to-1 1-bit multiplexer.

The overall delay of the surrounding logic has been computed by summing up the delay of all these blocks.

Matlab model of the whole architecture

As a final step, a comprehensive Matlab model of the whole architecture (CLIMA & surrounding logic) has been realized. More specifically, 2 different scripts are used:

- The first one uses the Matlab model of the LiM input cell interface and of the S-Box output multiplexer.
- The second one uses power, area and delay parameters given by the Synopsys synthesis for both the LiM input cell interface and the whole S-Box.

Both scripts simply sum up the power and area contributions of the CLIMA array and the surrounding logic. The delay of the whole architecture, instead, it is analyzed in the following section 2.11.

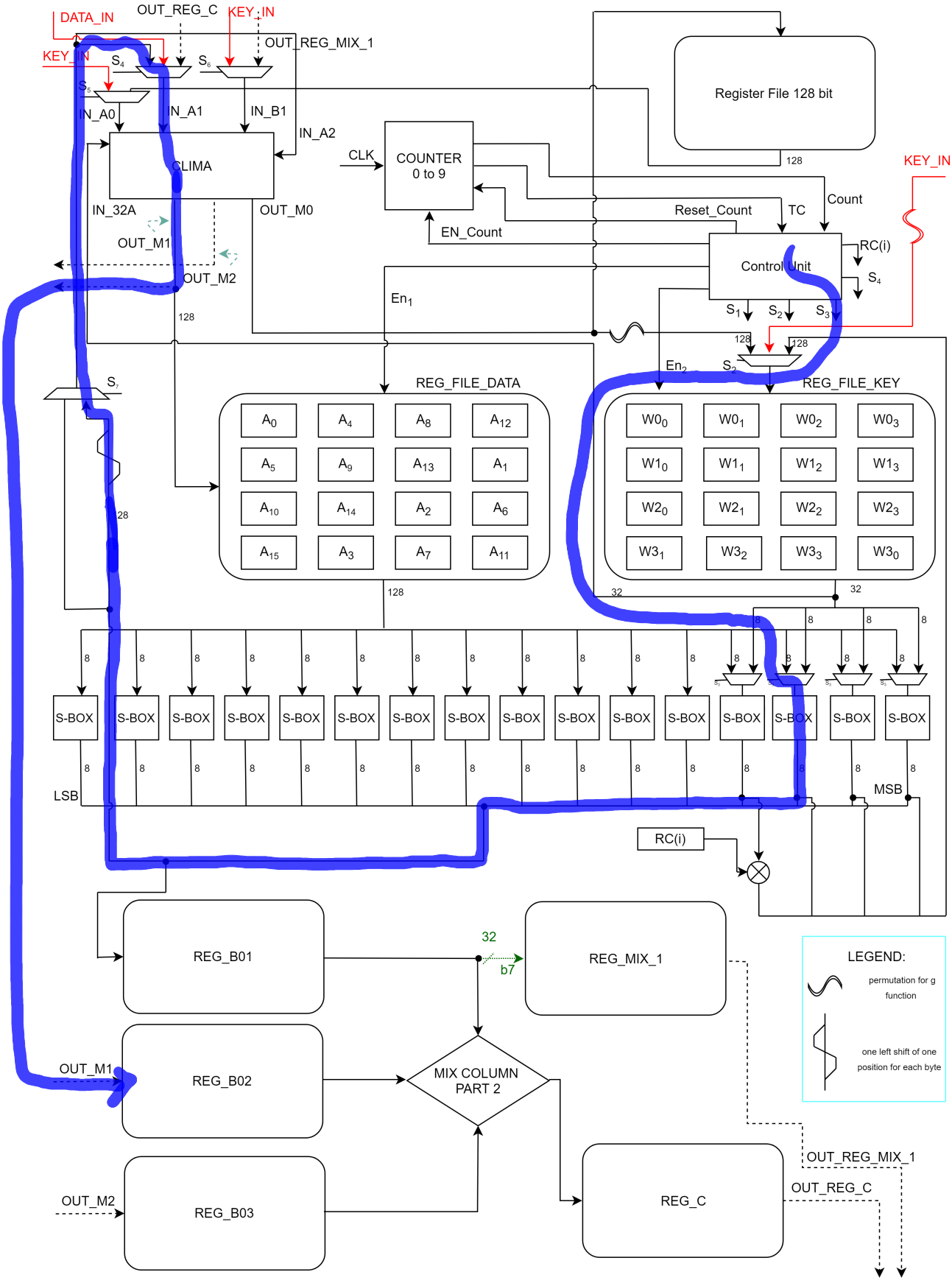

Figure 2.6: Complete architecture with Critical Path highlighted

Critical Path of the whole architecture

The critical path of the whole architecture is highlighted in figure 2.6. As one can see, it goes though the surrounding logic and enters into the CLIMA array. Since the CLIMA array is an asynchronous block, the critical path has to take into account also its delay. As one can see from the figure 2.6, the critical path after the CLIMA array, goes into the register file B02. Therefore, in order to compute the delay of the whole architecture, including both the CLIMA array and the surrounding logic, the two delay contributions are added together. An overhead contribution, as previously mentioned in the description of the flip-flop model, has to be taken into account:

$$T_{total} = T_{CLIMA} + T_{SurroundingLogic} + T_{Overhead} = 1.3 \cdot (T_{CLIMA} + T_{SurroundingLogic})$$

where

$$T_{Overhead} = 0.3 \cdot (T_{CLIMA} + T_{SurroundingLogic}) \tag{2.1}$$

Final remarks

In this step a Matlab script that models the CLIMA surrounding logic has been written. Unfortunately the control unit and the decoder were too complex to be modelled by Matlab, so Synopsys has been used to compute the main parameters, like area, power and delay.
The interface and the S-Box have been modelled by using basic blocks like flip-flops and multiplexers, but their static power was too different with respect to the Synopsys one, so another model which contains the Synopsys values has been written for both of them.
It is clear that the model carried out in this step is a very rough approximation of the CLIMA surrounding logic but due to its complexity this was the best possible approximation.

STEP 7

Introduction

The purpose of this last step is to use all the models retrieved in the previous steps, to combine them with the technology group cell model and to perform a complete test in order to compute delay, area and power consumption of the whole architecture including the CLIMA array.
These test results have to be compared with the Synopsys synthesis ones in order to see if the models built until now are a good approximation of the whole architecture.
Furthermore, the first AES implementation, without Logic in Memory, has been synthesized with Synopsys in order to be able to compare better the performance differences between the two implementations.
From now on the AES implementation without Logic in Memory will be called CMOS architecture, while the AES implementation with Logic in Memory will be called LiM architecture.
Synopsys simulations have been performed with 45 nm NAND2 library, while Matlab ones with LOP-2005 technology. Area and power results for both Synopsys and Matlab simulations have been carried out at $F_{CLK} = 100\,\text{MHz}$.

CMOS architecture (without Logic in Memory) synthesis results

By using Synopsys, the CMOS architecture has been synthesized and the following results have been obtained:

	CMOS architecture	Measurement unit
Pdyn	1.58	mW
Pstatic	0.78	mW
Ptot	2.36	mW
Area	33047.31	μm^2

It has to be remarked that all the power values and the area have been obtained with a frequency of $F_{CLK} = 100\,\text{MHz}$ and a technological node of 45 nm.
The minimum clock period for which the slack is zero is the following one:

	CMOS architecture	Measurement unit
$\boldsymbol{T_{CLK}\vert_{MIN}}$	1.33	ns
$\boldsymbol{F_{CLK}\vert_{MAX}}$	751.88	MHz

Power, area and delay results of the CLIMA surrounding logic

In the following section, the results of the Matlab model and the ones of the Synopsys synthesis for the CLIMA external logic have been reported.
From the Synopsys synthesis of the CLIMA surrounding logic, the following results have been obtained:

	CLIMA surrounding logic	Measurement unit
Pdyn	1.01	mW
Pstatic	0.43	mW
Ptot	1.44	mW
Area	17910.31	μm^2
$\boldsymbol{T_{CLK}\vert_{MIN}}$	1.71	ns
$\boldsymbol{F_{CLK}\vert_{MAX}}$	584.79	MHz

Table 3.1: Synopsys results for the CLIMA surrounding logic

It has to be remarked that all the power values and the area have been obtained with a frequency of $F_{CLK} = 100\,\text{MHz}$ and a technological node of 45 nm. The $T_{CLK}|_{MIN}$ is the minimum clock period for which the slack is zero.
In the following table instead, the Matlab results for the surrounding logic have been reported:

	Surrounding logic (mat)	Surrounding logic (syn)	Meas. unit
Pdyn	1.4	1.10	mW
Pstatic	1.9	0.74	mW
Ptot	3.3	1.84	mW
Area	13015	7832.4	μm^2
$\boldsymbol{T_{CP}}$	3.3	1.92	ns

Table 3.2: Matlab results for the CLIMA surrounding logic

It has to be remarked that there are two Matlab models for the surrounding logic: in the column `Surrounding logic (mat)` the results of the one which uses the Matlab model also for the output multiplexer of the S-Box are reported, while in the column `Surrounding logic (syn)` there are the results of the model which uses Synopsys value for the entire S-Box. In this table T_{CP} stands for the delay of the critical path of the surrounding logic evaluated by using Matlab and it does not include the contributions of setup time, clock to output delay and skew. For this reason it is has not been called T_{CLK}.
By comparing the table 3.1 and 3.2, it is possible to see that the results obtained by the Matlab models are pretty close to the Synopsys ones. As discusses in the section 2.4, the output multiplexer modelled by Matlab has a static power and a delay too different with respect to the Synopsys ones and this influences the overall results of the surrounding logic. In this case the best approximation for all the parameters is represented by the surrounding logic modelled by using the S-Box results computed by Synopsys (column `Surrounding logic (syn)`).
For what concerns the area, the best approximation in this case is obtained by the Matlab model of the S-Box (column `Surrounding logic (mat)`). However the Synopsys value for the area is still higher and this probably because when it performs the synthesis, it does some optimisation to go faster despite increasing the area.

Power, area and delay results for the CLIMA array

In this section, the CLIMA model has to be verified and compared with the results obtained from the previous report [1]. However the technological group has produced a new version of the basic LiM cell with more accurate power and area values. For this reason a new VHDL simulation has been executed and different values for power and area have been obtained with respect to the previous report [1].

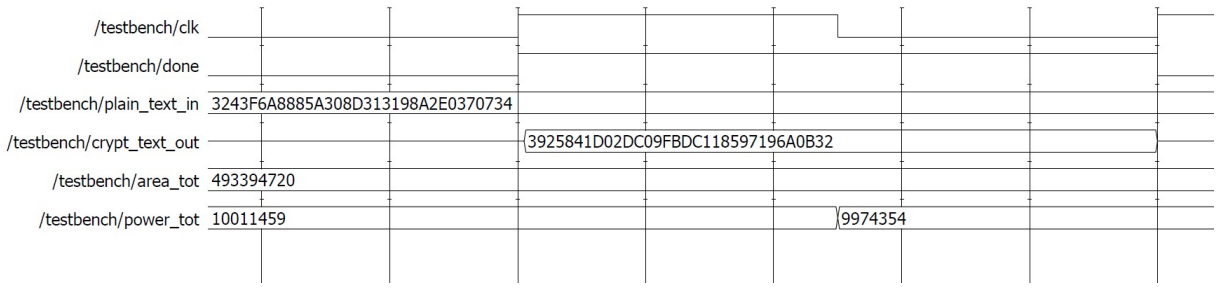

Figure 3.1: VHDL simulation - Power and area results

The total power of the CLIMA is:

$$P_{TOT} = 9\,974\,354\,\text{nW} + P_{MUX} + P_{interface} = 9.97\,\text{mW} + P_{MUX} + P_{interface} \tag{3.1}$$

While the total area is:

$$A_{TOT} = 493\,394\,720\,\text{nm}^2 + A_{MUX} + A_{interface} = 493.39\,\mu\text{m}^2 + A_{MUX} + A_{interface} \tag{3.2}$$

It has to be remarked that only `M1` and `M2` basic cells have the multiplexer (as discussed in section 1.5). For this reason the final power will be the sum of the power obtained by the simulation plus the power of 256 multiplexers, by taking into account their switching activities. Moreover each basic cell has its interface which consumes a certain power and has a certain area, as described in the section 1.3. So in the final power and area expressions, the contributions of the multiplexers and of the interfaces have to be added.
For what concerns the Matlab CLIMA model, it has to be highlighted that the technology group has provided 3 different versions of the LiM basic cell:

- CMOS LiM basic cell model (without Memristor model);
- Memristor LiM basic cell model, with the output buffer which has been modelled by Matlab;
- Memristor LiM basic cell model, with the output buffer which has been modelled by using Cadence Virtuoso parameters.

In the following table all the 3 versions results in terms of power, area and delay have been reported:

	CLIMA - CMOS version		**CLIMA - Memristor version 1**		**CLIMA - Memristor version 2**		**Meas. unit**
	CLIMA (mat)	**CLIMA (syn)**	**CLIMA (mat)**	**CLIMA (syn)**	**CLIMA (mat)**	**CLIMA (syn)**	
Pdyn	0.94	0.55	0.91	0.52	0.91	0.52	mW
Pstatic	3.32	0.50	15.04	12.22	18.19	15.38	mW
Ptot	4.26	1.05	15.95	12.74	19.10	15.9	mW
Area	4642.8	8004.7	4709.4	8071.4	4709.4	8071.4	μm^2
Delay	0.31	0.31	5.51	5.51	5.51	5.51	ns

Table 3.3: Matlab results for the CLIMA array models

In the column `CLIMA - CMOS version`, the parameters of CMOS LiM basic cell model (without Memristor model) have been reported; in the column `CLIMA - Memristor 1 version`, the parameters of Memristor LiM basic cell model with the output buffer modelled by using Matlab have been reported while in the column `CLIMA - Memristor 2 version`, the parameters of the last version of Memristor LiM basic cell model with the output buffer characterized by Cadence Virtuoso have been reported.
For each version of the CLIMA, there are two different scripts due to the different models of the interface (section 1.3): the matlab model of the interface is referred as `CLIMA (mat)` while the Synopsys one is indicated as `CLIMA (syn)`.
As it it possible to see from the table 3.3, the delay of the CLIMA array implemented with Memristor is very high with respect to the CMOS version one. Since the surrounding logic delay is much lower than the CLIMA array, the clock frequency is limited by the CLIMA array itself. It should be noted that to improve the throughput, one could duplicate the entire architecture and performs two encryption in the same amount of time. However this solution is not recommended because it means doubling the total power and the area and it would be very inefficient.
Regarding the other results, the best approximation for the CLIMA array is obtained by looking at the column `CLIMA - Memristor version 2` and the sub-column `CLIMA (syn)`. This because in this column, the output buffer of the LiM basic cell has been characterized by Cadence Virtuoso, while the interface of each basic cell has been modelled by using the Synopsys values.

The total power obtained by the Matlab model of the whole CLIMA array is actually pretty close to the one provided by the VHDL simulation, keeping in mind that in the VHDL simulation, neither the interface nor the multiplexer power consumption have been computed.
The area value, instead, requires a more detailed analysis: to understand better why there is so much difference between the Matlab value and the VHDL value, it is important to analyze the area of the interface, of the LiM basic cell and of the multiplexer:

	Interface (syn)	**Interface (mat)**	**LiM cell - Memristor version**	**Mux**	**Meas. unit**
Area	18.09	10.01	1.19	0.21	μm^2

Table 3.4: Interface, LiM cell and mux area

From the above table (table 3.4), it is possible to see that if one multiplies the LiM cell area for the number of the cells present in the CLIMA array, he would obtain the same number provided by the VHDL simulation. However the interface area is one order of magnitude greater than the LiM cell one and almost two order of magnitude higher than the multiplexer area. On top of that each cell has an interface so its contribution is the main reason for which the Matlab area is so different with respect to the VHDL one.
It should be noted that the interface area modelled as 48 multiplexers (`Interface (mat)` column) is smaller with respect to the one modelled by using Synopsys value, but still higher then the LiM basic cell and the multiplexer. To obtain a smaller area of the entire CLIMA array (table 3.3), one could consider one of the two column `CLIMA - Memristor version 1 or 2` and the sub-column `CLIMA (mat)`, which uses the 48 multiplexers model for the interface.

Comparison between CMOS and LiM architectures

In this section the CMOS architecture (without LiM), synthesized by Synopsys, and the LiM architecure, modelled by Matlab, are compared.
The choice to use the Matlab model of the LiM architecture, for this comparison, has been made mainly because Synopsys can not synthesize the CLIMA array and even if a VHDL model has been simulated, it computes only area and power of the CLIMA array considering only the LiM basic cell, without interface and input multiplexer. Furthermore the CLIMA array total power obtained by the Matlab model is very close to the one provided by the VHDL simulation so we expect to do a fair comparison by considering a Matlab model, which is also more accurate than the VHDL one.
For the Matlab model of the LiM architecture, the CLIMA - Memristor version 2 (see table 3.3), both `CLIMA (mat)` and `CLIMA (syn)` models, and both models of the surrounding logic (see table 3.2) have been used.

	CMOS architecture	**LiM architecture**		**Meas. unit**
		architecture (mat)	**architecture (syn)**	
Pdyn	1.58	2.30	1.60	mW
Pstatic	0.78	20.00	16.10	mW
Ptot	2.36	22.30	17.70	mW
Area	33047.31	17724	15904	μm^2
$\boldsymbol{T_{CLK}\vert_{MIN}}$	1.33	11.47	9.66	ns
$\boldsymbol{F_{CLK}\vert_{MAX}}$	751.88	87.18	103.52	MHz
Latency	54	204	204	Clock cycles

Table 3.5: Comparison among AES different implementations

These results have been obtained with a 45 nm NAND2 library for the CMOS architecure and a LOP-2005 technology (45 nm) for the LiM architecture.
Power and area of all the architecture have been obtained by using a clock frequency equals to $F_{CLK} = 100$ MHz.
From the table 3.5, it is possible to see how the CMOS architecture is better than the LiM architecture basically in everything. The difference in terms of power and clock period is one order of magnitude in favour of the CMOS solution.
The area value deserves a separate dissertation: the CMOS architecture is certainly bigger but it can encrypt five data per time while the LiM architecture can only encrypt one datum per time with a very high latency. Moreover it should be noted that the area value obtained by the Matlab module is underestimated. It is possible to obtain a more accurate value by considering an overhead of 1.3 for the LiM architecture. In this case the area results would be the following ones:

	CMOS architecture	**LiM architecture**		**Meas. unit**
		architecture (mat)	**architecture (syn)**	
Area	33047.31	23041.20	20675.20	μm^2

Table 3.6: Area comparison with overhead

As it is possible to see the area is still smaller for the LiM architecture but this difference is still not enough to justify the losses in terms of performance.

To clarify better this last point, the throughput of both architectures have been computed by using the following formula:

$$\text{Throughput} = \frac{n_{bit}}{n_{clock} \cdot T_{clock}} \tag{3.3}$$

where n_{bit} is the number of bit encrypted per time, while n_{clock} is the number of clock cycle required to finish one encryption. Since in both the architecture it is impossible to start a new encryption before ending the previous one $n_{clock} = latency$.
It has to be remarked that the CMOS architecture can encrypt five data per time:

$$n_{bit} = 5 \cdot 128 \text{ bits} = 640 \text{ bits} \tag{3.4}$$

while the LiM architecture can encrypt one datum per time:

$$n_{bit} = 128 \text{ bits} \tag{3.5}$$

Between the two models of the LiM architecture, the clock frequency changes, so two different throughput have been obtained.

	CMOS architecture	**LiM architecture**	
		architecture (mat)	**architecture (syn)**
Throughput	8.91 Gbps	54.70 Mbps	64.95 Mbps

Table 3.7: Throughput comparison

As one can see from the table 3.7, the difference in terms of throughput is more than two order of magnitude between the two hardware implementations.
To be noted that even though there are two model of the LiM architecture, the hardware implementation is still the same. What changes is just the evaluation of parameters like area, power and delay.

Performances comparison with other AES HW implementations

In this section a comparison among different AES hardware implementations has been done. The data have been taken by the paper [2].

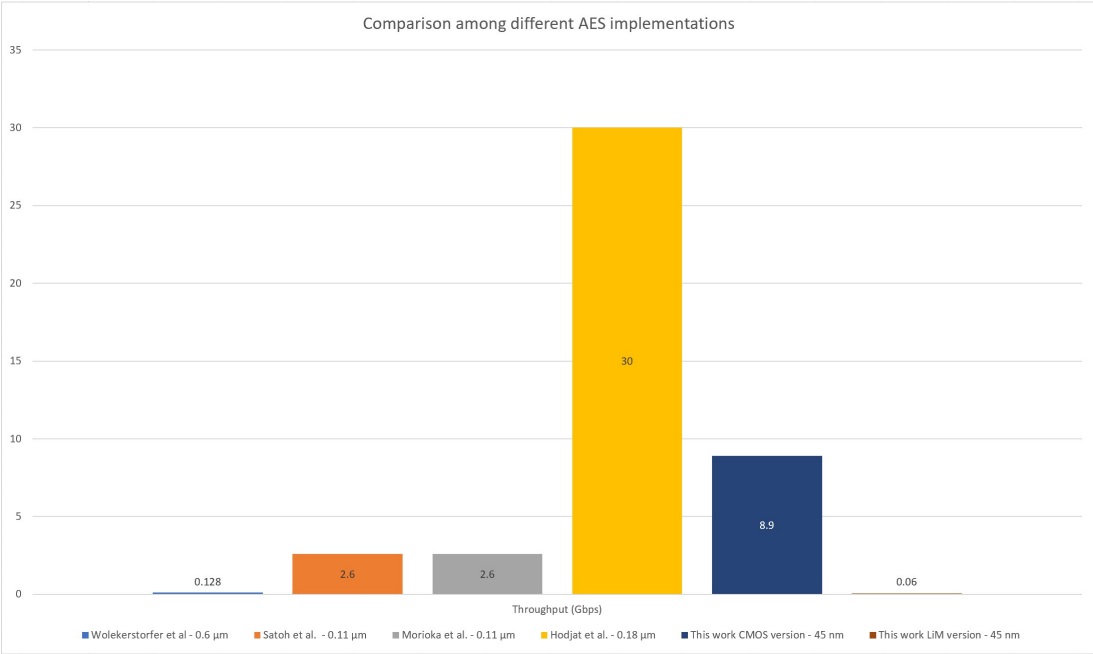

Figure 3.2: Comparison between different AES HW implementations

As it is possible to see from the figure 3.2, the CMOS implementation, even though it uses a newer technology node, has a throughput which is greater than some other solutions. Anyhow there are high speed AES implementations which can reach 130 Gbps.
It should be noted though that the aim of this project was not to create the fastest AES hardware implementation but to compare the CMOS architecture (without LiM) with the LiM one.
Therefore, by comparing the LiM implementation of the AES algorithm based on Memristor technology with other implementations, it is possible to notice that it has the lowest throughput among all.

Final remarks

The AES algorithm performs very different and complex operations. The CMOS architecture (without Logic in Memory) was not so difficult to design, while the LiM one has represented a challenge due to the algorithm itself and due to the Memristor technology which is not very suitable for this algorithm.
Moreover as we have already discussed in the previous report [1], not all the operations have been implemented in memory: half of the algorithm is performed outside the memory which means that we need a very big external logic, other than the memory itself, to execute the entire algorithm. On top of that the LiM architecture has a very high latency which makes worst the throughput, and this mainly because we need to do some operations inside the memory, then we have to take the data outside because there are some operations to be performed with the external logic and then we rewrite the data in the memory to continue the encryption of the data.
So it is clear that a logic in memory architecture can help when the whole algorithm or at least the most expensive operations can be done inside the memory without the need to move the

data from one side to the other. In the AES algorithm this does not happen and the result is that the LiM solution is worst in everything with respect to the no logic in memory architecture. In conclusion we can say that the AES algorithm is not suitable for a Logic In Memory implementation based on Memristor.

Bibliography

[1] M. Spada, R. Massa, and B. Martino, *FINAL REPORT AES*.

[2] L. Ali, I. Aris, F. S. Hossain, and N. Roy, "Design of an ultra high speed aes processor for next generation it security", *Elsevier*, 2011.