POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Logic in Memory implementation of an AES algorithm based on skyrmions

Relatori:
prof. Mariagrazia Graziano
prof. Maurizio Zamboni
prof. Marco Vacca

Candidato:
Edoardo Mandalari

Luglio 2020

Table of contents

Summary III

1 Fundamentals of skyrmion **1**
- 1.1 Historical facts . . . 1
- 1.2 Physical properties . . . 1
- 1.3 Elementary functionality of skyrmion . . . 2
  - 1.3.1 Skyrmion creation and annihilation . . . 2
  - 1.3.2 Skyrmion detection . . . 4
  - 1.3.3 Skyrmion motion . . . 6
- 1.4 Advanced functionality of skyrmion . . . 8

2 Logic-in-Memory architecture **13**
- 2.1 State of art . . . 13
- 2.2 Technologies supporting LiM architecture . . . 14
- 2.3 Configurable logic-in-memory architecture . . . 17

3 Advanced Encryption Standard algorithm **19**
- 3.1 Mathematical background . . . 19
- 3.2 Algorithm description . . . 20
  - 3.2.1 SubBytes . . . 22
  - 3.2.2 ShiftRow . . . 22
  - 3.2.3 MixColumns . . . 23
  - 3.2.4 AddRoundKey . . . 23
  - 3.2.5 Key scheduler . . . 24

4 AES algorithm based on magnetic skyrmion **26**
- 4.1 Memory array structure . . . 26
- 4.2 Writing process . . . 27
- 4.3 AddRoundKey operation . . . 32
- 4.4 SubBytes operation . . . 33
- 4.5 ShiftRow operation . . . 35

4.6 MixColumns operation 37
4.7 Reading process 39
4.8 Key expansion block 39
4.9 Control unit 41
4.10 VHDL Model 49
4.11 Simulation 51
4.12 Architecture performance 51

5 Conclusions **53**

Summary

In the last decade, the incessant miniaturization of CMOS devices, as predicted by Moore's law, caused limitations such as an increase of leakage current and power consumption. For this reason, many researchers began to investigate other fields in order to overcome these problematics. Among these, spintronics surely stands out. It exploits electron spin properties, instead of electric charge (which is exploited by CMOS technology), leading to many advantages. Indeed, spintronic devices can offer higher computing speed and storage capacities by dissipating less energy, compared with CMOS chips. Among all spintronic objects, skyrmion are the most interesting one because of their outstanding properties. Indeed, these quasi-particle objects require lower depinning current than other spintronic objects (like domain walls) and their size is smaller, allowing a denser storage of information. For this reason, they represent a promising technology that in the future could become the key of modern computing devices.

Chapter 1 of this work focuses on skyrmion characteristics, examining all applications in which they are used to carry information. In particular, it has been studied how skyrmions can be nucleated, annihilated, moved and detected. In addition, it shows some papers in which basic logic functions, such as AND, NOT, OR, have been implemented and simulated by micromagnetic software.

Chapter 2 examines another important topic: modern computing architecture. It is known that most of them are based on the *Von Neumann* paradigm, which is based on the exchange of data between the central processor unit (CPU) and a memory. Consequently, this paradigm performance is heavily based on the speed of CPU and memory. However, in the last decades, CPU performance increased much more than memory ones, leading to a bottleneck: CPU is limited by memory speed. Logic-in-memory approach attempts to overcome this limitation by: (1) bringing computation inside memory, deleting the bottleneck, and (2), as a consequence, external memory requests decrease drastically, reducing the overall power consumption. The goal of this thesis has been to exploit skyrmion in order to implement a logic-in-memory architecture.

Next step has been to choose an algorithm that is able to exploit logic-in-memory characteristic. Advanced Encryption Standard (AES) has all key properties, indeed,

it has high level of parallelism and its operations are not that complex. Chapter 3 describes the algorithm, providing a mathematical background in order to let the reader understand easily how operations are performed.

Eventually, Chapter 4 illustrates how the algorithm has been implemented. In particular, all AES operation has been described, explaining how the writing process works and how skyrmion are guided inside the nanotracks. Moreover, due to the complexity of the algorithm, it has been used a technique called "linked" state machines, in which, instead of having only one FSM controlling the whole algorithm, there are multiple one that call each other whenever a certain state is reached.

Chapter 1

Fundamentals of skyrmion

1.1 Historical facts

The existence of skyrmions as particle-like states in acentric magnets was first studied theoretically by Bogdanov [1], who carried out and presented results on experiments related on micromagnetic theory of surface-induced Dzyaloshinskii-Moriya interaction [2]. In particular, a theory on the existence of chiral skyrmions in B20-type bulk helimagnets caused an uproar among scientific community [3]. However, it was 2009 when the first experimental observation of skyrmions in B20-type chiral materials made by Mühlbauer [4] was performed, leading to physicists to focus more on skyrmions. Recently, observing skyrmions at room temperature without a magnetic field has been achieved [5], representing a key milestone for the practical applications of skyrmions.

1.2 Physical properties

Skyrmions are topologically stable field configurations with particle-like properties which have been predicted theoretically and observed experimentally in condensed matter systems. A skyrmion is characterized by three numbers: the Pontryagin number Q_s, the vorticity Q_v and the helicity Q_h. It is called a skyrmion (anti-skyrmion) when the Pontryagin number Q_s is positive (negative). Vorticity of a skyrmion is defined by the winding number of the spin configurations projected into the s_x-s_y plane [6]. The skyrmion number Q_s can be evaluated by taking into account this formula:

$$Q_s = -\frac{1}{4\pi}\int dxdy\Big(\mathbf{m}(r)\cdot\Big(\partial_x\mathbf{m}(r)\times\partial_y\mathbf{m}(r)\Big)\Big) \tag{1.1}$$

A remarkable property is that the helicity number Q_h does not contribute to the topological number, which is uniquely determined by the type of the Dzyaloshinskii-Moriya interaction (**DMI**) [7]. Different values of helicity Q_h determines a different skrymion configuration. In particular, a skyrmion with the helicity 0 and π corresponds to the **Néel** skyrmion, while a skyrmion with the helicity $\pi/2$ and $3\pi/2$ corresponds to the **Bloch** skyrmion [8].

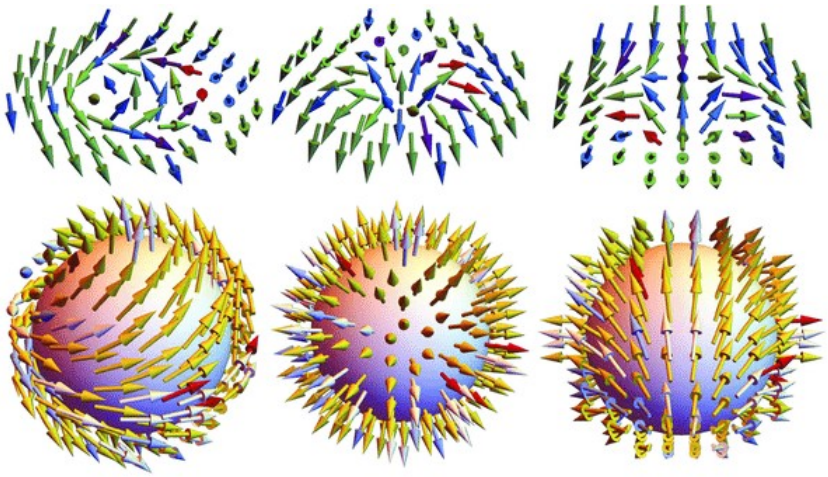


Figure 1.1: (Top) Bloch (left) and Néel (middle) skyrmions with topological charge $Q_s = 1$ and polarity $p = 1$. Antiskyrmion (right) with topological charge $Q_s = 1$ and $p = 1$. (Bottom) For Néel skyrmion (left), Bloch skyrmion (middle), and antiskyrmion (right) the moments wrap around a unit sphere upon application of stereographic projection. Figures extracted from [9].

1.3 Elementary functionality of skyrmion

In order to employ skrymions in practical application, it is important to pinpoint how these objects can be created, moved, detected and manipulated. Therefore, these elementary functionality are discussed in this section.

1.3.1 Skyrmion creation and annihilation

Skyrmion creation (nucleation) can be carried out in different ways: by taking into account an electrical current, or a magnetic field, or local heating using laser irradiation [10][11][12]. However, the most reliable and consistent way of nucleating a skrymion is by injecting an external electrical current. Nucleation of skyrmion in nanostructures of thin magnetic films has been achieved in [10], by exploiting the

spin transfer torque (**STT**). Regarding nucleation by electric current, the procedure consists in simply injecting electical current into the disk as Figure 1.2 shows.

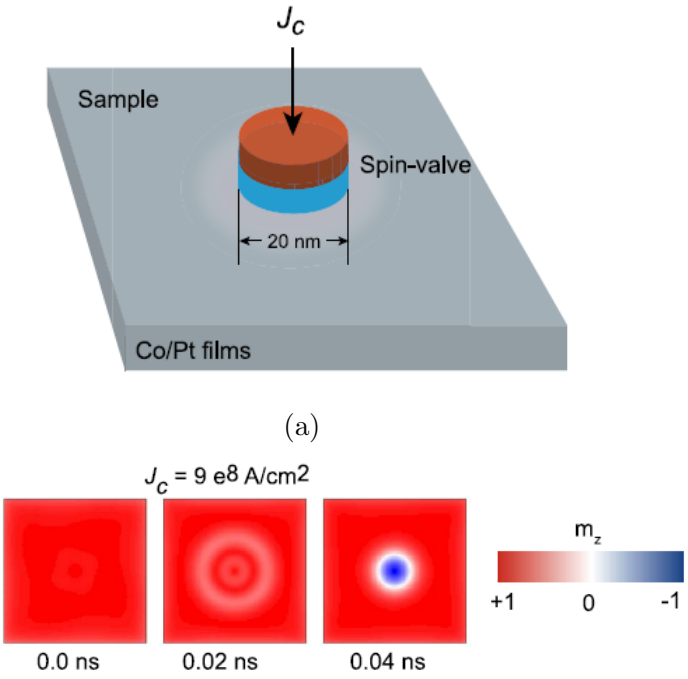


Figure 1.2: (a) Schematic of skyrmion creation by injection of an electrical current. (b) Time-resolved magnetization dynamics of a disk under a current density of $J_c = 9 \times 10^8 A/cm^2$ for skyrmion creation.

Beside from conventional approaches, skryrmions can be obtained also by converting a domain wall[1](**DW**). A study carried out by Zhou *et al.* [14] explains how this phenomena occurs. After a domain wall has been created inside a nanowire (Figure 1.3), an external electrical current is injected, causing the motion of the domain wall. As soon as the DW reaches the interface, both of its end points are pinned at the junction interface, whereas the central region of DW proceeds its motion. Eventually, a skrymion is obtained.

In [15] Iwasaki *et al.* demonstrated the creation of skyrmion in a stripline-shaped system with a square notch structure (Figure 1.4). Also in this method the spin transfer torque assume a key role: as soon as electrical current flows, the spin texture at the notch expands due to the STT. Afterwards, the presence of the DMI

[1]In magnetism, a domain wall is an interface separating magnetic domains. Their structure and dynamics depend on the local properties of the material [13].

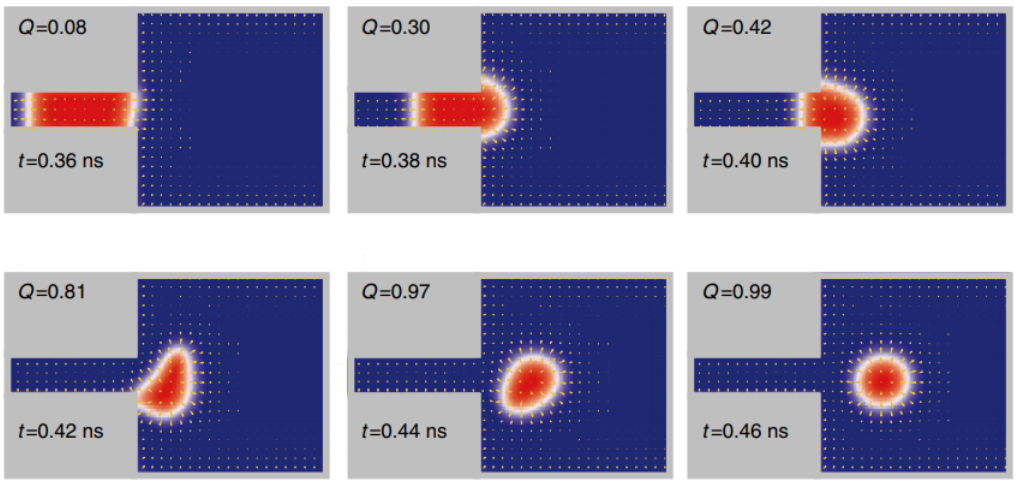


Figure 1.3: Snapshots of a domain wall conversion into a skyrmion. Figures extracted from [14].

cause the curve of the spins, creating the skyrmion. Iwasaki *et al.* studied this process by evaluting different parameters: the depth (d) and the width (w) of the notch and the angle (θ) of the notch corner. They found out that, if d is not large enough (relative to the skyrmion radius), only a small part of the skyrmion can be nucleated, meaning that the skyrmion fades away after a short time. However, if d is too large ($d-w$ very small), the nucleation cannot be occur. Regarding θ, several angles have been evaluated and it turned out that $\theta = 90°$ is the ideal angle.

Skyrmion annihilation can be achieved by exploiting similar method used for nucleation: by applying local heat, magnetic field or electrical current. In addition to these methods, skyrmion can be annihilated by simply pushing them to the boundary under a driving current [16].

1.3.2 Skyrmion detection

Skyrmion detection can be accomplished by taking into account the Hall effect (**THE**) or the magnetoresistance effect [17][18]. In [19], Crum *et al.* demonstrated a spin-averaged electrical detection mechanism for single skyrmions in a CPP-geometry[2]. In this experiment, two magnetic thin-film heterostructures have

[2]Current perpendicular-to-plane (CPP) giant magnetoresistance (GMR) effect is a resistance change that depends upon the relative angle of the magnetization vectors in magnetic layers separated by thin non-magnetic layers, which can be utilized for magnetic sensor applications [20].

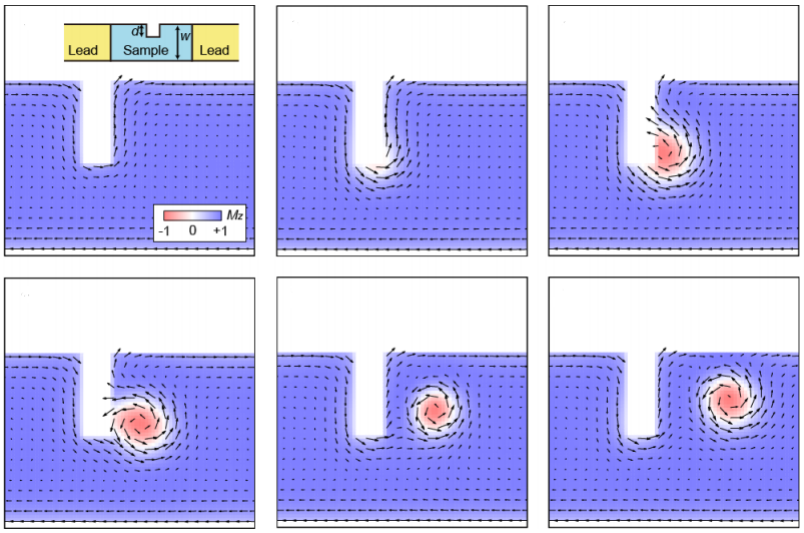


Figure 1.4: Simulation snapshots showing the nucleation of a skyrmion around a rectangular notch. Blue(red) represents the $+z(-z)$ component of the magnetization. Figures extracted from [15].

been considered (Figure 1.5), in particular fcc[3] overlayers of Pd/Fe and Pd/Pd/Fe on single crystal bulk fcc-Ir(111). The main reason why these materials has been choosen lays on the fact that they generate large Dzyaloshinskii Moriya interactions (DMI), which it has a key role in manipulating skyrmions.

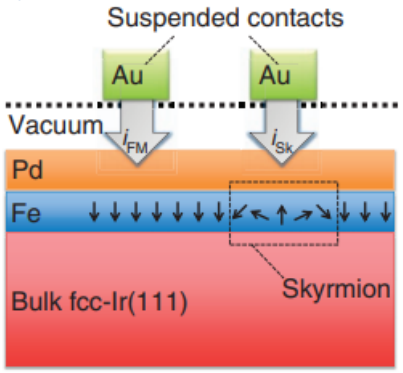


Figure 1.5: Illustrative heterostructure cross-section for the perpendicular reading of single nanoskyrmions. Figure extracted from [19].

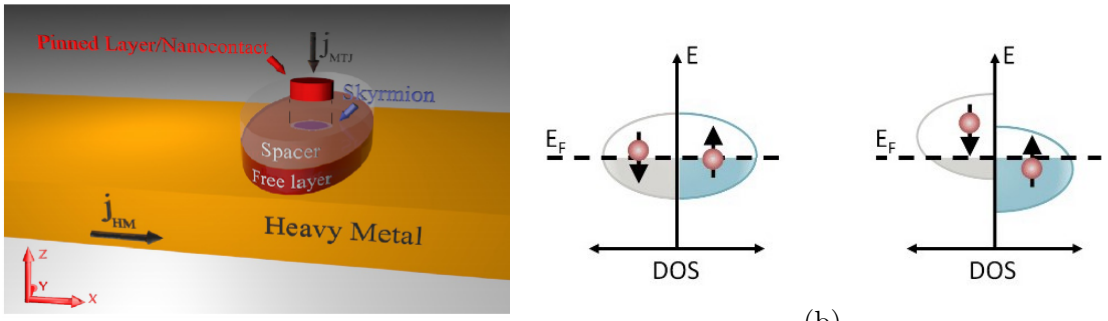
In another study [21], skyrmion detection has been performed by exploiting the

[3]Face-centered cubic lattice (fcc), has lattice points at the eight corners of the unit cell plus additional points at the centers of each face of the unit cell.

tunnel magnetoresistence effect at room temperature. The detection device consists of a heavy metal (HM) layer, used, as in the previous case, in order to achieve high values of DMI and spin hall effect (SHE). The ferromagnetic layers is used as point contact for depositing the MTJ (Figure 1.6a).

Spin-polarized current

Whenever an electrical current passes through a ferromagnetic layer with fixed magnetization, it gets spin-polarized. It can be observed that the density of states (DOS) of a ferromagnetic metal differs from the DOS of a normal metal. As Figure 1.6b shows, each spin has a different DOS, indeed, the spin-up band at the Fermi level is mostly filled, while there are many empty states available in the spin-down band. Therefore, the conduction electrons injected in the ferromagnet feel a different value of resistivity, according to their spin: the spin-down electrons have more states to scatter into, meaning that the resistivity is higher ($\rho_{\uparrow}$) than the one felt by spin up electrons ($\rho_{\downarrow}$).



(a) (b)

Figure 1.6: (a) Schematic of the device. (b) Schematic of the density of states at the Fermi level in a metal (left) and in a ferromagnetic metal (right). In the latter, the density of states at the Fermi level is unbalanced for spin-up and spin-down branches. Figure extracted from [22].

1.3.3 Skyrmion motion

Skyrmion can be moved by injecting an eletrical current through the spin Hall effect (SHE) or the spin transfer torque (STT). Sampaio *et al.* studied skyrmion motion in a nanotrack by taking into account the STT [10]. Firstly, spin-polarized currents

are injected in-plane with adiabatic and non-adibatic spin tranfer torque [4] causing skyrmion motion. As Figure 1.7 shows, initially, the skyrmion ($t < 1$ns) has a longitudinal speed (along x-direction) and a transverse speed in the $+y$-direction for $\beta > \alpha$ and the $-y$ direction for $\beta < \alpha$. It can be observed that the vertical motion along y stops after Δt, this phenomena occurs because of the repulsive interaction caused by DMI. Eventually, the skyrmion moves with an horizontal speed which is approximately equal to $u\beta/\alpha$, (u is proportional to the current density j).

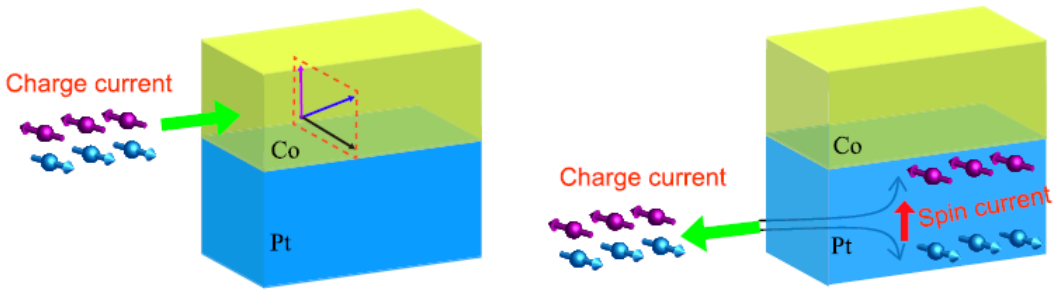
Figure 1.7: Skyrmion motion in a nanotrack. Figure extracted from [10].

As discussed previously, spin Hall effect can be used in order to move skyrmion. This phenomena originates in spin-Hall devices, where a ferromagnetic (FM) thin film is deposited above a heavy metal (HM) substrate. In these devices, electrical current is injected inside the HM layer. Due to spin-dependent scattering mechanisms, the electrons will be diverted perpendicularly to their direction and to the orientation of their spin. Consequently, an accumulation of charges region deploys at the sides of the wire due to the SHE, each side is populated by electrons with a well defined spin orientation. For istance, let us consider a current that flows in the $+x$-direction and that the velocity acquired by the electrons is directed along the $+z$-axis, charges accumulate at the top surface of the wire, making their spin orientation pointing along the $+y$-axis. In addition, this spin current flowing in the z-direction and polarized along the y-direction, can be collected by the FM thin film deposited above the HM substrate. This transverse spin current then will interact with the magnetization of the HM layer, again through the STT mechanism [23].

[4] Spin transfer torque is calculated by means of these formula, depending if it is an adiabatic or non-adiabiatic function:

$$\tau_{adiab} = u\mathbf{m} \times \left(\frac{\partial \mathbf{m}}{\partial x} \times \mathbf{m} \right), \quad \tau_{non-adiab} = \beta \mathbf{m} \times \left(\mathbf{m} \times \frac{\partial \mathbf{m}}{\partial x} \right) \tag{1.2}$$

β is the non-adiabatic parameter, whereas α is the Gilbert damping which is kept constant.



(a) (b)

Figure 1.8: Illustration of current-induced skyrmion motion along a nanotrack through either (a) STT or (b) SHE, respectively. Figure extracted from [7].

1.4 Advanced functionality of skyrmion

Thanks to their properties (low depinning current density, small size, low power dissipation), skrymions can be manipulated in order to accomplish different tasks such as duplication, merge or logic functions. In particular, a study carried out by Zhang *et al.* [6] illustrates how skyrmion can be duplicated and merged (Figure 1.9). Firstly, skrymion is nucleated in a Y-shaped junction and, after applying an electrical current, is converted into a domain-wall pair. As soon as the DW reaches the central region of the Y-shaped junction, the domain-wall pair is split into two domain wallpairs. Afterwards, they are converted back into two skyrmions, leading to the duplication of the skyrmion. The information is stored by the position and timing of a skyrmion. During this process, skyrmion quantum numbers (Q_s, Q_v, Q_h) change: (1, 1, 0) $\rightarrow$ (0, 0, 0) $\rightarrow$ (2, 1, 0). In a similar way, skyrmion can be merged by converting in DW pair, and then converting back.

Duplication and merge operations can be used in order to implement basic logic function, such as AND and OR gates (Figure 1.10). First of all, it is important to highlight that in skrymion logic, the "presence" of the skyrmion corresponds to 1 logic, whereas the "absence" of the skyrmion corresponds to 0 logic. Regarding the OR gate, it can be implemented by using a merge structure, indeed, whenever one of the input has a skrymion ($A = 1$, $B = 0$ or $A = 0$, $B = 1$), simply this skyrmion, after been converted into DW, moves inside the Y-shaped junction and then, after been converted back, reaches the output. If there is a skyrmion in both input ($A = 1$, $B = 1$), these are merged in one skyrmion. The AND gate structure differs from previous one: the output nanowire in the Y-shaped junction is slightly wider. This small change leads to an huge difference, indeed, whenever there is only one skyrmion in the input nanowire ($A = 1$, $B = 0$ or $A = 0$, $B = 1$), the converted DW scatters towards the edge of junction, until it annihilates. Therefore,

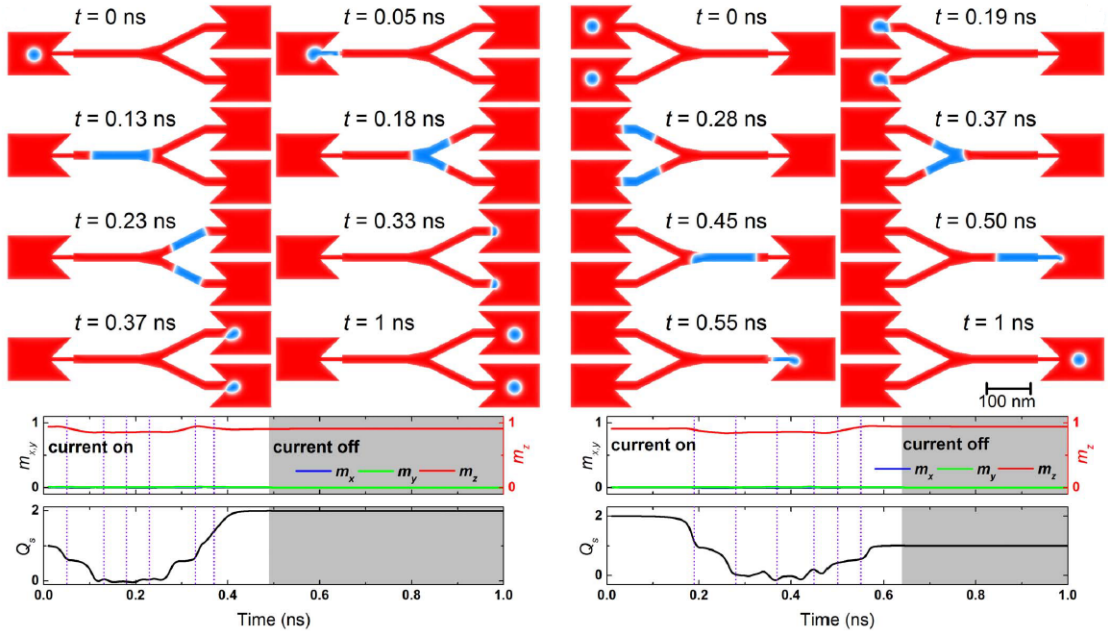


Figure 1.9: The top figures show the magnetization configuration at eight different times; the middle figures show the time evolution of the average spin components m_x, m_y, m_z; the bottom figures show the time evolution of the skyrmion number Q_s. Duplication snapshots are on the left, merging on the right.

only when there are two skyrmions in the input nanowires ($A = 1$, $B = 1$), the DW can overcome the Y-shaped junction and converted back to skrymion. Trivial is the case in which there is no input, obviously there is no output.

In [24] a different approach has been followed in order to implement skyrmion-based logic gates. The authors of this work define the system, that can be obtained by using these gate, "conservative", meaning that skyrmion are not lost after the operation ends. This is huge advantage because there is no need to nucleate a new skyrmion whenever another operation is performed, leading to less power consume. In addition, thanks to this property, it is possible to link different basic gates in order to get a complex logic function. Two different conservative basic logic function has been proposed: AND/OR and INV/COPY. Since this function is conservative, the total number of skyrmions N provided to the inputs is the same at the outputs. This conservative logic function is performed by the structure shown in 1.11, with micromagnetic simulations. Skyrmions are pushed towards the $+y$-direction due to the spin-Hall effect, at the same time, the skyrmion-Hall effect generates a x-directed force which is opposed by repulsion from the track boundaries. However, as soon as the skyrmions reaches the central junction, they are free to move, therefore,

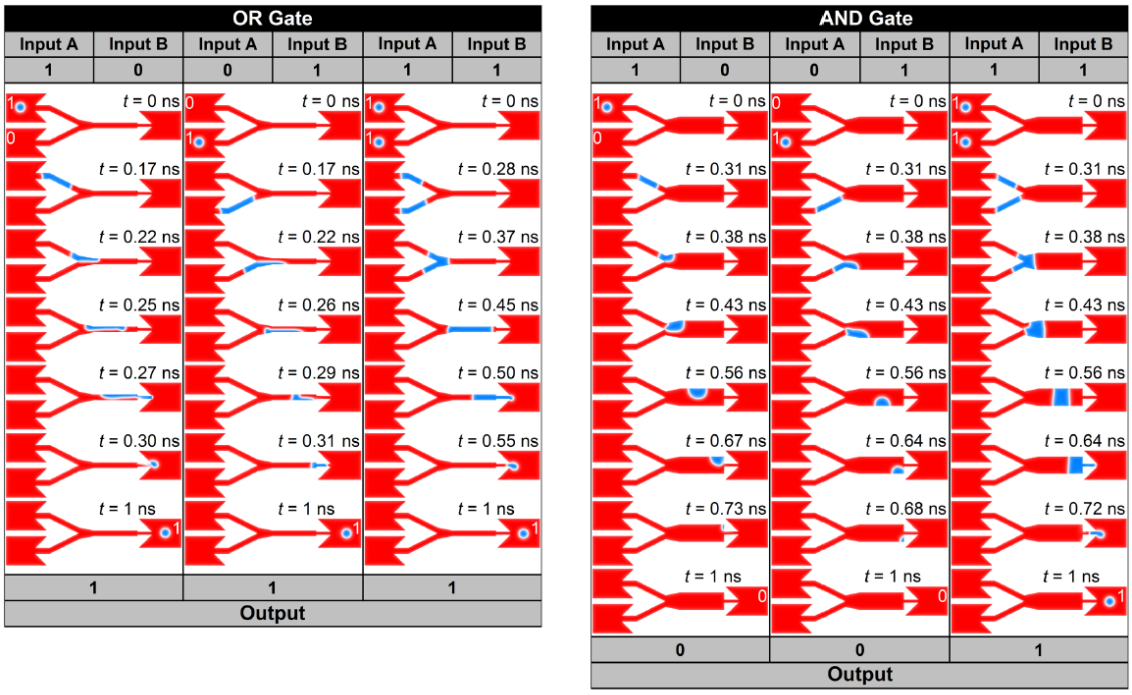


Figure 1.10: Skyrmion-based OR and AND gate.

the skyrmion-Hall effect causes leftward skyrmion propagation unless repulsed by a second skyrmion. This phenomena occurs when $A = 0$, $B = 1$, for this reason the left output nanotrack represents the OR function, whereas the right one is the AND function. Interesting is the case in which there are two skyrmions as input: the skyrmion on the right track cannot change direction due to the repulsion skyrmion-skyrmion. INV/COPY gate provides the same result of a basic NOT gate (Figure 1.12). In other words, this structure has two inputs: IN, which is the input that needs to be negated, and $CTRL$, whose aim is to make the gate working by introducing skyrmion-skryrmion repulsion. In particular, the NOT operation takes place when $CTRL = 1$, indeed, when $IN = 0$, the skryrmion on the right track changes direction as soon as the hole is reached due to skrymion hall effect, carrying out a skymion at the NOT output. Instead, when $IN = 1$, the skyrmion-skyrmion repulsion acts, meaning that only the skyrmion in the middle track turns on the left. Moreover, this gate provides an additional functionality: the left (COPY1) and the right (COPY2) tracks duplicates the IN skyrmion.

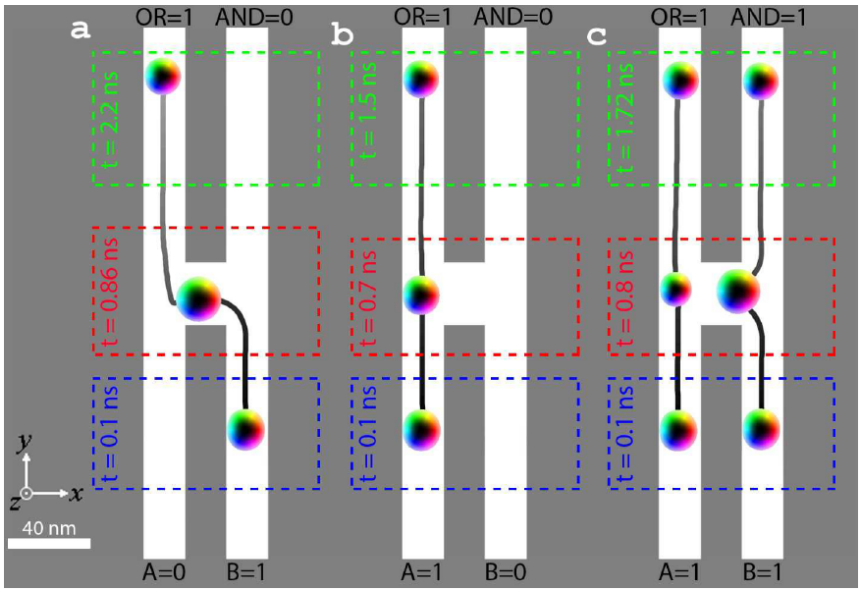


Figure 1.11: Micromagnetic simulation of conservative AND/OR logic gate with input combinations (a) $A = 0$, $B = 1$; (b) $A = 1$, $B = 0$; and (c) $A = B = 1$. Figure extracted from [24].

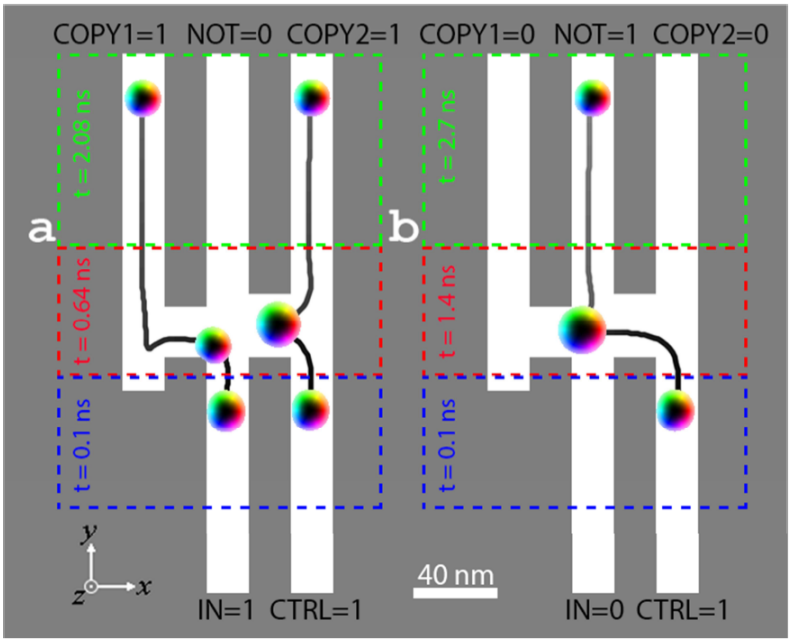


Figure 1.12: Micromagnetic simulation of conservative INV/COPY logic gate with input combinations (a) $IN = 1$, $CTRL = 1$; (b) $IN = 0$, $CTRL = 1$. Figure extracted from [24].

In addition to these two logic gates, a synchronization mechanism has been proposed (Figure 1.13). This element behaves differently as function of the electrical current injected: whenever this value is under a threshold, the skyrmion gets stuck in this structure indefinitely; whenever the current is great enough, the skyrmion size gets smaller, letting it to overcome the notch structure.

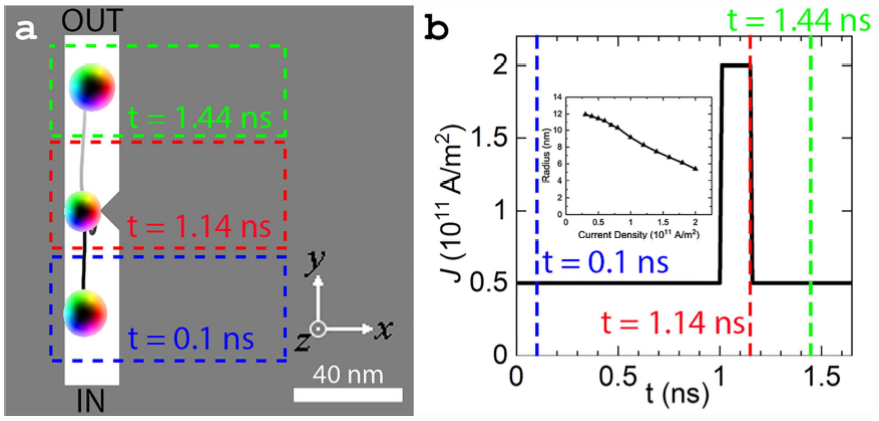


Figure 1.13: Micromagnetic simulation of conservative (a) notch element and (b) the electrical current injected as function of time. Figure extracted from [24].

Thanks to these three components (AND/OR, INV/COPY gates and the notch element), more complex function logic can be implemented. For example, a full-adder structure can be created, as Figure 1.14 shows. As it can be seen, a XOR function can be implemented by taking into account two INV/COPY gates and then merging the outputs. The notch element allows to synchronize skyrmions that come out from different gates.

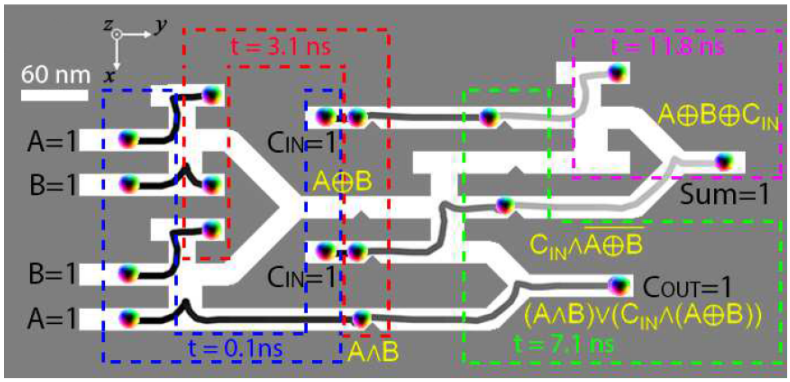


Figure 1.14: Micromagnetic simulation of conservative full adder. Figure extracted from [24].

Chapter 2

Logic-in-Memory architecture

Nowadays, most of all computing systems are based on the *von Neumann* paradigm. This paradigm is based on the exchange of data between a Central Processing Unit (CPU) and a memory. In particular, the CPU performs operations, and then results are stored back in the memory. This data-exchanging mechanism is influenced by the speed of the CPU and memory. However, in the last decades, CPU performances increased much more than memory ones, leading to a so-called "bottleneck": CPU is limited by memory speed [25]. Especially, in data-intensive algorithm, this limitation has an huge impact on power consumption due to a large quantity of memory access. As a consequence of *von Neumann* paradigm drawbacks, alternative architecture have been studied. In particular, the approach **Logic-in-Memory** (LiM) attempts to overcome those limitations. Its key benefits are mainly: (1) Bringing the computation directly inside the memory, solving the memory wall problem; (2) data are computed directly inside the memory without the need to move them between the computing and the storage units, drastically reducing the amount of memory accesses and, therefore, the overall power consumption.

2.1 State of art

In-memory architecture can be classified in four different categories [26]:

(A) Computation-near-Memory. Although logic and storage are brought closer, they are still two separate entities. Interaction between them is possible thanks to 3D stacked integration technologies [27]. This approach benefits both length of interconnections, by reducing them, and memory bandwidth.

(B) Computation-in-Memory. The structure of the memory array is not modified, while its intrinsic analog functionality is exploited to perform computation. In particular, in-memory computation is achieved by reading data from the

memory which is then sensed by sense amplifiers (SAs). The result is then written back in the memory array [28].

(C) Computation-with-Memory. In this kind of approach, the memory is seen as a Content Addressable Memory (CAM), that fetches data from a look-up-table (LUT) [29].

(D) Logic-in-Memory. Data are processed inside the memory, allowing to exploit full bandwidth. As a consequence, memory access are less frequent than with other architecture, meaning that there is less power consumption [30].

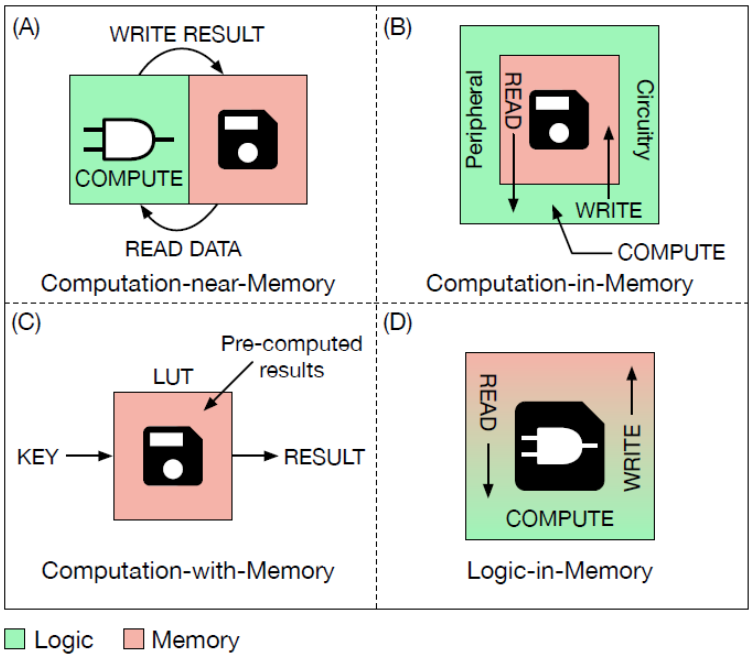


Figure 2.1: Main in-memory architectures. Figure extracted from [26].

2.2 Technologies supporting LiM architecture

Several studies show possible implemention for LiM paradigm, by exploiting different technologies. In [31] perpendicular NanoMagnetic Logic (pNML) has been taken into account. In this technology, magnets with perpendicular magnetic anisotropy (PMA) are exploited [32]. Binary information is encoded by considering two stable magnetization states. The logic 0 is represented by the magnetization up, whereas the logic 1 by the down magnetization. In order to create a nucleation point, focused ion beam (FIB) is used, this tecnique irradiates on a single spot, reducing

the magnetic anisotropy [33] and, consequentally, making the magnet more sensible to external magnetic fields. The magnetization state of a magnet depends on the ferromagnetic (F) and/or anti-ferromagnetic (AF) coupling, that it is influced by the position of neighbour magnets. Nevertheless, since coupling fields are not strong enough, it is important to guarantee the presence of an external field (clock) that let possible the information propagation.

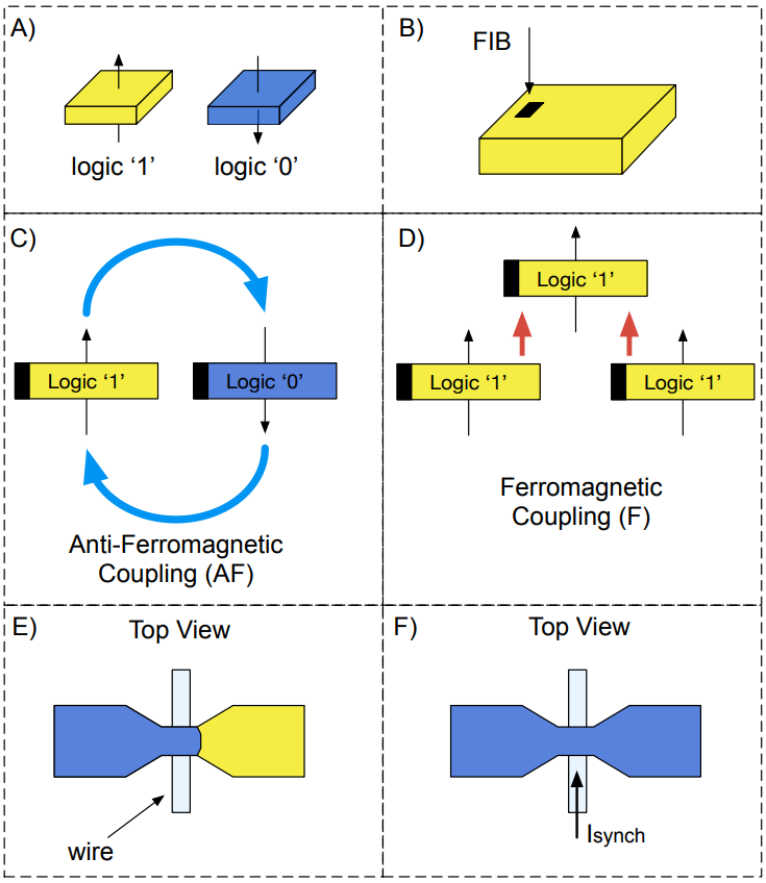


Figure 2.2: (a) pNML elementary cells; (b) Nucleation by FIB irradiation; (c) Anti-ferromagnetic coupling between adjacent magnets lying in the same plane; (d) Ferromagnetic coupling between magnets lying above each other in different layers; (e) DWs pinned within a notch; (f) Energy barrier reduced by the in-plane field. Figure extracted from [31].

Thanks to the monolithic 3D integration of pNML, logic-in-memory architecture can be easily implemented, in a similar way as systolic architectures [34]. Figure 2.3 shows an implementation of a 2-bit accumulator, by applying pNML technology.

Matsunaga *et al.* proposed a logic-in-memory architecture based on magnet tunnel junction (MTJ) in combination with MOS transistors [35]. Thanks to its

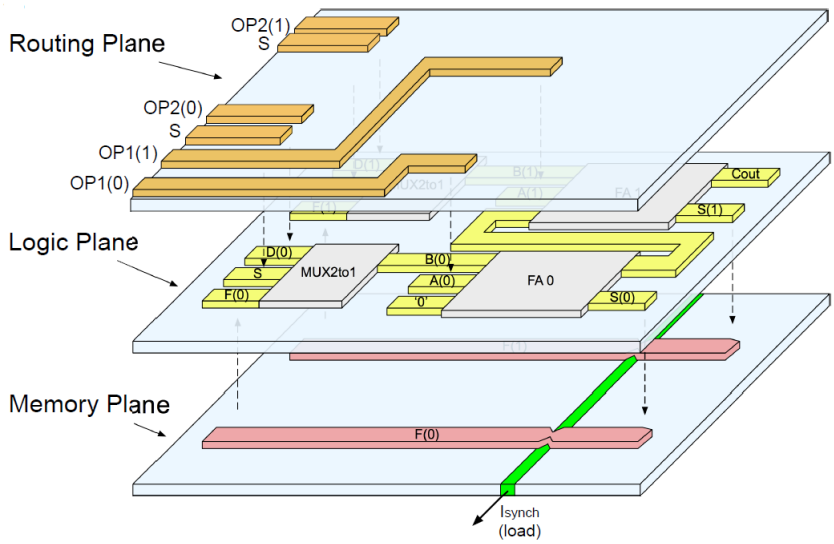


Figure 2.3: A 2-bit accumulator exploiting the monolithic 3D integration of pNML technology. Figure extracted from [31].

properties, such as low access time and small dimensions compared to CMOS technology, MTJ is able to fully take advantage of the logic-in-memory architecture.

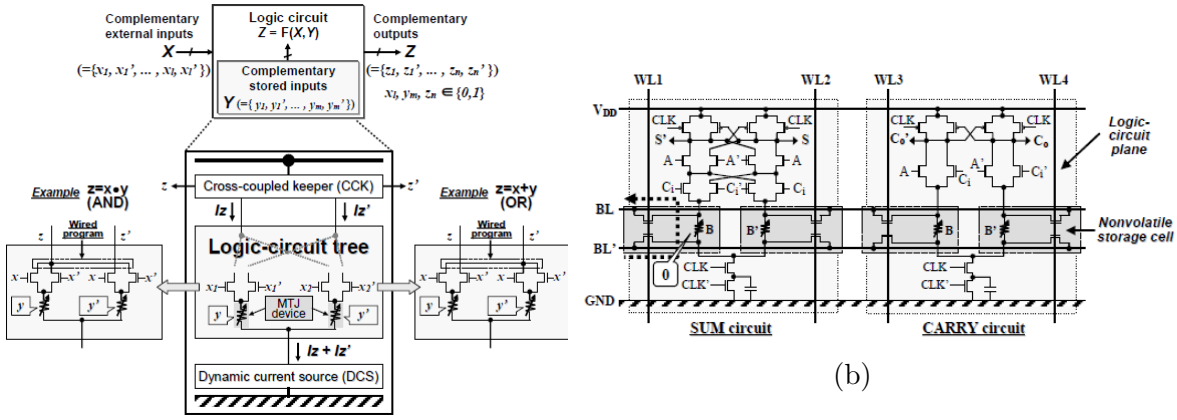


Figure 2.4: (a) General structure of an MTJ-based logic-in-memory circuit. (b) Overall circuit structure of the full adder with nonvolatile stored inputs. Figure extracted from [35].

In Figure 2.4a, it can be seen the general structure of this logic-in-memory circuit. It is possible to pinpoint three basic components: cross-coupled keeper (CCK) whose

aim is to provides binary outputs, z and z' (these values depends on signal current I_z and I'_z), a dynamic current source (DCS), that cuts off steady current from V_{DD} to GND, and logic-circuit tree, that can be customized in order to get the wanted logic function. Figure 2.4b shows a possible application of this architecture: a full adder. Dynamic power dissipation is reduced to 23% then CMOS-based full Adder, thanks to the fact that the present circuit structure makes it possible to reduce the number of current paths from V_{DD} to GND. The proposed nonvolatile logic-in-memory circuit makes it possible not only to eliminate the static power consumption but also to reduce the chip area. In the nonvolatile logic-in-memory circuit, write time of an MTJ is one of the most important elements, because it also dominates the write energy when updating stored inputs (Figure 2.5).

	CMOS	Proposed
Delay	224 ps	**219 ps**
Dynamic power (@500MHz)	71.1 µW	**16.3 µW**
Write time	2 ns/bit	**10 ns/bit (2 ns/bit)** *1)
Write energy	4 pJ/bit	**20.9 pJ/bit (6.8 pJ/bit)** *1)
Static power *2)	0.9 nW	**0.0 nW**
Area (Device counts) *3)	333 µm^2 (42 MOSs)	**315 µm^2** (34 MOSs + 4 MTJs)

Figure 2.5: Full-adder specs, CMOS-based vs. MTJ-based architecture. Figure extracted from [35].

2.3 Configurable logic-in-memory architecture

In [26], an interesting logic-in-memory approach has been proposed. In particular, this architecture is a sort of mix of all in-memory computing (Section 2.1). This increases the level of flexibility since the architecture can adapt to the algorithm that needs to be implemented. Indeed, as Figure 2.6 illustrates, operations that can be performed in-memory are sent to CLiM array. On the other hand, whenever an operation cannot be execute in-memory, data are elaborated by computing-near-memory unit. In addition, some extra-array (row and column) logic are available in case cells need to exchange data with neighbours. The main property of a CLiM cell is being able to configure itself to perform different types of operations. Beside local data computation inside each cell, CLiM cells are interconnected between them:

- Intra-row operation among cells that belongs to the same row;
- Intra-column operation among cells that belongs to the same column;

- Inter-row operation among cells that belongs to different rows;
- Inter-column operation among cells that belongs to different columns.

An example of how these interconnections can be exploited is the computation of an array multiplier (AM). This logic block can be built by connecting two ripple carry adder (RCA), different cells can be connected together to propagate the carry, which, in turn, can be implemented by full adders. In other words, each CLiMa cell implements a single full adder, and by exploiting intra-row operation, RCA can be built. Eventually, AM is built by carrying out inter-row computation.

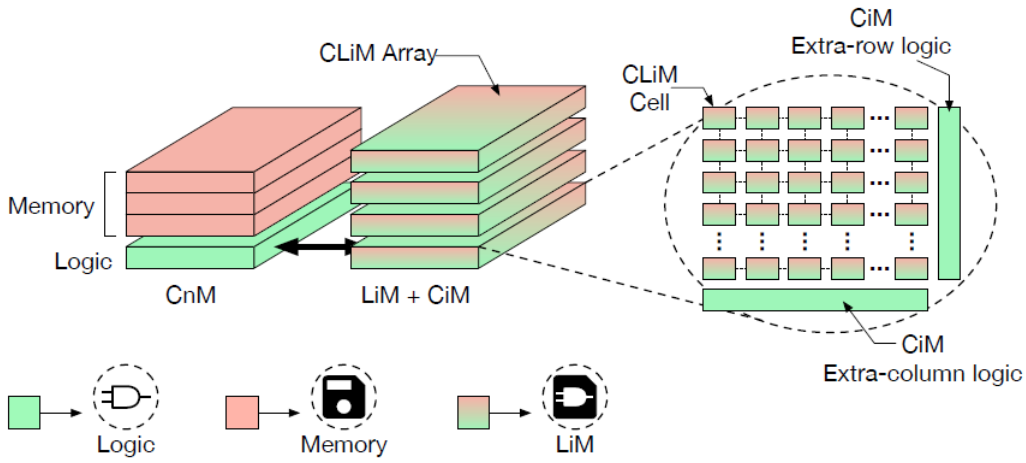


Figure 2.6: CLiMa structure. Figure extracted from [26].

Chapter 3

Advanced Encryption Standard algorithm

The Advanced Encryption Standard (AES) is a block cipher[1] algorithm adopted as standard for the encryption of data by the United States of America. It was developed by two belgian cryptographers, Joan Daemen e Vincent Rijmen, who won the AES selection process. Thanks to its speed, AES can be implemented in both software and hardware, providing a good level of protection and security.

3.1 Mathematical background

In AES algorithm data are split in blocks of byte, each bit of every byte represents the coefficient of a polynomial expression of the finite field $GF(2^8)$, called *Galois field* [37]:

$$\{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0\} \Rightarrow b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$$

For example, let us consider the conversion of an hexadecimal-base number in $GF(2^8)$:

$$(49)_{16} = 4 \cdot 16 + 9 \cdot 16^0 = (73)_{10} = (1001001)_2 = (x^7 + x^4 + x^1)_{GF(2^8)}$$

The addition of two $GF(2^8)$ elements can be carried out by performing a sum mod 2 operation for each bit:

[1] A block cipher is a deterministic algorithm in which data are fixed-length groups of bits, these blocks are, then, transformed by taking into account a key [36].

- $GF(2^8)$: $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = (x^7 + x^6 + x^4 + x^2)$
- Binary: $(01010111)_2 \oplus (10000011)_2 = (11010100)_2$
- Hexadecimal: $(57)_{16} \oplus (83)_{16} = (D4)_{16}$

The symbol $\oplus$ represents the XOR operation, that provides the same output of mod 2 sum. In $GF(2^8)$, $A(x) \cdot B(x)$ can be performed by multplying polynomials followed by a modular reduction with an irreducible eighth degree polynomial $m(x) = x^8 + x^4 + x^2 + x + 1$ [37]. Let us consider an example in order to understand better this operation. In particular, let us suppose to evaluate the multiplication of $f(x)g(x)$ where $f(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$ and $g(x) = x$:

$$x \times f(x) = (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \bmod m(x)$$

If $b_7 = 0$, the result is a polynomial of degree less than 8, which is already in reduced form, meaning that no further operations are needed. However, if $b_7 = 1$, then reduction modulo $m(x)$ needs to be performed by taking into account this equation:

$$x^8 \bmod m(x) = \left[m(x) - x^8\right] = (x^4 + x^3 + x + 1)$$

To sum up, multiplication by x – that has an huge importance in AES mix columns operation – can be carried out as 1-bit left shift and, in case of $b_7 = 1$, followed by a conditional bitwise XOR with (00011011), which corresponds to $(x^4 + x^3 + x + 1)$:

$$x \times f(x) = \begin{cases} b_6b_5b_4b_3b_2b_10 & \text{if } b_7 = 0, \\ b_6b_5b_4b_3b_2b_10 \oplus 00011011 & \text{if } b_7 = 1. \end{cases}$$

3.2 Algorithm description

In AES algorithm, the plaintext is divided in fixed length block of 128 bits, whereas the key can assume three different values: 128, 192 or 256 (Table ??). AES operates on a 4×4 matrix, defined as **state**, which is modified at every stage of encryption and decryption. After the last stage is performed, chipertext is copied to an output matrix. The key size used for an AES cipher specifies the number of transformation rounds. An AES round carries out four different operations. In particular:

- SubBytes: each byte is replaced by examining a S-box;
- ShiftRow: bytes in the same row rotates;
- MixColumns: $GF(2^8)$ multiplication are performed;
- AddRoundKey: each bytes executes a bitwise XOR with the key;

Key size	4/16/128	6/24/192	8/32/256
Plaintext block size	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size	4/16/128	4/16/128	4/16/128
Expanded key size	44/176	52/208	60/240

Table 3.1: AES parameters, data are expressed in words, bytes and bits.

As Figure 3.1 shows, when the algorithm starts, an additional AddRoundKey operation is executed; the final round does not performs a MixColumns transformation.

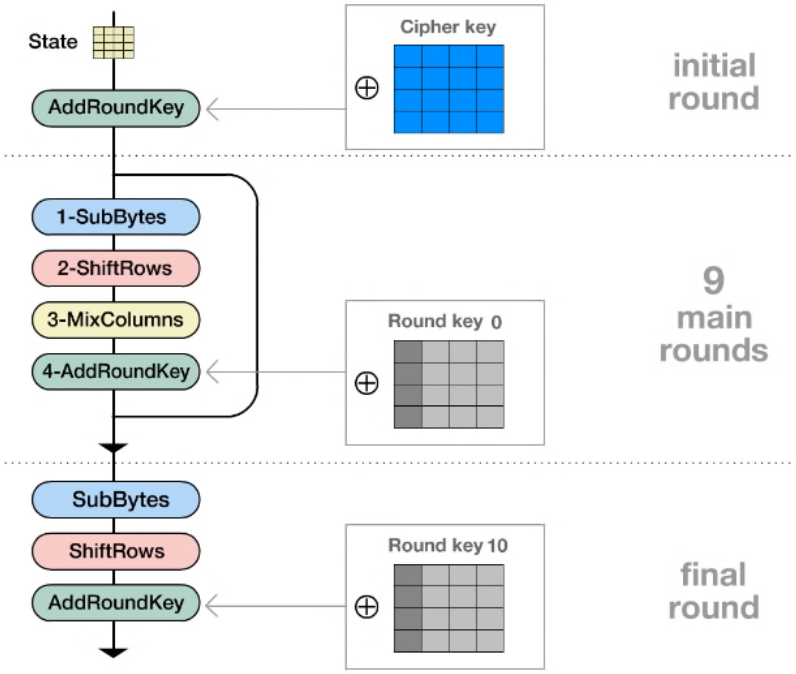


Figure 3.1: AES rounds.

3.2.1 SubBytes

In this operation, each byte is substituted by means of a S-box. In particular, each word is split in two groups of four bits, the most significant group selects the row of the S-box, whereas the least significant one selects the column. The element that replaces the initial value belongs to the row and column choosen.

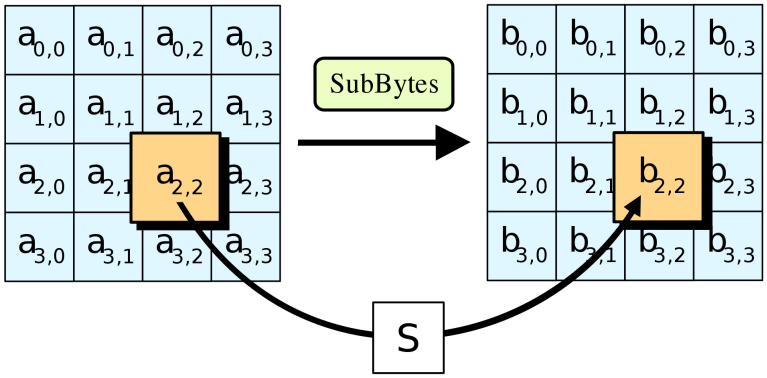


Figure 3.2: SubBytes operation.

3.2.2 ShiftRow

ShifRow trasformation involves the shift of each row by a number of position. In particular, no shift is performed by the first row, one shift for the second one, two for the third one and, eventually, three for the fourth one.

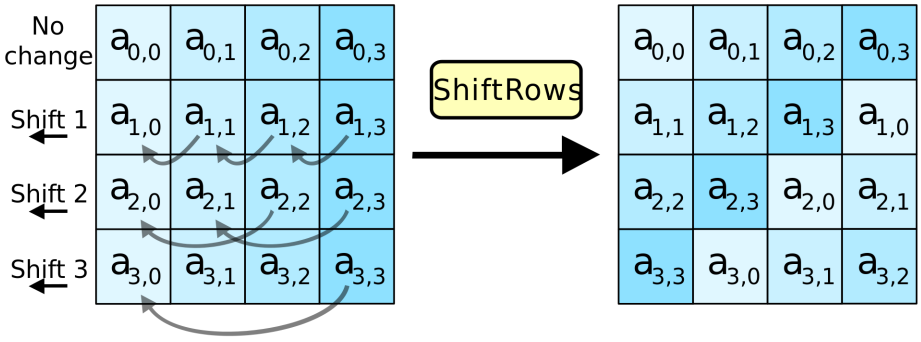


Figure 3.3: ShiftRow operation.

3.2.3 MixColumns

In this operation, each column undergoes to a invertible linear transformation. In particular, a multiplication with a constant matrix occurs, whose coefficient are determined by fixed polynomial $c(x) = 3x^3 + x^2 + x + 2$.

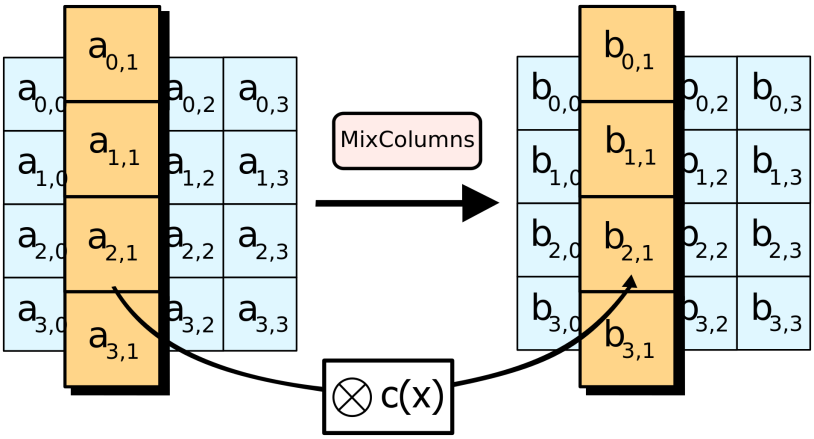


Figure 3.4: MixColumns operation.

3.2.4 AddRoundKey

Each element of the state matrix is XORed with each element of the key, whose values differs from each round.

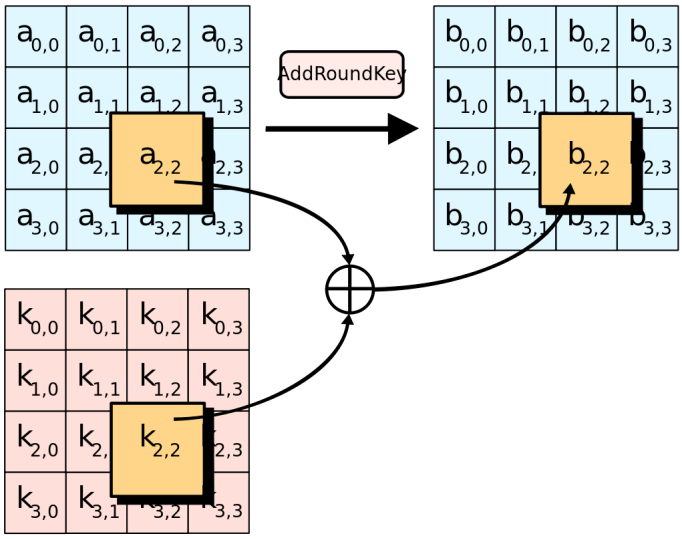


Figure 3.5: AddRoundKey operation.

3.2.5 Key scheduler

At each round, the key undergoes to several transformation (Figure 3.6). In particular, each element of the last column ($k^0_{i,3}$) is rotated of one position:

$$[k^0_{0,3}\ k^0_{1,3}\ k^0_{2,3}\ k^0_{3,3}] \Rightarrow [k^0_{1,3}\ k^0_{2,3}\ k^0_{3,3}\ k^0_{0,3}]$$

Afterwards, each words is replaced by means of the S-box:

$$[k^0_{1,3}\ k^0_{2,3}\ k^0_{3,3}\ k^0_{0,3}] \Rightarrow [k^0_{s(1,3)}\ k'_{s(2,3)}\ k'_{s(3,3)}\ k'_{s(0,3)}]$$

Next step involves the use of constant matrix 4×10:

$$RCON = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1b & 36 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{bmatrix}$$

The S-boxed elements are XORed with one of the column of $RCON$, this one is choosen in function of the round (i.e. round $1 \rightarrow Rcon[0]$):

$$\begin{bmatrix} k^0_{s(1,3)} \oplus Rcon[0,i] \\ k^0_{s(2,3)} \oplus Rcon[1,i] \\ k^0_{s(3,3)} \oplus Rcon[2,i] \\ k^0_{s(0,3)} \oplus Rcon[3,i] \end{bmatrix} = \begin{bmatrix} k^0_{r(1,3)} \\ k^0_{r(2,3)} \\ k^0_{r(3,3)} \\ k^0_{r(0,3)} \end{bmatrix}$$

Then, the result is XORed again with the first column of the key:

$$\begin{bmatrix} k^0_{r(1,3)} \oplus k^0_{0,3} \\ k^0_{r(2,3)} \oplus k^0_{1,3} \\ k^0_{r(3,3)} \oplus k^0_{2,3} \\ k^0_{r(0,3)} \oplus k^0_{3,3} \end{bmatrix} = \begin{bmatrix} k^1_{0,0} \\ k^1_{1,0} \\ k^1_{2,0} \\ k^1_{3,0} \end{bmatrix}$$

Eventually, the columns of the original key left are XORed with the columns of the new key:

$$\begin{bmatrix} k^1_{0,0} \oplus k^0_{0,1} \\ k^1_{1,0} \oplus k^0_{1,1} \\ k^1_{2,0} \oplus k^0_{2,1} \\ k^1_{3,0} \oplus k^0_{3,1} \end{bmatrix} = \begin{bmatrix} k^1_{0,1} \\ k^1_{1,1} \\ k^1_{2,1} \\ k^1_{3,1} \end{bmatrix}, \begin{bmatrix} k^1_{0,1} \oplus k^0_{0,2} \\ k^1_{1,1} \oplus k^0_{1,2} \\ k^1_{2,1} \oplus k^0_{2,2} \\ k^1_{3,1} \oplus k^0_{3,2} \end{bmatrix} = \begin{bmatrix} k^1_{0,2} \\ k^1_{1,2} \\ k^1_{2,2} \\ k^1_{3,2} \end{bmatrix}, \begin{bmatrix} k^1_{0,2} \oplus k^0_{0,3} \\ k^1_{1,2} \oplus k^0_{1,3} \\ k^1_{2,2} \oplus k^0_{2,3} \\ k^1_{3,2} \oplus k^0_{3,3} \end{bmatrix} = \begin{bmatrix} k^1_{0,3} \\ k^1_{1,3} \\ k^1_{2,3} \\ k^1_{3,3} \end{bmatrix}$$

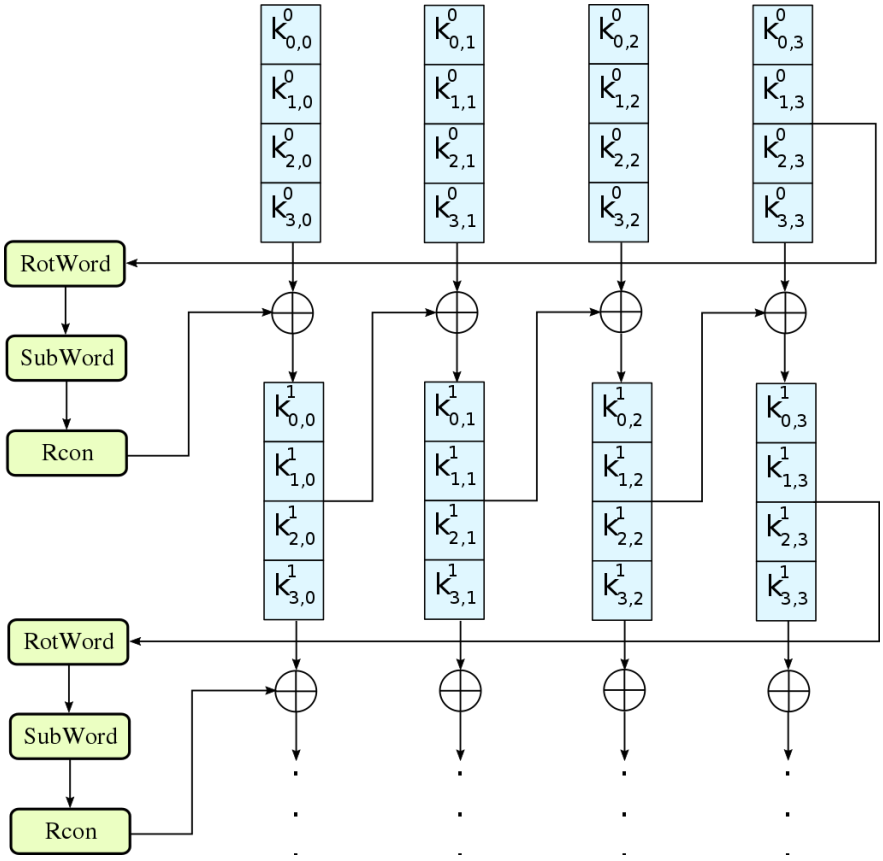


Figure 3.6: Key scheduler schematics.

Chapter 4

AES algorithm based on magnetic skyrmion

4.1 Memory array structure

The architecture based on skyrmions that has been developed is illustrated in Figure 4.16. In particular, it is structured as a matrix 4×4, in which each element represents a single word. Thanks to the fact that skyrmions are already non-volatile information carriers, data memorization occurs in an easier way than traditional CMOS-based architecture. Indeed, it is enough to nucleate them inside a nanotrack for having a storing of information: it is not necessary to move them anywhere else. In other words, whenever a bit needs to stored, the control unit sends a signal to a write head, placed at the beginning of the racetrack, that nucleates the skyrmion and shift it.

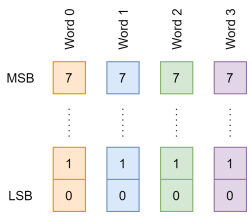


Figure 4.1: Words organization.

Therefore, since there are in total 16 words, a racetrack is present in each of them, leading to 16 nanotrack totally. During the design process, it has been decided to position the write head of each racetrack on the bottom of the word. This has been done in order to exploit better skyrmions properties. In addition, the MSB occupies the top cell (each word is composed of 8 cells, since it represents a byte), therefore,

the first skyrmion to be nucleated is the MSB (Figure 4.1). This action is repeated until the LSB is initialized.

4.2 Writing process

The writing process requires the use of a shift register. Input data is, firstly, saved in this block, and, then, by enabling the control signal CTRL_WORD_SHIFT, each bit comes out from the register. The output controls the write head MTJ_W.

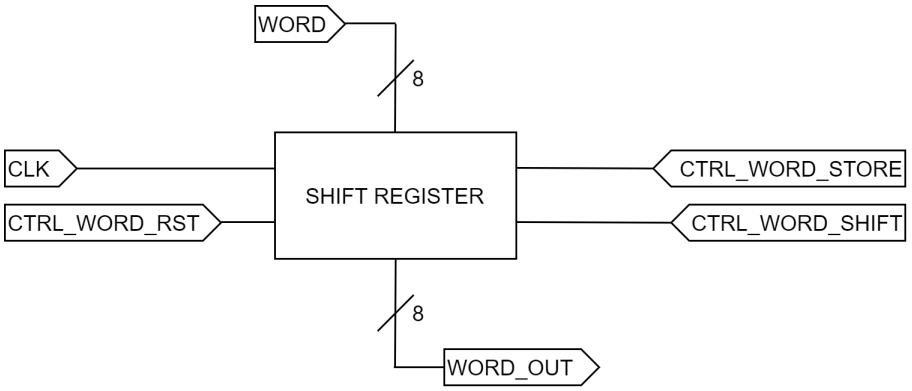


Figure 4.2: Words organization.

In particular, Figure 4.3 shows how each cells interface with the racetrack. After being nucleated, CTRL_V_{bl} is asserted, leading to skyrmion movement. This process is repeated until all skyrmion have been written. Eventually, as soon as all of them reach the first lateral track labeled V_{mov}, the control unit stops the injection of current V_{bl}. As a consequence, skyrmions stand still until another operation begins.

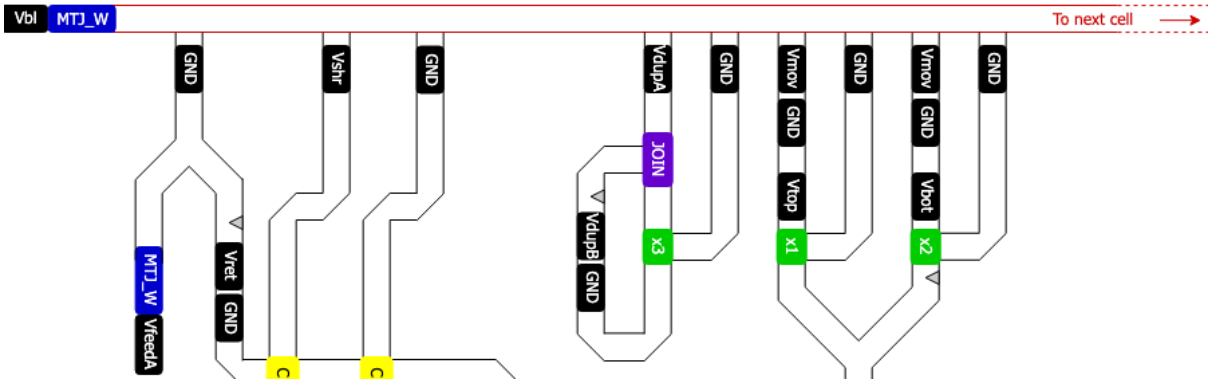


Figure 4.3: Writing process structure.

Deviation and lateral input block

Before explaining next process steps, it is important to describe how skyrmions can be deflected and which kind of physical structure allows this operation. Experiments have been carried out by exploiting a micromagnetic simulator called *Mumax*: a GPU-accelerated simulation software that, given a structure and given a subdivision of this structure into cells (Figure 4.4), solves the LLG equation in each cell providing, among the possible outputs, the time evolution of the magnetization. In writing the code to be simulated is necessary to describe the structure and the initial magnetization state. In each cell a different value of current is injected.

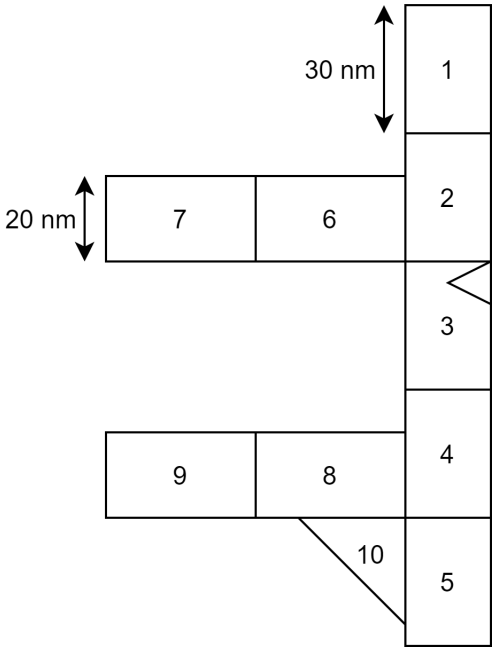
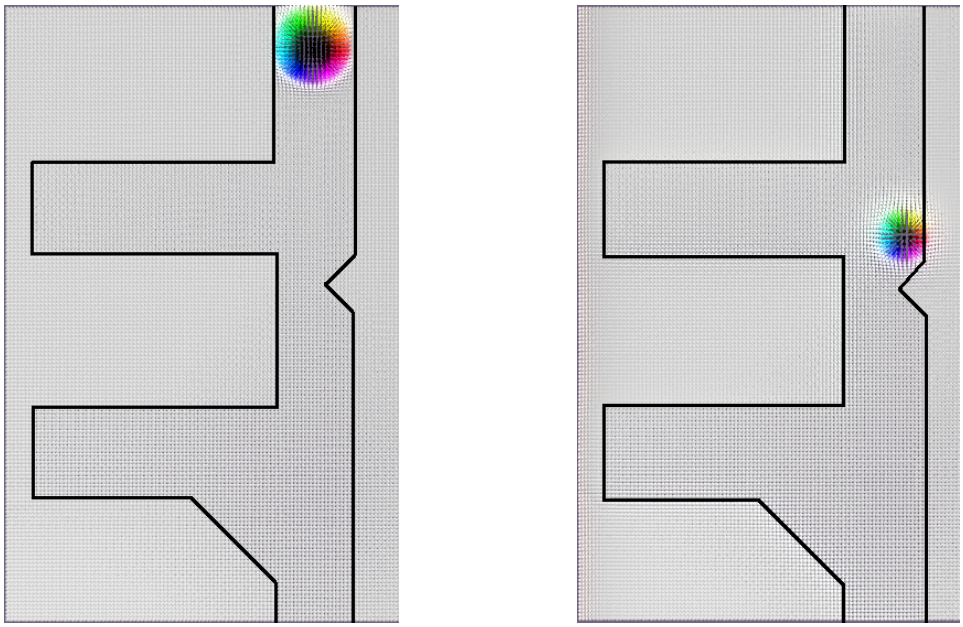
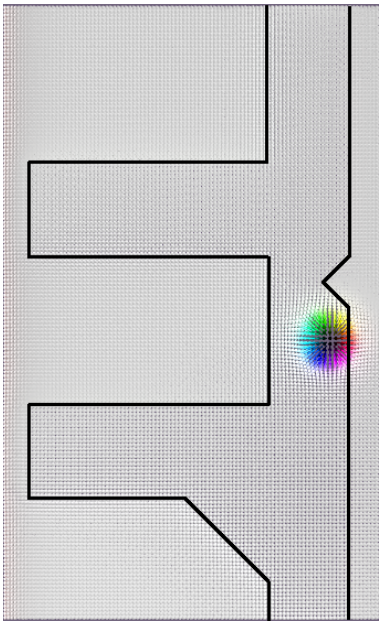


Figure 4.4: Nanotracks splitted in cells.

In this structure, skyrmion is, firstly, pushed down by applying a current in the racetrack whose direction is toward the bottom. If this is not enough, skyrmion gets stuck due to the notch. In case of deviation, another current is injected in the lateral track, pushing the skyrmion to the left (Figure 4.5). On the other hand, in case of skyrmion going straight, the current density in the racetrack needs to have a high value ($2 \times 10^{11} \mathrm{A/m^2}$), in order to overcome the notch (Figure 4.6). Moreover, another lateral track has been added for the sake of studying skyrmion behaviour in case of lateral input. Whenever skyrmion reaches the end of the lateral input track, it gets driven down by the current of the racetrack, independently from its value (Figure 4.7).

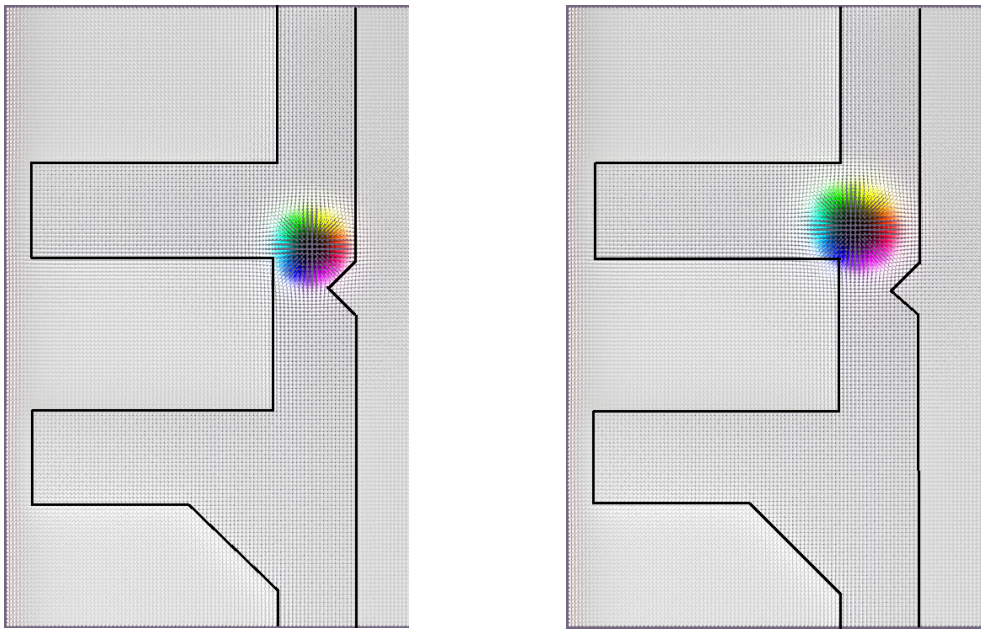


(a) (b)

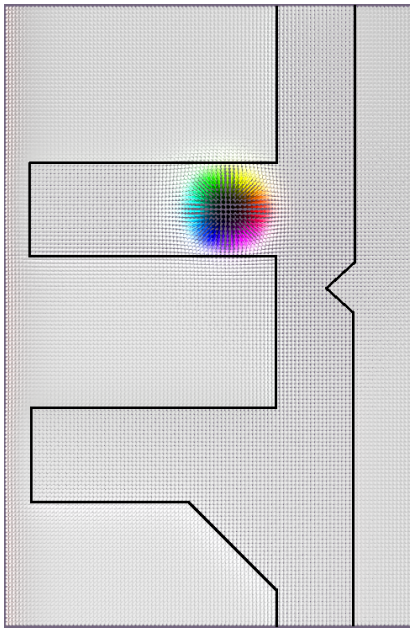


(c)

Figure 4.5: *Mumax* simulation with $J_{racetrack} = 2 \times 10^{11}\mathrm{A/m^2}$ and $J_{lateralTrack} = 0\mathrm{A/m^2}$. Skyrmion does not change track and overcome the notch.

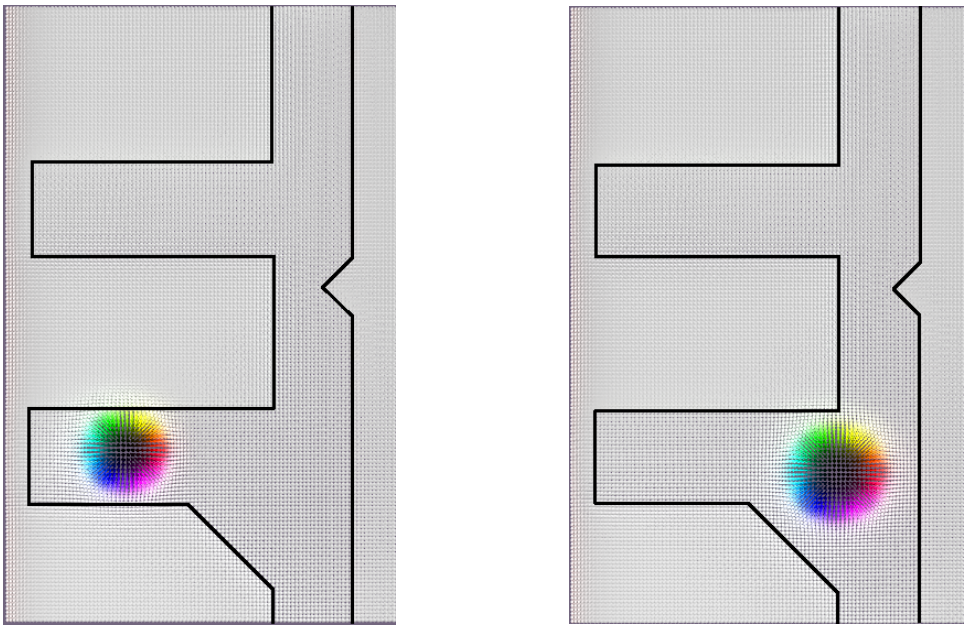


(a) (b)

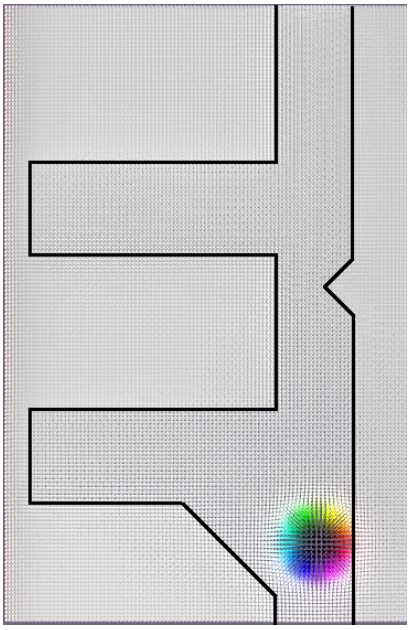


(c)

Figure 4.6: *Mumax* simulation with $J_{racetrack} = 5 \times 10^{10}\text{A/m}^2$ and $J_{lateralTrack} = 2 \times 10^{11}\text{A/m}^2$. Skyrmion changes track due to $J_{lateralTrack}$.



(a) (b)



(c)

Figure 4.7: *Mumax* simulation with $J_{racetrack} = 5 \times 10^{10}\text{A/m}^2$ and $J_{lateralTrack} = 5 \times 10^{10}\text{A/m}^2$. Skyrmion comes out from lateral track and is pushed down in the racetrack.

4.3 AddRoundKey operation

AddRounKey operation starts as soon as skyrmion reaches the first lateral track, labelled V_{mov}. Here, respectively, CTRL_V_{mov} and CTRL_V_{top} are asserted, leading to the skyrmion to reach the duplication element (x2). The bottom output keeps moving until it is stopped by the notch element of the synchronizing net. On the other hand, the top output is sent back to the racetrack and then flushed, by enabling CTRL_V_{bl}.

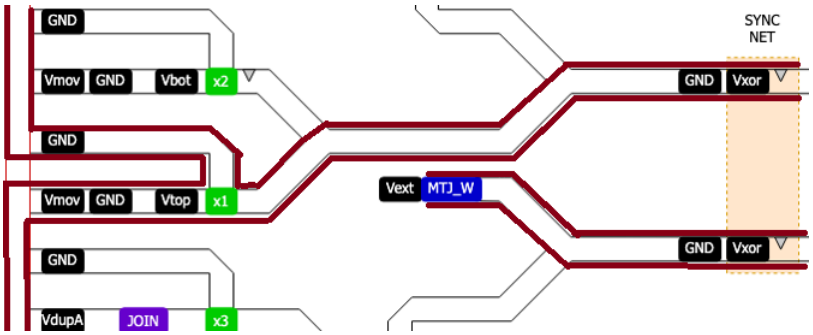


Figure 4.8: AddRoundKey datapath: inputs initialization.

The write head in Figure 4.8 nucleates a skyrmion if the bit of the key is a '1' logic. In that case, CTRL_V_{ext} is enabled causing the movement of the key skyrmion, that ends as soon as the synchronizing net is approached. Afterwards, the actual XOR operation takes place: skyrmions are guided inside the INV/COPY gates (whose functionality is explained in Section 1.4) by applying CURRENT_V_{xor}.

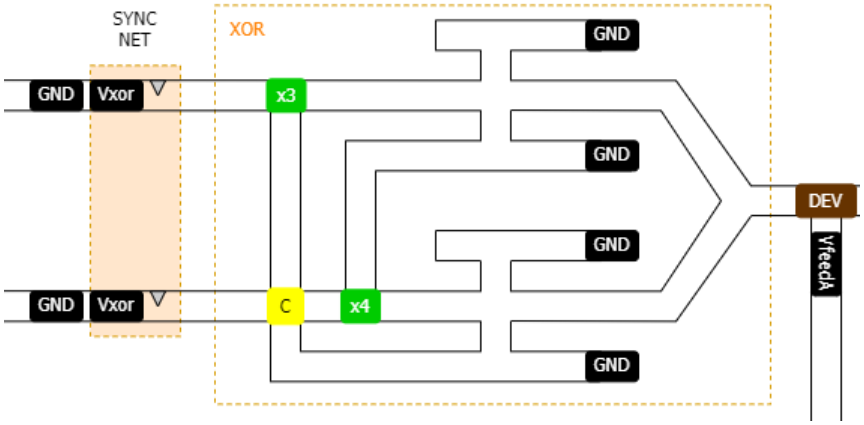


Figure 4.9: AddRoundKey datapath: XOR operation.

Outputs are, then, joined together, providing the XOR result. Next step is sending back the output by exploiting the deviation block: CTRL_V_{feedA} is asserted,

enabling skyrmion deflection. Subsequently, CTRL_V_{dev} and CTRL_V_{ret} are asserted, bringing the output inside the racetrack and, at the same time, ending the AddRoundKey operation.

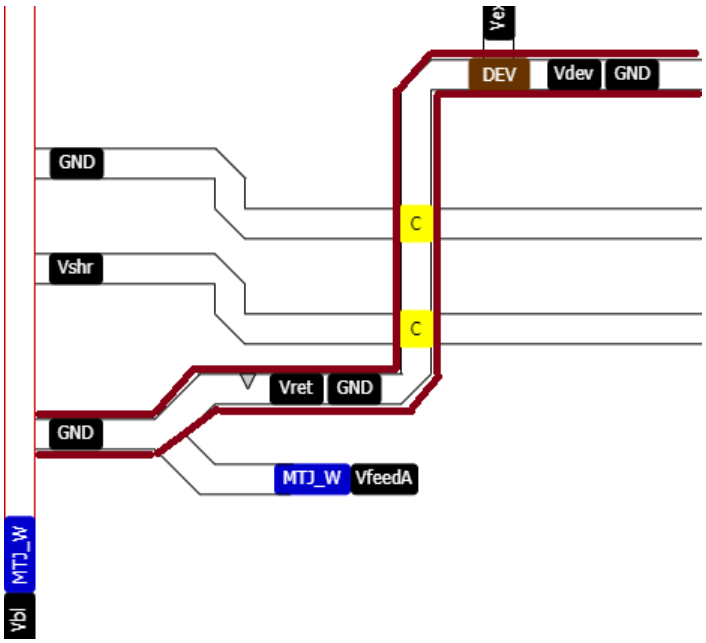


Figure 4.10: AddRoundKey datapath: sending back output.

4.4 SubBytes operation

SubBytes first steps are the same as AddRoundkey. The main difference involves the deviation block at the end of the XOR block: skyrmion are not deviated, but sent straight. Here, the output is read by a MTJ_R that, in presence of a skyrmion, generates an impulse of 10ps.

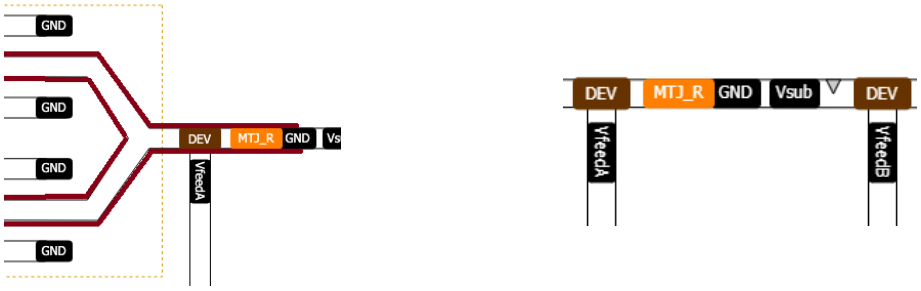


Figure 4.11: SubBytes datapath: (a) skyrmion path and (b) reading.

Since the impulse generated by the MTJ_R lasts for dozens picoseconds, this value must be stored, otherwise, it will not be sensed at next clock rising edge. SR latch are used:

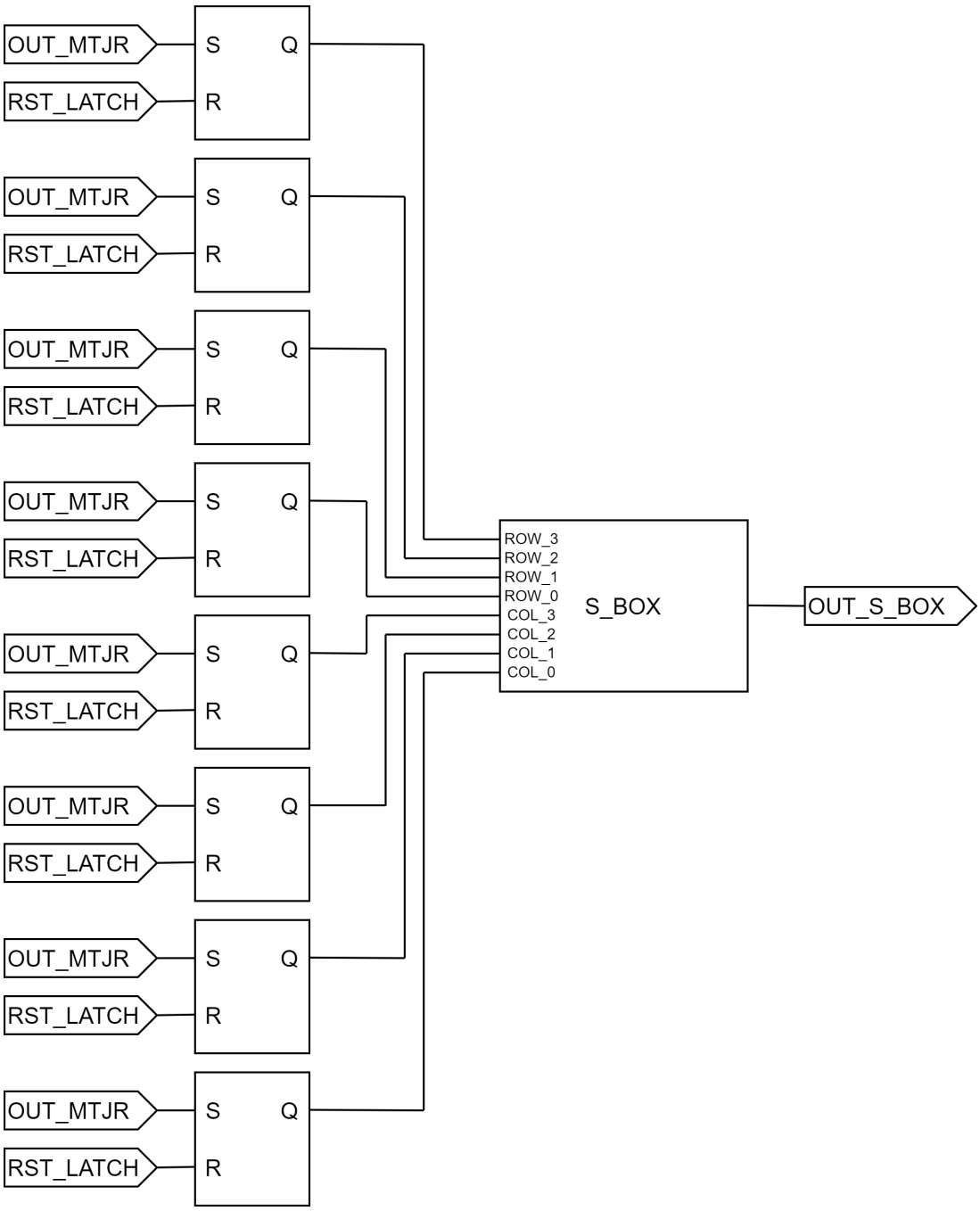


Figure 4.12: SR Latch.

Latch outputs are sent to a look-up-table (S-BOX), that contains the value of the byte that needs to be substituted. At this point, control unit compares the latch output with S-BOX output. Table below shows all four cases that can occur:

LATCH	S_BOX	
0	0	The subytes operation ends
0	1	A skyrmion must be nucleated
1	0	The skyrmion sensed is annihilated
1	1	The skyrmion sensed is guided back to the racetrack

Table 4.1: Truth table.

In the second case (the first one is trivial), a skyrmion must be nucleated. This can be performed by exploiting the write head in Figure 4.13.

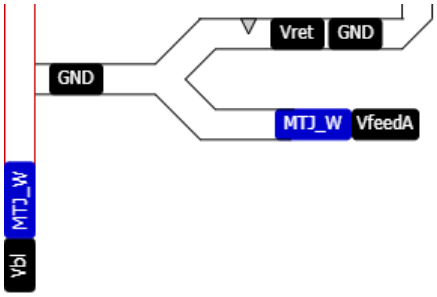


Figure 4.13: SubBytes operation: second case.

In the third case, the MTJ_R senses a control signal, that enables the skyrmion annihilation. In the last case, the procedure consists in guiding back the skyrmion to the racetrack. This is executed by enabling the second deviation block by asserting CTRL_V_{feedB}.

4.5 ShiftRow operation

Figure 4.14 shows the elements needed in order to perform ShiftRow operation. In particular, after completing SubBytes operation, skyrmion are pushed down in the racetrack until they reach the lateral track labelled V_{shr}. Here, by asserting CTRL_V_{shr}, skyrmions from words of the same row get in a sort of "circular" nanotrack, thanks to the join elements. After the first word overcomes the last join, three different cases can occur, depending on the number of shift that needs to be performed. As soon as skyrmion reaches its cell destination, deviation block is enabled thanks to CTRL_V_{back}. ShifRow operation ends.

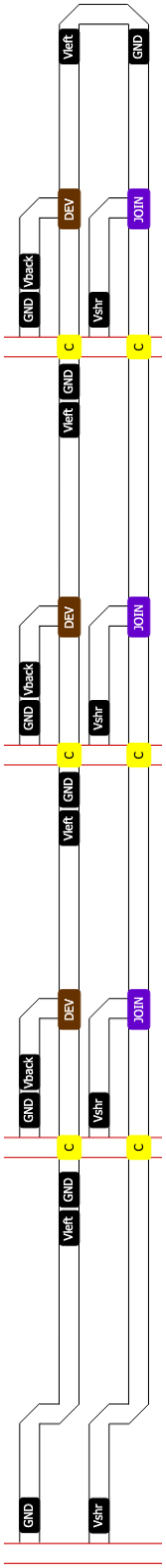


Figure 4.14: ShiftRow operation.

4.6 MixColumns operation

Since its complexity, this operation has been splitted in two sub-process. Firstly, all multiplication with costant matrix are performed, and, then, XOR operation between all results is executed.

Multiplication sub-process

First step involves the use of the duplication element whose lateral track is labelled V_{dupA} (Figure 4.16). This must be done in order to preserve the initial data, since MixColumns operation is repeated for four times. Next step depends on which kind of multiplication needs to be performed. In particular, in case of X1, skyrmion are pushed inside the first lateral track (from the bottom) labelled V_{mov}. In case of X3, skyrmion gets inside the second lateral track labelled V_{mov}. This is done for two reasons: storing the initial data[1] and reading the MSB (Section 3.1).

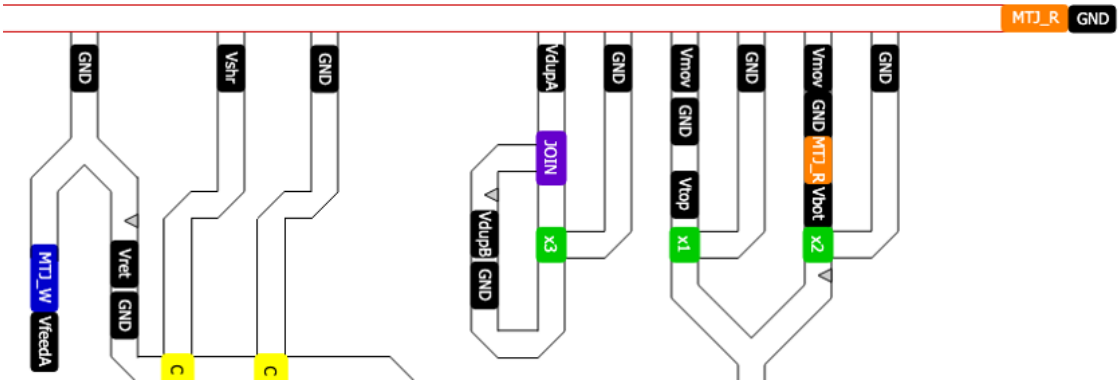


Figure 4.15: Multiplication operation: initialization. This figure shows the last cell of the word, in which there is a read head in the second lateral track labelled V_{mov}.

Afterwards, since the multiplication for 2 can be carried out as 1-bit left shift, this can be done by, simply, asserting the CTRL_V_{bl}. This causes the skyrmion to reach next cell first lateral track labelled V_{mov}. It is important to highlight that, in case of X2, the MSB reading occurs at the end of the racetrack. Moreover, as explained in SubBytes operation, the output of the read head must be stored in a latch in order to preserve the information. Therefore, two latches for each word needs to be used. Then, control unit checks latches output and, in case they are '1', a XOR operation with a constant value needs to be performed (Section 3.1). After this operation is executed, two different case can occur: in case of X3, the results

[1] Multiplication for three can be done by exploiting a mathematical trick: $(11)_2$ can be written as $(01)_2 \oplus (10)_2$. This means that X3 can be performed as X2 and the result is XORed with initial data. For example, $(56)_{16} \cdot (03)_{16} = (56)_{16} \cdot [(56)_{16} \cdot (02)_{16}]$.

must be feeded back in order to compute the second round of XOR operation with the initial data, which is sent to the synchronization net by asserting CTRL_V_{bot}; in case of X2 and X1, skyrmions keep going straight and are blocked by the notch after the deviation block.

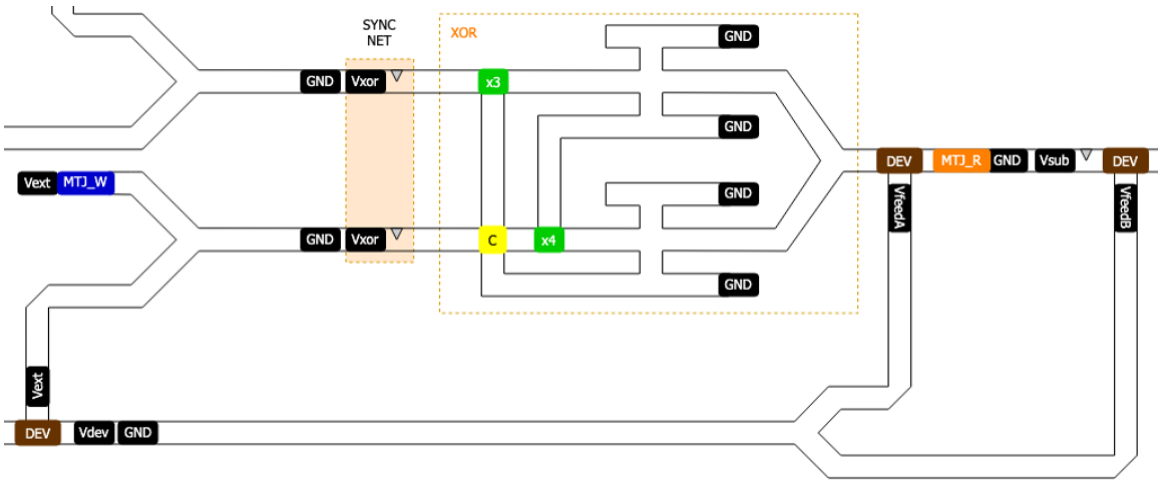


Figure 4.16: Multiplication operation: XOR operation.

Addition sub-process

After all multiplication ends, the second sub-process of MixColumns operation begins. As Figure 4.17 shows, firstly, two preliminary XOR operations are performed, and, then, outputs are XORed again. Therefore, each MixColumn round needs two addition sub-process.

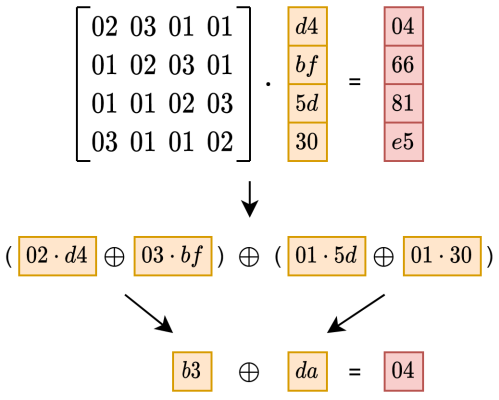


Figure 4.17: Addition operation: schematic.

As Figure 4.18 illustrates, skyrmion are pushed into a "circular" nanotrack (similiar to the one used for ShiftRow operation) thanks to join elements. It is important to highlight that only skyrmions not belonging to "host words[2]" get inside this track.

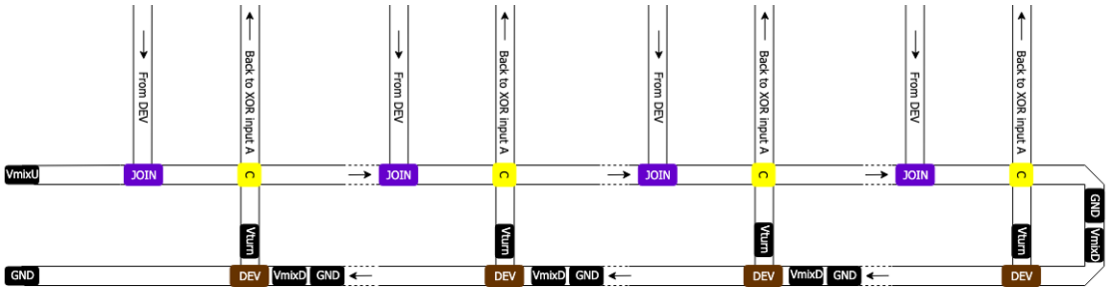


Figure 4.18: Addition operation: skyrmion transfer.

Skyrmion destination depends on the addition round. For example, in the first round, skyrmions coming from word 3 transfer in word 1, whereas those coming from word 2 transfer in word 0. Aftewards, XOR operations take place and the output is sent again to the "circular" track: addition output of word 1 gets inside XOR block of word 0, here last operation is executed, ending the first round of MixColumns. Multiplication and addition sub-process are repeated other three times.

4.7 Reading process

At the end of each round, data is written in an output register, by exploiting the read head at the end of each racetrack. By implementing in this way, not only the final output is avalaible to be read, but also partial results of each round.

4.8 Key expansion block

This section explains how key scheduler is performed. Firstly, words from column 3 are extracted from a register file. After rotating these data, their bits are split in two groups of four and compared with S-BOXes. Subsequently, outputs are XORed with the elements of column 0 and a costant value RCON. The result is written in the register file of column 0 and, at the same time, XORed with the elements of column 1. Same procedure is repeated until all key columns are updated. As it can be seen in Figure 4.28, multiplexer inputs are the inital key and the updated key.

[2]Host words are those words in which the XOR operation takes place. For example, in the first round of XOR sub-process, host words are 00 and 01, in the second one only 00.

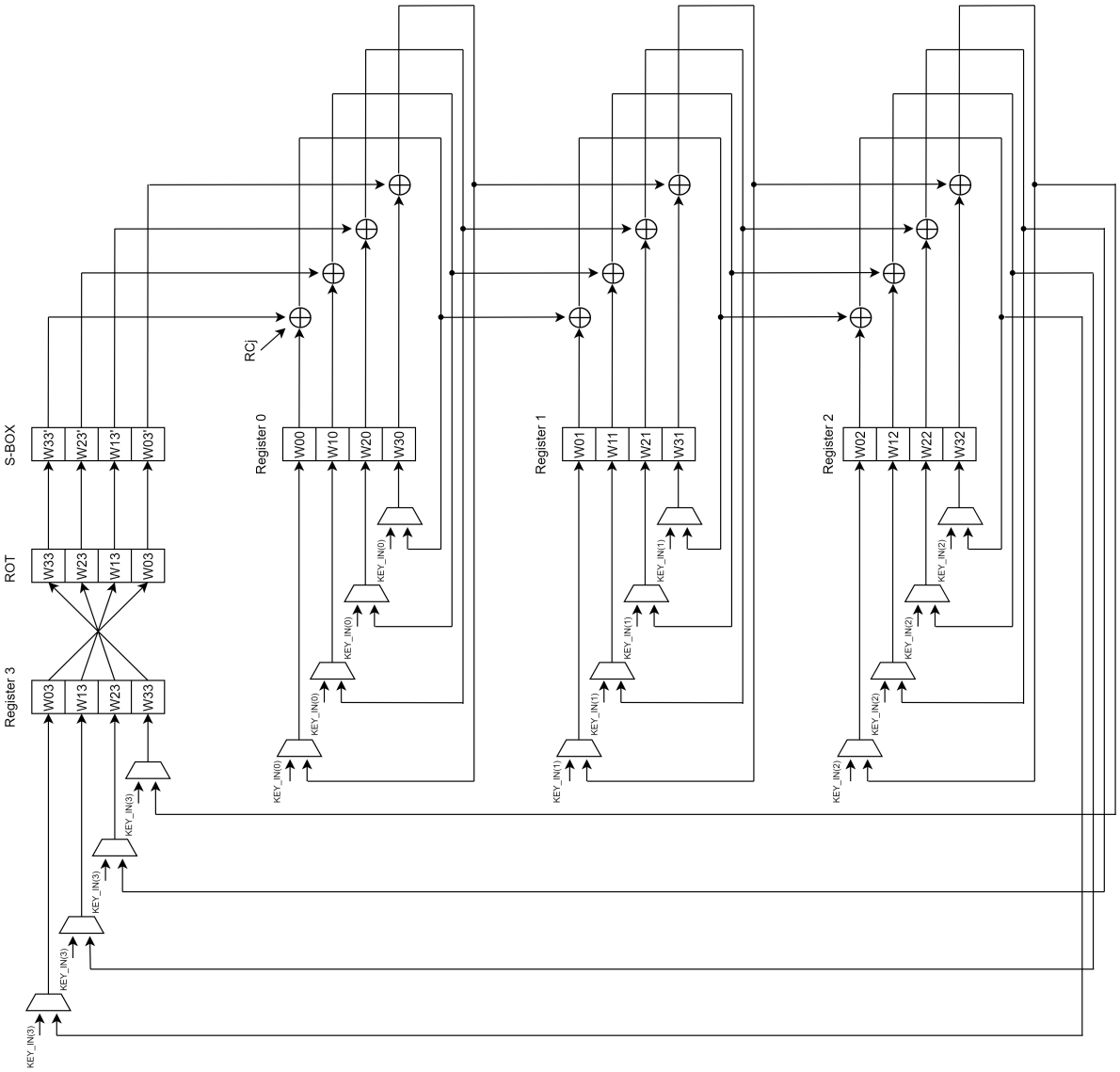


Figure 4.19: Key expansion block datapath.

4.9 Control unit

Due to the complexity of the algorithm, it has been decided to implement linked FSMs (Figure 4.20), instead of having only one FSM that controls the whole algorithm. In particular, the master FSM executes "some states" until state S_A is reached. Here, a START signal enables a low-hierarchically FSM, allowing that state machine to move from IDLE state into the counting sequence. When the low-hierarchically state machine reaches state S_B, the signal DONE is asserted, which acts as input to the master FSM, allowing to move it to next state. The low-hierarchically state machine returns to the IDLE state.

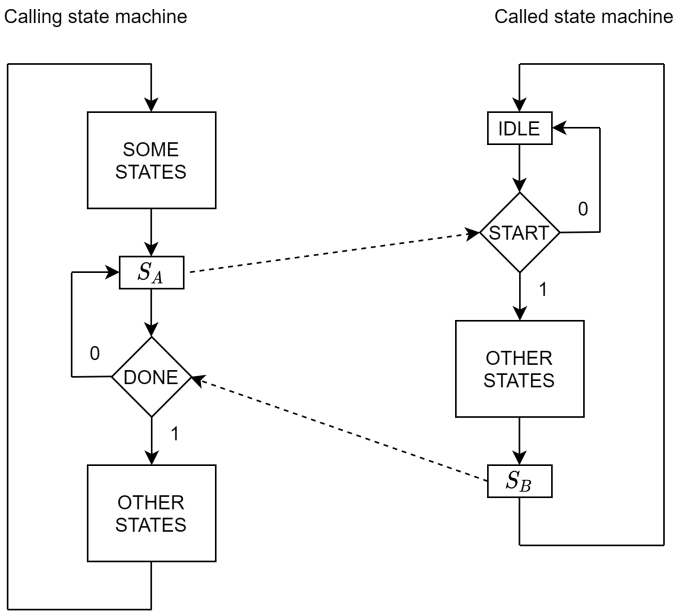


Figure 4.20: Linked state machines.

Each AES operation has its own state machine, except for MixColumn operation. Indeed, due to its complexity, it has been decided to split it in multiplication FSM and addition FSM. Beside enabling each operation, the master state machine executes the writing process and the reading process. As Figure 4.21 shows, after completing S_21, master FSM starts next round and update a counter that takes into account the round number. When round number = 10, the writing of the final result begins, ending the algorithm computation. SubBytes FSM performs different computation as function of the output of latches and the output of the S-BOX (Figure 4.23). In addition, ShifRow FSM receives a input signal (ROTATE) generated by the master that identify the number of shifts that needs to be done. In a similar way, master FSM sends a signal (MULTIPLICATION) to multiplication FSM that specify which product to execute (Figure 4.24). Moreover, since the addition

sub-process in MixColumns requires four rounds, it has been decided to split this operation in two FSM, a master state machine that performs the addition (Figure 4.26) and a slave one that performs the skyrmion transfers between words of the same column (Figure 4.27). These state machines communicate with signal START and DONE.

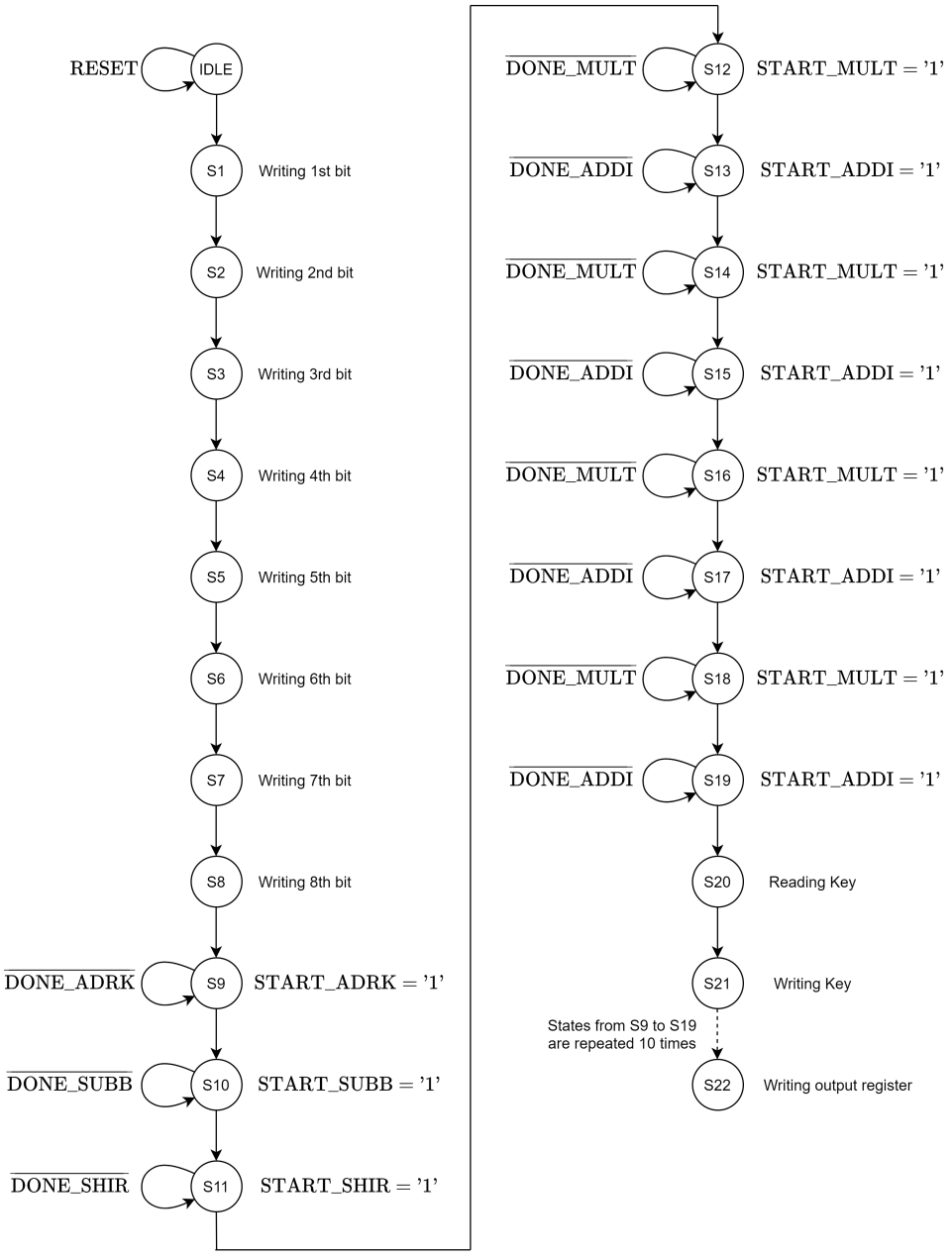


Figure 4.21: MasterFSM diagram state.

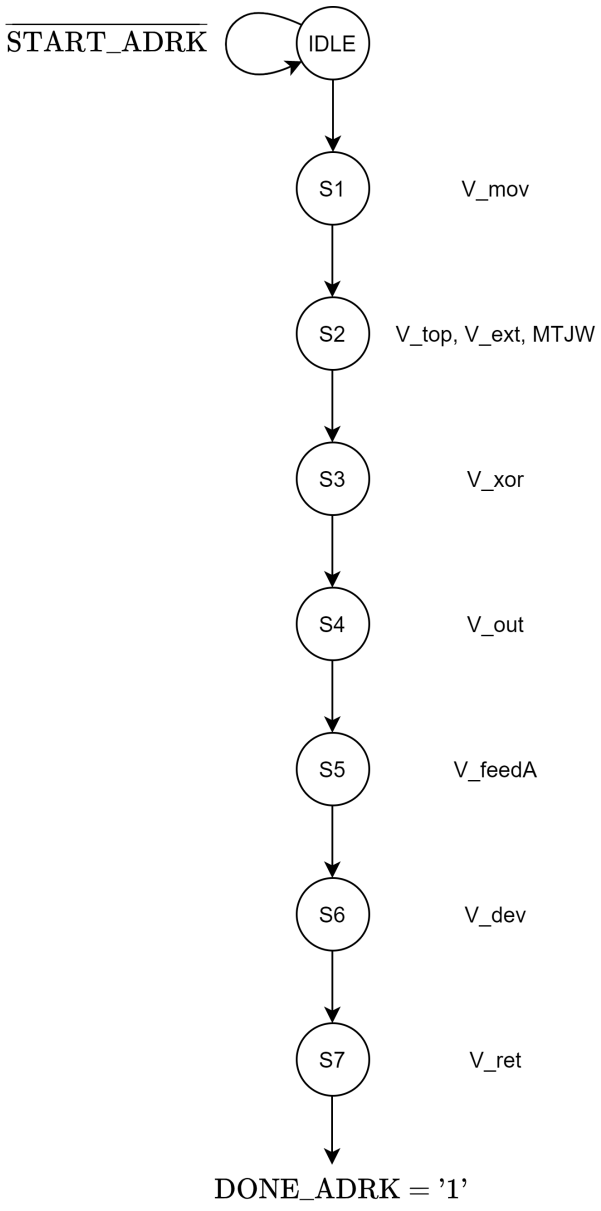


Figure 4.22: AddRoundKeyFSM diagram state.

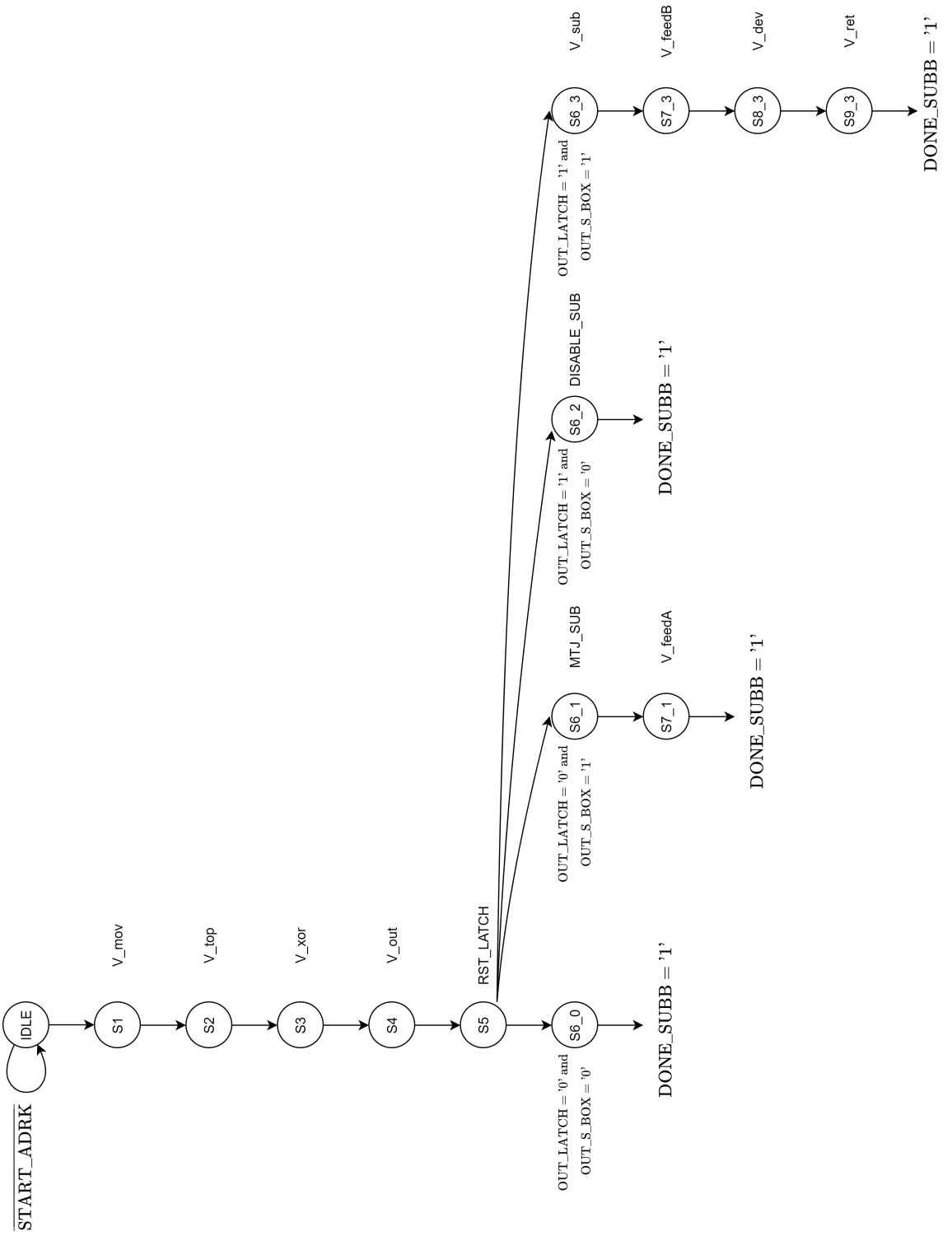


Figure 4.23: SubBytesFSM diagram state.

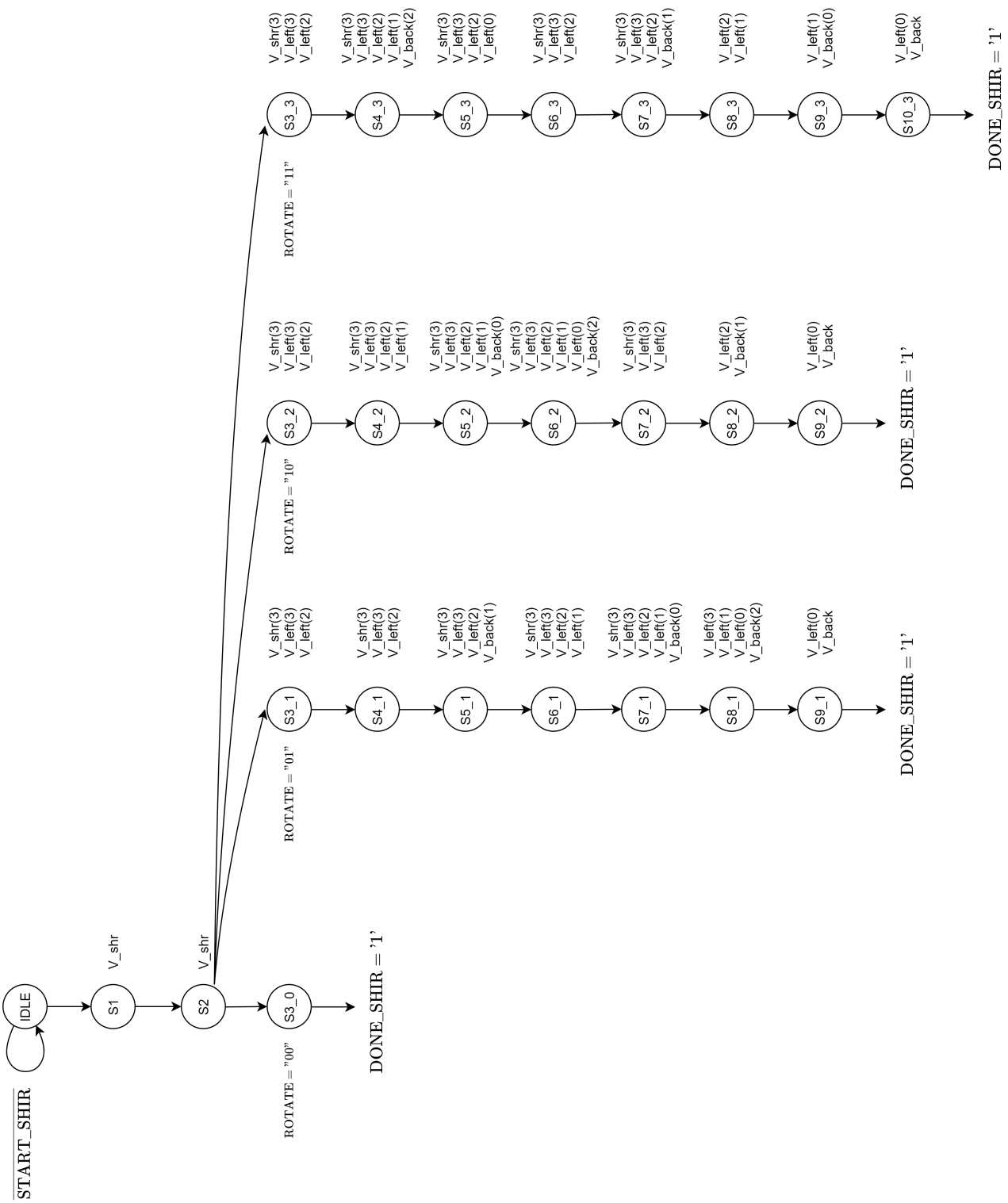


Figure 4.24: ShifRowFSM diagram state.

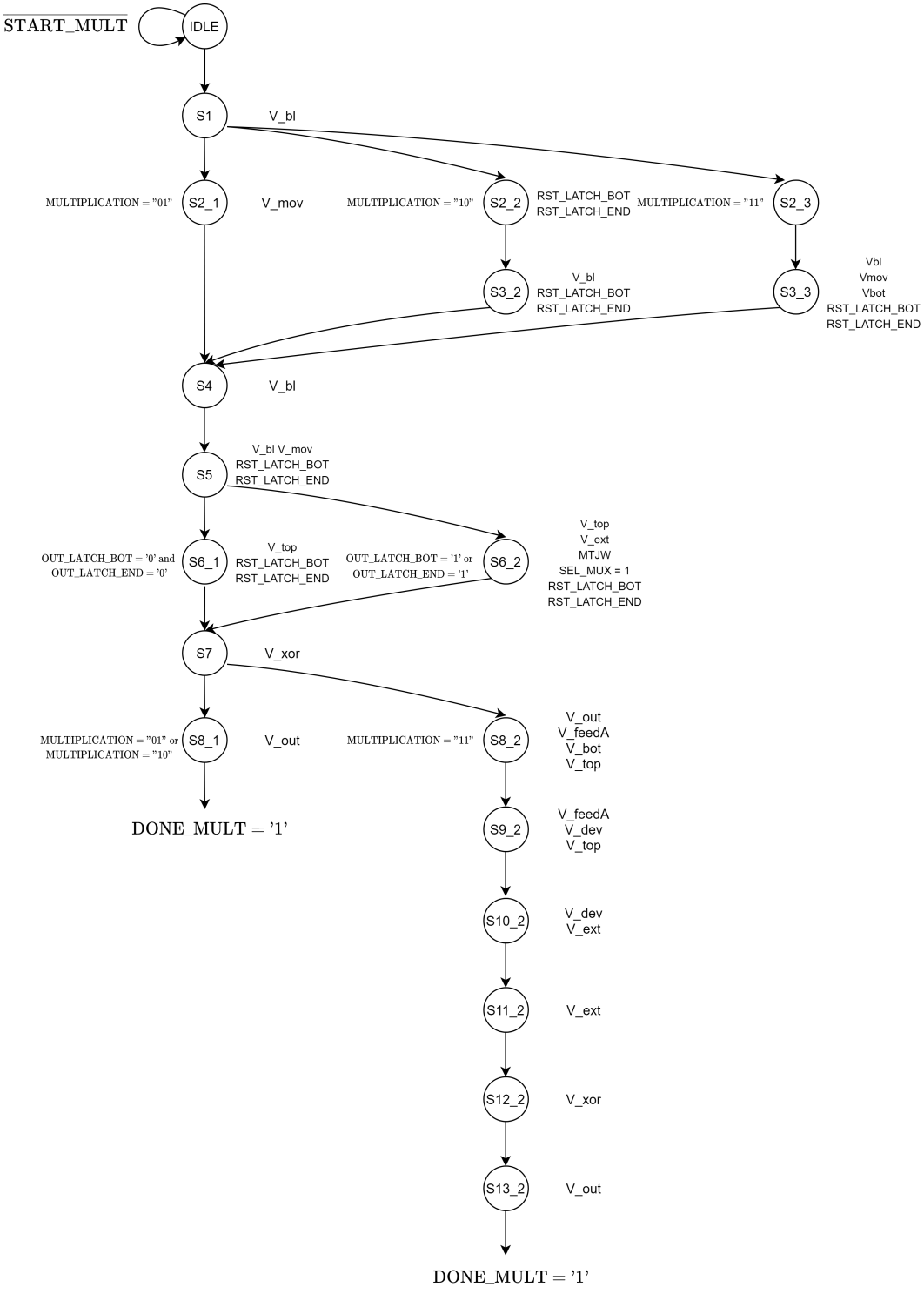


Figure 4.25: MultiplicationFSM diagram state.

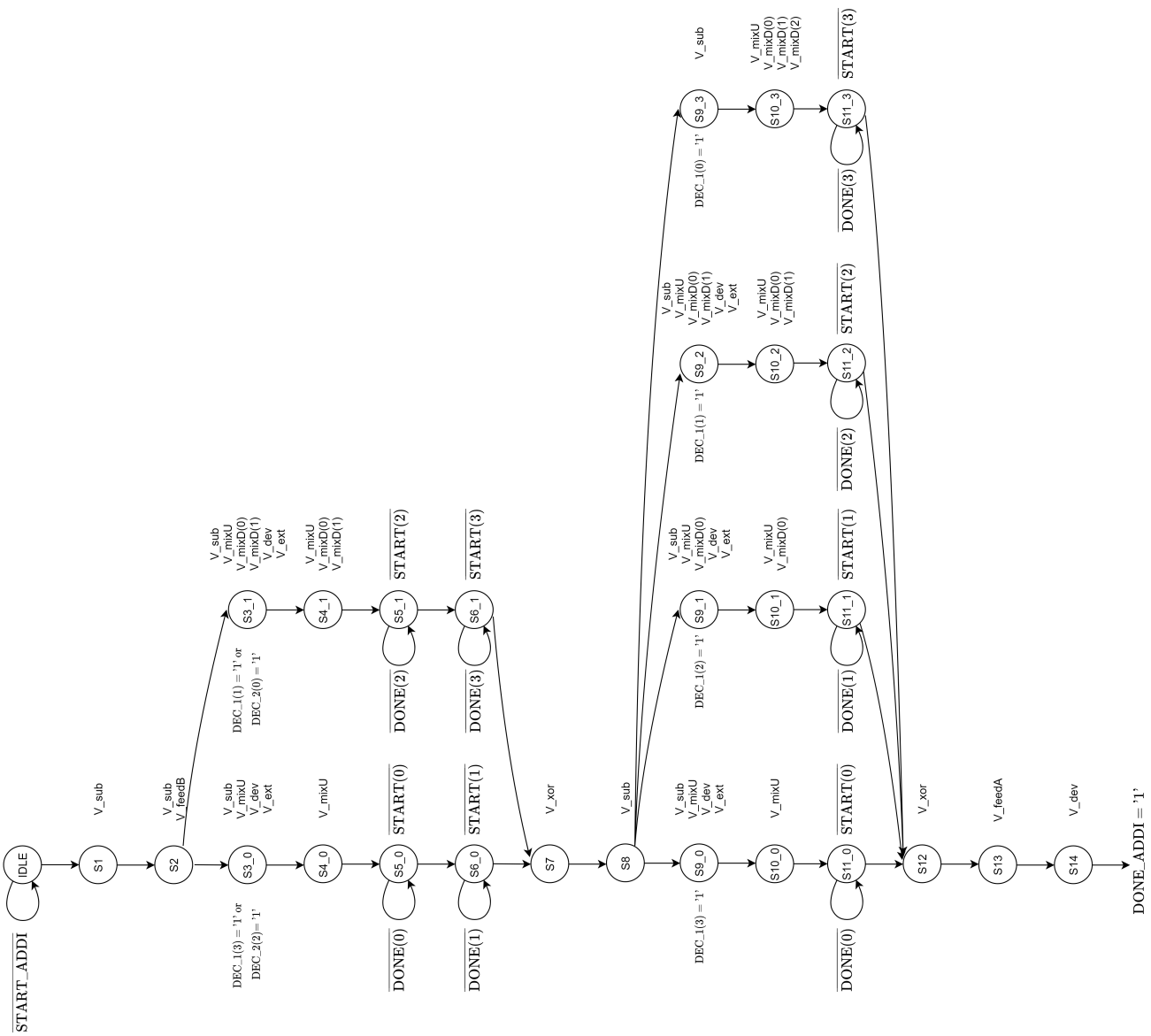


Figure 4.26: AdditionFSM diagram state: master.

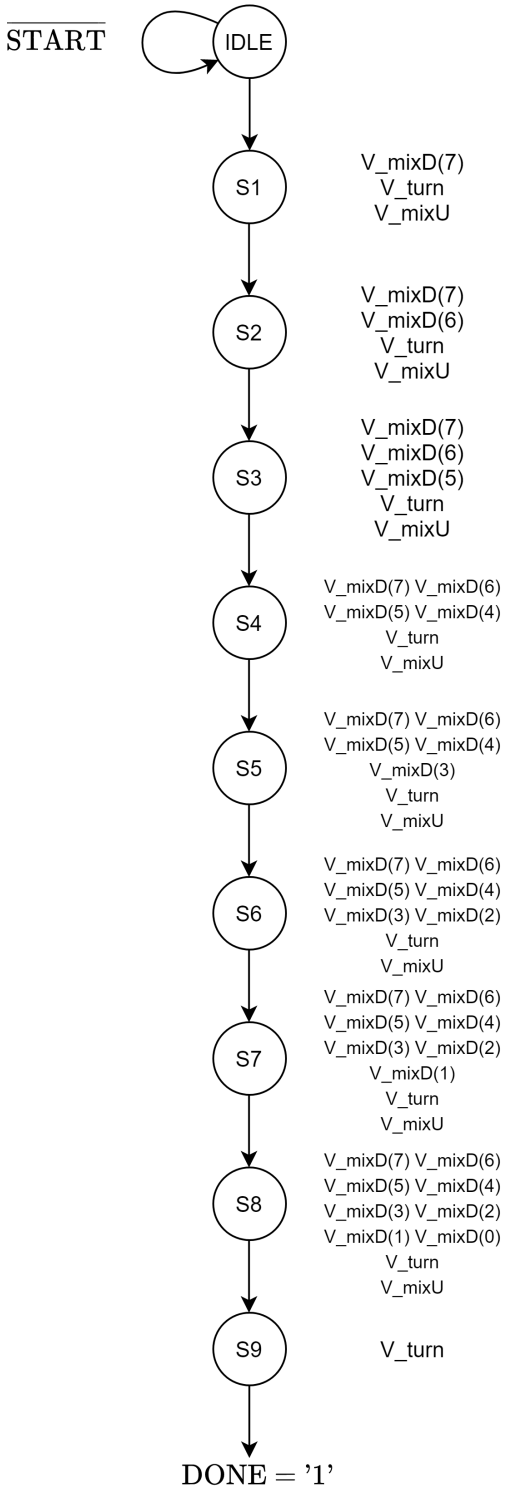


Figure 4.27: AdditionFSM diagram state: slave.

4.10 VHDL Model

To verify the functioning of the architecture that have been discussed, a VHDL behavioural description has been implemented. The main goal has been on reproducing in the simplest way possible the behaviour of each of the components used inside the cells. In particular, these following components have been taken into account:

- **Line**: In the line element, the simulation step is 10 ps. Whenever a skyrmion is detected at the input, its position will be (0.0, 0.0); coordinates are refreshed every 10 ps only if the current is higher than the depinning current value. If this is the case, the new x coordinate is computed as the old x coordinate, plus the elapsed time multiplied by the horizontal speed, while the y coordinate remains always equal to zero. Therefore, only the horizontal movement of the skyrmion is taken into account. Whenever a skyrmion reaches the end of the line, it is emitted by generating an impulse of 10 ps.

- **Join**: The join element is quite similar to the line: instead of one input, two are taken into account. However, they are considered as point-like and coincident, that is, a skyrmion can enter from two different inputs, but both these inputs are considered exactly like the single input of a line element. In addition, this implementation lacks of the skyrmion-skyrmion collision at the junction. From this point on, the description of the gate is exactly the same as for the line: the y coordinate never changes and the x coordinate is updated each 10 ps.

- **Notch**: There are two different depinning current: one allows skyrmion movement, the others let the skyrmion overcome the notch. Subsequently, skyrmion speed changes as function of the injected current. If the current applied is equal or higher than the current value needed for making the skyrmion go through the notch, this element can be considered as simple line. On the other hand, whenever the current injected is intermediate between the two depinning current values, a more complex behaviour is described. Firstly, it is important to understand if the skyrmion, whose coordinates are being updated, crossed already the notch: in this case, updated x coordinate can be evaluated as usual (this time using the lowest velocity value). If instead it still has to go through the notch, it is important to check if there other skyrmion already stuck in the notch. The minimum distance from the notch that is allowed for the considered skyrmion, as a function of the number of blocking skyrmions, is computed by adding to the diameter of a skyrmion the minimum skyrmion-skyrmion distance. If $notch\_distance = notch\_xCoordinateold\_xCoordinate$ is greater than $minimum\_distance$ by at

least $\Delta\_distance = horizontalVelocity \cdot elapsedTime$, then the new x coordinate can again be computed as usal, adding to the old x coordinate the value of $\Delta\_distance$. On the other hand, if the skyrmion is not allowed to complete its movement due to the presence of other blocked skyrmions, then the skyrmion queues up.

- **MTJR**: If a skyrmion has been detected and if the input current is greater than depinning current, an impulse is generated; in any other case the output is 0.
- **MTJW** If the input CTRL is asserted, an impulse is generated (meaning that a skyrmion has been nucleated); in any other case the output is 0.
- **Duplication**: If a skyrmion is detected on the input, and if the input current is greater than depinnin current, both outputs generate an impulse (the input skyrmion is duplicated); in any other case both outputs are 0.
- **Deviation**: This element takes into account two different depinning currents, similarly to notch element. There are two inputs and two outputs. If the current of the racetrack is greater than notch depinning current, this element behaves like a line. In case it is lower but greater than the current needed to move it, it will gets stuck at the notch. Here, in case the skyrmion needs to change track, a lateral current is applied.
- **Cross**: If a skyrmion has been detected on input A and the current in this input is greater than depinning current, the output A generates an impulse. Similarly, if a skyrmion has been detected on input B and the current in this input is greater than depinning current, the output B generates an impulse.

4.11 Simulation

Simulations have been carried out by using ModelSim software, Figure below shows the benchmark used in order to verify the functionality of the architecture. Signal that have been plotted are: DATA IN (it is a matrix 4x4 of the data that needs to be encrypted), DATA OUT (it is a matrix 4x4 of the data that have been encrypted, these register are updated each round) and ROUND that it is a counter that is updated each round.

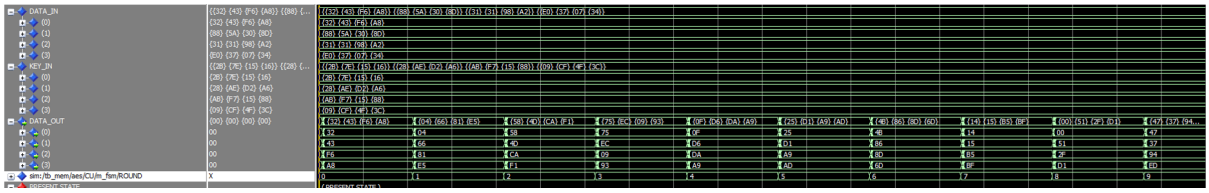


Figure 4.28: Simulation results.

Afterwards, output data have been compared with a python script (it uses the package PyCryptodome [38]) by providing same data input and key.

4.12 Architecture performance

Finally, some parameters have been evaluated: throughput, latency and power consumption. In particular, it has been considered writing latency, that it is the time needed in order to nucleate the skyrmions of a word, and encryption latency. Critical path has been also taken into account and it corresponds to the time from when the skyrmion is nucleated in the racetrack to when it reaches next cell (740 ps). Regarding power dissipation, it have been evaluated by considering both the sheet resistance[3] and the electrical current that flows in the track. Power can be obtained

[3]Sheet resistance, often called sheet resistivity, is a measure of resistance of thin films that are nominally uniform in thickness. In a regular three-dimensional conductor, the resistance can be written as:

$$R = \rho \frac{L}{A} = \rho \frac{L}{Wt} \tag{4.1}$$

where ρ is the resistivity, A is the cross-sectional area, and L is the length. The cross-sectional area can be split into the width W and the sheet thickness t. By managing Equation (4.1), the sheet resistance R_s can be obtained:

$$R = \frac{\rho}{t} \frac{L}{W} = R_s \frac{L}{W} \tag{4.2}$$

by taking into account this equation:

$$P = R_s \cdot I^2 \tag{4.3}$$

The tracks in which skyrmions move is composed of two layers, a platinum layer (Pt) and a cobalt layer (Co). The sheet resistance value of this particular alloy is 40Ω [39]. Table below shows power consumption for each AES operation:

Operation	Power(nW)
Writing	6.4
AddRoundKey	153.6
SubBytes	104.96
ShiftRow	32.8
MixColumns(Multiplication)	153.6
MixColumns(Addition)	27.2

Table 4.2: Power consumption for each operation.

In order to have a complete analysis, it has been choosen to evaluate power dissipation of control unit. This has been accomplished by exploiting a design compiler (*Synopsis*) that has an useful tool that allows to calculate power consumption. Eventually a comparison[4] with other AES implementation has been carried out:

Design	Power	Latency	Throughput
Feldhofer [40]	4.5 µW	10.32ms	12.4 Kbps
Kaps [41]	20.23 µW	1.07ms	119.6 Kbps
Lin [42]	40.9 mW	30ns	4.2 Gbps
Hawng [43]	0.69 µW	3.56ms	35.9 Kbps
This work	1.1 mW	3.55µs	36 Mbps

Table 4.3: Architecture parameters compared with previous works.

[4]Since the power consumed by the FSM is much greater (∼ mW) than the one dissipated by the datapath (∼ nW), only the first one has been considered in the comparison.

Chapter 5

Conclusions

Starting from the description of skyrmion basic functionality, it has been shows how these magnetic objects can be manipulated in order to perform logic functions such AND/OR and INV/COPY.
Secondly, it has been showed how modern computing devices are limited due to the presence of a "bottleneck" between CPU and memory performance. This limitation can be solved by using a logic-in-memory architecture in which logic and memory become only one entity, and not two separated one. Thanks to their property (small size, low depinning current, low power consumption), skyrmion assumes the role of information carrier in this kind of architecture. Advanced Encryption Standard algorithm has been taken into account in order to show how powerful mixing skyrmions and logic-in-memory architecture is. In particular, Chapter 4 illustrated how skyrmions can be manipulated by only applying an electrical current and how complex AES operation such as ShiftRow and MixColumns can be implemented. From an energetic point of view, it has been shown that each operation dissipates a really low amount of power ($\sim$ nW). On the other hand, controlling all signals that enable the injection of current in each track requires an higher amount of energy ($\sim$ mW). A future study can be focused on optimizing control unit in order to reduce the power consumption. The architecture proposed showed how high throughput can be reached ($\sim$ 37 Mbps) without consuming too much energy. This result has been compared with previous work.
In addition, there are still some issues that need to be solved: it has been assumed the possibility of allowing two nanotracks to cross without causing any alteration of the information. Until today, any solution have been provided in literature. Until today, any solution have been provided in literature. Another assumption made in this thesis that should be verified from a physical point of view is related to the possibility of separating two metal traces with some dielectric in between, indeed this could cause the annihilation of the skyrmion. This is extremely important in cross regions in which sneaky current could deviate the skyrmion from its path. If this

assumption was proved wrong, some other mechanisms for dynamically controlling the skyrmion motion needs to be studied, otherwise the architecture would not work properly.

Bibliography

[1] Alexei N Bogdanov and DA Yablonskii. "Thermodynamically stable vortices in magnetically ordered crystals. The mixed state of magnets". In: *Zh. Eksp. Teor. Fiz* 95.1 (1989), p. 178.

[2] AN Bogdanov and UK Rößler. "Chiral symmetry breaking in magnetic thin films and multilayers". In: *Physical review letters* 87.3 (2001), p. 037203.

[3] Ulrich K Roessler, AN Bogdanov, and C Pfleiderer. "Spontaneous skyrmion ground states in magnetic metals". In: *Nature* 442.7104 (2006), pp. 797–801.

[4] Sebastian Mühlbauer et al. "Skyrmion lattice in a chiral magnet". In: *Science* 323.5916 (2009), pp. 915–919.

[5] Seonghoon Woo et al. "Observation of room-temperature magnetic skyrmions and their current-driven dynamics in ultrathin metallic ferromagnets". In: *Nature materials* 15.5 (2016), pp. 501–506.

[6] Xichao Zhang, Motohiko Ezawa, and Yan Zhou. "Magnetic skyrmion logic gates: conversion, duplication and merging of skyrmions". In: *Scientific reports* 5.1 (2015), pp. 1–8.

[7] Wang Kang et al. "Skyrmion-electronics: An overview and outlook". In: *Proceedings of the IEEE* 104.10 (2016), pp. 2040–2061.

[8] István Kézsmárki et al. "Néel-type skyrmion lattice with confined orientation in the polar magnetic semiconductor GaV 4 S 8". In: *Nature materials* 14.11 (2015), pp. 1116–1122.

[9] Alexey Kovalev and Shane Sandhoefner. "Skyrmions and Antiskyrmions in Quasi-Two-Dimensional Magnets". In: *Frontiers in Physics* 6 (Sept. 2018). DOI: 10.3389/fphy.2018.00098.

[10] João Sampaio et al. "Nucleation, stability and current-induced motion of isolated magnetic skyrmions in nanostructures". In: *Nature nanotechnology* 8.11 (2013), pp. 839–844.

[11] Niklas Romming et al. "Writing and deleting single magnetic skyrmions". In: *Science* 341.6146 (2013), pp. 636–639.

[12] Wataru Koshibae and Naoto Nagaosa. "Creation of skyrmions and antiskyrmions by local heating". In: *Nature communications* 5.1 (2014), pp. 1–11.

[13] Dan A Allwood et al. "Magnetic domain-wall logic". In: *Science* 309.5741 (2005), pp. 1688–1692.

[14] Yan Zhou and Motohiko Ezawa. "A reversible conversion between a skyrmion and a domain-wall pair in a junction geometry". In: *Nature communications* 5.1 (2014), pp. 1–8.

[15] Junichi Iwasaki, Masahito Mochizuki, and Naoto Nagaosa. "Current-induced skyrmion dynamics in constricted geometries". In: *Nature nanotechnology* 8.10 (2013), p. 742.

[16] Wang Kang et al. "Magnetic skyrmions for future potential memory and logic applications: Alternative information carriers". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 119–124.

[17] CH Li et al. "Electrical detection of charge-current-induced spin polarization due to spin-momentum locking in Bi 2 Se 3". In: *Nature nanotechnology* 9.3 (2014), p. 218.

[18] Christian Hanneken et al. "Electrical detection of magnetic skyrmions by tunnelling non-collinear magnetoresistance". In: *Nature nanotechnology* 10.12 (2015), p. 1039.

[19] Dax M Crum et al. "Perpendicular reading of single confined magnetic skyrmions". In: *Nature communications* 6.1 (2015), pp. 1–8.

[20] T Furubayashi et al. "Current-perpendicular-to-plane giant magnetoresistance in spin-valve structures using epitaxial Co 2 FeAl 0.5 Si 0.5/Ag/Co 2 FeAl 0.5 Si 0.5 trilayers". In: *Applied Physics Letters* 93.12 (2008), p. 122507.

[21] Riccardo Tomasello et al. "Electrical detection of single magnetic skyrmion at room temperature". In: *AIP Advances* 7.5 (2017), p. 056022.

[22] Sara G. Miralles. "Molecular Spintronic Devices: from Molecular Spin Valves to Spin-OLEDs". PhD thesis. June 2017.

[23] J Ping Liu, Zhidong Zhang, and Guoping Zhao. *Skyrmions: topological structures, properties, and applications*. CRC Press, 2016.

[24] Maverick Chauwin et al. "Conservative skyrmion logic system". In: *arXiv preprint arXiv:1806.10337* (2018).

[25] A. M. Peskin. "A Logic-in-Memory Architecture for Large-Scale-Integration Technologies". In: *Proceedings of the ACM Annual Conference - Volume 1*. ACM 72. Boston, Massachusetts, USA: Association for Computing Machinery, 1972, 1225. ISBN: 9781450374910. DOI: 10.1145/800193.805818. URL: https://doi.org/10.1145/800193.805818.

[26] Giulia Santoro, Giovanna Turvani, and Mariagrazia Graziano. "New logic-in-memory paradigms: An architectural and technological perspective". In: *Micromachines* 10.6 (2019), p. 368.

[27] Dae Hyun Kim et al. "Design and analysis of 3D-MAPS (3D massively parallel processor with stacked memory)". In: *IEEE Transactions on Computers* 64.1 (2013), pp. 112–125.

[28] Shaahin Angizi, Zhezhi He, and Deliang Fan. "PIMA-logic: a novel processing-in-memory architecture for highly flexible and energy-efficient logic computation". In: *Proceedings of the 55th Annual Design Automation Conference*. 2018, pp. 1–6.

[29] Roman Kaplan et al. "A resistive cam processing-in-storage architecture for dna sequence alignment". In: *IEEE Micro* 37.4 (2017), pp. 20–28.

[30] Shoun Matsunaga et al. "Fabrication of a nonvolatile full adder based on logic-in-memory architecture using magnetic tunnel junctions". In: *Applied Physics Express* 1.9 (2008), p. 091301.

[31] Fabrizio Riente et al. "Towards logic-in-memory circuits using 3d-integrated nanomagnetic logic". In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE. 2016, pp. 1–8.

[32] Markus Becherer et al. "Magnetic ordering of focused-ion-beam structured cobalt-platinum dots for field-coupled computing". In: *IEEE Transactions on Nanotechnology* 7.3 (2008), pp. 316–320.

[33] Stephan Breitkreutz et al. "Nanomagnetic logic: Demonstration of directed signal flow for field-coupled computing devices". In: *2011 Proceedings of the European Solid-State Device Research Conference (ESSDERC)*. IEEE. 2011, pp. 323–326.

[34] Giovanni Causapruno et al. "Reconfigurable systolic array: From architecture to physical design for NML". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.11 (2016), pp. 3208–3217.

[35] Shoun Matsunaga et al. "MTJ-based nonvolatile logic-in-memory circuit, future prospects and issues". In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE. 2009, pp. 433–435.

[36] Alfred J Menezes et al. *Handbook of applied cryptography*. CRC press, 1996.

[37] Leonard Eugene Dickson. *Linear groups: With an exposition of the Galois field theory*. Courier Corporation, 2003.

[38] *Python Cryptography Toolkit*. https://pypi.org/project/pycrypto/.

[39] M Cormier et al. "Effect of electrical current pulses on domain walls in Pt/Co/Pt nanotracks with out-of-plane anisotropy: Spin transfer torque versus Joule heating". In: *Physical Review B* 81.2 (2010), p. 024407.

[40] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. "AES implementation on a grain of sand". In: *IEE Proceedings-Information Security* 152.1 (2005), pp. 13–20.

[41] Jens-Peter Kaps and Berk Sunar. "Energy comparison of AES and SHA-1 for ubiquitous computing". In: *International Conference on Embedded and Ubiquitous Computing*. Springer. 2006, pp. 372–381.

[42] Shin-Yi Lin and Chih-Tsun Huang. "A high-throughput low-power AES cipher for network applications". In: *2007 Asia and South Pacific Design Automation Conference*. IEEE. 2007, pp. 595–600.

[43] David D Hwang et al. "AES-Based Security Coprocessor IC in 0.18-*muhboxm* CMOS With Resistance to Differential Power Analysis Side-Channel Attacks". In: *IEEE Journal of Solid-State Circuits* 41.4 (2006), pp. 781–792.