POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Computing architectures based on skyrmions

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

Candidata:
Chiara Cannavò

Dicembre 2019

Acknowledgments

In the nine months that were needed for producing the results reported in this thesis, many people sustained me (and put up with me). First of all I want to thank professor Sandro Santucci from university of L'Aquila, for the patience he had in reading the first chapters of this work and in sending me a really appreciated feedback. I would like to thank my father, for all the support he gave me and for all the times he was available for a discussion. Finally, I would like to thank my mother, my sisters and all the friends that in these months gave me all the advices and sustain I could ever hope for.

Chiara Cannavò
Torino, December 2019

Table of contents

1. Introduction

It is well known that the critical issue encountered by Moore's law since the beginning of the new millennium is the power dissipation, linked to the blast of leakage current due to the continuous miniaturization of CMOS devices. Spintronics is a technological field that aims to use the electron spin properties, rather than its electric charge, for storing and manipulating information. For this reason it is expected to carry some great advantages with respect to traditional electronic devices, such as non-volatility and low power consumption.

In the spintronics field many different possibilities have been explored. In some cases, like for the bubble memories, some of them even became a commercial product. However, bubble memories soon became uncompetitive in terms of cost and completely disappeared from the market within a decade [20]. The main issues were the need of applying an external magnetic field for manipulating them, besides their not so small dimensions. As a result, the great strides made in improving semiconductor memories completely wiped them out.

Another example are the domain walls (DWs), which have already been extensively studied both from a theoretical and from an experimental point of view. Many designs have also been proposed, like the DW-based racetrack memories, where the binary data is encoded like a sequence of spin-up or spin-down magnetic domains separated by DWs. However, the main issue of DWs is the high current density required for allowing their movement (depinning current density) with a reasonable velocity, and this raises again the issue of the Joule heating and of the consequent power dissipation.

Skyrmions are one of the possibilities for carrying information that are being explored in the field of spintronics. They show many advantages with respect to domain walls, like a lower depinning current density and a smaller size, which allows a denser storage of information. Theoretical studies have in fact proven that the depinning current density for isolated skyrmions can be as low as 1×10^8 A/m^2, two or three orders of magnitude lower with respect to the depinning current density needed by domain walls [20]. Concerning its size, a skyrmion can have a diameter of only few nanometres, while a domain wall is hardly below 30 nm [20]. Since the mechanism which allows their motion while applying a current density is the same as for DWs, also their velocities are comparable under the same current applied; however, since the smaller skyrmions' dimensions allow to pack the same amount of information in a smaller space, even when moving at a lower velocity with respect to DWs they still allow to obtain a higher throughput. For this reason they are promising information carriers for future non-volatile, ultra-dense and low-power logic devices and memories.

Chapter 2 of this thesis tries to sum up the main characteristics and physical laws involved in the manipulation of skyrmions. A number of applications are also briefly presented, either to offer a more practical interpretation of the physical phenomena described, or to show the possibilities offered by the use of skyrmions as information carriers.

As proven by some of these applications, which have already appeared in literature, skyrmions can be exploited as information carriers both in logic gates and in racetrack memories. The goal of this thesis is to explore the possibilities offered by some of these designs when building a skyrmionic computing architecture. In particular, the model of logic gates used throughout the whole thesis comes from [5]. These logic gates, as it will be detailed more carefully in chapter 2, allow to perform the *AND*, *OR* and *NOT* elaboration of bits that are represented by skyrmions. The gates proposed in [5], however, were tested via micromagnetic simulations assuming a uniform current density throughout the gates, which is not a realistic assumption: the first aim of this thesis, then, is to prove that the behaviour of these structures is correct even when imposing the current density that should be found in reality inside the heavy metal layer of these gates. The results of the micromagnetic simulations performed with this goal can be found in chapter 3.

Once the behaviour of these gates is proven correct even under realistic assumptions, it makes sense to try to use them inside a more complex architecture. In chapter 4 the full adder proposed in [5] is used, slightly modified, to build a N-bit ripple carry adder. The structure of the adder is optimized in order to reduce as much as possible the energetic inefficiencies, which are due to the need of providing skyrmions that are not needed as information carriers but only as enable signals; the performances of the adder are analysed as well, by assuming for the main physical constants and parameters the values that were found in the micromagnetic simulations of chapter 3. The design of a complex structure allows to identify some of the main challenges that still need to be solved from a physical and technological point of view: the most crucial point, in fact, is the absence, in literature, of something that could behave like the electrical via in the PCB (printed circuit boards). To date, in fact, no solution that allows two nanotracks to cross has been proposed yet.

Finally, the architectural analysis continues on a different path. Chapter 5 starts from the study of a logic in memory (LiM) architecture that has already appeared in literature, and adapts it in order to find a memory array based on skyrmions with the same functionalities and that allows the same algorithms to be executed. The LiM architecture addressed in chapter 5 allows a large flexibility and adaptability to a wide range of different algorithms, apart from introducing a mechanism useful to reuse all those skyrmions that have already been nucleated but that are no more useful for the computations; the resulting structure, however, is very complex, heavy and not optimized.

Giving up on the flexibility and on the energy-saving structures that were introduced in chapter 5, a new LiM architecture is studied in chapter 6. Focusing on the execution of a specific algorithm (the minimum/maximum search algorithm proposed in [37]), the resulting skyrmionic memory array becomes much lighter and smaller, and the execution more efficient. The LiM array obtained, however, is also less flexible and less efficient from an energetic point of view.

2. Physics and applications of skyrmions

In the following the main physical laws and properties of skyrmions will be described, together with the issues in using them as information carriers. Some applicative aspects, that can either be useful to better understand the practical meaning of some physical properties, or that can be helpful in solving the main issues, will be presented too. The aim of this chapter is not to give an exhaustive and detailed insight into the complex physical phenomena linked to these topological configurations, but just to offer an overview on the principal difficulties and on the main mechanisms involved when trying to use skyrmions as information carriers inside computing architectures. In the end of the chapter, moreover, will be described some applications which exploit and sum up well all the physical characteristics exposed in the first part. Among them will be presented also the results that are the starting point for the architectural analysis that is the subject of this thesis.

2.1. Physical properties

To employ skyrmions as information carriers it is of vital importance the ability of performing some elementary tasks, including their creation, manipulation, detection and eventually annihilation. The first topic addressed in this section is the structure of a skyrmion and what are the main energetic contributions that compete in its creation. Then will be described the laws of motion and the main possibilities available to move it along the device; finally it will be detailed how is possible to nucleate it and to detect it. The topic of the skyrmion radius will be discussed at the end of the section, since it exploits some results presented in other papers.

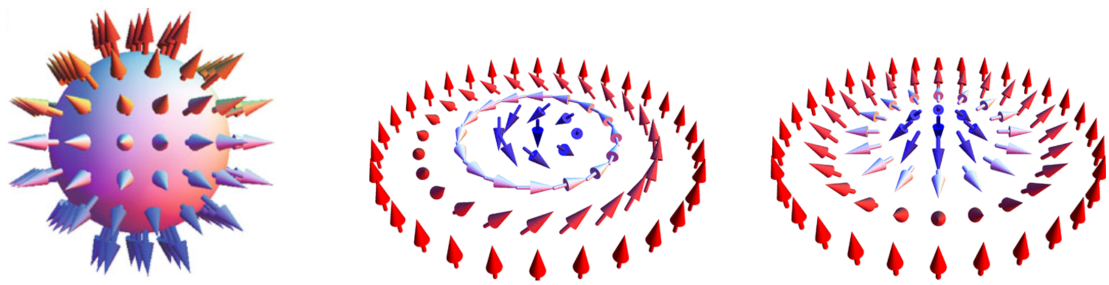
2.1.1. Topological properties

2.1.1.1. Topological charge

A magnetic skyrmion is a non-collinear 2D configuration of magnetic moments resulting from the competition of different energetic terms (which will be detailed in section 2.1.2.1). This configuration has a whirling structure, like shown in figure 2.1, and is described from a topological point of view by the topological charge Q, also called skyrmion number or Pontryagin number; its expression is [1]

$$Q = \frac{1}{4\pi} \int dxdy(\partial_x \boldsymbol{m} \times \partial_y \boldsymbol{m}) \cdot \boldsymbol{m} \tag{2.1}$$

where $\boldsymbol{m}$ is the unit vector representing the orientation of the local magnetic moment. The skyrmion number counts how many times the magnetic spins constituting the structure of the skyrmion can be wrapped around a unit sphere: all the spins at the boundary are collected into a single vector mapped on one pole of the sphere, and this is possible only because these spins point all in the same direction; the core is mapped at the opposite pole of the sphere, while the intermediate spins are mapped on the remaining parts of the sphere. In the case of a skyrmion-like structure, the topological charge is an integer equal to ± 1.



(a) (b) (c)

Figure 2.1. (a) Mapping of the Néel skyrmion onto the unit sphere. Figure extracted from [35]. (b) Bloch skyrmion, (c) Néel skyrmion. Figures extracted from [20].

[1]Like observed in [23], the sign in this formula is not consistently defined in literature. An example in which the topological charge is defined with a minus sign can be found in [43].

The magnetization of a skyrmion in polar coordinates [19] is described by

$$\boldsymbol{M} = M_S \begin{bmatrix} cos\phi(\varphi)sin\theta(r) \\ sin\phi(\varphi)sin\theta(r) \\ cos\theta(r) \end{bmatrix} \tag{2.2}$$

while the expression of $\boldsymbol{r}$ in polar coordinates is $\boldsymbol{r} = r(cos\varphi,sin\varphi)$. Inserting equation 2.2 into equation 2.1 it can be obtained that

$$Q = \frac{1}{4\pi}\int_0^\infty dr \int_0^{2\pi} d\varphi \frac{d\theta(r)}{dr}\frac{d\phi(\varphi)}{d\varphi}sin\theta(r) = \frac{1}{4\pi}cos\theta(r)\Big|_{r=0}^{r=\infty}\phi(\varphi)\Big|_{\varphi=0}^{\varphi=2\pi} \tag{2.3}$$

This equation will be commented later in section 2.1.1.3.

The in-plane magnetization angle ϕ is assumed to be a linear function of the azimuthal angle φ [19], so that

$$\phi = m\varphi + \gamma \tag{2.4}$$

This equation will be discussed later in section 2.1.1.5. For now is enough to remember this: let's assume that the skyrmion's structure lays in the xy-plane; the magnetic moments are organized on circumferences that are concentric with the skyrmion core (of course in first approximation and only if the skyrmion is not being deformed); the angle giving the position of each magnetic moment along the corresponding circumference with respect to the x-axis is φ, with unit vector $\hat{\varphi}$. Then, ϕ is the angle that the spin at position φ has with respect to the unit vector $\hat{\varphi}$. For example, in the skyrmion represented in figure 2.1c, each moment has angle $\phi = 0$, while in the skyrmion shown in figure 2.1b each moment has an angle $\phi = \frac{\pi}{2}$.

2.1.1.2. Dzyaloshinskii-Moriya Interaction

The Dzyaloshinskii-Moriya Interaction (DMI) is a key element for the stabilization of magnetic skyrmions. The bulk DMI comes from the breaking of the bulk inversion symmetry ($\boldsymbol{r} \nrightarrow -\boldsymbol{r}$) and from the presence of atoms with high spin-orbit coupling in ferromagnetic alloys (for example B20 materials). The interfacial DMI (i-DMI) comes instead from the breaking of the structure inversion symmetry

($z \nrightarrow -z$) at the interfaces of a multilayer system, where a thin layer of ferromagnetic material is deposited above a substrate made of a material with large spin-orbit coupling (for example Co on Pt) [9, 13].

These two types of DMI give rise to two different types of skyrmion structures, both shown in figure 2.1: the Bloch skyrmion (also spiral skyrmion) and the Néel skyrmion (also hedgehog skyrmion). In both cases there is a central domain and an outer domain, both with out-of-plane magnetization, separated by a domain wall. When the domain wall has a circular chirality (either clockwise or counterclockwise) the skyrmion is a Block skyrmion: in this case the magnetization rotates in the tangential plane while going from the core to the tail of the skyrmion. If the domain wall has a radial chirality (either inward or outward), then the skyrmion is of Néel type and the magnetization rotates in the radial plane [9, 20].

The DMI is an interaction between two magnetic spins mediated by the presence of a third non-magnetic atom with a strong spin-orbit coupling (SOC). The Hamiltonian of this interaction is

$$H_{DM} = -\boldsymbol{D}_{12} \cdot (\boldsymbol{S}_1 \times \boldsymbol{S}_2) \tag{2.5}$$

where $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ are the two magnetic spins and $\boldsymbol{D}_{12}$ is the DMI vector, perpendicular to the plane containing the three atoms involved. Starting from a ferromagnetic state with $\boldsymbol{S}_1$ parallel to $\boldsymbol{S}_2$, the DMI tilts $\boldsymbol{S}_1$ with respect to $\boldsymbol{S}_2$ by a rotation around $\boldsymbol{D}_{12}$ [8]. This energetic term is minimized when the two magnetic spins are perpendicular to each other. At the same time, however, the exchange energy (detailed in section 2.1.2.1) in a ferromagnetic material is minimized when all the magnetic spins are aligned with each other: skyrmions are the result of the competition of these two mechanisms and of the minimization of the energy inside the system, like explained in section 2.1.2.1. The Néel skyrmion is the configuration minimizing the energy when $\boldsymbol{D}_{12} \perp \boldsymbol{R}_{12}$, the Bloch skyrmion instead minimizes the energy for $\boldsymbol{D}_{12} \parallel \boldsymbol{R}_{12}$ [8], like shown in figure 2.2c, where $\boldsymbol{R}_{12}$ is the vector joining spin $\boldsymbol{S}_1$ with spin $\boldsymbol{S}_2$.

The asymmetry due to the presence of the non-magnetic atom in the lattice or at the interface with the bottom metal layer is needed because, in this way, the DMI cannot be compensated by the DMI coming from a symmetric triangle [8].

The magnitude and sign of the vector $\boldsymbol{D}_{12}$ depend on the materials involved, on the interface and on the strength of the spin-orbit coupling of the non-magnetic

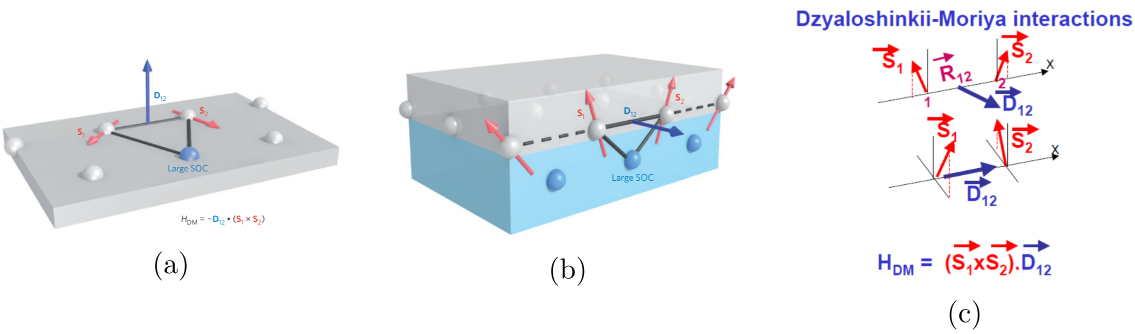


Figure 2.2. (a) DMI vector generated by the interaction of two atomic spins with an atom with strong SOC (blue) in a ultrathin magnetic film. (b) DMI vector generated at the interface between a FM thin layer (grey) and a metal with strong SOC (blue). Figures extracted from [8]. (c) The top configuration gives rise to a Néel skyrmion, the bottom configuration to a Bloch skyrmion. Figure extracted from [1].

atom, and has an important role in determining the size of the resulting skyrmion, like detailed in section 2.1.6.

Spin-orbit coupling

While considering the orbiting motion of the electron around the proton from the point of view of the electron, the same atomic system can be seen like the proton orbiting around the electron. Due to Ampere's law, this positive charge motion generates a magnetic field, which of course will interact with the magnetic moment (proportional to the spin) of the electron: this phenomenon is known as spin-orbit coupling (SOC).

2.1.1.3. Vorticity number

Another parameter useful in characterizing the skyrmion topological structure is the vorticity number m, defined as the winding number of the spin configurations projected into the xy-plane [20]. The winding number of an oriented curve counts how many times the curve encircles a well defined point in a plane in the counter-clockwise direction. Describing this curve in polar coordinates (r,θ), the winding number can be computed like

$$W = \frac{\theta(end) - \theta(start)}{2\pi} \tag{2.6}$$

Since the initial and the final position, when following the full path of the curve around the point, must be coincident, then the two θ angles must differ by an integer multiple of 2π: this implies that the winding number is either a positive or negative integer.

In the specific case of magnetic topological structures, the winding number can be defined also like the total variation of the magnetization angle when moving counterclockwise along a circle traced around the centre of the structure, divided by 2π [30, 1]. Each value of the magnetization angle can be mapped into a point on the circle, which in this context is formally called order-parameter space. According to the oriented movement along the order-parameter space, the winding number is computed.

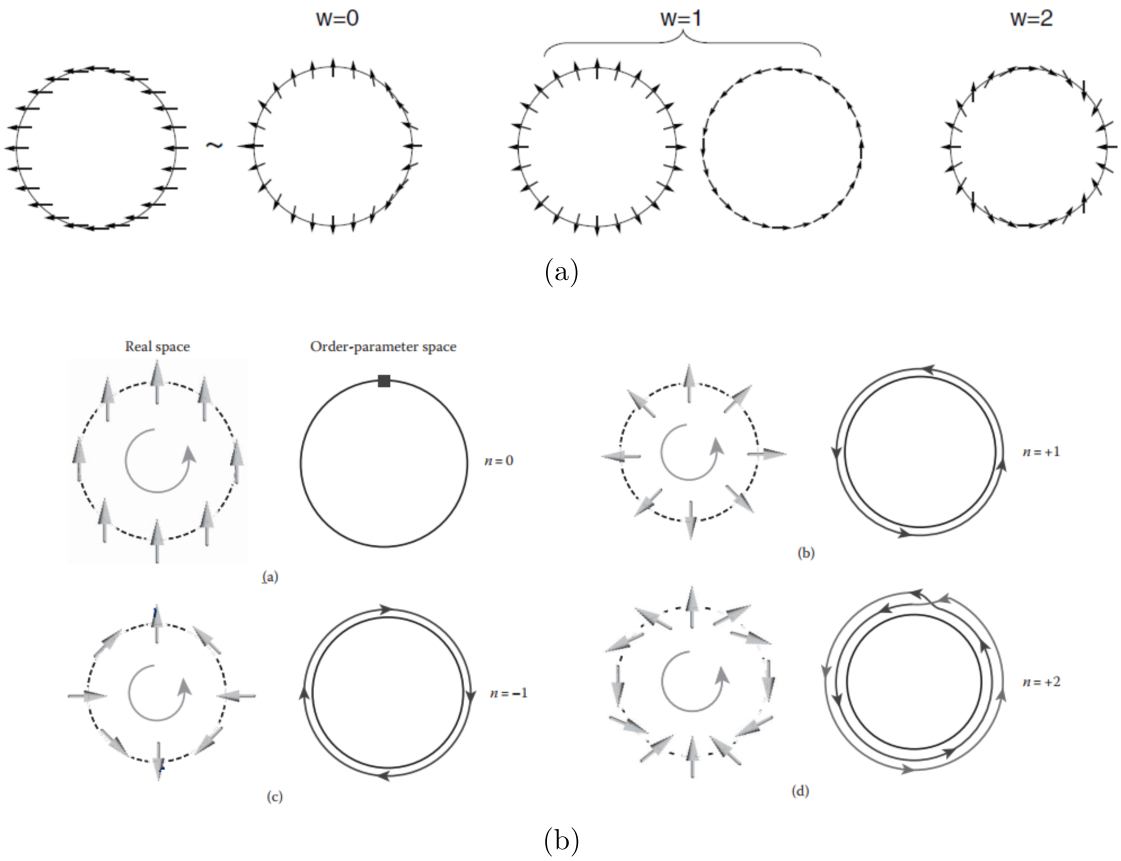


Figure 2.3. (a) Examples of computation of the winding number for different spin configurations. The two cases with $W = 1$ can be assimilated to a Néel skyrmion (left) and to a Bloch skyrmion (right). Figure adapted from [3]. (b) Other examples of computation of the winding number, including the mapping on the order-parameter space. Figure extracted from [24].

The skyrmion number can be computed also through the vorticity number, using the relation

$$Q = \frac{m}{2}\left[\lim_{r\to\infty} cos(\theta(r)) - cos(\theta(0))\right] \tag{2.7}$$

which can be recognized simply like a rewriting of equation 2.3, where $m = \frac{1}{2\pi}\phi(\varphi)|_{\varphi=0}^{\varphi=2\pi}$ is the vorticity that has just been defined. Observing 2.2 it can be deduced that $cos(\theta(r))$ is the z-component of the unit vector representing the local magnetic moment. So, the skyrmion number depends both on the vorticity and on the direction of the magnetization at the tail and at the core of the skyrmion. Is important to notice that, in a skyrmion, the direction of the core is always opposite to the direction of the tail, which is the same of the background magnetization present in the material.

2.1.1.4. Helicity number and polarity

The third parameter needed to describe the skyrmion structure is the helicity number γ, determined uniquely by the type of DMI that intervenes in the energetic competition. A Bloch skyrmion (bulk DMI) is characterized by $\gamma = \frac{\pi}{2}$ or $\gamma = \frac{3\pi}{2}$, while a Néel skyrmion (interfacial DMI) corresponds to $\gamma = 0$ or $\gamma = \pi$. It's important to notice that the helicity does not contribute to the topological number.

Finally, the polarity p describes the orientation of the centre with respect to the z-direction: if $p = 1$ the magnetization of the core points in the positive z-direction, vice versa if $p = -1$.

2.1.1.5. Meaning of the helicity number

Looking back at equation 2.4, which expresses the in-plane magnetization angle ϕ, it can be recognized now that m is the vorticity number and γ the helicity number, while φ is the azimuthal angle of the polar coordinates system, describing the position of a point in the xy-plane. For both Bloch and Néel skyrmions the vorticity number is equal to 1, like it can be observed comparing figure 2.3a with the two structures shown in figure 2.1. In section 2.1.1.4 it has been said that for a Bloch skyrmion $\gamma = \frac{\pi}{2}$ or $\gamma = \frac{3\pi}{2}$, while for a Néel skyrmion $\gamma = 0$ or $\gamma = \pi$. Summing up:

Néel:

$$\begin{cases} \phi = \varphi \\ \phi = \varphi + \pi \end{cases} \tag{2.8}$$

Bloch:

$$\begin{cases} \phi = \varphi + \frac{\pi}{2} \\ \phi = \varphi + \frac{3\pi}{2} \end{cases} \tag{2.9}$$

From these expressions is easy to predict one of the experimental results reported in section 2.1.1.7: if $\gamma = 0$ the in-plane magnetization is aligned with the unit vector $\hat{\varphi}$ and the spins will be out-going, while if $\gamma = \pi$ the magnetization is antiparallel with respect to $\hat{\varphi}$ and the spins will be in-going. Similarly, if $\gamma = \frac{\pi}{2}$ the in-plane magnetization has a 90° phase difference with respect to $\hat{\varphi}$ and the spins of the Bloch skyrmion will rotate in a counterclockwise direction, while if $\gamma = \frac{3\pi}{2}$ the phase difference is by −90° and the spins rotate in a clockwise direction. The confirmation to these statements can be easily found in the top half of figure 2.4.

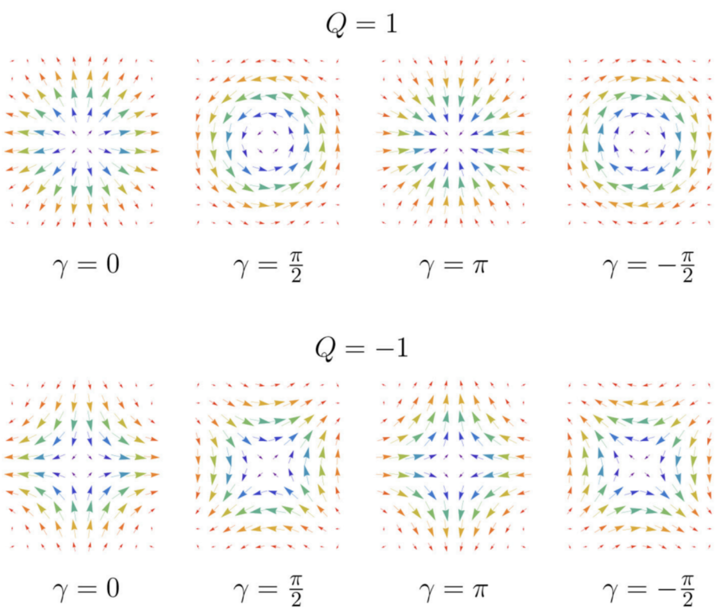


Figure 2.4. Magnetization configuration of skyrmions ($Q = 1$) and antiskyrmions ($Q = -1$) with fixed polarity ($p = 1$) and varying helicity γ. The length and direction of the arrows represents the in-plane magnetization component, while the colour represents the magnitude of the out-of-plane component: blue stands for $+z$, red for $-z$. Figure extracted from [12].

2.1.1.6. Topological protection

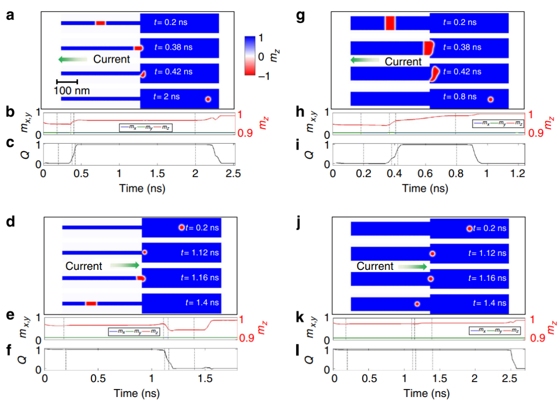
The topological charge of skyrmions is always an integer equal to +1, at least if this particle is in a region larger than its diameter. For this reason, even if its spin texture may be deformed (for example due to the presence of impurities in the material), the Pontryagin number doesn't change and as a result the skyrmion can be neither destroyed nor separated into pieces: it is said to be topologically protected. This protection fails only when the skyrmion touches the sample edges, because in this condition the topological charge is allowed to change continuously: in this condition the skyrmion can be annihilated and the information it carries gets lost.

2.1.1.7. Practical examples

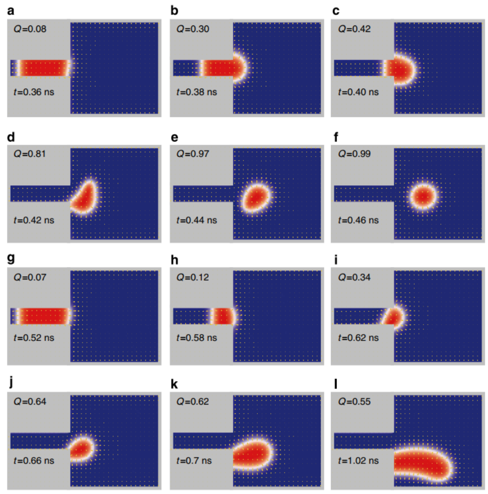
Some examples, which show well the practical meaning of all the parameters presented up to this point, can be found in [40], where the results presented in [43] are exploited.

In [43] is demonstrated the possibility to convert in a reversible way a domain wall (DW) pair into a skyrmion and vice versa by using a junction made of a narrow nanowire ($W < 2R$, where W is the width of the nanotrack and R the skyrmion radius) connected to a wide nanowire ($W > 2R$). Like explained in [40], when a topological object, like a skyrmion, that previously was in a wide region enters a narrow region, it loses its topological numbers, like the skyrmion number and the helicity, and becomes a DW pair, which is a non-topological object ($Q = 0$); when it is ejected again into a new wide region, it is assigned new topological numbers to adjust the physical properties of the new region, which may be different from the properties of the former wide region. Skyrmions, as mentioned, are topologically protected only when the sample is enough large, and this protection is broken when they touch an edge: this is what happens when the skyrmion enters the narrow junction.

Another result of [43] that has been used in [40] (like explained in section 2.2.1) is shown in figure 2.5b (g-l frames): using the same junction but reducing the current density needed to move the domain wall, the particle obtained is not a skyrmion, but a meron. A meron is a different type of topological object, very similar to a skyrmion, but with $|Q| = \frac{1}{2}$: this means that its spins can wrap only the north pole



(a)



(b)

Figure 2.5. (a) Reversible conversion DW pair→skyrmion (a-c) and vice versa (d-f). g-l: when the size of the narrow part of the junction increases the DW pair is still converted into a skyrmion, but with more fluctuations in the radius and the shape (g-i); the skyrmion instead cannot be converted back into a DW pair (j-l). The middle plots show the time variation of the components of the magnetization; the bottom plots show the time variation of the skyrmion number. (b) Detail on the conversion from DW pair. a-f: when the current density is high enough the right DW pins at the junction, while the remaining part of the DW pair continues to move: as a result, the DW is deformed into a curve shape. When the other DW reaches the junction the skyrmion is formed. g-l: when the current density is reduced a meron is formed; the meron remains in contact with the nanotrack edge. Figures extracted from [43].

or the south pole of the unit sphere, like shown in figure 2.6. Its peculiarity is to remain attached to the sample edges even during its movement.

The first part of [40] verifies how different physical properties of the materials on the left and on the right side of a narrow nanotrack influence the assignment of new topological numbers to the skyrmion ejected from the junction, and allows to understand the practical meaning that these numbers have when considering the skyrmion's structure.

In figure 2.7a the parameter that changes from left to right is the sign of the DMI, which is positive on the left side and negative on the right side. As mentioned in section 2.1.1.4, the helicity is uniquely determined by the DMI. Since the skyrmions considered in this article are only of Néel type, the helicity can be either $\gamma = 0$ or $\gamma = \pi$. On the left side the triplet of numbers (Q,m,γ) (topological charge, vorticity

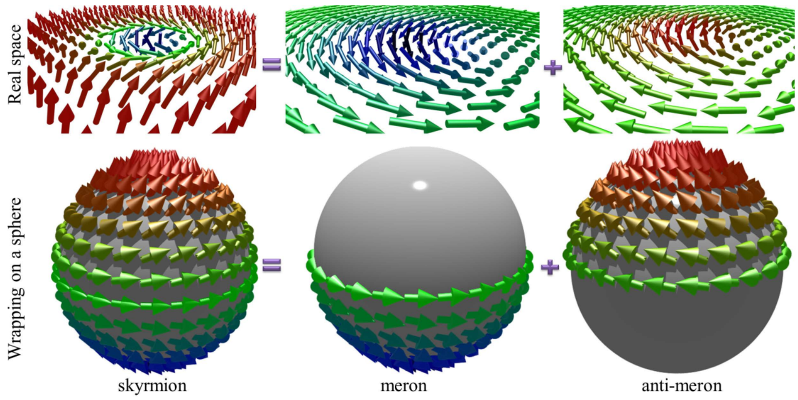
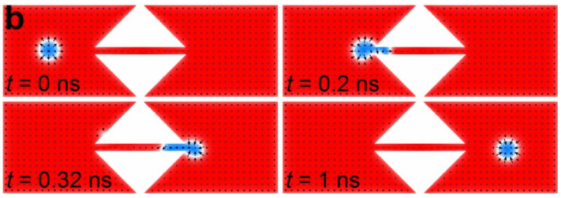


Figure 2.6. Splitting a skyrmion in two halves, a meron and an anti-meron are obtained. A skyrmion covers the entire unit sphere, while a meron covers only half of it: so the topological charge of a meron is $\pm\frac{1}{2}$. Figure extracted from [22].

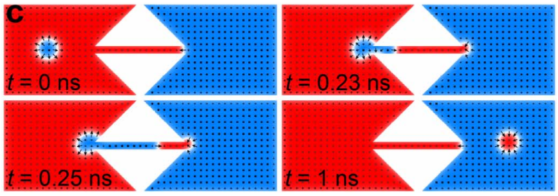
and helicity; this notation will be used from now on in all the rest of this thesis) is (1,1,0), whereas on the right side it is (1,1,π) (while in the junction it becomes (0,0,0)). Observing the picture, it can be noticed that the direction of the core has remained $-z$, while the spins from out-going have become in-going. So the helicity, like anticipated in section 2.1.1.5, determines the radial direction of the spins: when $\gamma = 0$ the spins point outwards, when $\gamma = \pi$ they point inwards.

In figure 2.7b the sign of the DMI is the same in both regions, while the direction of the background magnetization is reversed from left to right. According to equation 2.7, the topological charge depends both on the vorticity and on the direction of the spins at the core and at the tail of the skyrmion, and the spin direction at the tail is always the same as the background magnetization. Reversing the background magnetization then the second factor of 2.7 is reversed (from the picture it can be observed that the core now points in the $+z$ direction and the tail in the $-z$ direction), while the vorticity remains unchanged: as a consequence the topological number becomes $Q = -1$ and the magnetic texture obtained is called antiskyrmion. Moreover, in this condition also the helicity changes from $\gamma = 0$ to $\gamma = \pi$ to minimize the energy of the system. So, the topological numbers change as (1,1,0) $\rightarrow$ (-1,1,π).

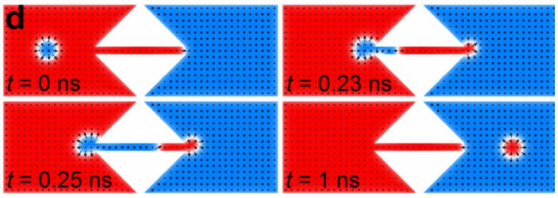
Finally, in 2.7c both the sign of the DMI and the background magnetization are reversed from left to right. Combining the two effects discussed above, then, the skyrmion becomes an antiskyrmion due to the background reversal, while the helicity remains unchanged and so the spin direction remains out-going. The change of the topological numbers is (1,1,0) → (−1,1,0)



(a)



(b)



(c)

Figure 2.7. Skyrmion→DW pair→skyrmion conversion. Red(blue) is the $+z(-z)$ direction of the magnetization. (a) Reversing the sign of the DMI (1,1,0) → (1,1,π) (b) Reversing the background magnetization (1,1,0) → (−1,1,π) (c) Reversing both DMI and background magnetization (1,1,0) → (−1,1,0). Figures extracted from [40].

Skyrmions and antiskyrmions

Like mentioned, antiskyrmions are spin textures very similar to skyrmions, but with topological charge equal to −1 [2]. They can still be mapped on a unit sphere, like shown in figure 2.8. Some examples of antiskyrmions can be found also in the bottom half of figure 2.4, where is shown the disposition of the in-plane magnetization by varying the helicity.

According to the examples just discussed and to the theory exposed up to now, the skyrmion and antiskyrmion main figures are summed up in table 2.1 (if the background magnetization is along $+z$, then $\lim_{r\to\infty} cos(\theta(r)) - cos(\theta(0)) = 2$, while if it is along $-z$ $\lim_{r\to\infty} cos(\theta(r)) - cos(\theta(0)) = -2$).

[2]Some attention must be paid in the definition of antiskyrmions. Many examples in literature define as antiskyrmions those particles with $Q = -1$, while other authors consider antiskyrmions those particles that have $m = -1$, usually regardless of the topological charge. This inconsistency is evident when reading [40] and its supplementary information, [21], [13], [25] and [4]. In this thesis we conform ourselves to the first definition.

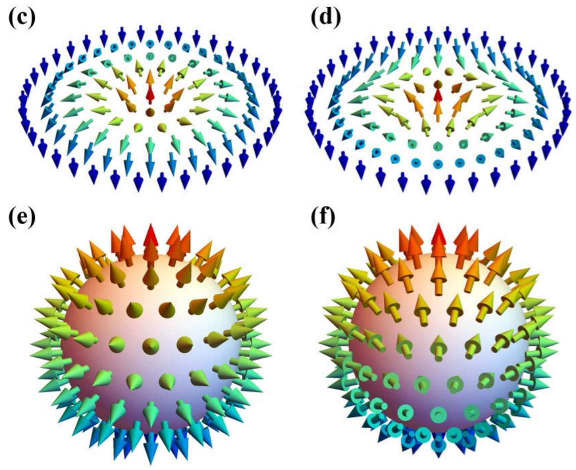


Figure 2.8. Mapping of a skyrmion (1,1,0) and of an antiskyrmion (−1, − 1,0) with $p = 1$ on the unit sphere (order-parameter space). Figure extracted from [14].

Table 2.1. Summary of topological numbers for skyrmions and antiskyrmions

	BACKGROUND: $+z$	BACKGROUND: $-z$
SKYRMION	$Q = 1$, $m = 1$	$Q = 1$, $m = -1$
ANTISKYRMION	$Q = -1$, $m = -1$	$Q = -1$, $m = 1$

The confirmation to this table can be found in the supplementary information of [40], where the figure 2.9 is shown.

2.1.2. Micromagnetic model

The static properties of skyrmions can be studied theoretically with the help of the micromagnetic model, a theory used to describe the magnetization of a material from the nanoscale to the microscale. This length scale is large enough for avoiding the use of all the mathematical operators required by quantum mechanics, but at the same time it is small enough to carefully describe magnetization patterns like skyrmions or domain walls, among the others [19]. This theory allows to model the relationship between the spatial distribution of the effective magnetic field $\boldsymbol{H}_{eff}$, determined by the energetic contributions competing in the system, and the magnetization vector field $\boldsymbol{M}$ [9]. The key equation on which the micromagnetic model is based is the Landau-Lifshitz-Gilbert (LLG) equation.

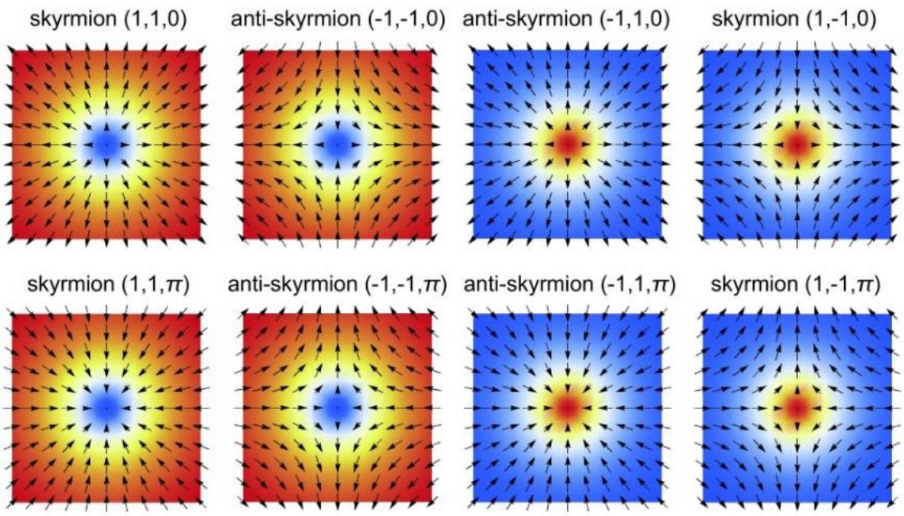


Figure 2.9. Examples of skyrmions and antiskyrmions by varying vorticity, helicity and background magnetization (and so polarization). Each triplet of numbers reports (Q,m,γ); red(blue) denotes the $+z(-z)$ direction of the magnetization. Figure extracted from the supplementary information of [40].

2.1.2.1. LLG equation

This equation relates the effective field $\boldsymbol{H}_{eff}$ to the time evolution of the magnetization vector field $\boldsymbol{M}$. It is a torque equation, and its expression is

$$\frac{d\boldsymbol{M}}{dt} = -\gamma_0 \boldsymbol{M} \times \boldsymbol{H}_{eff} + \alpha \left(\boldsymbol{M} \times \frac{d\boldsymbol{M}}{dt} \right) \tag{2.10}$$

where $\boldsymbol{M}$ is the magnetization, $\boldsymbol{H}_{eff}$ is the effective magnetic field (not necessarily an external magnetic field, it can be also the field experienced locally by the magnetic moments inside the material), γ_0 is the gyromagnetic ratio and α the Gilbert damping coefficient. The first term describes the precession movement that the magnetic moments perform around the effective magnetic field when they are not fully aligned with it: this is known as Larmor precession. While performing this precession movement the magnetization also relaxes along the field line, finally becoming aligned with it, in order to minimize the energy of the system: this is modelled by the second term, containing the Gilbert damping.

It can be demonstrated [19] that only the direction of the magnetization changes with time, while its magnitude remains constantly equal to the saturation magnetization M_S.

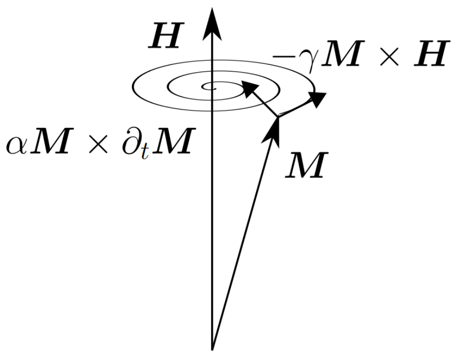


Figure 2.10. Sketch of the Larmor precession performed by a magnetic moment around a field line of $\boldsymbol{H}_{eff}$, and of the Gilbert damping contribution which, determining a spiraling motion around the field line, eventually makes the magnetic moment align with it. Figure extracted from [19].

The direction of the effective magnetic field is the direction in which the magnetization will have the minimum of the micromagnetic energy: therefore the effective field can be written in terms of the micromagnetic energy [19] as

$$\boldsymbol{H}_{eff} = -\frac{1}{\mu_0}\frac{\partial E_V}{\partial \boldsymbol{M}} \tag{2.11}$$

where E_V is the micromagnetic energy density.

Micromagnetic energy density

The contributions to the micromagnetic energy come from the exchange energy, the Zeeman energy, the demagnetizing field energy, the anisotropy energy and the DMI. Also other terms could be considered, like the RKKY (Ruderman-Kittel-Kasuya-Yosida) interaction, but when considering skyrmions in most of the cases they are neglected, so they won't be considered here.

Exchange energy The exchange interaction (sometimes also Heisenberg interaction) is the phenomenon which makes the magnetic moments inside a ferromagnetic material align with each other, allowing them to generate a magnetic field observable from the external. If the sign of the exchange constant J present in the Hamiltonian of this interaction is positive, then the material is a ferromagnetic material and the spins align parallel to each other; if the sign of J is negative, instead, the material is an antiferromagnet and the spins arrange antiparallel to each other.

The energy density of the exchange interaction is [20]

$$E_V = A[\nabla \boldsymbol{m}]^2 \tag{2.12}$$

where A is the exchange constant.

Zeeman energy When applying an external magnetic field, the magnetic moments tend to align with its field lines: this is due to the Zeeman interaction. Its energy density is [43]

$$E_V = -\mu_0 \boldsymbol{M} \cdot \boldsymbol{H}_{ext} \tag{2.13}$$

where $\boldsymbol{H}_{ext}$ is the magnetic field applied externally.

Demagnetizing field energy The magnetic induction is $\boldsymbol{B} = \mu_0(\boldsymbol{H} + \boldsymbol{M})$, where $\boldsymbol{H}$ is the magnetic field and $\boldsymbol{M}$ is the magnetization. Since $\boldsymbol{M} = \chi \boldsymbol{H}$, where $\chi = \mu_r - 1$ is the susceptibility, then $\boldsymbol{B} = \mu_0(1 + \chi)\boldsymbol{H} = \mu_0 \mu_r \boldsymbol{H}$.

Inside a magnetic material $\boldsymbol{H}$ is directed oppositely with respect to $\boldsymbol{M}$, like shown in figure 2.11. For this reason, when considering the value of $\boldsymbol{B}$ inside the material, $\boldsymbol{H}$ contributes by reducing this value. This is why it is called demagnetizing field (while it is named stray field outside the material).

The energy density associated to the demagnetizing field $\boldsymbol{H}_d$ is

$$E_V = -\frac{\mu_0}{2} \boldsymbol{M} \cdot \boldsymbol{H}_d(\boldsymbol{M}) \tag{2.14}$$

Anisotropy energy The magnetocrystalline anisotropy is the property of some magnetic materials which makes some directions for the magnetization more energetically favourable than others; it is is linked to the spin-orbit coupling and to the atomic structure of the material. As a result, some directions may be easier to be magnetized than others. In particular, when this energetically favourable direction (called easy-axis) is perpendicular to the material, then we're dealing with the perpendicular magnetic anisotropy (PMA). The PMA can be found in some layered ultrathin films, like for example Pd/Co [19].

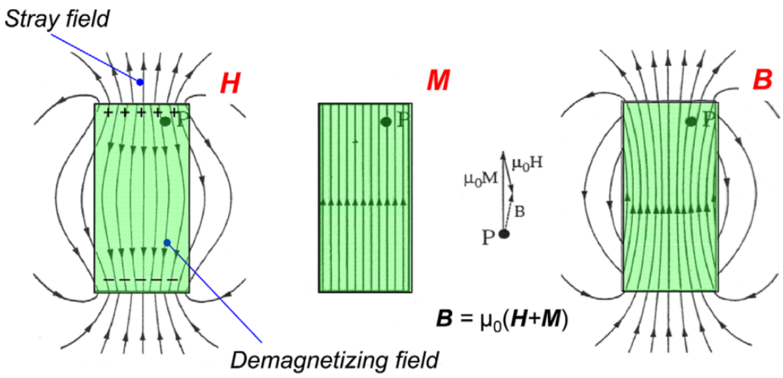


Figure 2.11. Depiction of the magnetic field $\boldsymbol{H}$, the magnetization $\boldsymbol{M}$ and the magnetic induction $\boldsymbol{B}$ inside a magnetic material. The magnetic field $\boldsymbol{H}$ takes the name of demagnetizing field inside the material and of stray field outside it. Figure adapted from [7].

The energy density for the PMA is [43]

$$E_V = K_u[1 - (\boldsymbol{m} \cdot \hat{\boldsymbol{z}})^2] \tag{2.15}$$

where K_u is the uniaxial anisotropy constant and the unit vector $\hat{\boldsymbol{z}}$ represents the easy-axis.

In layered structures of ultrathin films, where the surface effects are not negligible, a form of PMA arises at the interface due to exchange interactions between some of the electronic orbitals of the materials involved. It has been discovered that the strength of this PMA can be modified by applying an external electric field: this field modifies the occupation of the orbitals and so is able to affect the exchange interactions [19].

As a result, if K_u is the PMA constant, the voltage controlled PMA will induce a variation $\Delta K_{uv}E$ dependent on the electric field applied, so that the expression of the total PMA constant is [42]

$$K_{uv} = K_u + \Delta K_{uv}E \tag{2.16}$$

Of course, according to the sign of the z-component of the electric field, the PMA can either increase or decrease with respect to its bias level K_u.

DMI energy The bulk DMI energy density is [19]

$$E_V^{(bulk)} = D\boldsymbol{m} \cdot (\nabla \times \boldsymbol{m}) \tag{2.17}$$

The interfacial DMI energy density is instead (assuming to consider an ultrathin film, where $\frac{\partial \boldsymbol{m}}{\partial z} = 0$) [43]

$$E_V^{(interfacial)} = D[m_z(\nabla \cdot \boldsymbol{m}) - (\boldsymbol{m} \cdot \nabla)m_z] \tag{2.18}$$

Summing all these energy density contributions together the total energy density $E_{V,TOT}$ is obtained. The total energy of a system of volume V is

$$E_{TOT} = \int_V E_{V,TOT} dV \tag{2.19}$$

The spin configuration then is found by minimizing the total energy E_{TOT} [1].

2.1.3. Motion

There are many possibilities both for nucleating and for moving skyrmions: some of them consist in applying an external magnetic field or temperature gradients. However, is clear than none of these options can be easily used inside an integrated circuit based on skyrmions, where it would be preferable to use current-based mechanisms instead.

When it comes to current, there are mainly two mechanisms available, both exploiting the spin-transfer-torque phenomenon.

2.1.3.1. Spin Transfer Torque (STT)

When a charge current is injected into a material with a certain magnetization pattern, the spin of each conduction electron will interact with the magnetization vector field. This interaction leads to the formation of two torques: one tends to align the spin of the electron with the direction of the local magnetic moment, while the other, equal and opposite due to the conservation of the total angular momentum, at the same time tries to align the local magnetization with the direction of the electron spin. This is the spin-transfer-torque (STT) mechanism.

The STT contributes in modifying the orientation of the local magnetic moment: for this reason it can be included in the LLG equation. Since the current flows inside the ferromagnet along the in-plane direction, this torque is indicated as $\boldsymbol{\tau}_{IP}$. It is composed by two contributions. The first is the adiabatic STT, where adiabatic refers to the assumption that the spin of the electron passing through the magnetic material relaxes fast enough so that is always aligns with the local magnetic moment [19]. The second term of $\boldsymbol{\tau}_{IP}$ has been added phenomenologically to explain unexpected experimental results and it is the non-adiabatic STT: the adiabatic approximation fails when the magnetization pattern changes so quickly in space that the electrons are not fast enough to align their spin with the local magnetic moment.

The expression of $\boldsymbol{\tau}_{IP}$ is

$$\boldsymbol{\tau}_{IP} = \frac{\gamma_0 \hbar P}{2\mu_0 e M_S}(\boldsymbol{j} \cdot \nabla)\boldsymbol{m} - \frac{\gamma_0 \hbar P}{2\mu_0 e M_S}\beta \boldsymbol{m} \times (\boldsymbol{j} \cdot \nabla)\boldsymbol{m} \tag{2.20}$$

where the first term is the adiabatic STT and the second term is the non-adiabatic STT. Here γ_0 is the gyromagnetic ratio, P is the polarization coefficient of the in-plane electrical current, e is the electron charge, M_S is the saturation magnetization, $\boldsymbol{j}$ is the in-plane electrical current flowing through the ferromagnet, $\boldsymbol{m}$ is the normalized magnetization, and β is the non-adiabaticity factor, quantifying the relative strength of the non-adiabatic STT with respect to the adiabatic STT.

The term $\boldsymbol{\tau}_{IP}$ is added to the LLG equation when a current is flowing along the in-plane direction of a ferromagnet. The geometry used for the skyrmion motion that uses this effect is then called current-in-plane (CIP) geometry and is shown in figure 2.12a.

As mentioned, the spins of the conduction electrons exert a torque on the spin texture of the skyrmion; at the same time, the spin texture exerts a torque equal in magnitude and opposite in sign on the spin of the conduction electrons. As a result, the magnetic moments of the skyrmion subjected to the torque will rotate, allowing the movement of the particle, and at the same time the conduction electrons are deflected from the original direction of the current flux: this is known as topological

Hall effect, and the reason behind this is the Berry phase.

The Berry phase is the rotation that a vector experiences while moving along a closed path on a curved surface. The skyrmion, as already mentioned, is a 2D structure, but the magnetic moments that constitute it can be organized on a unit sphere: assuming that the conduction electrons are able to follow exactly the orientation of the local magnetic moment of the skyrmion (adiabatic approximation), while crossing the particle they gain a Berry phase. This Berry phase is the reason behind the emergent magnetic field experienced by the conduction electrons. This field will make the conduction electrons experience a Lorentz force

$$F = q(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{2.21}$$

If the skyrmion texture is localized in the xy plane, the emergent field points along the z-axis, so the Lorentz force belongs to the xy plane and is perpendicular to the motion of the conduction electrons, making them deflect from their original direction [19].

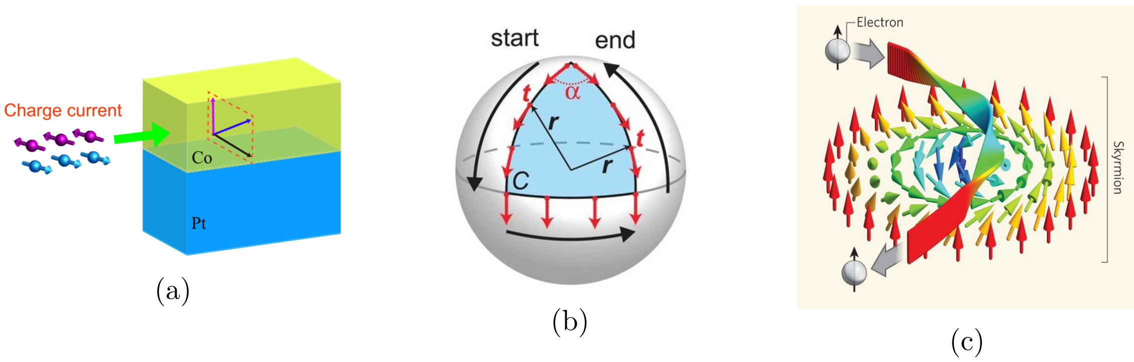


Figure 2.12. (a) CIP configuration: a spin-polarized current flows directly inside the FM layer (yellow) deposited above the HM layer (blue). Figure extracted from [20]. (b) Sketch of the Berry phase α acquired by a vector t (depicted in red) in the movement along the oriented curve on the curved surface of a sphere. Figure extracted from [11]. (c) Effect of the torque exerted by the skyrmion texture on the spin of the conduction electron and depiction of the topological Hall effect. Figure extracted from [32].

2.1.3.2. Spin Hall Effect (SHE)

The spin Hall effect is a phenomenon that originates in spin-Hall devices, where a ferromagnetic (FM) thin film is deposited above a heavy metal (HM) substrate, like shown in figure 2.13. Here the electrical current is injected inside the HM layer. Due to spin-dependent scattering mechanisms (which include also SOC [29]), the electrons will experience a deflection perpendicular to their flow direction and to the orientation of their spin. As a result, the SHE leads to an accumulation of charges at the sides of the wire, and each side is populated by electrons with a well defined spin orientation. For example, if the current flows in the $+x$-direction and if the anomalous velocity acquired by the electrons is directed along the $+z$-axis, making them accumulate at the top surface of the wire, their spin orientation will be along the $+y$-axis. Moreover, 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 [26, 29].

Since this time the current that interacts with the magnetization is directed perpendicularly to the film plane, this configuration is called current-perpendicular-to-plane (CPP) geometry.

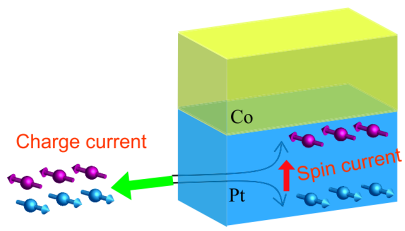


Figure 2.13. CPP configuration: an electrical current flows inside the HM layer (blue); the spin-Hall effect determines the creation of a spin-polarized current directed in the vertical direction that is collected by the FM layer (yellow) deposited above. Figure extracted from [20]

The expression of the torque $\boldsymbol{\tau}_{SHE}$ associated to this phenomenon is

$$\boldsymbol{\tau}_{SHE} = -\frac{\gamma_0 \hbar j \theta_{sh}}{2\mu_0 e M_S t_f} \boldsymbol{m} \times (\boldsymbol{m} \times \boldsymbol{p}) \tag{2.22}$$

where t_f is the thickness of the FM layer, $\boldsymbol{p}$ is the spin-current polarization direction ($+y$ in the example reported above), j is the current density and θ_{sh} is the spin Hall angle. The spin Hall angle measures the efficiency of the conversion from charge current J_{ch} to spin current J_s and depends on material parameters [39]. Its expression is

$$\theta_{sh} = \frac{J_s}{J_{ch}} \tag{2.23}$$

2.1.3.3. Thiele equation and skyrmion Hall effect

When assuming that the skyrmion moves without deforming its texture, the time-dependent evolution of the magnetization pattern can be written like

$$\boldsymbol{M}(\boldsymbol{r},t) = \boldsymbol{M}_0[\boldsymbol{r} - \boldsymbol{R}(t)] \tag{2.24}$$

where $\boldsymbol{M}_0$ is the initial configuration of the skyrmion when located in the axis origin, and $\boldsymbol{R}(t)$ is the position of the skyrmion's centre of mass at time t [27]. Thiele recognized that, in this case, the time derivative of the magnetization can be written as

$$\frac{d\boldsymbol{M}}{dt} = \frac{\partial \boldsymbol{R}}{\partial t}\frac{\partial \boldsymbol{M}}{\partial \boldsymbol{R}} = \frac{\partial \boldsymbol{R}}{\partial t}\left(-\frac{\partial \boldsymbol{M}}{\partial \boldsymbol{r}}\right) = -(\boldsymbol{v}\cdot\nabla)\boldsymbol{M} \tag{2.25}$$

Substituting the time derivatives that appear in the LLG equation with the expression 2.25, and converting the LLG equation into a force density equation, like detailed in [27], the LLG equation can be rewritten into the Thiele equation [19]. Its expression for the CIP configuration is

$$\boldsymbol{G} \times (\boldsymbol{v}_s - \boldsymbol{v}_d) + \mathcal{D}(\beta \boldsymbol{v}_s - \alpha \boldsymbol{v}_d) + \nabla \boldsymbol{V}(\boldsymbol{r}) = 0 \tag{2.26}$$

Here $\boldsymbol{G} = (0,0,G) = (0,0,4\pi Q)$ is the gyromagnetic coupling vector; $\boldsymbol{v}_d$ is the drift velocity of the skyrmion core; $\boldsymbol{v}_s$ is the velocity of the conduction electrons, where $\boldsymbol{v}_s = -\frac{Pa^3}{2eM_S}\boldsymbol{j}$ (P is the spin polarization of the electrical current and a is the lattice constant); $\mathcal{D}$ is the dissipative force tensor, where $\mathcal{D} = \left(\begin{smallmatrix}\mathcal{D}_{xx} & 0\\ 0 & \mathcal{D}_{yy}\end{smallmatrix}\right)$ and $\mathcal{D}_{xx} = \mathcal{D}_{yy} = \int_{unit\ cell}(\partial_i \boldsymbol{m}\cdot\partial_j \boldsymbol{m})dxdy$ for both skyrmions and antiskyrmions; β is the non-adiabaticity factor; α is the Gilbert damping; $\nabla \boldsymbol{V}(\boldsymbol{r})$ represents the repulsion forces due to process impurities, the nanotrack edges or due to skyrmion-skyrmion repulsion.

Considering a skyrmion that moves far away from the edges along the x-axis (so,

$\boldsymbol{v}_{s,x} \neq 0$, $\boldsymbol{v}_{s,y} = 0$ and $V = 0$), it can be found that [41]

$$\begin{cases} v_{d,x} = \left(\frac{\beta}{\alpha} + \frac{G^2}{\alpha}\frac{\alpha-\beta}{G^2+(\alpha\mathcal{D})^2}\right) v_{s,x} \\ v_{d,y} = \left(\mathcal{D}G\frac{\alpha-\beta}{G^2+(\alpha\mathcal{D})^2}\right) v_{s,x} \end{cases} \tag{2.27}$$

The Thiele equation for the CPP configuration is

$$\boldsymbol{G} \times \boldsymbol{v}_d - \alpha\mathcal{D}\cdot\boldsymbol{v}_d + 4\pi\mathcal{B}\cdot\boldsymbol{J}_{HM} + \nabla\boldsymbol{V}(\boldsymbol{r}) = 0 \tag{2.28}$$

Here $\mathcal{B} = \left(\begin{smallmatrix}\mathcal{B}_{xx} & 0\\ 0 & \mathcal{B}_{yy}\end{smallmatrix}\right)$, where $\mathcal{B}_{xx} = \mathcal{B}_{yy}$ for skyrmions and $\mathcal{B}_{xx} = -\mathcal{B}_{yy}$ for antiskyrmions, is the tensor linked to the STT effect quantifying the efficiency of the spin Hall-spin torque over the spin texture of the skyrmion [18], and it can be determined starting from the spin configuration; $\boldsymbol{J}_{HM} = \frac{\boldsymbol{J}_s}{\theta_{sh}}$ is the electrical current density flowing in the HM, where $\boldsymbol{J}_s$ is the spin current density and θ_{sh} is the spin Hall angle of the HM [20, 18].

From the Thiele equation of the CPP configuration it can be proven that the velocity components of both skyrmion and antiskyrmion are [15, 17]

$$\begin{cases} v_{d,x} = \frac{-j\alpha\mathcal{D}\mathcal{B}_{xx}}{(\alpha\mathcal{D})^2+Q^2} \\ v_{d,y} = \frac{jQ\mathcal{B}_{xx}}{(\alpha\mathcal{D})^2+Q^2} \end{cases} \tag{2.29}$$

It has been demonstrated in [20] that the driving efficiency of the CPP configuration is much higher with respect to the efficiency of the CIP configuration: applying the same current density, the skyrmion velocity obtained with the CPP geometry is higher than the velocity obtained with the CIP geometry.

The drift velocity $\boldsymbol{v}_d$ for both geometries includes not only a component $v_{d,x}$ parallel to the direction of the driving current, but also a transverse component $v_{d,y}$ perpendicular to it that drives the skyrmion towards the track edges. The term inside the Thiele equation that gives rise to this component is the Magnus force $\boldsymbol{G} \times \boldsymbol{v}_d$. Since $\boldsymbol{G} = (0{,}0{,}4\pi Q)$, the reason why the skyrmion is subjected to this force is that it carries a topological charge different from 0.

With both configurations, increasing the current density also the skyrmion velocity will increase. However, there is a limit current density above which the repulsive forces from the edges of the nanotrack, due to the tilting of the magnetization induced by the DMI [15, 16] and taken into account with the term $\nabla \boldsymbol{V}(\boldsymbol{r})$, are not strong enough to balance the Magnus force, so that the skyrmion collides with the edges and gets annihilated due to the breaking of the topological protection.

The Magnus force behaves like the Lorentz force for electrical charges and gives rise to a phenomenon very similar to the traditional Hall effect, even if here the skyrmion does not carry any electrical charge but only a topological charge. This is why this effect is called skyrmion Hall effect.

Like discussed in [18] and in [15], reversing the sign of the magnetization and thus the sign of the topological charge, turning it from $Q = +1$ to $Q = -1$, the topological Magnus force $\boldsymbol{G} \times \boldsymbol{v}_d$ is reversed, since $\boldsymbol{G} = (0{,}0{,}4\pi Q)$ is strictly related to the skyrmion number. As a consequence, reversing the sign of the topological charge, the skyrmions become antiskyrmions and will be accumulated at the opposite edge of the sample, in strict analogy to what happened with the Hall effect for electrical charges. Of course, when the Magnus force deviates the trajectory of an antiskyrmion, the resulting effect is called antiskyrmion Hall effect.

This can be proven also looking at the expression of $v_{d,y}$ for both geometries:

$$v_{d,y} = \begin{cases} \left(\mathcal{D}G\frac{\alpha-\beta}{G^2+(\alpha\mathcal{D})^2}\right) v_{s,x} & \text{for CIP} \\ \frac{jQ\mathcal{B}_{xx}}{(\alpha\mathcal{D})^2+Q^2} & \text{for CPP} \end{cases} \tag{2.30}$$

The $v_{d,y}$ component for the CIP case is directly proportional to $G = 4\pi Q$: reversing the sign of the topological charge also the velocity component will be reversed. The component for the CPP case is proportional directly to Q and thus behaves in the same way. So, applying the same current density, both skyrmions and antiskyrmions propagate in the x-direction with the same speed, while they exhibit equal and opposite transverse velocities.

The (anti)skyrmion Hall angle is defined as the angle between the direction of the applied current and the direction of the resulting motion of the texture, and its expression is

$$\Phi_{sk} = arctan\left(\frac{v_y}{v_x}\right) \tag{2.31}$$

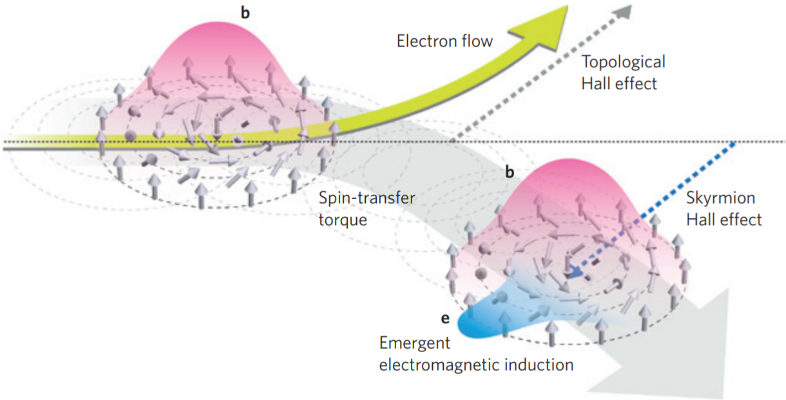


Figure 2.14. Schematic representation of the topological Hall effect and of the skyrmion Hall effect. Electrons are deflected by the Lorentz force due to the emergent magnetic field of the skyrmion, and this results into the topological Hall effect. The velocity of the skyrmion has a transverse component due to the Magnus force in the Thiele equation, and this is the skyrmion Hall effect. Due to the time variation of the emergent magnetic field carried by the skyrmion, is present also an emergent electric field, that is, emergent electromagnetic induction. Figure extracted from [31].

Focusing on the CPP geometry, both the skyrmion and the antiskyrmion Hall angle [15, 18] are equal to

$$\Phi_{sk} = arctan\left(-\frac{Q}{\alpha\mathcal{D}}\right) \tag{2.32}$$

This equality can be easily found by substituting equation 2.29 inside the definition of the (anti)skyrmion Hall angle.

Due to some differences inside the symmetry of the spin texture constituting the antiskyrmions, the antiskyrmion Hall angle, differently from the skyrmion Hall angle, is dependent on the angle θ that the applied current density has with the x-direction, so that its complete expression actually is

$$\Phi_{ask} = arctan\left(-\frac{Q}{\alpha\mathcal{D}}\right) - 2\theta \tag{2.33}$$

It has been proven in [15] that if the current is injected along the direction $\theta = \frac{1}{2}arctan(-\frac{Q}{\alpha\mathcal{D}})$, so that $\Phi_{ask} = 0$, the antiskyrmion Hall effect is cancelled and the texture moves exactly along the current direction, without any transverse motion. Since the maximum speed at which both skyrmions and antiskyrmions can move inside a nanotrack without being annihilated is limited by the competition between the (anti)skyrmion Hall effect and the edge repulsion, enabling a motion with zero antiskyrmion Hall angle can largely increase the maximum velocity of antiskyrmions, allowing a higher throughput for the devices potentially based on them.

2.1.3.4. Mitigation of the skyrmion Hall effect

Since the skyrmion Hall effect may lead to the annihilation of the particle and so of the information it carries, it is a well known issue in the design of skyrmionic devices. Looking at the expression of $v_{d,y}$ in 2.27, it can be noticed that if $\alpha = \beta$ the transverse velocity component is cancelled and the skyrmion Hall effect disappears. However, these two parameters depend on material properties, and it's clear that is impossible to rely on this equality from a design point of view; of course, thanks to the repulsion from the edges of the nanotrack, the skyrmion is able to travel along the nanotrack even if β is not too different from α, but in this case the current density must remain below a certain threshold so that the Magnus force doesn't overcome the repulsive forces from the boundaries, and this of course limits the maximum throughput of the device. So, it's clear why some other solutions to the skyrmion Hall effect must be found, and these solutions must be able to work both with the CIP and with the CPP geometry (even if the CPP configuration has a higher driving efficiency with respect to CIP).

In [10] two methods have been proposed for engineering a potential well, needed for confining the skyrmion in the centre of the nanotrack and preventing its annihilation. The first method proposed tunes the magnetic anisotropy along the width of the nanotrack, reducing the value of the PMA in the centre with respect to the edges. This will form a path of lower resistance all along the nanotrack, since there the magnetization will be allowed to flip more easily due to a lower value of the effective field. Of course, however, there is still a certain value of velocity above which the repulsion from the edges won't be enough and the skyrmion can be destroyed. The patterning of the PMA can be performed by ion irradiation, combined together with high-resolution litography. Of course this must be done during the fabrication step and is a static control, without any possibility of change during the lifetime of the device.

The second method proposed is to add more ferromagnetic material at the edges of the nanotrack. Doing so the demagnetization field is increased at the inner edges of the modified nanotrack and decreased at its centre: so there is again a magnetic potential well which forces the skyrmion to move at the centre of the track. The threshold velocity above which annihilation happens in this case is even higher that the one that can be obtained through PMA patterning.

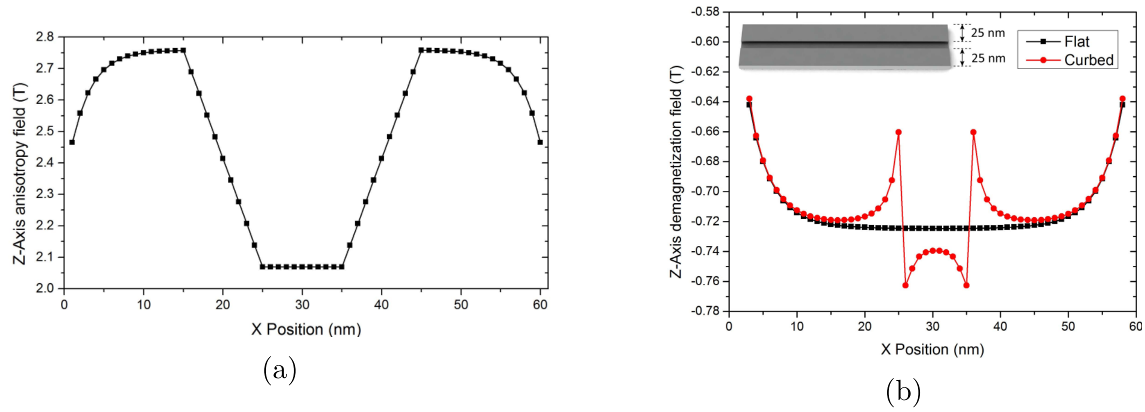


Figure 2.15. (a) z-axis anisotropy field along the width of the nanotrack after patterning of the PMA constant: K_u has a lower value at the centre of the nanotrack. (b) z-axis demagnetization field for the traditional nanotrack (black curve) and for the modified nanotrack shown in the inset (red curve). Figures extracted from [10].

A completely different possibility is presented in [41]. The structure of the device described in this article is reported in figure 2.16: a FM layer with positive background magnetization is separated from a bottom FM layer with opposite background magnetization by an insulating spacer. Below the bottom FM layer, a HM substrate allows the flow of a current in the x-direction and is able to generate a spin-polarized current vertically directed with spin direction along $+y$. The peculiarity of this structure is to have an antiferromagnetic (AFM) exchange coupling between the top and the bottom FM layers. The Hamiltonian for this kind of interaction is

$$H_{inter} = -A_{inter} \sum_i \boldsymbol{m}_i^T \cdot \boldsymbol{m}_i^B \tag{2.34}$$

where T stands for top, B stands for bottom and A_{inter}, the interlayer AFM exchange stiffness, is negative. As a result, if the magnetic moments of the top layer point in one direction, the moments of the bottom layer will point exactly in the opposite direction, in order to minimize the total energy of the system, equal to $H_{total} =$

$H_T + H_B + H_{inter}$.

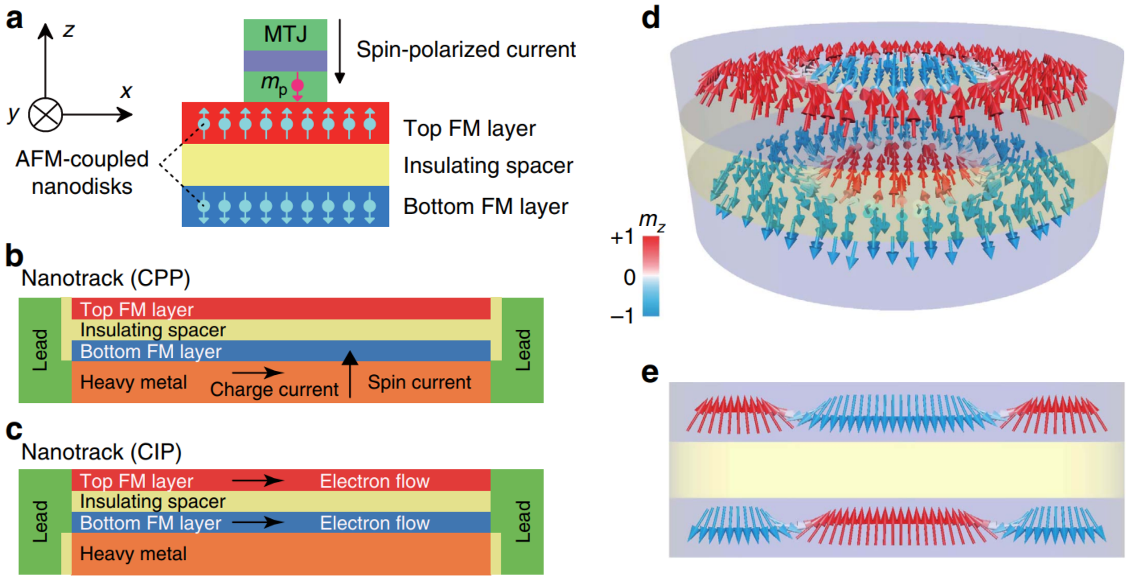


Figure 2.16. Schematic of the AFM exchange coupled bilayer system. (a) MTJ write-head needed for the nucleation of a single skyrmion in the top FM layer. (b-c) Bilayer nanotrack where the CPP geometry (b) or the CIP geometry (c) is exploited for the skyrmion motion. (d) Illustration of the bilayer-skyrmion. (e) Side view of the bilayer skyrmion along the diameter section. Figure extracted from [41].

In this structure is possible to nucleate a skyrmion by injecting a current inside the MTJ shown in figure 2.16 (the nucleation of a skyrmion by allowing a current flow through a MTJ is explained in section 2.1.5); the resulting spin-polarized current is not able to reach the bottom FM layer, so if the AFM exchange coupling is not strong enough only a single skyrmion will be nucleated in the top layer. If instead the coupling between the two layers is strong, the nucleation of a skyrmion in the top FM layer will induce the nucleation of a skyrmion with opposite topological charge (since the spin directions are opposite, as detailed by equation 2.7) also in the bottom FM layer. These two skyrmions (magnetic bilayer skyrmion) are bounded and move together along the track. If the current is injected according to the CIP geometry, the texture of both skyrmions will be subjected to the torque from the conduction electrons. If instead is adopted the CPP geometry, only the bottom skyrmion will be subjected to the torque of the spin-polarized current coming from the SHE: the top skyrmion will move only due to the AFM exchange coupling.

The key point is that the two textures have opposite topological charge: for this reason, the $\boldsymbol{G} \times \boldsymbol{v}^{(d)}$ term (Magnus force) that appears in the Thiele equation of both CIP and CPP geometry is cancelled. Like mentioned in section 2.1.3.3, in fact, $\boldsymbol{G} = (0,0,4\pi Q)$ depends on the topological charge, and if the sign of the topological charge is reversed also the Magnus force will change its sign. Thanks to the bound connecting the skyrmions inside the two layers, the total Magnus force on the system of the bilayer skyrmion is exactly zero. In this way is possible to obtain the movement along a straight line without any skyrmion Hall effect: this allows to obtain a system in which the velocity of the information carriers can reach even $1000\,\mathrm{ms}^{-1}$. Moreover, in the article is demonstrated that the bilayer skyrmion maintains the same (low) depinning current density of a single skyrmion, and that the CPP geometry is again the most efficient configuration, like it happened in the case of single skyrmions.

In the same article is proposed also a second method to nucleate a bilayer skyrmion, exploiting the result presented in [43]: it is enough in fact first to nucleate an AFM-coupled DW pair, to move it along a narrow track an then, through a narrow-wide junction geometry, to convert it into a bilayer skyrmion, similarly to what has been discussed in section 2.1.1.7.

2.1.4. Nucleation

The nucleation of skyrmions can be obtained in many different ways: by means of an electrical current, of magnetic fields, or even with local heating using laser irradiation; again, from an application point of view the nucleation through the injection of an electrical current is the most promising mechanism.

2.1.4.1. STT

In [33] has been studied the nucleation of a single skyrmion in a thin magnetic film by injecting a out-of-plane spin-polarized current perpendicularly to the film plane. By changing the simulation parameters it has been studied also the dependence of the threshold current density on the Gilbert damping, on the magnitude of the DMI and on the PMA coefficient.

2.1.4.2. Notch

Another mechanism for the nucleation of skyrmions has been proposed in [16]. Here a notch inside the ferromagnetic material is exposed to a magnetic field and to a flowing current. Even if the magnetic field considered in the article is directed along $+z$, the spin direction along the edges of the sample is in-plane due to the DM interaction. When injecting an electric current, the STT and the DMI together make the spins first swell out around the corner, then twist and point down at the core of the new skyrmion, which after some fluctuations in the radius size will detach from the corner and move in the sample due to the STT from the current. The authors have studied the dependence on the sign of the magnetic field, on the sign of the current density and on the shape and dimension of the notch. It has been proved that the essential feature is the spin pattern along the edge of the sample, together with the direction of the injected current: a current with opposite sign is not able to generate a skyrmion, due to the unique direction that the spin movement has in the Larmor precession. If the in-plane component of the spin is guaranteed and if the current has the correct direction (given the sign of the applied magnetic field), then even a notch with round shape would allow the nucleation.

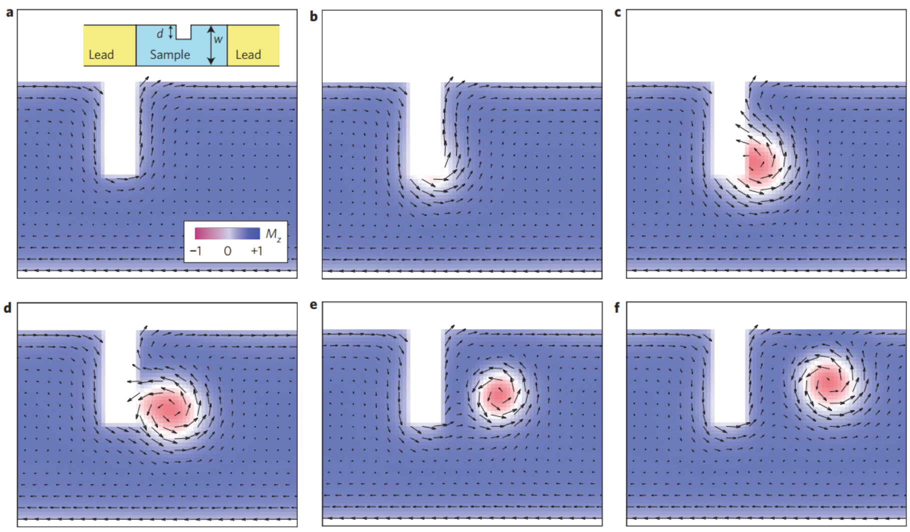


Figure 2.17. Simulation snapshots showing the nucleation of a skyrmion around a rectangular notch. Blue(red) represents the $+z(-z)$ component of the magnetization. Figure extracted from [16].

2.1.5. Detection

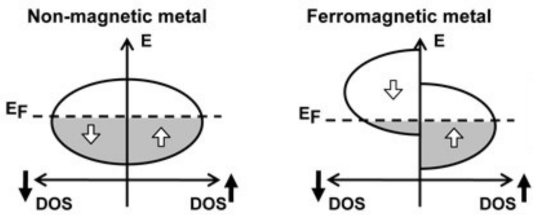
The detection of skyrmions can be accomplished mainly through either the topological Hall effect or the magnetoresistance effect; only the latter method, however, can be easily performed in a fully electrical way, and so is the easiest to be implemented in an electronic device [20].

In [36] the electrical detection of a single skyrmion based on the tunnel magnetoresistance (TMR) effect at room temperature has been detailed. The structure of the proposed read-head is made by a HM layer, above which is deposited an ultra-thin ferromagnetic layer, in order to achieve a strong i-DMI (interfacial DMI). This ferromagnetic layer will host the skyrmion and is at the same time the free layer of a MTJ; it has an elliptical cross section with the major axis oriented along the y-direction. The pinned layer of the MTJ has a fixed out-of-plane easy axis of the magnetization and is a nano-contact with 50 nm of diameter, since the average value of the skyrmion diameter, with the parameters chosen for the simulation, is around 40 nm. First of all, the skyrmion nucleation is achieved, like proposed in [33], by injecting a current pulse through the MTJ.

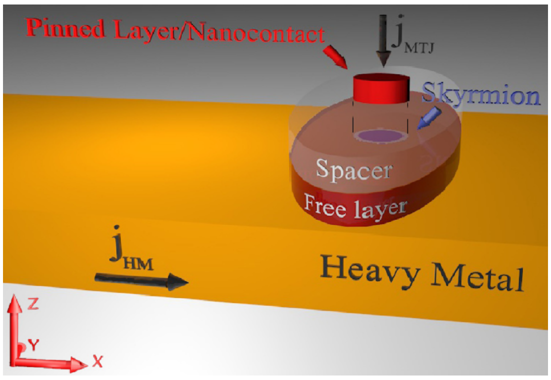
Spin-polarized current

The current passing through the ferromagnetic layer with fixed magnetization becomes spin-polarized. Like explained by [29], in fact, the density of states (DOS) of a ferromagnetic metal is different from the DOS of a normal metal: like shown in figure 2.18a, in this case the DOS is different for the two spin states, so that the spin-up band at the Fermi level is mostly filled, while there are many empty states available in the spin-down band. So, the conduction electrons injected in the ferromagnet will encounter a different resistivity, according to their spin: the spin-down electrons have more states to scatter into, and as a result they see a higher resistivity ($\rho_\downarrow$) compared to the one ($\rho_\uparrow$) seen by the spin up electrons. So, overall, this leads to the spin-polarization of the current injected, where the degree of spin-polarization is given by

$$P = \frac{\rho_\downarrow - \rho_\uparrow}{\rho_\downarrow + \rho_\uparrow} \tag{2.35}$$



(a)



(b)

Figure 2.18. (a) DOS of a metal and of a ferromagnetic material. The misalignment of the spin-up and of the spin-down band in the energy diagram determines a higher number of states available in the spin-down band. Adapted from [2]. (b) Structure of the detection-head proposed in [36]. The MTJ structure is exploited both for the detection and for the nucleation, while a current flowing in the HM layer controls the motion of the skyrmion in the free layer via SHE. Figure extracted from [36].

Once the skyrmion is nucleated, the main problem to solve is its thermal drift below the detection head. This thermal drift induces shape deformations, apart from the so called breathing (expansions and compressions of the spin texture), and most of all makes the skyrmion follow a Brownian motion, which forces it to stay most of the time away from the detection head: that's why it cannot be detected by means of just a current flowing through the read-head. The motion of the skyrmion must instead become controlled, and must force it to pass periodically below the detection area. To do so, an electrical microwave current is made flow in the HM layer in the x-direction: in this way, thanks to the SHE, the skyrmion is forced to move along the y-axis of the free layer. The consequence of this periodic passage is a periodic variation of the out-of-plane component of the magnetization: then, due to the tunnel magnetoresistance effect, the resistance that a current density J_{MTJ} across the MTJ encounters will change periodically, and this allows the detection of the skyrmion.

2.1.6. Skyrmion size

Like observed in [38], a bit of confusion can be found in literature about the topic of the skyrmion size, and many different and non-equivalent expressions have been used up to now. Skyrmions are made of an inner core, an outer domain and a wall separating them; so, when dealing with the skyrmion size, first of all is necessary to distinguish between the contour of the region where $m_z = 0$, whose radius is called skyrmion size R, and the wall width w surrounding the core. These two quantities are visually defined in figure 2.19.

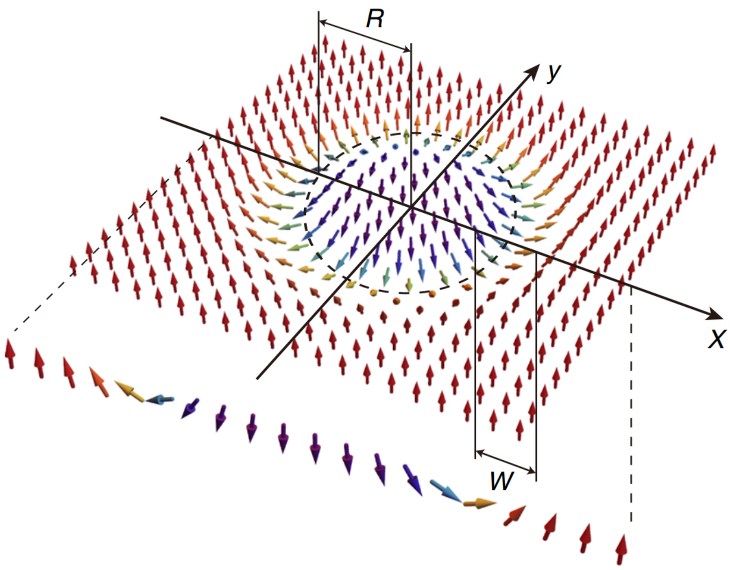


Figure 2.19. Schematic of a Néel skyrmion showing the visual definition of the skyrmion size R and of the wall width w. Figure extracted from [38].

The authors of this article focus on Néel skyrmions stabilized by i-DMI in a ultra-thin ferromagnetic film; they confirm that the same results apply also to Bloch skyrmions stabilized by bulk DMI. However, the basic assumption is that the thickness of the film is much smaller than both R and w, so that the demagnetizing field can be neglected. If the thickness of the film increases, the model they present and all the 2D theories behind it are no longer applicable.

The energy terms taken into account are the exchange energy, the DMI energy, the anisotropy energy and the Zeeman energy, where A is the exchange constant, D is the DMI coefficient, K is the perpendicular easy-axis anisotropy and B the perpendicular magnetic field. In the article it is first found the general and exact expression of the total energy E including these terms. The skyrmion radius and the wall width are those values that minimize the energy of the system: so, their dependence on A, D, K and B is found by minimizing the total energy with respect to R and w separately. This is the main innovation that this article brings on the topic: usually the domain wall width was either considered constant or dependent on the value of R. Doing so, instead, the simulation points are quite perfectly interpolated by the curve obtained by minimizing the exact expression of E with respect to R and w and by varying the values of A, K, D and B that appear in it. In figure 2.20 are shown the four plots, where the symbols represent the simulation data and the solid lines are the results obtained theoretically from the exact expression of the energy.

However, doing so is not possible to get the complete expression of R and w, expression which would be useful to have a clear idea about their dependence on the four parameters. For this reason, in the same article the expression of the energy was approximated under the assumption $R >> w$, and minimizing it again with respect to R and w the following expressions can be found for $B = 0$:

$$R = \pi D \sqrt{\frac{A}{16AK^2 - \pi^2 D^2 K}} \tag{2.36}$$

$$w = \frac{\pi D}{4K} \tag{2.37}$$

If $B \neq 0$ there is no closed-form solution; however, the approximated dependency of both R and w on B can still be simulated by varying its value in the two minimized expressions. In this way the dashed curves shown in figure 2.20 are obtained.

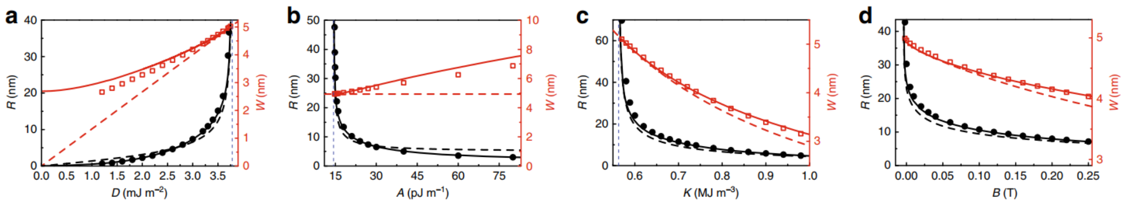


Figure 2.20. Comparison between simulation data (symbols), exact theoretical result (solid curve) and approximated theoretical result (dashed curve) for both the skyrmion size R and the wall width w. The limit of the region where the skyrmion is the configuration minimizing the energy of the system is shown by the vertical dashed line. Figure extracted from [38].

From equation 2.36, imposing that R is a real and finite number, it can be derived that

$$16AK > \pi^2 D^2 \tag{2.38}$$

From this expression is possible to determine the limit value for A, D and K: these values are reported in the plots like a vertical dashed line, which agrees well with the simulations even if it has been derived from an approximate expression. If these three parameters do not respect the range obtainable from equation 2.38, the stable state is not a skyrmion but stripe domains.

In figure 2.21 is shown the comparison with the results obtained from two different articles. In particular, the one indicated as *Ref. [26]* is [42], whose results are commented in section 2.2.3. From this comparison is possible to realize the validity of the model proposed, and how it should be trusted more than different models proposed in other articles.

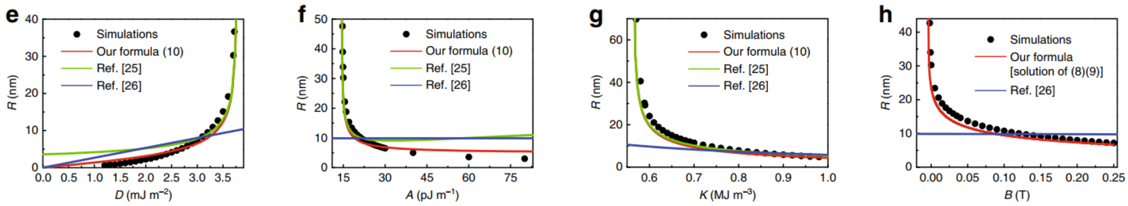


Figure 2.21. Skyrmion size: comparison between simulation data, the results provided by the approximated expression of the energy and the results obtained in two different articles. Figure extracted from [38].

In article [38] the skyrmions are assumed to be isolated inside an infinite medium, so the edge effects are not taken into account. In [6] instead are studied the effects that the repulsive forces from the boundaries of a tapered nanotrack have on the skyrmion size. The nanotrack used in the simulations, shown in figure 2.22, exploits the results of [41]: it is in fact made by two FM layers separated by a spacer, so that the skyrmion Hall effect is cancelled thanks to the AFM exchange coupling between the two layers, like already discussed in section 2.1.3.4. The skyrmion movement is achieved through the SHE.

The results shown in the article are reported in figure 2.23. The variation of the skyrmion size according to the track width is shown in figure 2.23a: while the skyrmion moves along the track, the edges exert a repulsive force on its texture, shrinking it. In particular, at the end of the nanotrack, where the width is of 30 nm, the skyrmion diameter is 10 nm. It's difficult to obtain such a small size directly with the nucleation of a skyrmion via STT: that's why a tapered nanotrack could become useful in increasing the packaging density of information in a skyrmionic

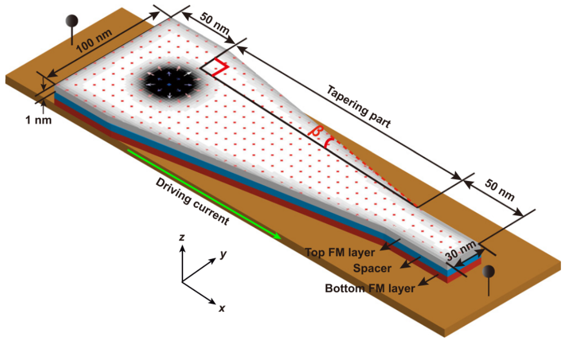
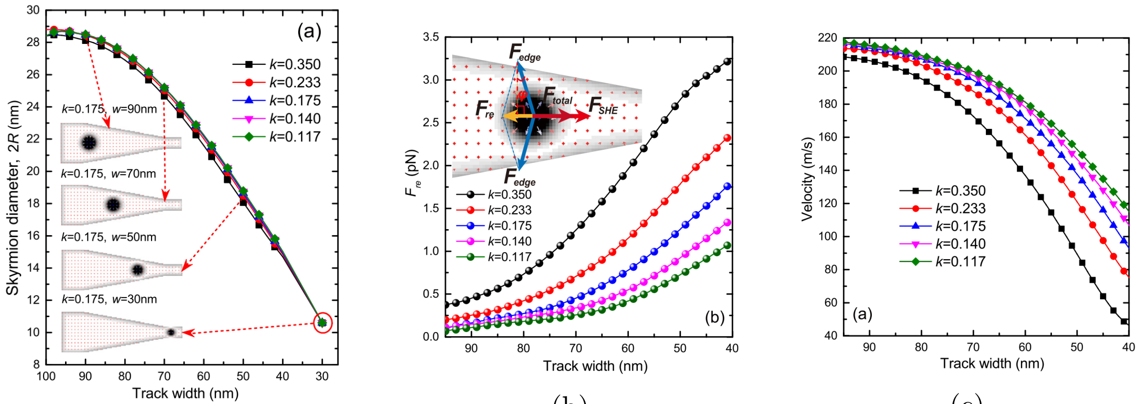


Figure 2.22. Tapered nanotrack used in [6]: two FM layers are AFM exchange-coupled, so that the skyrmion Hall effect is suppressed. The skyrmion is moved by a current flowing in the HM layer via SHE. The varying width of the nanotrack exerts a compression on the skyrmion texture. The slope is defined by $k = tan(\beta)$. Figure extracted from [6].



(a) (b) (c)

Figure 2.23. (a) Dependence of the diameter of the skyrmion on the width of the nanotrack. The dependence on the slope k of the nanotrack is very weak. (b) The resultant of the repulsive forces from the edges increases when the skyrmion radius decreases; increasing the slope of the nanotrack the force component that opposes to the skyrmion's motion increases. The inset shows the force components competing in the motion. (c) The velocity of the skyrmion decreases when the track width decreases; increasing the slope of the nanotrack the velocity decreases as well. Figures extracted from [6].

memory.

The authors observe also that the repulsive force exerted by the non-parallel edges makes the velocity decrease with the track width (figure 2.23c). The summary of the forces competing in the system is shown in the inset of figure 2.23b: each edge exerts a force $\boldsymbol{F}_{edge}$ on the texture, and the sum of these two forces is $\boldsymbol{F}_{re}$, opposite in sign with respect to the force $\boldsymbol{F}_{SHE}$ responsible of the motion of the skyrmion. Like shown in figure 2.23b, $\boldsymbol{F}_{re}$ increases while the width of the track decreases, and opposing to $\boldsymbol{F}_{SHE}$ it makes the resulting skyrmion velocity decrease.

The result concerning the skyrmion size shown in figure 2.23a is confirmed also by the simulation results presented in [10], where the width of the considered nanotrack spans between 28 nm and 10 nm.

2.2. Applications

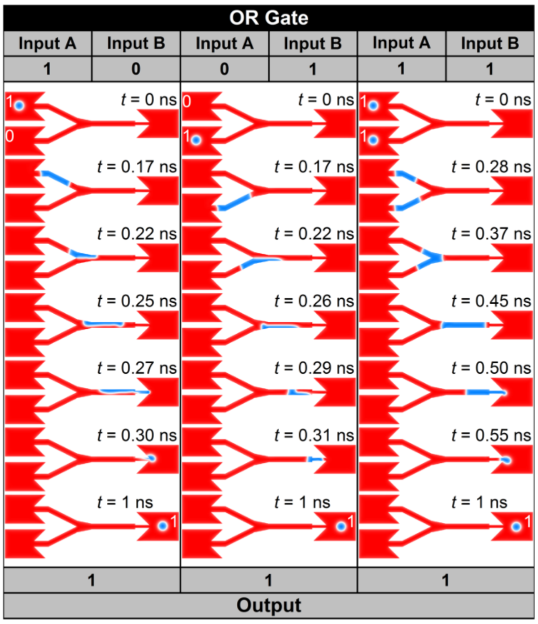
Like already mentioned in the introduction to this thesis, skyrmions present many advantages with respect to domain walls: they need a lower depinning current density and they are smaller, so overall they allow either to consume less power while maintaining the same throughput, or to increase the throughput while maintaining the same power consumption.

It is a fact that a lot of research has already been done on DW racetrack memories: however, thanks to the result shown in [43], is now possible to reversibly convert DW pairs into skyrmions and vice versa. This allows to exploit all the work already done on the optimization of racetrack memories, together with all the advantages presented by skyrmions. Not only: as it will be shown in this section, different designs for skyrmionic logic gates have been proposed. This allows even to realize some logic in memory: the information could be stored in the form of DW pairs, which can then be elaborated by skyrmionic gates after a conversion, and the result of the elaboration can be stored again like DW pair somewhere else.

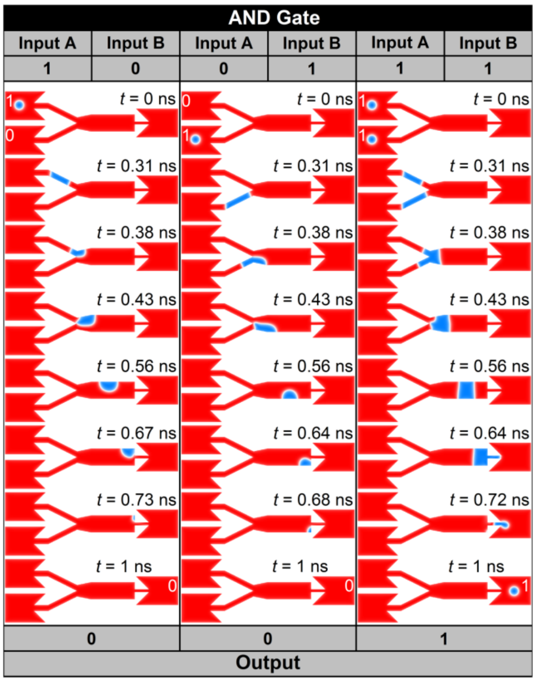
Of course, as soon as the basic logic gates are available, it is then possible to build more complex computing architectures: this topic will be discussed in chapter 4.

2.2.1. Logic gates using the DW pair-skyrmion reversible conversion

In the first part of [40] is studied the variation of the topological numbers of skyrmionic structures according to the physical parameters of the material used for the wide part of a nanojunction, exploiting the reversible conversion from DW pair to skyrmion proposed in [43]. This topic has been detailed in section 2.1.1.7. In the second part of [40] the DW pair-skyrmion conversion is exploited for designing the basic logic gates. Here a skyrmion signifies a logic 1, while its absence represents a 0.



(a)



(b)

Figure 2.24. (a) *OR* gate: snapshots of micromagnetic simulation at different instants showing the cases $1+0$, $0+1$ and $1+1$. The $0+0$ case is trivial. (b) *AND* gate: snapshots of micromagnetic simulations for the cases $1 \cdot 0$, $0 \cdot 1$ and $1 \cdot 1$. The $0 \cdot 0$ case is trivial. Figures extracted from [40].

Figure 2.24a shows the structure of the *OR* gate in different simulation points and according to different input combinations. In the $1+0$ and the $0+1$ case, the only skyrmion on the input is converted into a DW pair thanks to the presence of the narrow junction, is propagated along the structure and is converted back into a skyrmion in the wide output region. When two skyrmions are present they are both converted into a DW pair, and when these two structures meet at the central junction of the Y structure they merge into a single DW pair, which is then propagated towards the output and converted back into a single skyrmion like before.

In figure 2.24b is represented the structure of the *AND* gate. It is very similar to the *OR* gate, with the only difference of a wider bottom half in the Y central junction. In this way, considering the $1 \cdot 0$ and the $0 \cdot 1$ cases, when injecting the same current density as before, the current density in the bottom half of the junction will be lower with respect to the case of the *OR* gate, so the DW pair, since it is pushed towards a wider region, is converted not into a skyrmion but into a meron. Since the meron has the peculiarity of remaining attached to the sample edge, when it reaches the output junction it is driven away from the nanotrack: so in both cases the output of the function is 0, as expected. In the $1 \cdot 1$ case, instead, the two DW pairs are still able to merge into a single DW pair, which is propagated towards the output and converted back into a single skyrmion, correctly providing a 1 on the output.

The cases $0+0$ and $0 \cdot 0$ are not shown since they are trivial: if we don't provide any skyrmion on the inputs of the gates, the output will be for sure 0.

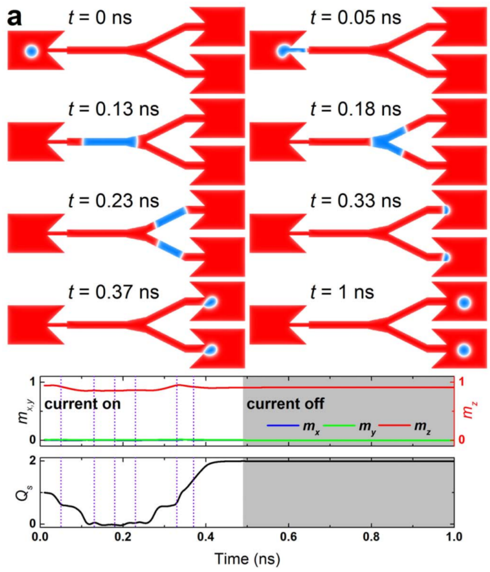


Figure 2.25. Duplication of a skyrmion. The middle panel shows the time evolution of the magnetization components m_x, m_y and m_z. The bottom panel shows the time evolution of the skyrmion number: from $Q = 1$ (one skyrmion) it becomes $Q = 0$ (DW pair) and then $Q = 2$ (two skyrmions). Figure extracted from [40].

It is worth noticing that, using the same structure of the *OR* gate but exchanging output and inputs, the gate obtained is able to perform the duplication of a single skyrmion, as shown in figure 2.25. Like observed by the authors, the capability of duplicating the information carried by a single skyrmion is very important for any skyrmionic device.

Even if not specified by the authors of the article, is worth noticing that these gates could in principle be used inside a conservative skyrmionic logic system: the skyrmions on the output of each of these gates could in fact be used in the remaining part of a larger circuit to trigger the computation of other logic functions, like proposed in [5] for a different type of logic gates (see 2.2.2).

2.2.2. Logic gates for conservative logic systems

The application proposed in [5] is a different method to realize the basic logic gates. It exploits the results shown in [10], adopting an additional layer of FM material at the edges of the nanotrack to achieve the confinement of the skyrmion, like shown in figure 2.26. Moreover, the system that can be obtained from these gates is defined ”conservative”, because once the skyrmions have crossed the whole gate, allowing the computation of the logic function, they can be collected at the other end and used to trigger the computation of new functions, without the need of nucleating new skyrmions, which is an energetically expensive operation. Finally, the way in which is taken advantage of the complex physics of skyrmions in realizing the computations deserves an additional mention: here the features that in other skyrmionic devices represent a problem, like the edge-skyrmion repulsion, the skyrmion-skyrmion repulsion, and most of all the skyrmion Hall effect, are not only tolerated, but actually exploited for realizing the computation.

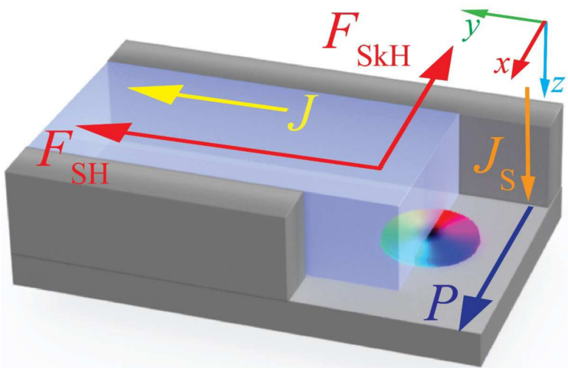
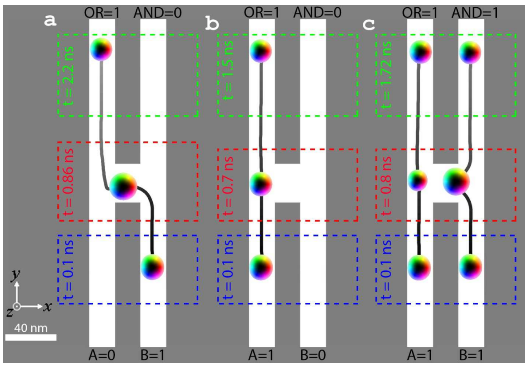
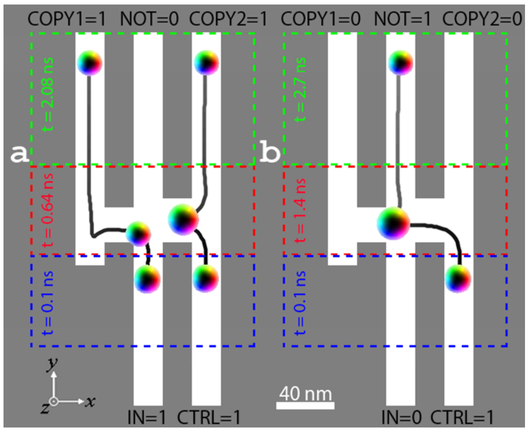


Figure 2.26. Structure of the nanotrack used in [5], made of *Pt* HM (blue) and *Co* FM (gray, with polarization *P*). The current *J* injected in the HM is converted via SHE into the spin-polarized current J_S, producing on the skyrmion (multicolor circle) a force F_{SH} in the direction of the electrical current $(+y)$ and a force F_{SkH}, due to the skyrmion Hall effect, directed along $-x$. Figure extracted from [5].

The structure of the gates proposed is shown in figure 2.27. Also in this case a logic 1 is represented by a skyrmion and a 0 by its absence. The device shown in 2.27a implements the *AND* and the *OR* function at once: remembering that each skyrmion is subjected at the same time to a force component along $+y$ and to a second force component directed along $-x$, is easy to understand the working mechanism behind the gate. When a single skyrmion is present (case $A = 0$, $B = 1$ or $A = 1$, $B = 0$), it moves along its track as long as there is no way to move towards the $-x$-direction. If the skyrmion is in the rightmost track and it reaches the central junction, the skyrmion Hall effect will make it move towards left and change its track. This won't happen only if there is already a skyrmion occupying the left track ($A = 1$, $B = 1$): in this case the skyrmion-skyrmion repulsion prevails and both skyrmions continue on their respective track, correctly providing a logic 1 on both outputs.



(a)



(b)

Figure 2.27. (a) *AND/OR* gate: schematic of the behaviour of the gate for the cases ($A = 0$, $B = 1$), ($A = 1$, $B = 0$), ($A = 1$, $B = 1$). The ($A = 0$, $B = 0$) case is trivial. (b) *COPY/NOT* gate: schematic of the behaviour of the gate for the cases $IN = 1$ and $IN = 0$. Figures extracted from [5].

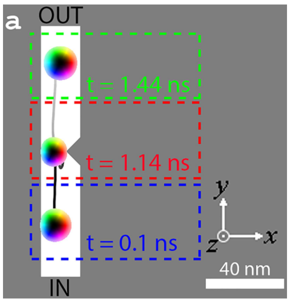
In 2.27b is shown the structure of the *INV/COPY* gate. This time, to allow the correct functioning, the *CTRL* input must always be set to 1. Then, if both skyrmions are present, the skyrmion-skyrmion repulsion in the central junction will make the rightmost go towards the *COPY2* output, and together with the skyrmion Hall effect it will make the leftmost reach the *COPY1* output. In this way the value of the input ($IN = 1$) is copied on the two *COPY* outputs, while it is inverted on the *NOT* output. If a single skyrmion is present, the skyrmion Hall effect will make it go towards the $-x$ direction, but it is not strong enough to make it reach the *COPY1* track: again, the value of the input is copied on the two *COPY* outputs and inverted on the *NOT* output.

Having the implementation of both the *AND*/*OR* function and of the *NOT* function, it is in principle possible to realize any boolean logic function. However, here the correct functioning of the device is based on the forces that arise from the interactions of skyrmions inside the same logic gate. It is then of vital importance to synchronize their movement and to control their timing: differently from the traditional electric circuits, in which (at least in lumped circuits) the speed of the signal propagation can be approximated as infinite, here the speed of skyrmions is limited and the time they take in propagating along each track must be carefully considered. That's why in the same article has been proposed also the structure of a signal synchronizer, shown in figure 2.28a. It is realized with a 7 nm wide notch that blocks the passage of the skyrmion, which is around 20 nm wide.

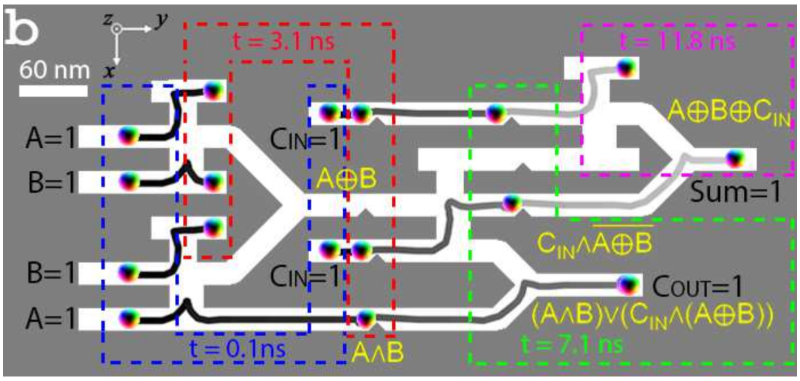
Like discussed in section 2.1.6, a tapered nanotrack has the effect of decreasing the radius of a skyrmion passing through it. However, the repulsion from the edges gives rise to a force component antiparallel with respect to the force $\boldsymbol{F}_{SH}$, which is coming from the SHE and is responsible for the skyrmion motion. The force component $\boldsymbol{F}_{SH}$ is directly proportional to the current density applied ([6]): as a result, if the current density is not enough, the skyrmion propagation will be blocked by the notch. The passage of the skyrmion is allowed only when a higher current density is applied: in this way the force component $\boldsymbol{F}_{SH}$ becomes able to overcome the repulsion force coming from the edges, the skyrmion diameter is reduced due to edge repulsions, and the information can go through the restriction and continue the propagation along the circuit.

In this way the current needed to allow the data propagation becomes at the same time also the clock needed for synchronizing it: the low logic level of the current will correspond to a value higher than the depinning current, but lower than the value needed for allowing the passage of the skyrmions through the notches present in the circuit; this high value will be applied periodically only for a short period of time, so that the resulting waveform of the current will have a very small duty cycle.

The last structure proposed in the paper is the one of a *FULL ADDER*, shown



(a)



(b)

Figure 2.28. (a) Signal synchronizer: the 7 nm wide notch blocks the passage of the skyrmion until a higher current density level is applied. (b) *Full Adder* structure, built using the *INV/COPY* gate, the signal synchronizer and some join tracks. Figures extracted from [5].

in figure 2.28b. For understanding its behaviour is important to notice that the structure of the *INV/COPY* gate is able to perform also different logic computations, if the assumption of having always an input $CTRL = 1$ is relaxed. It's easy to verify that the same structure corresponds to the truth table 2.2. The names of the inputs and of the outputs of the gate have been redefined, like shown in figure 2.29, for avoiding any confusion about the actual logic function implemented.

Table 2.2. Truth table for the *INV/COPY* gate

A	B	OUT2	OUT1	OUT0
0	0	0	0	0
0	1	0	1	0
1	0	1	0	0
1	1	1	0	1

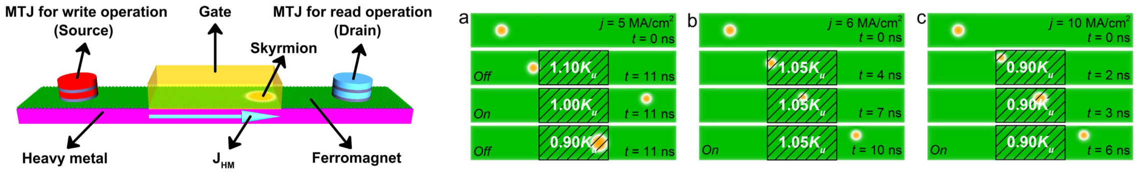
$$\begin{cases} OUT2 = A \\ OUT1 = \overline{A} \cdot B \\ OUT0 = A \cdot B \end{cases} \tag{2.39}$$

Figure 2.29.

The structures proposed in this article will be the starting point first for the micromagnetic analysis (chapter 3), then for the architectural analysis (chapter 4) that are the topic of this thesis. They will be used in chapters 5 and 6 as well.

2.2.3. Skyrmion-based transistors

In [42] it was proposed a skyrmionic device able to behave like a conventional CMOS transistor. Its structure is shown in figure 2.30a.



(a) (b)

Figure 2.30. (a) Structure of the skyrmionic transistor: two MTJ are used respectively for writing and for reading the skyrmion, whose motion is achieved via SHE. The gating effect is obtained applying a voltage on the gate terminal and tuning the magnetic anisotropy of the gate region of the nanotrack. (b) Snapshots of micromagnetic simulation: if no voltage is applied $K_{uv} = K_u$ and the transistor is on. If $K_{uv} > K_u$ the skyrmion encounters a potential barrier and pins at the left of the gate region; if $K_{uv} > K_u$ it finds a potential well and pins at the right side of the gate area. In both cases, increasing the current density is possible to depin the skyrmion and turn the transistor on. Figures extracted from [42].

In this device, a MTJ is used for nucleating the skyrmion and a second MTJ is used for reading it, just like described in section 2.1.5; the skyrmion is moved from source to drain via the SHE by a J_{HM} flowing in the bottom HM layer. Finally, the gating is obtained by applying a voltage on the gate terminal: this will induce a variation of the PMA parameter K_{uv}, according to the relation

$$K_{uv} = K_u + \Delta K_{uv} E \tag{2.40}$$

where E is the applied electric field.

In the *off* state both the current density and the electric field are applied, while in the *on* state only the electric field is turned off, so that $K_{uv} = 1.00K_u$. Like equation 2.40 shows, the electric field increases the value of the PMA; the energy of the skyrmion is

$$E_{sk} = -\frac{D^2\pi^4}{4K\pi + \frac{16}{\pi}B} + 38.7A \tag{2.41}$$

while its radius, according to this article, is

$$R_{sk} = -\frac{D\pi^2}{2K\pi + \frac{8}{\pi}B} \tag{2.42}$$

where K is the PMA constant, D is the DMI magnitude, A is the exchange constant and B is the magnetic field. However, like discussed in section 2.1.6, this equation for the radius of the skyrmion has been proven wrong in [38], according to which the correct (approximated) definition of the radius is

$$R_{sk} = \pi D\sqrt{\frac{A}{16AK^2 - \pi^2 D^2 K}} \tag{2.43}$$

where the label of each parameter has remained the same. In this way it is represented the dependence of the radius not only on D and K (whereas the dependence on B is more complex to be represented), but also on the exchange interaction stiffness A. However, the key point here is that the dependence on the PMA constant has remained similar: if the PMA increases the skyrmion radius decreases, and vice versa, decreasing the PMA the radius increases. According to equation 2.41, increasing the magnetic anisotropy the energy of the skyrmion increases too and vice versa, decreasing the magnetic anisotropy the energy decreases.

As a result, from the snapshot (a) in 2.30b it can be observed that, if the electric field is turned off, the skyrmion is free to go through the voltage-gated region and to reach the drain. If the electric field increases the PMA, it means that the skyrmions finds a potential barrier on the left side of the gated region and stops there. If, in the contrary, the field decreases the PMA, the skyrmion finds at first a potential well and is able to enter the gated region (and its radius there increases, like predicted); then, going out of the potential well it finds a potential barrier, and if the current density is not enough it won't be able to exit the potential well, remaining pinned at the other side of the gated region. However, in both cases there is always a current density threshold above which the skyrmion is able to overcome the barrier and reach the drain. In the article the authors have studied the dependence of this threshold on the PMA value and on the intensity of the DMI. Finally, it has been demonstrated that the same behaviour is verified even if the size of the nanotrack is reduced, and this indicates the good scalability of this skyrmionic transistor, which could be used as a component of hybrid skyrmionic-electronic devices.

3. Micromagnetic analysis

In section 2.2.2 the key principle behind the implementation of some basic skyrmionic logic gates has been described. The results presented in the article, however, were obtained by assuming a uniform distribution of the current density throughout the whole gate: this, of course, is an unrealistic assumption. The aim of this chapter is to verify whether those structures are able to correctly work within a simulation that receives as input a realistic current distribution.

3.1. Methods

The results presented in the article [5] were obtained by using the software *mumax*^{3}, a GPU-accelerated simulation software that, given a structure and given a subdivision of this structure into cells, 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; besides that, to obtain any kind of result is also necessary to add some kind of excitation, either in the form of a magnetic field or of an electric current. It's even possible to provide as input a spin-polarized current, whose direction must be specified by a vector. In this particular case, a spin-polarized electric current directed along $+z$ and with polarization direction along $-y$ must be provided: this current can be set uniformly in the whole structure (and this is what was done in [5]), but is also possible to specify different current density values inside the structure by subdividing it in regions; the version of *mumax*^{3} that has been used in this thesis supports up to 255 different regions, and each of them can in general be specified not only by a different current density, but also by different material parameters. In this case, however, it is assumed that the material parameters are uniform in the whole structure and take the values specified in table 3.1. These values have been adopted from [5] and from [10].

Table 3.1. Material parameters used in *mumax*^{3} simulations

Symbol	**Description**	**Value**
M_{sat}	Saturation magnetization	5.8×10^5 A/m
A_{ex}	Exchange stiffness	1.5×10^{-11} J/m
α	Gilbert damping coefficient	0.1
ξ	STT non-adiabacity	0.35
D_{ind}	DMI constant	3.0×10^{-3} J/m^2
Ku_1	Magneto-crystalline anisotropy constant	6×10^5 J/m^3
Ku_2	Magneto-crystalline anisotropy constant	1.5×10^5 J/m^3
$Temp$	Temperature	0 K
Pol	Spin polarization	1
Λ	Slonczewski parameter	1
ε'	Slonczewski secondary STT term	0
θ_{SH}	Spin Hall angle	1

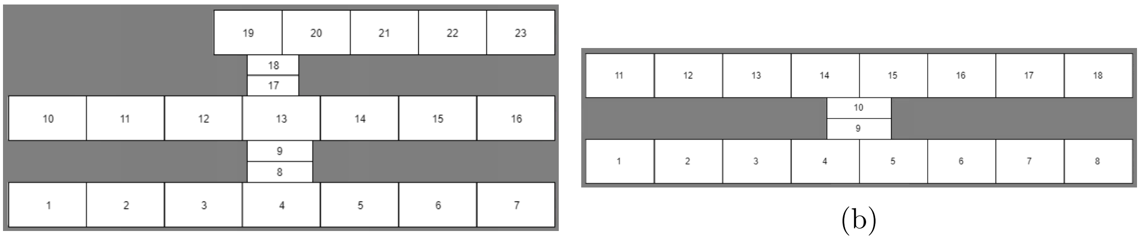
The values of current density needed by *mumax*^{3} to initialize the 255 regions in which the gate is divided are provided *COMSOL Multiphysics*. With this simulation software is possible to describe the full structure of the gate, including the platinum tracks, which are not taken into account by *mumax*^{3}. So, the gate is described both in terms of structure and in terms of materials. Here the track is made of cobalt and the metal traces of platinum: the resistivity of platinum was set to $\rho_{Pt} = 9.8 \times 10^{-8}\,\Omega\,\mathrm{m}$ and the resistivity of cobalt to $\rho_{Co} = 5.6 \times 10^{-8}\,\Omega\,\mathrm{m}$. Another parameter that needs to be specified is the relative permittivity of both materials: it was set respectively to $\varepsilon_{Pt} = 0.7347$ and to $\varepsilon_{Co} = -1.1825$. The value of all the other parameters was left equal to the default one, since they are not needed in performing these specific simulations.

To induce a current distribution inside the structure, a voltage difference is applied across the gate. The voltage is applied only in correspondence of the platinum tracks; however, since cobalt is conductive, a certain distribution of current density will be found in the cobalt layer as well; this topic will be discussed later more in detail.

The skyrmion is driven by the spin-current which originates from the charge current injected inside the HM layer: the SHE converts this charge current into a spin current with an efficiency described by the parameter θ_{SH}, the spin-Hall angle, presented in section 2.1.3.2. Being an efficiency factor, the spin-Hall angle can be at most equal to 1. In the following simulations, its value is always assumed to be exactly 1, but it can be changed into a lower value with very simple modifications. Since $\theta_{SH} = 1$, the values of current density present inside the platinum layer are exactly equal to the values of spin-current density that must be set inside the *mumax*^{3} code to initialize the regions constituting the gate structure: so, it is enough to sample the charge-current density present inside the platinum traces and to use those same values inside the *mumax*^{3} code, without applying any scaling factor.

In order to sample the current density distribution, the platinum traces have been divided into a grid, whose parameters can be set according to the preferences. In the following simulations, the grid is placed exactly in the middle with respect to the thickness of the platinum layer. The horizontal tracks are then divided into a user-defined number of regions, and the samples of current density are taken in correspondence of the central point inside each region. In figure 3.1a and 3.1b are shown the regions chosen for the simulations of the *INV/COPY* and of the *AND/OR* gate. As already mentioned, up to 255 regions could be exploited, but the simulation time would be prohibitive: a compromise between simulation time and resolution has led to the choice of only 23 regions in the case of the *INV/COPY* gate and of 18 regions in the case of the *AND/OR* gate.

The geometrical parameters describing each structure to be simulated, together with the voltage value to be applied across the gate and the spacing of the sampling grid in the horizontal directions, are all provided to *COMSOL Multiphysics* by a *parameters.txt* file. The values sampled by *COMSOL Multiphysics* are then read and plotted by *MATLAB*, which also blends together into a single file all the files containing the current density values sampled by *COMSOL*; finally, a *C* program reads the file provided by *MATLAB*, along with the parameters file describing the structure of the gate, and writes the *mumax*^{3} code accordingly, taking into account also the preferences of the user regarding the type of simulation that must be performed. The *MATLAB* and *C* code used for writing the *mumax*^{3} code for each



(a) (b)

Figure 3.1. Position inside the *INV/COPY* gate (a) and inside the *AND/OR* gate (b) of the regions used for assigning variable current density values in the *mumax*^{3} simulations. Each regions is assigned a single current density value, which is sampled by *COMSOL* in the central point of the corresponding region. The number identifying each region is used in the *mumax*^{3} code for labelling it and for associating it the correct current density value.

simulation are reported in appendix A, together with the *parameter.txt* files describing the structures that will be detailed in the following sections of this chapter.

3.2. *INV/COPY* gate

The first structure that will be considered is the *NOT/COPY* gate: due to its higher complexity, in fact, this gate has a larger number of structural parameters that can be tuned, and so a larger number of degrees of freedom in launching the simulation. This, in turn, permits to obtain many kinds of different results, which will allow a more in depth analysis of the effects that each structural parameter has on the result of the simulation. The topic of the *AND* gate will be covered only when these effects will become very clear.

3.2.1. Uniform current density

The first attempt made has been to deduct the approximative dimensions for the structure of the gate from the figures provided in [5]. Measuring the width and length of each track and comparing them with the length scale of 40 nm provided in the article, the dimensions reported in table 3.2 were obtained. Those parameters are all used for describing the structure of the gate inside the *mumax*^{3} model. From now on, this structure will be referred to as *NOT_Structure 1*.

Track length is the length of the bottom and middle track; concerning the top

Table 3.2. Dimensions for the *NOT* gate - NOT_Structure 1

Parameter	**Value**
track length	256 nm
track width	20 nm
bottom junction width	30 nm
top junction width	25 nm
bottom junction height	20 nm
top junction height	20 nm
x-coordinate junctions start	113 nm
x-coordinate top track start	100 nm
offset top junction	0 nm
external boundary width	34 nm
thickness layer	0.4 nm

track, its starting point is located at *x-coordinate top track start* and its length, if needed, must be computed as *track length* − *x-coordinate top track start*. *Track width* controls the width of the three main tracks, while the horizontal and vertical dimensions of the two junctions are controlled respectively by *bottom(top) junction width* and *bottom(top) junction height*. The x-coordinate for the left wall of both junctions is given by *x-coordinate junctions start*: in this way, both junctions can be rigidly moved along the whole length of the gate. An additional degree of freedom is given by *offset top junction*: this parameter controls the relative position of the top junction with respect to the bottom junction, and it can be positive, negative or null. So, actually, the x-coordinate of the left wall of the top junction is not just *x-coordinate junctions start*, but *x-coordinate junctions start* + *offset top junction*. Finally, a boundary all around each track is needed for confining the skyrmion and avoiding its annihilation due to the skyrmion Hall effect: *external boundary width* controls only the horizontal width of the boundary on the right side of the bottom track and on the left side of the top track, that is, the outermost boundaries; all the other boundaries are chosen so that they fill all the available space left between one track and the other.

Figure 3.2 shows a sketch of the gate where the most important dimensions are reported.

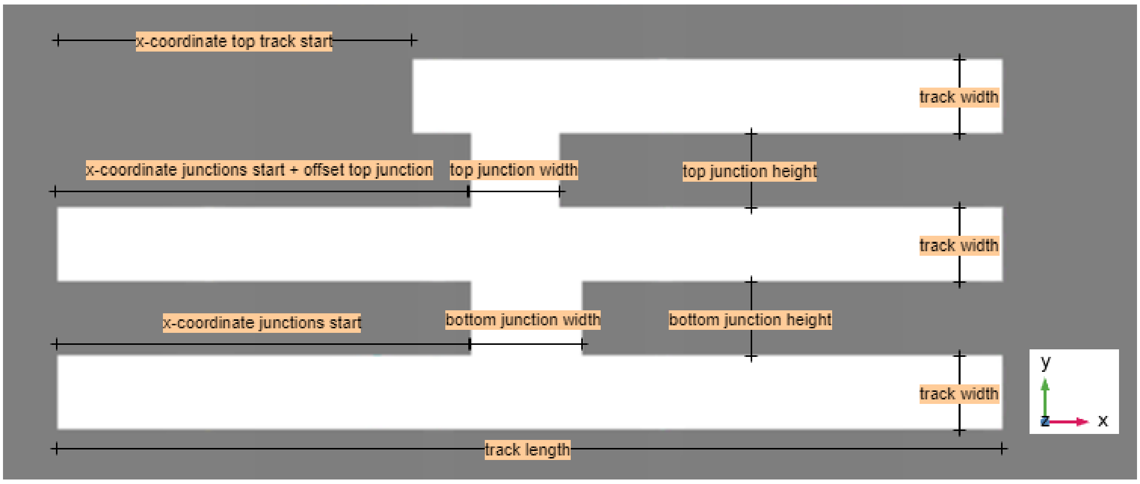


Figure 3.2. Structural parameters of the *INV/COPY* gate.

The thickness used for this first simulation has been of 0.4 nm for both the track and the platinum layer, and so of 0.8 nm for the boundaries (the thickness of the boundaries is equal to the sum of the thickness of the track, plus the thickness of the metal trace below it): this was the thickness value chosen in [5].

The first simulation was performed by imposing a uniform current density of $5 \times 10^{10}\,\mathrm{A/m^2}$ all across the gate, in order to verify its correct functioning in the basic conditions. However, with these simulation parameters, a single skyrmion in the bottom track travels through both junctions and goes out from the top track. So, the behaviour of the gate is wrong. When tested with two skyrmions at once, since the repulsive interaction between the two skyrmions is not enough, the top skyrmion correctly goes out from the top track, while the bottom skyrmion reaches the middle track, tries to go through the top junction, but since its dimension (which has increased while crossing the bottom junction) is too large, it comes back and goes out from the middle track.

In figure 3.3 and 3.4 are shown simulation snapshots taken in some key instants of the skyrmions' movement in the two cases. The shape of the *NOT* gate could be recognized thanks to the tilting of the magnetization vector along the edges of the structure; however, for facilitating the observation of the figures, the position of the internal boundaries of the structure has been highlighted by means of a black line.

The fact that the bottom skyrmion, when alone, is able to reach the top track,

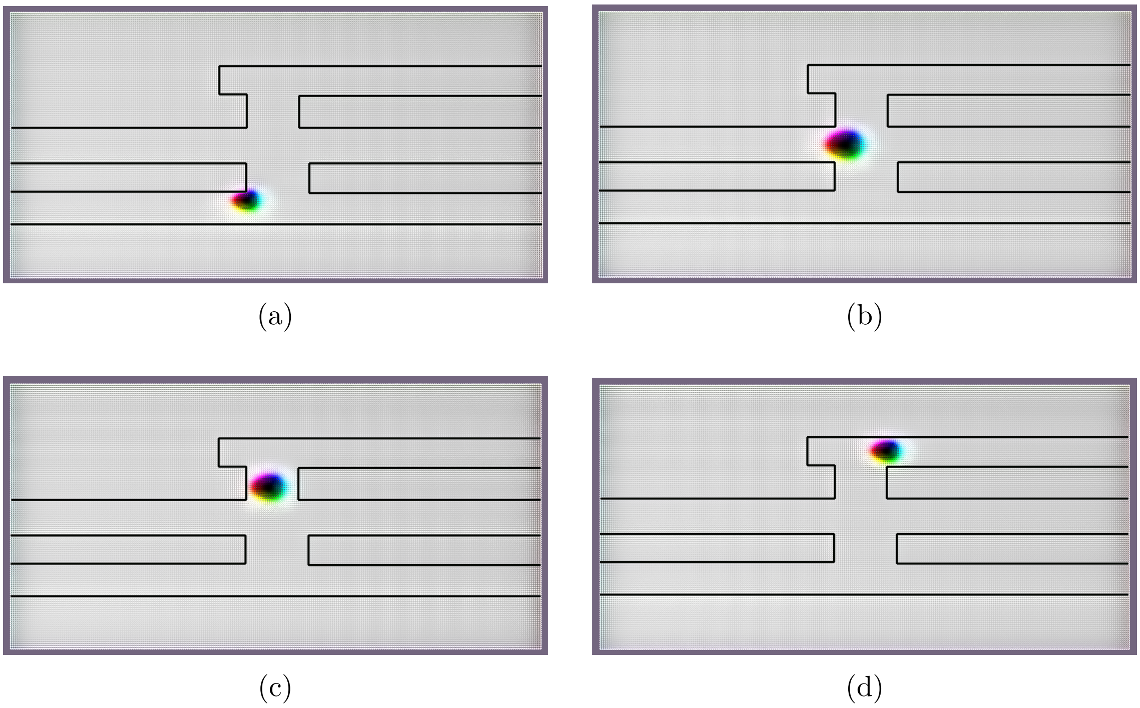
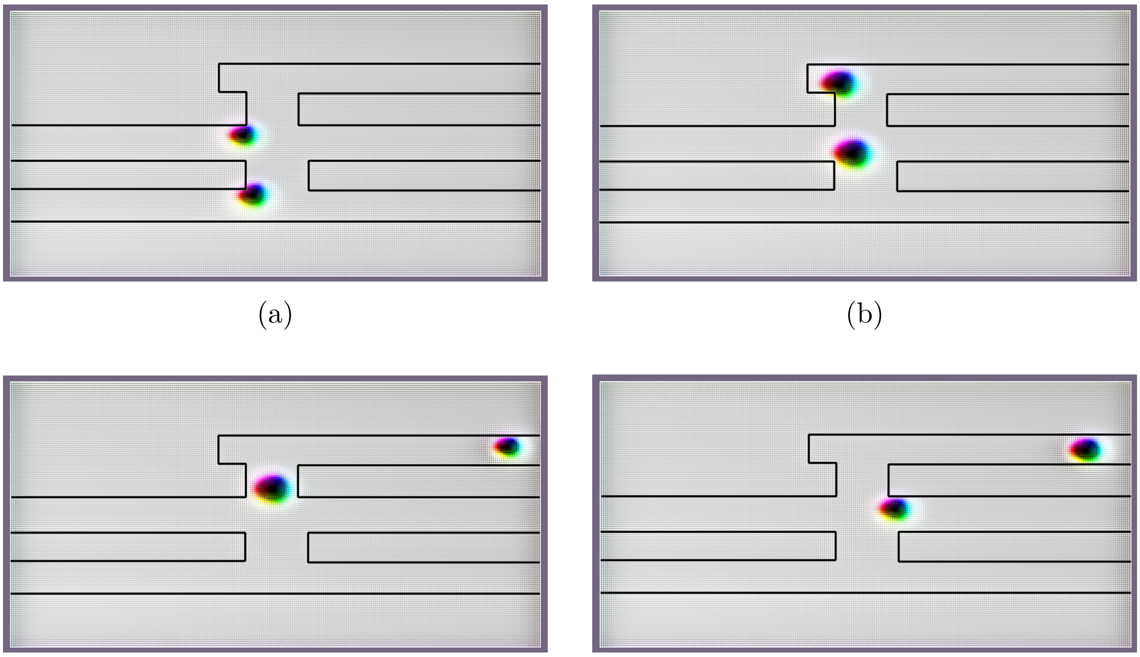


Figure 3.3. *mumax*^{3} simulation of *NOT_Structure 1* with a uniform current density of 5×10^{10} A/m^2. A single skyrmion is able to cross both junctions and reach the top output: the behaviour of the gate is wrong.

means that the width of the two junctions should be reduced in order to prevent this from happening; at the same time, some other geometrical parameter should be tuned in order to increase the skyrmion-skyrmion repulsion when two skyrmions at once are present in the gate.

However, the aim of this chapter is to verify the behaviour of the gates with a non-uniform current density. One can easily imagine that, by changing the distribution of the parameter that makes the skyrmion move, also the behaviour of the skyrmion, together with its timing, is likely to change. So, already knowing that the first version of the *NOT* gate has a low probability of correctly working, the same structure has been tested also with a non-uniform current density, in order to evaluate from the start how much the behaviour of the gate can change by switching to a realistic current distribution.



(a) (b) (c) (d)

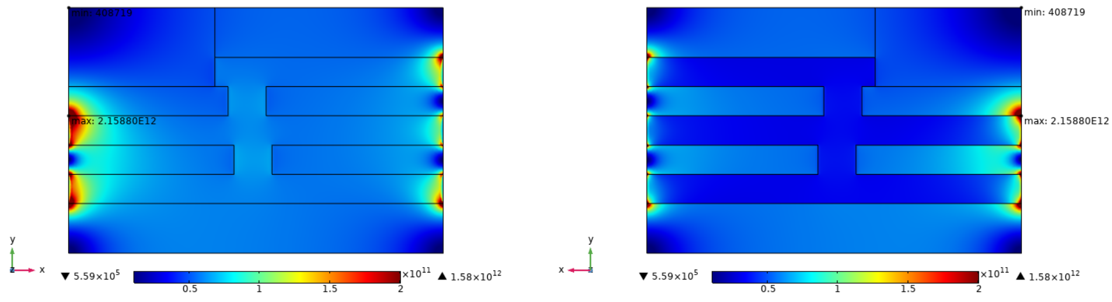
Figure 3.4. *mumax*^{3} simulation of *NOT_Structure 1* with a uniform current density of 5×10^{10} A/m^2. The repulsive interaction between the two skyrmions is not enough and the bottom skyrmion is able to enter the middle track: the behaviour of the gate is wrong.

3.2.2. Realistic current distribution

The voltage value needed to have a current distribution centred around the 5×10^{10} A/m^2 used in [5] is equal to 1 mV. In figure 3.5 is shown the current distribution inside the gate as reported by *COMSOL Multiphysics*, when viewed from the top (only cobalt is visible) and from the bottom (both the platinum traces and the cobalt boundaries are visible).

By applying 1 mV across *NOT_Structure 1*, a non-uniform current density distribution is induced inside the platinum traces. However, as mentioned in section 3.1, the resistivity of cobalt is about one half with respect to the resistivity of platinum: this means that the highest values of current density will be concentrated in the cobalt layer, not in the platinum traces, where they would be more useful (the conversion from charge current to spin current due to the SHE can take place only in the platinum layer).

The main problem that figure 3.5 shows, however, is not related to the mean



(a) (b)

Figure 3.5. Current density distribution estimated by *COMSOL Multiphysics* for *NOT_Structure 1* with a voltage of 1 mV applied, when looking from top (a) and from bottom (b) of the gate. Due to the higher resistivity of platinum, the mean value of the current distribution there is smaller with respect to what happens inside cobalt, as shown in (b).

value of the current density inside the two materials, but to its peak value. The maximum current density value inside the structure is in fact of $2.16 \times 10^{12}\ \mathrm{A/m^2}$, located at the interface between cobalt and platinum. This value is a big problem from an applicative point of view, since the gate is likely to start melting above a value of $1 \times 10^{12}\ \mathrm{A/m^2}$. Moreover, it would be better to remain under a value approximately equal to $20 \times 10^{10}\ \mathrm{A/m^2}$, to avoid that the skyrmion may be expelled from the track due to a too high Magnus force which, from a certain point on, is no more balanced by the repulsion forces from the track edges.

However, is interesting to notice where the highest values of current density are concentrated. They are all located in the points where the voltage is applied: the current in fact is forced to pass through those small areas (the cross-section of platinum, for *NOT_Structure 1*, is of 20 nm × 0.4 nm), and only then it is free to expand in all the rest of the structure, including cobalt, according to the different resistivity values; the same applies to the ground contacts, at the other end of the gate.

Verifying the value of the current density samples with the *MATLAB* script becomes evident how these border effects, due only to how the voltage is applied, influence the behaviour of the current all along the platinum traces. As figure 3.6 shows, the sampled values are highly variable along the trace, and assume the highest values precisely at the two ends. So, using this kind of structure, the behaviour of the skyrmion would be highly influenced by the border effects: as a result, the same gate

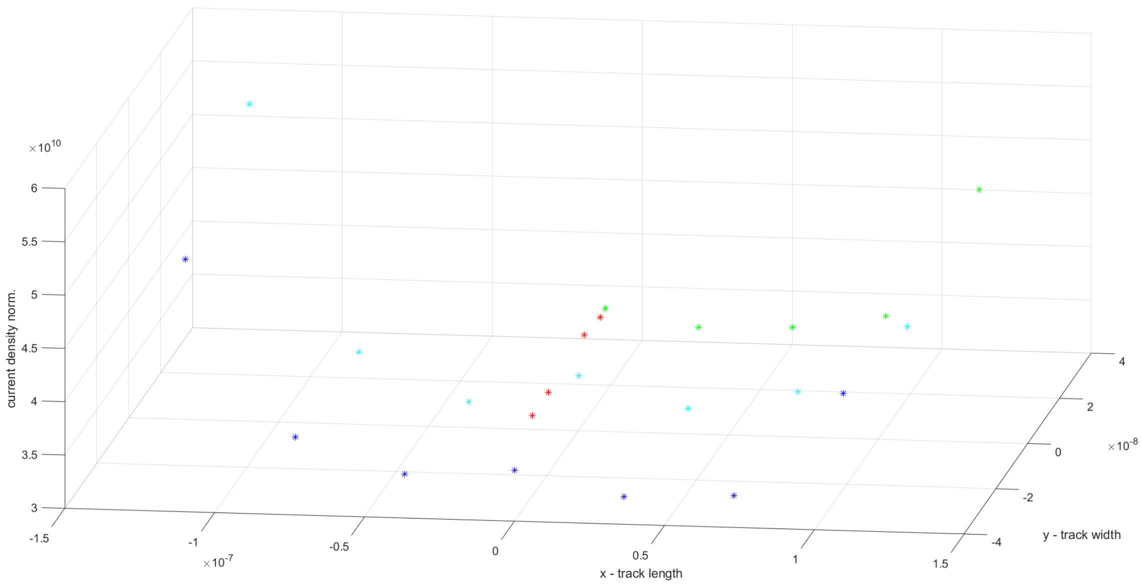
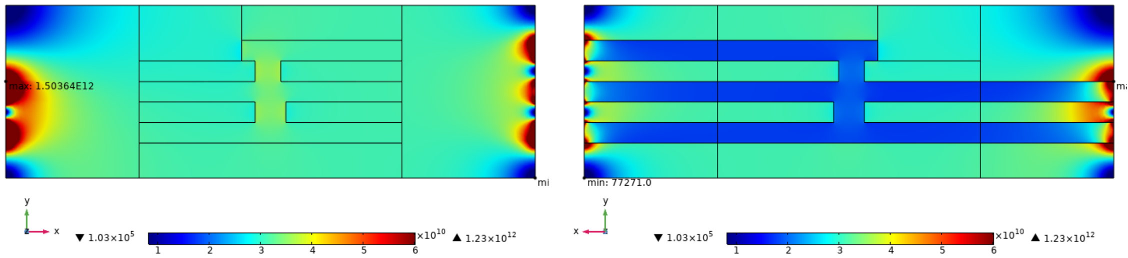


Figure 3.6. *MATLAB* 3D plot of the current density values sampled inside the platinum layer, considering *NOT_Structure 1* with 1 mV applied. The blue points are extracted from the bottom track, the red points from the two junctions, the cyan points from the middle track and the green points from the top track.

would have a different behaviour when inserted inside a circuit composed of many different gates in cascade, with a voltage difference applied only at the beginning and at the end of the circuit.

A solution to this problem could be the insertion of two regions of proper length at the two extremities of the gate, so as to allow the current to stabilize before reaching the actual gate core and thus avoiding that the border effects may influence the skyrmion motion. The length of these stabilization regions has been chosen equal to 130 nm. The current density distribution in the new structure is reported in figure 3.7: looking at the figure is clear that now the border effects are confined far away from the gate, so the current behaviour along the platinum traces is likely to be much more regular. Not only: separating the contact region from the actual gate structure, it is now possible, in principle, to tune the cross-section of the contacts used for applying the voltage difference in order to reduce the maximum value of the current density. Doing so, the peak value of $1.5 \times 10^{12}\ \mathrm{A/m^2}$ reported in figure 3.7 could be reduced down to $20 \times 10^{10}\ \mathrm{A/m^2}$ or even less, according to the preferences and to the space available in the circuit: in this way any damage to the gate structure would be prevented. So, even if it is well above the safe-operating threshold, the maximum current density value is not actually a problem, and can be ignored from now on.



(a) (b)

Figure 3.7. Current density distribution estimated by *COMSOL Multiphysics* for *NOT_Structure 1* with a voltage of 1 mV applied and two stabilization regions 130 nm long (*NOT_Structure 2*), when looking from top (a) and from bottom (b) of the gate. The stabilization regions on the left and on the right of the gate prevent the border effects from reaching the gate core.

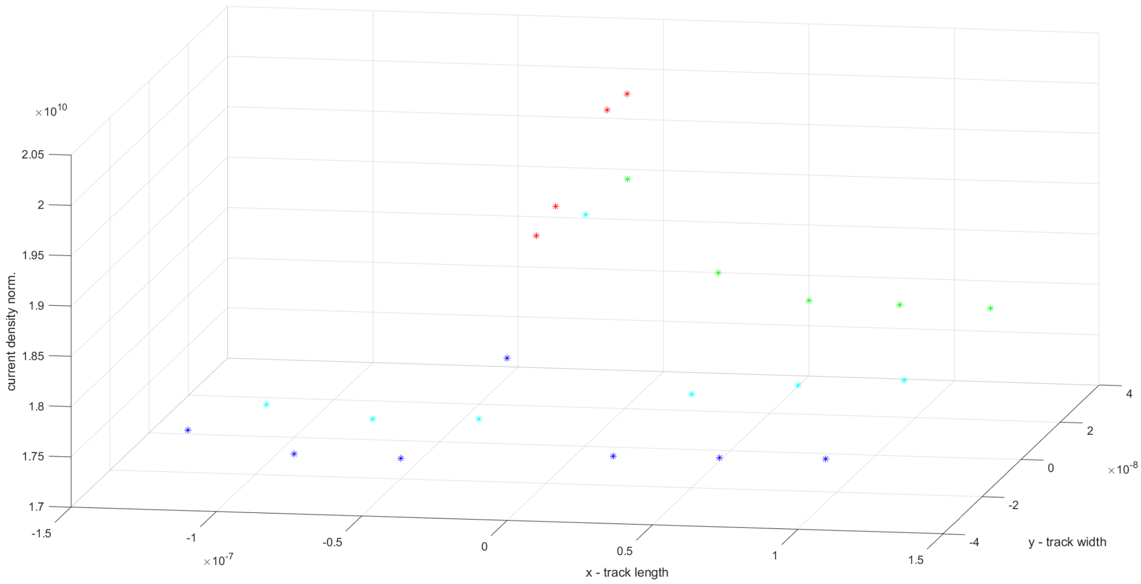


Figure 3.8. *MATLAB* 3D plot of the current density values sampled inside the platinum layer, considering *NOT_Structure 1* with 1 mV applied and two stabilization regions 130 nm long (*NOT_Structure 2*). The blue points are extracted from the bottom track, the red points from the two junctions, the cyan points from the middle track and the green points from the top track. The pattern is now much more regular, but the sampled values are too low.

The behaviour of the current density along the x direction is now much more regular, as figure 3.8 shows (the sampled values, assuming the reference coordinate to be in the middle of the gate, are still from −128 nm to 128 nm). However, pushing away the border effects, the values reached inside the traces are now too low. In order to come back to a mean value of 5×10^{10} A/m^2, is now necessary to apply 2.8 mV at the two ends of the gate.

The parameters describing the new gate structure are summed up in table 3.3. This structure will be referred to as *NOT_Structure 2*: it is exactly equal to *NOT_-Structure 1*, apart from the insertion of the two stabilization regions, whose length is defined by the parameter *size contact*.

Table 3.3. Dimensions for the *NOT* gate - NOT_Structure 2

Parameter	**Value**
size contact	130 nm
track length	256 nm
track width	20 nm
bottom junction width	30 nm
top junction width	25 nm
bottom junction height	20 nm
top junction height	20 nm
x-coordinate junctions start	113 nm
x-coordinate top track start	100 nm
offset top junction	0 nm
external boundary width	34 nm
thickness layer	0.4 nm

Since the structure of the gate core hasn't changed, when excited with a uniform current density the gate behaves exactly like reported in section 3.2.1, so there is no need to repeat the simulations already discussed.

When excited with a realistic current density, the overall behaviour of the gate doesn't change: a single skyrmion goes out from the top track, and when two skyrmions are present one goes out from the middle track and one from the top track. This means that the current distribution obtained with the elongated structure is sufficiently uniform, so the behaviour of the skyrmion and its timing don't change significantly. However, this doesn't mean that, when switching from a uniform to a realistic current density, the behaviour always remains the same: exciting *NOT_Structure 1* with the current distribution shown in figure 3.6, in fact, when a single skyrmion is present, it is the middle output the one that switches to 1, and not the top output as it happened imposing a uniform current density. The current distribution shown in figure 3.6 in fact is far from uniform, to a point that it is enough to substantially change the behaviour of the skyrmion.

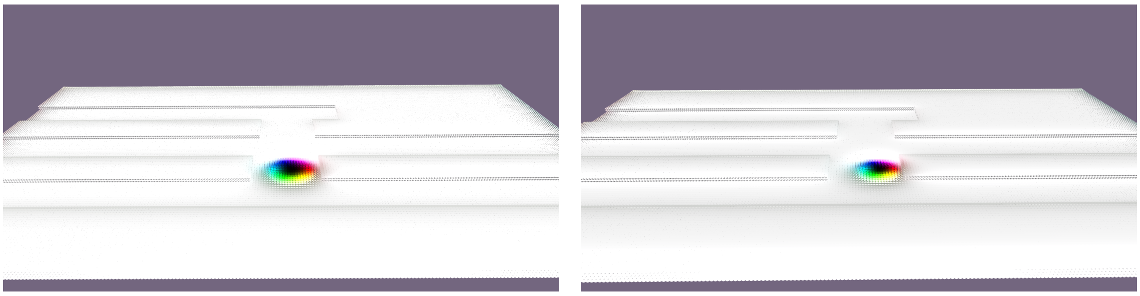
3.2.3. Final version

As already mentioned, the fact that the bottom skyrmion is able to reach the top track when it is the only particle present inside the gate could mean that the width of the two junctions must be reduced. This modification leads to *NOT_Structure 3*, whose parameters are summed up in table 3.4.

Table 3.4. Dimensions for the *NOT* gate - NOT_Structure 3

Parameter	**Value**
size contact	130 nm
track length	256 nm
track width	20 nm
bottom junction width	27 nm
top junction width	25 nm
bottom junction height	20 nm
top junction height	20 nm
x-coordinate junctions start	113 nm
x-coordinate top track start	100 nm
offset top junction	0 nm
external boundary width	34 nm
thickness layer	0.8 nm

The parameter *bottom junction width* has been reduced from 30 nm to 27 nm. Also the parameter *thickness layer* has changed: the reason is that the lattice constant of cobalt is equal to 406.95×10^{-12} pm, while that of platinum is of 392.42×10^{-12} pm. A thickness of only 0.4 nm is technologically very difficult to be realized for both materials, since it implies the deposition of a single atomic layer. Doubling the thickness the technological realization should become a bit easier; this change, moreover, influences also the behaviour of the skyrmion, since it modifies the current distribution inside the gate. The thickness of the layer has apparently an effect also on the size of the skyrmion: in figure 3.9 is reported the comparison between *NOT_Structure 3* and the same structure, but with *thickness layer* equal to 0.4 nm (*NOT_Structure 5*: its parameters *.txt* file is reported in appendix A). In both pictures the skyrmion is crossing the bottom junction, so is possible to make a comparison, in the two cases, between its size and the width of the bottom junction, equal to 27 nm in both structures. It is clear that in the case of *thickness layer* = 0.4 nm the skyrmion is smaller: as a result, looking at the remaining part of the simulation, the same skyrmion is able to cross also the top junction and to go out from the top track, making the gate fail the computation.

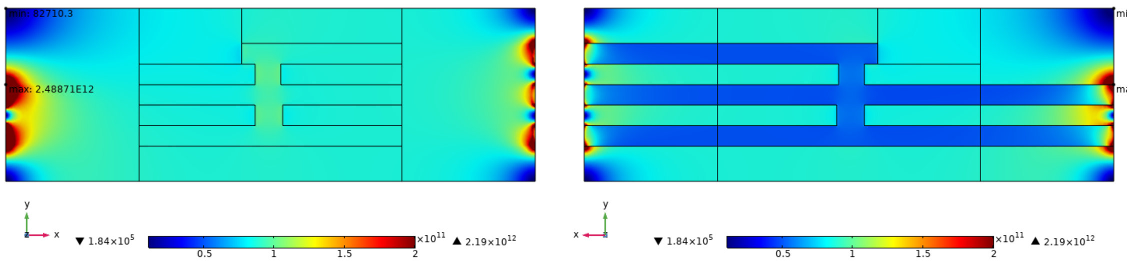


(a) (b)

Figure 3.9. Comparison between *NOT_Structure 3* (a) and the same geometry with *thickness layer* = 0.4 nm (b). In the latter case the skyrmion size is smaller: this allows the skyrmion to enter the top junction and go out from the top output, making the computation fail. So, the thickness of the layer influences the skyrmion's size and the computation results. The gate is viewed from the bottom to better highlight the position of the boundaries.

However, the increase in the skyrmion size obtained by doubling *thickness layer* is not enough, alone, to confine the bottom skyrmion in the bottom track when the gate hosts two skyrmions at once. Simulating *NOT_Structure 2* with *thickness layer*= 0.8 nm (*NOT_Structure 6*: again, its parameters *.txt* file is reported in appendix A), the increased size is enough to prevent the bottom skyrmion, when alone, from entering the top 25 nm-wide junction, but as just mentioned the bottom junction is still to wide, so that the repulsive interaction between the two skyrmions is not enough to make the bottom skyrmion change direction: as a result, both the top and the middle output switch to 1 and the gate fails. This is why both *thickness layer* and *bottom junction width* have been changed when defining *NOT_Structure 3*.

Figure 3.10 and figure 3.11 report respectively the current distribution inside the structure when viewed from top and from bottom and the values of the samples collected in the platinum layer.



(a) (b)

Figure 3.10. Current density distribution estimated by *COMSOL Multiphysics* for *NOT_Structure 3* with a voltage of 2.8 mV applied, when looking from top (a) and from bottom (b) of the gate.

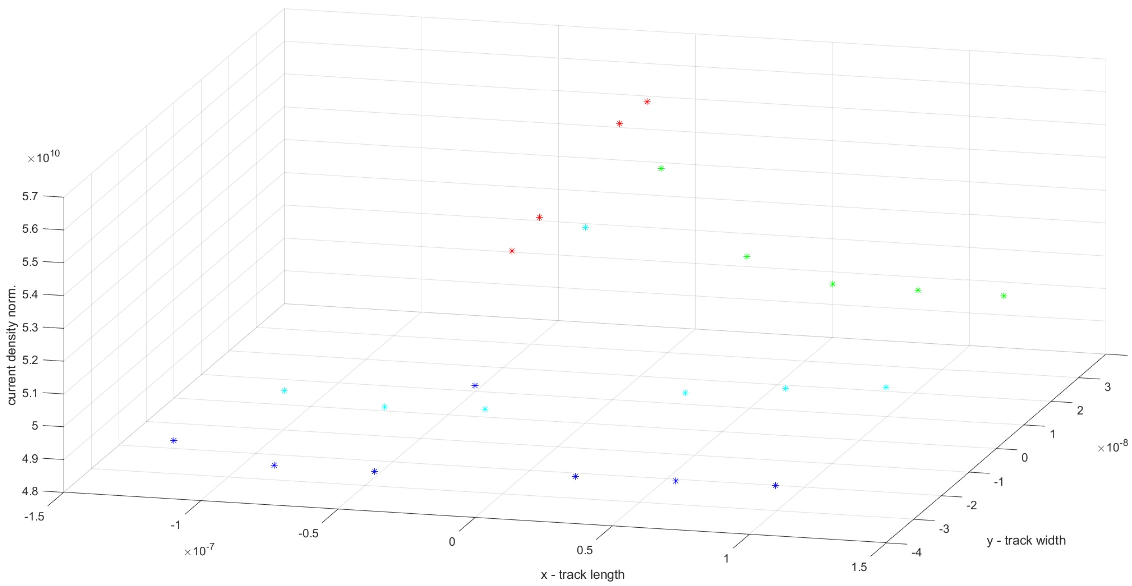
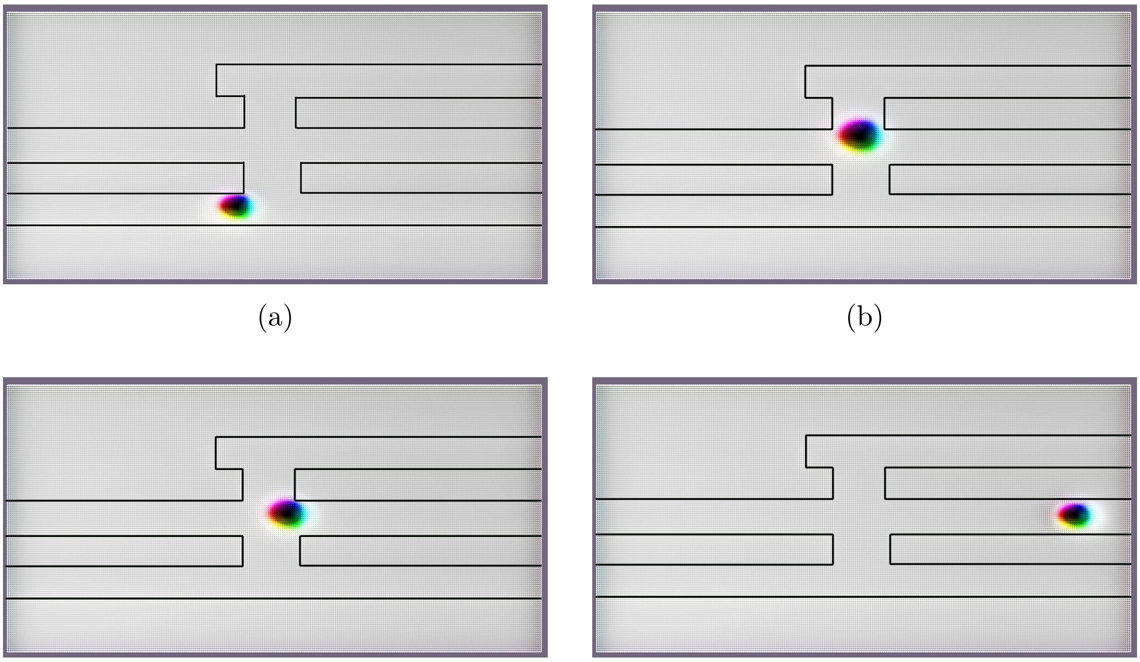


Figure 3.11. *MATLAB* 3D plot of the current density values sampled inside the platinum layer, considering *NOT_Structure 3* with 2.8 mV applied. The blue points are extracted from the bottom track, the red points from the two junctions, the cyan points from the middle track and the green points from the top track.

This version of the gate is fully working, as demonstrated by the simulation snapshots reported in figure 3.12 and 3.13. When a single skyrmion is present, after crossing the bottom junction and trying to enter the top junction, it comes back and goes out from the middle output. This happens because the increased *thickness layer* has made the skyrmion size increase, thus preventing the skyrmion from entering the top junction.



(a) (b) (c) (d)

Figure 3.12. *mumax*^{3} simulation of *NOT_Structure 3* with the current density obtained applying 2.8 mV. A single skyrmion crosses the bottom junction and goes out from the middle output: the behaviour of the gate is correct.

When two skyrmions are injected the repulsive interaction, assisted by the reduced *bottom junction width*, is enough to make the bottom skyrmion change direction and go back inside the bottom track, while the top skyrmion can follow its path along the top track. Not only: with these geometrical parameters, the timing of the two skyrmions hasn't a large difference, so the two outputs are almost synchronized with each other, as shown in figure 3.13d. Moreover, in some structures with different geometrical parameters, it might happen that the repulsive interaction between the bottom and the top skyrmion at the bottom junction could make the top skyrmion move backwards a bit, as long as the other skyrmion is nearby. This phenomena of course slows down the computation. However, this doesn't happen in this particular structure: the movement of the skyrmions is smooth and almost perfectly synchronized. This means that there is no time wasted during the computation, and so the gate could hardly become faster while maintaining the same voltage applied. For all these reasons, this version of the *INV/COPY* gate will be considered the final one.

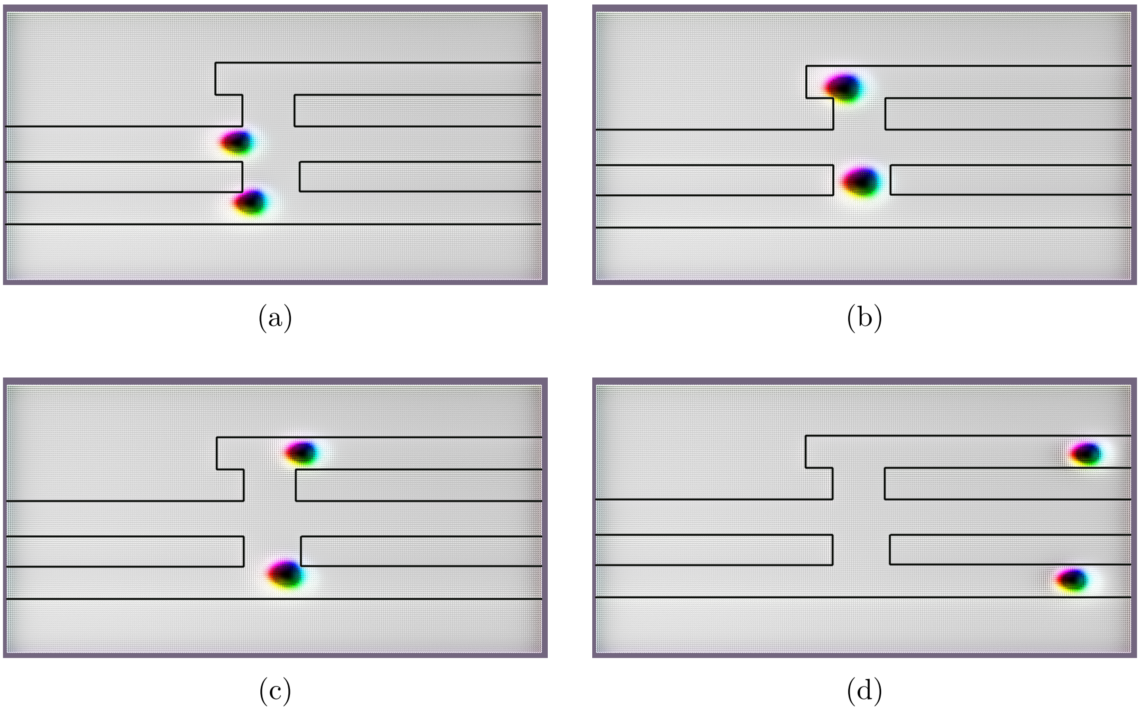


Figure 3.13. *mumax*^{3} simulation of *NOT_Structure 3* with the current density obtained applying 2.8 mV. : the behaviour of the gate is correct, so *NOT_-Structure 3* is fully working.

Of course, *NOT_Structure 3* is not the only geometry able to correctly work. For example, another version could be the one characterized by the parameters reported in table 3.5. However, in this case the timing difference between the two skyrmions becomes much larger: when the top skyrmion arrives at the end of the top track, the bottom skyrmion is still entering the bottom track. Still, the point that should be underlined here is that, once a completely working version has been found, other correct versions can be derived from it by slightly varying one or more parameters at once.

Table 3.5. Dimensions for the *NOT* gate - NOT_Structure 4

Parameter	Value
size contact	130 nm
track length	256 nm
track width	20 nm
bottom junction width	26 nm
top junction width	26 nm
bottom junction height	20 nm
top junction height	20 nm
x-coordinate junctions start	113 nm
x-coordinate top track start	100 nm
offset top junction	-4 nm
external boundary width	34 nm
thickness layer	0.8 nm

3.3. *AND/OR* gate

With the experience acquired in tuning the parameters of the *INV/COPY* gate in order to make everything work, finding the proper structure able to implement the *AND/OR* functions becomes really straightforward, also thanks to the reduced number of parameters that must be managed.

The methodology remains the same: first the solution proposed in [5] is tested both with a uniform and with a realistic current density (this time the stabilization regions 130 nm long will be included from the very beginning); the results of the simulations will then be discussed in order to find a correct version with acceptable performances by properly tuning *thickness layer* and some other parameters.

The material parameters used in the *mumax*^{3} simulations remain the ones reported in table 3.1; also the parameters used in the *COMSOL Multiphysics* simulations didn't change from the ones discussed in section 3.1.

3.3.1. Uniform current density

The dimensions deducted from the figures of [5] are reported in table 3.6. The resulting structure will be referred to as *H_Structure 1*.

Table 3.6. Dimensions for the *AND/OR* gate - H_Structure 1

Parameter	Value
size contact	130 nm
track length	256 nm
track width	20 nm
junction width	30 nm
junction height	20 nm
external boundary width	34 nm
thickness layer	0.4 nm

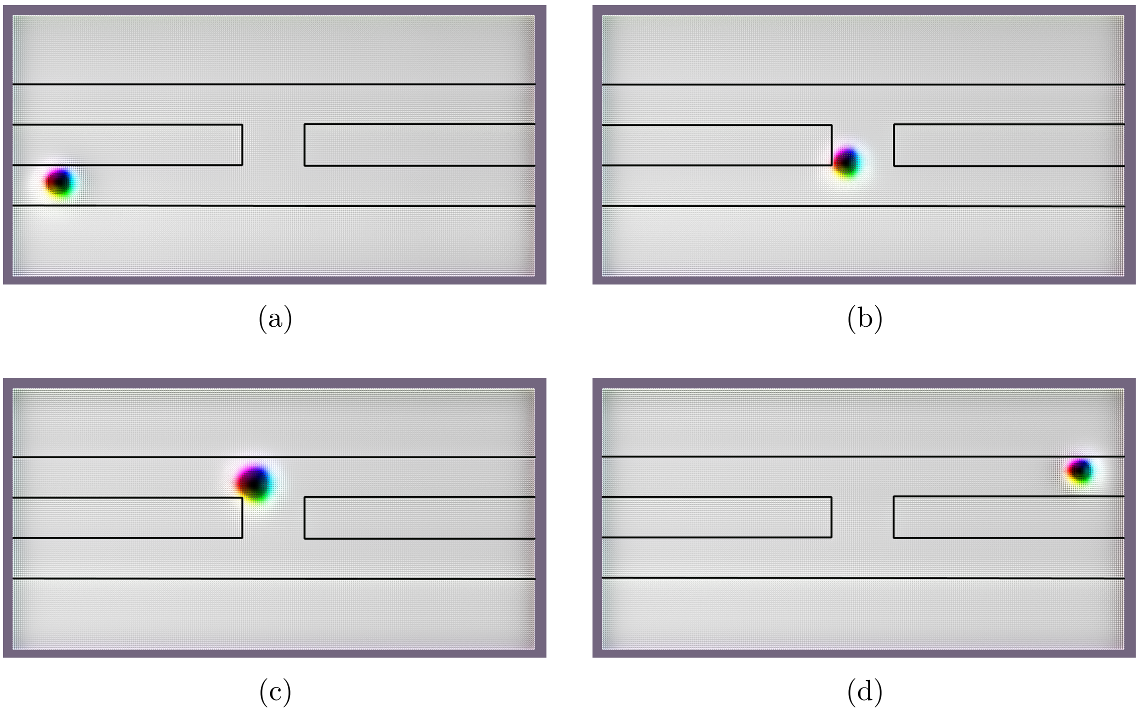
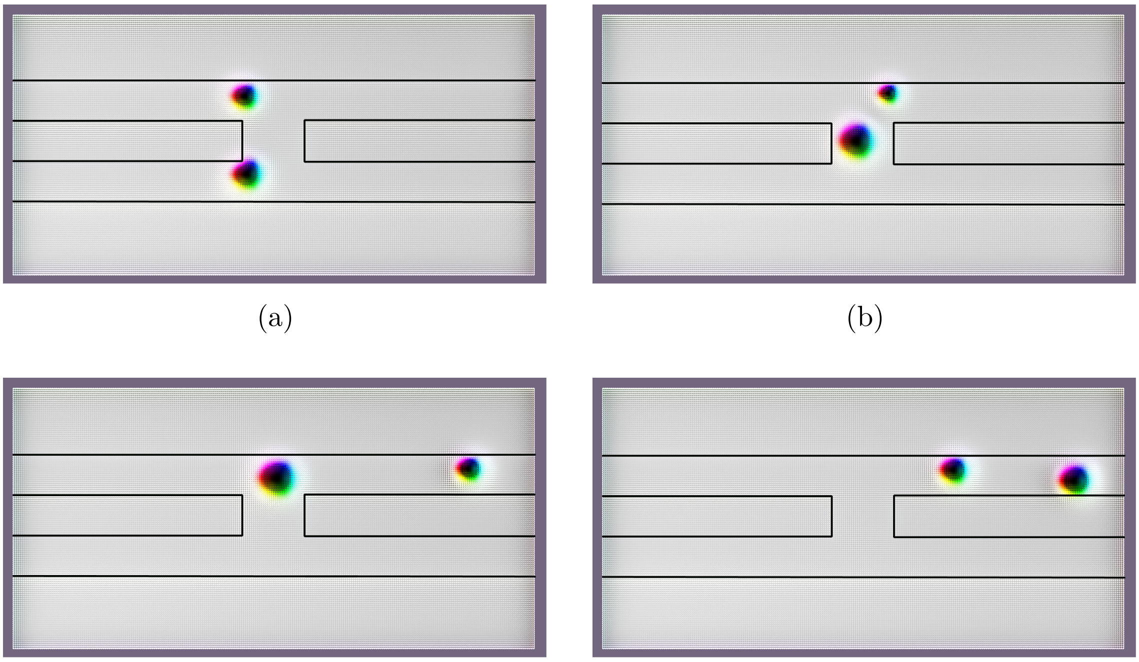


Figure 3.14. *mumax*^{3} simulation of *H_Structure 1* with a uniform current density of 5×10^{10} A/m^2. A single skyrmion is able to cross the junction and reach the top output: the behaviour of the gate is correct.



(a) (b) (c) (d)

Figure 3.15. *mumax*^{3} simulation of *H_Structure 1* with a uniform current density of 5×10^{10} A/m^2. The repulsive interaction between the two skyrmions at the junction is not enough to confine the bottom skyrmion inside the bottom track: the behaviour of the gate is wrong.

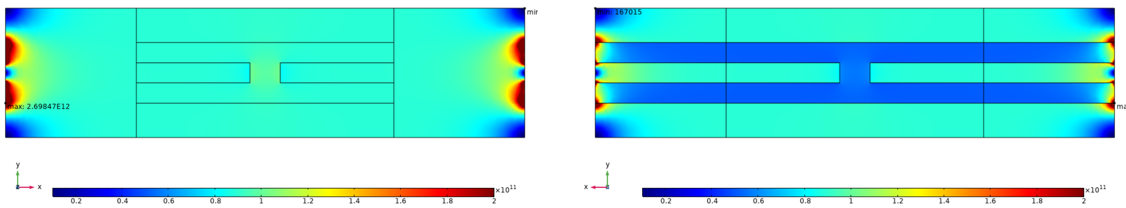
Imposing a uniform current density of 5×10^{10} A/m^2 and simulating the behaviour of a single skyrmion in the bottom track, the gate works as expected and the skyrmion exits from the top output. However, with two skyrmions at once the gate fails: when the bottom skyrmion is crossing the junction, the repulsive interaction is enough to make the top skyrmion shrink in size, but it is not strong enough to confine the bottom skyrmion inside the bottom track. As a result, both skyrmions go out from the top output.

Some simulation snapshots of the two conditions are shown in figure 3.14 and in figure 3.15.

3.3.2. Realistic current distribution

When imposing a voltage of 2.8 mV across the gate described in table 3.6, the current density distribution obtained is the one shown in figure 3.16. Like it happened with the *INV/COPY* gate, the maximum current density obtained (here equal to $2.7 \times 10^{12}\,\mathrm{A/m^2}$) is well above the critical threshold of $20 \times 10^{10}\,\mathrm{A/m^2}$; however, thanks to the possibility of patterning the contacts, which are now separated from the actual gate structure, this peak value is not a problem and it can be ignored.

Also in this case a length of 130 nm for the two stabilization regions is enough for avoiding that the border effects can influence the skyrmion motion. This is confirmed also by the pattern of the sampled current density values provided by the *MATLAB* script: the plot is reported in figure 3.17.



(a) (b)

Figure 3.16. Current density distribution estimated by *COMSOL Multiphysics* for *H_Structure 1* with a voltage of 2.8 mV applied, when looking from top (a) and from bottom (b) of the gate. The border effects are far from the core of the gate.

As it happened for the *INV/COPY* gate, when switching from the uniform current density to a realistic distribution, the overall behaviour of the gate doesn't change: a single skyrmion makes the top output switch to 1, but when two skyrmions are present they both go out from the same track. This is the third and final confirmation: the current density inside the gate is almost uniform, so also the behaviour of the skyrmion doesn't change too much.

3.3.3. Final version

As already discussed in section 3.2.3, both the thickness of the layers and the width of the junctions can be used to control the size and so the motion of the skyrmion. In this particular case, the repulsive interaction between the two skyrmions is not strong enough, so it is necessary to make the crossing of the junction a bit more difficult for the bottom skyrmion, in order to assist the repulsive interaction between skyrmions with the repulsive force between the skyrmion and the track boundaries. To do so, is possible either to increase the thickness of the two layers,

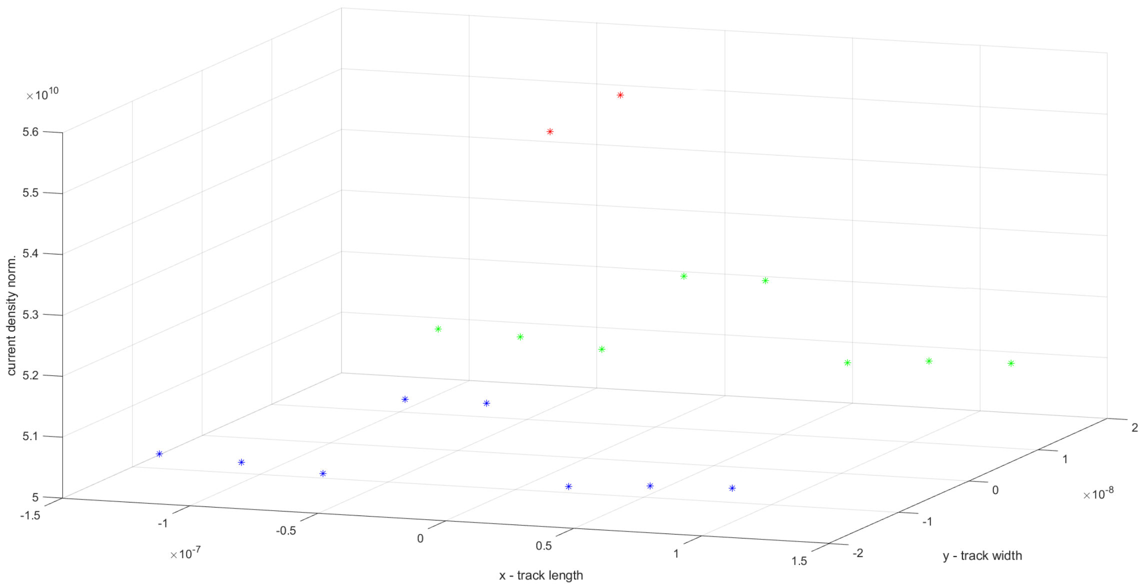


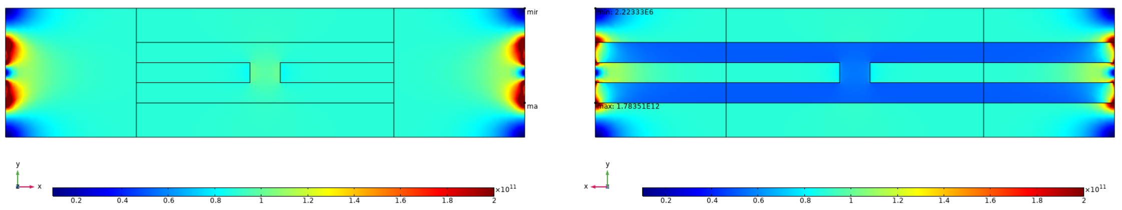
Figure 3.17. *MATLAB* 3D plot of the current density values sampled inside the platinum layer, considering *H_Structure 1* with 2.8 mV applied. Assuming $\theta_{SH} = 1$, these values will be used as they are inside the $mumax^3$ code. The blue values are sampled inside the bottom track, the red values inside the junction and the green values inside the top track. The pattern along the x direction, especially at the two sides of the track, is quite uniform: this means that the border effects are avoided.

or to reduce the junction width; it could happen, however, that both changes need to be made, as it happened in the case of the *INV/COPY* gate.

Table 3.7. Dimensions for the *AND/OR* gate - H_Structure 2

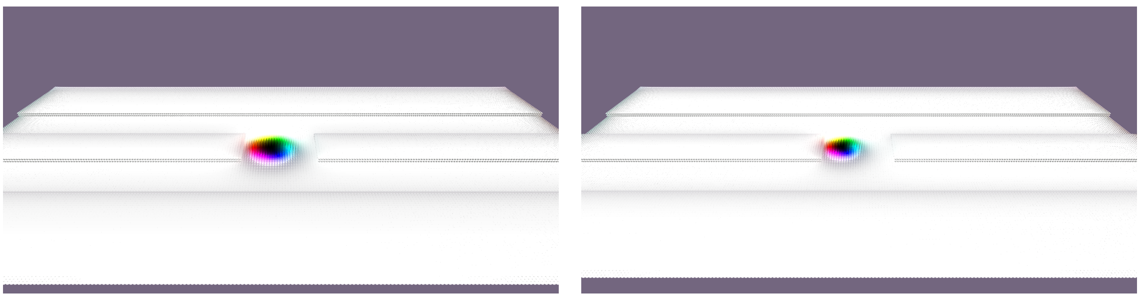
Parameter	Value
size contact	130 nm
track length	256 nm
track width	20 nm
junction width	30 nm
junction height	20 nm
external boundary width	34 nm
thickness layer	0.8 nm

The first attempt made was to double the parameter *thickness layer*, for the reasons explained in section 3.2.3. This leads to the definition of *H_Structure 2*, described in table 3.7. The current distribution inside the structure is reported in figure 3.18.



(a) (b)

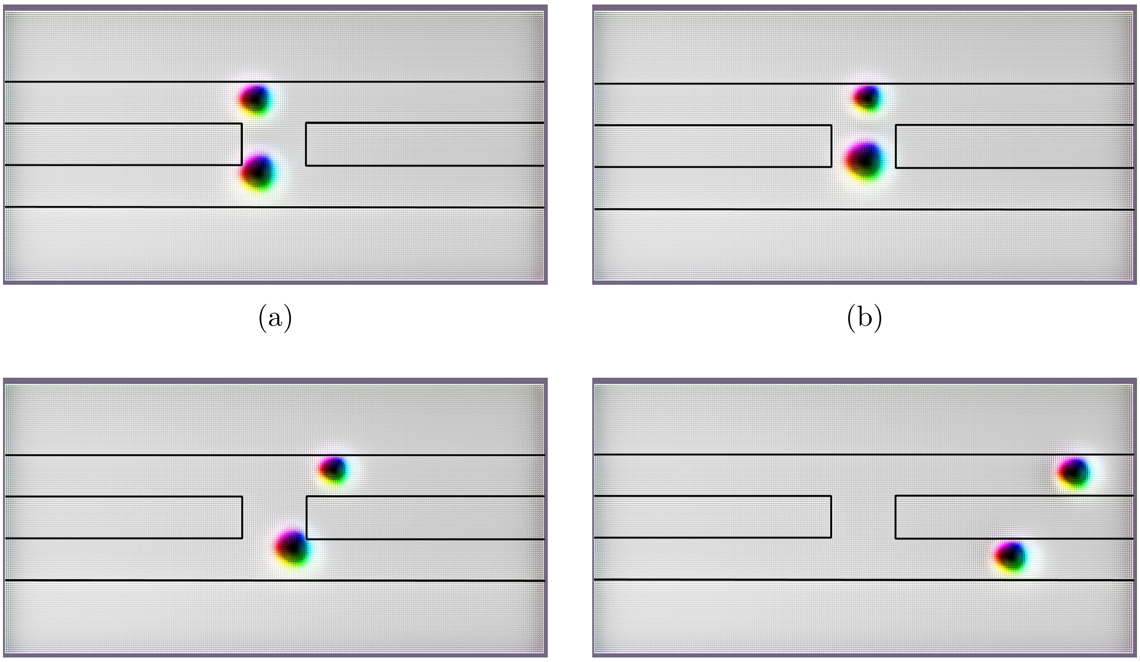
Figure 3.18. Current density distribution estimated by *COMSOL Multiphysics* for *H_Structure 2* with a voltage of 2.8 mV applied, when looking from top (a) and from bottom (b) of the gate.



(a) (b)

Figure 3.19. Comparison between *H_Structure 2* (a) and the same geometry with *thickness layer* = 0.4 nm (b) (*H_Structure 1*): in the latter case the skyrmion size is smaller. The gate is viewed from the bottom to better highlight the position of the boundaries.

In figure 3.19 is shown the effect of the increase of *thickness layer*: as it happened for the *INV/COPY* gate, when the thickness is doubled also the skyrmion size increases. Figure 3.20 shows that this time this size increase alone is enough to correct the result of the computation: thanks to the increased repulsion from the track boundaries, this time the repulsive interaction between the two skyrmions is enough to confine the bottom skyrmion inside the bottom track. At the same time, the behaviour of the gate with a single skyrmion inside the bottom track doesn't change: the top output still correctly switches to 1, since the bottom skyrmion is still able to cross the junction and reach the top track. So, *H_Structure 2* is fully working. Also in this case the interaction between the two skyrmions at the junction is not strong enough to make the top skyrmion move backwards inside the top track as long as the bottom skyrmion is nearby. However, here the synchronization of the two skyrmions is less accurate, and the time difference between the two arrival moments is larger with respect to the one obtained with *NOT_Structure 3*. Still, this version is fully working and will be considered the final one.



(a) (b) (c) (d)

Figure 3.20. *mumax*^{3} simulation of *H_Structure 2* with the current density obtained by applying 2.8 mV. The repulsive interaction between the two skyrmions at the junction is aided by the increased skyrmion-track edges repulsion, so that the bottom skyrmion remains confined inside the bottom track: the behaviour of the gate is correct, and *H_Structure 2* is fully working.

As already discussed, this doesn't mean that this is the only working structure. Small modifications to one or more parameters at once can be made, finding many different versions, all fully working. For example, one possibility could be to reduce both the width and the height of the junction, apart from increasing the thickness of the two layers. This is what was done for structure *H_Structure 3*, described in table 3.8.

Table 3.8. Dimensions for the *AND/OR* gate - H_Structure 3

Parameter	Value
size contact	130 nm
track length	256 nm
track width	20 nm
junction width	25 nm
junction height	14 nm
external boundary width	34 nm
thickness layer	0.8 nm

3.4. Balancing of current density

Now that both the *INV/COPY* gate and the *AND/OR* gate have been studied and verified, a final optimization should be made. Since the resistivity of cobalt is about half with respect to the resistivity of platinum ($\rho_{Co} = 5.6 \times 10^{-8}\,\Omega\,\mathrm{m}$ and $\rho_{Pt} = 9.8 \times 10^{-8}\,\Omega\,\mathrm{m}$), the highest concentrations of current density are found in the cobalt layer, not in the platinum traces. This raises a problem of efficiency: only the charge current flowing inside the platinum traces, in fact, is available for the conversion into spin-current due to the SHE, and the skyrmion moves only thanks to the spin-current. It could be put into movement also by the STT mechanism (presented in section 2.1.3.1) by the charge current which diffuses inside the cobalt layer, but it has been said in section 2.1.3.3 that the driving efficiency of the CPP configuration is much higher with respect to that of the CIP: in [20] it has been proven that, with $\beta = 0.6$ (while in our case $\beta = 0.35$), with $J = 5 \times 10^6\,\mathrm{A/cm^2}$ the velocity induced by the STT mechanism is about ten times lower with respect to the one due only to the SHE. For this reason, the STT contribution to the skyrmion motion can be reasonably neglected. So, it is only a matter of efficiency: it could be useful to find a structure where the vertical and horizontal dimensions are tuned so that the current density inside platinum is closer to the one that can be found in cobalt. This is the topic of this section. The structures that will be presented here, however, won't be studied via micromagnetic simulations, so their correct behaviour is not guaranteed.

To analyse the results provided by *COMSOL Multiphysics*, a new *MATLAB* script has been developed. The code can be found in appendix A, together with the *parameters.txt* files used for the current distribution analysis.

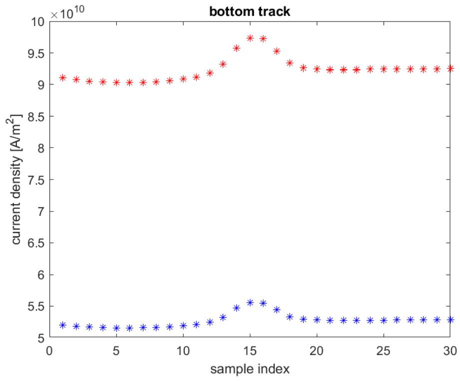
3.4.1. *INV/COPY*

3.4.1.1. Original version

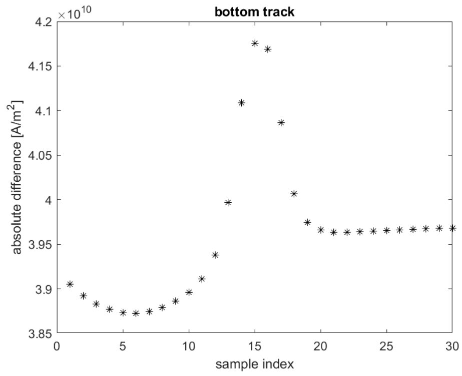
The first structure to be analysed is *NOT_Structure 3*, which is the final version found in section 3.2.3. The results of this analysis will serve as a reference for the following optimizations.

The *COMSOL Multiphysics* simulation file has been slightly changed to allow an automatic sampling in the whole structure. So, this time the sampling grid is placed both in the middle of the platinum layer and in the middle of the cobalt layer; the samples are still taken in correspondence of the middle point inside each sampling cell, and the user is still able to vary the number of sampling cells by means of the *parameters.txt* file. In this particular case there is no limitation on the number of samples that can be taken, because these samples won't be used for a $mumax^3$ simulation: for this reason, the sampling resolution has been slightly increased.

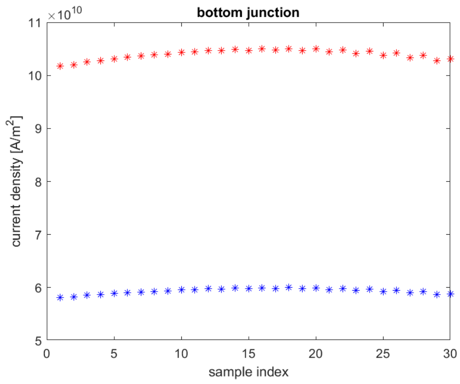
Two sets of samples are taken for each part of the gate structure: one set comes from the platinum layer, the other from the cobalt layer. For example, considering the bottom track, 30 samples are taken along the x direction inside the platinum layer and 30 samples inside the cobalt layer. So, considering the sample number $\#i$ inside the array taken from the cobalt layer and the sample with the same index inside the array taken from the platinum layer, the x and y coordinates will be the same, while the only difference will be in the z coordinate and of course in the value of the sampled current density. For this reason, is possible to plot both arrays on a graph having as x coordinate the index of the sample (plots 3.21a, 3.21c, 3.21e, 3.21g and 3.21i), and is possible also to compute the difference between the two arrays in order to plot the absolute difference, value by value, with respect to the sample index (plots 3.21b, 3.21d, 3.21f, 3.21h and 3.21j).


bottom track
current density [A/m^2]
sample index

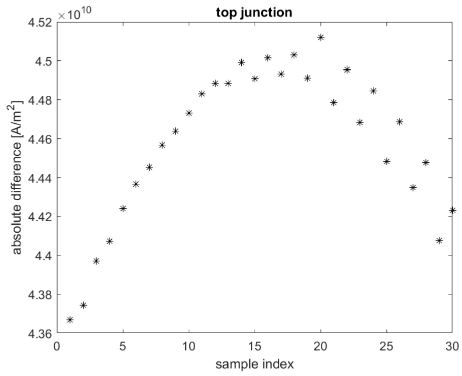
(a)


bottom track
absolute difference [A/m^2]
sample index

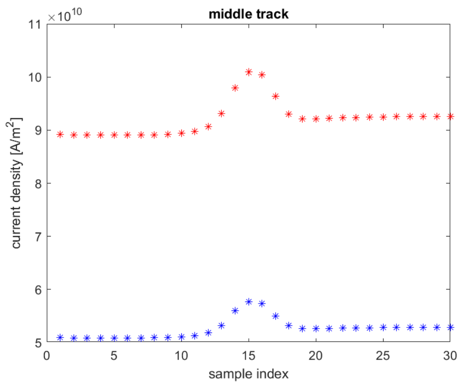
(b)


bottom junction
current density [A/m^2]
sample index

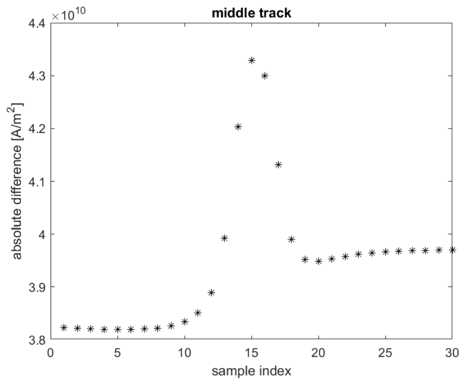
(c)


top junction
absolute difference [A/m^2]
sample index

(d)


middle track
current density [A/m^2]
sample index

(e)


middle track
absolute difference [A/m^2]
sample index

(f)

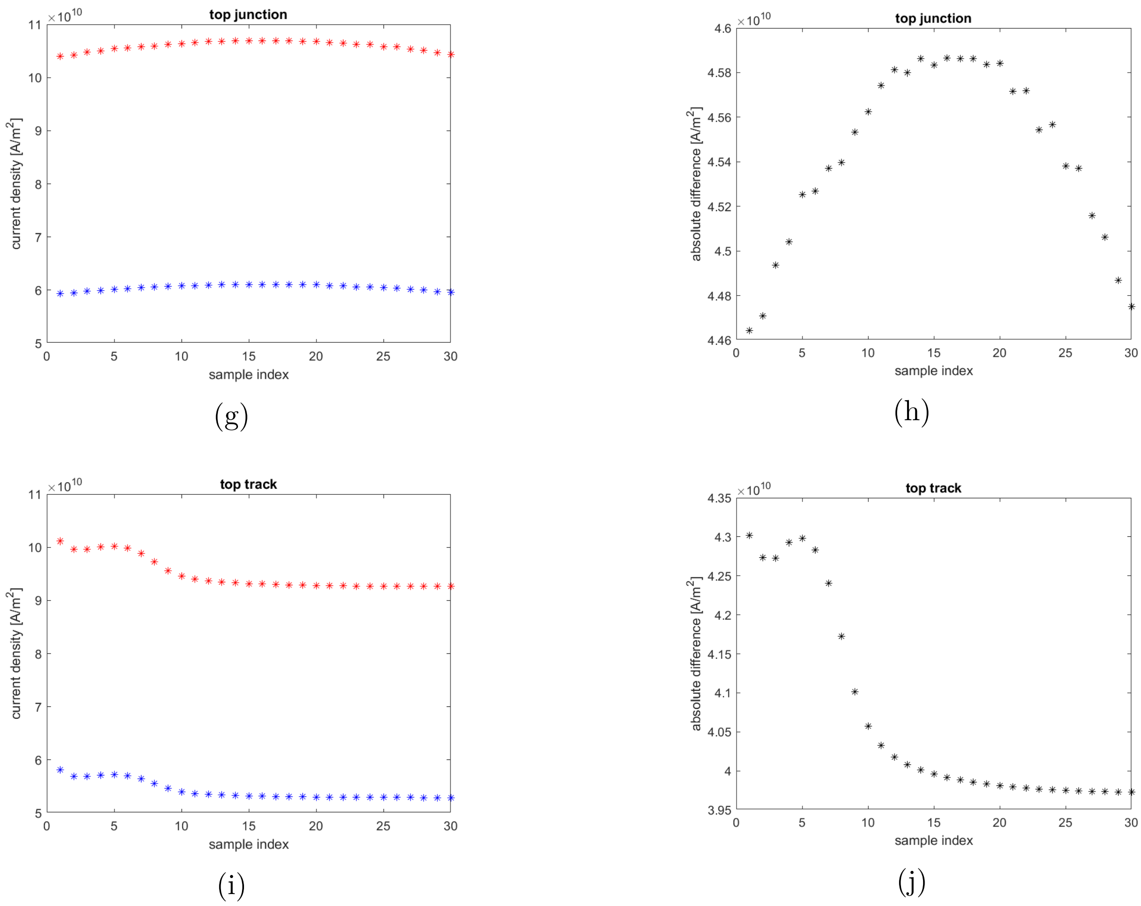
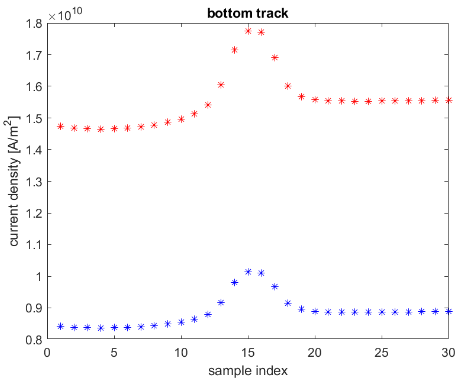


Figure 3.21. Current density samples taken in the platinum and in the cobalt layers inside the original version of the *INV/COPY* gate. Plots (a), (c), (e), (g) and (i) show the sampled values (the red samples come from cobalt, the blue samples from platinum), while plots (b), (d), (f), (h) and (j) show the absolute difference computed value-by value. The average absolute difference is around 4.4×10^{10} A/m^2.

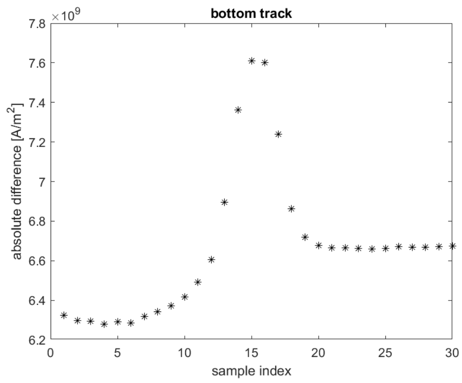
It can be observed from the right column of plots, all showing the absolute difference value-by-value, that the difference between the two current densities is not very large. However, as already discussed, the more this value is reduced, the more the efficiency of the structure will improve.

3.4.1.2. First version

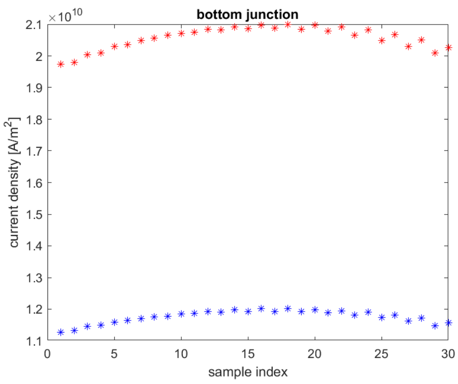
The first attempt has been to increase the thickness of the platinum layer while maintaining fixed the cobalt thickness, and at the same time to apply a scaling factor greater than 1 to the dimensions characterizing the gate core. In the results shown in figure 3.22, the parameter *thickness layer Pt* has been increased up to 80 nm, while all the horizontal dimensions of the track have increased by a factor 6. Doing so, the gate maintains the same proportions and becomes a bit larger.



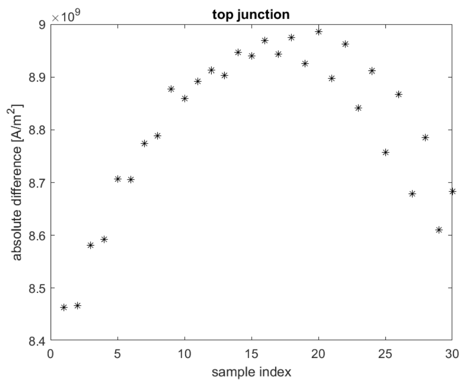
(a)



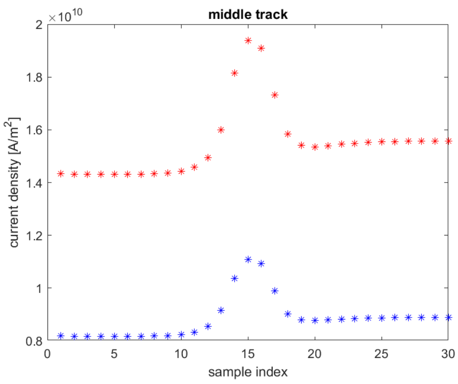
(b)



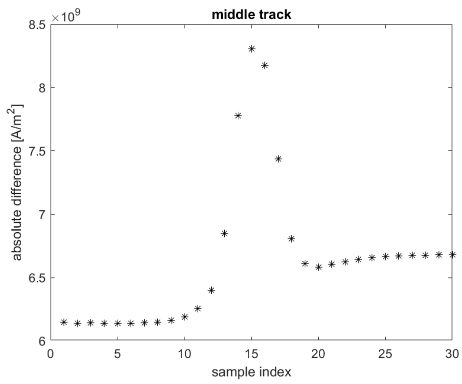
(c)



(d)



(e)



(f)

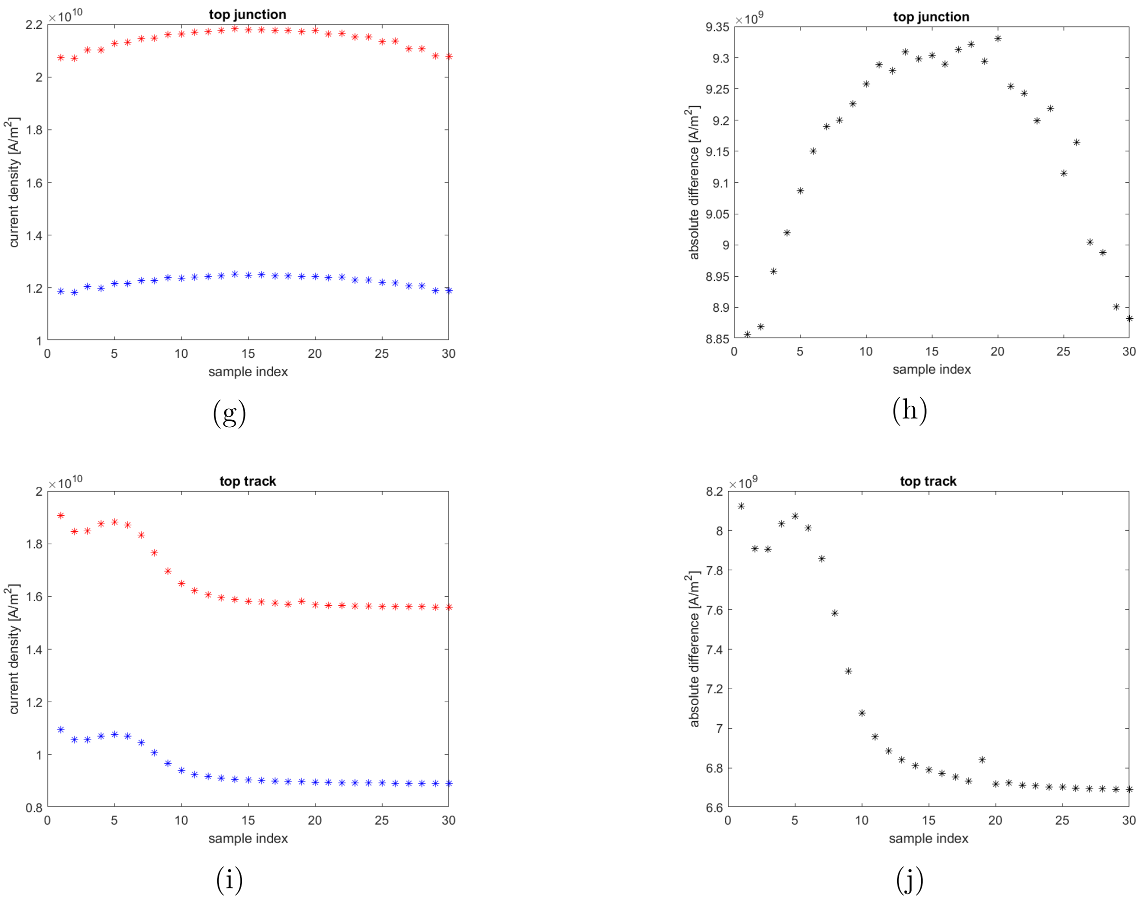
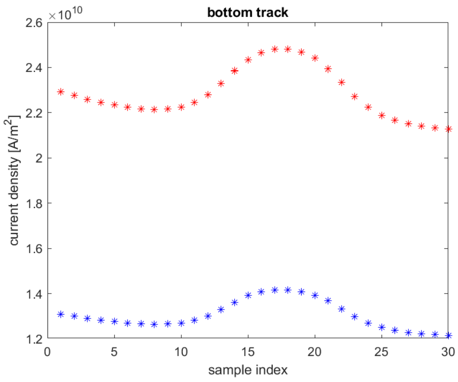


Figure 3.22. Current density samples taken in the platinum and in the cobalt layers inside the first modified version of the *INV/COPY* gate. Plots (a), (c), (e), (g) and (i) show the sampled values (the red samples come from cobalt, the blue samples from platinum), while plots (b), (d), (f), (h) and (j) show the absolute difference computed value-by value. The average absolute difference is around 8.8×10^9 A/m^2.

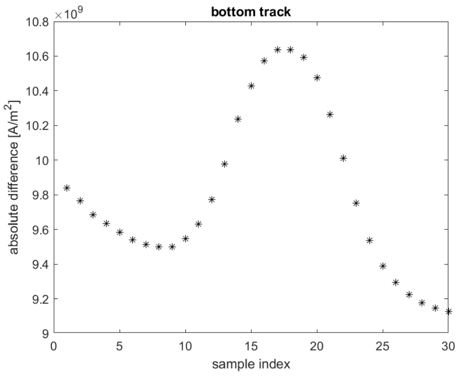
Looking at the plots showing the absolute difference, is clear that the goal has been achieved: now the largest difference between the two current densities is lower than 9.35×10^9 A/m^2. Of course, the reduced difference is due also to the reduced absolute value of both current densities: looking at the left column of plots, is easy to see that the current density inside the cobalt is still a bit less than twice the current density present inside platinum, and this of course comes from the different resistivity values. Still, if the aim is to make the two sets of samples as similar as possible, this particular version would be the solution to the problem.

3.4.1.3. Second version

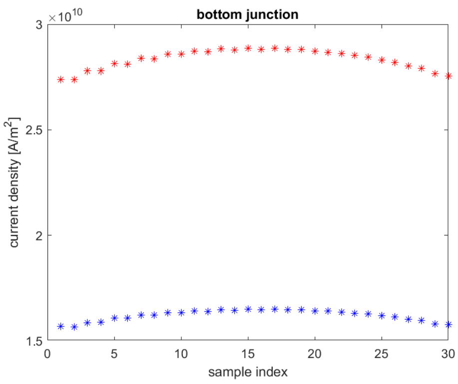
A final attempt has been made by increasing a bit more the parameter *thickness layer Pt* (up to 100 nm) and scaling by a different factor the geometry parameters along the x direction with respect to the ones along the y direction. The aim, in fact, was to make the gate wider, not only larger. For this reason, a scaling factor of 10 has been applied to the parameters *track width*, *external boundary width*, *bottom/top junction width* and *bottom/top junction height* (it was chosen to maintain the same proportions for the two junctions, since their dimensions are critical for the behaviour of the gate). A scaling factor of only 4 has been applied instead to the parameters *size contact*, *track length*, *x-coordinate junctions start*, *x-coordinate top track start* and *offset top junction*. The results of the sampling are reported in figure 3.23



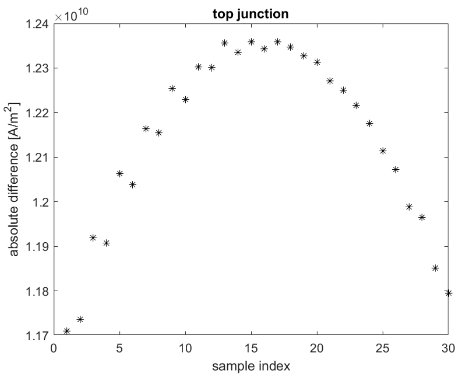
(a)



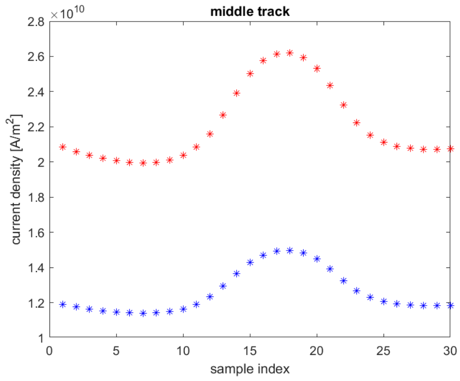
(b)



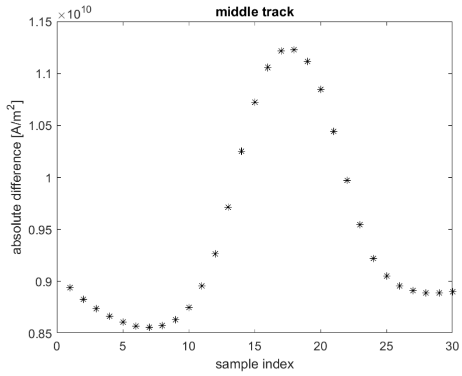
(c)



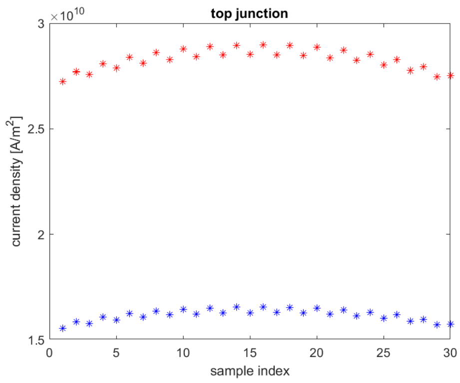
(d)



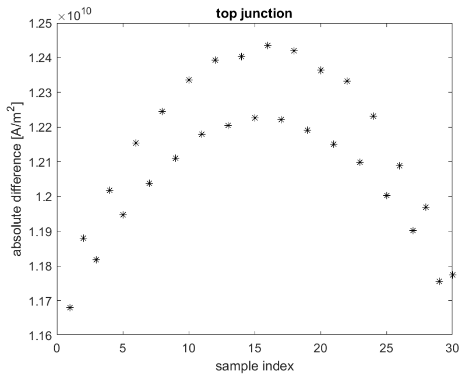
(e)



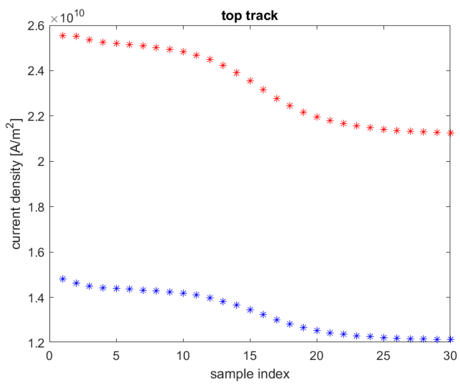
(f)



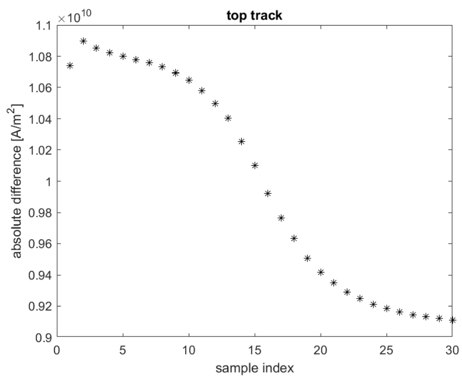
(g)



(h)



(i)



(j)

Figure 3.23. Current density samples taken in the platinum and in the cobalt layers inside the second modified version of the *INV/COPY* gate. Plots (a), (c), (e), (g) and (i) show the sampled values (the red samples come from cobalt, the blue samples from platinum), while plots (b), (d), (f), (h) and (j) show the absolute difference computed value-by value. The average absolute difference is around 1×10^{10} A/m^2.

Also the results obtained from this version are very good, even if the maximum absolute difference here is a bit higher with respect to the first modified version. Still, also this structure could be a possible and valid solution to the problem.

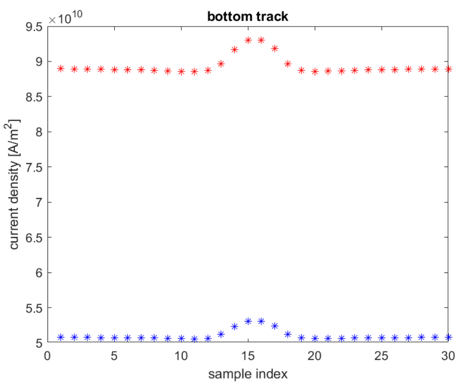
3.4.2. *AND/OR*

The same optimization has been performed also on the *AND/OR* gate, following exactly the same procedure. The same modifications to the structure have been applied as well. The reason is that, if the *INV/COPY* and the *AND/OR* gate were to be inserted inside the same system, it would be better to have the same thickness for the two layers and the same width for the tracks: the behaviour of the skyrmion, in fact, highly depends on these parameters, as it has been demonstrated previously in this chapter. So, once that a version of the *INV/COPY* gate has been accepted, it's better to make the modifications to the *AND/OR* gate as similar as possible to the ones already performed.

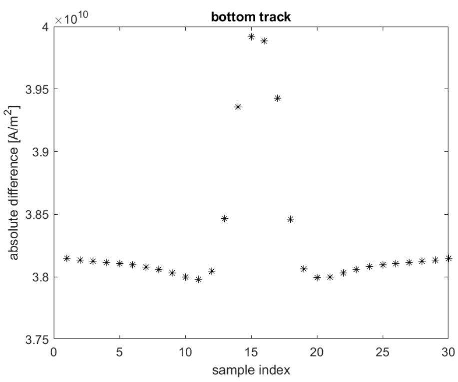
3.4.2.1. Original version

The version taken as reference for the *AND/OR* gate is *H_Structure 2*, presented in section 3.3.3. First the original version is analysed, in order to have a basis for the comparison with the modified versions.

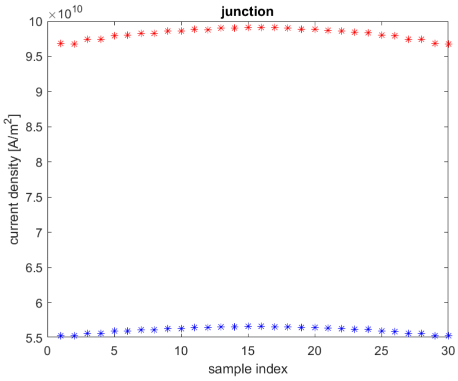
Looking at the plots reported in figure 3.24, is possible to recognize values very similar to the one already encountered in section 3.4.1.1, apart from the absolute difference values, which are a bit smaller, but not in a significant way.



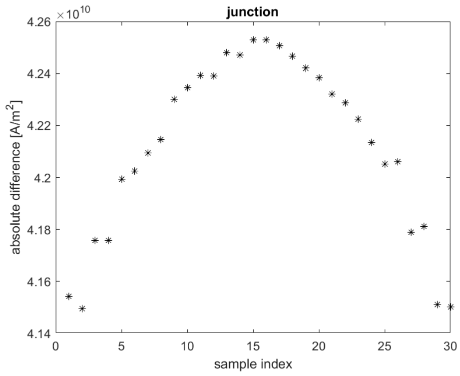
(a)



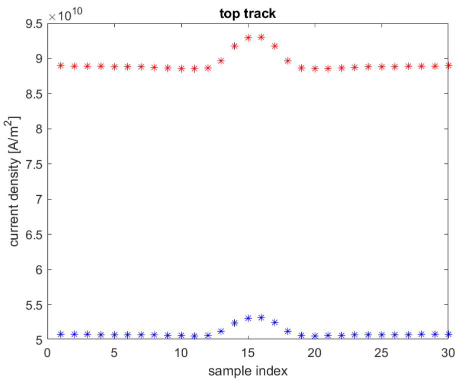
(b)



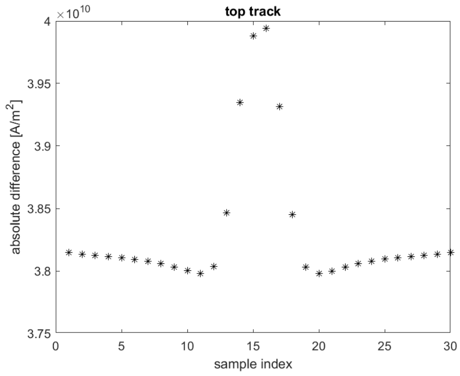
(c)



(d)



(e)



(f)

Figure 3.24. Current density samples taken in the platinum and in the cobalt layers inside original version of the *AND/OR* gate. Plots (a), (c) and (e) show the sampled values (the red samples come from cobalt, the blue samples from platinum), while plots (b), (d) and (f) show the absolute difference computed value-by value. The average absolute difference is around 4×10^{10} A/m^2.

3.4.2.2. First version

In the first modified version, the parameter *thickness layer Pt* is increased up to 80 nm and a scaling factor of 6 is applied to all the horizontal dimensions. Doing so the absolute difference values are greatly decreased, as it happened for the *INV/COPY* gate. Also in this case the mean absolute difference value is a bit smaller than for the *INV/COPY*; the maximum difference value is lower than 8.15×10^{10} A/m^2, so this structure is fully satisfying the requests and can be considered a possible solution to the problem of balancing the current density values inside the platinum and the cobalt layers.

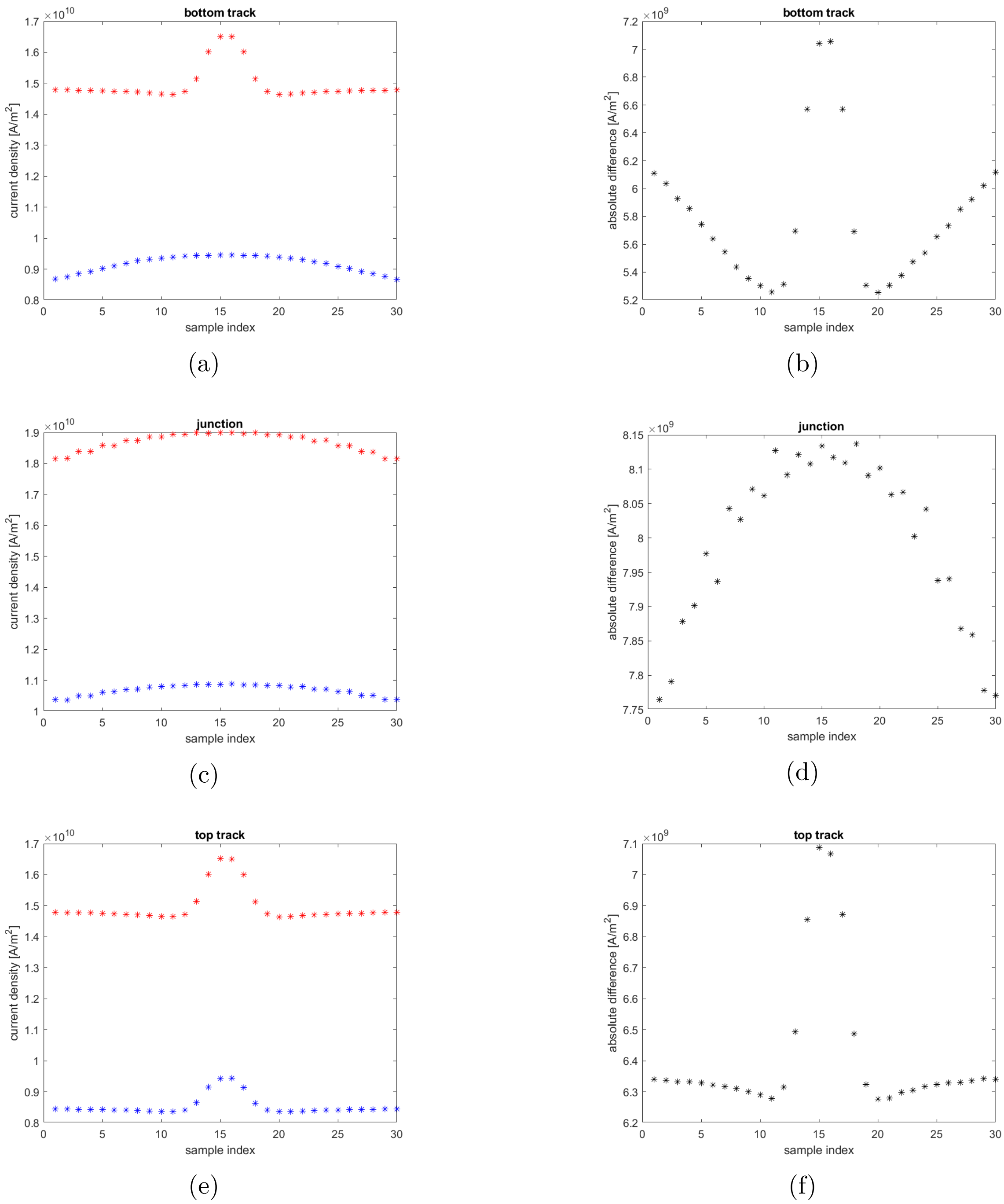
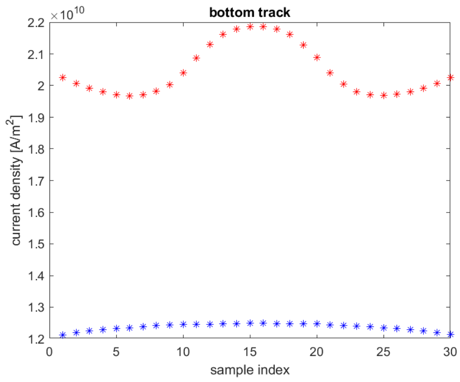


Figure 3.25. Current density samples taken in the platinum and in the cobalt layers inside the first modified version of the *AND/OR* gate. Plots (a), (c) and (e) show the sampled values (the red samples come from cobalt, the blue samples from platinum), while plots (b), (d) and (f) show the absolute difference computed value-by value. The average absolute difference is around 7.5×10^9 A/m^2.

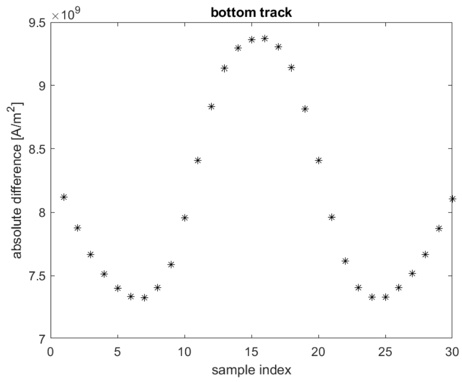
3.4.2.3. Second version

The second modification consists in increasing the parameter *thickness layer Pt* from 80 nm to 100 nm and in modifying the scaling constant between the parameters along the x and the y direction. In particular, *external boundary width*, *track width*, *hole height* and *hole width* have been increased by a factor 10 with respect to the original version, while *size contact* and *track length* have been scaled by only a factor 4.

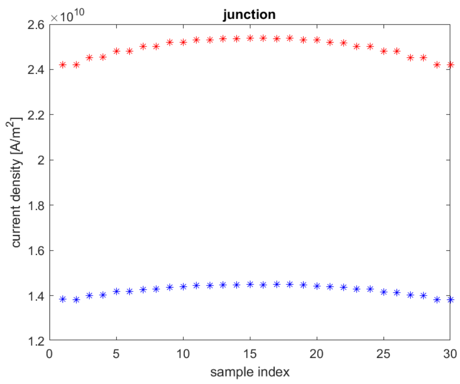
Also in this case the results are good, but not as good as the ones obtained in the previous section. The maximum difference, however, is equal to 1.09×10^{10} A/m^2, which is still good enough, if the aim is to make the current density values in the platinum as similar as possible to the values in the cobalt. So, as it happened for the *INV/COPY* gate, also this second modified version is a valid one.



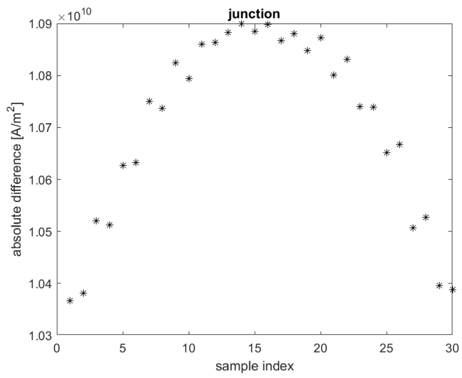
(a)



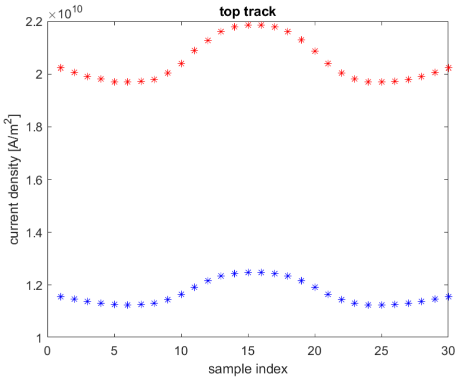
(b)



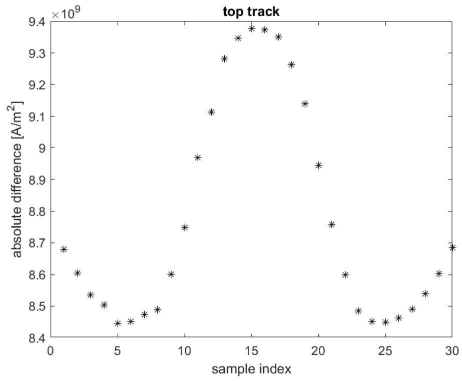
(c)



(d)



(e)



(f)

Figure 3.26. Current density samples taken in the platinum and in the cobalt layers inside the second modified version of the *AND/OR* gate. Plots (a), (c) and (e) show the sampled values (the red samples come from cobalt, the blue samples from platinum), while plots (b), (d) and (f) show the absolute difference computed value-by value. The average absolute difference is around 9.5×10^9 A/m^2.

4. Ripple carry adder

In section 2.2.2 the structure and the working principle of some basic logic gates have been described, and in chapter 3 it has been proven their correct functioning under realistic conditions. The second task of this thesis is to investigate if these logic gates can be used to build a more complex computing architecture.

In [5] it has been said that these gates can build a conservative logic system: the skyrmions that have propagated through one gate can in fact be collected at the output and used again to trigger new computations in the following gates, without the need of nucleating new skyrmions, which is an energetically expensive operation. Then, what happens if these logic gates are put one after the other? Is it really possible to reuse these skyrmions, without the need of nucleating new ones? And what are the limits of a structure like this? The *INV/COPY* gate needs one control skyrmion in input in order to perform its task: then how many skyrmions in total must be provided in input to a computing architecture, in order to make it work? Can these skyrmions be taken from the set of skyrmions that have already been nucleated?

Finally, other topics that must be addressed are: how complex is to build a computing architecture based on skyrmions? What are the main issues to be solved? And once this architecture is ready and working, what are its performances?

The aim of this chapter is to address each of these questions and to present the computing architecture that has been developed in response to them. Having already the implementation of the Full Adder structure from [5], the architecture that has been developed is a generic N-bit ripple carry adder, but in general any other kind of computing architecture could be realized by following a similar methodology.

4.1. First version

4.1.1. Half Adder - first version

The first step towards the creation of a skyrmionic ripple carry adder is to derive the structure of a Half Adder (HA). This can be done simply by starting from the structure of the Full Adder already provided in [5] and reported in figure 2.28b. Deleting all the gates not strictly needed for the computation of the functions $A \oplus B$ and $A \cdot B$, the structure that remains is shown in figure 4.1.

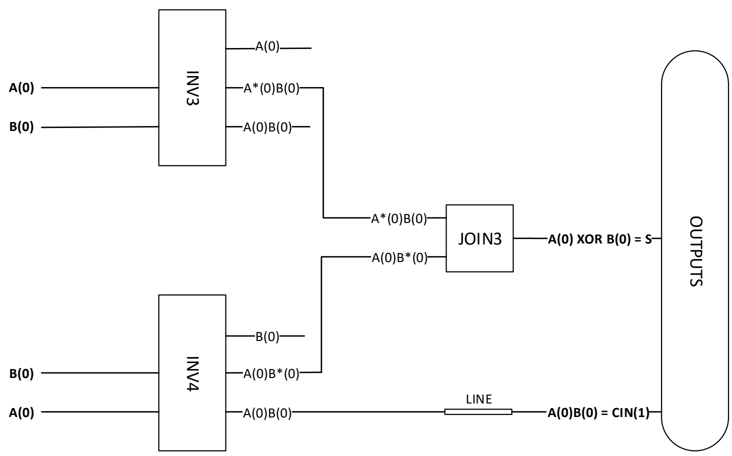


Figure 4.1. Basic scheme for the Half Adder. Each input must be provided twice, three signals are not used and only two outputs (sum and carry-out) are available. One *line* element is needed for synchronizing the two outputs.

This is a high level scheme, where the details of each logic gate are hidden inside a simple rectangle showing the name of the gate. The link between each of the high-level blocks that will be used from now and the gates presented in section 2.2.2 is shown in figure 4.2.

The structure of the HA reported in figure 4.1 is very simple: two *INV/COPY* gates are used according to table 2.2 to provide as output respectively $\overline{A(0)}B(0)$ and $A(0)\overline{B(0)}$. These two outputs are then combined by a *JOIN* gate. It must be noticed that if both inputs were equal to 1, the *JOIN* gate would concatenate the two skyrmions, thus providing two consecutive 1 on its output. This gate then is not exactly coincident with the *OR* gate shown in 2.2.2. However, here the two inputs

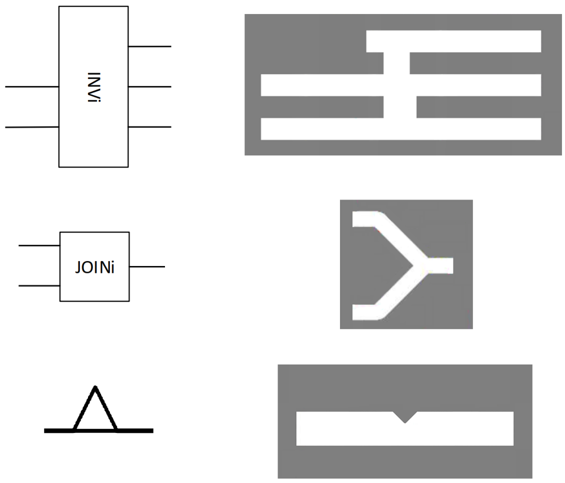


Figure 4.2. High-level representation (left) for each of the basic logic gates (right) used in the ripple carry adder. From top to bottom: inverter, join and notch (signal synchronizer) gate.

are mutually exclusive, so the *JOIN* gate can be used in place of the more complex *OR* gate to provide the function $A(0) \oplus B(0)$ on its output.

The carry bit is produced at the output of the inverter *INV4*. To synchronize this output with the sum bit it has been added a skewing line. The only function of this element is to add a delay on the propagation of the skyrmion, simulating what happens in an actual nanotrack.

This is the basic structure of a Half Adder. It is worth noticing that up to three skyrmions are wasted: *INV3* produces $A(0)$ and $A(0)B(0)$, while *INV4* produces $B(0)$, and none of them is needed for producing the outputs. So, it might be a good idea to try to collect them somewhere and maybe use them as ready-to-use skyrmions when needed, instead of nucleating new ones. The main problem is that the value of each of these signals is unknown: if our aim is to recycle the already nucleated skyrmions, we should at least know if the signal they represent is a 1, and so if a skyrmion is actually present in the nanotrack. The topic of recycling the skyrmions that are no more needed for the computations will be discussed with more detail in chapter 5.

Both $A(0)$ and $B(0)$ must be provided twice on the input of the HA; not only: also the Full Adder (FA) shown in figure 2.28b needs each input to be doubled. The aim of this architectural analysis is trying to reduce the number of skyrmions to be nucleated and at the same time to make each structure as symmetrical as possible, in order to allow a higher modularity and to make easier the extension of the adder to a generic N-bit adder: for this reason, the HA already shown has been modified into the structure shown in figure 4.12. In this new scheme some gates have been added, while the ones already present in figure 4.1 have maintained their label and their function.

INV1 and *INV2* have the responsibility of duplicating respectively $A(0)$ and $B(0)$. Both of these two gates need an input fixed to 1 in order to work as *INV/COPY* gate, and these two skyrmions are an overhead that must be paid in order to allow the architecture to correctly work. All the inputs are synchronized by means of one notch each: these skyrmions in fact are nucleated externally with respect to the HA, which must be able to work even if each input takes a different amount of time before being available for the propagation in the circuit.

Also the FA will need a copy of each of its inputs, that are $A(i)$, $B(i)$ and $COUT(i-1)$. Concerning $A(i)$ and $B(i)$, their value is unknown and the only possibility for reducing the number of skyrmions to be produced is to nucleate each of them just once, and then duplicate them with one *INV/COPY* gate each. As mentioned, the *INV/COPY* gate needs one input (*CTRL*) to be fixed to 1, so in principle also the FA will need at least two additional skyrmions (three if we consider also the input $COUT(i-1)$). The most interesting point, however, is that each *INV/COPY* gate provides also the complemented version of its unknown input: in the case of the HA, *INV1* and *INV2* provide not only two copies of $A(0)$ and $B(0)$, but also $\overline{A(0)}$ and $\overline{B(0)}$. Now, it must be remembered that the basic version of the HA (shown in figure 4.1) had three signals that were not exploited: two of them were $A(0)$ and $B(0)$. So, using these unused skyrmions for computing $A(0)+\overline{A(0)}$ and $B(0)+\overline{B(0)}$, is possible to have two signals that are for sure equal to 1: these signals can then be provided towards the next FA in order to allow the duplication of its inputs $A(i)$ and $B(i)$. The computation of $A(0)+\overline{A(0)}$ and $B(0)+\overline{B(0)}$ is done respectively by the gates *JOIN1* and *JOIN2*, which provide two outputs equal to 1. Also in this case the two inputs are mutually exclusive, so there is no way to get two consecutive skyrmions on the output (which would produce an error).

The FA at the next stage will need also the input $COUT(i-1)$ to be doubled.

Also in this case there is no problem: the third input that was not exploited in the basic version of the HA was exactly equal to $A(0)B(0)$, which is the value needed by the FA. So, is enough to propagate this value towards the output of the HA and to provide it in input to the FA.

Finally, to understand the placement of the *line* elements along the circuit, it could be useful to look at figure 4.13. In this scheme, the red numbers indicate the depth of the logic level for each element. The inputs are labelled with a 0; *INV1* and *INV2*, which receive them, are at level 1, and their outputs are at level 1 as well; *INV3* and *INV4* are at level 2... and so on. The *line* elements are inserted either to force the synchronization of all the outputs from the HA, or in all the cases in which a gate receives two inputs belonging to different logic levels. For example, *JOIN2* receives $\overline{B(0)}$ at level 1 and $B(0)$ at level 2: *INV4* will take a certain time to produce the output $B(0)$, so the signal $\overline{B(0)}$ needs to be delayed by an equal amount of time, in order to be synchronized with the other input of gate *JOIN2*. The assumption here is that the delay introduced by each *line* element is exactly coincident with the amount of time required by each gate for the elaboration. However, this is not a heavy hypothesis: from an applicative point of view this could be guaranteed simply by tuning the length of each nanotrack.

In figure 4.12 three crosses appear: these structures are needed in the schematic to signify the crossing of two nanotracks. This topic will be discussed more in detail in section 4.1.2. However, the important point to underline here is that in this work it has been assumed that these crosses do not introduce any delay on the signal propagation. If it was required to leave this hypothesis, a new design of the HA should be made, because a new logic level should be assigned to these crosses too.

So, the structure obtained for the HA allows for sure the computation of a two-bit sum with an overhead of two skyrmions: there are no skyrmions wasted and the FA after it (*FA(1)*) doesn't need any input skyrmion apart the ones encoding the value of $A(1)$ and $B(1)$.

4.1.2. Cross

In a traditional PCB (printed circuit board) the crossing of two lines would be solved simply by introducing two vias and moving a small piece of one of the two traces into another layer. With skyrmions this topic is a bit more difficult to solve, and to date no solution has appeared yet in literature. The major problem, with skyrmions, is in the need of maintaining the contact between the HM and the FM layers, because otherwise the DM interaction, responsible for stabilizing the skyrmion and actively involved in its nucleation mechanism, would vanish. It becomes quite difficult, then, to think how a structure equivalent to the usual via used in PCBs would be realized without renouncing to the presence of the DMI. For sure it becomes much easier to find a solution to the problem that remains inside a 2D plane, instead of a structure which exploits a 3D space.

This said, probably the most promising possibility would be to exploit the voltage controlled PMA, already described in section 2.1.2.1 and exploited in the application described in section 2.2.3. In this way, by properly patterning some nanocontacts, needed for applying a voltage control, in correspondence of the input to each path, it would be possible to create a potential barrier that doesn't allow the skyrmions to enter the path they should avoid. Doing so, it would be possible to actively guide each particle along the path it must follow. The problem of course comes from the need of actively controlling at least one voltage signal per each cross structure.

Another possibility, that however has still to be verified with micromagnetic simulations, could be to adopt the structure shown in figure 4.3. Like mentioned in section 2.2.2, each skyrmion driven via SHE by a current flowing in the HM layer below the nanotrack is subject to two force components: the former drives the skyrmion along the nanotrack ($+y$ direction in figure 2.26), while the latter is the Magnus force that moves the skyrmion towards left with respect to the direction of the current flow ($-x$ direction in figure 2.26). As a consequence, like already discussed, each skyrmion will follow its nanotrack as long as it doesn't have any other choice, but it will move towards left as soon as it has the possibility to do so.

In 4.3a the skyrmion coming from input A arrives at the central junction and, due to the skyrmion Hall effect, is driven towards the rightmost track for output A'. The skyrmion coming from input B, in figure 4.3b, is again driven towards the B' output due to the skyrmion Hall effect. Finally, when both inputs are equal to 1, like in figure 4.3c, the two skyrmions will be subject to a repulsive skyrmion-skyrmion interaction that will drive skyrmion A towards input B and skyrmion B towards the leftmost nanotrack for output A'. So, the two skyrmions are exchanged. However,

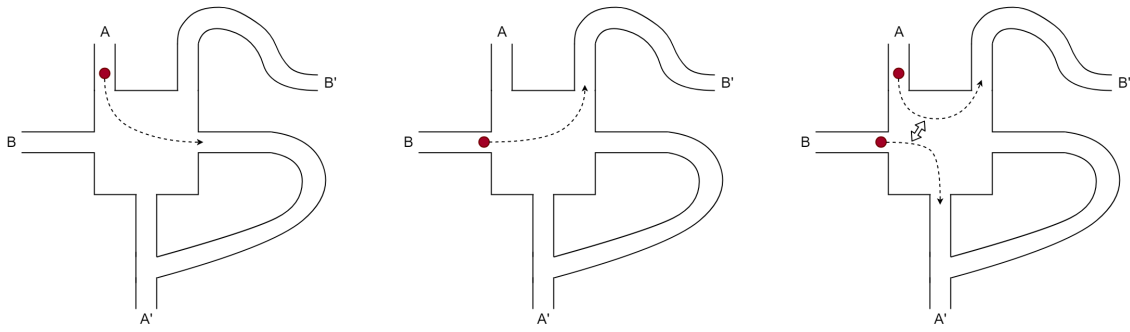


Figure 4.3. Basic scheme for a possible solution to the crossing of two nanotracks. When only one skyrmion is present, it is driven towards the correct output by the skyrmion Hall effect. When two skyrmions are present their skyrmion-skyrmion repulsion is exploited. This structure is only a suggestion, and it could be necessary to tune some parameters in order to make it work.

this is not a problem: what is essential is that $if(A = 1\ and\ B = 1) \implies A' = 1\ and\ B' = 1$.

Since this structure has not yet been verified with the help of micromagnetic simulations, it is not clear what could happen in terms of current in the central junction. Moreover, it could happen that with this exact geometry some skyrmions may not be able to propagate until the output, so it could be necessary to tune some parameters like the angles, the exact dimension and position of each nanotrack, besides their shape and their curves. However, the key principle behind the structure should remain approximately the same.

In any case, it should be possible to allow two tracks to cross each other without the need of any multilayer structure, like the one used for traditional vias: as a result, it can be assumed that two tracks can cross each other and that a solution to the problem should be available.

4.1.3. Full Adder - first version

The structure of the FA has already been shown in 2.28b. However, that version needs each input to be provided twice and is not able to generate any logic 1 that could be provided by *FA(i)* to *FA(i+1)* to allow the duplication of its inputs. So, that structure has been modified like shown in figure 4.14.

Focusing on *FA(1)*, *INV1* and *INV2* receive the two signals equal to 1 generated by the HA together with the inputs $A(1)$ and $B(1)$. In this way, the duplication of the input signals is done without any additional cost, like already discussed in section 4.1.1; the input *CIN(1)*, moreover, is provided twice directly by the HA itself. Concerning the outputs from the FA, *JOIN1* and *JOIN3* are dedicated to the production of two signals equal to 1, so that they can be provided to the following FA in the chain and again allow the duplication of its inputs. Also in this case the duplication of the output carry can be obtained without any additional cost: it is enough to use the bottom output from *INV6* and combine it in *OR* with the bottom output of *INV3* by using the gate *JOIN5* (also in this case the two inputs to *JOIN5*, $[A(i) \oplus B(i)]CIN(i)$ and $A(i)B(i)$, are mutually exclusive); both these signals were not exploited in the starting version of the FA shown in 2.28b. This is a very good result: it means that is possible to build a generic N-bit ripple carry adder by concatenating self-sustaining full adder structures. The only cost that must be paid is of two additional skyrmions to be provided on the input of the half adder at the head of the chain.

It is worth noticing that in the case of the FA one signal remains not exploited: it is the top output of *INV6*, equal to $CIN(i)$. It won't be used in this design of the adder, but being equal to the carry input to each FA it could be exploited to perform some more advanced elaborations.

Here both lines and notches have been used for synchronizing the data propagation. The technique used for deciding where to introduce them was the same as the one adopted in 4.1.1, and the logic depth of each element is underlined by the red numbers that appear in figure 4.14. The aim here was to make the HA the critical path of the whole circuit: for this reason a row of notches has been inserted after three levels of logic, composed by the cascade of two inverters and one *join* gate. In all the cases in which the need was to synchronize either the outputs from the FA or two inputs to the same gate, some lines have been used instead.

4.1.4. Adder - first version

Once the structure of both the HA and the FAs has been defined, is enough to concatenate them to build the structure of the ripple carry adder. Some more crosses are needed at the interface between the different elements, like shown in figure 4.4.

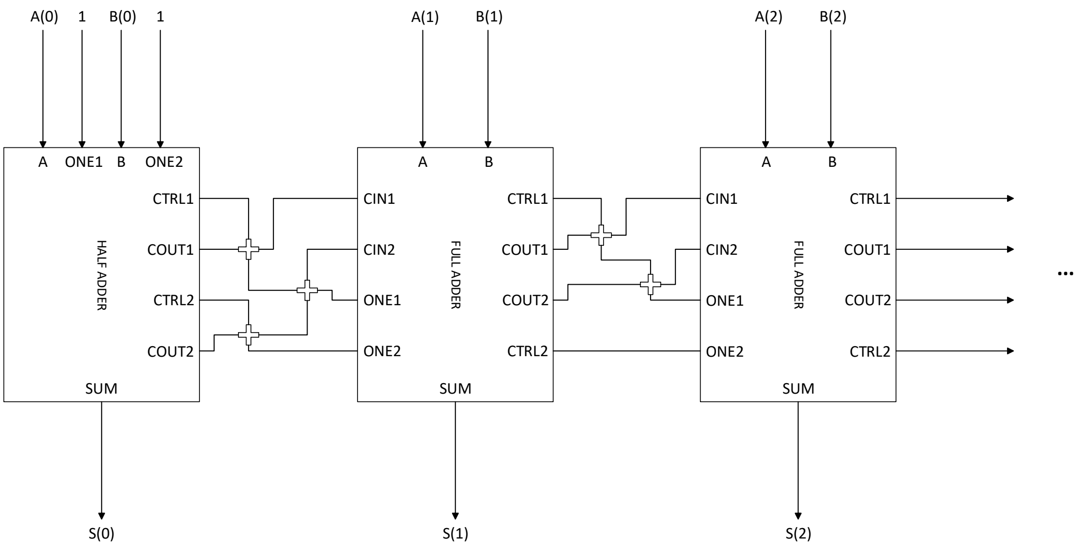


Figure 4.4. Sketch of the interfaces among a HA and two consecutive FAs, showing all the *cross* elements needed for connecting each output with the corresponding input.

In skyrmionic devices, like already mentioned, the synchronization in the propagation of the different skyrmions is of vital importance. In a ripple carry adder each element needs the output carry from the previous stage, in order to obtain the correct result: this means that each FA has to wait the propagation of the output-carry skyrmion from the previous stage, before receiving its inputs and going on with the computation. The key difference with the traditional combinational electric circuits is that the signals do not maintain their value until it is no more needed, but they are transformed into a particle that propagates along the circuit: if the timing of this particle is wrong, then the result of the computation will be wrong as well.

For this reason a skewing structure, shown in figure 4.5, is needed at the input of the adder. Each input bit $A(i)$, $B(i)$ is delayed with respect to the bits $A(i-1)$, $B(i-1)$ by an amount of clock cycles equal to the depth of the pipeline of the previous stage. For example, the HA has only one pipeline stage on its inputs (notches from 01 to 04 in figure 4.12): then the skewing structure at the input of *FA(1)* is made by only one notch; *FA(1)* has two rows of notches inside its structure, so the skewing network before *FA(2)* will be made by $1+2=3$ notches... and so on. The law that gives the number of notches at the input of each element is $Notch(i) = 1+2(i-1)$, where $1 \leq i \leq N-1$ is the index of the bit, N is the adder's parallelism, and

$Notch(0) = 0$.

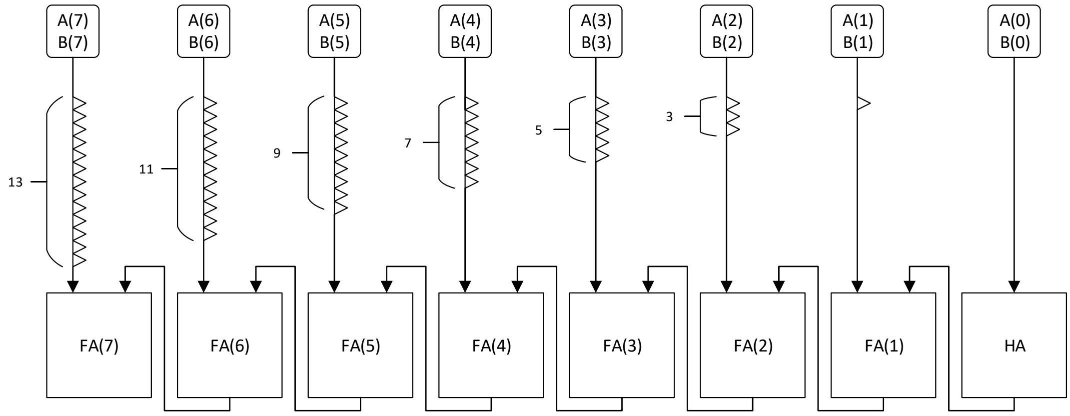


Figure 4.5. Skewing network needed at the input of the first version of the ripple carry adder in order to synchronize the input data together with the output-carry chain.

4.1.5. VHDL description

The behaviour of the adder has been simulated with *ModelSim*. To do so, a VHDL description of its components is needed. This description must be a bridge between what happens in an actual physical device and the needs of simplification and abstraction of the electronic designs: that is, it must describe the physical movement of the skyrmions inside the nanotrack, while providing at the same time a simplified interface towards the higher abstraction level, allowing to ignore all the physical details of each gate.

The VHDL model of the elements *INV/COPY*, *join*, *line* and *notch* was already provided by a previous work. Here this model will be described in broad terms, in order to give an idea about the main approximations and simplifications done in simulating the movement of the skyrmions inside each structure. The values chosen for each physical parameter used in the code are reported in table 4.1: some of them have been extracted from literature and should be reasonable, while others come directly from the micromagnetic simulations already discussed. Of course, these are mean and approximate values, especially the speed values, since all the simulations considered in chapter 3 use a variable current-density distribution.

The VHDL code for each of these gates can be found in appendix B.

Table 4.1. Values chosen for the physical constants

CONSTANT NAME	VALUE
Horizontal speed	150 m/s
Vertical speed	40 m/s
Depinning current density	1.24×10^{10} A/m^2
Notch depinning current density	2×10^{11} A/m^2
Horizontal speed with notch depinning current density	484 m/s
Minimum skyrmion-skyrmion distance	22 nm
Skyrmion diameter	18 nm
Low current density value	5×10^{10} A/m^2
High current density value	2×10^{11} A/m^2

4.1.5.1. Not gate

A sketch showing the structure of the *INV/COPY* gate, together with the coordinates (measured in nanometres) of the most relevant points for understanding its model, is shown in figure 4.6. These coordinates are the ones describing *NOT_-Structure 3*, which was presented in section 3.2.3.

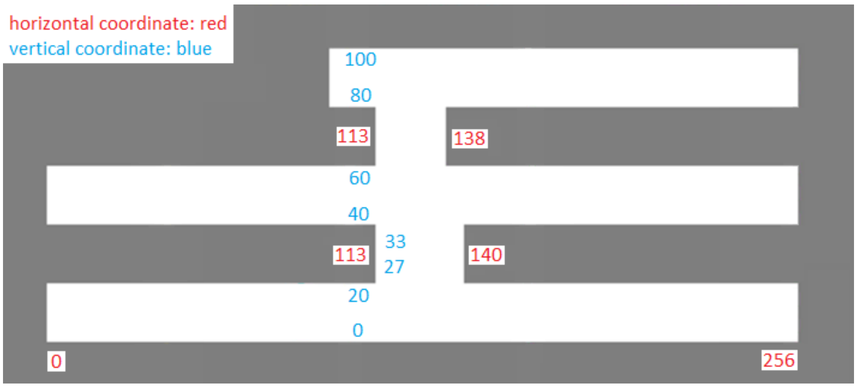


Figure 4.6. Coordinates of the most relevant points inside the structure of the *INV/COPY* gate. Each value is measured in nanometres. The red coordinates are referred to an horizontal axis, the blue coordinates to a vertical axis.

Every time that a skyrmion is detected on one of the two inputs, a variable counting the number of skyrmions present inside the gate is incremented and the corresponding coordinates are inserted inside an array: if the skyrmion was detected on the bottom track it is assigned the coordinates (0.0, 10.0), whereas if it was detected on the middle track it is assigned the coordinates (0.0, 50.0). Each simulation step lasts 10 ps: this means that every 10 ps the coordinates of each skyrmion inside the gate are updated. This is done by a function which receives the old coordinates, the value of the current, the elapsed time and the index of the skyrmion. The new coordinates are computed only if the value of the current applied is higher than the value of the depinning current. The x coordinate is computed by multiplying the velocity along x by the elapsed time and adding the old horizontal coordinate, while the y coordinate is found by adding the old vertical coordinate to the velocity along y multiplied by the elapsed time.

The value of the x and of the y velocity are chosen according to the old position of the skyrmion:

- If its coordinates (x,y) were $113 < x < 140$ and $y < 20$, it may be about to change track. This, however, can happen only if there are no other skyrmions in position $113 < x < 140$ and $40 < y < 60$, that is, if there are no skyrmions that are already occupying the middle nanotrack nearby the junction region: if this is the case, then, the skyrmion is allowed to change track and its velocity is purely vertical, otherwise the velocity will be purely horizontal.
- If the skyrmion is in the middle track, so $40 < y < 60$, and it has $113 < x < 138$, it will for sure change track since no repulsive effect is possible there, and again its velocity is purely vertical.
- If the skyrmion is somewhere in the top junction, again its velocity will be purely vertical.
- If it is inside the bottom junction with $33 < y < 50$, it means that it is performing a curve towards the middle track and it has both a horizontal and a vertical velocity component. If instead it is inside the bottom junction with $20 < y < 33$, the velocity is purely vertical.
- In all the other cases, the velocity is purely horizontal.

A skyrmion is emitted every time that the x coordinate computed is larger than 256: then, according to the value of the y coordinate, it is decided whether the top, the middle or the bottom output has to be set to 1 for 1 ns, which is the width of each pulse identifying a skyrmion.

From this overview it's clear that the complex phenomena involved in the interaction between the two skyrmions inside the junction area are not taken into account. Actually, all these phenomena, which have been partially detailed in chapter 3, are often too complex even to be forecasted, and only a micromagnetic simulation can reveal, according to the structural parameters of the gate, what is likely to happen inside that region.

4.1.5.2. Line element

The model for the *line* element is very simple with respect to the *INV/COPY* gate. The simulation step is again of 10 ps. Every time a skyrmion is detected at the input of the line it is assigned the coordinates (0.0, 0.0); these coordinates are updated 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. So, in this case any transverse movement of the skyrmion inside the nanotrack is completely ignored (actually, the nanotrack width is not even considered as a parameter). Again, a skyrmion is emitted if the computed x coordinate is larger than the track length. It could happen that two or more skyrmions may verify this condition at the same time: if this is the case, the simulation step must be reduced. The width of the pulse at the output, which signifies the presence of a skyrmion, is equal to 10 ps.

4.1.5.3. Join gate

The *join* gate is quite different from a simple *line* element, but its VHDL description is instead quite the same. Here the two inputs 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. This means that the variable counting the number of skyrmions inside the gate can be incremented in two different conditions, but the coordinates assigned to the new detected skyrmions are always equal to (0.0, 0.0). Moreover, a possible skyrmion-skyrmion collision at the junction is not taken into account. From this point on, the description of the gate is exactly the same as for the *line*: the simulation step is of 10 ps, the y coordinate is always considered equal to 0 (again the track width is not taken into account), and the x coordinate is computed in the same way. Also in this case a skyrmion is emitted when its x coordinate becomes larger than the track length, and it may happen that two or more skyrmions verify this condition simultaneously: again, the solution to this problem is to reduce the simulation step. Also here the output pulse width is equal to 10 ps.

4.1.5.4. Notch

Here two depinning current values must be taken into account: the first one is the usual depinning current value, necessary to put the skyrmion into movement; the second value is the current needed for the skyrmion to go through the notch. These two current values correspond to two different values for the horizontal velocity (also in this case the track width and the transversal movements are not taken into account, since the y coordinate is always maintained fixed to 0.0).

The behaviour of the skyrmion then depends on the value of the current applied: if it is below the traditional depinning current no movement is allowed at all, as it happened in all the previous gates.

If the current applied is equal or higher than the current value needed for making the skyrmion go through the notch, the signal synchronizer becomes a simple delay line and its model is exactly the same used for the *line* element: if this is the case, in order to compute the new x position, the highest velocity value must be used.

If the current applied is intermediate between the two depinning current values, a more complex behaviour is described. First of all, is necessary to understand if the skyrmion whose coordinates are being updated (the simulation step is again of 10 ps) has already crossed the notch: if this is the case, is enough to compute its new x coordinate like always (this time using the lowest velocity value). If instead it still has to go through the notch (and so the distance $old\_xCoordinate - notch\_xCoordinate \leq 0$), first is necessary to understand if it is the only skyrmion that is moving before the notch, or if there are any others. The number of skyrmions that are present between the skyrmion currently considered and the notch is stored in the variable *blocking_skyrmions*, which can also be equal to 0. The minimum distance from the notch that is allowed for the considered skyrmion, as a function of the number of blocking skyrmions, is computed like the diameter of a skyrmion plus the minimum skyrmion-skyrmion distance, multiplied by *blocking_skyrmions*:

$$minimum\_distance = blocking\_skyrmions \cdot (SKYRMION\_DIAMETER+ \\ +SK\_SK\_MIN\_DISTANCE) \tag{4.1}$$

If the skyrmion's distance from the notch ($notch\_distance = notch\_xCoordinate - old\_xCoordinate$) is larger than $minimum\_distance$ by at least $\Delta\_distance = horizontalVelocity \cdot elapsedTime$ (so if $notch\_distance - \Delta\_distance > minimum\_distance$), then the new x coordinate can again be computed like always, adding to the old x coordinate the value of $\Delta\_distance$. On the contrary, if the skyrmion is not allowed to complete its movement due to the other $blocking\_skyrmions$ skyrmions packed right before the notch (and so if $notch\_distance - \Delta\_distance \leq minimum\_distance$), then the skyrmion must be placed right before the last of the packed skyrmions, that is, the new x coordinate must be equal to $notch\_xCoordinate - minimum\_distance$. It is worth noticing that, if $blocking\_skyrmions = 0$, the new x coordinate of the skyrmion will be coincident with the notch position: this means that the notch is simplified as a point-like structure.

The distances taken into account in this section are represented in figure 4.7

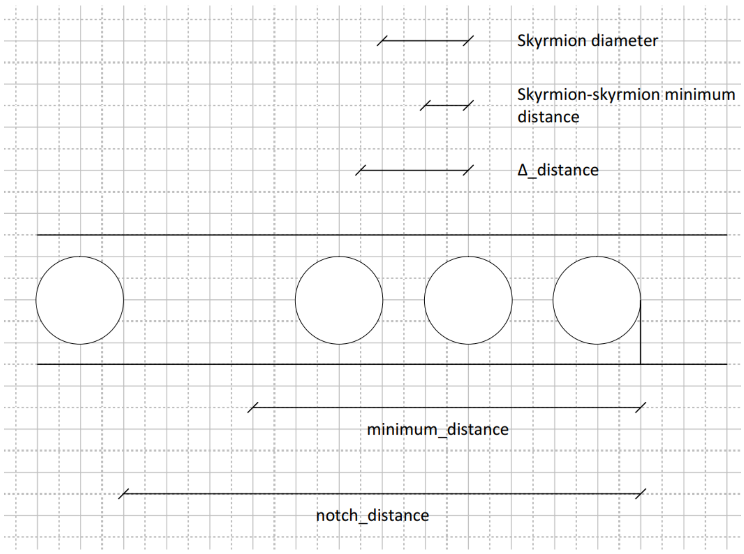


Figure 4.7. Sketch representing the definition of the distances used for describing the behaviour of the *notch* element.

As usual, a skyrmion is emitted whenever the x coordinate becomes larger than the track length; if this condition is verified by more than one skyrmion at once the simulation step must be reduced. The width of the output pulse is of 10 ps.

4.1.5.5. Cross

The model used for the *cross* elements is very basic and high-level: it consists in two simple assignments $A' = A$ and $B' = B$, where A and B are the two input signals and A' and B' are the corresponding output signals. In this way is has been possible to include inside the structure of the adder an element which behaves like the cross described so far. When a more stable and verified structure will be available, its VHDL description can be used to substitute this very basic model without changing anything else, as long as the interface (the *port map*) remains the same.

4.1.6. Simulation of adder

4.1.6.1. Tuning of the delay elements

As already stated several times, the timing of the skyrmions inside the structures considered in this thesis is of vital importance: if one skyrmion is just a bit faster than another, the whole behaviour of the resulting adder may be compromised. In this particular version of the adder, the main feature is the use of the line elements, that are used as delay lines needed for the skyrmion synchronization. By looking at figures 4.12 and 4.14 it can be noticed that the delay lines work in parallel either to a NOT gate, or to a *join* gate. For sure the NOT gate will be slower with respect to the *join* gate: for this reason, the length of both the *line* element and of the *join* element must be tuned in order to make them introduce a delay on the skyrmion propagation as similar as possible to the one introduced by the NOT gate.

The delay introduced by the NOT gate changes according to the input: if one skyrmion is injected in the bottom input, it will come out from the middle output and the resulting delay is of 2.346 ns; if a single skyrmion enters the top input, it comes out from the top output and the delay is of 2.449 ns; finally, if both skyrmions are provided, they will come out from the top and from the bottom output, with a delay of about 2.974 ns (measured from the rising edge of the inputs to half width of the high-level on the slowest output pulse). A good guess for the delay that the *join* and the *line* elements must introduce could be of 2.5 ns: knowing that the horizontal speed is of 150 m/s, this means that both elements must have a length of 375 nm. Using these values, the behaviour of both the HA and of the FA has been verified correct with all the possible input combinations.

4.1.6.2. Simulation results

The critical path for this version of the adder is given by the cascade of two inverters and one *join* gate inside the HA structure. This is why is enough to simulate the HA and to observe the obtained waveform in order to decide the period needed for the clock cycle.

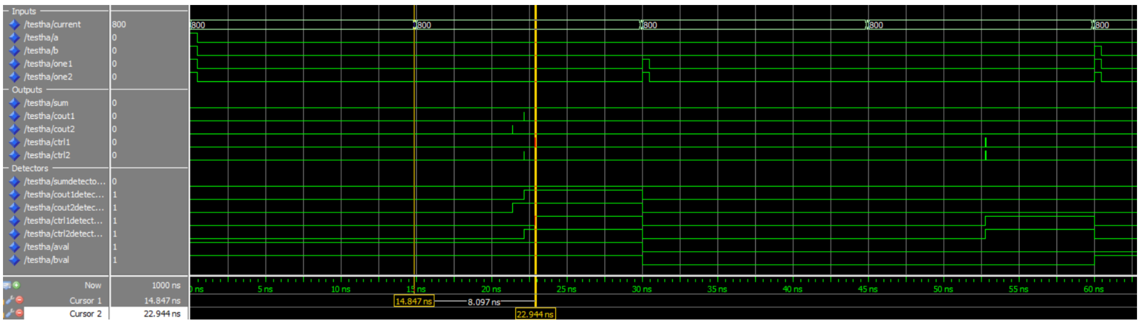


Figure 4.8. ModelSim snapshot showing the simulation of the first HA version. The delay highlighted between the two cursors is the time needed for the propagation through two inverters and one *join* gate.

From figure 4.8 it can be noticed that, once the input skyrmions have overcome the notches 01-04, they need a bit more than 8 ns in order to reach the output (this value of course depends on the value of the physical constants chosen for the simulation). Then a reasonable value for the clock cycle could be of 10 ns, where the high current density needed for the signal synchronization is applied for 150 ps. This leads to an operating frequency of 100 MHz.

Using this value for the clock period, both the HA and the FA structure have been verified again for all the possible combinations of the inputs. A 16-bits wide ripple carry adder has been verified in the conditions 0 + 0, 65535 + 65535, 0 + 65535 and with 25 couples of random inputs. The latency is about 318 ns long.

4.2. Increase of performance

The version of the N-bit adder presented until now has two main problems: first of all, the use of the *line* elements. Due to their presence, the design and the verification of both HA and FA must proceed by trials and errors: every time that a physical constant, that could be even the size of a single inverter, is changed, also the delay of the inverters and so the length of the lines required for the data synchronization is likely to change. Moreover, the first version of both HA and FA can be made faster by reducing the critical path and making it equal to the delay of just a single inverter. To do so, is enough to eliminate all the *line* elements and add in their place some rows of notches.

4.2.1. Half Adder - second version

The new version of the HA is presented in figure 4.15. While the basic structure has remained the same, all the *lines* have disappeared and some rows of notches have appeared to substitute them. To verify the correctness of the placement of the notches, as usual, some red numbers showing the logic depth of each element have been inserted in the schematic. The notches from 11 to 16 are not necessary for the data synchronization, but are needed for breaking the critical path; the notches from 21 to 28 are needed also for assuring the data synchronization.

4.2.2. Full Adder - second version

Also the basic structure of the FA has remained the same, like 4.16 shows. Here the notches 51-58 are only needed for breaking the critical path, while all the other notches are needed also for the data synchronization.

It is worth noticing that here the length of the *join* elements is not critical. There is only one *join* element which works in parallel with a *NOT* gate, and it is the *join 4* shown in figure 4.16; however, its output arrives directly on a row of notches, which assure the data synchronization, whatever the length of the line. In all the other cases, more *join* elements work in parallel occupying an entire clock cycle: if their length is reduced, their outputs will arrive sooner at the row of notches placed on their outputs, without any other consequence. For this reason, in the following simulations the length of the *join* elements' line has been reduced down to 256 nm, to be sure that the critical path is associated to the NOT gate.

4.2.3. Adder - second version

The only difference in the structure of the adder itself is in the complexity of the skewing network, which here is slightly increased. The HA now has three levels of pipeline, while each FA introduces six levels. So, the law describing the number of notches required for skewing each bit $1 \leq i \leq N-1$ is $Notch(i) = 3+6(i-1)$, with $Notch(0) = 0$. The new structure is shown in figure 4.9.

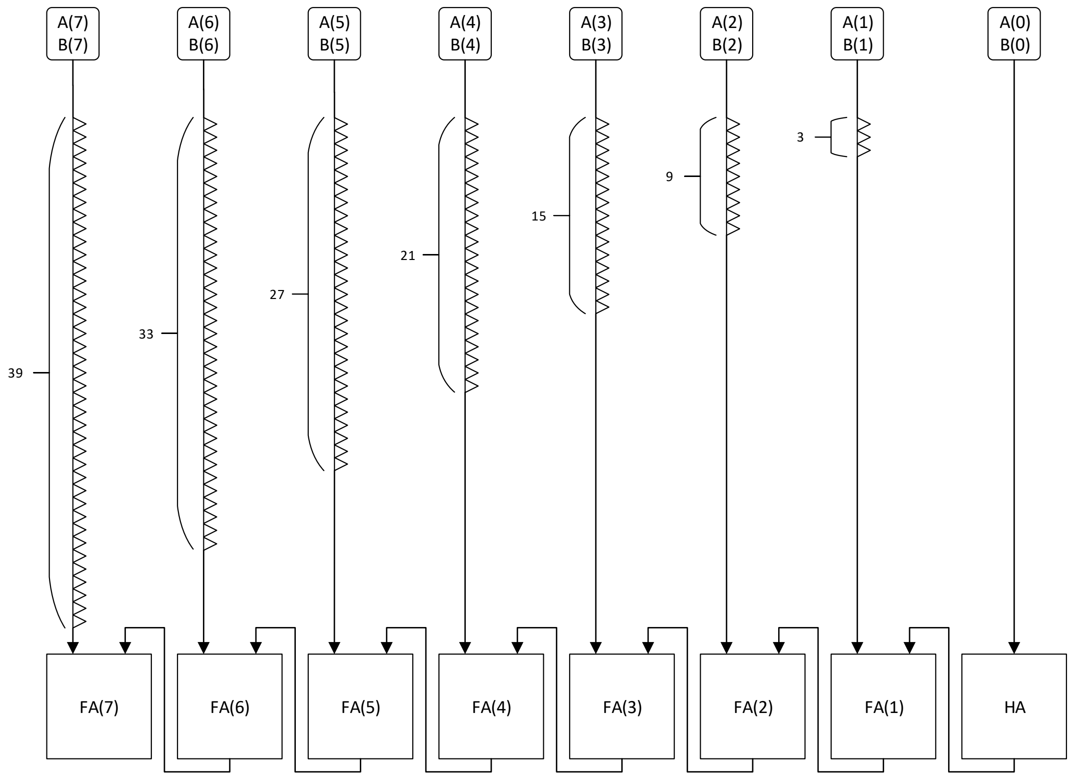


Figure 4.9. Skewing network needed at the input of the second version of the ripple carry adder.

4.2.4. Simulation of adder v2

This time the critical path inside the entire structure of the generic N-bit adder is defined by the delay of a single inverter, or of a single join gate, according to which is the largest between the two. With a length of 256 nm for the *join* elements, as mentioned, the *NOT* gate should be the slowest between the two. In order to decide the length of the clock period, is enough to simulate the HA and concentrate the attention on the behaviour, let's say, of *INV1* and of *JOIN1* (reference in figure 4.15).

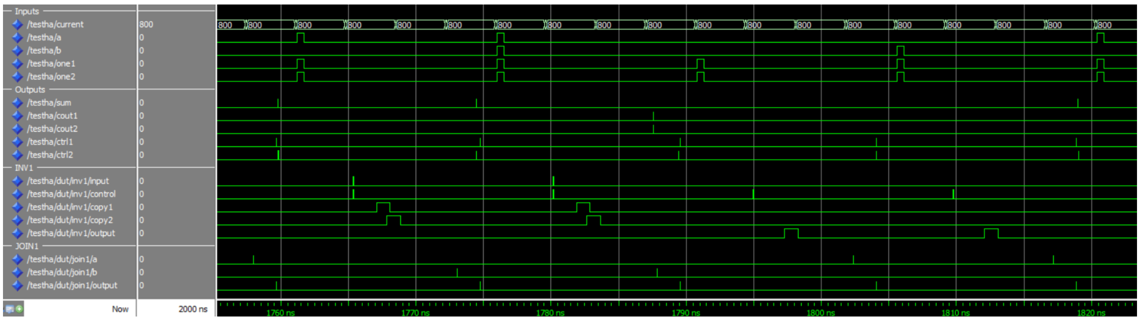


Figure 4.10. ModelSim snapshot showing the simulation of the second version of the HA. Also the input and output waveforms of the *INV1* and *JOIN1* gates appear. The delay of *INV1* is larger than the delay of *JOIN1*, and so the inverters represent the critical path of the second version of the ripple carry adder.

The simulation snapshot reported in figure 4.10 shows that *INV1* has a delay of about 2.99 ns, while *JOIN1* of only 1.73 ns: thus the critical path is defined by the delay of the inverter, as desired. The chosen clock period is equal to 3.8 ns, where the time length of the current spike is again of 150 ps: the operating frequency then is equal to 263.1 MHz, almost three times the frequency achieved with the first version of the adder.

The 16-bits adder has been verified in the 0 + 0, 65535 + 65535 and 0 + 65535 cases and with 25 couples of random inputs. The latency is equal to 355.6 ns.

4.3. Pipelining

Having now a version with enhanced performances, it makes sense to try to verify the behaviour of the adder when providing new data at each clock cycle. In the previous simulations, in fact, the interval adopted between each set of data was equal to the latency of the adder, and so the fact that the sum bits came out at different instants of time wasn't such a big deal. The skyrmions representing the sum output are in fact provided at different instants of time, just like the carry-out bit, that has to be propagated all along the chain: if the aim is to have a new result every clock cycle, then it is necessary to synchronize all these sum bits with one another. This is done by an additional skewing structure at the output of the adder, like shown in figure 4.11. Here the delay between the bit $S(i)$ and the bit $S(i-1)$ is equal to the number of pipeline stages inside $FA(i)$: if $FA(i)$ has six pipeline stages, like in this case, then the sum bit $S(i-1)$ must be delayed by six clock cycles, and so six notches are needed at the output of $FA(i-1)$. So, the law that gives the number of output notches is $Notch(i) = 6(N-1-i)$ for $0 \leq i \leq N-1$, where N is the adder parallelism.

Since the structure of the adder hasn't changed, the clock period is again of 3.8 ns. After a latency of 357.7 ns, the adder is able to provide a new result every clock period. Its behaviour has been verified in the 0+0, 65535+65535 and 0+65535 cases and with 60 couples of random inputs.

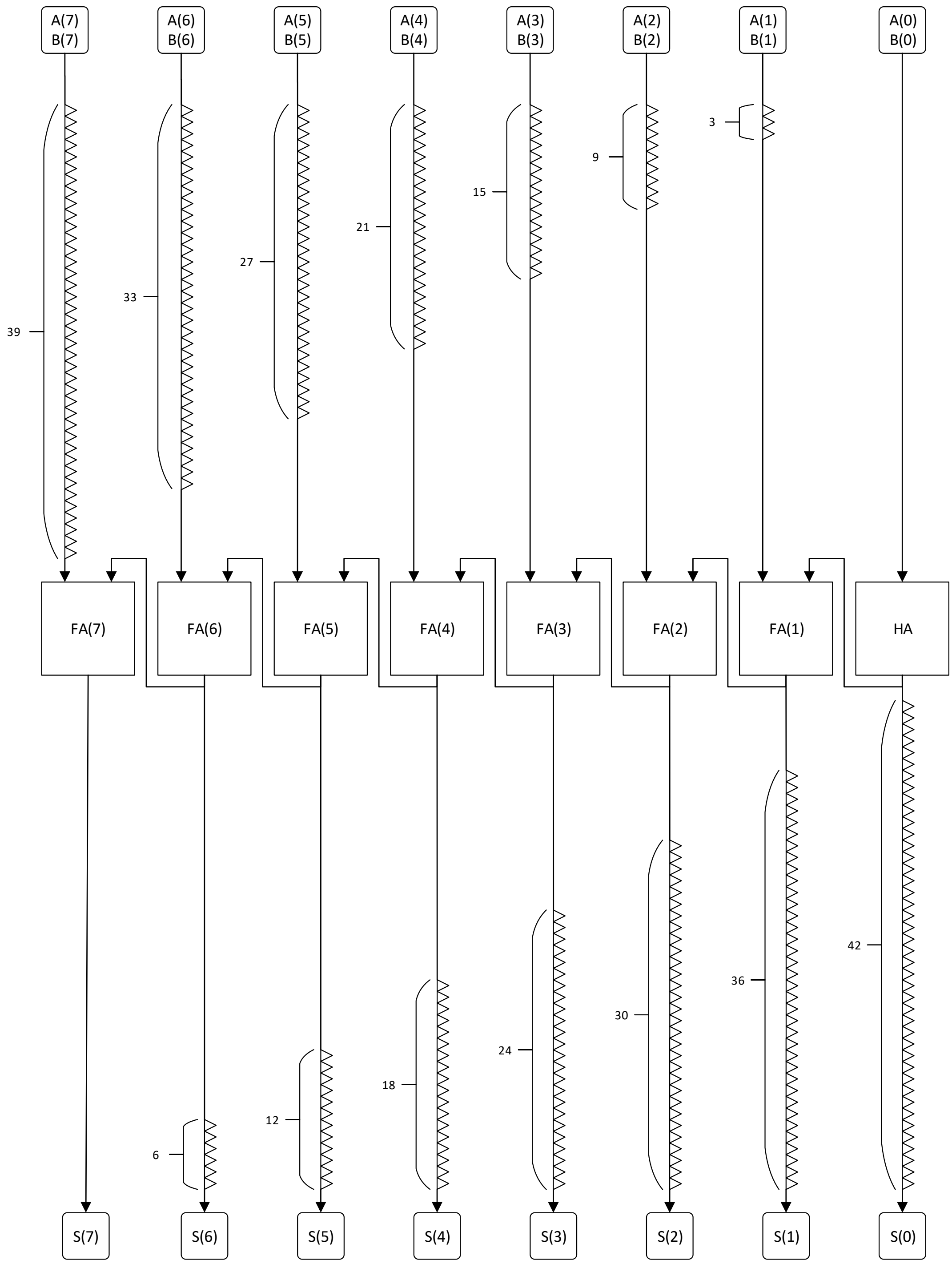


Figure 4.11. Sketch showing the skewing network both on the input and on the output of the second version of the ripple carry adder.

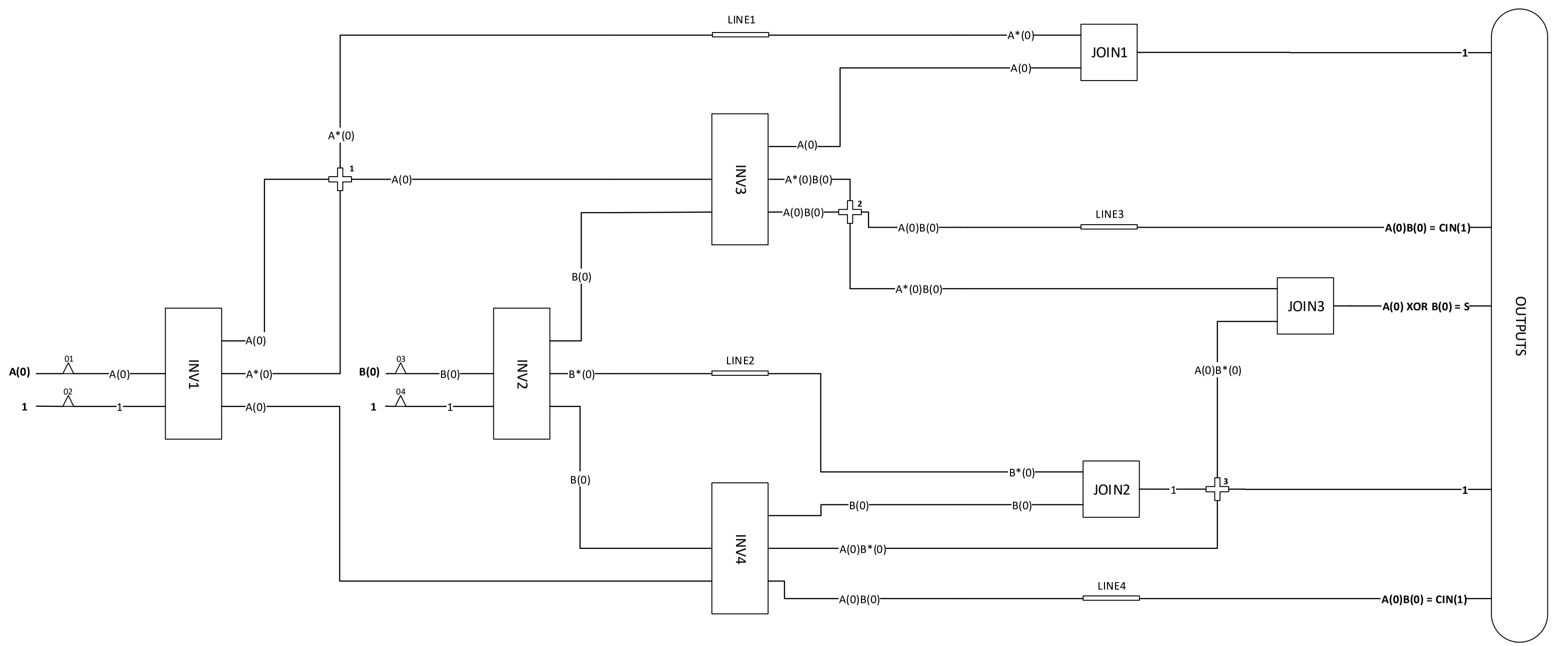


Figure 4.12. First version for the Half Adder. Each input is provided just once, while two carry-out and two signals equal to 1 come out from the structure towards the following Full Adder. Some *line* elements are still needed for the data synchronization. Three *cross* elements appear as well. One level of pipeline is present on the inputs, in order to assure the correct synchronization of the input data.

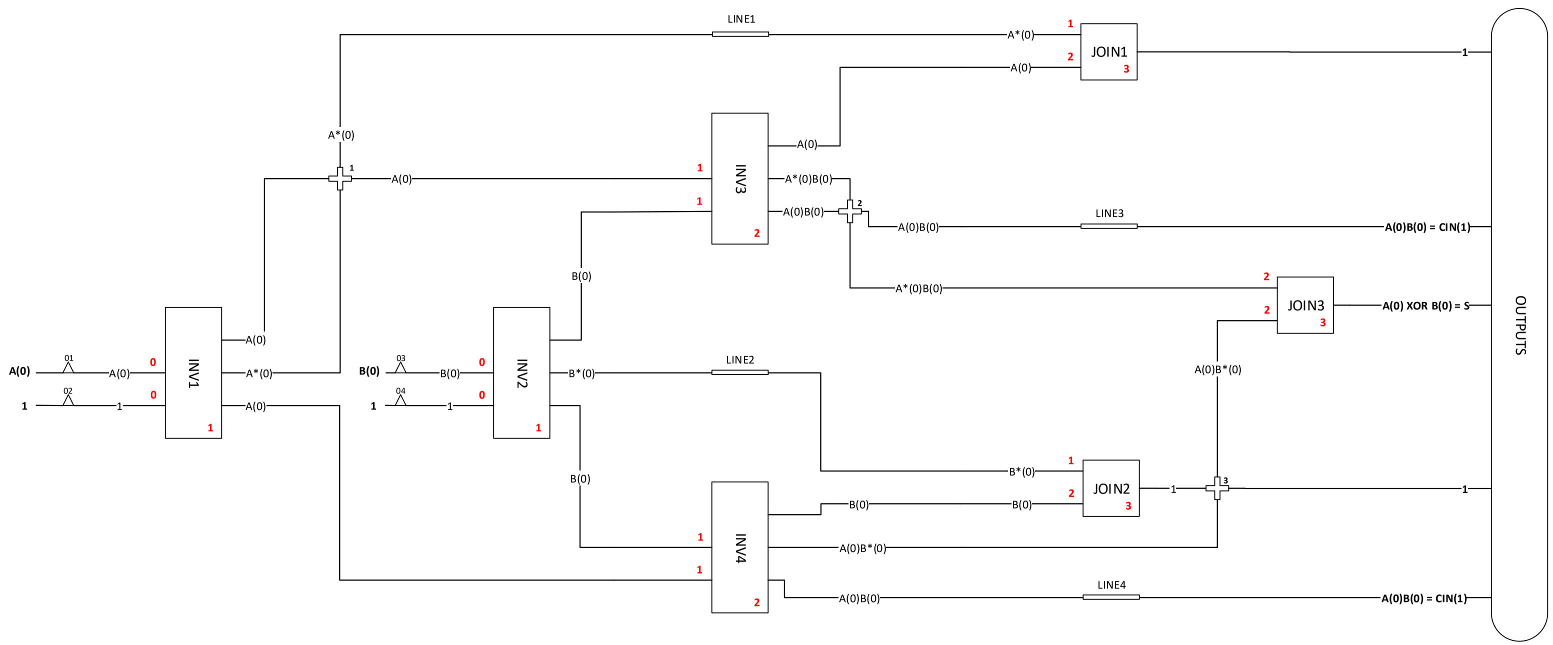


Figure 4.13. First HA version showing the logic depth for each element (red numbers). The *line* elements are placed whenever two inputs to the same logic gate have different logic depths. They are used to synchronize the outputs from the HA as well. The *cross* elements are assumed not to introduce any delay on the signal propagation and so they are not associated to any logic level.

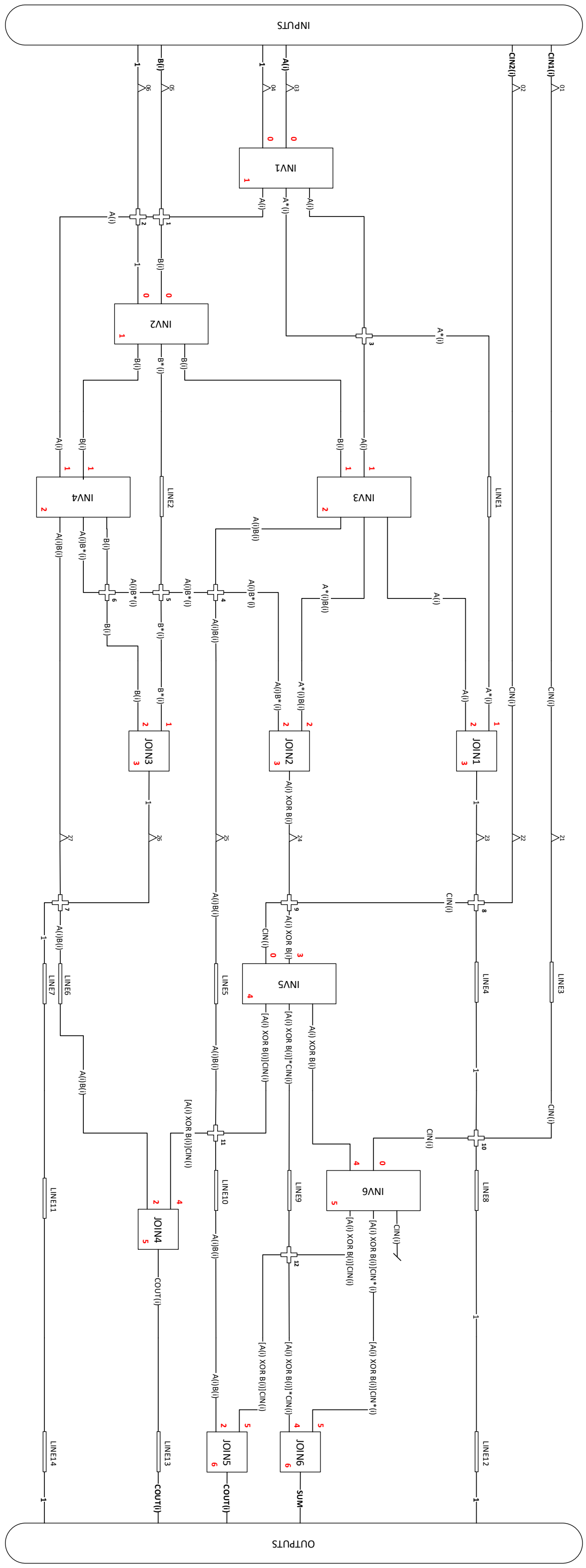


Figure 4.14. First version for the FA. Apart from $A(i)$ and $B(i)$, whose value is unknown and they must be nucleated according to the input data, all the remaining inputs are provided by the previous stage. All of the outputs, apart from the sum bit, are provided twice towards the following stage. One level of pipeline is used at the input to assure the data synchronization, and one more level of pipeline is in the middle of the structure to break the critical path and make it equal to the delay of the HA. Some *line* elements are still needed for the data synchronization. The red numbers represent the logic depth of each element.

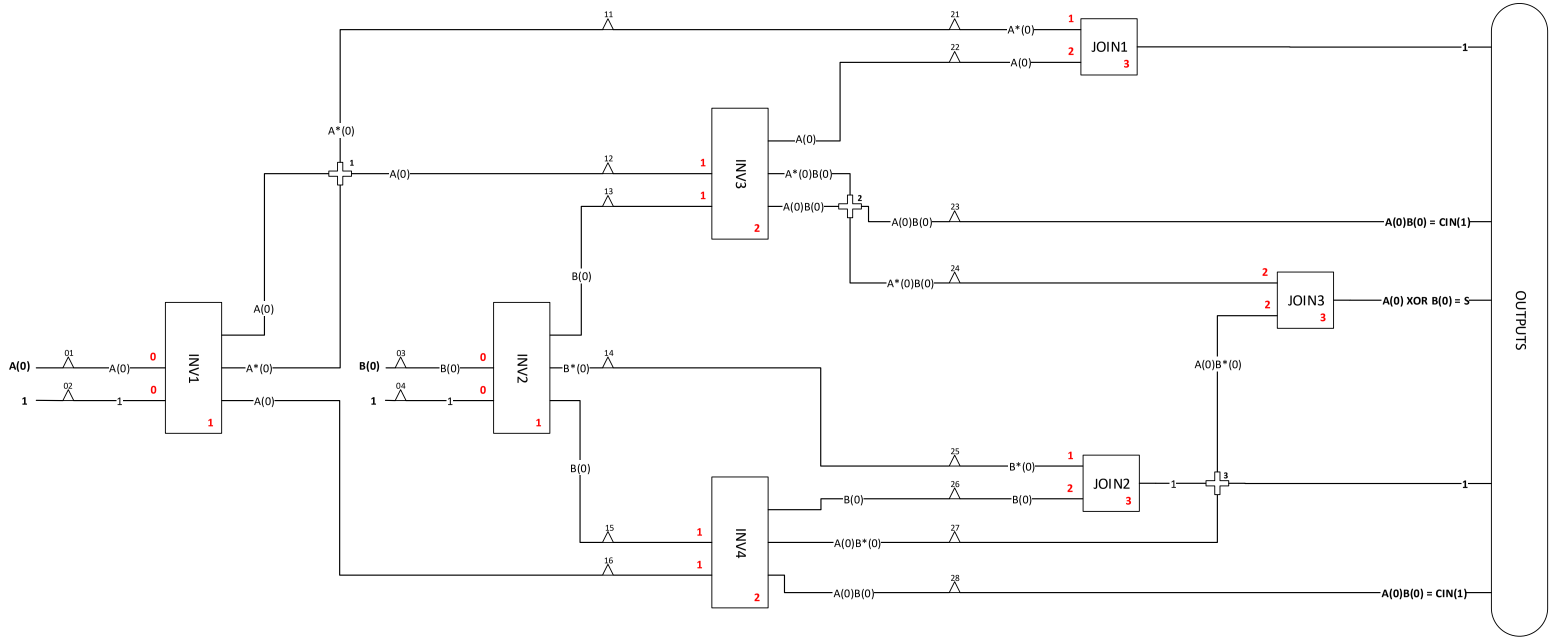


Figure 4.15. Second version of the HA. All the *line* elements have disappeared and the number of pipeline stages is now equal to three. The red numbers represent the logic depth of each element.

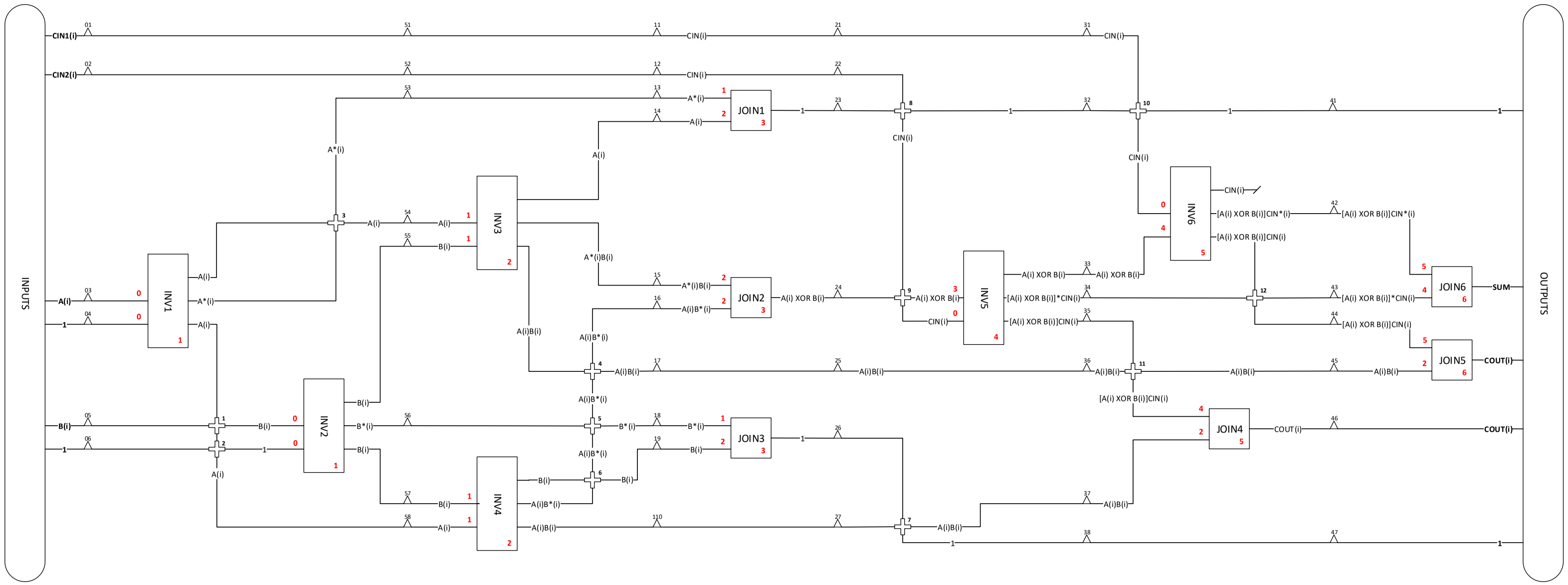


Figure 4.16. Second version of the FA. All the *line* elements have disappeared and there are six pipeline stages. The red numbers represent the logic depth of each element.

5. Logic in memory - first architecture

The final aim of this thesis has been to use the logic gates from [5] to build a logic-in-memory (LiM) architecture.

The first attempt that was made consists in a direct mapping from a CMOS architecture that had already appeared in literature: in [34] was in fact proposed a model for LiM cells that allows a high degree of flexibility and adaptability to a wide range of different algorithms. In figure 5.1 is shown the high-level organization of the original cells, which from now on will be used as a model: as it can be observed from the figure, each cell is able to communicate with the surrounding ones, exchanging both results and carry bits.

The simplest among the depicted cells is cell 00, that is, the cell at the intersection between the first row and the leftmost column. The bold red numbers that appear in the picture label the multiplexers needed for directing the data flow. Cell 00 is able to elaborate the stored bit together with a value coming from the external world and provided by the input *EXT_IN*; starting from these data, both the configurable logic and the full adder perform their computations and provide a result. Multiplexer *2* selects the result of interest, which may be transmitted towards the other cells or could be needed to update the value stored in the cell, while the other result is discarded. The value stored in the memory cell can be updated not only with the result produced by the cell itself, but also with a value coming from the bit line, thanks to multiplexer *1*: this is the method used to initialized the memory at the very beginning of the elaboration.

Either the value stored in the cell or the result produced by the logic blocks can be sent to the cell 10, passing through multiplexer *3*; the output of multiplexer

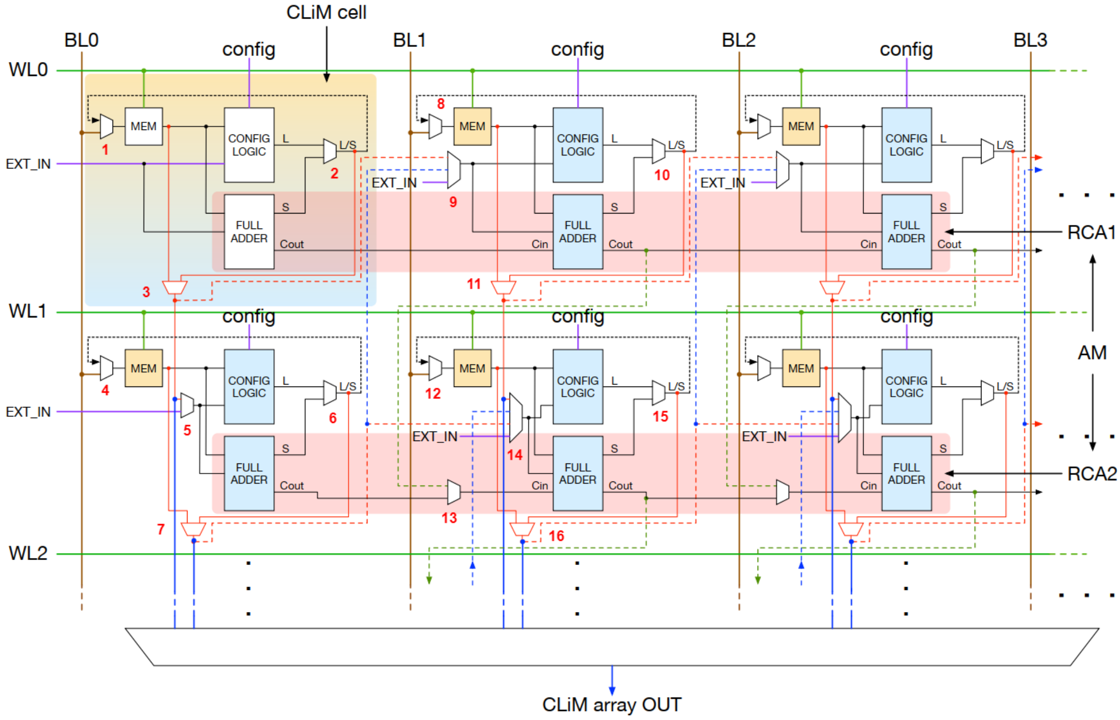


Figure 5.1. Structure of the memory array proposed in [34]: each cell contains a FA and a configurable logic block, and is able to exchange the results with the cells nearby, according to the needs of the algorithm to be implemented. Figure adapted from [34].

3 is one of the two possible choices available for the second operand of cell 10. Differently from cell 00, in fact, inside this cell, slightly more complex, is present also multiplexer *5*, which allows to choose between two values for the second operand of the elaboration, while the first operand remains the stored value, as before. The function of multiplexers *4*, *6* and *7* hasn't changed with respect to the corresponding multiplexers inside cell 00.

Multiplexer *9* inside cell 01 has three inputs: the second operand here can be either the result of cell 00, or the result of cell 10, or the external input. The input carry to the full adder, moreover, comes from cell 00; since the structure of all the other cells inside row 0 will be equal to the one of cell 01, this additional input allows to configure the whole row as a ripple-carry adder, where the carry chain is established by the connections between the carry-out of cell *(0,i)* (row *0*, column *i*) and the carry-in of cell *(0,i+1)*. This possibility, as discussed more in detail in [34], ensures a great flexibility of the structure and even allows to use the memory array as a multiplier.

Cell 11 is the most complex out of the four, and it's the one that is used in the majority of the memory array: while cell 01 is used through the whole row 0 and cell 10 through the whole column 0, the structure of all the remaining cells, apart from cell 00 which has its own organization, will copy the one of cell 11. In this cell, multiplexer *14* has now four inputs: the second operand can be either the result of cell 01, or of cell 10, or of cell 20 (which lays outside the picture), or the external input. Also the input carry comes from a multiplexer (*13*), and can be either the output carry of cell 10, or the output carry of cell 01: this allows the memory array to work as an array multiplier, as detailed in [34].

From this brief explanation it should be clear now that is enough to convert into skyrmion-based cells only the cells 00, 10, 01 and 11, because the remaining cells inside the array are all equal to one or the other.

Since all the memory cells are connected to one another and exchange their results, is clear also that their functioning must be delayed in time: if cell 10 needs the result of cell 00 in order to perform its computations, it cannot for sure start the elaboration together with cell 00. This means that the memory array must be controlled by a Finite State Machine (FSM) that launches the elaboration of cell 00, waits until its results are available, and only then enables cell 10 to start its own computations. Of course, as soon as cell 00 has finished the first elaboration, it can receive new data in order to go on and produce new results. This sequence applies throughout the whole memory array: all the cells connected to cell 10 will have to wait before receiving their own start, and so the cells after them.

Figure 5.2 shows the time requirements for each of the cells. Cell 00, marked in red, is the first cell that starts the elaboration, since its second operand can only come from the external world. When the results of cell 00 are available, cell 10, which doesn't need anything else, can start the computation. The results of cell 00 are received by cell 01 as well; however, this cell needs also the results of cell 10 in order to start; that's why it is marked in green, together with cell 20: they both need the results of cell 10, and so their functioning is allocated in the third phase of the elaboration. Finally, cell 11 needs the results of cell 10, 01 and 20, and so it must be allocated in the fourth phase.

Another key point, of course, is the need for cell 10 to maintain stable its results

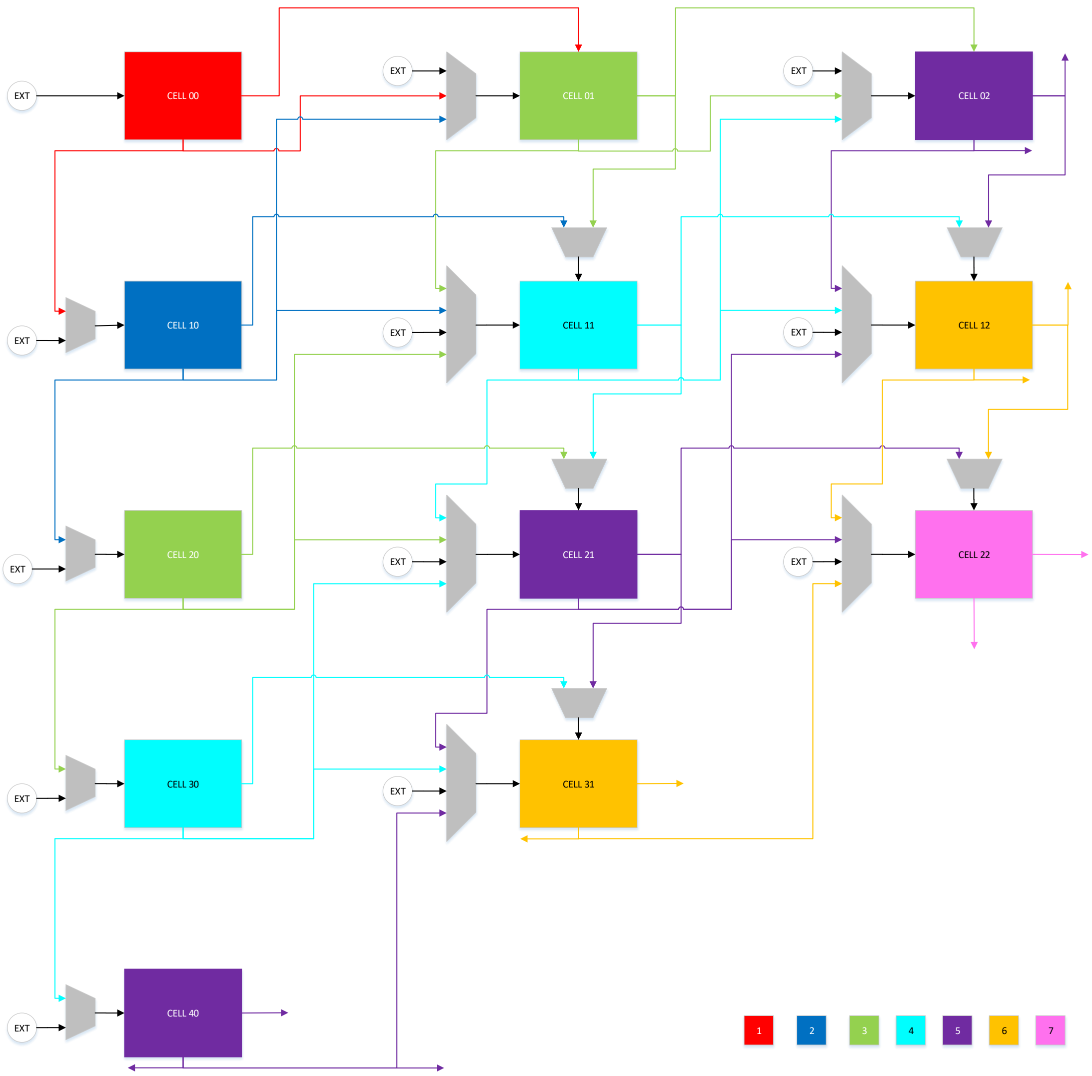


Figure 5.2. Scheme showing the timing requirements for some of the cells inside the array. The colour of each cell corresponds to the phase into which its functioning time is allocated, as detailed by the legend inside the figure; the same colour applies also to the outputs from the cell. Each cell can start its elaboration only when all the inputs are available.

until cell 11, which is the most distant in time regarding the start of the computation, has received them. Following the same reasoning, also cell 00 needs to maintain available its results until cell 01 has read them. This means that, exploiting the particle-nature of skyrmions, each of these cells could even receive the new data and start a new computation while maintaining available the skyrmions on its outputs.

This, however, is just a matter of optimization: first of all is necessary to find a correctly working FSM, and only then the architecture can be optimized and made faster. For this reason, the assumption from now on will be that each cell will be frozen in its functioning cycle until all its results have been collected and read by the proper cells connected to it.

Since each cell contains a number of multiplexers, and since potentially there could be the need to drive two or more of them at the same time, according to the particular instant in which each cell is working and to the needs of the particular algorithm that is being implemented, is necessary also that each cell has its own FSM. Considering four cells only, their four FSMs will then be coordinated by a master FSM, which receives their status signals and decides what to do accordingly. The FSM of each cell (let's call them slave FSM) will receive a *start* signal from the master FSM and will notify it when the result is available. Each slave FSM, moreover, will receive from the external world some signals that depend on the particular algorithm that needs to be implemented. These signals will be used for deciding, for each multiplexer, which input should be selected. Since these signal depend on the particular algorithm chosen they cannot come from the master FSM, and must be provided from outside directly to each cell. In the schematics of the FSMs reported at the end of this chapter, these signals, coming from the external and arriving directly to the slave FSM of competence, are all named as *DES_x*, where x depends on the particular signals. Some examples are *DES_EXTIN_00*, towards the slave FSM of cell 00, or *DES_DATA_X0*, towards the slave FSM of cell 10.

A final point that should be underlined here regards the *configurable logic* block that appears inside each cell in figure 5.1. The particular type of logic gates presented in [5] and discussed up to now are predetermined in their behaviour by their own shape: the *AND/OR* gate has a particular shape that will never allow it to behave as an *INV/COPY* gate. In [28], however, a design for reconfigurable logic gates has been proposed: by simply changing the voltage pattern applied to the structure, the logic function of the gate changes. To build a reconfigurable logic block using the gates proposed in [5] a much more complex structure should be designed instead, together with the control signals needed to allow its functioning. Since the structure of the memory array is already quite complex, the reconfigurable logic block inside each cell in figure 5.1 has been changed in this work into a fixed logic block composed by a single *AND/OR* logic gate. A more advanced logic elaboration could be realized, however, by simply replacing this single gate with a more complex logic block.

5.1. Cell 00

The structure of cell 00 is reported in figure 5.22; the FSM that controls it is described in figures 5.23 and 5.24, where are detailed also the control signals that are activated during each state. The VHDL describing both the datapath and the FSM of the cell is reported in appendix C. Since the datapath is huge and quite complex, some small portions of it will be shown in this chapter little by little, as the explanation of the structure goes on, in order to facilitate the comprehension.

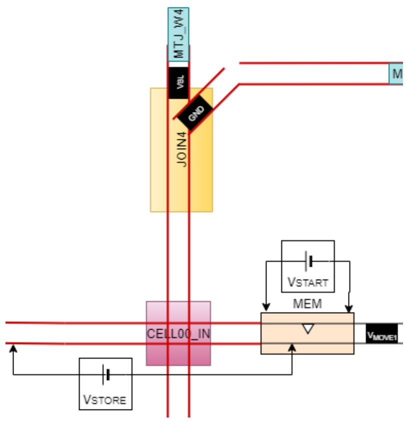


Figure 5.3. Portion involved in the nucleation and in the storing of a new skyrmion inside the memory element.

In realizing these cells, the results proposed in many different articles have been exploited. The mechanism used for initializing the memory comes from [44]. Differently from the article this is a random access memory, not just a racetrack: however, similarly to what happens in the article, the skyrmions are nucleated at the beginning of the bitline by a writing head (labelled as *MTJ_W4*) and put into movement by applying a voltage V_{BL} all along the bitline; to identify the point where the voltage must be applied, some black contacts carrying the name of the voltage signal (including *GND*) have been inserted in the picture.

As soon as the desired skyrmion reaches the input of its destination cell, the voltage V_{BL} is switched off and the voltage V_{STORE}, applied orthogonally with respect to the previous one, is turned on, so that only the skyrmion at the intersection between the bitline and the cell input (this intersection is labelled *CELL00_IN* in the picture) is pushed towards right, entering the memory element. This memory element is simply a notch, introduced in chapter 2 and widely used in chapter 4 as a synchronization element for the data flow. Since the skyrmion that reaches the notch will stay there until a current peak is applied, the notch behaves exactly like a memory element.

This current peak is generated when the voltage source V_{START} is turned on. This voltage source is different from the sources such as V_{STORE} or V_{BL}, because the current value needed to make the skyrmion go through the notch is much higher than the value needed to simply put it into movement. For this reason, the VHDL description of these voltage generators is different: while V_{STORE} and V_{BL} are described by the component *voltage_genL*, V_{START} is described by *voltage_genH*. The difference in the behaviour between these two components is that *voltage_genL* allows a current flow of intensity *CURRENT_LOW* as long as its control signal is active, while *voltage_genH* generates a single peak of current of intensity *CURRENT_HIGH* and then turns off, if its control signal is active for a single clock cycle.

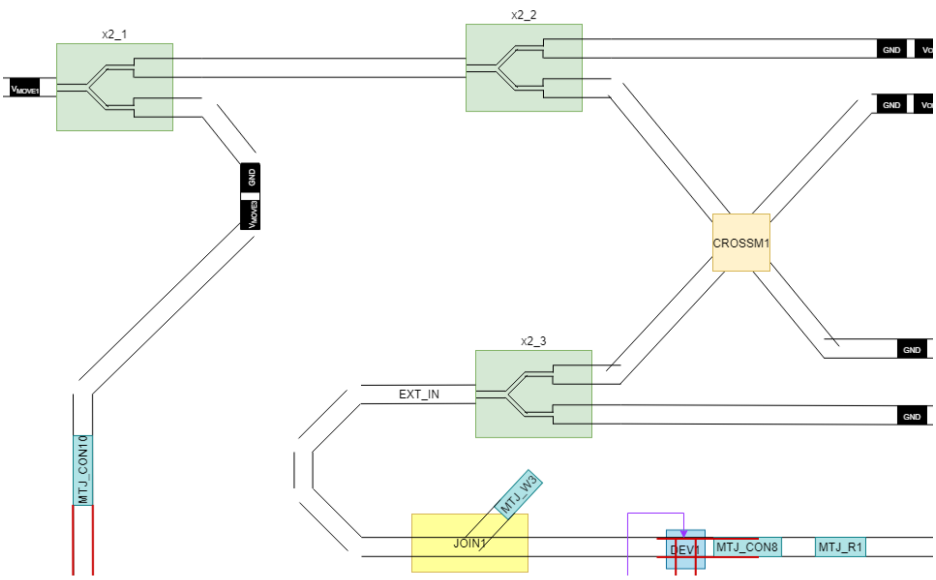


Figure 5.4. Path followed by the first and the second operand towards the computational area.

The current that allows the skyrmion to leave the memory element's output and to reach the region where both the FA and the logic are allocated is generated by V_{MOVE1}, again of type *voltage_genL*. Along its path the skyrmion finds some duplication elements, needed to provide its value both to the two computational elements and to the structure that implements the multiplexer *3* of figure 5.1. These duplication elements have been presented in [40] and exploit the reversible conversion from skyrmion to domain wall pair and vice versa, realised thanks to a chain of large and narrow junctions. So, pushed by *CURRENT_Vmove1*, the skyrmion goes through the elements *x2_1* and *x2_2*, through the element *CROSSM1* (exactly the same type of cross used also in chapter 4), and finally reaches the input of the FA and of the logic block.

To perform the data elaboration, however, the memory cell must receive also the second operand, which in the case of cell 00 can come only from the external world. Not only: this data must reach the computational region together with the skyrmion that just come out of the memory element. To do so, the output of a writing head *MTJ_W3* is conveyed inside the track by the element *JOIN1*, exactly the same join structure used also in chapter 4, and the skyrmion just nucleated is moved along the track again by the current *CURRENT_Vmove1*, the same that moves also operand 1. Before reaching the computational region it is duplicated by element *x2_3* and goes through *CROSSM1*.

When the voltage V_{MOVE1} is turned on, also the write heads *MTJ_W1* and *MTJ_W2* are activated. Their presence is due to the structure of the FA, which is the same that has been developed in chapter 4: in order to correctly work, the FA must receive as input two additional skyrmions, which do not carry any information but are just needed as enable signals. The duty of those two write heads is to provide these two skyrmions to the FA. The two skyrmions are put into movement by the same current which transports also the two operands, so they will reach the input of the FA at the same time, and they will be given back by the FA together with the other results (the sum plus the two output carries).

Two key points must be discussed here. The first revolves around the timing requirements: in the description of these cells many assumptions around the topic of timing synchronization have been made. These assumptions, however, are not

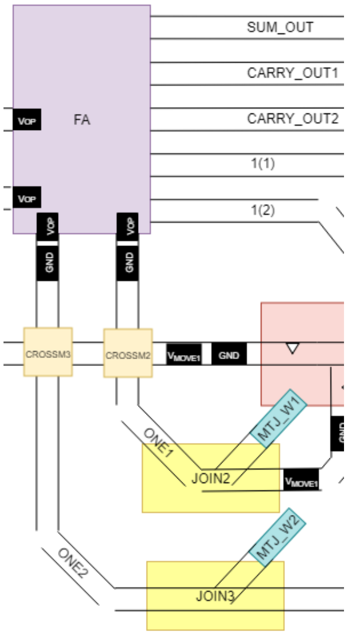


Figure 5.5. Nucleation of the two skyrmions needed as enable by the FA and outputs from the FA.

difficult to be satisfied: technologically speaking is in fact enough to tune the length of this or of that track in order to tune the time that each skyrmion needs for going from one point to another, and so synchronizing their movement is not particularly challenging.

The second point is a bit more tricky and involves the different voltages applied. Simply looking at figure 5.22 and remembering that each black rectangle is a voltage contact, is easy to guess how many different voltages are involved in controlling each of these cells. The value of these voltages, however, is in many cases always the same: the only request is to have a current flowing inside the tracks at least equal to *CURRENT_LOW* (that is, a current density higher than the depinning threshold), so, as long as the skyrmion can move, there is no reason for changing the value of the voltage applied. As already mentioned, it is necessary to have a second voltage value, able to induce a current density equal to *CURRENT_HIGH*, otherwise the skyrmion wouldn't be able leave the memory element; this however isn't different from what already assumed in chapter 4. To make the FA work, in fact, is necessary to provide a current waveform that assumes a low (not null) value for most of the time, and then a peak value for a small extent of the clock period;

since this kind of waveform is needed also in these memory cells due to the presence of the FA, the voltage that imposes *CURRENT_LOW* can be made equal to the one that produces the low value of the current used in the FA, and the voltage that imposes *CURRENT_HIGH* can be made equal to the voltage that produces the peak.

The most tricky point in controlling all these voltages, however, is due to the need of having regions without any voltage applied adjacent to regions where instead a current is flowing to make the skyrmion move. This condition corresponds, inside figure 5.22, to all the points where a *GND* contact is placed nearby a contact with a different label. Technologically speaking, it should be possible to realize this condition by cutting the metal trace for a very small extension and interposing between the two pieces an insulating material. Since the voltages needed are not high (in chapter 3 values around 1 mV have been used), there shouldn't be problems of breakdown; it should also be possible to allow the skyrmion to overcome this region, if very small, without heavily compromising the DMI which stabilizes it at the interface between the heavy metal and the ferromagnetic material.

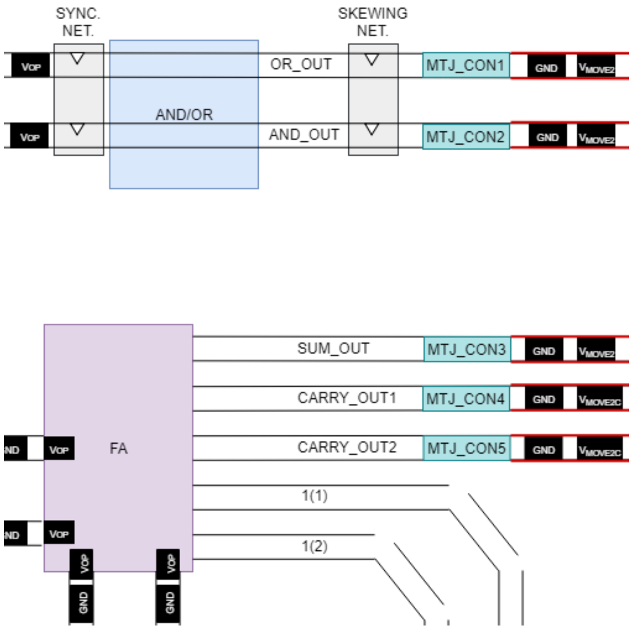


Figure 5.6. Logic block and FA, together with the contacts used for applying V_{OP}.

When the data arrives the the beginning of the computational region of the cell, V_{MOVE1} is switched off and V_{OP} is applied. As already explained, V_{OP} has exactly the same waveform used for controlling the adder of chapter 4, so its VHDL description must be different from the one of the other voltage sources already encountered: the component that describes this voltage source is *vclock_gen*. Since the final version of the FA has a pipeline with 6 stages (including also the row of notches placed right on the input), two set of notches have been inserted also before and after the *AND/OR* logic gate: *SYNC. NET.* is composed by a single row of notches, while *SKEWING NET.* is made by five rows of notches one after the other. Their presence, however, is not fundamental, because the movement of the skyrmions is controlled by the voltage applied: even if the output of the *AND/OR* block were available before the second clock cycle since the switching on of V_{OP}, the skyrmions wouldn't be able to move on along the trace, because the voltage V_{MOVE2} would still be turned off.

The elements denoted as *MTJ_CON* are made by the sequence of a reading head and of a writing head. Elements of this type have been inserted in the picture wherever a change in the technology of the tracks is needed. Inside figure 5.22, in fact, two types of tracks can be distinguished: the former type is denoted by black tracks, the latter by red tracks. The reason for this difference is the need, in some regions of the cell, to suppress the Magnus force that makes the skyrmion turn towards left as soon as the possibility is available. In these regions the confinement adopted in [5] wouldn't work, because it is good only to avoid the annihilation of the skyrmion at the edges of the nanotrack: as demonstrated by the working principle of the gates presented in [5], the Magnus force is still present and makes the skyrmion move towards left at each junction.

Like already discussed in chapter 2, a possibility for completely cancelling the Magnus force is to realize an antiferromagnetic coupling between two layers of ferromagnetic material separated by an insulating spacer. Doing so, with the same type of writing head used also for the component *MTJ_W3*, is possible to nucleate a skyrmion in the top layer and an antiskyrmion in the bottom layer; thanks to their opposite topological charge and to the coupling between the two FM layers, the Magnus force is cancelled by construction and the bilayer skyrmion moves along a straight line even if there is the possibility for it to turn left at the junctions present along the track. So, in all the cases in which this kind on technology is needed, a red track has been used in figure 5.22, in place of a black track, that uses the technology exploited in [5]. It is worth noticing that a red track is used also at the input of the cell: in [44], in fact, it is used exactly the technology proposed in [41].

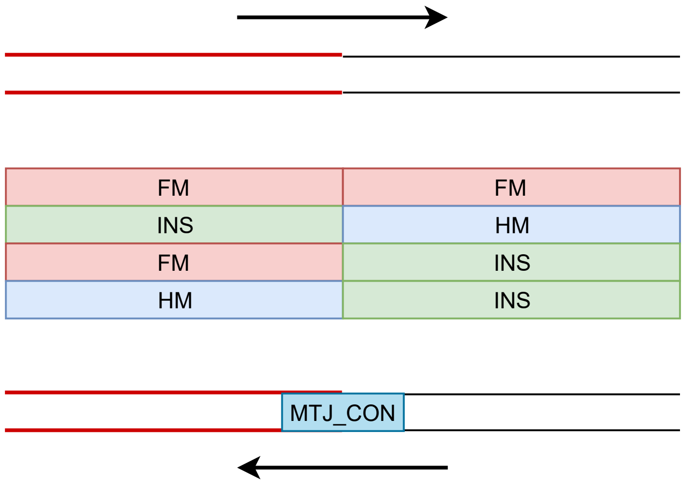


Figure 5.7. Layers composing the technology used for the red and for the black tracks. A conversion head is required when going from a black to a red track, while it is not needed in the opposite direction of movement. FM stands for ferromagnetic layer, HM for heavy metal layer, INS for insulating layer.

Then, why are the elements *MTJ_CON* needed? These elements are placed only in the points where a black track (single FM layer above a platinum trace, with the confinement proposed in [10]) is replaced by a red track (antiferromagnetically coupled FM layers), and not vice versa, as shown schematically in figure 5.7. If one imagines to put the two types of track one next to the other, in fact (and assuming to have between them the space needed to put in contact the two HM traces, so that the current continues to flow from one track to the other), is easy to imagine that a bilayer skyrmion coming out from the red track splits into a separated skyrmion and a separated antiskyrmion as soon as the coupling vanishes; the skyrmion will find, going on along its path, the confinement structure used in the black tracks and will remain inside the track, while the antiskyrmion is no longer useful and can be destroyed. To do so, is should be enough to interrupt the bottom FM track with an insulating layer, as shown in figure 5.7: as a result, the antiskyrmion will be expelled from the bottom track, leaving the skyrmion alone in the top track, confined by the boundaries. If the direction of the movement instead is opposite, that is, from the black track to the red track, it could happen that the bottom skyrmion, as soon as the confinement vanishes, is expelled from the top track without being able to induce the formation of an antiskyrmion in the coupled layer. For this reason, it should be enough to detect the presence of the skyrmion before it is expelled, and to control accordingly a writing head placed just at the beginning of the red track. In this way, even if the skyrmion gets expelled from the track, the information doesn't get lost and a bilayer skyrmion can correctly be nucleated inside the red track, if needed. The sequence of the read and of the write head needed for this conversion, as mentioned, is summed up by the component *MTJ_CON*.

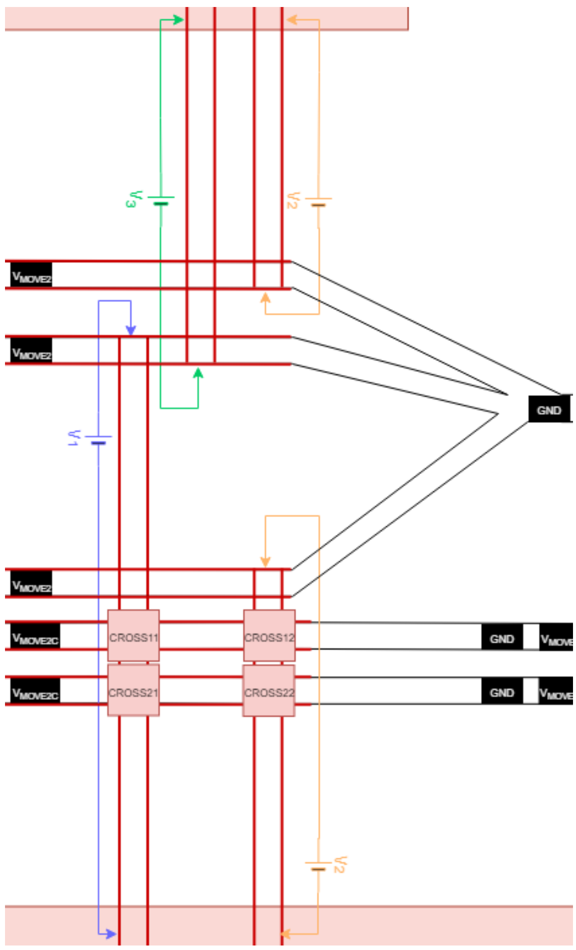


Figure 5.8. Structure that implements the multiplexer *2* of figure 5.1.

The red tracks are needed after the computational structures due to the multiplexer *2* of figure 5.1: having computed the result of the sum, of the *AND* and of the *OR* between the two operands, it is now necessary in fact to choose one of the three results, in order to send it to the neighbouring cells and maybe to store it inside the memory element of cell 00 itself. So, there is the need of a multiplexer with three inputs. Since with skyrmions the information is always conserved, the two skyrmions at most that will be discarded have to go out from the structure as well: for this reason the multiplexer has one main output, plus four other secondary outputs. Among these four secondary outputs, two of them are traces controlled respectively by the voltages V_1 and V_3, while the remaining two outputs are both controlled by the voltage V_2. The selection of the input to be transmitted is done in the following way: first the voltage V_{MOVE2}, together with V_{MOVE2C}, which controls the movement of the output carry, is turned on in order to allow the three (at most) input skyrmions to reach the intersection with the first secondary output, which is controlled by V_1, and then they are both turned off. In order to allow the output carry to overcome the crosses *CROSS11*, *CROSS12*, *CROSS21* and *CROSS22*, V_{MOVE2C} is switched on for one more state. At this point the selection can take place: if the result of the *OR* is desired, then V_1 is switched on, so that both the skyrmion carrying the value of the *AND* and the skyrmion coming out from the FA are forced to move along the first secondary output, crossing both *CROSS11* and *CROSS21* in the vertical direction, up to the bottom tank. At this point V_{MOVE2} can be turned on again, allowing the selected result to reach the output of the multiplexer (that is, the *GND* contact right at the end of the multiplexing structure). If instead the *SUM* result is desired, the voltage V_{MOVE2} will be turned on again for a small amount of time, allowing the three skyrmions to reach the second secondary output: then, turning on V_3, both outputs of the *AND/OR* gate will be flushed away towards the top tank; applying again V_{MOVE2}, the only skyrmion left can reach the output of the multiplexer, just like before. Finally, if the result desired is the output of the *AND*, first the three skyrmions will reach the final secondary output; then, turning on V_2, the top and the bottom skyrmion will be flushed respectively towards the top and the bottom tank, leaving the middle skyrmion alone. In any case, one result will be available at the end of the multiplexing structure, while the skyrmions carrying the value of the two other results are collected inside the top or the bottom tank, or maybe inside both of them (if V_2 was turned on).

The top and the bottom tank are two structures able to collect all the skyrmions that have been nucleated inside the cell, but that are not needed as information

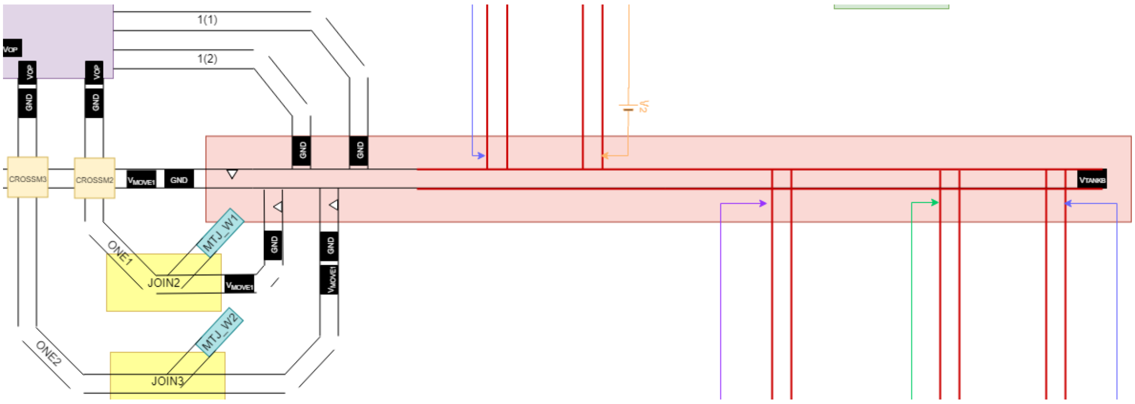


Figure 5.9. Structure of the bottom tank, with the inputs and outputs involved in the movement of the skyrmions that are used as enable by the FA.

carriers in the current computational cycle. Since nucleating new skyrmions is an energetically expensive operation, having spare skyrmions ready to be used when needed should highly decrease the power consumption associated to each computation. The top and the bottom tank inside each cell of this design are able to collect all the unused outputs of each multiplexing operation, together with the two input skyrmions needed by the FA to correctly perform its computation. Focusing on these two skyrmions, the FA gives them back as an output at the end of the computation cycle (as already explained in chapter 4): the two skyrmions then reach the input of the bottom tank thanks to *CURRENT_Vop*, and will stay there from the moment when the voltage V_{OP} is turned off. Before starting a new computational cycle, all the skyrmions available at the inputs of the tank will be pushed inside the its middle nanotrack by applying *CURRENT_Vtankb* (which is of type *vclock_gen*). The bottom tank has three outputs, differently from the top tank, which has only one output. The three outputs are connected to the nearby traces according to a priority order: inside the bottom tank, in fact, there will be for sure at least two skyrmions (the two skyrmions provided by the FA at the end of the elaboration). These two skyrmions must be given back to the FA in order to allow a new computation: for this reason, when *CURRENT_Vtankb* is applied, since those tracks inside the bottom tank are of the black type (that is to say, the Magnus force is still present), the skyrmion coming out from the track labelled as *1(1)* will occupy the output of the tank that enters the element *JOIN3*, preventing other skyrmions to occupy the same output thanks to the skyrmion-skyrmion repulsion; at the same time, the skyrmion coming out from *1(2)* will occupy the output that goes towards the element *JOIN2*. Both skyrmions will occupy the corresponding output until a current peak is provided, thanks to the notches that block them.

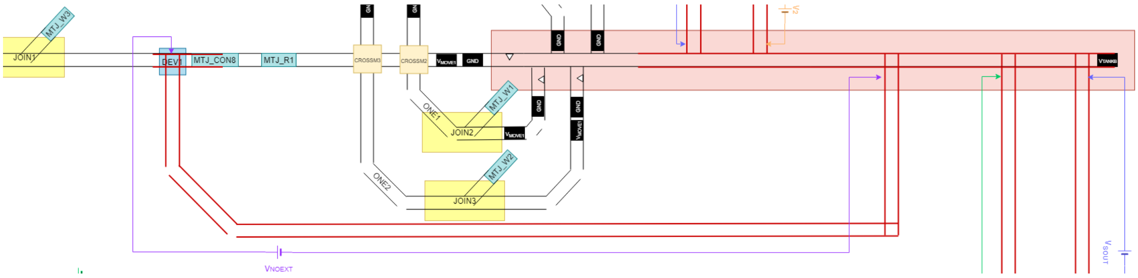


Figure 5.10. Structure of the bottom tank, with the connections needed to put back inside the tank a skyrmion that came out from the lowest priority output.

If the tank hosts more than two skyrmions, one more skyrmion will go out from the remaining output, which, thanks to is placement, has the lowest priority. If desired, this skyrmion could be used as a second operand in the following computational cycle, without the need of nucleating a new skyrmion with the *MTJ_W3* head. However, the second operand desired could be equal to 0, that is, no skyrmion is required: what if the tank hosts more than two skyrmions? When the current peak allows the two skyrmions with highest priority to go out, also a third skyrmion will go out from the tank. Since its presence is not desired, this condition must be detected and, in case, the skyrmion must be put back inside the tank. To do so, the read head *MTJ_R1* detects the presence of the skyrmion and notifies it to the FSM of cell 00, and at the same time *CURRENT_Vmove1* will push it inside the component *DEV1*: if a skyrmion was detected and its presence is undesired, the voltage V_{NOEXT} will be turned on, flushing it away back inside the bottom tank. If instead the skyrmion detected is desired, *CURRENT_Vmove1* will be turned on again, allowing the skyrmion to go across *JOIN1* and in the end to reach the input of the two logic blocks, together with operand 1. If, on the contrary, no skyrmion is detected while operand 2 should be equal to 1, then a new skyrmion must be nucleated: exactly as it happened in the first half of the cycle, *MTJ_W3* will be turned on together with *CURRENT_Vmove1*, so that the just nucleated skyrmion is able to reach the computational area together with operand 1.

These operations, however, take place only inside the second half of the cycle, that is, after the first iteration has been completed and the first results have come out. Before them, is necessary to complete the first cycle transmitting the results to the nearby cells and updating the value stored in the memory element.

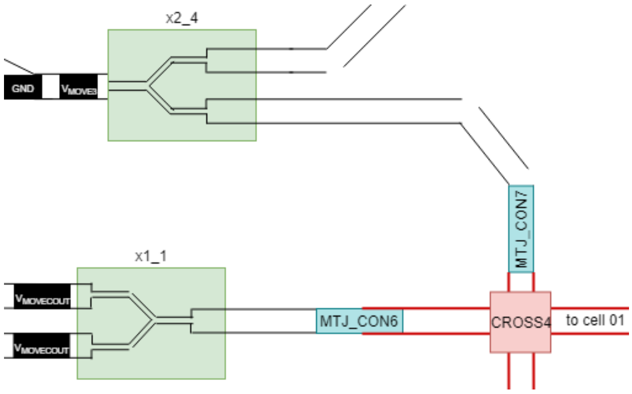


Figure 5.11. Element that duplicates the result and merging element that compresses the two output carries into a single particle.

First of all, the skyrmion which survived the selection among the three results is pushed by *CURRENT_Vmove3* through the element *x2_4* (so that the result will be available also for the store operation, later in the cycle), through *CROSS4*, up to the junction controlled by voltage *V_SOUT*.

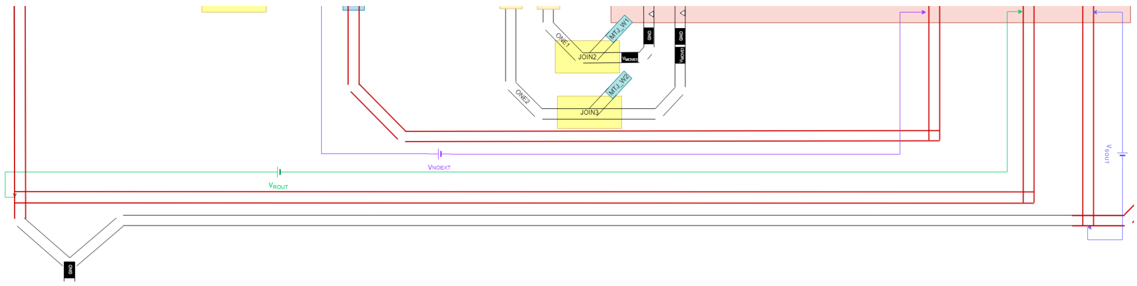


Figure 5.12. Structure that implements multiplexer *3* of figure 5.1.

CURRENT_Vmove3, however, moves also the skyrmion carrying the value stored inside the memory element, which was doubled by element *x2_1* and is ready to be used: this skyrmion so will moved by *CURRENT_Vmove3* up to the junction controlled by V_{ROUT}. Now the multiplexing operation carried out by multiplexer *3* of figure 5.1 takes place: if the output of the cell has to be the result previously selected, V_{ROUT} is turned on, flushing the value of the stored element towards the bottom tank; if instead the value of the memory element is desired, V_{SOUT} is turned on, flushing away the skyrmion carrying the value of the result computed. In any case, the undesired skyrmion is collected inside the bottom tank. Then V_{MOVE3} is turned on again, allowing the selected skyrmion to reach the *GND* contact right at the output of the multiplexer.

Now that the output has been chosen, the cycle is almost over. The remaining things to do are to choose what kind of data must be used to update the memory element, to transmit the outputs (the one just selected, plus the carry-out bit) towards the nearby cells, and to request a new start to the master FSM.

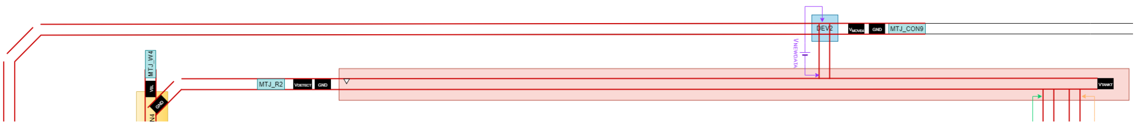


Figure 5.13. Structure of the top tank.

First of all, the skyrmion that was moved up to *MTJ_CON9* by *CURRENT_-Vmove3* is pushed inside *DEV2* by applying for a short time the voltage V_{MOVE4}. Then, if the result of the computation must be used to update the memory element, V_{MOVE4} is applied again, so that the skyrmion continues along the track until it reaches the next *GND* contact: there it will wait for the master FSM to turn on, after the activation of the signal *REQUEST_NEW_STORE* by the slave FSM, the voltage V_{STORE}, so that a new cycle can begin.

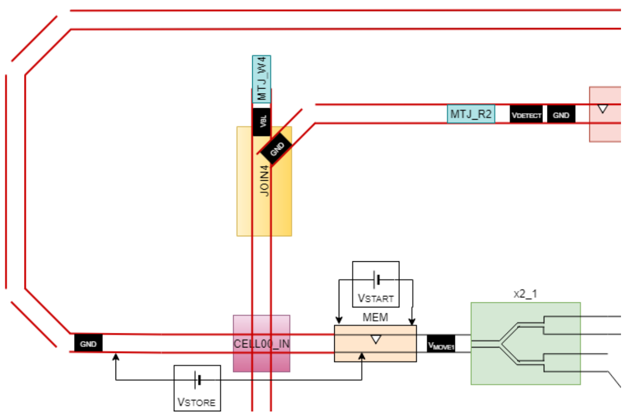


Figure 5.14. Connection between the output from the top tank to the input of the cell.

If instead the data to be written inside the memory element must come from the external, that is, from the bitline, the skyrmion inside *DEV2* will be flushed towards the input of the top tank by the activation of $V_{NEWDATA}$; then, according to the value desired for the data to be written in the memory element, two possibilities are available: either the data desired is 0, and so nothing has to be done, or the data desired is 1; if this is the case, V_{TANKT} is activated until the skyrmion that may be inside the top tank comes out: then V_{DETECT} turns on, allowing the skyrmion to pass through the read head *MTJ_R2*: if the skyrmion is actually there, the signal *REQUEST_NEW_START* is activated by the slave FSM. The master FSM, detecting the activation of this signal, will activate V_{BL}, which allows the skyrmion to go out from *JOIN4* and to reach the inside of *CELL00_IN*; then it will switch off V_{BL} and turn on V_{STORE}, allowing the data to update the content of the memory element without the need of nucleating new skyrmions. If instead the desired data is 1, while no skyrmion has been detected by *MTJ_R2*, the slave FSM activates the signal *REQUEST_NEW_START_W*, so that the master FSM knows that it has to activate the write head *MTJ_W4* together with V_{BL}; then the FSM turns it off and activates V_{STORE} to store the skyrmion inside the memory element. Finally, if the desired data to be written inside the cell comes from the external and it is equal to 0, after pushing the result of the computation inside the top tank, the signal *REQUEST_NEW_START* is activated: in this way the master FSM activates V_{BL} (doing so, if the bitline is by chance filled with skyrmions, they will be shifted by some positions, allowing a 0 to be positioned inside *CELL00_IN*), and then activates V_{STORE}, allowing the 0 to be ”stored” inside the memory element.

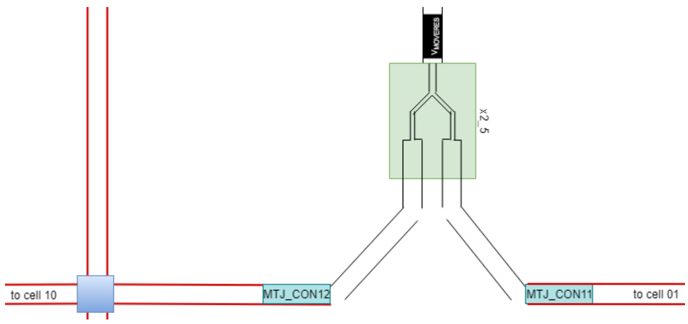


Figure 5.15. Duplication element placed at the output of the cell.

This is the sequence that is performed in order to update the memory element. However, this happens only after the results have been transmitted to the cells nearby. When the skyrmion has just been moved either by *CURRENT_Vmove4* until the input of *CELL00_IN*, or by $V_{NEWDATA}$ to the input of the top tank, the slave FSM freezes waiting for the activation of the signal *READY_FOR_DATA_-RIGHT*, which means that the destination cells, equal to cell 10 and cell 01 in the case of cell 00, are ready to accept new data from cell 00. This has the aim of avoiding that the skyrmions carrying the value of both the result and of the carry-out bit pile up at the input of each destination cell, since this would result into an error. As mentioned, a possible optimization would be to allow cell 00 to start a new computation while maintaining the results of the previous cycle fixed at the output ports and ready to be sent towards the destination cells.

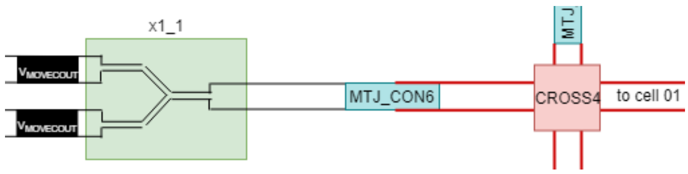


Figure 5.16. Merging element placed on the path of the output carry.

When the signal *READY_FOR_DATA_RIGHT* is activated, the slave FSM turns on the voltage sources $V_{MOVERES}$ and $V_{MOVECOUT}$: in this way the skyrmion of the result (or of the value that was stored in the cell) is duplicated before being sent to cell 10 and to cell 01 (both have, inside their structure, a *GND* contact for voltage $V_{MOVERES}$), while the two carry-out skyrmions are packed into a single skyrmion by the element *x1_1* and then are sent towards cell 01, where the *GND* contact for $V_{MOVECOUT}$ is placed. The element *x1_1* exploits again the results of [40], and is used just to simplify the routing between the cells; another possibility, since the two carry skyrmions have the same value, would be to send just one of the two towards cell 01, while the other could be collected inside the bottom tanks and maybe reused again, instead of nucleating a new second operand. Finally, to tell the master FSM that the results of the cell have been correctly transferred, the slave FSM activates the signal *RES_AVAILABLE*.

When the results have been transmitted and the memory element updated with the new data, the cycle ends. However, is easy to guess that the very first iteration is slightly different from all the subsequent ones: in the first iteration, in fact, is necessary to activate the write heads *MTJ_W1* and *MTJ_W2* in order to nucleate the skyrmions needed by the FA to perform the computation; these write heads then won't ever be used again, since the two skyrmions nucleated are collected by the bottom tank and kept in the system. At the same time, at the very beginning is necessary to nucleate a skyrmion for the second operand (unless the desired value is equal to 0), while in all the following cycles the corresponding skyrmion could come out from the bottom tank, without the need of activating *MTJ_W3*. For this reason, as soon as the *START* signal provided to the slave FSM is activated, the voltage V_{TANKB} is turned on, and according to the value of the skyrmion detected on the lowest priority output the voltage V_{NOEXT} and *MTJ_W3* are directed, as already explained. All the remaining steps are not different from what happens in the very first iteration.

5.2. Cell 10

The first cell that starts the elaboration as soon as the results of cell 00 are available is cell 10. The structure of this and of the remaining cells is very similar to the one of cell 00, with only some small and localized differences due to the different number of inputs and of outputs to be provided; the same applies to the corresponding slave FSM: the differences are few and well localized. For this reason, this section and the following ones will focus just on the relevant differences, without repeating the sequence of steps to be performed from the start to the end of the computation in order to correctly coordinate the datapath components.

Figure 5.25 shows the datapath of cell 10, while the slave FSM is divided in figure 5.26 and 5.27. The VHDL describing it, as usual, can be found in appendix C, together with the VHDL of the remaining cells.

As already discussed in this chapter, the only difference between cell 10 and cell 00 lays in the number of possible inputs available to be chosen as operand 2, that is, in the number of inputs to multiplexer *5* in figure 5.1: while cell 00 can take as second operand only a value coming from the external world, cell 10 can choose as operand also the result of cell 00. This is the only meaningful difference in the whole cell. Knowing this, is easy to detect where the main changes have been applied to the datapath of cell 00: as shown in figure 5.25, the result provided by cell 00 and put into movement by *CURRENT_Vmoveres* (switched on by the FSM of cell 00) goes through the element *CROSS3* and arrives up to the *GND* contact right before

DEV3.

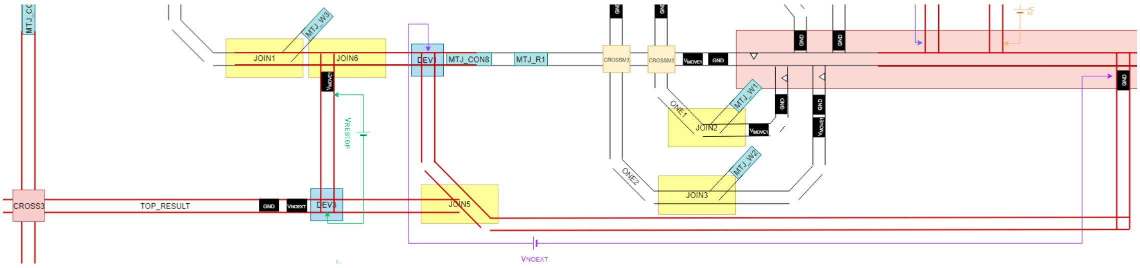


Figure 5.17. Structure implementing multiplexer *5* in figure 5.1.

This result will be available before the slave FSM of cell 10 receives the *START* from the master FSM: the slave FSMs of cells 01, 20 and 11, in fact, receive their very first start only when the cell that has to produce their inputs has already notified the master FSM that the result are available to be consumed, through the activation of the signal *RES_xx_AVAILABLE*, where *xx* is the label of the cell. So, the slave FSM of cell 10 will receive the very first start only after the signal *RES_00_-AVAILABLE* has been activated. At the same time, as already said, the slave FSM of cell 00 remains frozen until the signal *READY_FOR_NEWDATA_RIGHT_00* is activated, as already explained before. This signal is activated only when all the cells receiving the results of cell 00, in this case, have collected them and are ready to accept new results. For this reason is vital that the results are collected as soon as possible, so as not to slow down the overall work of the memory array. This is why, in the case of cell 10 and more in general of all the cells different from 00, the very first action that is performed is the activation of V_{NOEXT} for the amount of time needed to place each input skyrmion inside the corresponding *DEV* component: in the case of cell 10, the only input skyrmion is labelled as *TOP_RESULT*, and activating V_{NOEXT} for the proper amount of time it will enter the component *DEV3*. Doing so, the input of the cell is now free from skyrmions and new data can be accepted: this is notified through the activation of *READY_FOR_NEW_DATA*. Of course, in order to activate the signal *READY_FOR_NEWDATA_RIGHT_00*, the same procedure must be done also by cell 01, which receives the results of cell 00 as well. Since this procedure is done after the activation of the *START* signal, and since cell 01 cannot start until the results of cell 10 are available as well, cell 00 will have to remain frozen for a little more. More details concerning the mechanism that allows each slave FSM to know when to start the elaboration will be given in section 5.3.

At this point, focusing on the new multiplexer inside cell 10, there are two possibilities available: either the desired data for the second operand is exactly the result just imported, or the second operand must be nucleated according to a value provided externally, that is, activating *MTJ_W3* according to the desired value. In the former case, the skyrmion that is inside *DEV3* can be pushed towards the main track by activating V_{RESTOP}; at the same time, V_{START} is applied as well, to allow the skyrmion inside the memory element to cross the notch, so that in the next state the enable skyrmions needed by the FA can be nucleated and V_{MOVE1} can be applied, carrying all the data right at the input of the computational blocks. If instead the second operand must come from outside, the skyrmion inside *DEV3* is moved towards the input of the bottom tank by applying the voltage V_{NOEXT}, the same that is used also from the second iteration on to avoid that an undesired skyrmion that came out from the bottom tank can take the place of the second operand when a value 0 is required. At the same time V_{START} is applied. In the following state, as in the other case, the two enable skyrmions needed by the FA are nucleated and V_{MOVE1} is applied; according to the value desired for the external skyrmion, *MTJ_W3* may or may not be activated. Of course, it could be possible to detect the value of the result of cell 00 that has been rejected and maybe reuse right away that same skyrmion instead of activating *MTJ_W3*: this, however, would contribute to complicating the FSM, and for this reason this option has been avoided.

From this moment to the end of the computation the data flow is exactly the same as before: this can be verified by inspecting the FSM shown in figures 5.26 and 5.27. The only additional difference to the datapath is due to the need of providing the result not to two, but to three different cells (cell 20, cell 11 and cell 01): this is why the duplication element *x2_6* has been added at the output port.

In this cell and in all the remaining ones, as already said, a new cycle of computations can start only when new results are provided as input. In this particular case, new results from cell 00 must be available at the input of the cell. The mechanism that allows the cell to decide whether to start or not, according to the signals coming from the master FSM and to the status signals coming from the datapath, will be discussed more in detail in section 5.3.

When a new *START* is detected, the FSM allows the skyrmion just stored in the memory element to cross the notch by applying V_{START}, it applies V_{TANKB} to allow at most three skyrmions to go out from the bottom tank, it applies V_{NOEXT} to import the new data provided by cell 00, and it activates the signal *READY_FOR_-NEW_DATA* to notify that all the data have been imported. Then, according to the desired source for the second operand, either V_{RESTOP} or V_{NOEXT} is activated. If the result of cell 00 has been chosen, V_{MOVE1} is then applied until a possible skyrmion coming out from the lowest priority output of the bottom tank can reach the inside of component *DEV1*: then, according to the output provided by the read head *MTJ_R1*, V_{NOEXT} may or may not be turned on, in order to flush away any undesired skyrmion. If, on the contrary, the second operand was chosen to be an external data, after applying V_{NOEXT} to move away the skyrmion inside *DEV3* and V_{MOVE1} to move a possible skyrmion coming out from the bottom tank until the inside of *DEV1*, the same sequence of steps already seen for cell 00 is performed: if a skyrmion is present and the desired value is 0, the skyrmion is put back inside the tank; if the value of the skyrmion (either 0 or 1) corresponds to the desired one, V_{MOVE1} will transport it, together with all the other data, towards the input of the computational blocks; if, finally, there is no skyrmion where it should be, *MTJ_W3* is activated in order to nucleate it. In any case, in the end V_{MOVE1} will allow the skyrmion that has survived the selection process to reach the input of the FA and of the *AND/OR* gate, after being duplicated by *x2_3*. From this moment on, all the steps remain the same with respect to the first cycle.

5.3. Master FSM and cell 01

In this section the focus will be essentially on the signals that allow the slave FSMs to correctly synchronize their behaviour without loading of work the master FSM, which must be fast and reactive to the interrupt requests of all the cells inside the array. The activities of the master FSM are simply a consequence of the requests sent by the slave FSMs and, having read the previous sections of this chapter, shouldn't be hard to be understood.

The structure of the master FSM is reported in figures 5.34, 5.35, 5.36, 5.37 and 5.38. It is composed essentially by a loop and by a number of interrupt service routines (ISRs) equal to the number of cells present in the array. After the reset state, the first operation done by the FSM is the initialization of the memory array, activating the write heads at the beginning of each bitline and the voltage that allows the skyrmion to reach the input of the cell; in the state *S2*, applying the voltage V_{STORE}, each memory element is initialized. Only at this point the elaboration can start: for this reason, in state *S3* cell 00 receives the *START_00* signal. From this point on, the master FSM will continue to loop around a state of idle, ready to answer to any interrupt request coming from the cells of the array. Each interrupt signal is determined by the *OR* combinations of the signals that are then investigates inside the corresponding ISR, as shown in figure 5.18.

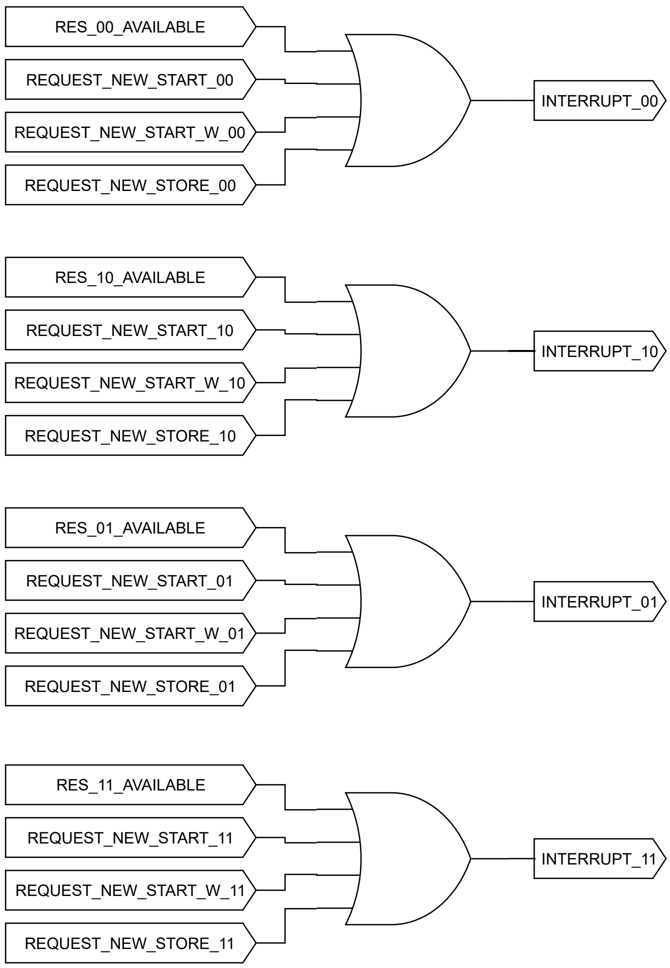


Figure 5.18. Generation of the *INTERRUPT_xx* signals from the status signals activated by each slave FSM.

Figure 5.19 shows, for the first four cells, the signals involved in deciding when the slave FSM can start a new elaboration and in notifying the other slave FSMs when the cell is available for receiving new results.

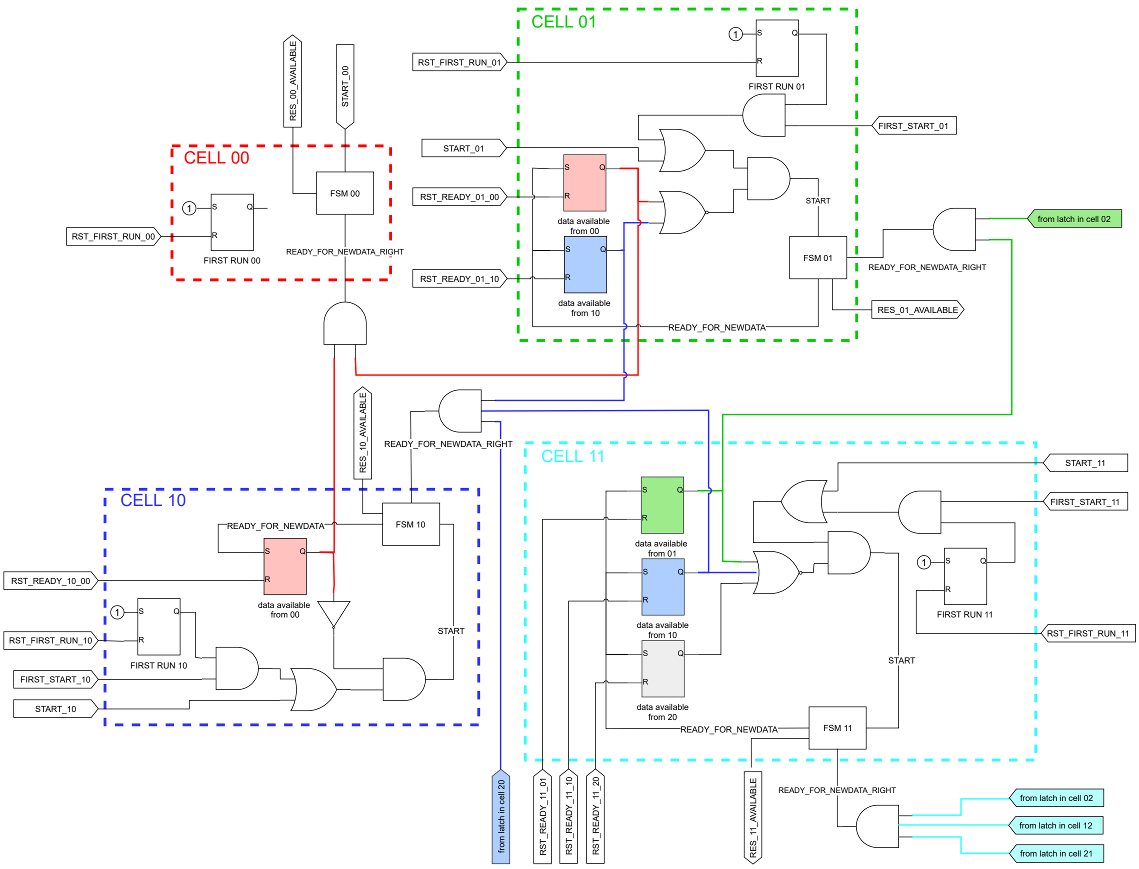


Figure 5.19. Sketch of the structure implemented for each cell in order to correctly activate the *START* signal received in input by each slave FSM. This structure allows also the transfer of the results from each cell towards its destination cells. The coloured latches are used to report whether new data are available; the colour of each latch corresponds to the colour that identifies the cell that has to send the data (the source cell): the red latches, for example, all correspond to cell 00; in particular, the red latch inside cell 01 reports whether the data coming from cell 00 are available to be used by cell 01.

At the very beginning of the elaboration, when the master FSM is in the reset state, all the latches inside each cell (shown in figure 5.19) are initialized to 1. So, when cell 00 receives its very first start from the master FSM, the signal *READY_-FOR_NEW_DATA_RIGHT* is equal to 1, signifying that both cell 10 and cell 01 are ready to accept the results of cell 00. As soon as cell 00 transfers the results to cell 10 and 01, its slave FSM activates the signal *RES_AVAILABLE* towards the master FSM: as a consequence, the master FSM enters the interrupt service routine (ISR) of cell 00, and seeing that *RES_00_AVAILABLE* has been activated it resets the red latches inside both cell 10 and cell 01, meaning that the data from 00 has already been sent and the cells are no more available to accept new results; it also turns on, for a clock period, the signal *FIRST_START_10* (in general, during the state *ISRxx_S1* of each ISR, it is activated the *FIRST_START_xx* of all the cells allocated in the next phase with respect to the cell that activates them, as detailed in figure 5.2); then the master FSM resets the latch *FIRST_RUN* inside cell 00: this latch, which was initialized to 1, will be equal to 0 from now on, signifying that cell 00 has already performed its very first iteration. After this sequence of operations, since all the data have been correctly transferred, cell 00 is free to start a new elaboration, so it requests a new start to the master FSM through the activation of either *REQUEST_NEW_STORE*, *REQUEST_NEW_START_W* or *REQUEST_NEW_START*: after all the steps required for correctly storing the new data inside the memory element (steps that have already been detailed in section 5.1), the master FSM will activate the signal *START_00*, allowing the cell to start the second iteration.

The latch *FIRST_RUN* inside cell 10 is still equal to 1: the *AND* combination of this signal with the signal *FIRST_START_10*, activated by the FSM for a clock cycle, is equal to 1; at the same time, the red latch had just been reset by the master FSM, signifying that the data sent by cell 00 and needed to start the elaboration are available: as a result, the signal *START* turns on and the first iteration of cell 10 starts. As soon as the cell accepts the data from 00 by making the skyrmion enter the element *DEV3*, the FSM 10 turns on for a clock cycle the signal *READY_-FOR_NEW_DATA*: as a consequence, one of the latches that prevent cell 00 to send new data becomes again equal to 1. At this point, however, the red latch inside cell 01 is still equal to 0, so, even if cell 00 had new results ready to be sent, it would have to wait until cell 01 imports the previous ones.

When also cell 10 produces its first results, its FSM investigates around the value of signal *READY_FOR_NEWDATA_RIGHT*: since all the blue latches inside cell 01, cell 11 and cell 20 (which are the destinations for the results of cell 10) are still equal to 1 from the initialization, the cell is able to send the results it has produced.

So, after activating its $V_{MOVECOUT}$ and $V_{MOVERES}$, it turns on the signal *RES_-AVAILABLE* towards the master FSM. This makes the master FSM enter the ISR 10: as a result, the blue latches inside cell 01, cell 11 and cell 20 will be reset to 0, signifying that the cells are no more available to accept new results from cell 10, and at the same time both *FIRST_START_01* and *FIRST_START_20* will be activated: as shown in figure 5.2, in fact, these are the only two cells allowed to work during the third phase. Finally, the master FSM resets the latch *FIRST_RUN* inside cell 10. When the master FSM receives the request of a new start from cell 10, then, after performing all the steps needed to allow the new data to be memorized inside the cell, it grants a new start to the cell by activating the signal *START_10*.

It can be noticed here that, apart from cell 00, which is a very particular case, all the other cells have two different types of *START* signals coming from the master FSM. Let's consider cell 10: the signal *FIRST_START_10* is activated during the state *ISR00_S1* inside the ISR 00, as a consequence of the completed transfer of the new data. This works if the cell has never performed any computation and is waiting to start. If there was a single *START* signal, instead of *FIRST_START_10* and *START_10*, the FSM would receive a new start any time that the cell 00 has produced new results; this shouldn't happen, because each of these FSMs has its individual work flow and may take a different time to produce the results, according also to the control signals it receives from the external. For this reason, after the very first iteration, each slave FSM decides in complete autonomy, together with the master FSM, when to start a new iteration, by activating one of the three signals which request a new start to the master FSM and waiting for the activation of signal *START_xx*. This is why the latch *FIRST_RUN* inside each cell is needed: this latch is used to mask any other activation of *FIRST_START_xx*, which may mess up the behaviour of the cell that receives it.

Let's focus now on the behaviour of cell 01. With respect to cell 10, the differences inside cell 01 are in the number of inputs to multiplexer *9* in figure 5.1, and in the use of the carry provided by cell 00. Up to now, in fact, the input carry to the FA was always set equal to 0, both in cell 00 and in cell 10, and so there was no need for connecting a nanotrack to the input of the FA. Now, instead, the carry coming out from cell 00 must be duplicated, due to the particular structure of the

FA, and provided in input to it.

The datapath of the cell is shown in figure 5.28, while the FSM is described in figures 5.29 and 5.30.

Having already analysed the behaviour of cell 10, there should be no difficulties in understanding how the structure equivalent to multiplexer *9* in figure 5.1 works, so this time the analysis of the first and last steps of the iteration will be focused more on the consequences that the signals that appear in figure 5.18 have on the workflow of the cell.

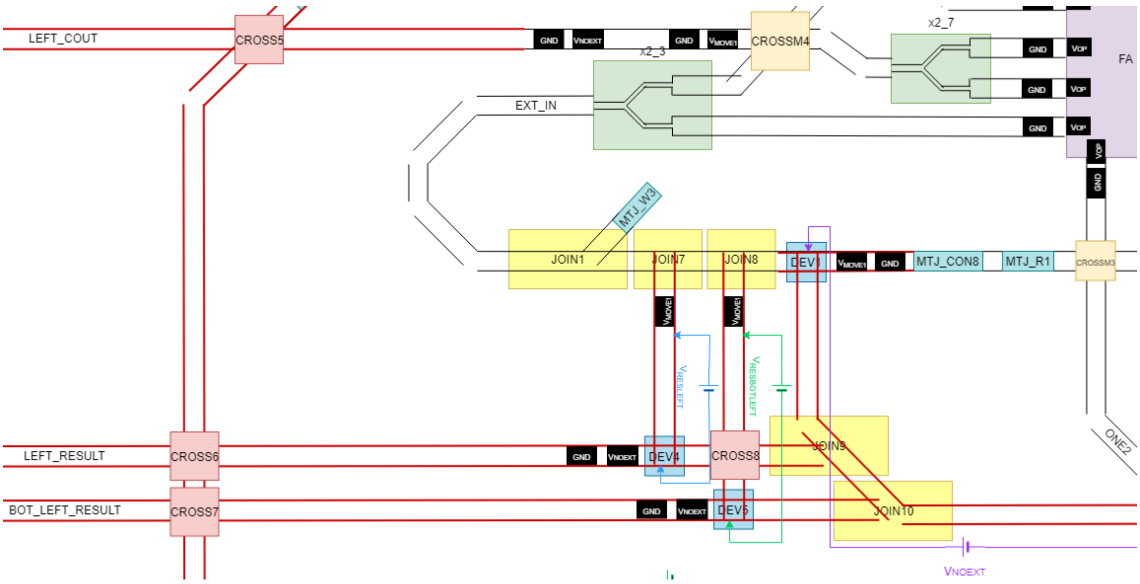


Figure 5.20. Input results to cell 01 and multiplexing structure needed to select one of them.

When cell 01 receives *FIRST_START_01* from the ISR 10, since both the red and blue latch shown in figure 5.19 are equal to 0, meaning that all the data needed are available, it is able to start its very first iteration. Like already explained before, the first operation that is performed is the importation of the data by activating the voltage V_{NOEXT}, so that in the same state also the signal *READY_FOR_NEW_-DATA* is activated, changing to 1 the value stored in the red and in the blue latch: in this way both cell 00 and cell 10 know that they are free to send new data towards cell 01. The activation of V_{NOEXT} makes the skyrmions labelled as *LEFT_RESULT* and *BOT_LEFT_RESULT* (respectively the result of cell 00 and cell 10) enter the components *DEV4* and *DEV5*; at the same time, also the skyrmion labelled as *LEFT_COUT* (coming from cell 00) is imported. Then a selection process almost identical to the one already described for cell 10 is performed: if one of the two results is desired as input for operand 2, either $V_{RESLEFT}$ or $V_{RESBOTLEFT}$ is activated; in the following state, apart from activating V_{MOVE1} and the two writing heads for nucleating the two enable skyrmions needed by the FA, also V_{NOEXT} is activated once more, in order to flush away towards the bottom tank the skyrmion that was not chosen. If instead the desired input for operand 2 is a value coming from the external, the steps are exactly the same as the ones described for cell 10.

Then the behaviour is the same as for the other cells already analysed: the result is computed and placed right at the output of the cell, ready to be exported. Then the cell waits for the activation of *READY_FOR_NEWDATA_RIGHT*: since at the first iteration this signal is equal to 1 from the initialization (the green latches inside cell 11 and cell 02 are already storing a 1), no freezing of the FSM happens; however, in a following iteration these latches may be still equal to 0 (if the destinations cells have already received results, but they still hadn't the time to consume them), so at this point a freezing of the FSM may happen. This is why it is so important to import the results as a first thing, when starting a new iteration.

When *READY_FOR_NEWDATA_RIGHT* becomes equal to 1 the result can be exported: cell 01 activates $V_{MOVERES}$ and $V_{MOVECOUT}$, moving the skyrmions towards their destination, and then activates the signal *RES_AVAILABLE* towards the master FSM, and waits for its acknowledge. The main FSM may in fact be busy in serving some other interrupt request, so cell 01 must maintain active the signal until it receives the acknowledge from the master FSM (this is true also for all the other cells: any activation of the signal *RES_AVAILABLE* freezes the slave FSM until the corresponding acknowledge is activated); apart from activating *ACK_-RES01_AVAILABLE* the master FSM, since the results of cell 01 have just been exported, resets the green latches inside cell 11 and cell 02 and activates the signals *FIRST_START_11* and *FIRST_START_30* (both cells are allocated in the fourth phase of figure 5.2). Now: cell 01 had received the start signal together with cell 20, which is allocated in the third phase as well; however, it may happen that cell 20 is slower in producing its result, due to the particular workflow it has to follow, and as a result the grey latch inside cell 11 may still be equal to 1 from the initialization: this would mean that cell 11 is still available to receive new results, which still have to be send (of course the same applies to the condition where cell 20 is faster than cell 01). The presence of a latch that is still equal to 1 prevents the FSM 11 from receiving a *START* signal equal to 1, and so the activation of *FIRST_SIGNAL_11* by cell 01 (during ISR 01) is lost. However, this doesn't result into an error, and the structure shown in figure 5.18 is correctly working also in this case: let's assume in fact that cell 01 finishes its computation before cell 20. After exporting its results and receiving *ACK_RES01_AVAILABLE* from the master FSM, the FSM 01 will send a request for a new start and go on with the next iteration. Then, finally, sooner or later the results from cell 20 will be available: in their ISR, inside the state *ISRxx_S1*, apart from setting to 0 the corresponding latch inside cell 11, both cell 01 and cell 20 activate the signal *FIRST_START_11*: this means that, if the activation of *FIRST_START_11* by cell 01 was lost due to the grey latch, which was still equal to 1, now that cell 20 has finished that latch will be reset to 0 and a new impulse on *FIRST_START_11* will be provided, finally allowing the FSM 11 to start its first iteration.

So, thanks to the structure shown in figure 5.18, the structure of the master FSM is kept simple and neat, allowing a faster detection and answer to the interrupt requests coming from the cells of the array. If this same control was to be implemented by the master FSM alone, the complexity of the machine would grow very quickly, due to the difficulties in keeping trace, for each cell, of when all the inputs are provided, and to forecast when its outputs may be available.

A final comment deserves the decision block, labelled *ALL_INPUTS_AVAILABLE_xx*, inside the ISR of cell 10, cell 01 and cell 11 (figures 5.36, 5.37 and 5.38). The signal *ALL_INPUTS_AVAILABLE_xx* is exactly the signal produced by the *NOR* combination of all the latches, inside a certain cell, which indicate whether the cell is available for receiving new data (the same signal used as input to the *AND* gate that produces the *START* signal inside each cell). If this signal is equal to 0, it means that there is at least one latch equal to 1, that is, there are still some data that must be received before starting a new elaboration. Placing the decision block based on this signal in that position inside the three ISRs, all the subsequent investigations regarding the remaining status signals can be avoided. Not only: if the ISR wasn't sensible to this signal too, it could happen that the cell continues asking for a new start, which is always granted by the master FSM, keeping it uselessly busy, without being able to actually start the computation, since the latches prevent it from doing so. So, thanks to that decision block, the behaviour of the master FSM receives a not negligible speed up. Actually, it is not only a matter of optimization: the behaviour of the machine would be wrong without that decisional block, because if the signal that the cell keeps activating without being able to actually start is *REQUEST_NEW_START_W*, the master FSM would continue to nucleate new skyrmions at the beginning of the bitline, trying to satisfy the requests of the cell, and this of course would result into an error.

Let's come back to cell 01 now. When the second elaboration starts, the slave FSM allows the skyrmion just stored in the memory element to come out from it, it makes at most three skyrmions go out from the bottom tank, it imports the new data provided by cell 00 and cell 10, activating at the same time *READY_-FOR_NEWDATA*, so that the red and the blue latch are again set to 1, and it activates the voltage $V_{OUTTANK}$, so that the lowest priority skyrmion which may have just come out of the bottom tank stops right before *DEV1*. Then the selection process takes places: either one of the two results provided is chosen, or they are both flushed away towards the bottom tank. Then, activating V_{MOVE1} until the skyrmion that had stopped right before *DEV1* enters it, the same steps performed also for cell 10 take place, in order to decide if that skyrmion can continue towards the computational area together with the other data, or if it has to go back inside the tank. Then the same steps performed also in the first iteration are repeated.

5.4. Cell 11

Having already analysed in detail the previous three cells, almost nothing new is left to be said about the cell 11. The datapath is shown in figure 5.31, while the FSM is divided in figures 5.32 and 5.33. Cell 11 is the most complex out of the four, since multiplexer *14* of figure 5.1 has now four inputs, and multiplexer *13* also is added, in order to allow the choice between two possible input carries, one coming from cell 01 and one from cell 10.

When the cell receives the *FIRST_START_11* signal that actually makes it start (as already discussed in section 5.3), the first action that is performed, apart from taking out the skyrmion stored in the memory element, is to import the new data

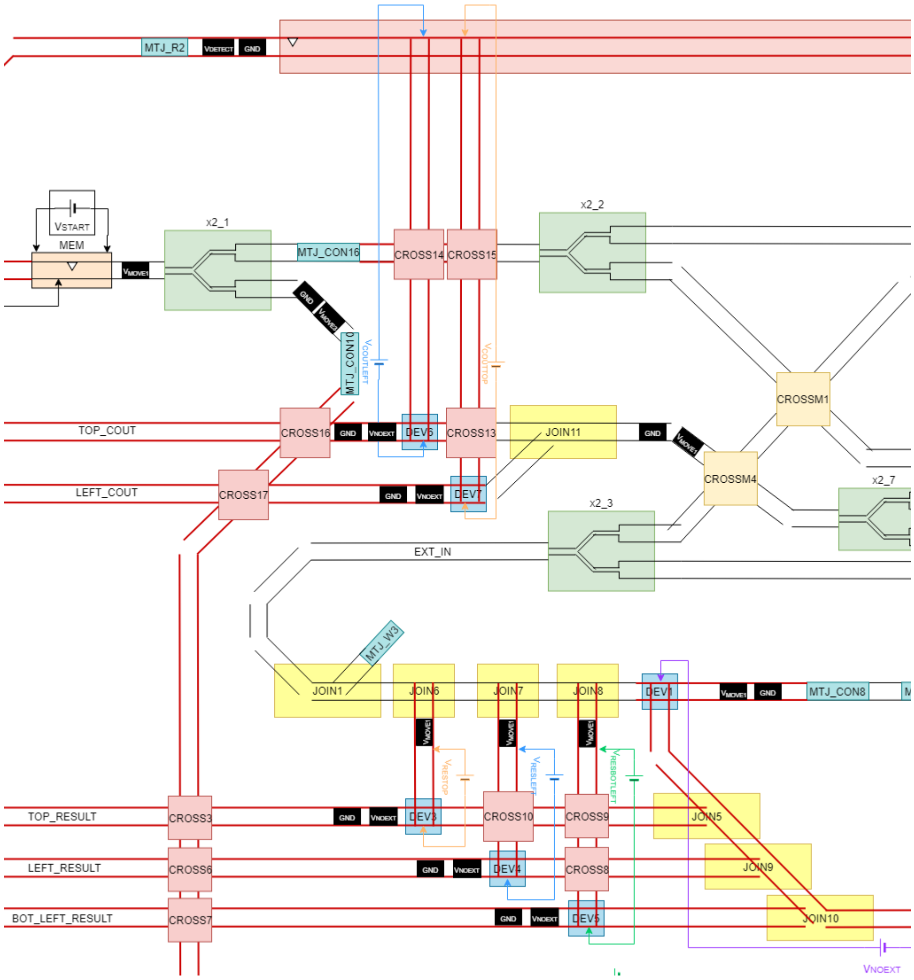


Figure 5.21. Results in input to cell 11 and multiplexing structure needed to select one of them.

applying an impulse on V_{NOEXT}: doing so, the elements *DEV3*, *DEV4*, *DEV5*, *DEV6* and *DEV7* will host at most one skyrmion each, depending on the results produced by the previous cells. At the same time the signal *READY_FOR_NEWDATA* is asserted, to restore back to 1 the value stored in the latches, needed to notify to the other slave FSMs the availability of the cells to accept new data. Then two selection processes take place; first of all, it is decided which skyrmion among *TOP_COUT* and *LEFT_COUT* must be flushed away towards the top tank: this is done by applying either $V_{COUTLEFT}$ or $V_{COUTTOP}$; the skyrmion that has been chosen as input carry to be used in the following computations, instead, will remain for the moment inside the corresponding *DEV* element. Immediately after this first selection, in fact, takes place the choice of the result to be used as second operand: if the top, the left or the bottom left result is chosen, V_{RESTOP}, $V_{RESLEFT}$ or $V_{RESBOTLEFT}$ respectively are turned on. Then, applying both V_{NOEXT} and V_{MOVE1} at the same time, and nucleating the enable skyrmions needed by the FA, the skyrmions that were not chosen are flushed towards the bottom tank and the input carry, the second operand and the first operand can all reach the computational area, together with the enable skyrmions. If instead the value desired for the second operand must come from the external world, first of all the skyrmions hosted in *DEV3*, *DEV4* and *DEV5* are pushed towards the bottom tank by applying V_{NOEXT}; then V_{MOVE1} is turned on, *MTJ_W1* and *MTJ_W2* are activated, and *MTJ_W3* is controlled according to the preferences. The following steps, finally, are the same as always.

When the *START* signal, produced as detailed in figure 5.19, becomes active, the slave FSM imports the new data, activates *READY_FOR_NEWDATA*, reads the skyrmion inside the memory element and takes at most three skyrmions out of the bottom tank. Then two consecutive selection processes take place, and since the following steps are simply the adaptation of the corresponding states inside the FSMs already analysed to the selection process just described, the remaining states won't be explained here.

5.5. VHDL code

To verify the functioning of the memory array, together with the FSMs that have been discussed until now, a VHDL behavioural description of the memory array has been implemented. In writing the code, the focus has been mainly on reproducing in the simplest way possible the behaviour of each of the components used inside the cells: the aim was in fact to verify if the timing of a FSM machine designed while thinking about the physical movement of skyrmions was correct. To do so, each component has to read its inputs and produce its outputs only when the equivalent physical version of that same component would able to do so when controlled by that same FSM, and not in any other instant of time. This implies that most of the work has been focused in controlling the instant of switch of inputs and outputs, and not in how the single component is described internally: the behavioural description that has been adopted is in fact at a very high level. The main limitation in this analysis around the switching time of each signal is the assumption that, as soon as the component is enabled, the output switches instantly: this means that all the components that have been described from a behavioural point of view have a null delay. This assumption, however, should have an effect only on the length of the clock period needed to make the FSMs transition from one state to the other, due to the cascade of elements with non-zero delay, and it shouldn't affect the functionality of the structure, thanks to the use of enable signals for each component.

Each component, in fact, is sensitive to the changes on its inputs in any instant of time, independently from the particular voltage that is turned on. Each of them, however, receives as input the current that flows inside the environment into which that particular component is inserted: for example, the *MTJ_R* components can receive only one current value, while the *DEV* elements receive always two different current values, which become different from zero according to the particular path that the skyrmion must follow. These current signals are produced by the voltage source components and are different from zero only when the particular voltage source which produces them is enabled by the FSM. For this reason these current signals, which are not needed in a high level behavioural description of the components, are used as enable signals. The outputs of a particular component, then, are allowed to change only when the current that the component receives is different from zero: if this is not the case, all outputs will be equal to 0, independently from what happens on the inputs. This is what would happen in a physical realization of these components: even if a skyrmion is present, if the current that moves it is null the outputs of the components will remain 0.

In the following list is offered, for each component, a brief description of its behaviour. The code that implements each of them can be found in appendix C.

Voltage_genL If the control signal in input is equal to 1, the output *CURRENT* becomes equal to *CURRENT_LOW*; in any other case *CURRENT* is equal to 0.

Voltage_genH If the control signal in input is equal to 1, the output *CURRENT* becomes equal to *CURRENT_HIGH*; in any other case *CURRENT* is equal to 0.

Vclock_gen If the control signal in input is equal to 1, the output *CURRENT* copies the input signal *CURRENTclk*, which is the same signal provided as clock signal to the FSMs; in any other case *CURRENT* is equal to 0.

MTJ_R If a skyrmion has been detected (no matter the value of the current in input) and if the input current becomes different from 0, the output experiences an impulse; in any other case the output remains fixed to 0.

MTJ_W If the input *CTRL* is becomes active, the output experiences a pulse (that is, a skyrmion is nucleated); in any other case the output remains fixed to 0.

MTJ_CONV It is composed by a *MTJ_R* and by a *MTJ_W*. The *MTJ_R* detects the presence of a skyrmion on the input of the component and, when the current is on, notifies it by enabling an internal signal. The *MTJ_W* receives this internal signal as control input and nucleates the skyrmion accordingly.

CROSS with Magnus force If a skyrmion has been detected on input *A* (no matter the current applied) and the current related to that input is turned on, the output *A'* experiences an impulse; in any other case it remains fixed to 0. Similarly, if a skyrmion has been detected on input *B* (no matter the current applied) and the current related to that input is turned on, the output *B'* experiences an impulse; in any other case it remains fixed to 0.

CROSS without Magnus force The VHDL description is exactly the one of component *CROSS with Magnus force*.

Duplication element If a skyrmion is detected on the input (no matter the current applied), and if the input current is different from 0, both outputs experience an impulse (the input skyrmion is duplicated); in any other case both outputs remain fixed to 0.

Merging element If one or two skyrmions are detected on input (no matter the current applied) and if the input current is different from 0, the output experiences an impulse (a single skyrmion is ejected); in any other case the output remains fixed to 0.

Deviation element A skyrmion can be detected on the input no matter the current applied, and if no current is flowing both outputs are equal to 0. If the current on the main track, called *CURRENT*, assumes a value different from 0, the outputs are enabled, but nothing happens yet (the skyrmion reaches the centre of the structure). If *CURRENT* is applied again, the main output, called *OUT_SK*, experiences an impulse; if instead is the other input current, called *CURRENTDEV*, to become different from 0, it is the secondary output, called *OUT_SK_DEV*, to experience an impulse. In any other case the two outputs remain fixed to 0.

Bottom tank A skyrmion can be detected on any of the seven inputs of the component no matter the current applied. When the input current assumes the value *CURRENT_LOW*, depending on the number of skyrmions present inside the structure, one or more of the three outputs from the core of the structure experience an impulse: this is done by inducing an impulse on three internal signals. These three internal signals are the input of three notches, whose outputs are the outputs of the component itself. The notches receive the same current that is provided as input to the component.

Top tank A skyrmion can be detected on any of the three inputs of the component no matter the current applied. When the input current assumes the value *CURRENT_LOW*, if at least one skyrmion is present inside the structure, the internal signal which represents the output from the core of the structure experiences an impulse. This internal signal is the input of a notch, whose output is the output of the component itself.

Top tank (only cell 11) The description is the same as for *Top tank*, with the only difference in the number of inputs, here equal to five.

Results multiplexer A skyrmion can be detected on any of the three inputs no matter the current applied. When the current related to one of the three selection inputs (*CURRENT_V1*, *CURRENT_V2* or *CURRENT_V3*) becomes different from 0, according to the value of the skyrmion detected on the input chosen the main output from the multiplexer may or may not experience an impulse. Then, according to the number of skyrmions to be rejected, the secondary outputs (*V1OUT*, *V3OUT*, *V2OUTt* and *V2OUTb*) may experience one or more consecutive impulses.

Result/Stored element multiplexer A skyrmion can be detected on any of the two inputs no matter the current applied. When the current related to one of the two selection inputs (*CURRENT_Vst* or *CURRENT_Vres*) becomes different from 0, according to the value of the skyrmion detected on the input chosen the main output from the multiplexer may or may not experience an impulse. Then, according to the value of the skyrmion to be rejected, one of the two the secondary outputs (either *VstOUT* or *VresOUT*) may experience one impulse.

SRlatch It is a standard SR latch: when *RST* is active the output is reset to 0, and when an edge is detected on *SET* the output switches to 1.

SRlatch_H It is exactly like *SRlatch*, with the difference that it is initialized to 1, and only *RST* can reset it to 0.

Cell_input A skyrmion can be detected on any of the two inputs no matter the current applied. When the current directed along the vertical direction (*CURRENT_T*) becomes different from 0, an internal variable is set. When the horizontal current (*CURRENT_L*) is turned on, if a skyrmion was detected on either of the two inputs, the right output experiences an impulse. Some internal variables are used to deal with the possibility that a skyrmion is detected on both outputs, but maybe only the horizontal current is applied.

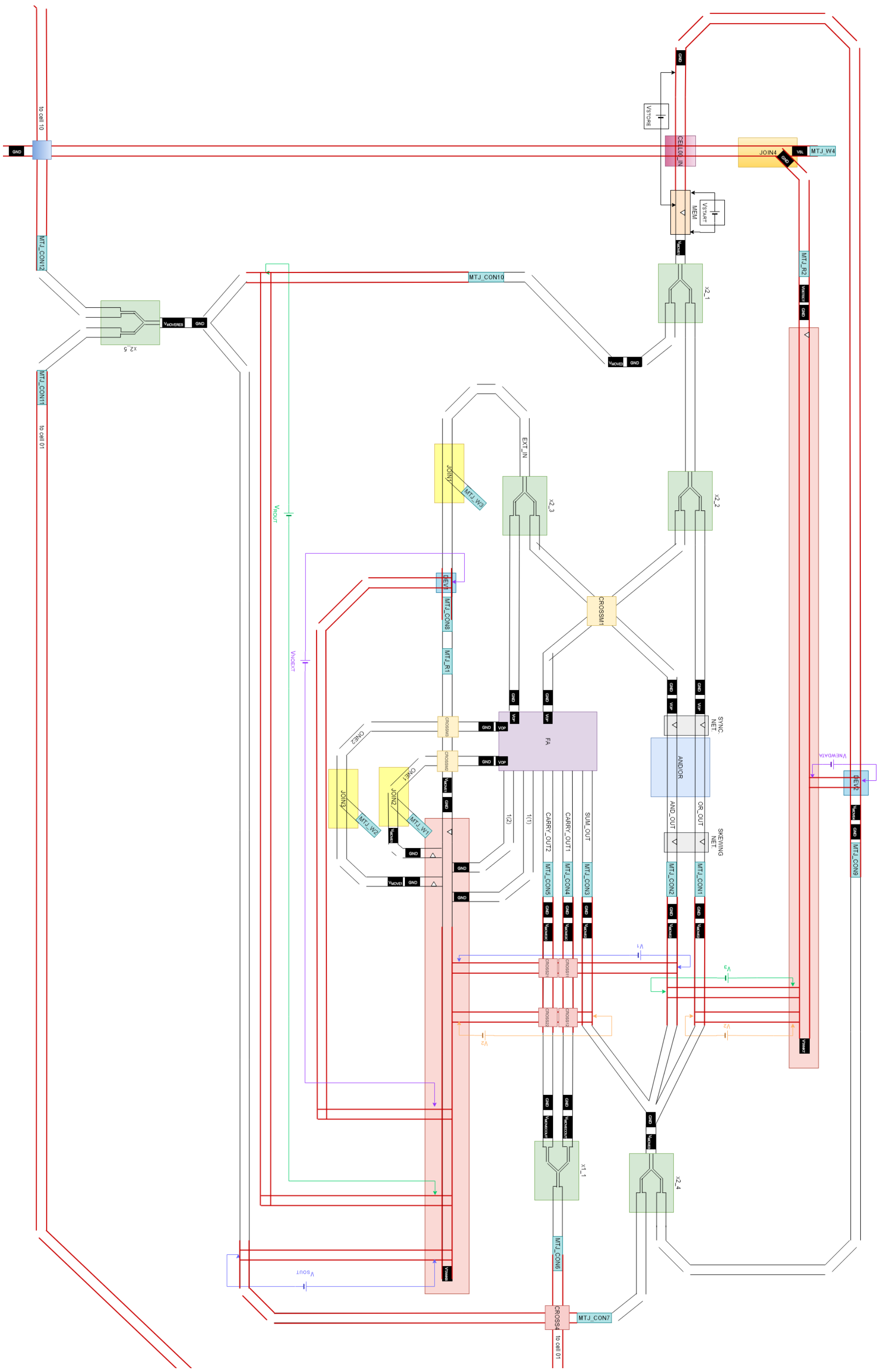


Figure 5.22. Cell 00 datapath

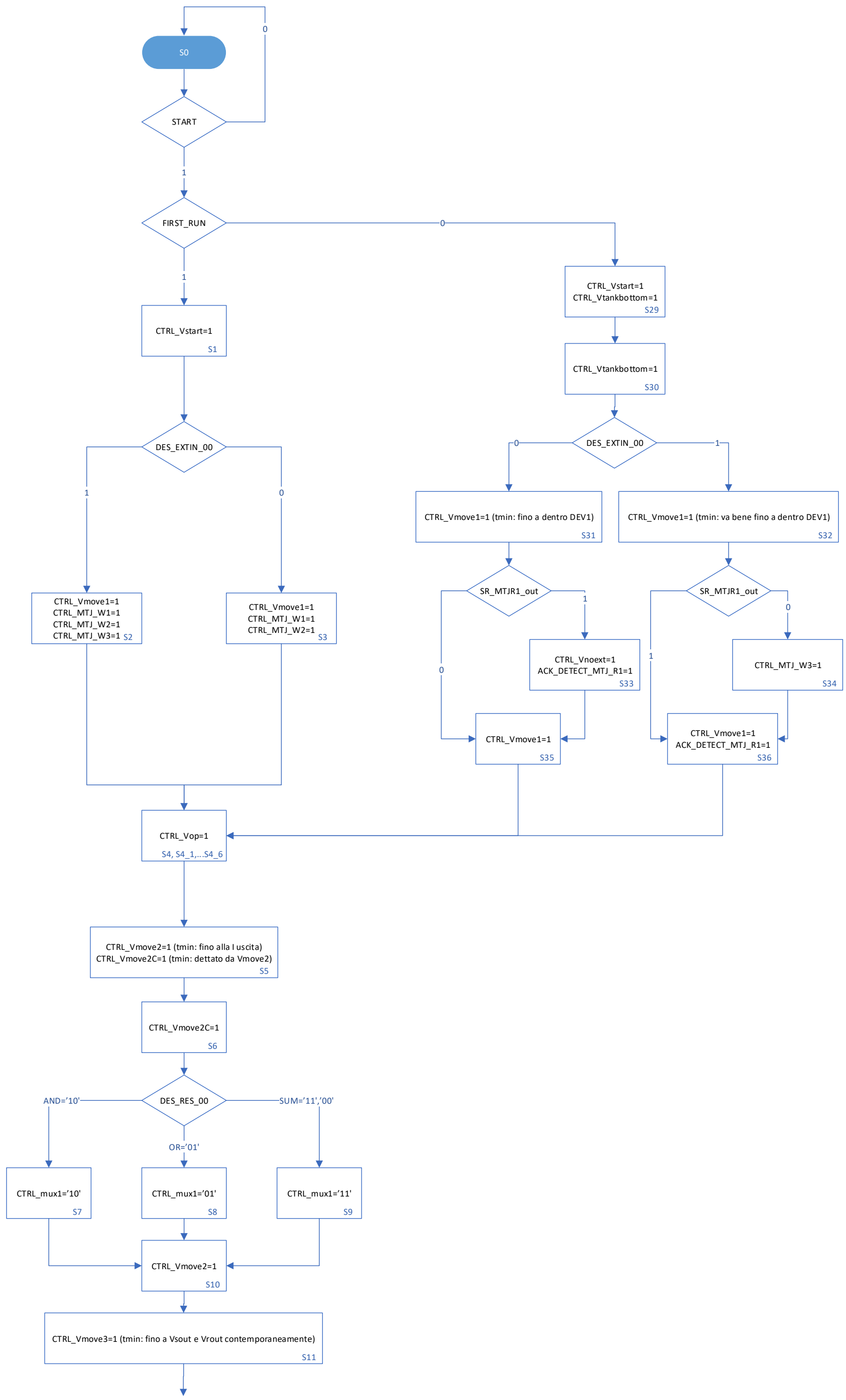


Figure 5.23. Cell 00 FSM, top half

Figure 5.24. Cell 00 FSM, bottom half

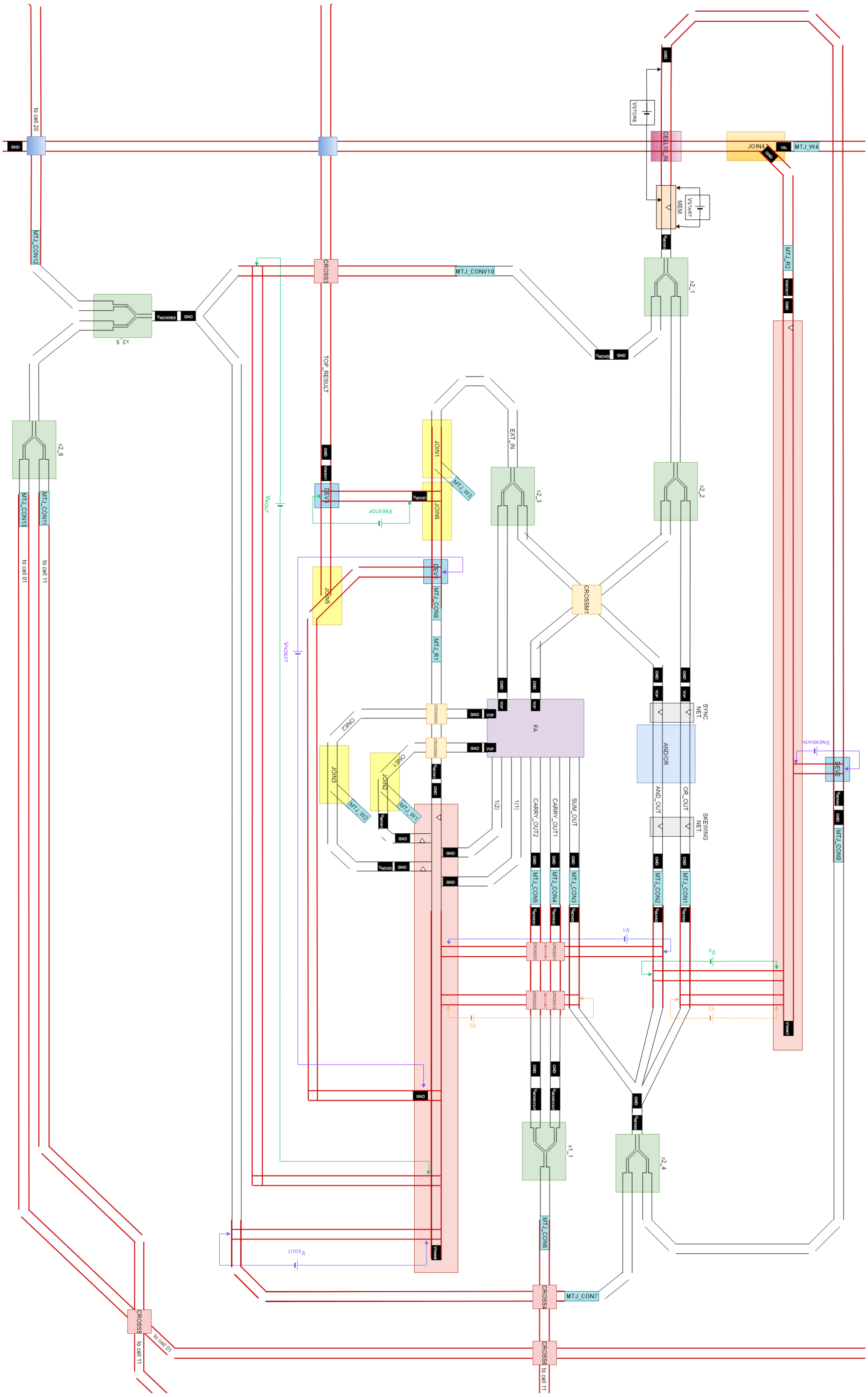


Figure 5.25. Cell 10 datapath

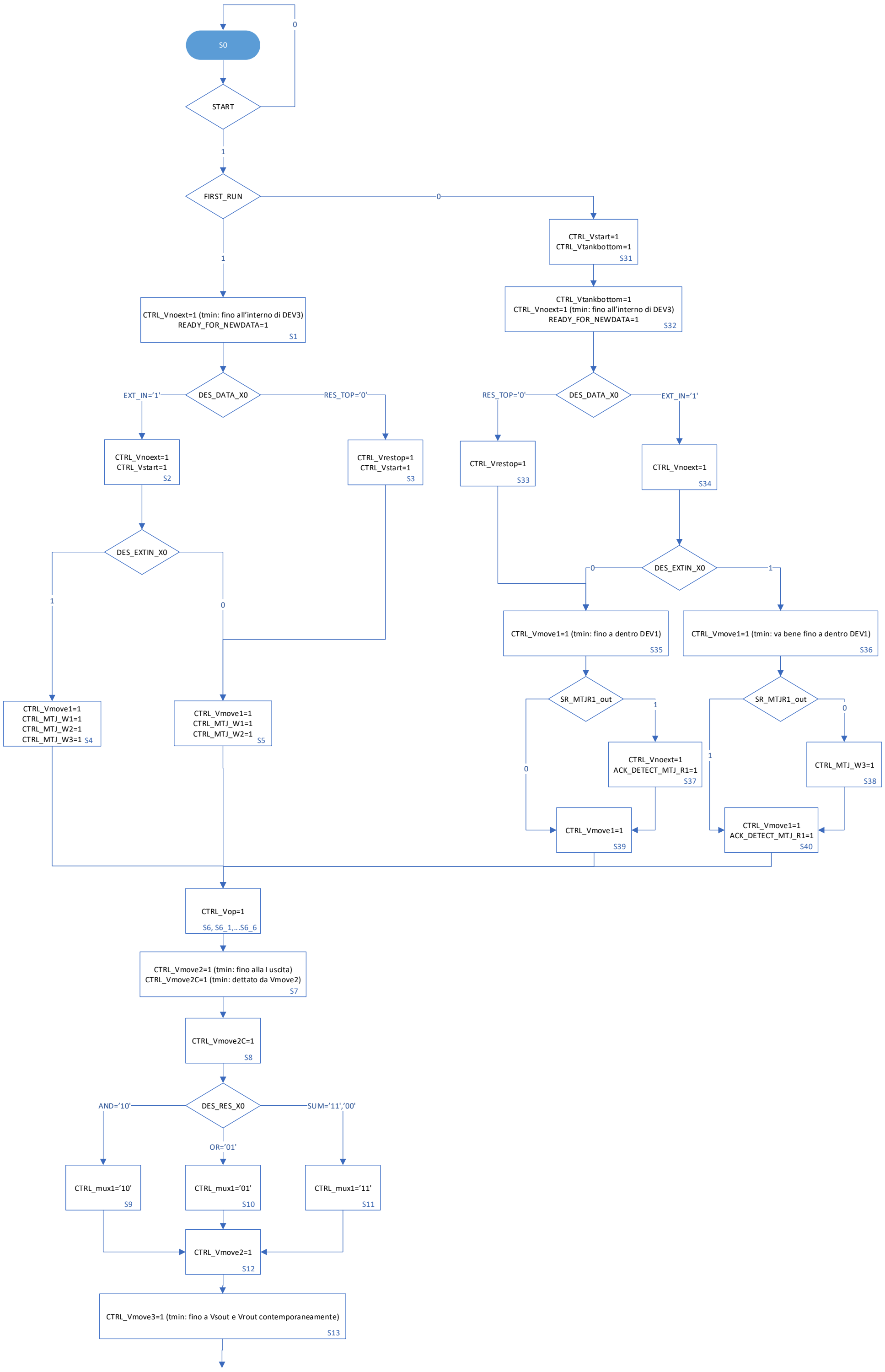


Figure 5.26. Cell 10 FSM, top half

Figure 5.27. Cell 10 FSM, bottom half

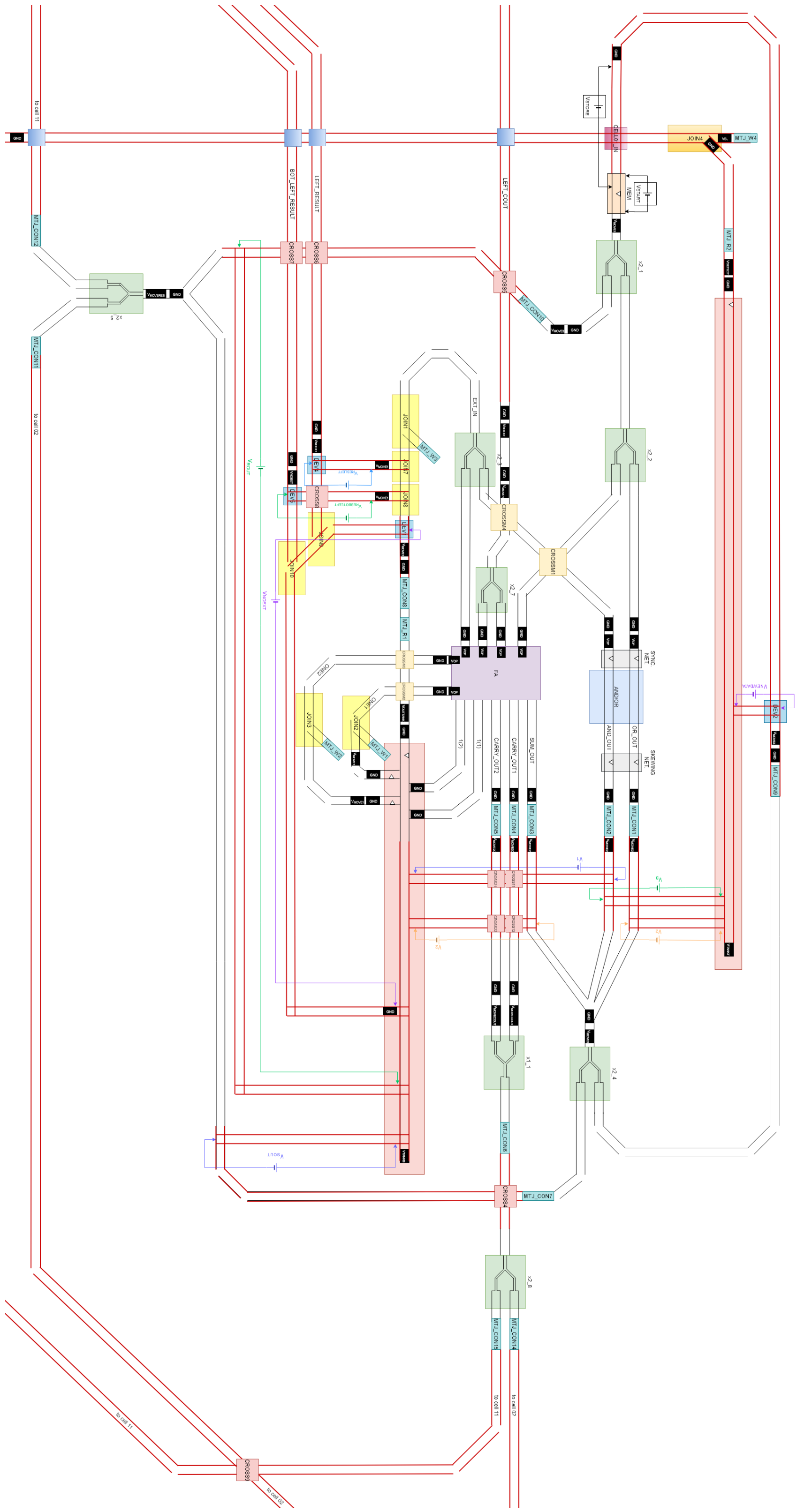


Figure 5.28. Cell 01 datapath

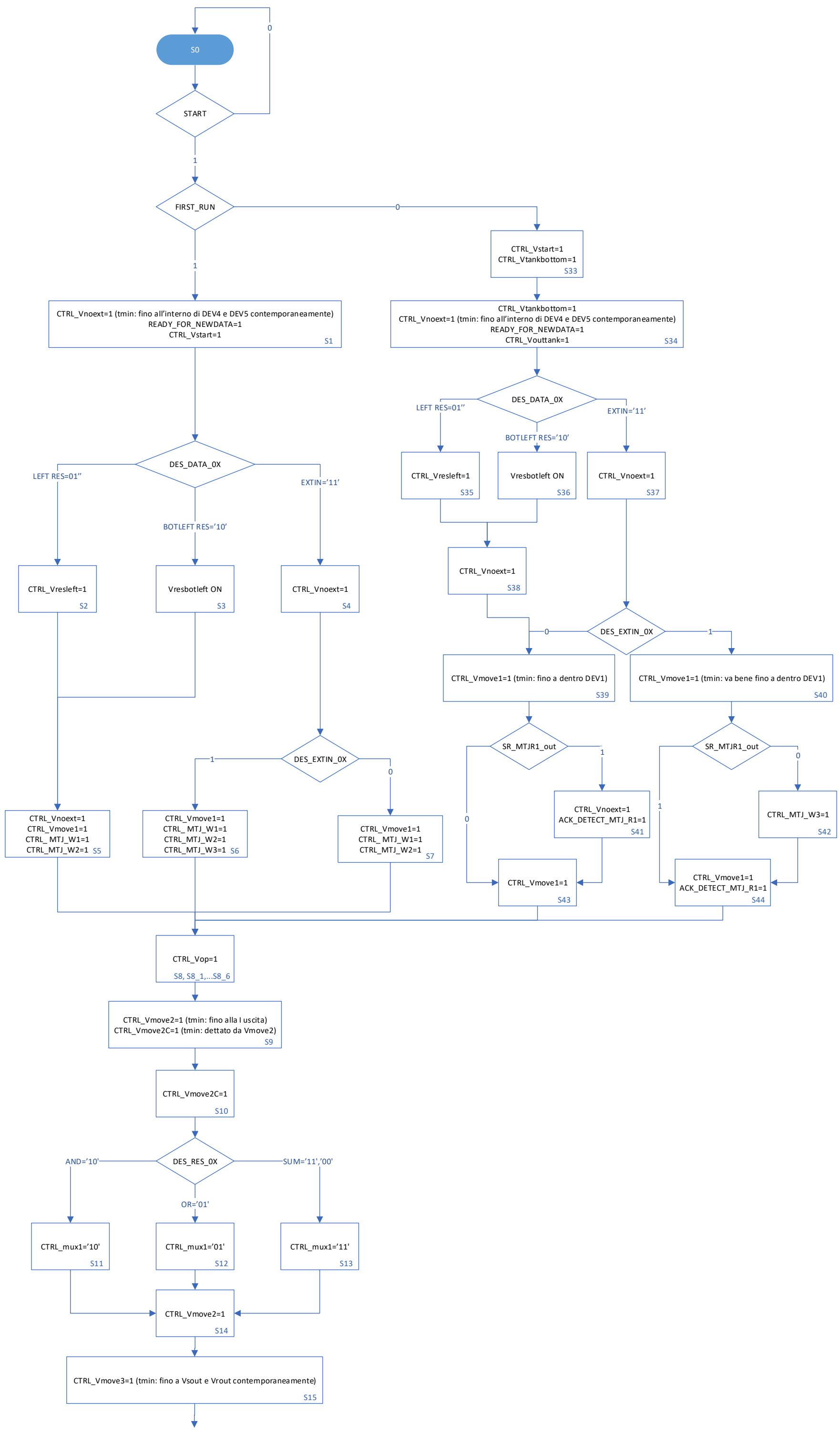


Figure 5.29. Cell 01 FSM, top half

Figure 5.30. Cell 01 FSM, bottom half

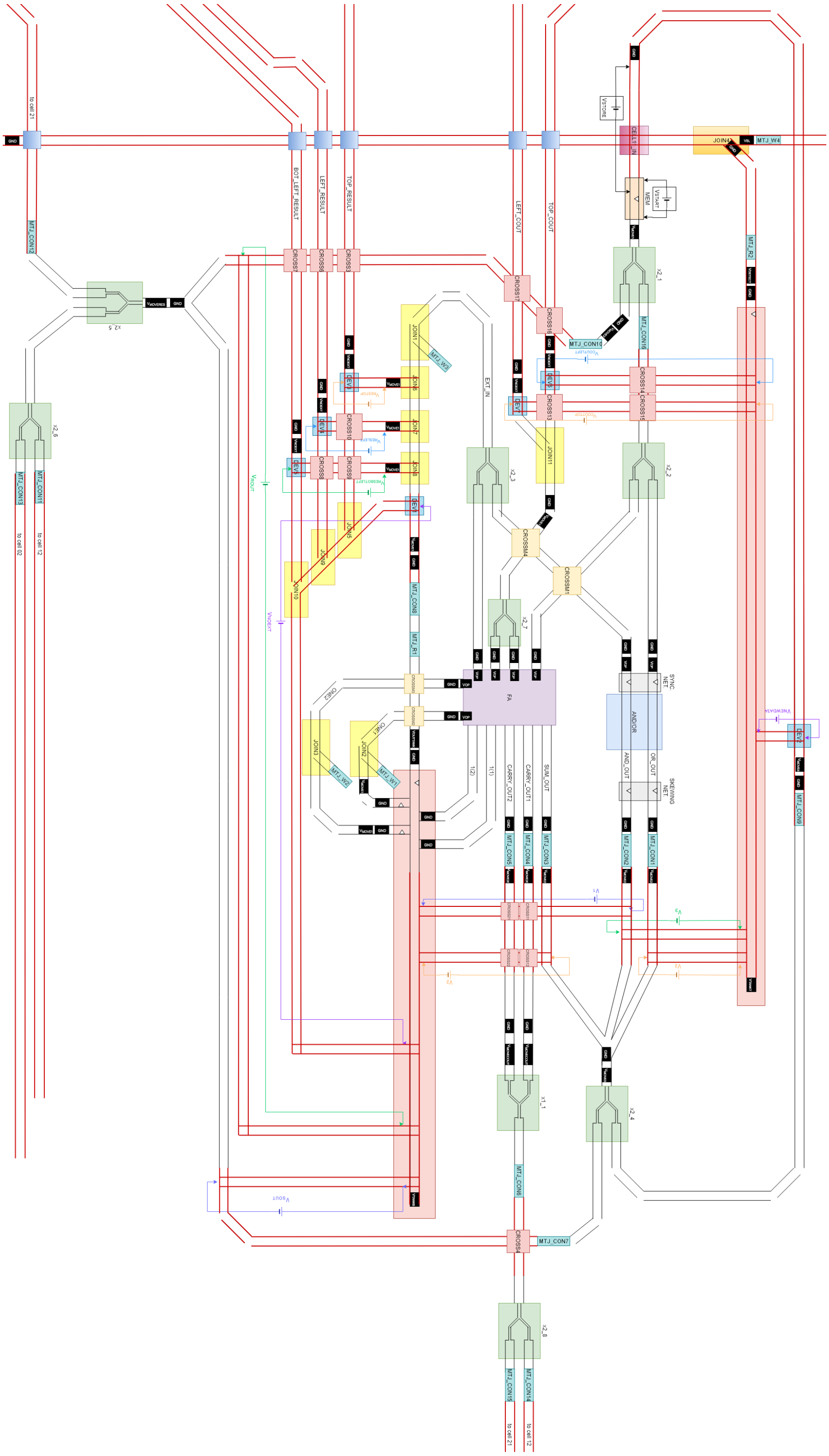


Figure 5.31. Cell 11 datapath

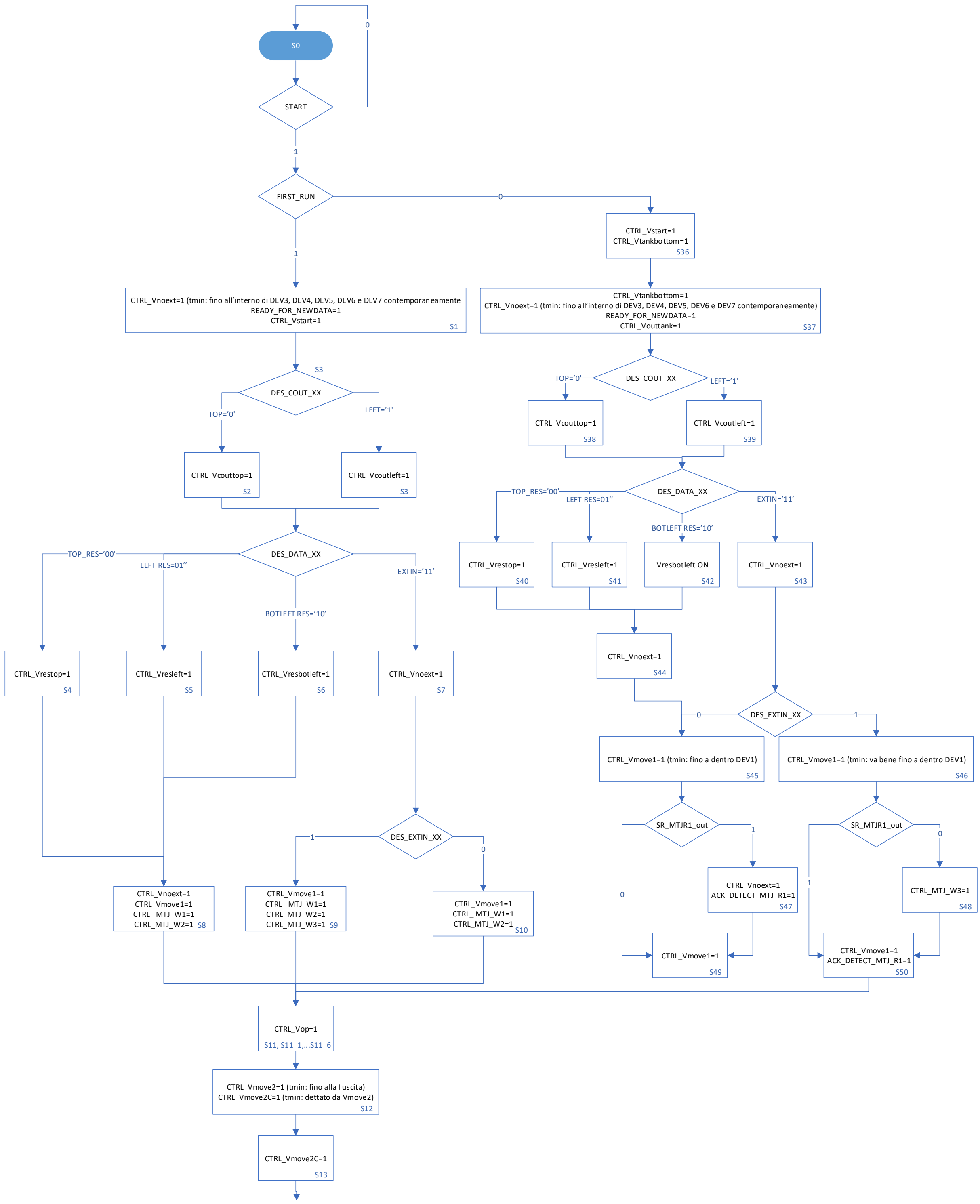


Figure 5.32. Cell 11 FSM, top half

Figure 5.33. Cell 11 FSM, bottom half

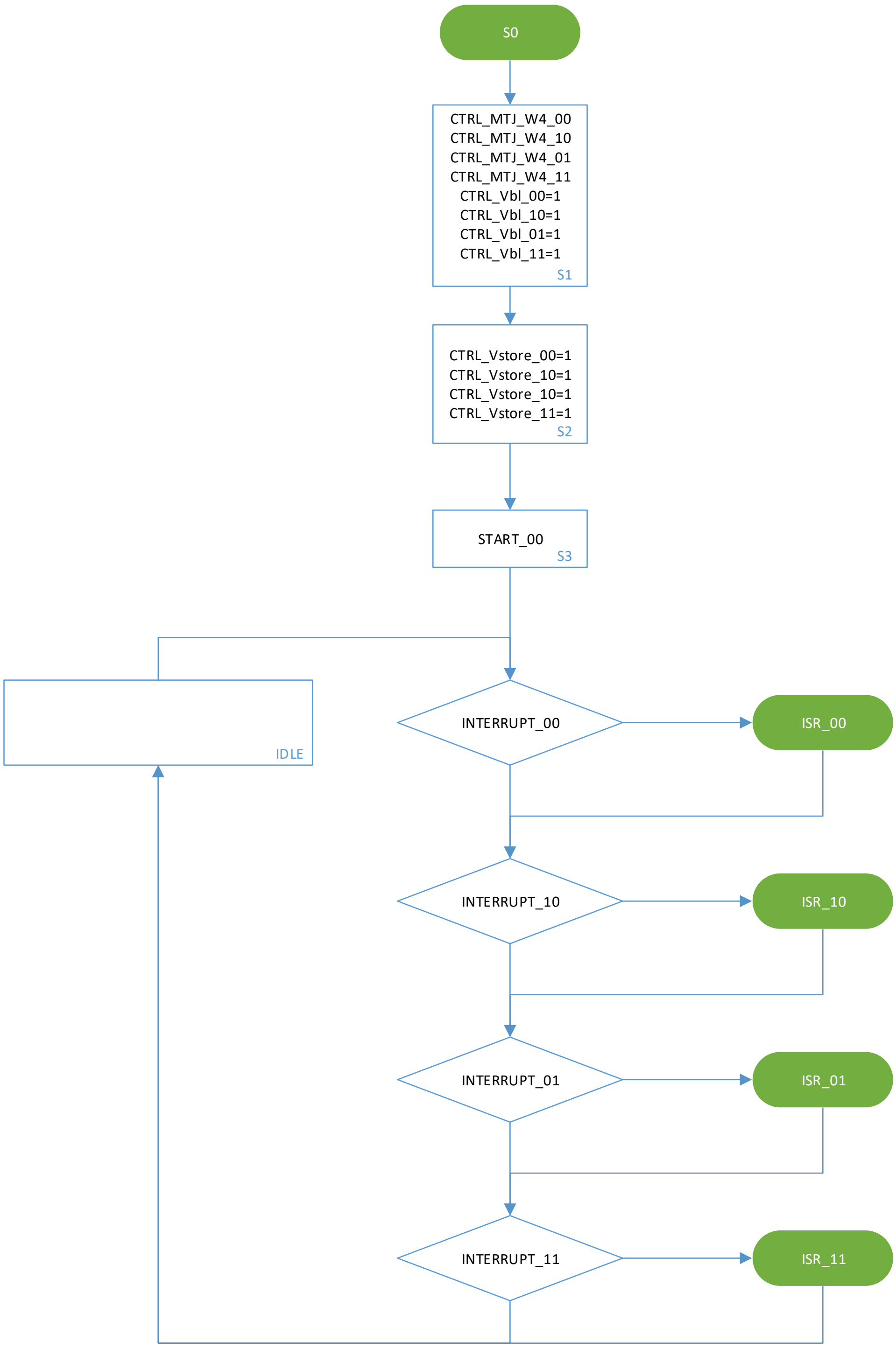


Figure 5.34. Master FSM, idle loop

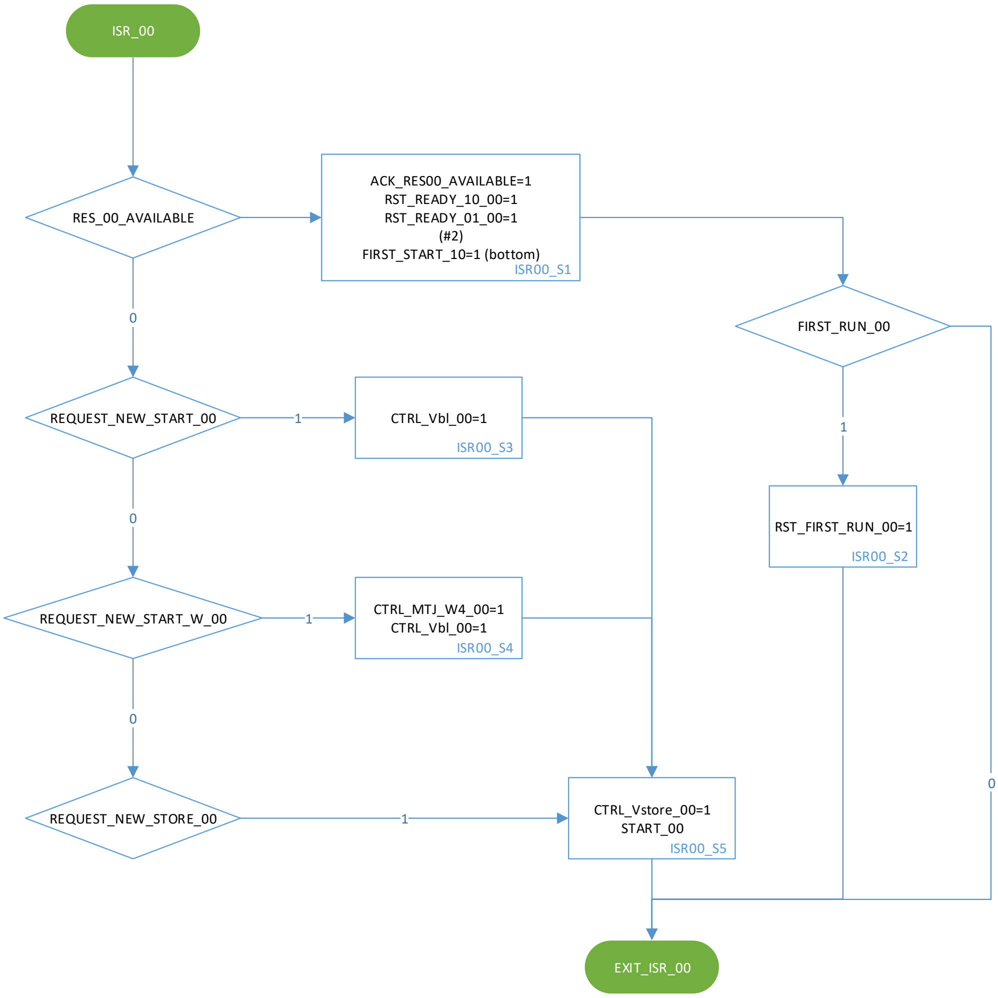


Figure 5.35. Master FSM, ISR 00

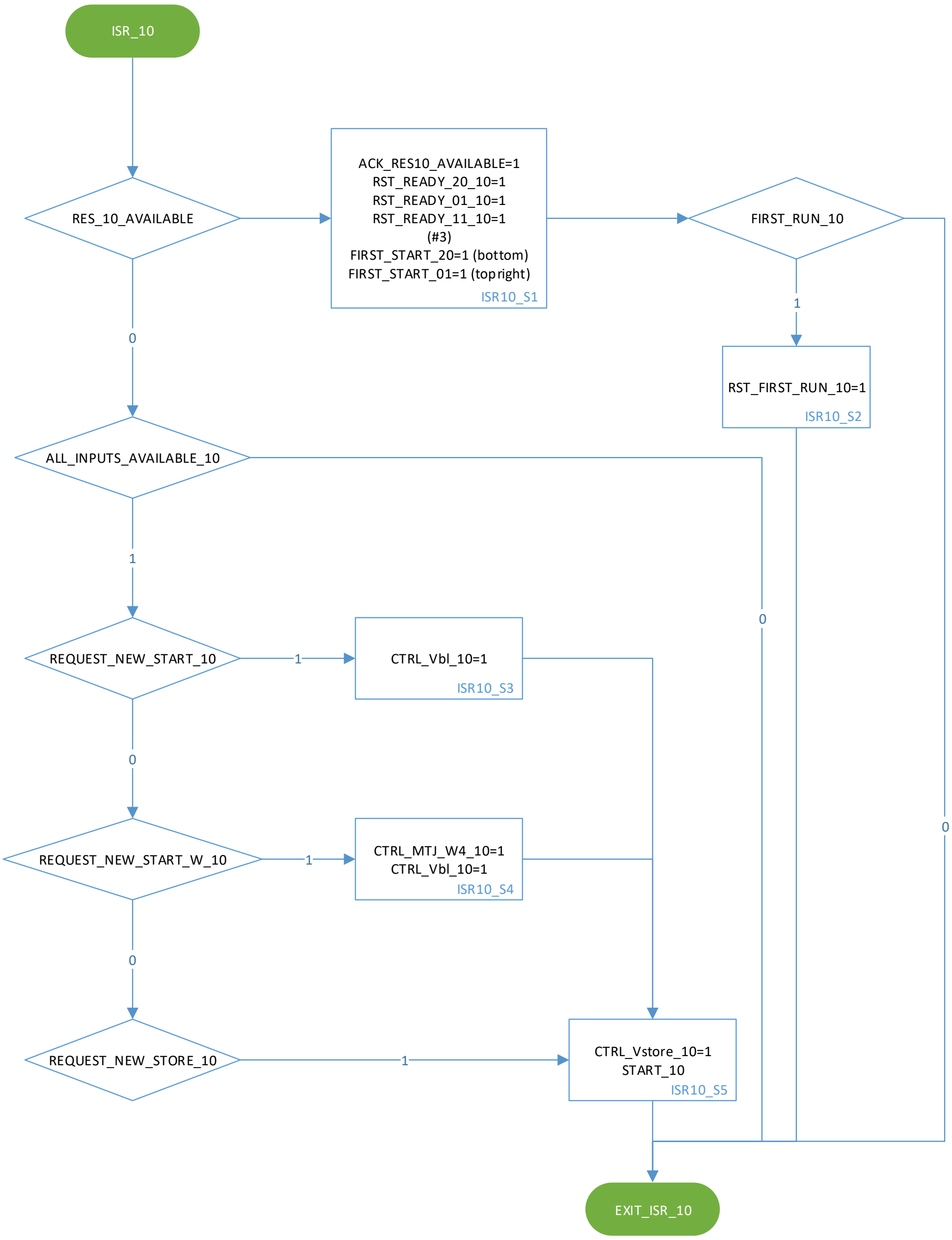


Figure 5.36. Master FSM, ISR 10

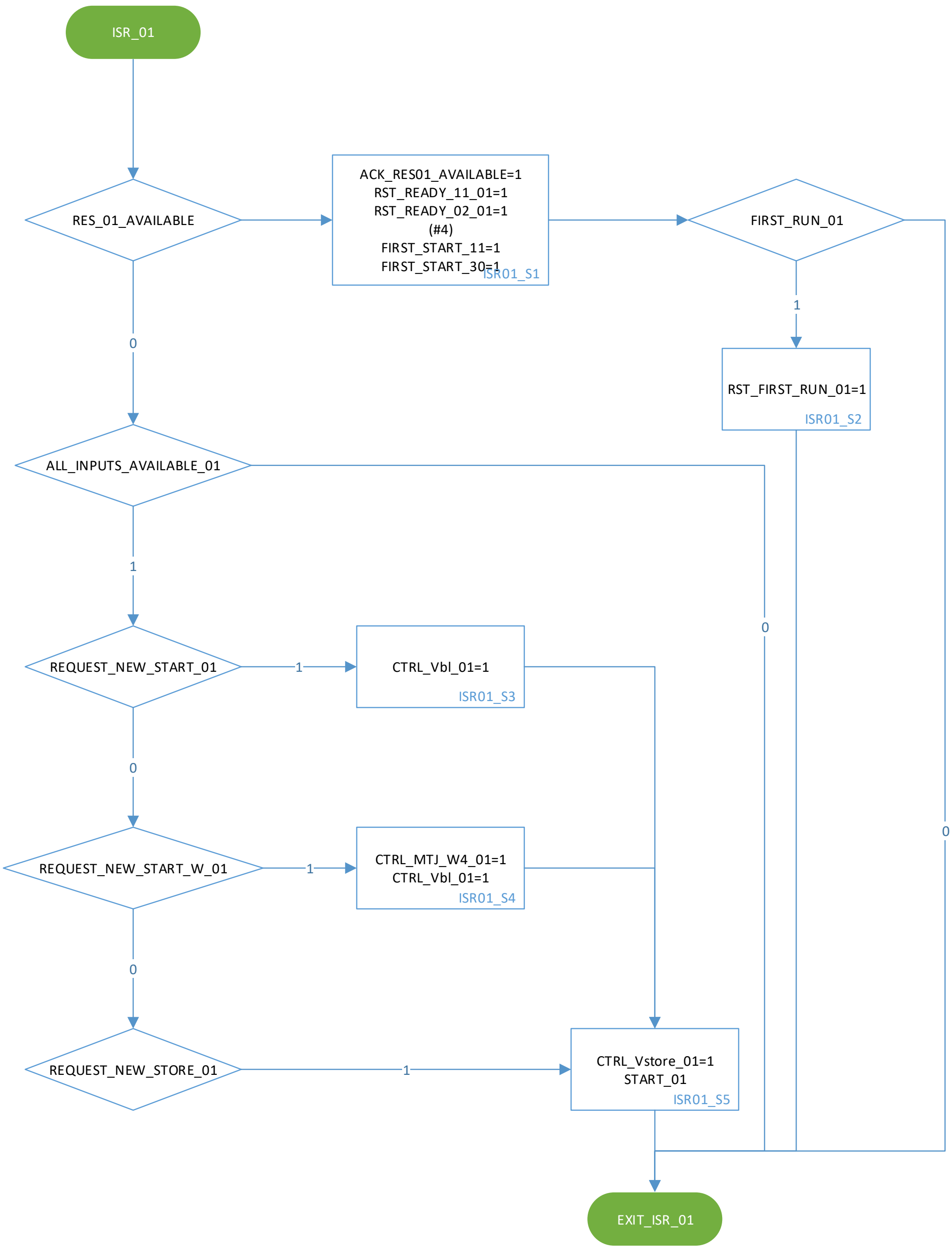


Figure 5.37. Master FSM, ISR 01

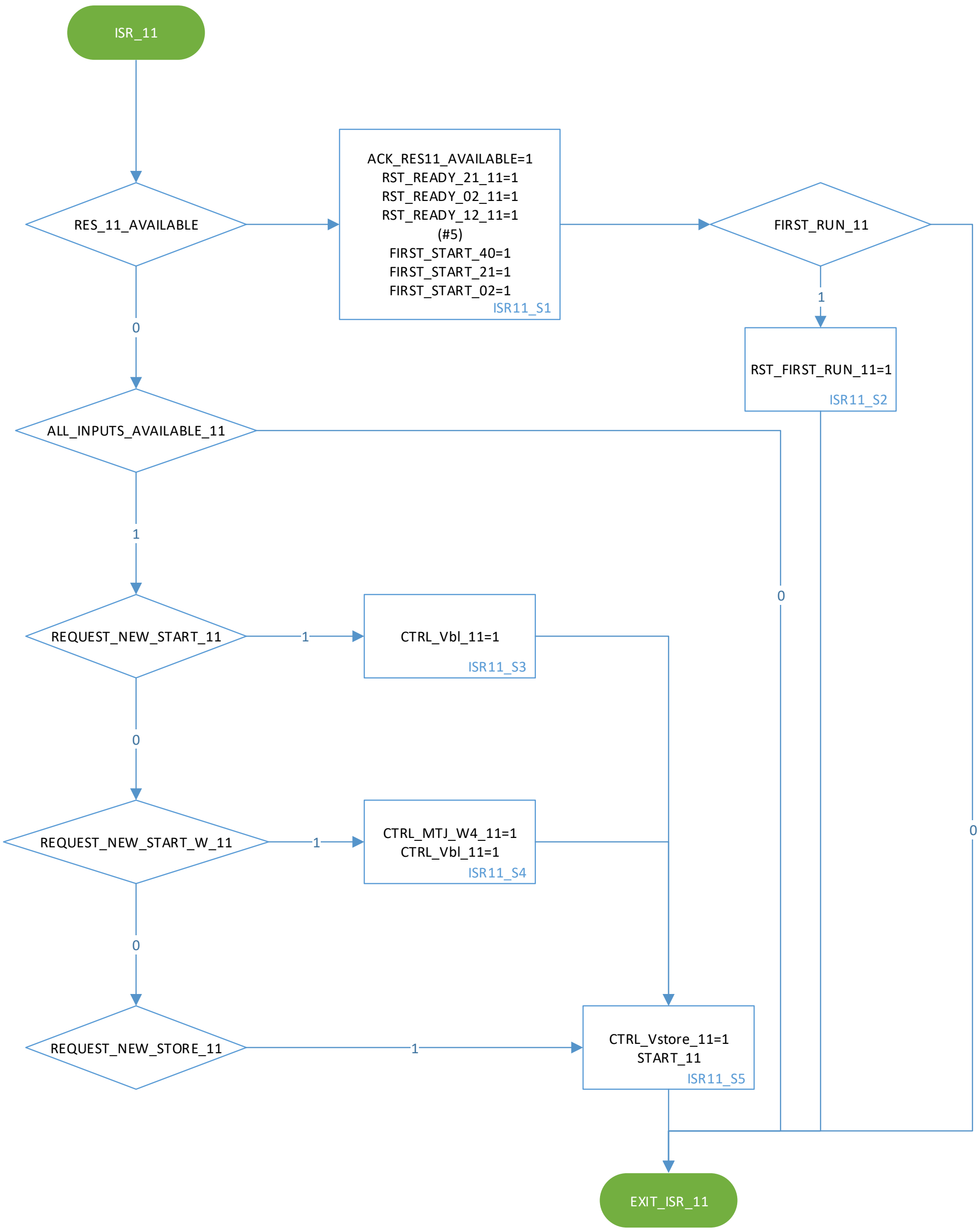


Figure 5.38. Master FSM, ISR 11

6. Logic in memory - second architecture

The structure for the logic-in-memory array described in chapter 5 is very powerful and allows almost a full flexibility and adaptability to the request of the particular algorithm that has to be implemented. Due to this huge flexibility, however, the structure of the array and most of all the FSM that controls the memory itself is very complex and not optimized. A simpler structure could be derived from that one by renouncing to some capabilities of the array, like for example the transfer of the result both to the cells on the right and on the bottom and to the cell on the top right.

Another possibility could be to specialize the structure of each cell: one memory row, for example, could contain only full adders, while the row after that could be specialised in the computation of the *AND/OR* logic functions. In this way it would be avoided the production of two results at the same time, so it wouldn't be necessary to to choose one of them before going on with the computations.

Another reason for the complexity of the array comes from the desire of reducing as much as possible the energetic inefficiencies by collecting back all the skyrmions that are no more needed for the computation cycle. If the top and the bottom tank were eliminated the complexity of each cell would decrease: this, however, would deteriorate the energetic performances of the structure.

So, it is possible to simplify the structure already analysed, renouncing either to its flexibility or to its energetic performances.

The aim of this chapter, however, isn't to find a simpler version for a LiM array by starting from the structure already analysed. One of the reasons why the array studied in chapter 5 is so complex is that it allows the execution of many different algorithms, without being specifically built for any of them. For this reason, in order to find a simpler and more efficient LiM structure, a specific algorithm application has been studied. The article that has been taken as a reference for this chapter is [37], where a LiM architecture capable of a minimum/maximum search inside the memory array has been proposed. The algorithm and the memory cells presented in the article are analysed in detail in section 6.1. After the description of the steps performed by the search algorithm in order to find the maximum or the minimum number stored inside the array, it will be shown the memory array based on skyrmions that has been developed starting from the original array version. Finally, in section 6.3 will be presented all those blocks that belong to the datapath of the memory, but that are at the same time needed for the control of its behaviour. These blocks, described only approximately in [37], have been developed specifically for the control of the memory array presented in section 6.2.

6.1. Original architecture and control blocks

Let's assume to have a memory array containing a set of N numbers, each represented on N_{bit} bits: each row of memory contains N_{bit} cells, each storing a single bit of the corresponding number. The aim of the algorithm is to find the maximum (or the minimum) out of these numbers. Let's assume that the search is aimed at finding the maximum value: a possibility for doing this is to use a shifting mask, like shown in figure 6.1.

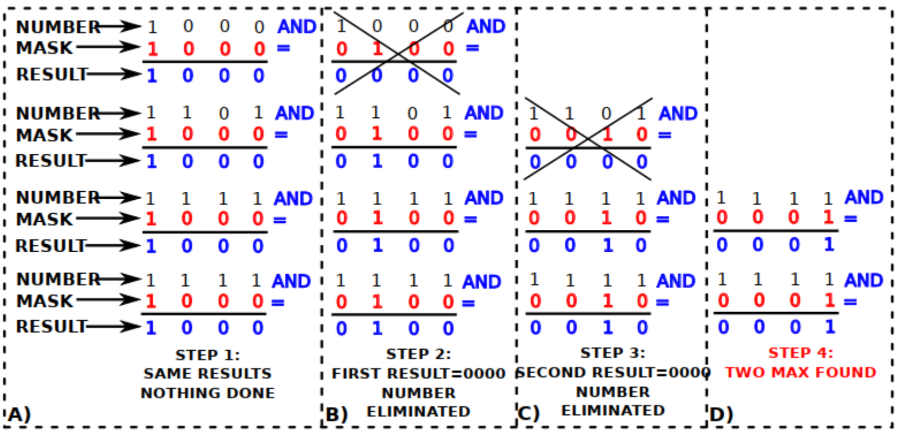


Figure 6.1. Steps of the algorithm described in [37]. Figure extracted from [37].

A shift register containing N_{bit} bits is initialized with a 1 in the MSB (most significant bit) position and with a series of 0 in the remaining bits; then the value of the N_{bit} bits of the mask is provided in parallel to each of the N rows of cells containing the N numbers to be analysed. Each memory cell contains an AND gate: during each search step, each cell will perform an AND operation on the bit it stores together with the bit that it receives from the mask register. For example, let's say that the aim is to find the maximum number in a set of three numbers represented on 4 bits each: the mask register will be initialized to the value 1000, and these bits will be provided in parallel to the three rows composing the memory array: the cells in the column 0 will receive the MSB, set to 1, while the three remaining columns will all receive a 0. At this point, the 12 cells inside the array will compare the value they store with the bit received from the mask register. The output of the AND gate contained inside each cell will be 1 only if the bit stored in it is equal to 1, and if at the same time the bit coming from the mask is 1 as well. So, in the first step of the algorithm, only the cells belonging to the first column (and containing the MSB of the three numbers) will be allowed to produce a 1 as an output: if this happens, it means that the MSB of the corresponding number is equal to 1. Since the aim is to find the maximum number, if some cells produce a 1 while others produce a 0, this already allows to discard all the numbers that do not provide a 1 as output from the AND gate: this, in fact, implies that the MSB of the corresponding number is equal to 0, and so that number will be for sure be lower than a number that provides as output a 1. However, if all the cells produce a 0 (or a 1), no action can be performed, because at this point all the numbers seem equally low (or equally high), so the algorithm must go on.

The second step, either if some rows have already been discarded or if all of them must still be considered, consists in a shift towards right of the mask register, so that in this particular case it will contain the value 0100. Then the same steps are repeated: a bitwise AND is performed in parallel within each row; all those rows that will produce a 1 as output of the AND gate contain a 1 in the $MSB - 1$ position of the number they store, while all the remaining rows contain a 0 in that position and can be discarded, because the number they contain is for sure lower with respect to the other numbers, which have a 1 in that position.

The algorithm is iterated until the mask register has performed N_{bit} shifts towards right: this in fact means that all the bits inside the numbers have been compared with the bit set to 1 contained in the mask, and so all the rows that were not of interest have already been discarded. The rows that have remained, then, are the ones that contain the result of the search. It is interesting to notice that the algorithm works correctly even if the same maximum number is contained in two or more rows at the same time: those rows, in fact, will all survive the comparison and reach the end of the algorithm together, as shown for example in figure 6.1.

The same algorithm is able to perform the search for the minimum value with just a simple change: instead of rejecting all those rows that during the particular search step produce a 0, is enough to reject all those that produce a 1 instead. If the cell inside each row that is comparing its value with the 1 coming from the mask register (let's call it "active cell") produces a 1, in fact, it means that the number stored in that row has a 1 inside that position: so, that number for sure will be larger than the numbers considered in the same search step that have a 0 in that position, and so it can be rejected.

It is clear that each memory cell must contain an *AND* gate, needed to perform the comparison with the bit that comes out from the mask register. At each comparison step only one of the cells inside each row (the "active cell") will have an *AND* output equal to 1, and all the following decisions must be taken considering in parallel the output of all these N "active cells". In order to reduce the number of signals that must be analysed at each search step, is then possible to produce a single signal in output from each row by cascading a series of *OR* gates. Each *OR* gate will have as inputs the output of the *AND* gate of the cell i and the output of the *OR* of the cell $i-1$: in this way, all the outputs of the *AND* gates are put in *OR* together, and if the active cell inside that particular row produces a 1, then for sure the output of the last *OR* inside the row will be 1 on its turn. Then, according to the type of search, that particular row may or may not be discarded.

As mentioned in the article, the memory array behaves as a traditional memory array, able to perform standard read and write operations, and now and then it can be used to do a min/max search as well. To grant the capabilities of a traditional memory array, then, bitlines connecting the cells inside the same column and word lines activating the cells inside the same row must be included as well. These considerations lead to the structure shown in figure 6.2. This is the structure of the cell that will be used as a model for the construction of the memory array, detailed in section 6.2.

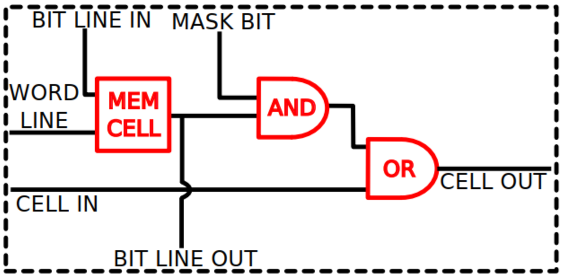


Figure 6.2. Structure of the LiM cells proposed in [37]. Figure extracted from [37].

All the decisions taken starting from the *OR*-wise output of each row are performed by additional datapath blocks: this is one of the main advantages of this architecture, since in this way a less complex FSM will be needed to control the whole memory structure.

The organization of these blocks inside the memory architecture is shown schematically in figure 6.3.

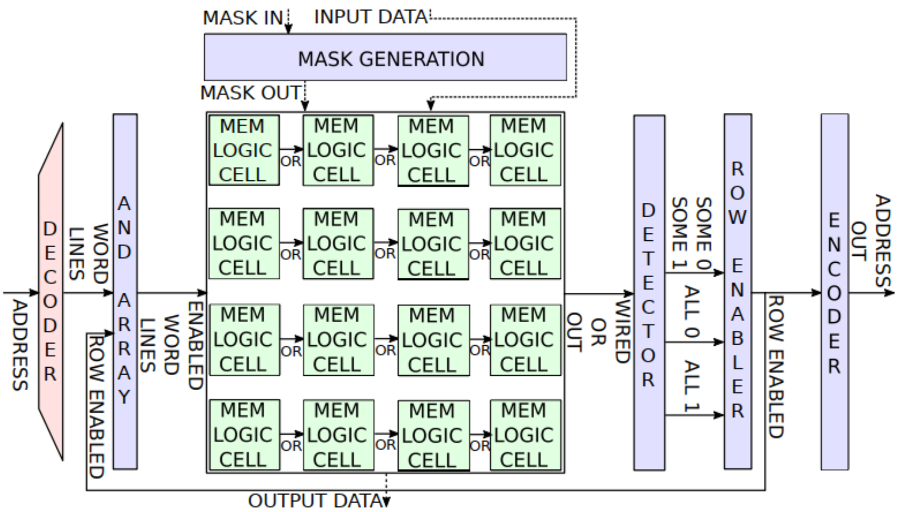


Figure 6.3. Structure of the smart memory presented in [37]. Figure extracted from [37].

In [37] only a brief description for each of these block is provided and no scheme is available. Since the behaviour of these blocks is extremely important for the comprehension of the whole memory structure, a more detailed explanation will be provided in section 6.3, where will be shown also the structures that have been developed while thinking about the necessities of the memory array described in section 6.2.

6.2. Memory array structure

The memory array based on skyrmions that has been developed starting from the original LiM array is shown in figure 6.14. The array, the control signals and all the control blocks described in section 6.3 have been defined and developed in order to suit an array hosting three numbers with three bits each, but of course the the generalization to any different value of N and N_{bit} is straightforward.

When looking at figure 6.14, the first fundamental difference with respect to the organization proposed in [37] that must be underlined is the placement of the numbers inside the array. Until this point, in fact, the numbers where occupying one row each, with the MSB bit in the leftmost position inside the row and the LSB (less significant bit) in the rightmost. This is the most straightforward representation of numbers when using a CMOS-based memory array. The main difference when developing a memory array based on skyrmions, however, is in the fact that skyrmions are already non-volatile information carriers. This means that is enough to nucleate them inside a nanotrack for having a storing of information: it isn't necessary to move them anywhere else. This is the principle exploited in the well known racetrack memories, where skyrmions are nucleated by a write head and moved along a racetrack, implementing in this way the memorization of information with a serial organization. In this case, however, a serial organization of the information is not desired: the aim here is to analyse in parallel the value of N different numbers, each composed by N_{bit} bits. This implies the presence in parallel of N racetracks: in each racetrack a writing head nucleates N_{bit} skyrmions, according to the value of each bit in the corresponding number, and these skyrmions remain hosted by the racetrack until an elaboration is requested. This explains why, in the memory array shown in figure 6.14, each number is occupying a column instead of a row, with the MSB bit placed at the input of the bottom cell and the LSB at the input of the top cell. Each column is in fact a racetrack: during the initialization of the memory each writing head, placed at the beginning of the racetrack, nucleates first the MSB bit, which is then pushed towards the bottom by applying the voltage V_{bl}; then the $MSB-1$ bit follows, and so on until the LSB bit, which will be the last to be nucleated and will stop in correspondence of the top cell. Figure 6.4 should dissipate any possible doubt concerning the organization of the numbers inside the array.

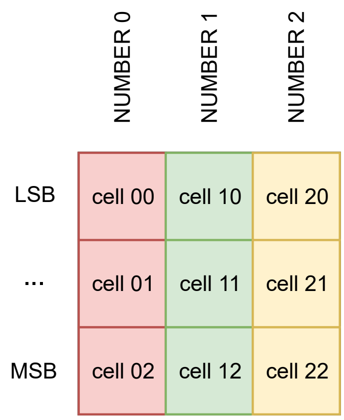


Figure 6.4. Organization of the numbers inside the skyrmionic memory array of figure 6.14.

This organization implies, of course, that the N_{bit} bits coming from the mask register must be distributed along the columns, and not along the rows: the MSB of the mask register must be provided in parallel to cell 02, cell 12 and cell 22, the $MSB-1$ bit to cell 01, cell 11 and cell 21, and so on.

In order to know the value to be nucleated along each racetrack, it's necessary to look bit by bit at the words that the external world desires to store inside the memory. To do so, three shift registers (more in general, N shift registers) are employed, as shown in figure 6.5. At each clock cycle, if the signal *CTRL_WORD_-SHIFT* is asserted, the registers are shifted towards left by one bit, and the writing heads at the head of each racetrack are controlled according to the value that comes out from the corresponding shift register: in this way, if the leftmost bit inside the register is a 1, a skyrmion is nucleated, while no action is performed if the bit is set to 0. This writing operation can take place on the three racetrack in parallel: it is simply a matter of controlling the signals which guide the writing operation. This control is operated by the control blocks detailed in section 6.3.

Of course, concerning the signals that appear in figure 6.5, both the inputs $WORDx$ and the controls $CTRL\_WORDx\_STORE$ are activated by the external world, while the remaining signals are managed by the memory.

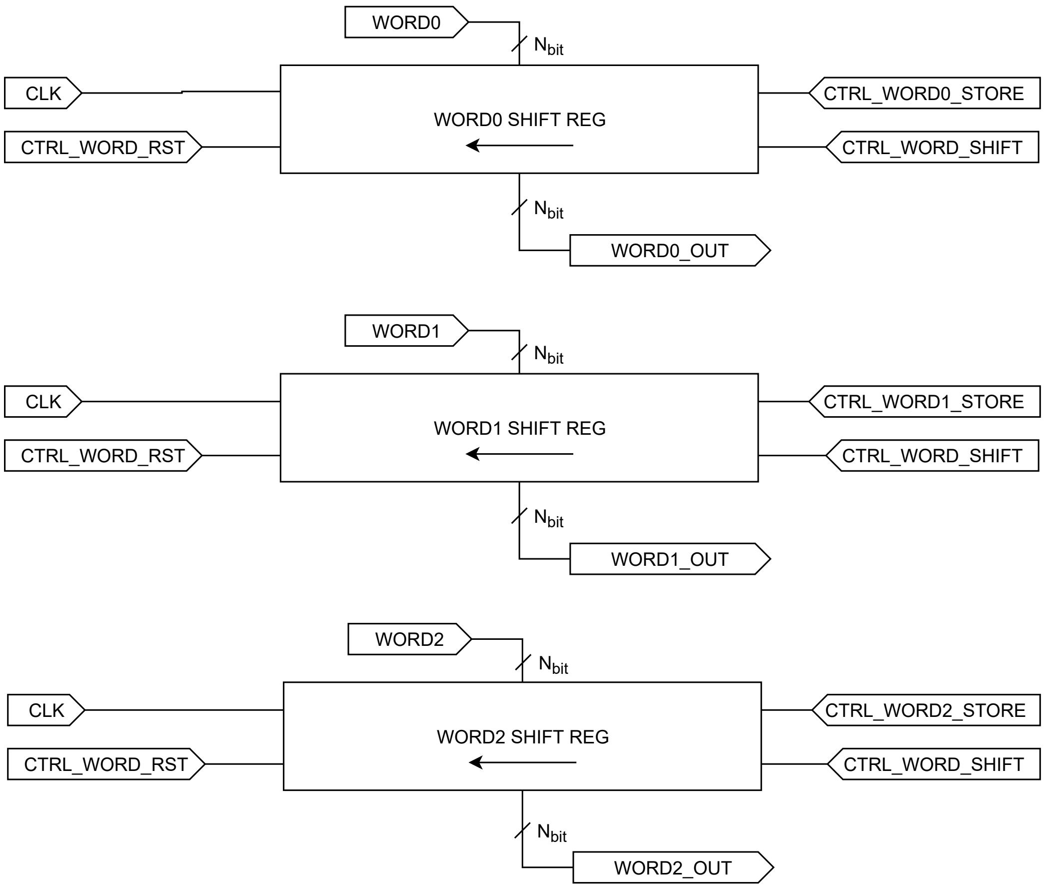


Figure 6.5. Shift registers hosting the words to be written in the array.

Let's focus now on the structure of the array shown in figure 6.14. Once nucleated by the writing head MTJ_WBx and put into movement by the voltage V_{Bx} (where x stands for the index of the column), the skyrmions reach the blue areas Dxx. These rectangles are inserted in the drawing simply to signify the region where each skyrmion stops inside the racetrack. From that point on, each skyrmion can move right (towards the input of the elaboration region), towards left (more on this later), or maybe towards bottom, if a new impulse is applied on V_{Bx}. Still, for the implementation of the algorithm described in section 6.1, only the movement towards right is of interest. As soon as each skyrmion inside the racetrack reaches the proper blue rectangle, that is, the proper stop region, the voltage V_{Bx} is turned off: as a consequence, the skyrmions representing the bits of each number stop inside the racetrack, and this corresponds to the memorization of the information.

This information may be read with a reading operation. To do so, the voltage V_{opx} is applied. Doing so, each skyrmion enters the elaboration region and, as soon as a peak of current occurs (so, the voltage V_{opx} should provide at least one peak of current, at some point), the skyrmion goes past the notch, enters the H structure of the *AND/OR* gate and, since no other input is applied, goes out from the top output of the gate (the *OR* output). In this way, pushed a bit forward by the voltage V_{opx}, the skyrmion reaches the read head MTJ_Rxx, where its value is detected. Finally, the skyrmion may be expelled from the track or it may stop right at the end of it until a new skyrmion arrives and pushes it out from the track: in any case, it won't be collected and reused later, as it happened in the cells described in chapter 5.

When the skyrmion is pushed by the voltage V_{opx}, before going past the notch, it crosses a green duplication element. This element is exactly the one used also in chapter 5, proposed in [40] and exploiting the technology described in [43]. Thanks to this element, the value of the bit represented by the skyrmion is not lost, because as soon as the skyrmion is pushed towards the elaboration area, it is also duplicated and a copy of it is placed back inside the blue "stop region" of the racetrack. Moreover, it is not needed any additional voltage source to control this operation, because everything can work simply with the voltage V_{opx}, needed for moving the skyrmion towards the read head.

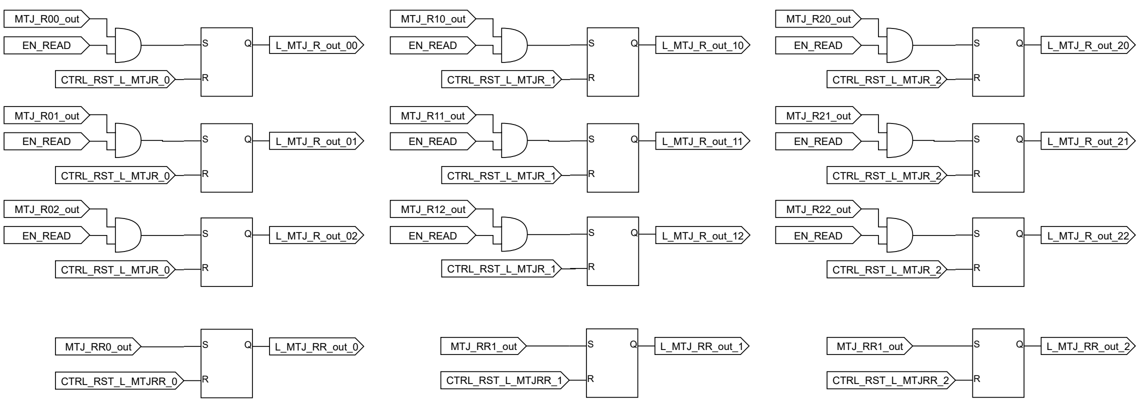


Figure 6.6. Latches storing the output of the read heads that appear inside the memory array.

Of course, the read head generates only a brief impulse when the skyrmion passes below it. This impulse must be stored somewhere, together with the remaining $N_{bit} - 1$ impulses from the other read heads, which are dealing at the same time with the other bits of the same word: the requested word in fact has to be read externally by the electronic component which requested it, and it is not guaranteed that this component is able to capture the brief impulses of the read heads. For this reason, each read head is connected to a SR latch (which is sensitive to the rising edges on the output the corresponding read head only if the signal *EN_READ* is active): the outputs of these latches are then used to form a bus, which is selected by a multiplexer according to the memory word that was requested. The latches used inside the array, together with the signals that control them, are shown in figure 6.6, while the multiplexer used to select the memory word requested is shown in figure 6.7. The signals that appear inside these two pictures are managed and activated by the control blocks that will be detailed in section 6.3.

Now that the topic of both the reading and the writing operations has been covered, it is time to concentrate on the operations needed for implementing the search algorithm. Let's assume that three numbers have been written and are stored inside their respective racetrack, waiting for V_{opx} to be applied. If a search operation is desired, first of all the mask register must be initialized and its value must be used in order to nucleate the skyrmions needed for performing the *AND* operation inside each column. The signals that control the mask register (again a shift register, but this time the shift direction is towards right) are shown in figure 6.8.

So, the first operation to be done is the activation of the signal *CTRL_MASK_-STORE*: doing so, the value that is fixed at the input, with only the MSB set to 1, will be stored inside the register. Three copies of the output from the mask register are sent in parallel to the three columns of cells, and the three bits composing the value of the mask register's output are used to drive the write head MTJ_Wxx inside each cell. In particular, in the first step of the algorithm only the write heads MTJ_W02, MTJ_W12 and MTJ_W22 will nucleate a skyrmion, while the remaining heads won't perform any action, since their control bits are still set to 0.

When these skyrmions are nucleated according to the output from the mask register, the voltage V_{opx} can be applied. Differently from what happened with a

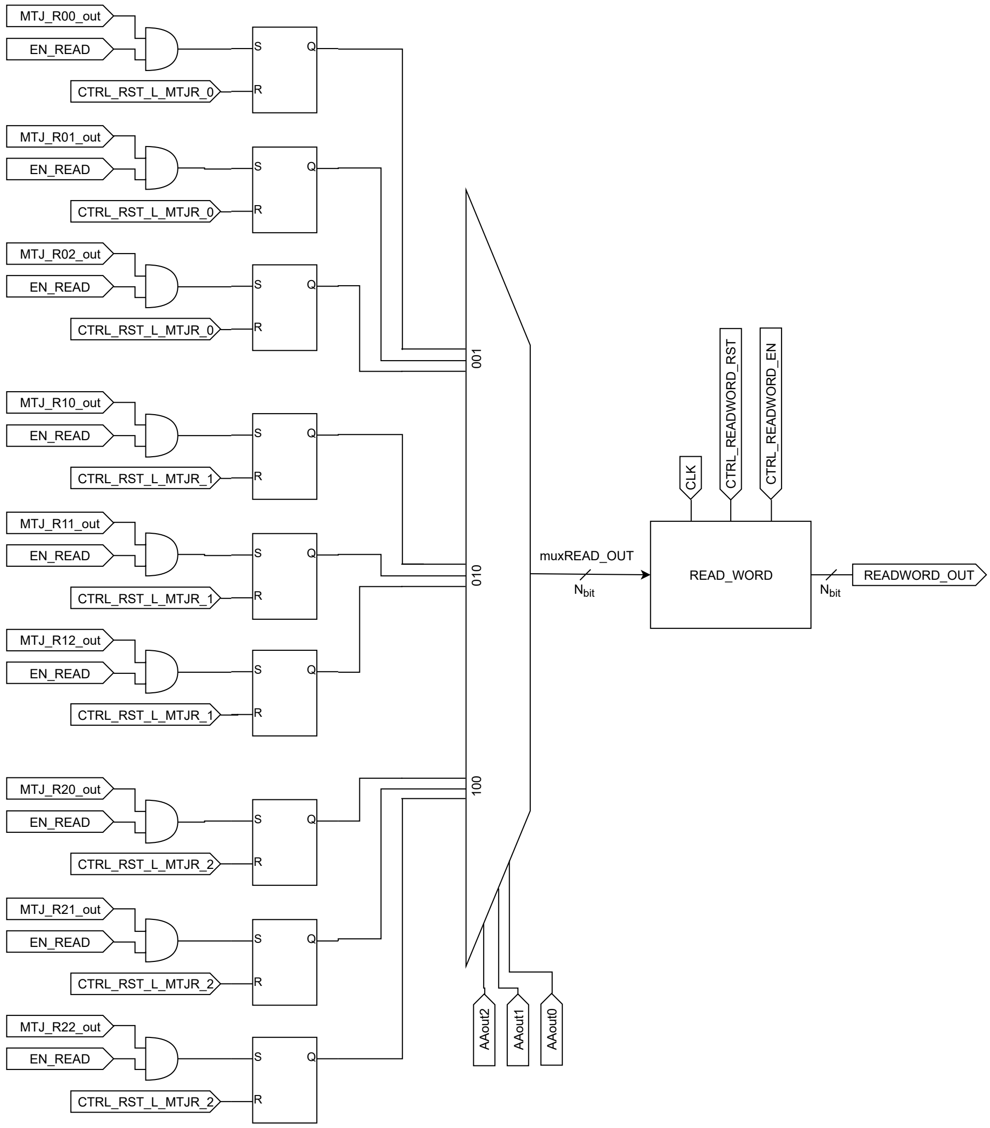


Figure 6.7. Multiplexer used to select the memory word requested from the external.

read operation, this time there may be up to two skyrmion that are pushed towards the *AND/OR* structure: if this is the case, the output of the *AND* will be 1, just like the output of the *OR*; if instead only one skyrmion is present, only the output of the *OR* will switch to 1. The output of the *OR* is ignored in this operation, and its value, even if it is detected by the read heads *MTJ_Rxx*, it is not stored inside the latches of figure 6.6 because the signal *EN_READ* is not asserted.

It should be underlined here that, thanks to the duplication elements across

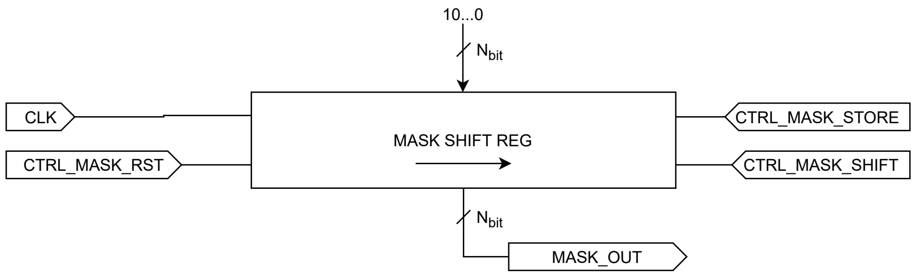


Figure 6.8. Mask register: at each step of the algorithm the content of the register is shifted towards right by one position.

which the skyrmion passes through before crossing the notch at the input of the *AND/OR* gate, the value stored in the racetrack is always restored back, so that there is no need to write back its value by activating the write head at the beginning of the racetrack. It would be possible to avoid the use of these duplication elements by exploiting the shift registers shown in figure 6.5 for keeping trace of the information to be written back: if at each shift the bit coming out from the left of each register is injected back on the right, the word stored inside the register wouldn't get lost after each write operation. Every time that a skyrmion is extracted from the racetrack, then, the word stored inside the corresponding shift register could be used again to perform a new write operation inside the racetrack, with the aim of restoring back the information that was lost. The control of the memory would become a bit more complex, but no additional memory elements would be needed for allowing this write-back operation, apart from a slight modification of the word-shift register, which now would need also a serial input and a serial output, apart from a parallel input and output.

As already explained, only one cell inside each column, that is, the "active cell" of that particular search step, will produce a 1 on the output of the *AND* gate: all the other *AND* outputs inside that particular column will be 0. This means that, joining the output from the *AND* gates together, only one skyrmion (at most) will be coming out from the output of the join. As a result, to implement the chain of bitwise-*OR* gates, in the memory array based on skyrmions is enough to use a chain of join elements, connecting together all the outputs from the *AND* gates. Then, to detect the actual presence of a skyrmion, the read head MTJ_RRx is placed at the output of the last join element; also the output from this read head is stored inside a latch, as shown in figure 6.6, so that the signal is maintained stable and can be analysed by the control blocks, which must take their decisions according to the value detected.

A final comment, before concluding the topic of the memory array structure, must be spent around the left output from the Dxx elements. The skyrmion is allowed to leave the racetrack from that output only when the voltage V_{trxx} is applied. The destination of that path is always the corresponding cell inside the column (the word) on the right: this connection, in fact, allows each cell to use as operand the value stored inside the corresponding cell inside the word on the left, that is, it allows to perform bitwise operations between adjacent words. This is not a request of the search algorithm presented in [37] and for this reason, even if the controls needed to manage this movement of data have been implemented, they are not actively controlled by the FSM. The possibility of performing also this kind of operation has been taken into account only to formulate a slightly more powerful and generic memory array, but if the desire is to actively use this functionality, as soon as an algorithm is chosen and a specific application is desired, the FSM that controls the memory must be modified and tailored in order to fit that specific algorithm.

6.3. Control blocks

Looking at the memory array shown in figure 6.14, some concerns may arise from the number of signals that are again needed to control the whole array. Even if these signals are similar in their name, in fact, the moment in which they must be turned on to control the specific cell to which they are dedicated changes depending on the steps of the algorithm, on the particular values needed to be stored and on many other conditions. A FSM that controls all these signals one by one, then, must be for sure very complex, almost as complex as the FSM described in chapter 5. In this particular case, however, the strength of the structure proposed in [37] is to use a set of datapath blocks, which are useful for deciding what actions must be performed according to the results obtained in each step of the search algorithm. These same blocks, however, are extremely useful also to mask and select in a finer way all the signals that are needed by the skyrmionic memory array, for example for nucleating the skyrmions, for moving them, for detecting their presence and so on. Thanks to these same control blocks, then, the FSM can be made much simpler, even completely independent on the particular number of words that the array is able to host.

For this reason, the functionality of each of the control blocks depicted in figure 6.3 is of vital importance for the correct behaviour of the array and must be discussed in detail. In the following, a in-depth explanation of the role of each control block is provided, together with the scheme that has been developed in order to fit the particular memory array shown in figure 6.14: as already mentioned, a generalization to any other value of both N and N_{bit} is straightforward.

6.3.1. Detector

The main aim of the *detector* block is to identify all those conditions in which some words can be excluded from the comparisons to be performed in the future steps of the search algorithm. The condition in which some words can be excluded is verified when the outputs of the bitwise *OR* performed inside each column are not uniform, that is, some outputs are 0 while others are 1. As suggested in [37], to verify this condition is enough to use two trees of *AND* gates, whose outputs are then combined by a single *XNOR* gate. The structure derived from this description is shown in figure 6.9.

In this figure, the signals $L\_MTJ\_RR\_out\_x$ correspond to the bitwise-*OR* outputs coming out from the array (they are the reading of the MTJ_RRx at the end of each column, after being latched in the memory elements of figure 6.6), while the selection signals $LATCHx\_OUT$ come from the block *row disabler* and are equal to 1 when the corresponding row must be rejected from the comparisons.

The left tree produces as output a 1 only when all its inputs are equal to 1; at the same time, the right tree produces a 1 only when all the inputs are equal to 0. If neither condition is verified, then some words can be excluded from the comparisons of the next steps: this condition is signified by the activation of the signal *ENABLE_Rowdis*, which is obtained by the *XNOR* combination of the two outputs from the trees.

As discussed in [37], for the correct functioning of the algorithm is essential that the bitwise-*OR* coming from the words that must not be taken into account are excluded from the inputs to these trees of *AND* gates: for this reason some multiplexer are interposed. The other input of each multiplexer has been chosen

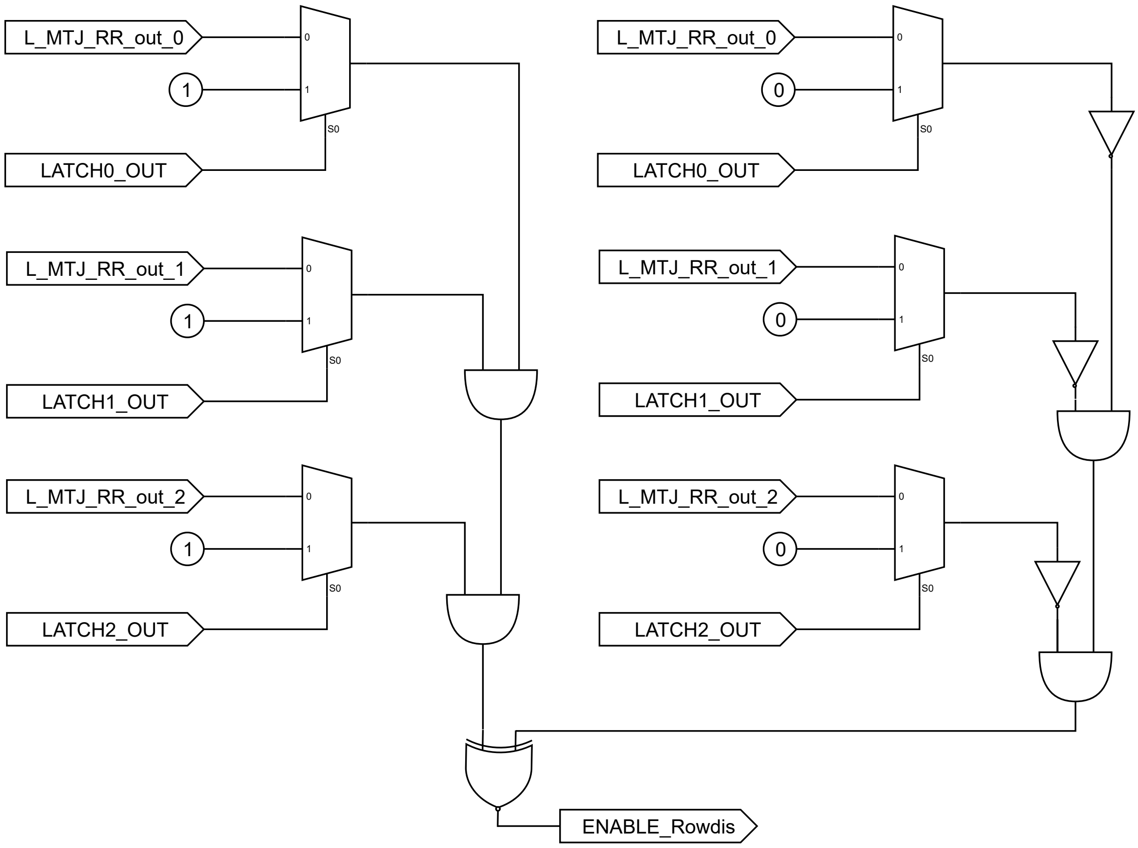


Figure 6.9. Structure of the *detector* block: two trees of AND gates combine together the outputs of the bitwise-*OR*.

according to the following reasoning: the ”transparent” value for the left tree, that is, the value that is never able, alone, to determine the value of the output, is equal to 1: this means that the values that must not be taken into account should be masked by a value fixed to 1, since by itself it is not able to change the output of the AND tree. Vice versa, the ”transparent” value for the right tree is equal to 0, and so that is the value chosen for the other input of the multiplexers.

6.3.2. Row disabler

The *detector* block produces as output the signal *ENABLE_Rowdis* which, when active, identifies all those conditions in which some words can be discarded. For this reason this signal is used as enable to the block *row disabler*, whose structure is shown in figure 6.10.

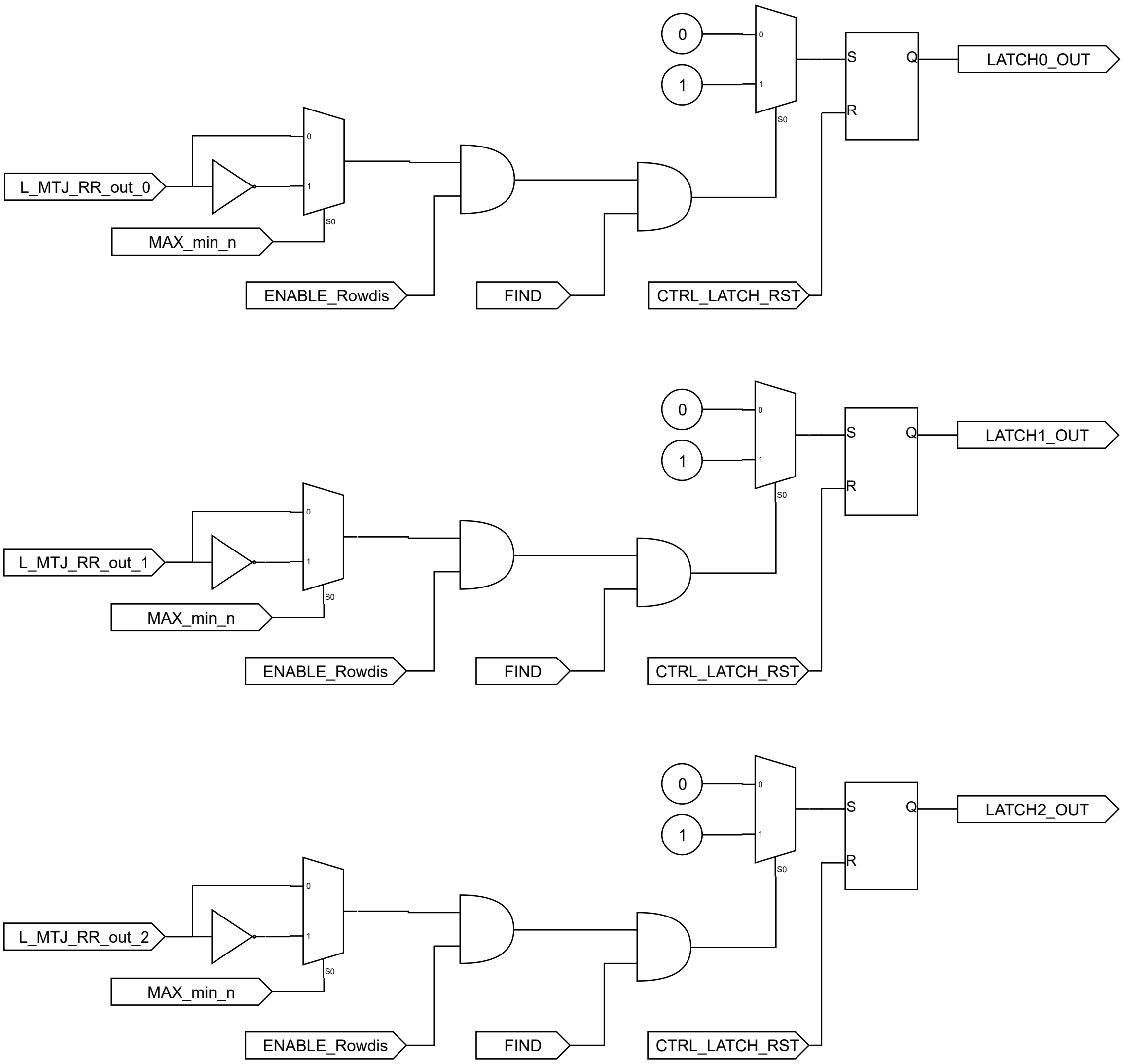


Figure 6.10. Structure of the row disabler. A latch storing a 1 means that the corresponding row must be rejected from the future comparisons.

The main components of *row disabler* are the latches needed to remember whether the corresponding word must be considered in the comparisons. At the very beginning of the search operation each of these latches is reset to 0, and only when the set of conditions that drive the bunch of gates at its input are satisfied, the value stored inside the latch becomes 1, signifying that from that moment on the corresponding word must be ignored. The latches which switch to 1 during the search are reset back to 0 only at the end of the algorithm, when the desired memory word has been found.

The conditions that must be satisfied in order to reject a certain word are the following: first of all, it is necessary to know if the aim of the search is the maximum or the minimum value stored in the memory. If the maximum is desired, then the word may be rejected if the output of the bitwise-*OR* is equal to 0, vice versa if the minimum is desired. However, the value coming out from the bitwise-*OR* must be considered only if there are other outputs different from that value, that is to say, if the signal *ENABLE_Rowdis*, produced by the *detector*, is active. Finally, of course, the memory array must be in the *find* mode, and not for example in the *read* or in the *write* mode, where it should behave like an ordinary memory array. Only if all these conditions are satisfied at the same time, the output of the last *AND* gate switches to 1, enabling the input of the latch to have a rising edge, which is readily stored by the memory element: from now on the latch will store a bit equal to 1, signifying that the corresponding word must be ignored in the following steps of the algorithm.

6.3.3. Encoder

The encoder is the block that allows the external world to know, at the end of the search operation, what is the address of the word found. Its structure is shown in figure 6.11.

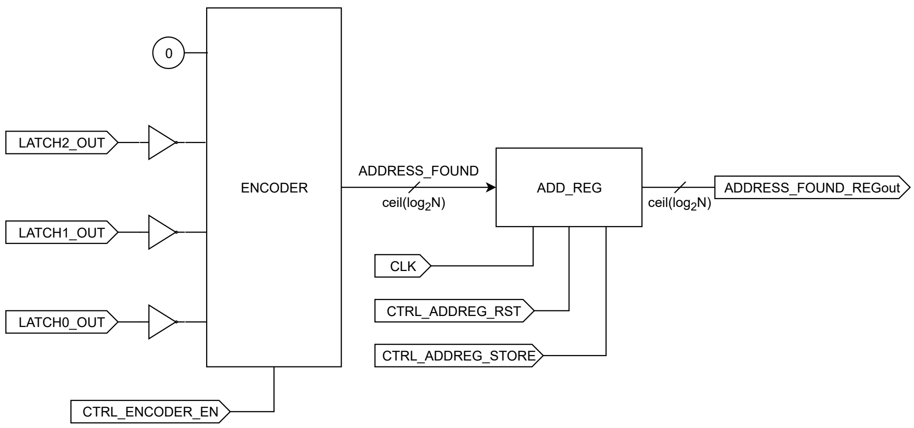


Figure 6.11. Structure of the encoder. It is a priority encoder, so only one address will be provided to the external, even if more than one word corresponding to the search needs has been found.

The encoder used in this block is a priority encoder: this means that, even if more than one word corresponding to the search parameters has been found, the address that will be provided to the external will be only one. However, since the array and the algorithm itself allow to find more than a single word that satisfies the requests, in order to exploit this capability it would be enough to modify the electronic component, allowing it to use more than one output port at the same time.

The address of the word that has been found at the end of the search process is recorded by the latch (or the latches, if more than one word has been found) that is still equal to 0 inside the *row disabler* block. This address is transmitted to the external by enabling the encoder to activate its output according to this input values, and at the same time allowing the register after the decoder to store the value of the address just encoded.

6.3.4. And array

The aim of the *and array* is to activate selectively the word lines of the memory array, according to the outputs from the *row enabler* block. In the particular case of a skyrmionic memory array, this block is exploited, together with the *FSM-array adapter* discussed in section 6.3.5, for preventing the activation of all those signals (for example, the voltage generators' control signals) that would make the corresponding memory column start the computations. The structure of the component is shown in figure 6.12.

The signals that are directly used to understand whether a certain memory word has to be considered or not are the outputs $AAoutx$. These signals can become active only if the corresponding latch inside the *row disabler* is still equal to 0: as soon as its content switches to 1, in fact, the output of the *AND* gate inside the *and array* block goes to 0 and the corresponding memory word is disabled, since its $AAoutx$ signal becomes equal to 0.

Another condition is needed in order to activate the $AAoutx$ signals. As mentioned, these signals are used to select a word inside the array, whatever the modality (read, write or find) in which the memory is currently. For this reason, these signals must be equal to 1 when the decoder, which is used when a specific memory word must be read/written, is activated. Both in the read and in the write mode all the latches inside the *row enabler* block are reset to 0, so it is enough for the decoder to activate one of its outputs (in figure 6.12 they are called $DEC1$, $DEC2$

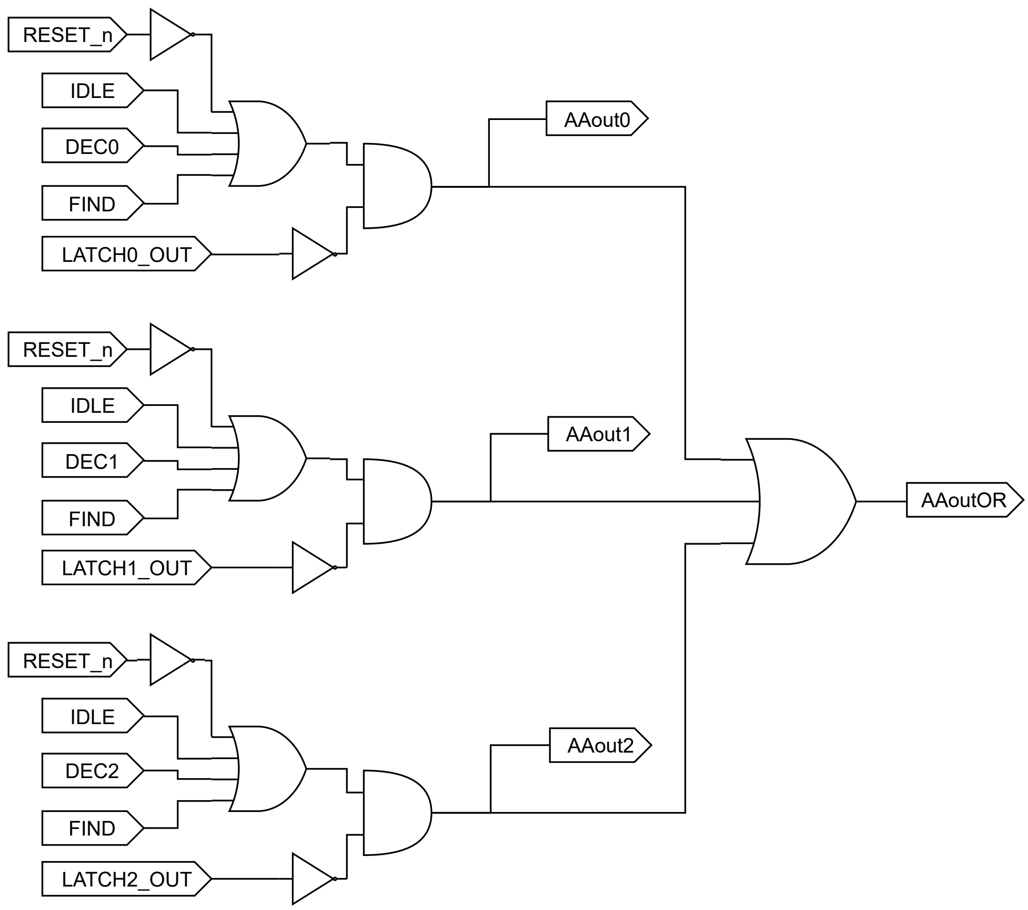


Figure 6.12. Structure of the *and array*.

and $DEC3$) in order to activate the corresponding *AAoutx* signal, thus selecting the memory word of interest.

When the memory array is in the *find* mode, however, all the words inside the memory must be considered, at least until some of them get disabled by the latches inside *row disabler*. During this operation mode, so, the decoder is not used: this is why the signal *FIND* is one of the inputs of an *OR* gate, together with decoder outputs. If the memory is in the *find* mode, the *FIND* signal is asserted, thus allowing all the memory words to be selected at the beginning of the algorithm. *FIND* will remain asserted until the end of the search operation, and the latches inside *row enabler* will take care of masking the memory words that must be excluded from the comparisons during each step.

A third condition that allows the activation of *AAoutx*, as long as the output of the corresponding latch is equal to 0, is the activation of the signal *RESET_n* (active low) from the external. This signal is the same used for the reset of the FSM. The reason why this signal is used for activating *AAoutx* will become clear in section 6.3.5. Finally, the last condition that allows the activation of *AAoutx* (again, as long as the latches inside *row enabler* are reset to 0) is the activation of the signal *IDLE*. Also in this case the reason why this signal is used as input to the *OR* gate will become clear in section 6.3.5.

Each of the signals *AAoutx* is one of the inputs of a *OR* gate, whose output is the signal *AAoutOR*. This is done with the aim of simplifying the FSM that controls the memory, whose structure is shown in figure 6.15. Exploiting the signal *AAoutOR*, in fact, is possible to write a unique procedure for dealing with the writing requests from the external world, independently on the number of words stored inside the array and on the particular output line from the decoder that has been activated. More details on the structure of the FSM will be given in section 6.3.5.

As it can be observed from the structure of the FSM, to perform the writing operation of a single word, in general N_{bit} clock cycles are needed. During these clock cycles the output of the decoder must remain stable, so that the signal *AAoutx* is kept active through the whole operation. The array is able, in principle, to write three words at the same time, but to do so there would be the need of a decoder able to activate three output lines at the same time (that is, a decoder with N input ports, in general, where N is the number of words stored in the memory).

6.3.5. FSM-array adapter and FSM structure

An additional block has been developed with respect to the ones proposed in [37]. The aim of this block is to generate, starting from the few signals activated by the FSM and from the output signals of the block *and array*, all the control signals that are necessary to actually drive the memory array during its tasks. The FSM machine in figure 6.15, in fact, asserts only generic signals, which in most of the cases never arrive directly to the component that needs them. These signals are asserted according to the operations to be done, assuming that there is always at least one memory cell that needs them. Whether the actual signals will be activated starting from these generic signals, it depends only on the decisions performed by this block, shown in figure 6.13.

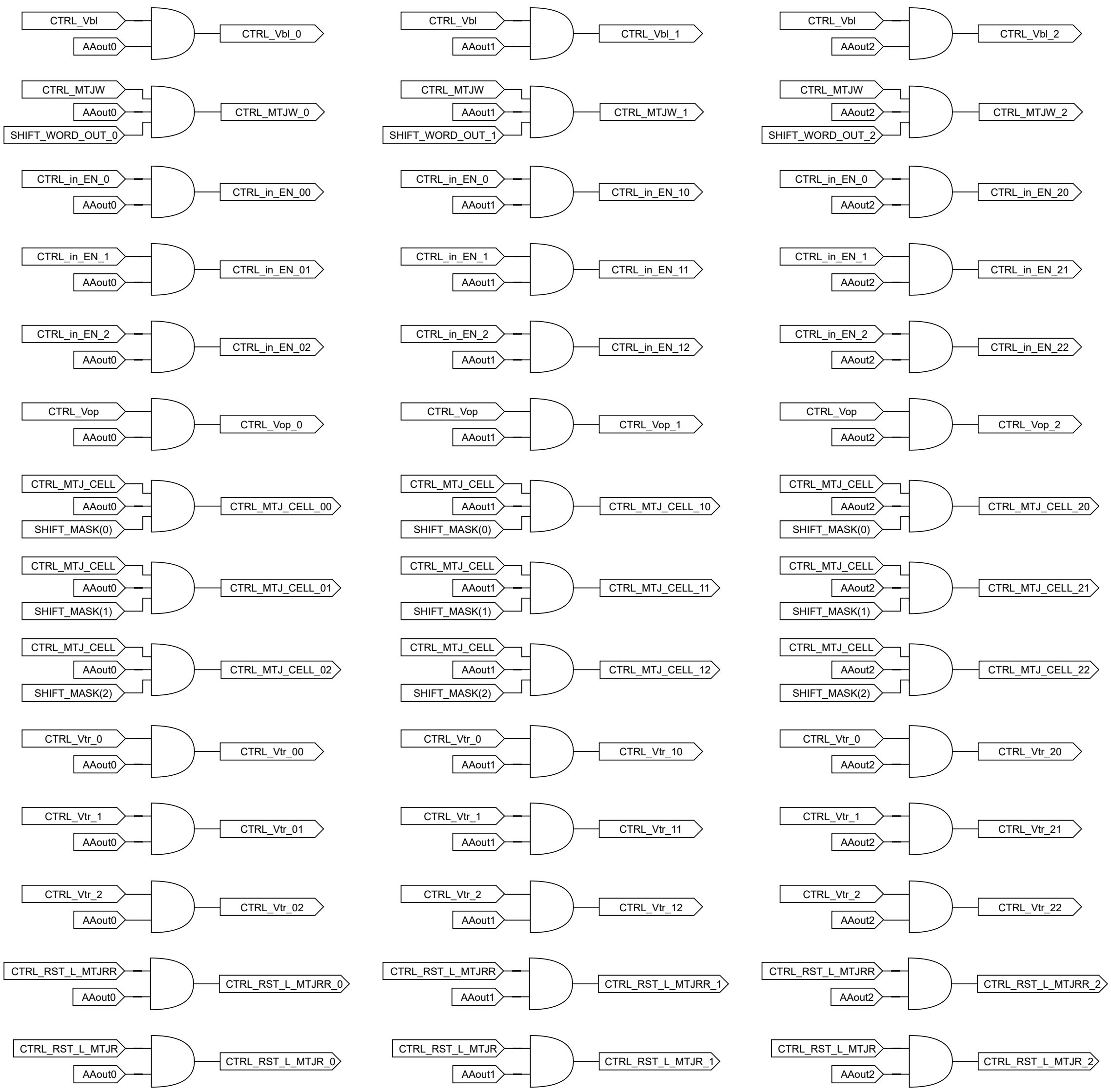


Figure 6.13. Structure of the *FSM-array adapter*, needed to translate the generic signals activated by the FSM into the actual signals needed by the memory array.

This block is simply an array of *AND* gates, which receive on one side the generic signal activated by the FSM, and on the other side the signal *AAoutx*, activated by the block *and array* and needed to know on which memory word the action requested must be performed.

To guide the reader in the analysis of the signals that are generated by this block, a description state-by-state of the actions performed by the FSM is offered in the following. The order in which these states will be described follows the order in which each operation is logically placed during the use of the memory: this means that first the reset state, then the initialization of the memory, then the states related to the *find* mode, and finally the states linked to a read operation will be described.

reset During the reset state all the memory elements present inside the architecture are reset: this means that the signals asserted are

- *CTRL_WORD_RST*
- *CTRL_MASK_RST*
- *CTRL_RST_L_MTJRR*
- *CTRL_RST_L_MTJR*
- *CTRL_LATCH_RST*
- *CTRL_ADDREG_RST*
- *CTRL_READWORD_RST*

Among these signals, both *CTRL_RST_L_MTJRR* and *CTRL_RST_L_MTJR* pass through the *FSM-array adapter* before being sent to the components that need them, while all the remaining signals are provided directly. The aim in asserting both *CTRL_RST_L_MTJRR* and *CTRL_RST_L_MTJR* is to reset the latches inside the memory array that store the signal coming out from the read heads. During the reset state, also the latches inside the *row enabler* are reset: this means that, even if the decoder is not activating any output line yet, and even if *FIND* is still equal to 0, since the signal *RESET_n* coming from the external is asserted, all the signals *AAoutx* will become active, thus allowing the reset of the latches inside the memory array (their reset signal is put in *AND* with the proper *AAoutx* by the *FSM-array adapter*). As shown in figure 6.13, three *AND* gates receive the signal *CTRL_RST_L_MTJRR*, and put it in *AND* together with either *AAout0*, *AAout1* or *AAout2* to determine the signals *CTRL_RST_L_MTJRR_0*, *CTRL_RST_L_MTJRR_1* and *CTRL_RST_L_MTJRR_2*; a similar thing happens with the signal *CTRL_RST_L_MTJR*. In this way the FSM activates only two signals, but since the memory array hosts three words, actually six different signals will be asserted at the same time, since all of the three signal *AAoutx* are active.

S0 The FSM remains in the reset state as long as the signal *RESET_n* is asserted. When this signal becomes inactive, the FSM moves to the state *S0*. This state is an idle state, where the FSM waits for new requests, which could be a read, a write or a find operation. This state will be discussed more than once, respectively in states *S9*, *S4* and *S6*, to prove how the signals that are asserted here are used to complete each operation and restore back the employed registers to their default value. Here a list of the signals that are activated is provided, to prove how, after the *reset* state, no relevant action is performed.

- *IDLE*: the idle signal is activated.
- *CTRL_RST_L_MTJRR*, *CTRL_RST_L_MTJR*: they are the reset signals of the latches needed for storing the output of the read heads inside the array. Even if *FIND*, the decoder outputs and *RESET_n* are all inactive, thanks to the activation of the signal *IDLE* the latches are able to receive a reset impulse; the latches, however, have already been initialized in the *reset* state, so actually no relevant action is performed.
- *CTRL_LATCH_RST*: the latches inside the row disabler receive their reset signal directly, so they will be kept in the reset state also during *S0*.
- *CTRL_MASK_STORE*: by activating this signal, the mask register stores the value fixed at its input, that is, all bits equal to 0 apart from the MSB, which is set to 1. Doing so, the register is now ready to provide the value to the memory array, as soon as a search operation is requested.

Before going on with the description of the remaining states, a disclaimer must be inserted here. It should be very clear that the structure of a FSM is tightly linked to the behaviour of the components which form the datapath that the machine controls. If the timing of one component changes, also the FSM structure must change as well. Now, the aim of this and of the previous chapter has been the project of a LiM array based on skyrmions; in the previous chapter the project has been tested in VHDL, and a VHDL description is offered also for the array discussed in this chapter. This VHDL description always tries to remain as close as possible to the actual behaviour of the components that would be used in a physical realization of the array, but this may be not always possible, depending on the particular situation. In this case, for example, great difficulties arise from the desire of describing in a behavioural way the mechanism of nucleation and of movement of the skyrmions along the racetrack, until each skyrmion stops exactly in correspondence of the cell that must elaborate it. The main difficulty arises exactly from the desire of a behavioural description: if the skyrmions were described as in the gates used in chapter 4, in fact, the problem wouldn't be so difficult to be solved.

The components that raise the largest problem in this description are the ones labelled as *Dxx*. These are not even actual components, in the physical realization of the array: they would simply be a point in the racetrack where a number of paths arrive and where the skyrmion is able to take one direction or the other, according to the voltage that is turned on. This is where the difficulties begin. The skyrmions are described in VHDL as pulses imposed on a signal: this said, the behavioural VHDL description of the components *Dxx* becomes very tricky. This description is such that, after detecting a skyrmion on one of their inputs, they store it inside themselves in correspondence of the first rising edge on one of the three currents they are sensitive to (either the racetrack current, or the current due to the activation of either V_{opx} or V_{trxx}); then, a new rising edge on one of these three currents makes the skyrmion inside the component go out from the selected output. This kind of description is inevitable if the choice is to describe the component from a behavioural point of view: in an actual physical component, in fact, there would be no need for these sequential activations and deactivations of voltage controls, because the timing of the operation is determined uniquely by the velocity of the particle: these controls could even be turned on all at once and the result of the operation would still be correct. At a behavioural level, where the skyrmions are simply pulses and no propagation mechanism inside the components of the array is simulated, this kind of description for the *Dxx* is really inevitable; the timing of the operation must then be determined by the succession of the events on the control signals, imposed by the FSM, since it cannot be decided by the movement of the skyrmion. This is the first reason why the VHDL description of this array starts to be less adherent to the reality.

In a physical realization of the racetrack, it would be enough to nucleate three skyrmions one after the other, tuning the time distance between the impulses on the writing head depending on the amount of current imposed by the activation of V_{bl} and on the velocity of each skyrmion; at the same time, applying a single impulse on V_{bl} (long enough to allow the complete movement of the skyrmions), each particle would reach the corresponding cell input, without any kind of difficulty. A behavioural VHDL description of this mechanism is almost impossible: the *Dxx* VHDL components, first of all, are all connected to the same input signal, that is the output of the writing head at the beginning of the corresponding racetrack: this means that, as soon as a skyrmion is nucleated, they all sense it at their input, all at the same time. Even if some delay lines where put between one *Dxx* component and the other, in order to introduce a delay on the detection of the input skyrmion, the problem wouldn't be solved, because they also sense the activation of the same current (linked to V_{bl}) all at the same time, and the activation of this current messes up the behavioural description of the component. For this reason, the only way to obtain a description as close as possible to the actual physical realization of the array is to do the following: first the MSB is nucleated, moved along the racetrack and then placed inside the component *Dx2*. Then this component is deactivated: this means that it receives an enable signal, and when this signal is not asserted the component is not able to detect neither an input skyrmion, nor a rising edge on any of its input currents. Then a second skyrmion is nucleated, moved along the racetrack and stored inside *Dx1*, which is then deactivated. Finally, the last skyrmion is nucleated, moved and stored inside *Dx0*: only at this point the other two *Dxx* component can be enabled again, ready to allow the skyrmion to go out from one of the three possible outputs.

As a result, the FSM is a bit different from the one that would control the equivalent physical array: the consequences of these differences are in the need of introducing the signals *CTRL_in_EN_2* and *CTRL_in_EN_1*, whose meaning is completely unrelated from the physical realization of the array, and in the need of applying a series of pulses on the signal *CTRL_Vbl* during each state. Of course, it is possible to modify the description of the components *Dxx*, maybe making them less "behavioural" and more adherent to the reality of the facts: doing so also the FSM would become more adherent to the machine that would control the actual physical array.

Having said this, is now possible to discuss the remaining states, particularly those involved into a writing operation, with a slightly deeper knowledge about the reason why some signals are used.

S7 The first operation that must be done in order to use the memory is to initialize it. To do so, an address must be provided from the external, together with the word to be written inside the array, and the address must be maintained stable at the input of the decoder until the write operation is completed. Moreover, the memory input *READ_WRITE_n* must be driven accordingly from the external. After a new word has been stored in one of the registers shown in figure 6.5, the decoder activates one of the lines *DEC0*, *DEC1* or *DEC2* in figure 6.12: as a consequence the corresponding *AAoutx*, which up to now was equal to 0, becomes active, since the corresponding latch inside *row enabler* is still reset to 0. Differently from what happened in the *reset* state, this time only one *AAoutx* at the time will be activated, unless the decoder has more than one input port, able to accept more than one address at the same time. The activation of *AAoutx* makes *AAoutOR* (figure 6.12) switch to 1, so the FSM enters state *S7*.
During this state, the FSM activates the signals *CTRL_MTJW* and *CTRL_-Vbl*. *CTRL_MTJW* is needed for allowing the nucleation of a skyrmion at the beginning of the selected racetrack, according to the value of the MSB inside the word register of reference. As figure 6.13 shows, the three signals *CTRL_MTJW_0*, *CTRL_MTJW_1* and *CTRL_MTJW_2* are generated by computing the *AND* of *CTRL_MTJW*, of *AAoutx* and of the bit coming out from the word register of competence: each of the signal *CTRL_MTJW_x* then can become active only if the corresponding memory racetrack has been chosen as a destination for the writing operation (*AAoutx* is active) and if the bit that must be represented is a 1 (*SHIFT_WORD_OUT_x* is 1): if the bit to be represented is 0, in fact, no nucleation must be performed.
Thanks to the activation of *CTRL_Vbl* (which is again put in *AND* with *AAoutx*), the skyrmion that has just been nucleated can then move along the racetrack. Five pulses are imposed on this signal, in order to allow the MSB to reach the component *Dx2*.
During the *S7* state, also the signal *CTRL_WORD_SHIFT* is activated. This has as consequence the shift of all the word registers towards left by one position. There is no possibility, in fact, that these registers may contain

any kind of information that must be preserved: if more than one register is storing some kind of information, it means that this information is being used in the current writing step and has to be properly shifted. At the end of the writing step the memory will be ready to accept new data inside these registers, together with the address(es) where this data must be stored. Shifting the word registers involved in the write operation, of course, has the aim of making available for the next writing step, performed in state *S8*, the second bit to be written inside the racetrack.

S8 The operations performed in this state are very similar to the one already discussed for state *S7*. First of all, thanks to the deactivation of the component *Dx2*, a new skyrmion can be nucleated and moved without affecting the information already stored inside the racetrack. So, a new pulse is applied to *CTRL_MTJW*, in order to nucleate the $MSB - 1$ bit of the word, according to the value that is coming out from *SHIFT_WORD_OUT_x*; *CTRL_Vbl* experiences three consecutive pulsed, so that the skyrmions moves along the racetrack until it reaches the component *Dx1*; finally, the word registers are shifted once more towards left.

S9 This is the last state needed for performing the writing of a word inside the memory array. Now both *CTRL_in_EN_2* and *CTRL_in_EN_1* are set equal to 0, so that both *Dx2* and *Dx1* are deactivated. In this way it is possible to nucleate a new skyrmion by activating once more the signal *CTRL_MTJW*; then, by imposing a single pulse on *CTRL_Vbl*, this skyrmion is stored inside the component *Dx0*. This completes the writing operation. One final shift is imposed to the word registers: doing so, either the value initially stored in the register is completely shifted out, so that the register now is filled with zeros, or, if the register were modified with the introduction of a serial input and a serial output, with this final shift the initial word would be fully restored inside it.

Coming back to state *S0*, let's look again the signals that are asserted in that state:

- *IDLE*: the idle signal is activated again.
- *CTRL_LATCH_RST*: the latches inside the row disabler will be reset

again; however, they don't contain any relevant information yet, so this operation has no consequence.

- *CTRL_RST_L_MTJRR*, *CTRL_RST_L_MTJR*: the latches connected to the read heads inside the memory array, again, sense the activation of their reset signals, because even if *FIND* is still 0, *RESET_n* is no more active and the outputs of the decoder have already been turned off (the writing operation is over), since the *IDLE* signal is activated and the latches inside *row disabler* are reset, then all the signals *AAoutx* are allowed to switch.
- *CTRL_MASK_STORE*: the mask register stores again the value fixed at its input. It was already containing exactly that same value, so no relevant action is performed.

S1 If, during the idle state that is *S0*, the activation of the signal *START_FIND*, coming from the external, is detected, the machine moves to state *S1*. The first action that is performed is the activation of the signal *FIND*: doing so, as shown in figure 6.12, all the signals *AAoutx* will switch to 1 (since all the latches inside *row disabler* are still set to 0), thus enabling the following actions to be performed on all the memory words in parallel.
In order to start the search of the minimum/maximum word, the value of the mask must be distributed along all the columns inside the array. To do so, the FSM activates the signal *CTRL_MTJ_CELL*: this signal is then filtered by the *FSM-array adapter* block, which puts it in *AND* with the signals *AAoutx* and with the bit from the mask that must be written inside each cell. Let's consider for example the MSB of the mask, equal to 1: this bit is used in the *AND* that determines the value of *CTRL_MTJ_CELL_02*, of *CTRL_MTJ_-CELL_12* and of *CTRL_MTJ_CELL_22*; since all of the *AAoutx* signals, at this stage, are equal to 1, these three signals will be all set to 1 on their turn. This doesn't happen for the remaining *CTRL_MTJ_CELL_xx*, instead: even if *CTRL_MTJ_CELL* is active, together with the respective *AAoutx*, all the remaining bits of the mask are equal to 0, and so no skyrmion is nucleated by the write heads. As a result, only cell 20, cell 12 and cell 22, dedicated to the elaboration of the MSB of the three numbers, will have a skyrmion nucleated by the write head.

During this same state, also the signal *CTRL_Vop* is applied: this has the effect of turning on the current that allows the skyrmion stored inside each racetrack (also *CTRL_Vop* is manipulated by an *AND* with *AAoutx*, but every *AAoutx* is equal to 1 now) to go out from the *Dxx* component where it is placed and enter the cell where the elaboration will be performed. This same current pushes both this skyrmion and the skyrmion just nucleated towards the input of the *AND/OR* gate. Before the elaboration can start, a peak of current must be applied on *CURRENT_Vop*: it is of vital importance for the correct functioning of the *AND/OR* gate, in fact, that the couples of skyrmions are perfectly synchronized at the input: for this reason a couple of notches has been inserted at the input of each gate. Finally, thanks again to the voltage V_{opx}, each skyrmion is pushed through the gate, and the results of each *AND* are collected by the chain of *join* components, all the way towards the read head *MTJ_RRx*, where the presence or the absence of the skyrmion, according to the value of the MSB of each number, is detected. As soon as the output of this read head switches (if it switches), it is latched inside one of the memory elements shown in figure 6.6: as a consequence, the two *AND* trees of figure 6.9 decide whether the *row enabler* must be activated, and if this is the case, some of the latches inside figure 6.10 will be set to 1, thus disabling the corresponding word from the future comparisons.

The final action that is done during this state is the shift towards right by one position of the content inside the mask register: in this way, the register will be ready, in the next step, to provide the correct bit to any of the cells inside the array that must receive it.

One more signal, *CTRL_Vbl*, is activated during this state: the aim in doing this is allowing the skyrmion produced by the duplicating element inside each cell (according to the value of the input skyrmion) to be stored inside the component *Dxx*. Again, the activation of this signal is due only to the particular VHDL description that has been adopted for that component, and would have no real correspondence in a physical realization of the array.

S2 The first action that is performed in this step is to impose a pulse on the signal *CTRL_RST_L_MTJRR*: this reset won't be sensed by all the latches that were involved in the last step, because now some latches inside *row disabler*

may have been set to 1, masking the activation of this signal to the columns that have been discarded. However, even if those latches are not reset now, it is not a problem, because their output will be ignored from now on. They will be reset only at the end of the search operation, together with all the latches inside the array.
The remaining operations performed in this and in the following state are very similar to what already discussed: new skyrmions are nucleated only in the enabled columns according to the pattern described by the mask register, which is then shifted by one position. V_{opx} is turned on again, allowing the skyrmion to move and to reach the read head: then, according to the requests of the search, each of the columns that are currently being considered may or may not be disabled.

S3 The actions performed in this state are equal to the ones that appear in state *S2*, so they won't be discussed any further. It must be underlined, however, that in an array containing numbers represented on three bits, this is the final search step: at the end of this state the latches inside *row disabler* will contain the indication of the address of the word(s) found. For this reason, all the signals that must be activated from now on have the aim of concluding the search operation and of providing the result to the external, restoring at the same time the registers inside the memory, in order to be ready for a new operation.

S4 During this state is it activated the signal *CTRL_ENCODER_EN* of figure 6.11: as soon as its enable is turned on, its output won't be anymore equal to a high-impedance value, but will encode the address of one of the latches still equal to 0 it sees in its input. So, now the register shown in figure 6.11 will be able to transmit to the external the value of the address of the word found, provided on its input by the encoder: for this reason also the signal *CTRL_ADDREG_STORE* is turned on. During this state, finally, a new impulse is provided on *CTRL_RST_L_MTJRR*, so as to reset the latches that participated in the detection of the skyrmions in the last search step.
This is the last state involved in the writing operation. Now the machine comes back to the idle state *S0*, with all the latches that still need to be reset.

Let's look again at the signals that are activated during *S0*:

- *IDLE*: the idle signal is activated again.
- *CTRL_LATCH_RST*: activating this signal, the latches inside the *row disabler* are all reset back to 0. Their reset signal in fact comes directly from the FSM, without passing through the *adapter* block, so it will be detected by all the latches at the same time. Since from now on all the latches will contain again the same value, all the future actions will have the same effect throughout the whole memory structure, because no *AAoutx* signal will be able to mask them anymore.
- *CTRL_RST_L_MTJRR*, *CTRL_RST_L_MTJR*: the signals *FIND*, *RESET_n* and all the outputs from the decoder are inactive; however, since *IDLE* is asserted, and since all the latches inside the *row disabler* are being reset right now, all the signals *AAoutx* are able to experience an impulse. This allows the signals *CTRL_RST_L_MTJRR* and *CTRL_RST_L_MTJR* to be sensed by all the latches contained inside the memory array: in this way they are all reset to their default value.
- *CTRL_MASK_STORE*: the mask register stores again the value fixed at its input. In this way it will be ready for a new search operation, if desired.

S6 The last request the memory must be able to satisfy is a read request. In order to read a memory word an address must be provided externally, and this address must remain fixed at the input of the memory until the operation is over. When the address has been provided at the input of the decoder, which has also been enabled by the external world, the activation of the signal *READ_-WRITE_n* (which must be equal to 1 to signify that a reading is requested) makes the machine enter the reading routine. At this point, since the latches inside *row disabler* have been all reset to 0, but since only one signal among *DEC1*, *DEC2* and *DEC3* is enabled, then only one of the signals *AAoutx* will switch to 1: as a result, the following operations will be allowed only on the selected memory word, while all the remaining words inside the memory array won't be affected.

During this state, the signals *EN_READ* and *CTRL_Vop* are activated. It should be easy to guess, at this point, that *CTRL_Vop* will allow only the

skyrmions inside the selected racetrack to move towards right, inside the elaboration area, crossing the notch at the top input of the *AND/OR* gate, going through the gate itself, coming out from the *OR* output, up to the read head placed right at the output of the *OR* gate. The output of this read head is latched inside a memory element, since the signal *EN_READ*, which appears in figure 6.7, is enabled as well. As a result, only the latches related to the memory word selected will switch according to the pattern detected; finally, since the signal *AAoutx* linked to that particular word is turned on (because the address of the word must be maintained fixed at the input of the decoder until the read is over), the multiplexer of figure 6.7 will allow the output of the group of latches selected to pass through. Finally, this information is latched inside the register *READ_WORD*, so that the external world can read the requested word.
Then the machine comes back to state *S0*. Again, the signals that are activated here are:

- *IDLE*: the idle signal is activated again.
- *CTRL_LATCH_RST*: the latches inside the *row disabler*, already containing 0, are all reset again. Thanks to the activation of the *IDLE* signal, all the *AAoutx* signals will switch to 1.
- *CTRL_RST_L_MTJRR*, *CTRL_RST_L_MTJR*: since all the signals *AAoutx* are now equal to 1, the reset is sensed equally by all the latches present inside the array. In this way all the latches that participated to the previous operation are again restored back to 0.
- *CTRL_MASK_STORE*: the mask register stores again the value fixed at its input, even if the read operation hasn't changed the value already stored.

6.4. Conclusions

This concludes the description of this second LiM array. The strengths of this architecture with respect to the one presented in chapter 5 are for sure the simplicity of the structure and of the FSM that controls it. In particular, thanks to all the control blocks described in section 6.3, the array is completely independent on the number of words it hosts, while of course some small changes (which anyway shouldn't be hard to be made) are needed in order to adapt the control blocks to an array able to host words represented on a different number of bits.

The main weakness of this structure with respect to the LiM array of chapter 5 is the degraded power efficiency: this time, in fact, all the skyrmions that are no more needed in the computing cycle are all expelled from the nanotrack, sooner or later, without being accumulated in a structure able to put them again in movement according to the requests, as it was the tank (top and bottom) proposed in chapter 5.

The information is never lost: every time that a skyrmion is needed from the racetrack, its value is readily restored back by using the duplication elements inserted inside each cell. This doesn't involve the nucleation of a new skyrmion, a process which is energetically expensive; however, the duplication elements exploit the skyrmion-DW pair-skyrmion conversion proposed in [43], and it is a fact that the DW pairs need higher current densities with respect to skyrmions, in order to be put in movement. So, the energetic efficiency of the array could be improved. Any improvement, however, would also complicate the structure and FSM that controls it. As it always happens in electronics, then, it all reduces to a trade-off, in this particular case between energetic efficiency and complexity.

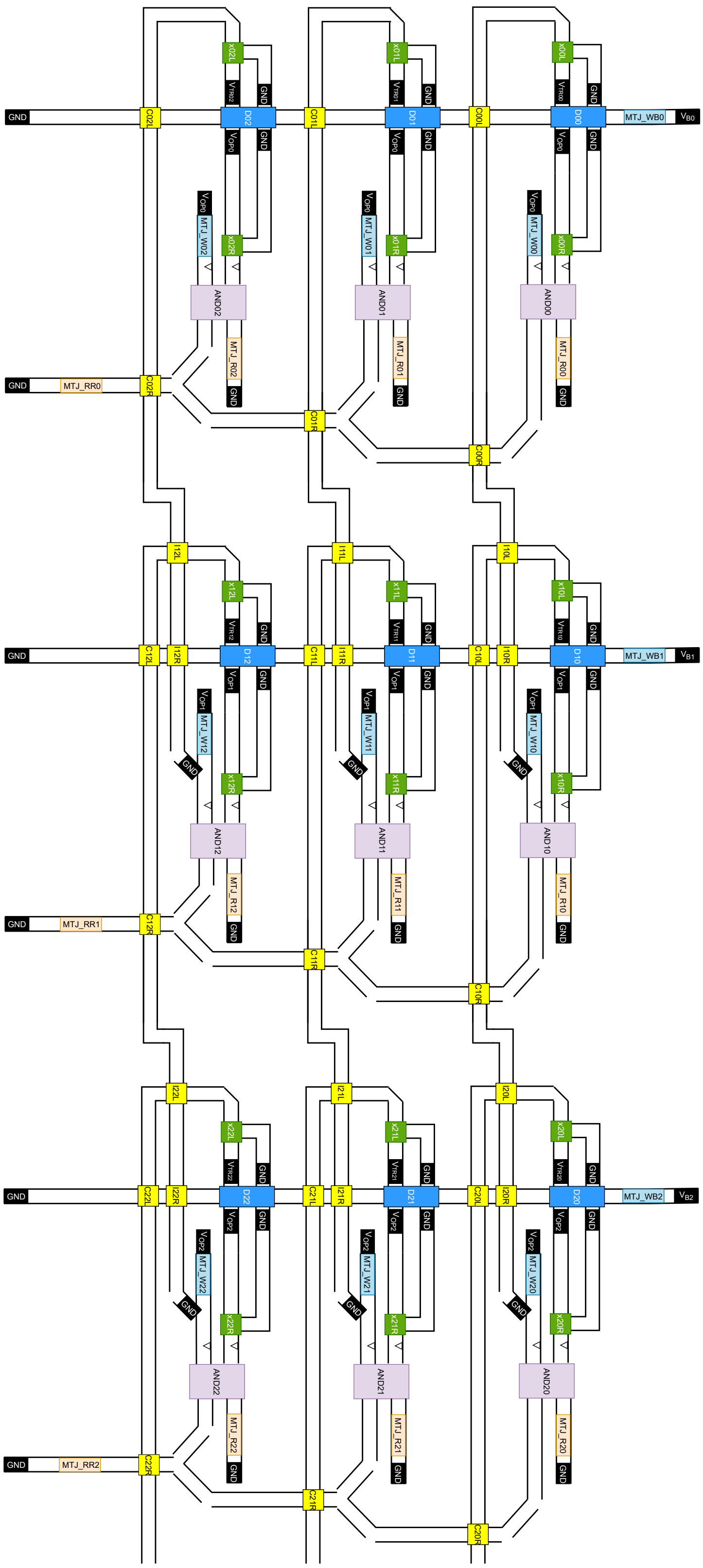


Figure 6.14. Datapath of the memory array showing the structure and connections of nine memory cells.

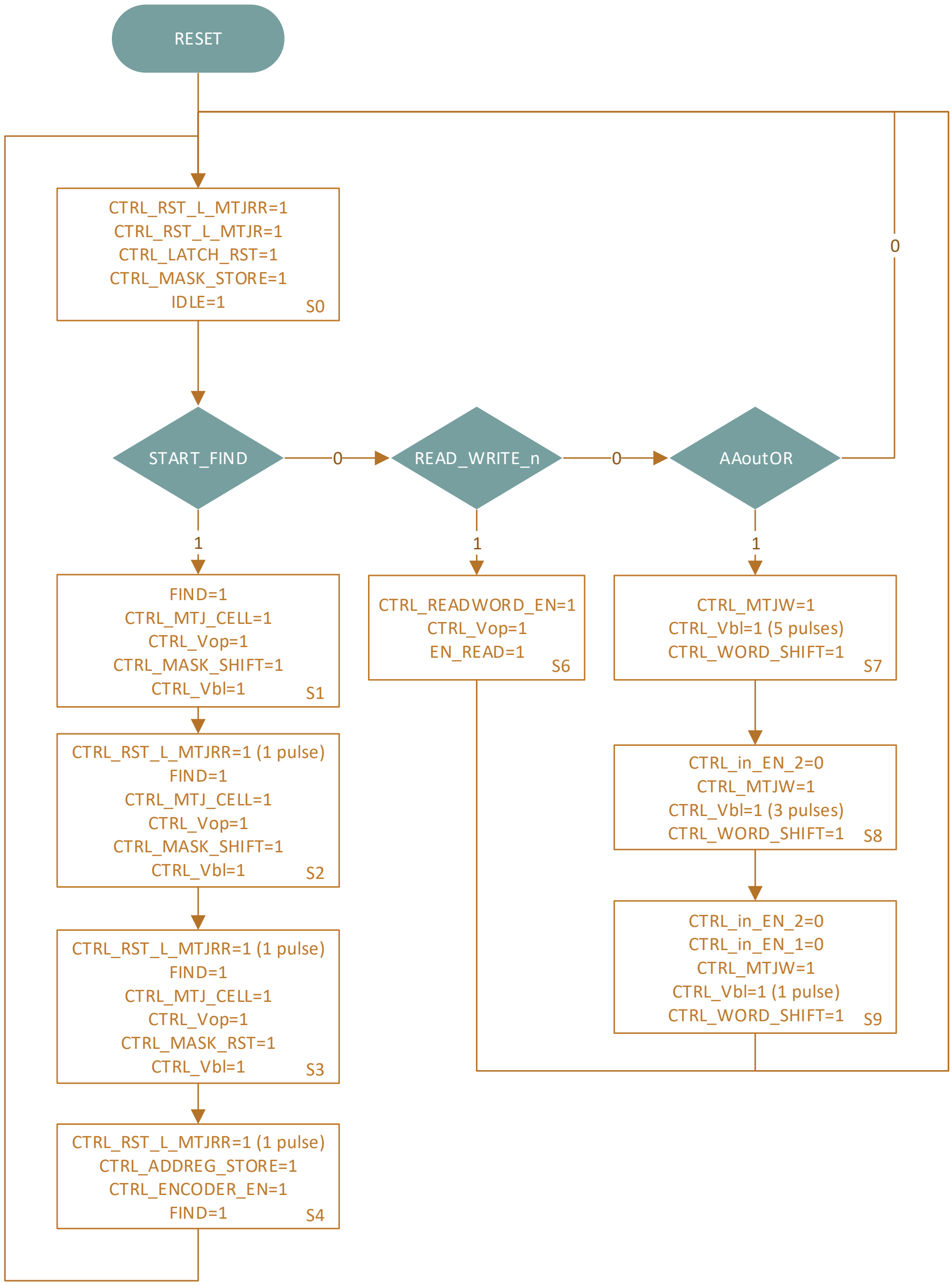


Figure 6.15. FSM controlling the whole memory datapath.

7. Summary and prospects for future studies

Starting from the logic gates proposed in [5], a micromagnetic analysis has been performed in order to verify their correct functioning in realistic current distribution conditions. The gates that have been tested are only the *AND/OR* and *INV/COPY* gates: no test has been performed yet on the synchronization element and on the full adder, which puts all these structures together. The testing of the full adder structure is for sure the most critical to be performed, not only for the huge computational resources that would be needed, but also for the need of tuning the physical dimensions of each element in order to assure the synchronization of the skyrmions inside each gate. The notches proposed in [5] can in fact reduce the synchronization problem, but have not the power to completely eliminate it: as showed in chapter 3, tiny differences in the size of each gate imply huge differences in the behaviour of the skyrmions that are hosted.

Different structures have been developed starting from these gates, whose behaviour has been proved correct even in realistic current conditions. First of all, a ripple carry adder has been designed and simulated. Most of the efforts have been spent in optimizing the structure proposed, in order to reduce as much as possible the number of skyrmions to be nucleated. The methodology adopted during this design phase, however, can be extended to the project of any type of electronic component based on skyrmions.

Finally, the topic of logic in memory architectures has been investigated. Two different versions of LiM arrays have been proposed, the former more general and flexible, the latter optimized for the execution of a particular algorithm. Both structures have been described in VHDL only from a behavioural point of view: one possible evolution of this work could be a more accurate description, which takes into account also the simulation of the skyrmion's movement inside the array structure.

Many other different LiM structures could be designed as well, following the same steps and reasoning adopted in this work. Since the physics of skyrmions is very rich, a large variety of mechanisms is available for manipulating them in the most different ways. For this reason, during the design phase is possible to find in literature the solution to almost any problem, thanks to the huge variety of applications that have already been studied and published.

Another possibility, finally, could be to start from the second LiM array, proposed in chapter 6, maybe slightly modifying it to make it more powerful and generic, and to use it in order to support the execution of other algorithms, apart from the minimum/maximum search algorithm already discussed.

Some issues, however, still need to be solved: the key assumption made behind this whole thesis work, in fact, is the possibility of allowing two nanotracks to cross without altering the information (the skyrmions) carried inside each of them. To date, no solution is yet available in literature around this topic, which must be solved before anything else, because otherwise even the structure of the ripple carry adder of chapter 4 would be impossible to be implemented.

Another assumption made in this thesis that should be verified from a physical point of view, finally, concerns the possibility to separate two metal traces with some dielectric in between without significantly altering the DMI, responsible for the stabilization of the skyrmion. As already explained in chapter 5, in fact, in some cases there is the need to apply two different voltage signals to two metal contacts placed one next to the other, in order to control the movement of the skyrmion according to the requests of the algorithm that is being executed. If this assumption were proved wrong, some other mechanisms for dynamically controlling the skyrmion motion should be found, because otherwise both LiM arrays would be impossible to be used.

Bibliography

[1] Magnetic vortices, skyrmions, etc. Accessed: 2019-07-30. URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwj4gbDA69zjAhVCjqQKHbFeDJoQFjAAegQIAhAC&url=http%3A%2F%2Fwww.sbfisica.org.br%2F~ebm%2Fix%2Farquivos%2FP13.pdf&usg=AOvVaw0iyBgAqvvkiFLNfWxaCbVS.

[2] Spintronics. Accessed: 2019-07-29. URL: http://asdn.net/asdn/electronics/spintronics.php.

[3] Hans-Benjamin Braun. Topological effects in nanomagnetism: from superparamagnetism to chiral quantum solitons. *Advances in Physics*, 61(1):1–116, 2012. doi:10.1080/00018732.2012.663070.

[4] Lorenzo Camosi, Nicolas Rougemaille, Olivier Fruchart, Jan Vogel, and Stanislas Rohart. Micromagnetics of antiskyrmions in ultrathin films. *Phys. Rev. B*, 97:134404, Apr 2018. doi:10.1103/PhysRevB.97.134404.

[5] Maverick Chauwin, Xuan Hu, Felipe Garcia-Sanchez, Neilesh Betrabet, Christoforos Moutafis, and Joseph S. Friedman. Conservative Skyrmion Logic System. *arXiv e-prints*, page arXiv:1806.10337, Jun 2018. arXiv:1806.10337.

[6] Xing Chen, Wang Kang, Daoqian Zhu, Xichao Zhang, Na Lei, Youguang Zhang, Yan Zhou, and Weisheng Zhao. Skyrmion dynamics in width-varying nanotracks and implications for skyrmionic applications. *Applied Physics Letters*, 111(20):202406, 2017. doi:10.1063/1.5005953.

[7] J M D Coey. New permanent magnets; manganese compounds. *Journal of Physics: Condensed Matter*, 26(6):064211, jan 2014. doi:10.1088/0953-8984/26/6/064211.

[8] Albert Fert, Vincent Cros, and João Sampaio. Skyrmions on the track. *Nature Nanotechnology*, 8:152 EP –, Mar 2013. URL: https://doi.org/10.1038/

nnano.2013.29.

[9] Giovanni Finocchio, Felix Büttner, Riccardo Tomasello, Mario Carpentieri, and Mathias Kläui. Magnetic skyrmions: from fundamental to applications. *Journal of Physics D: Applied Physics*, 49(42):423001, sep 2016. URL: https://doi.org/10.1088%2F0022-3727%2F49%2F42%2F423001, doi:10.1088/0022-3727/49/42/423001.

[10] H. Fook, C. A. C. Ian, W. Gan, I. Purnama, and W. Lew. Mitigation of magnus force in current-induced skyrmion dynamics. In *2015 IEEE International Magnetics Conference (INTERMAG)*, pages 1–1, May 2015. doi:10.1109/INTMAG.2015.7157020.

[11] Fereshte Ghahari, Daniel Walkup, Christopher Gutiérrez, Joaquin F. Rodriguez-Nieva, Yue Zhao, Jonathan Wyrick, Fabian D. Natterer, William G. Cullen, Kenji Watanabe, Takashi Taniguchi, Leonid S. Levitov, Nikolai B. Zhitenev, and Joseph A. Stroscio. An on/off berry phase switch in circular graphene resonators. *Science*, 356(6340):845–849, 2017. doi:10.1126/science.aal0212.

[12] Utkan Güngördü, Rabindra Nepal, Oleg A. Tretiakov, Kirill Belashchenko, and Alexey A. Kovalev. Stability of skyrmion lattices and symmetries of quasi-two-dimensional chiral magnets. *Phys. Rev. B*, 93:064428, Feb 2016. doi:10.1103/PhysRevB.93.064428.

[13] Markus Hoffmann, Bernd Zimmermann, Gideon P. Müller, Daniel Schürhoff, Nikolai S. Kiselev, Christof Melcher, and Stefan Blügel. Antiskyrmions stabilized at interfaces by anisotropic dzyaloshinskii-moriya interactions. *Nature Communications*, 8(1):308, 2017. doi:10.1038/s41467-017-00313-0.

[14] Siying Huang, Chao Zhou, Gong Chen, Hongyi Shen, Andreas K. Schmid, Kai Liu, and Yizheng Wu. Stabilization and current-induced motion of antiskyrmion in the presence of anisotropic dzyaloshinskii-moriya interaction. *Physical Review B*, 96, 09 2017. doi:10.1103/PhysRevB.96.144412.

[15] Siying Huang, Chao Zhou, Gong Chen, Hongyi Shen, Andreas K. Schmid, Kai Liu, and Yizheng Wu. Stabilization and current-induced motion of antiskyrmion in the presence of anisotropic dzyaloshinskii-moriya interaction. *Phys. Rev. B*, 96:144412, Oct 2017. doi:10.1103/PhysRevB.96.144412.

[16] Junichi Iwasaki, Masahito Mochizuki, and Naoto Nagaosa. Current-induced

skyrmion dynamics in constricted geometries. *Nature Nanotechnology*, 8:742 EP –, Sep 2013. Article. URL: https://doi.org/10.1038/nnano.2013.176.

[17] Junichi Iwasaki, Masahito Mochizuki, and Naoto Nagaosa. Universal current-velocity relation of skyrmion motion in chiral magnets. *Nature Communications*, 4:1463 EP –, Feb 2013. Article. URL: https://doi.org/10.1038/ncomms2442.

[18] Wanjun Jiang, Xichao Zhang, Guoqiang Yu, Wei Zhang, Xiao Wang, M. Benjamin Jungfleisch, John E. Pearson, Xuemei Cheng, Olle Heinonen, Kang L. Wang, Yan Zhou, Axel Hoffmann, and Suzanne G. E. te Velthuis. Direct observation of the skyrmion hall effect. *Nature Physics*, 13:162 EP –, Sep 2016. Article. URL: https://doi.org/10.1038/nphys3883.

[19] Øyvind Johansen. Electric Control of Skyrmion Dynamics and Spin Torque Oscillators in Magnetic Materials with Inversion Asymmetry. Master's thesis, Norwegian University of Science and Technology, 2016. URL: http://hdl.handle.net/11250/2405162.

[20] W. Kang, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao. Skyrmion-electronics: An overview and outlook. *Proceedings of the IEEE*, 104(10):2040–2061, Oct 2016. doi:10.1109/JPROC.2016.2591578.

[21] Wataru Koshibae and Naoto Nagaosa. Theory of antiskyrmions in magnets. *Nature Communications*, 7:10542 EP –, Jan 2016. Article. URL: https://doi.org/10.1038/ncomms10542.

[22] Shizeng Lin, Avadh Saxena, and Cristian Batista. Meron crystals in chiral magnets. 06 2014. URL: https://www.researchgate.net/publication/262919936_Meron_crystals_in_chiral_magnets.

[23] J. Ping Liu, Zhidong Zhang, and Guoping Zhao. *Skyrmions: Topological Structures, Properties, and Applications*, chapter 8, page 213. CRC Press, 2016.

[24] J. Ping Liu, Zhidong Zhang, and Guoping Zhao. *Skyrmions: Topological Structures, Properties, and Applications*, chapter 1, page 9. CRC Press, 2016.

[25] J. Ping Liu, Zhidong Zhang, and Guoping Zhao. *Skyrmions: Topological Structures, Properties, and Applications*, chapter 1, 8, pages 27, 213. CRC Press, 2016.

[26] J. Ping Liu, Zhidong Zhang, and Guoping Zhao. *Skyrmions: Topological Structures, Properties, and Applications*, chapter 8, pages 227–228. CRC Press, 2016.

[27] Ye-Hua Liu and You-Quan Li. Dynamics of magnetic skyrmions. 24(1):017506, 2015. doi:10.1088/1674-1056/24/1/017506.

[28] Shijiang Luo, Min Song, Xin Li, Yue Zhang, Jeongmin Hong, Xiaofei Yang, Xuecheng Zou, Nuo Xu, and Long You. Reconfigurable skyrmion logic gates. *Nano Letters*, 18(2):1180–1184, 2018. PMID: 29350935. doi:10.1021/acs.nanolett.7b04722.

[29] Mauricio Manfrini. *Spin orbit torques in magnetic materials (Kiran Sethu)*. PhD thesis, 06 2017. URL: https://www.researchgate.net/publication/323016714_Spin_orbit_torques_in_magnetic_materials_Kiran_Sethu.

[30] Christoforos Moutafis, Stavros Komineas, and J A. C. Bland. Dynamics and switching processes for magnetic bubbles in nanoelements. *Phys. Rev. B*, 79, 06 2009. doi:10.1103/PhysRevB.79.224429.

[31] Naoto Nagaosa and Yoshinori Tokura. Topological properties and dynamics of magnetic skyrmions. *Nature Nanotechnology*, 8:899 EP –, Dec 2013. Review Article. URL: https://doi.org/10.1038/nnano.2013.243.

[32] Christian Pfleiderer and Achim Rosch. Single skyrmions spotted. *Nature*, 465:880 EP –, Jun 2010. URL: https://doi.org/10.1038/465880a.

[33] J. Sampaio, V. Cros, S. Rohart, A. Thiaville, and A. Fert. Nucleation, stability and current-induced motion of isolated magnetic skyrmions in nanostructures. *Nature Nanotechnology*, 8:839 EP –, Oct 2013. Article. URL: https://doi.org/10.1038/nnano.2013.210.

[34] Giulia Santoro. Exploring new computing paradigms for data-intensive applications. 2019. URL: https://www.semanticscholar.org/paper/Exploring-New-Computing-Paradigms-for-Applications-Santoro/b24f5507e2d65c6cccfacc30fa9810293d64b83e.

[35] Shinichiro Seki and Masahito Mochizuki. *Skyrmions in Magnetic Materials*, chapter 1, page 2. Springer International Publishing, 2016.

[36] Riccardo Tomasello, Marco Ricci, Pietro Burrascano, Vito Puliafito, Mario Carpentieri, and Giovanni Finocchio. Electrical detection of single magnetic skyrmion at room temperature. *AIP Advances*, 7(5):056022, 2017. doi:10.1063/1.4975998.

[37] M. Vacca, Y. Tavva, A. Chattopadhyay, and A. Calimera. Logic-in-memory architecture for min/max search. In *2018 25th IEEE International Conference*

on Electronics, Circuits and Systems (ICECS), pages 853–856, Dec 2018. doi:10.1109/ICECS.2018.8617879.

[38] X. S. Wang, H. Y. Yuan, and X. R. Wang. A theory on skyrmion size. *Communications Physics*, 1(1):31, 2018. doi:10.1038/s42005-018-0029-0.

[39] Yi Wang, Praveen Deorani, Xuepeng Qiu, Jae Hyun Kwon, and Hyunsoo Yang. Determination of intrinsic spin hall angle in pt. *Applied Physics Letters*, 105(15):152412, 2014. doi:10.1063/1.4898593.

[40] Xichao Zhang, Motohiko Ezawa, and Yan Zhou. Magnetic skyrmion logic gates: conversion, duplication and merging of skyrmions. *Scientific Reports*, 5:9400 EP –, Mar 2015. Article. URL: https://doi.org/10.1038/srep09400.

[41] Xichao Zhang, Yan Zhou, and Motohiko Ezawa. Magnetic bilayer-skyrmions without skyrmion hall effect. *Nature Communications*, 7:10293 EP –, Jan 2016. Article. URL: https://doi.org/10.1038/ncomms10293.

[42] Xichao Zhang, Yan Zhou, Motohiko Ezawa, G. P. Zhao, and Weisheng Zhao. Magnetic skyrmion transistor: skyrmion motion in a voltage-gated nanotrack. *Scientific Reports*, 5:11369 EP –, Jun 2015. Article. URL: https://doi.org/10.1038/srep11369.

[43] Yan Zhou and Motohiko Ezawa. A reversible conversion between a skyrmion and a domain-wall pair in a junction geometry. *Nature Communications*, 5:4652 EP –, Aug 2014. Article. URL: https://doi.org/10.1038/ncomms5652.

[44] D. Zhu, W. Kang, S. Li, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao. Skyrmion racetrack memory with random information update/deletion/insertion. *IEEE Transactions on Electron Devices*, 65(1):87–95, Jan 2018. doi:10.1109/TED.2017.2769672.

Appendices

A. Micromagnetic simulations code

A.1. *Not/Copy* gate

A.1.1. Matlab code

```
clc
clear all
close all

%%
fp_parameters=fopen('PARAMETERS.txt','r');
parameters=fscanf(fp_parameters,'%*s\t%lf\n',[1,20]);
fclose(fp_parameters);

%%
nCell_X_bmtracks = parameters(14);
nCell_Y_bmtracks = parameters(15);
nCell_X_ttrack = parameters(16);
nCell_Y_ttrack = parameters(17);
nCell_X_holes = parameters(18);
nCell_Y_holes = parameters(19);

nCell_bottomtr = nCell_X_bmtracks*nCell_Y_bmtracks;
nCell_middletr = nCell_X_bmtracks*nCell_Y_bmtracks;
nCell_toptr = nCell_X_ttrack*nCell_Y_ttrack;
nCell_hole1 = nCell_X_holes*nCell_Y_holes;
nCell_hole2 = nCell_X_holes*nCell_Y_holes;

%%
tab_data_bottomtr =
↪  readtable('currdensnorm_bottomtr.txt','Format','%f%f%f%f\n');
tab_data_hole1 = readtable('currdensnorm_hole1.txt','Format','%f%f%f%f\n');
```

```
tab_data_middletr =
↪  readtable('currdensnorm_middletr.txt','Format','%f%f%f%f\n');
tab_data_hole2 = readtable('currdensnorm_hole2.txt','Format','%f%f%f%f\n');
tab_data_toptr = readtable('currdensnorm_toptr.txt','Format','%f%f%f%f\n');

%%
matrix_data_bottomtr = table2array(tab_data_bottomtr);
matrix_data_hole1 = table2array(tab_data_hole1);
matrix_data_middletr= table2array(tab_data_middletr);
matrix_data_hole2 = table2array(tab_data_hole2);
matrix_data_toptr = table2array(tab_data_toptr);

%%
Xvec_bottomtr = matrix_data_bottomtr(:,1);
Yvec_bottomtr = matrix_data_bottomtr(:,2);
Jvec_bottomtr = matrix_data_bottomtr(:,4);

Xvec_hole1 = matrix_data_hole1(:,1);
Yvec_hole1 = matrix_data_hole1(:,2);
Jvec_hole1 = matrix_data_hole1(:,4);

Xvec_middletr = matrix_data_middletr(:,1);
Yvec_middletr= matrix_data_middletr(:,2);
Jvec_middletr = matrix_data_middletr(:,4);

Xvec_hole2 = matrix_data_hole2(:,1);
Yvec_hole2 = matrix_data_hole2(:,2);
Jvec_hole2 = matrix_data_hole2(:,4);

Xvec_toptr = matrix_data_toptr(:,1);
Yvec_toptr = matrix_data_toptr(:,2);
Jvec_toptr = matrix_data_toptr(:,4);

%%
plot3(Xvec_bottomtr,Yvec_bottomtr,Jvec_bottomtr,'b*',Xvec_middletr,
↪  Yvec_middletr,Jvec_middletr,'c*',Xvec_toptr,Yvec_toptr,Jvec_toptr,'g*')
hold on
plot3(Xvec_hole1,Yvec_hole1,Jvec_hole1,'r*',Xvec_hole2,Yvec_hole2,Jvec_hole2,
↪  'r*')
grid on
xlabel('x - track length')
ylabel('y - track width')
zlabel('current density norm.')

%%
j=1;
jdensity=fopen('jdensity.txt','w');

%bottom track
```

```
for i=1:1:nCell_bottomtr
    fprintf(jdensity,'%d \n', Jvec_bottomtr(i));
    Jvec_whole(j) = Jvec_bottomtr(i);
    j=j+1;
end

%hole1
for i=(nCell_bottomtr+1):1:(nCell_bottomtr+nCell_hole1)
    fprintf(jdensity,'%d \n', Jvec_hole1(i-nCell_bottomtr));
    Jvec_whole(j) = Jvec_hole1(i-nCell_bottomtr);
    j=j+1;
end

%middle track
for i=(nCell_bottomtr+nCell_hole1+1):1:(nCell_bottomtr+nCell_hole1+nCell_middletr)
    fprintf(jdensity,'%d \n', Jvec_middletr(i-nCell_bottomtr-nCell_hole1));
    Jvec_whole(j) = Jvec_middletr(i-nCell_bottomtr-nCell_hole1);
    j=j+1;
end

%hole2
for i=(nCell_bottomtr+nCell_hole1+nCell_middletr+1):1:(nCell_bottomtr+nCell_hole1+nCell_middletr+nCell_hole2)
    fprintf(jdensity,'%d \n', Jvec_hole2(i-nCell_bottomtr-nCell_hole1-nCell_middletr));
    Jvec_whole(j) = Jvec_hole2(i-nCell_bottomtr-nCell_hole1-nCell_middletr);
    j=j+1;
end

%top track
for i=(nCell_bottomtr+nCell_hole1+nCell_middletr+nCell_hole2+1):1:(nCell_bottomtr+nCell_hole1+nCell_middletr+nCell_hole2+nCell_toptr)
    fprintf(jdensity,'%d \n', Jvec_toptr(i-nCell_bottomtr-nCell_hole1-nCell_middletr-nCell_hole2));
    Jvec_whole(j) = Jvec_toptr(i-nCell_bottomtr-nCell_hole1-nCell_middletr-nCell_hole2);
    j=j+1;
end

maxj=max(Jvec_whole)
minj=min(Jvec_whole)
```

A.1.2. C code

```
#include<stdio.h>
#include<stdlib.h>

int main ()
{
    FILE *fp_in, *fp_param, *fp_mx3;
    fp_in = fopen("..\\jdensity.txt", "r");
    fp_param = fopen("..\\PARAMETERS.txt", "r");
    fp_mx3 = fopen("..\\prova.mx3", "w");

    double jdensity;
    int i, j;
    int nReg=1;
    double coord_x;
    double coord_y;
    double parameters[20]={0};

    double sim_time=1e-9;
    int j_uniform=1;

    printf("Simulation time:\n");
    scanf("%lf",&sim_time);
    printf("Uniform current density?\n");
    scanf("%d",&j_uniform);


    for(i=1; i<21; i++) {
        fscanf(fp_param,"%*s\t%lf\n",&parameters[i]);
        //printf("%e\n",parameters[i]);
    }

    double size_contact         = parameters[1];
    double track_length     = parameters[2];
    double track_width      = parameters[3];
    double out_bound_width      = parameters[4];
    double thickness_layer    = parameters[5];
    double hole1_width        = parameters[6];
    double hole2_width        = parameters[7];
    double hole1_height        = parameters[8];
    double hole2_height        = parameters[9];
    double xcoord_holestart     = parameters[10];
    double xcoord_track3start  = parameters[11];
    double offset_hole2          = parameters[12];

    int nCell_X_bmtracks = parameters[14];
    int nCell_Y_bmtracks = parameters[15];
```

```
    int nCell_X_ttrack = parameters[16];
    int nCell_Y_ttrack = parameters[17];
    int nCell_X_holes = parameters[18];
    int nCell_Y_holes = parameters[19];

    double SHangle = parameters[20];

    double X_grid_spacing_bmtracks = track_length/nCell_X_bmtracks;
    double Y_grid_spacing_bmtracks = track_width/nCell_Y_bmtracks;
    double X_grid_spacing_ttrack = (track_length-xcoord_track3start)/nCell_X_ttrack;
    double Y_grid_spacing_ttrack = track_width/nCell_Y_ttrack;
    double X_grid_spacing_hole1 = hole1_width/nCell_X_holes;
    double Y_grid_spacing_hole1 = hole1_height/nCell_Y_holes;
    double X_grid_spacing_hole2 = hole2_width/nCell_X_holes;
    double Y_grid_spacing_hole2 = hole2_height/nCell_Y_holes;

    int nCell_bmtracks = nCell_X_bmtracks*nCell_Y_bmtracks;
    int nCell_ttrack = nCell_X_ttrack*nCell_Y_ttrack;
    int nCell_hole1 = nCell_X_holes*nCell_Y_holes;
    int nCell_hole2 = nCell_X_holes*nCell_Y_holes;


    fprintf(fp_mx3,"//MATERIAL PARAMETERS\n");
    fprintf(fp_mx3,"Temp = 0\n");
    fprintf(fp_mx3,"Msat = 5.8e5\n");
    fprintf(fp_mx3,"Aex = 1.5e-11\n");
    fprintf(fp_mx3,"alpha = 0.1\n");
    fprintf(fp_mx3,"Dind = 3.0e-3\n");
    fprintf(fp_mx3,"Ku1 = 6e5\n");
    fprintf(fp_mx3,"Ku2 = 1.5e5\n");
    fprintf(fp_mx3,"Xi = 0.35\n");
    fprintf(fp_mx3,"Pol = 1\n");
    fprintf(fp_mx3,"Lambda = 1\n");
    fprintf(fp_mx3,"AnisU = vector(0,0,1)\n");
    fprintf(fp_mx3,"EpsilonPrime = 0\n");
    fprintf(fp_mx3,"fixedlayer = vector(0,-1,0)\n");
    fprintf(fp_mx3,"B_ext = vector(0,0,0)\n\n");

    fprintf(fp_mx3,"//GEOMETRY PARAMETERS\n");
    fprintf(fp_mx3,"size_contact := %e\n",size_contact);
    fprintf(fp_mx3,"track_length := %e\n",track_length);
    fprintf(fp_mx3,"track_width := %e\n",track_width);
    fprintf(fp_mx3,"out_bound_width := %e\n",out_bound_width);
    fprintf(fp_mx3,"thickness_layer := %e\n",thickness_layer);
    fprintf(fp_mx3,"hole1_width := %e\n",hole1_width);
    fprintf(fp_mx3,"hole2_width := %e\n",hole2_width);
    fprintf(fp_mx3,"hole1_height := %e\n",hole1_height);
    fprintf(fp_mx3,"hole2_height := %e\n",hole2_height);
```

```
    fprintf(fp_mx3,"xcoord_holestart := %e\n",xcoord_holestart);
    fprintf(fp_mx3,"xcoord_track3start := %e\n",xcoord_track3start);
    fprintf(fp_mx3,"offset_hole2 := %e\n\n",offset_hole2);

    fprintf(fp_mx3,"//GRID SETTING\n");
    fprintf(fp_mx3,"grid_x := 256\n");
    fprintf(fp_mx3,"grid_y := 128\n");
    fprintf(fp_mx3,"grid_z := 4\n");
    fprintf(fp_mx3,"SetGridSize(256, 128, 4)\n\n");

    fprintf(fp_mx3,"////////////////////////////////////////////////////////////////\n\n");

    fprintf(fp_mx3,"cell_x := track_length/grid_x\n");
    fprintf(fp_mx3,"cell_y := (2*out_bound_width+3*track_width+hole1_height+hole2_height)/grid_y\n");
    fprintf(fp_mx3,"cell_z := 2*thickness_layer/grid_z\n");
    fprintf(fp_mx3,"SetCellSize(cell_x, cell_y, cell_z)\n");
    fprintf(fp_mx3,"SetPBC(0, 0, 0)\n\n");

    fprintf(fp_mx3,"/////////////////////////////////////////////////////////////\n\n\n");

    if(j_uniform==1) {
        fprintf(fp_mx3,"//STRUCTURE\n");
        fprintf(fp_mx3,"bottomtr := Rect(track_length,track_width).transl(0,-track_width/2-hole1_height-track_width/2,0)\n");
        fprintf(fp_mx3,"middletr := Rect(track_length,track_width).transl(0,0,0)\n");
        fprintf(fp_mx3,"toptr := Rect(track_length-xcoord_track3start,track_width).transl(track_length/2-(track_length-xcoord_track3start)/2,track_width/2+hole2_height+track_width/2,0)\n");
        fprintf(fp_mx3,"hole1 := Rect(hole1_width,hole1_height).transl(-track_length/2+xcoord_holestart+hole1_width/2,-track_width/2-hole1_height/2,0)\n");
        fprintf(fp_mx3,"hole2 := Rect(hole2_width,hole2_height).transl(-track_length/2+xcoord_holestart+hole2_width/2+offset_hole2,track_width/2+hole2_height/2,0)\n");
        fprintf(fp_mx3,"track_full := bottomtr.add(middletr).add(toptr).add(hole1).add(hole2)\n");
        fprintf(fp_mx3,"track := track_full.intersect(ZRange(-inf, 0))\n");
        fprintf(fp_mx3,"SetGeom(Universe().Sub(track))\n\n");

        fprintf(fp_mx3,"//REGIONS\n");
        fprintf(fp_mx3,"DefRegion(1, bottomtr)\n");
        fprintf(fp_mx3,"DefRegion(2, middletr)\n\n");

        fprintf(fp_mx3,"//INITIAL MAGNETIZATION\n");
```

```
        fprintf(fp_mx3,"m = uniform(0, 0, 1)\n");
        fprintf(fp_mx3,"m.setregion(1, NeelSkyrmion(1, -1).transl(-track_length/2+20e-9, -track_width-hole1_height, 0))\n");
        fprintf(fp_mx3,"m.setregion(2, NeelSkyrmion(1, -1).transl(-track_length/2+20e-9, 0, 0))\n\n");
    }
    else {
        fprintf(fp_mx3,"//STRUCTURE\n");
        fprintf(fp_mx3,"elem_cell_bmtracks := Rect(%e, %e)\n",X_grid_spacing_bmtracks,Y_grid_spacing_bmtracks);
        fprintf(fp_mx3,"elem_cell_ttrack := Rect(%e, %e)\n",X_grid_spacing_ttrack,Y_grid_spacing_ttrack);
        fprintf(fp_mx3,"elem_cell_hole1  := Rect(%e, %e)\n",X_grid_spacing_hole1,Y_grid_spacing_hole1);
        fprintf(fp_mx3,"elem_cell_hole2  := Rect(%e, %e)\n",X_grid_spacing_hole2,Y_grid_spacing_hole2);
        fprintf(fp_mx3,"\n\n");

        //bottom track
        for(j=0; j<nCell_Y_bmtracks ; j++) {
            coord_y = -track_width/2-hole1_height-track_width+Y_grid_spacing_bmtracks/2+j*Y_grid_spacing_bmtracks;
            for(i=0; i<nCell_X_bmtracks ; i++){
                coord_x = -track_length/2+X_grid_spacing_bmtracks/2+i*X_grid_spacing_bmtracks;
                fprintf(fp_mx3,"reg%d := elem_cell_bmtracks.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
                fprintf(fp_mx3,"defRegion(%d,reg%d)\n", nReg, nReg);
                nReg++;
            }
        }

        //hole1
        for(j=0; j<nCell_Y_holes; j++) {
            coord_y = -track_width/2-hole1_height+Y_grid_spacing_hole1/2+j*Y_grid_spacing_hole1;
            for(i=0; i<nCell_X_holes; i++){
                coord_x = -track_length/2+xcoord_holestart+X_grid_spacing_hole1/2+i*X_grid_spacing_hole1;
                fprintf(fp_mx3,"reg%d := elem_cell_hole1.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
                fprintf(fp_mx3,"defRegion(%d,reg%d)\n", nReg, nReg);
                nReg++;
            }
        }

        //middle track
        for(j=0; j<nCell_Y_bmtracks; j++) {
```

```
            coord_y = -track_width/2+Y_grid_spacing_bmtracks/2+j*Y_grid_spacing_bmtracks;
            for(i=0; i<nCell_X_bmtracks; i++){
                coord_x = -track_length/2+X_grid_spacing_bmtracks/2+i*X_grid_spacing_bmtracks;
                fprintf(fp_mx3,"reg%d := elem_cell_bmtracks.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
                fprintf(fp_mx3,"defRegion(%d,reg%d)\n", nReg, nReg);
                nReg++;
            }
        }

        //hole2
        for(j=0; j<nCell_Y_holes; j++) {
            coord_y = track_width/2+Y_grid_spacing_hole2/2+j*Y_grid_spacing_hole2;
            for(i=0; i<nCell_X_holes; i++){
                coord_x = -track_length/2+xcoord_holestart+offset_hole2+X_grid_spacing_hole2/2+i*X_grid_spacing_hole2;
                fprintf(fp_mx3,"reg%d := elem_cell_hole2.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
                fprintf(fp_mx3,"defRegion(%d,reg%d)\n", nReg, nReg);
                nReg++;
            }
        }

        //top track
        for(j=0; j<nCell_Y_ttrack; j++) {
            coord_y = track_width/2+hole2_height+Y_grid_spacing_ttrack/2+j*Y_grid_spacing_ttrack;
            for(i=0; i<nCell_X_ttrack; i++){
                coord_x = -track_length/2+xcoord_track3start+X_grid_spacing_ttrack/2+i*X_grid_spacing_ttrack;
                fprintf(fp_mx3,"reg%d := elem_cell_ttrack.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
                fprintf(fp_mx3,"defRegion(%d,reg%d)\n", nReg, nReg);
                nReg++;
            }
        }

        nReg--;
        fprintf(fp_mx3,"\n");

        fprintf(fp_mx3,"track_full := reg1");
        for (i=2; i<nReg+1; i++) {
            fprintf(fp_mx3,".add(reg%d)",i);
        }

        fprintf(fp_mx3,"\n");
```

```
        fprintf(fp_mx3,"track := track_full.intersect(ZRange(-inf, 0))\n");
        fprintf(fp_mx3,"SetGeom(Universe().Sub(track))\n\n");
        fprintf(fp_mx3,"\n\n");

        fprintf(fp_mx3,"//LOCAL CURRENT EXCITATION\n");
        for(i=1; i<nReg+1; i++) {
            fscanf(fp_in,"%lf \n", &jdensity);
            fprintf(fp_mx3,"j.setRegion(%d,vector(0, 0, %e))\n", i, jdensity*SHangle);
        }

        //bottom track
        fprintf(fp_mx3,"\n\n");

        fprintf(fp_mx3,"//INITIAL MAGNETIZATION\n");
        fprintf(fp_mx3,"m = uniform(0, 0, 1)\n");
        fprintf(fp_mx3,"m.setRegion(1,NeelSkyrmion(1,-1).transl(-track_length/2+20e-9,-hole1_height-track_width,0))\n");
        fprintf(fp_mx3,"m.setRegion(%d,NeelSkyrmion(1,-1).transl(-track_length/2+20e-9,0,0))\n\n",nCell_bmtracks+nCell_hole1+1);
    }

    fprintf(fp_mx3, "////////////////////////////////////////////////////////////\n\n\n");

    fprintf(fp_mx3,"//OUTPUT SAVE\n");
    fprintf(fp_mx3,"OutputFormat = OVF1_TEXT\n");
    fprintf(fp_mx3,"tableAdd(ext_topologicalcharge)\n");
    fprintf(fp_mx3,"tableautosave(1e-12)\n");
    fprintf(fp_mx3,"AutoSnapShot(m_full, 5e-11)\n");
    fprintf(fp_mx3,"AutoSave(m, 2e-11)\n\n");

    fprintf(fp_mx3,"//SIMULATION\n");
    if(j_uniform==1) {
        fprintf(fp_mx3,"J = vector(0, 0, 5e10)\n");
    }
    fprintf(fp_mx3,"Run(%e)\n",sim_time);

  fclose(fp_in);
  fclose(fp_param);
  fclose(fp_mx3);

  return 0;
}
```

A.1.3. Parameters file

A.1.3.1. Not_Structure 1

```
size_contact  0
track_length  256e-9
track_width  20e-9
out_bound_width  34e-9
thickness_layer  0.4e-9
hole1_width  30e-9
hole2_width  25e-9
hole1_height  20e-9
hole2_height  20e-9
xcoord_holestart  113e-9
xcoord_track3start  100e-9
offset_hole2  0
Vappl  100e-5
nCell_X_bmtracks  7
nCell_Y_bmtracks  1
nCell_X_ttrack  5
nCell_Y_ttrack  1
nCell_X_holes  1
nCell_Y_holes  2
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
```

A.1.3.2. Not_Structure 2

```
size_contact  130e-9
track_length  256e-9
track_width  20e-9
out_bound_width  34e-9
thickness_layer  0.4e-9
hole1_width  30e-9
hole2_width  25e-9
hole1_height  20e-9
```

```
hole2_height  20e-9
xcoord_holestart  113e-9
xcoord_track3start  100e-9
offset_hole2  0
Vappl  280e-5
nCell_X_bmtracks  7
nCell_Y_bmtracks  1
nCell_X_ttrack  5
nCell_Y_ttrack  1
nCell_X_holes  1
nCell_Y_holes  2
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
```

A.1.3.3. Not_Structure 3

```
size_contact  130e-9
track_length  256e-9
track_width  20e-9
out_bound_width  34e-9
thickness_layer  0.8e-9
hole1_width  27e-9
hole2_width  25e-9
hole1_height  20e-9
hole2_height  20e-9
xcoord_holestart  113e-9
xcoord_track3start  100e-9
offset_hole2  0
Vappl  280e-5
nCell_X_bmtracks  7
nCell_Y_bmtracks  1
nCell_X_ttrack  5
nCell_Y_ttrack  1
nCell_X_holes  1
nCell_Y_holes  2
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
```

```
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
```

A.1.3.4. Not_Structure 4

```
size_contact  130e-9
track_length  256e-9
track_width  20e-9
out_bound_width  34e-9
thickness_layer  0.8e-9
hole1_width  26e-9
hole2_width  26e-9
hole1_height  20e-9
hole2_height  20e-9
xcoord_holestart  113e-9
xcoord_track3start  100e-9
offset_hole2  -4e-9
Vappl  280e-5
nCell_X_bmtracks  7
nCell_Y_bmtracks  1
nCell_X_ttrack  5
nCell_Y_ttrack  1
nCell_X_holes  1
nCell_Y_holes  2
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
```

A.1.3.5. Not_Structure 5

```
size_contact  130e-9
track_length  256e-9
track_width  20e-9
out_bound_width  34e-9
thickness_layer  0.4e-9
hole1_width  27e-9
hole2_width  25e-9
hole1_height  20e-9
hole2_height  20e-9
xcoord_holestart  113e-9
xcoord_track3start  100e-9
offset_hole2  0
Vappl  280e-5
nCell_X_bmtracks  7
nCell_Y_bmtracks  1
nCell_X_ttrack  5
nCell_Y_ttrack  1
nCell_X_holes  1
nCell_Y_holes  2
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
```

A.1.3.6. Not_Structure 6

```
size_contact  130e-9
track_length  256e-9
track_width  20e-9
out_bound_width  34e-9
thickness_layer  0.8e-9
hole1_width  30e-9
hole2_width  25e-9
hole1_height  20e-9
hole2_height  20e-9
xcoord_holestart  113e-9
xcoord_track3start  100e-9
```

```
offset_hole2  0
Vappl  280e-5
nCell_X_bmtracks  7
nCell_Y_bmtracks  1
nCell_X_ttrack  5
nCell_Y_ttrack  1
nCell_X_holes  1
nCell_Y_holes  2
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
```

A.2. *And/Or* gate

A.2.1. Matlab code

```
clc
clear all
close all

%%
fp_parameters=fopen('PARAMETERS.txt','r');
parameters=fscanf(fp_parameters,'%*s\t%lf\n',[1,13]);
fclose(fp_parameters);

%%
nCell_X_topbottom = parameters(9);
nCell_Y_topbottom = parameters(10);
nCell_X_junc = parameters(11);
nCell_Y_junc = parameters(12);

nCell_bottomtr = nCell_X_topbottom*nCell_Y_topbottom;
nCell_toptr = nCell_X_topbottom*nCell_Y_topbottom;
nCell_junct = nCell_X_junc*nCell_Y_junc;

%%
```

```
tab_data_bottomtr =
↪  readtable('currdensnorm_bottomtr.txt','Format','%f%f%f%f\n');
tab_data_middlejun =
↪  readtable('currdensnorm_middlejun.txt','Format','%f%f%f%f\n');
tab_data_toptr = readtable('currdensnorm_toptr.txt','Format','%f%f%f%f\n');

%%
matrix_data_bottomtr = table2array(tab_data_bottomtr);
matrix_data_middlejun = table2array(tab_data_middlejun);
matrix_data_toptr = table2array(tab_data_toptr);

%%
Xvec_bottomtr = matrix_data_bottomtr(:,1);
Yvec_bottomtr = matrix_data_bottomtr(:,2);
Jvec_bottomtr = matrix_data_bottomtr(:,4);

Xvec_middlejun = matrix_data_middlejun(:,1);
Yvec_middlejun = matrix_data_middlejun(:,2);
Jvec_middlejun = matrix_data_middlejun(:,4);

Xvec_toptr = matrix_data_toptr(:,1);
Yvec_toptr = matrix_data_toptr(:,2);
Jvec_toptr = matrix_data_toptr(:,4);

%%
figure(1)
plot3(Xvec_bottomtr,Yvec_bottomtr,Jvec_bottomtr,'b*',Xvec_middlejun,⌋
↪  Yvec_middlejun,Jvec_middlejun,'r*',Xvec_toptr,Yvec_toptr,Jvec_toptr,'g*')
grid on
xlabel('x - track length')
ylabel('y - track width')
zlabel('current density norm.')

%%
j=1;
jdensity=fopen('jdensity.txt','w');
for i=1:1:nCell_bottomtr
    fprintf(jdensity,'%d \n', Jvec_bottomtr(i));
    Jvec_whole(j) = Jvec_bottomtr(i);
    j=j+1;
end

for i=(nCell_bottomtr+1):1:(nCell_bottomtr+nCell_junct)
    fprintf(jdensity,'%d \n', Jvec_middlejun(i-nCell_bottomtr));
    Jvec_whole(j) = Jvec_middlejun(i-nCell_bottomtr);
    j=j+1;
end

for i=(nCell_bottomtr+nCell_junct+1):1:(nCell_bottomtr+nCell_junct+nCell_toptr)
```

```
    fprintf(jdensity,'%d \n', Jvec_toptr(i-nCell_bottomtr-nCell_junct));
    Jvec_whole(j) = Jvec_toptr(i-nCell_bottomtr-nCell_junct);
    j=j+1;
end

maxj=max(Jvec_whole)
minj=min(Jvec_whole)
```

A.2.2. C code

```
#include<stdio.h>
#include<stdlib.h>

int main ()
{
    FILE *fp_in, *fp_param, *fp_mx3;
    fp_in = fopen("..\\jdensity.txt", "r");
    fp_param = fopen("..\\PARAMETERS.txt", "r");
    fp_mx3 = fopen("..\\prova.mx3", "w");

    double jdensity;
    int i, j;
    int nReg=1;
    double coord_x;
    double coord_y;
    double parameters[13]={0};

    double sim_time=1e-9;
    int j_uniform=1;

    printf("Simulation time:\n");
    scanf("%lf",&sim_time);
    printf("Uniform current density?\n");
    scanf("%d",&j_uniform);


    for(i=1; i<14; i++) {
        fscanf(fp_param,"%*s\t%lf\n",&parameters[i]);
        //printf("%e\n",parameters[i]);
    }

    double size_contact         = parameters[1];
    double track_length     = parameters[2];
    double out_bound_width    = parameters[3];
```

```
    double track_width      = parameters[4];
    double thickness_layer    = parameters[5];
    double hole_height      = parameters[6];
    double hole_width      = parameters[7];

    int nCell_X_topbottom = parameters[9];
    int nCell_Y_topbottom = parameters[10];
    int nCell_X_junc = parameters[11];
    int nCell_Y_junc = parameters[12];

    double SHangle = parameters[13];


    double X_grid_spacing_topbottom = track_length/nCell_X_topbottom;
    double Y_grid_spacing_topbottom = track_width/nCell_Y_topbottom;
    double X_grid_spacing_junc = hole_width/nCell_X_junc;
    double Y_grid_spacing_junc = hole_height/nCell_Y_junc;

    int nCell_topbottom = nCell_X_topbottom*nCell_Y_topbottom;
    int nCell_junc = nCell_X_junc*nCell_Y_junc;

    fprintf(fp_mx3,"//MATERIAL PARAMETERS\n");
    fprintf(fp_mx3,"Temp = 0\n");
    fprintf(fp_mx3,"Msat = 5.8e5\n");
    fprintf(fp_mx3,"Aex = 1.5e-11\n");
    fprintf(fp_mx3,"alpha = 0.1\n");
    fprintf(fp_mx3,"Dind = 3.0e-3\n");
    fprintf(fp_mx3,"Ku1 = 6e5\n");
    fprintf(fp_mx3,"Ku2 = 1.5e5\n");
    fprintf(fp_mx3,"Xi = 0.35\n");
    fprintf(fp_mx3,"Pol = 1\n");
    fprintf(fp_mx3,"Lambda = 1\n");
    fprintf(fp_mx3,"AnisU = vector(0,0,1)\n");
    fprintf(fp_mx3,"EpsilonPrime = 0\n");
    fprintf(fp_mx3,"fixedlayer = vector(0,-1,0)\n");
    fprintf(fp_mx3,"B_ext = vector(0,0,0)\n\n");

    fprintf(fp_mx3,"//GEOMETRY PARAMETERS\n");
    fprintf(fp_mx3,"size_contact := %e\n",size_contact);
    fprintf(fp_mx3,"track_length := %e\n",track_length);
    fprintf(fp_mx3,"out_bound_width := %e\n",out_bound_width);
    fprintf(fp_mx3,"track_width := %e\n",track_width);
    fprintf(fp_mx3,"thickness_layer := %e\n",thickness_layer);
    fprintf(fp_mx3,"hole_height := %e\n",hole_height);
    fprintf(fp_mx3,"hole_width := %e\n\n",hole_width);

    fprintf(fp_mx3,"//GRID SETTING\n");
    fprintf(fp_mx3,"grid_x := 256\n");
    fprintf(fp_mx3,"grid_y := 128\n");
```

```
    fprintf(fp_mx3,"grid_z := 4\n");
    fprintf(fp_mx3,"SetGridSize(256, 128, 4)\n\n");


    fprintf(fp_mx3,"////////////////////////////////////////////////////////////////\n\n");

    fprintf(fp_mx3,"cell_x := track_length/grid_x\n");
    fprintf(fp_mx3,"cell_y := (2*out_bound_width+2*track_width+hole_height)/grid_y\n");
    fprintf(fp_mx3,"cell_z := 2*thickness_layer/grid_z\n");
    fprintf(fp_mx3,"SetCellSize(cell_x, cell_y, cell_z)\n");
    fprintf(fp_mx3,"SetPBC(0, 0, 0)\n\n");

    fprintf(fp_mx3,"////////////////////////////////////////////////////////////\n\n\n");

    if(j_uniform==1) {
        fprintf(fp_mx3,"//STRUCTURE\n");
        fprintf(fp_mx3,"trackb := Rect(track_length, track_width).transl(0, -hole_height/2-track_width/2, 0)\n");
        fprintf(fp_mx3,"trackt := Rect(track_length, track_width).transl(0, hole_height/2+track_width/2, 0)\n");
        fprintf(fp_mx3,"jun  := Rect(hole_width, hole_height)\n");
        fprintf(fp_mx3,"track_full := trackb.add(trackt).add(jun)\n");
        fprintf(fp_mx3,"track := track_full.intersect(ZRange(-inf, 0))\n");
        fprintf(fp_mx3,"SetGeom(Universe().Sub(track))\n\n");

        fprintf(fp_mx3,"//REGIONS\n");
        fprintf(fp_mx3,"DefRegion(1, trackb)\n");
        fprintf(fp_mx3,"DefRegion(2, trackt)\n\n");

        fprintf(fp_mx3,"//INITIAL MAGNETIZATION\n");
        fprintf(fp_mx3,"m = uniform(0, 0, 1)\n");
        fprintf(fp_mx3,"m.setregion(1, NeelSkyrmion(1, -1).transl(-track_length/2+20e-9, -track_width, 0))\n");
        fprintf(fp_mx3,"m.setregion(2, NeelSkyrmion(1, -1).transl(-track_length/2+20e-9, track_width, 0))\n\n");
    }
    else {
        fprintf(fp_mx3,"//STRUCTURE\n");
        fprintf(fp_mx3,"elem_cell_track := Rect(%e, %e)\n",X_grid_spacing_topbottom,Y_grid_spacing_topbottom);
        fprintf(fp_mx3,"elem_cell_junc  := Rect(%e, %e)\n",X_grid_spacing_junc,Y_grid_spacing_junc);
        fprintf(fp_mx3,"\n\n");

        //bottom track
        for(j=0; j<nCell_Y_topbottom; j++) {
```

```
        coord_y = -hole_height/2-track_width+Y_grid_spacing_topbottom/2+j*Y_grid_spacing_topbottom;
        for(i=0; i<nCell_X_topbottom; i++){
            coord_x = -track_length/2+X_grid_spacing_topbottom/2+i*X_grid_spacing_topbottom;
            fprintf(fp_mx3,"reg%d := elem_cell_track.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
            nReg++;
        }
    }

    //middle junction
    for(j=0; j<nCell_Y_junc; j++) {
        coord_y = -hole_height/2+Y_grid_spacing_junc/2+j*Y_grid_spacing_junc;
        for(i=0; i<nCell_X_junc; i++){
            coord_x = -hole_width/2+X_grid_spacing_junc/2+i*X_grid_spacing_junc;
            fprintf(fp_mx3,"reg%d := elem_cell_junc.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
            nReg++;
        }
    }

    //top track
    for(j=0; j<nCell_Y_topbottom; j++) {
        coord_y = hole_height/2+Y_grid_spacing_topbottom/2+j*Y_grid_spacing_topbottom;
        for(i=0; i<nCell_X_topbottom; i++){
            coord_x = -track_length/2+X_grid_spacing_topbottom/2+i*X_grid_spacing_topbottom;
            fprintf(fp_mx3,"reg%d := elem_cell_track.transl(%e,%e,0)\n", nReg, coord_x, coord_y);
            nReg++;
        }
    }

    nReg--;
    fprintf(fp_mx3,"\n");

    fprintf(fp_mx3,"track_full := reg1");
    for (i=2; i<nReg+1; i++) {
        fprintf(fp_mx3,".add(reg%d)",i);
    }

    fprintf(fp_mx3,"\n");
    fprintf(fp_mx3,"track := track_full.intersect(ZRange(-inf, 0))\n");
    fprintf(fp_mx3,"SetGeom(Universe().Sub(track))\n\n");
    fprintf(fp_mx3,"\n");
```

```
        fprintf(fp_mx3,"//REGIONS\n");
        for(i=1;i<nReg+1;i++) {
            fprintf(fp_mx3,"defRegion(%d,reg%d)\n", i, i);
        }
        fprintf(fp_mx3,"\n\n\n");

        fprintf(fp_mx3,"//LOCAL CURRENT EXCITATION\n");
        for(i=1; i<nReg+1; i++) {
            fscanf(fp_in,"%lf \n", &jdensity);
            fprintf(fp_mx3,"j.setRegion(%d,vector(0, 0, %e))\n", i,
            ↪  jdensity*SHangle);
        }

        fprintf(fp_mx3,"\n\n");

        fprintf(fp_mx3,"//INITIAL MAGNETIZATION\n");
        fprintf(fp_mx3,"m = uniform(0, 0, 1)\n");
        fprintf(fp_mx3,"m.setRegion(1,NeelSkyrmion(1,-1).transl(-
        ↪  track_length/2+20e-9,-hole_height/2-track_width/2,0))\n");
        fprintf(fp_mx3,"m.setRegion(%d,NeelSkyrmion(1,-1).transl(-
        ↪  track_length/2+20e-9,hole_height/2+track_width/2,0))\n\n",
        ↪  nCell_topbottom+nCell_junc+1);
    }

    fprintf(fp_mx3,
    ↪  "///////////////////////////////////////////////////////////////\n\n");

    fprintf(fp_mx3,"//OUTPUT SAVE\n");
    fprintf(fp_mx3,"OutputFormat = OVF1_TEXT\n");
    fprintf(fp_mx3,"tableAdd(ext_topologicalcharge)\n");
    fprintf(fp_mx3,"tableautosave(1e-12)\n");
    fprintf(fp_mx3,"AutoSnapShot(m_full, 5e-11)\n");
    fprintf(fp_mx3,"AutoSave(m, 2e-11)\n\n");

    fprintf(fp_mx3,"//SIMULATION\n");
    if(j_uniform==1) {
        fprintf(fp_mx3,"J = vector(0, 0, 5e10)\n");
    }
    fprintf(fp_mx3,"Run(%e)\n",sim_time);

    fclose(fp_in);
    fclose(fp_param);
    fclose(fp_mx3);

    return 0;
}
```

A.2.3. Parameters file

A.2.3.1. H_Structure 1

```
size_contact  130e-9
track_length  256e-9
out_bound_width  34e-9
track_width  20e-9
thickness_layer  0.4e-9
hole_height  20e-9
hole_width  30e-9
Vappl  280e-5
nCell_X_topbottom  8
nCell_Y_topbottom  1
nCell_X_junc  1
nCell_Y_junc  2
SHangle  1
X_grid_spacing_topbottom  track_length/nCell_X_topbottom
Y_grid_spacing_topbottom  track_width/nCell_Y_topbottom
X_grid_spacing_junc  hole_width/nCell_X_junc
Y_grid_spacing_junc  hole_height/nCell_Y_junc
```

A.2.3.2. H_Structure 2

```
size_contact  130e-9
track_length  256e-9
out_bound_width  34e-9
track_width  20e-9
thickness_layer  0.8e-9
hole_height  20e-9
hole_width  30e-9
Vappl  280e-5
nCell_X_topbottom  8
nCell_Y_topbottom  1
nCell_X_junc  1
nCell_Y_junc  2
SHangle  1
X_grid_spacing_topbottom  track_length/nCell_X_topbottom
Y_grid_spacing_topbottom  track_width/nCell_Y_topbottom
X_grid_spacing_junc  hole_width/nCell_X_junc
```

```
Y_grid_spacing_junc  hole_height/nCell_Y_junc
```

A.2.3.3. H_Structure 3

```
size_contact  130e-9
track_length  256e-9
out_bound_width  34e-9
track_width  20e-9
thickness_layer  0.8e-9
hole_height  14e-9
hole_width  25e-9
Vappl  280e-5
nCell_X_topbottom  10
nCell_Y_topbottom  1
nCell_X_junc  1
nCell_Y_junc  2
SHangle  1
X_grid_spacing_topbottom  track_length/nCell_X_topbottom
Y_grid_spacing_topbottom  track_width/nCell_Y_topbottom
X_grid_spacing_junc  hole_width/nCell_X_junc
Y_grid_spacing_junc  hole_height/nCell_Y_junc
```

A.3. Balancing of current density

A.3.1. *Not/Copy* gate

A.3.1.1. Matlab code

```
clc
clear all
close all
%%
tab_data_bottomtr =
↪  readtable('currdensnorm_bottomtr.txt','Format','%f%f%f%f\n');
tab_data_hole1 = readtable('currdensnorm_hole1.txt','Format','%f%f%f%f\n');
tab_data_middletr =
↪  readtable('currdensnorm_middletr.txt','Format','%f%f%f%f\n');
```

```
tab_data_hole2 = readtable('currdensnorm_hole2.txt','Format','%f%f%f%f\n');
tab_data_toptr = readtable('currdensnorm_toptr.txt','Format','%f%f%f%f\n');

tab_data_bottomtr_Co =
↪  readtable('currdensnorm_bottomtr_Co.txt','Format','%f%f%f%f\n');
tab_data_hole1_Co  =
↪  readtable('currdensnorm_hole1_Co.txt','Format','%f%f%f%f\n');
tab_data_middletr_Co  =
↪  readtable('currdensnorm_middletr_Co.txt','Format','%f%f%f%f\n');
tab_data_hole2_Co  =
↪  readtable('currdensnorm_hole2_Co.txt','Format','%f%f%f%f\n');
tab_data_toptr_Co  =
↪  readtable('currdensnorm_toptr_Co.txt','Format','%f%f%f%f\n');


%%
matrix_data_bottomtr = table2array(tab_data_bottomtr);
matrix_data_hole1 = table2array(tab_data_hole1);
matrix_data_middletr= table2array(tab_data_middletr);
matrix_data_hole2 = table2array(tab_data_hole2);
matrix_data_toptr = table2array(tab_data_toptr);

matrix_data_bottomtr_Co = table2array(tab_data_bottomtr_Co);
matrix_data_hole1_Co = table2array(tab_data_hole1_Co);
matrix_data_middletr_Co= table2array(tab_data_middletr_Co);
matrix_data_hole2_Co = table2array(tab_data_hole2_Co);
matrix_data_toptr_Co = table2array(tab_data_toptr_Co);

%%
Xvec_bottomtr = matrix_data_bottomtr(:,1);
Yvec_bottomtr = matrix_data_bottomtr(:,2);
Jvec_bottomtr = matrix_data_bottomtr(:,4);

Xvec_hole1 = matrix_data_hole1(:,1);
Yvec_hole1 = matrix_data_hole1(:,2);
Jvec_hole1 = matrix_data_hole1(:,4);

Xvec_middletr = matrix_data_middletr(:,1);
Yvec_middletr= matrix_data_middletr(:,2);
Jvec_middletr = matrix_data_middletr(:,4);

Xvec_hole2 = matrix_data_hole2(:,1);
Yvec_hole2 = matrix_data_hole2(:,2);
Jvec_hole2 = matrix_data_hole2(:,4);

Xvec_toptr = matrix_data_toptr(:,1);
Yvec_toptr = matrix_data_toptr(:,2);
Jvec_toptr = matrix_data_toptr(:,4);
```

```
Xvec_bottomtr_Co = matrix_data_bottomtr_Co(:,1);
Yvec_bottomtr_Co = matrix_data_bottomtr_Co(:,2);
Jvec_bottomtr_Co = matrix_data_bottomtr_Co(:,4);

Xvec_hole1_Co = matrix_data_hole1_Co(:,1);
Yvec_hole1_Co = matrix_data_hole1_Co(:,2);
Jvec_hole1_Co = matrix_data_hole1_Co(:,4);

Xvec_middletr_Co = matrix_data_middletr_Co(:,1);
Yvec_middletr_Co= matrix_data_middletr_Co(:,2);
Jvec_middletr_Co = matrix_data_middletr_Co(:,4);

Xvec_hole2_Co = matrix_data_hole2_Co(:,1);
Yvec_hole2_Co = matrix_data_hole2_Co(:,2);
Jvec_hole2_Co = matrix_data_hole2_Co(:,4);

Xvec_toptr_Co = matrix_data_toptr_Co(:,1);
Yvec_toptr_Co = matrix_data_toptr_Co(:,2);
Jvec_toptr_Co = matrix_data_toptr_Co(:,4);

%%
figure(1)
x=1:1:length(matrix_data_bottomtr_Co(:,4));
plot(x,matrix_data_bottomtr(:,4),'b*',x,matrix_data_bottomtr_Co(:,4),'r*')
title('bottom track')
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(6)
plot(x,(matrix_data_bottomtr_Co(:,4)-matrix_data_bottomtr(:,
    4))./matrix_data_bottomtr_Co(:,4),'g*')
title('bottom track')
xlabel('sample index')
ylabel('relative difference')
figure(11)
plot(x,(matrix_data_bottomtr_Co(:,4)-matrix_data_bottomtr(:,4)),'k*')
title('bottom track')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')

figure(2)
x=1:1:length(matrix_data_hole1_Co(:,4));
plot(x,matrix_data_hole1(:,4),'b*',x,matrix_data_hole1_Co(:,4),'r*')
title('bottom junction')
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(7)
plot(x,(matrix_data_hole1_Co(:,4)-matrix_data_hole1(:,
    4))./matrix_data_hole1_Co(:,4),'g*')
```

```
title('bottom junction')
xlabel('sample index')
ylabel('relative difference')
figure(12)
plot(x,(matrix_data_hole1_Co(:,4)-matrix_data_hole1(:,4)),'k*')
title('top junction')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')

figure(3)
x=1:1:length(matrix_data_middletr_Co(:,4));
plot(x,matrix_data_middletr(:,4),'b*',x,matrix_data_middletr_Co(:,4),'r*')
title('middle track')
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(8)
plot(x,(matrix_data_middletr_Co(:,4)-matrix_data_middletr(:,4))./matrix_data_middletr_Co(:,4),'g*')
title('middle track')
xlabel('sample index')
ylabel('relative difference')
figure(13)
plot(x,(matrix_data_middletr_Co(:,4)-matrix_data_middletr(:,4)),'k*')
title('middle track')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')

figure(4)
x=1:1:length(matrix_data_hole2_Co(:,4));
plot(x,matrix_data_hole2(:,4),'b*',x,matrix_data_hole2_Co(:,4),'r*')
title('top junction')
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(9)
plot(x,(matrix_data_hole2_Co(:,4)-matrix_data_hole2(:,4))./matrix_data_hole2_Co(:,4),'g*')
title('top junction')
xlabel('sample index')
ylabel('relative difference')
figure(14)
plot(x,(matrix_data_hole2_Co(:,4)-matrix_data_hole2(:,4)),'k*')
title('top junction')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')

figure(5)
x=1:1:length(matrix_data_toptr_Co(:,4));
plot(x,matrix_data_toptr(:,4),'b*',x,matrix_data_toptr_Co(:,4),'r*')
title('top track')
```

```
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(10)
plot(x,(matrix_data_toptr_Co(:,4)-matrix_data_toptr(:,
↪ 4))./matrix_data_toptr_Co(:,4),'g*')
title('top track')
xlabel('sample index')
ylabel('relative difference')
figure(15)
plot(x,(matrix_data_toptr_Co(:,4)-matrix_data_toptr(:,4)),'k*')
title('top track')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')
```

A.3.1.2. Parameters file

Original version

```
size_contact  scale*130e-9
track_length  scale*256e-9
track_width  scalew*20e-9
out_bound_width  scalew*10e-9
thickness_layer_Co  0.8e-9
thickness_layer_Pt  0.8e-9
hole1_width  scalew*27e-9
hole2_width  scalew*25e-9
hole1_height  scalew*20e-9
hole2_height  scalew*20e-9
xcoord_holestart  scale*113e-9
xcoord_track3start  scale*100e-9
offset_hole2  scale*0
Vappl  280e-5
nCell_X_bmtracks  30
nCell_Y_bmtracks  1
nCell_X_ttrack  30
nCell_Y_ttrack  1
nCell_X_holes  2
nCell_Y_holes  15
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
```

```
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
scale  1
scalew  1
```

First version

```
size_contact  scale*130e-9
track_length  scale*256e-9
track_width  scalew*20e-9
out_bound_width  scalew*10e-9
thickness_layer_Co  0.8e-9
thickness_layer_Pt  80e-9
hole1_width  scalew*27e-9
hole2_width  scalew*25e-9
hole1_height  scalew*20e-9
hole2_height  scalew*20e-9
xcoord_holestart  scale*113e-9
xcoord_track3start  scale*100e-9
offset_hole2  scale*0
Vappl  280e-5
nCell_X_bmtracks  30
nCell_Y_bmtracks  1
nCell_X_ttrack  30
nCell_Y_ttrack  1
nCell_X_holes  2
nCell_Y_holes  15
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
scale  6
scalew  6
```

Second version

```
size_contact  scale*130e-9
track_length  scale*256e-9
track_width  scalew*20e-9
out_bound_width  scalew*10e-9
thickness_layer_Co  0.8e-9
thickness_layer_Pt  100e-9
hole1_width  scalew*27e-9
hole2_width  scalew*25e-9
hole1_height  scalew*20e-9
hole2_height  scalew*20e-9
xcoord_holestart  scale*113e-9
xcoord_track3start  scale*100e-9
offset_hole2  scale*0
Vappl  280e-5
nCell_X_bmtracks  30
nCell_Y_bmtracks  1
nCell_X_ttrack  30
nCell_Y_ttrack  1
nCell_X_holes  2
nCell_Y_holes  15
SHangle  1
X_grid_spacing_bmtracks  track_length/nCell_X_bmtracks
Y_grid_spacing_bmtracks  track_width/nCell_Y_bmtracks
X_grid_spacing_ttrack  (track_length-xcoord_track3start)/nCell_X_ttrack
Y_grid_spacing_ttrack  track_width/nCell_Y_ttrack
X_grid_spacing_hole1  hole1_width/nCell_X_holes
Y_grid_spacing_hole1  hole1_height/nCell_Y_holes
X_grid_spacing_hole2  hole2_width/nCell_X_holes
Y_grid_spacing_hole2  hole2_height/nCell_Y_holes
scale  4
scalew  10
```

A.3.2. *And/Or* gate

A.3.2.1. Matlab code

```
clc
clear all
close all

%%
```

```
tab_data_bottomtr =
↪ readtable('currdensnorm_bottomtr.txt','Format','%f%f%f%f\n');
tab_data_middlejun =
↪ readtable('currdensnorm_middlejun.txt','Format','%f%f%f%f\n');
tab_data_toptr = readtable('currdensnorm_toptr.txt','Format','%f%f%f%f\n');

tab_data_bottomtr_Co =
↪ readtable('currdensnorm_bottomtr_Co.txt','Format','%f%f%f%f\n');
tab_data_middlejun_Co =
↪ readtable('currdensnorm_middlejun_Co.txt','Format','%f%f%f%f\n');
tab_data_toptr_Co =
↪ readtable('currdensnorm_toptr_Co.txt','Format','%f%f%f%f\n');

%%
matrix_data_bottomtr = table2array(tab_data_bottomtr);
matrix_data_middlejun = table2array(tab_data_middlejun);
matrix_data_toptr = table2array(tab_data_toptr);

matrix_data_bottomtr_Co = table2array(tab_data_bottomtr_Co);
matrix_data_middlejun_Co = table2array(tab_data_middlejun_Co);
matrix_data_toptr_Co = table2array(tab_data_toptr_Co);

%%
Xvec_bottomtr = matrix_data_bottomtr(:,1);
Yvec_bottomtr = matrix_data_bottomtr(:,2);
Jvec_bottomtr = matrix_data_bottomtr(:,4);

Xvec_middlejun = matrix_data_middlejun(:,1);
Yvec_middlejun = matrix_data_middlejun(:,2);
Jvec_middlejun = matrix_data_middlejun(:,4);

Xvec_toptr = matrix_data_toptr(:,1);
Yvec_toptr = matrix_data_toptr(:,2);
Jvec_toptr = matrix_data_toptr(:,4);


Xvec_bottomtr_Co = matrix_data_bottomtr_Co(:,1);
Yvec_bottomtr_Co = matrix_data_bottomtr_Co(:,2);
Jvec_bottomtr_Co = matrix_data_bottomtr_Co(:,4);

Xvec_middlejun_Co = matrix_data_middlejun_Co(:,1);
Yvec_middlejun_Co = matrix_data_middlejun_Co(:,2);
Jvec_middlejun_Co = matrix_data_middlejun_Co(:,4);

Xvec_toptr_Co = matrix_data_toptr_Co(:,1);
Yvec_toptr_Co = matrix_data_toptr_Co(:,2);
Jvec_toptr_Co = matrix_data_toptr_Co(:,4);

%%
```

```
figure(1)
x=1:1:length(matrix_data_bottomtr_Co(:,4));
plot(x,matrix_data_bottomtr(:,4),'b*',x,matrix_data_bottomtr_Co(:,4),'r*')
title('bottom track')
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(4)
plot(x,(matrix_data_bottomtr_Co(:,4)-matrix_data_bottomtr(:,4))./matrix_data_bottomtr_Co(:,4),'g*')
title('bottom track')
xlabel('sample index')
ylabel('relative difference')
figure(7)
plot(x,(matrix_data_bottomtr_Co(:,4)-matrix_data_bottomtr(:,4)),'k*')
title('bottom track')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')

figure(2)
x=1:1:length(matrix_data_middlejun_Co(:,4));
plot(x,matrix_data_middlejun(:,4),'b*',x,matrix_data_middlejun_Co(:,4),'r*')
title('junction')
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(5)
plot(x,(matrix_data_middlejun_Co(:,4)-matrix_data_middlejun(:,4))./matrix_data_middlejun_Co(:,4),'g*')
title('junction')
xlabel('sample index')
ylabel('relative difference')
figure(8)
plot(x,(matrix_data_middlejun_Co(:,4)-matrix_data_middlejun(:,4)),'k*')
title('junction')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')

figure(3)
x=1:1:length(matrix_data_toptr_Co(:,4));
plot(x,matrix_data_toptr(:,4),'b*',x,matrix_data_toptr_Co(:,4),'r*')
title('top track')
xlabel('sample index')
ylabel('current density [A/m^2]')
figure(6)
plot(x,(matrix_data_toptr_Co(:,4)-matrix_data_toptr(:,4))./matrix_data_toptr_Co(:,4),'g*')
title('top track')
xlabel('sample index')
ylabel('relative difference')
figure(9)
```

```
plot(x,(matrix_data_toptr_Co(:,4)-matrix_data_toptr(:,4)),'k*')
title('top track')
xlabel('sample index')
ylabel('absolute difference [A/m^2]')
```

A.3.2.2. Parameters file

Original version

```
size_contact  scale*130e-9
track_length  scale*256e-9
out_bound_width  scalew*34e-9
track_width  scalew*20e-9
thickness_layer_Co  0.8e-9
thickness_layer_Pt  0.8e-9
hole_height  scalew*20e-9
hole_width  scalew*30e-9
Vappl  280e-5
nCell_X_topbottom  30
nCell_Y_topbottom  1
nCell_X_junc  2
nCell_Y_junc  15
SHangle  1
X_grid_spacing_topbottom  track_length/nCell_X_topbottom
Y_grid_spacing_topbottom  track_width/nCell_Y_topbottom
X_grid_spacing_junc  hole_width/nCell_X_junc
Y_grid_spacing_junc  hole_height/nCell_Y_junc
scale  1
scalew  1
```

First version

```
size_contact  scale*130e-9
track_length  scale*256e-9
out_bound_width  scalew*34e-9
track_width  scalew*20e-9
thickness_layer_Co  0.8e-9
thickness_layer_Pt  80e-9
hole_height  scalew*20e-9
hole_width  scalew*30e-9
```

```
Vappl  280e-5
nCell_X_topbottom  30
nCell_Y_topbottom  1
nCell_X_junc  2
nCell_Y_junc  15
SHangle  1
X_grid_spacing_topbottom  track_length/nCell_X_topbottom
Y_grid_spacing_topbottom  track_width/nCell_Y_topbottom
X_grid_spacing_junc  hole_width/nCell_X_junc
Y_grid_spacing_junc  hole_height/nCell_Y_junc
scale  6
scalew  6
```

Second version

```
size_contact  scale*130e-9
track_length  scale*256e-9
out_bound_width  scalew*34e-9
track_width  scalew*20e-9
thickness_layer_Co  0.8e-9
thickness_layer_Pt  100e-9
hole_height  scalew*20e-9
hole_width  scalew*30e-9
Vappl  280e-5
nCell_X_topbottom  30
nCell_Y_topbottom  1
nCell_X_junc  2
nCell_Y_junc  15
SHangle  1
X_grid_spacing_topbottom  track_length/nCell_X_topbottom
Y_grid_spacing_topbottom  track_width/nCell_Y_topbottom
X_grid_spacing_junc  hole_width/nCell_X_junc
Y_grid_spacing_junc  hole_height/nCell_Y_junc
scale  4
scalew  10
```

B. Adder VHDL code

B.1. Adder - first version

B.1.1. Gates

B.1.1.1. Globals

```
package GLOBALS is
  type coordinates_xy is array (0 to 1) of real;
  type parameters_array is array(0 to 9) of coordinates_xy;
  type bool_array is array(integer range <>) of boolean;
  type real_array is array(integer range <>) of real;

  constant HORIZONTAL_SPEED : real := 150.0;  --m/s
  constant VERTICAL_SPEED : real :=  40.0;  --m/s
  constant DEPINNING_CURRENT : real := 260.0;  --nA
  constant NOTCH_DEPINNING_CURRENT : real := 3200.0;  --nA
  constant HORIZONTAL_SPEED_HIGH : real := 484.0;  --m/s
  constant SKYRMION_DIAMETER : real := 18.0;  --nm
  constant SKYRMION_MIN_DISTANCE : real := 22.0;  --nm
  constant CURRENT_LOW : real := 800.0;  --nA
  constant CURRENT_HIGH : real := 3200.0;  --nA
  constant CLOCK_LOW : time := 9.85 ns;
  constant CLOCK_HIGH : time := 150 ps;
  constant CLOCK_PERIOD : time := 10 ns;
  constant INPUTS_HIGH : time := 500 ps;
end package GLOBALS;
```

B.1.1.2. Not/Copy

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SKYRMIONNOT is
  port( INPUT : in std_logic;
        CONTROL : in std_logic;
        CURRENT : in real;
        COPY1 : out std_logic;  --TOP
        COPY2 : out std_logic;  --BOTTOM
        OUTPUT : out std_logic  --MIDDLE
      );
end entity SKYRMIONNOT;

architecture BLACKBOX of SKYRMIONNOT is
  ------------ CONSTANTS -------------------------------------------
  constant TRACK_LENGTH : real := 256.0;    --nm
  constant HOLE_X_START : real := 113.0;    --nm
  constant HOLE_X_END : real := 140.0;    --nm
  constant HOLE2_X_START : real := 113.0;    --nm
  constant HOLE2_X_END : real := 138.0;    --nm
  constant HOLE_Y_BOTTOM : real := 20.0;    --nm
  constant HOLE_Y_TOP : real := 40.0;      --nm
  constant HOLE_Y_INT1 : real := 27.0;    --nm
  constant HOLE_Y_INT2 : real := 33.0;    --nm
  constant HOLE2_Y_BOTTOM : real := 60.0;    --nm
  constant HOLE2_Y_TOP : real := 80.0;    --nm
  constant TRACK_0_Y : real := 10.0;      --nm
  constant TRACK_1_Y : real := 50.0;      --nm
  constant TRACK_2_Y : real := 90.0;      --nm

  ------------ INTERNAL SIGNALS ------------------------------------
  signal emit : std_logic_vector (2 downto 0) := "000";
  signal inputPortState: std_logic_vector(1 downto 0) := "00";
  signal ACK : std_logic := '0';
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer;

  ----------- FUNCTIONS --------------------------------------------
  function  updatePosition (elapsedTimeNs: real; actualPosition: parameters_array; currentValue : real; index: integer) return coordinates_xy is
    variable speed : coordinates_xy;
    variable output : coordinates_xy;
```

```
    variable changeTrack : boolean;
    variable skyrmion_X_position : real;
  begin
    changeTrack := false;
    output(0) := 0.0;
    output(1) := 0.0;

    if(currentValue > DEPINNING_CURRENT) then
      speed(1) := 0.0;
      speed(0) := 0.0;

      if(actualPosition(index)(0) > HOLE_X_START and actualPosition(index)(0)<
      ↪  HOLE_X_END and (actualPosition(index)(1) <= HOLE_Y_BOTTOM )) then
        changeTrack := true;
        for i in 0 to 9 loop
          if(index /= i and actualPosition(i)(0) > HOLE_X_START and
          ↪  actualPosition(i)(0) < HOLE_X_END and actualPosition(i)(1) >=
          ↪  HOLE_Y_TOP and actualPosition(i)(1) <= HOLE2_Y_BOTTOM) then
            changeTrack := false;
          end if;
        end loop;
      end if;
      if(actualPosition(index)(0) > HOLE2_X_START and actualPosition(index)(0)<
      ↪  HOLE2_X_END and (actualPosition(index)(1) > HOLE_Y_TOP and
      ↪  actualPosition(index)(1) <= HOLE2_Y_BOTTOM)) then
        changeTrack := true;
      end if;

      if (changeTrack or (actualPosition(index)(1) > HOLE_Y_BOTTOM and
      ↪  actualPosition(index)(1) < HOLE_Y_INT2) or (actualPosition(index)(1) >
      ↪  HOLE2_Y_BOTTOM and actualPosition(index)(1)<HOLE2_Y_TOP)) then
        speed(1) := VERTICAL_SPEED;
        speed(0) := 0.0;
      elsif (actualPosition(index)(1)>=HOLE_Y_INT2 and
      ↪  actualPosition(index)(1)<TRACK_1_Y) then
        speed(1) := VERTICAL_SPEED;
        speed(0) := HORIZONTAL_SPEED;
      else
        speed(1) := 0.0;
        speed(0) := HORIZONTAL_SPEED;
      end if;

      output(0) := actualPosition(index)(0)+speed(0)*elapsedTimeNs;
      output(1) := actualPosition(index)(1)+speed(1)*elapsedTimeNs;
    end if;
    return output;
  end updatePosition;

begin
```

```

RECEIVER: process(INPUT, CONTROL, ACK)
begin
  if (ACK'event and ACK='1') then
    inputPortState <= "00";
  end if;
  if (INPUT'event and INPUT='1') then
    inputPortState(1) <= '1';
  end if;
  if (CONTROL'event and CONTROL='1') then
    inputPortState(0) <= '1';
  end if;
end process;


EVOLUTION:process
  variable v_TIME : time := 0 ns;
  variable skyrmion_number : integer := 0;
  variable skyrmion_position : parameters_array;
  variable skyrmion_position_old : parameters_array;
  variable skyrmion_number_old : integer := 0;
  variable result : coordinates_xy;
  variable timeNsReal : real := 0.0;
  variable trackBusy : bool_array(2 downto 0);
  variable write_index : integer := 0;
begin
  wait for 5 ps;
  v_TIME := now - v_TIME;
  timeNsReal := 0.01; --ns
  trackBusy(0) := false;
  trackBusy(1) := false;
  trackBusy(2) := false;
  ACK <= '0';
  if (inputPortState(0) = '1') then
    skyrmion_number := skyrmion_number +1;
    skyrmion_position(skyrmion_number-1)(0) := 0.0;
    skyrmion_position(skyrmion_number-1)(1) := TRACK_0_Y;
    ACK <= '1';
  end if;

  if (inputPortState(1) = '1') then
    skyrmion_number := skyrmion_number +1;
    skyrmion_position(skyrmion_number-1)(0) := 0.0;
    skyrmion_position(skyrmion_number-1)(1) := TRACK_1_Y;
    ACK <= '1';
  end if;

  if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
    skyrmion_position_old := skyrmion_position;
```

```
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      for i in 0 to skyrmion_number_old-1 loop
        result := updatePosition(timeNsReal,skyrmion_position_old,CURRENT,i);
        if (result(0) > TRACK_LENGTH ) then
          skyrmion_number := skyrmion_number-1;
          if result(1) > HOLE2_Y_TOP then
            emit(2) <= '1' after 5 ps;
            trackBusy(2) := true;
          elsif result(1) > HOLE_Y_TOP  then
            emit(1) <= '1' after 5 ps;
            trackBusy(1) := true;
          else
            emit(0) <= '1' after 5 ps;
            trackBusy(0) := true;
          end if;
        else
          write_index := write_index + 1;
          skyrmion_position(write_index) := result;
        end if;

      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;

      if (not(trackBusy(0))) then
        emit(0) <= '0' after 5 ps;
      end if;
      if (not(trackBusy(1))) then
        emit(1) <= '0' after 5 ps;
      end if;
      if (not(trackBusy(2))) then
        emit(2) <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= "000" after 15 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

```

```
    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
  end process;


  EMITTER: process(emit)
  begin
    if(emit(0)'event and emit(0)='1') then
      COPY1<='1';
    else
      COPY1<='0' after 1 ns;
    end if;
    if(emit(1)'event and emit(1)='1') then
      OUTPUT<='1';
    else
      OUTPUT<='0' after 1 ns;
    end if;
    if(emit(2)'event and emit(2)='1') then
      COPY2<='1';
    else
      COPY2<='0' after 1 ns;
    end if;
  end process;
end BLACKBOX;
```

B.1.1.3. Line

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use work.globals.all;

entity SKYRMIONLINE is
  port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
end entity SKYRMIONLINE;

architecture BLACKBOX of SKYRMIONLINE is
  ------------ CONSTANTS -------------------------------------------
```

```
  constant TRACK_LENGTH : real := 375.0;  --nm

  ------------ INTERNAL SIGNALS ------------------------------------
  signal emit : std_logic := '0';
  signal inputPortState: std_logic:= '0';
  signal ACK : std_logic := '0';
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer;

  ------------ FUNCTIONS -------------------------------------------
  function  updatePosition (elapsedTimeNs: real; actualPosition:
   ↪  parameters_array; currentValue : real ) return parameters_array is
    variable output : parameters_array;
  begin
    if(currentValue > DEPINNING_CURRENT) then
      for i in 0 to 9 loop
        output(i)(1) := 0.0;
        output(i)(0) := actualPosition(i)(0) + HORIZONTAL_SPEED*elapsedTimeNs;
      end loop;
    end if;
    return output;
  end updatePosition;

begin

  RECEIVER: process(INPUT, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= '0';
    end if;
    if (INPUT'event and INPUT='1') then
      inputPortState <= '1';
    end if;
  end process;


  EVOLUTION:process
    variable v_TIME : time := 0 ns;
    variable skyrmion_number : integer := 0;
    variable skyrmion_number_old : integer := 0;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;
    variable results : parameters_array;
    variable timeNsReal : real := 0.0;
    variable trackBusy : boolean;
    variable write_index : integer := 0;
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
```

```
    timeNsReal := 0.01;
    trackBusy := false;
    ACK <= '0';

    if (inputPortState = '1') then
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;

    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      results := updatePosition(timeNsReal, skyrmion_position_old, CURRENT);
      for i in 0 to skyrmion_number_old-1 loop
        if (results(i)(0) > TRACK_LENGTH and not(trackBusy)) then
          skyrmion_number := skyrmion_number-1;
          emit <= '1' after 5 ps;
          trackBusy := true;
        else
          if(results(i)(0) > TRACK_LENGTH and trackBusy) then
            report "More than one skyrmion reached the output in this step; Try
            ↪  reducing the simulation step; The second skyrmion to reach the
            ↪  output will be delayed by one step";
          end if;
          write_index := write_index + 1;
          skyrmion_position(write_index) := results(i);
        end if;
      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;

      if (not(trackBusy)) then
        emit <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= '0' after 5 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
```

```
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
  end process;


  EMITTER: process(emit)
  begin
    if(emit'event and emit='1') then
      OUTPUT<='1';
    else
      OUTPUT<='0' after 10 ps;
    end if;
  end process;
end BLACKBOX;
```

B.1.1.4. Join

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use work.globals.all;

entity SKYRMIONJOIN is
  port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
end entity SKYRMIONJOIN;

architecture BLACKBOX of SKYRMIONJOIN is
  ------------ CONSTANTS -------------------------------------------
  constant TRACK_LENGTH : real := 375.0;  --nm

  ------------ INTERNAL SIGNALS ------------------------------------
  signal emit : std_logic := '0';
  signal inputPortState: std_logic_vector(1 downto 0):= "00";
  signal ACK : std_logic := '0';
```

```
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer;

  ----------- FUNCTIONS --------------------------------------------
  function  updatePosition (elapsedTimeNs: real; actualPosition:
  ↪  parameters_array; currentValue : real ) return parameters_array is
    variable output : parameters_array;
  begin
    if(currentValue > DEPINNING_CURRENT) then
      for i in 0 to 9 loop
        output(i)(1) := 0.0;
        output(i)(0) := actualPosition(i)(0) + HORIZONTAL_SPEED*elapsedTimeNs;
      end loop;
    end if;
    return output;
  end updatePosition;

begin

  RECEIVER: process(A, B, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= "00";
    end if;
    if (B'event and B='1') then
      inputPortState(0) <= '1';
    end if;
    if (A'event and A='1') then
      inputPortState(1) <= '1';
    end if;
  end process;


  EVOLUTION:process
    variable v_TIME : time := 0 ns;
    variable skyrmion_number : integer := 0;
    variable skyrmion_number_old : integer := 0;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;
    variable results : parameters_array;
    variable timeNsReal : real := 0.0;
    variable trackBusy : boolean;
    variable write_index : integer := 0;
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
    timeNsReal := 0.01;
    trackBusy := false;
    ACK <= '0';
```

```

    if (inputPortState(0) = '1') then
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;
    if (inputPortState(1) = '1') then
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;

    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      results := updatePosition(timeNsReal, skyrmion_position_old, CURRENT);
      for i in 0 to skyrmion_number_old-1 loop
        if (results(i)(0) > TRACK_LENGTH and not(trackBusy)) then
          skyrmion_number := skyrmion_number-1;
          emit <= '1' after 5 ps;
          trackBusy := true;
        else
          if(results(i)(0) > TRACK_LENGTH and trackBusy) then
            report "More than one skyrmion reached the output in this step; Join gate does not account the skyrmion collisions yet. The skyrmions will be emitted in sequence with one step distance";
          end if;
          write_index := write_index + 1;
          skyrmion_position(write_index) := results(i);
        end if;
      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;

      if (not(trackBusy)) then
        emit <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= '0' after 5 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
```

```
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
  end process;


  EMITTER: process(emit)
  begin

    if(emit'event and emit='1') then
      OUTPUT<='1';
    else
      OUTPUT<='0' after 10 ps;
    end if;
  end process;
end BLACKBOX;
```

B.1.1.5. Notch

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use work.globals.all;
use work.all;

entity SKYRMIONNOTCH is
  port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
end entity SKYRMIONNOTCH;

architecture BLACKBOX of SKYRMIONNOTCH is
  ------------ CONSTANTS --------------------------------------------
  constant TRACK_LENGTH : real := 256.0;  --nm
  constant NOTCH_POSITION: real := 113.0;  --nm
```

```

------------ INTERNAL SIGNALS ------------------------------------
signal emit : std_logic := '0';
signal inputPortState: std_logic:= '0';
signal ACK : std_logic := '0';
signal skyrmion_position_debug : parameters_array;
signal skyrmion_number_debug : integer;

----------- FUNCTIONS ---------------------------------------------
function findSkyrmionsCloserToNotch (notch_distance: real_array(9 downto 0); index: integer) return integer is
variable output : integer := 0;
begin
  for i in 0 to 9 loop
    if(notch_distance(index) < notch_distance(i)) then
      output := output + 1;
    end if;
  end loop;
  return output;
end findSkyrmionsCloserToNotch;


function  updatePosition (elapsedTimeNs: real; actualPosition: parameters_array; currentValue : real ) return parameters_array is
  variable speed : coordinates_xy;
  variable output : parameters_array;
  variable blocking_skyrmions : integer;
  variable notch_distance : real_array(9 downto 0);
  variable delta_distance : real;
begin
  if(currentValue > DEPINNING_CURRENT) then
    if (currentValue < NOTCH_DEPINNING_CURRENT) then
      for i in 0 to 9 loop
        output(i)(1) := 0.0;
        notch_distance(i) := actualPosition(i)(0)-NOTCH_POSITION;
        delta_distance := HORIZONTAL_SPEED*elapsedTimeNs;
        if(notch_distance(i) > 0.0) then
          output(i)(0) := actualPosition(i)(0)+delta_distance;
        else
          blocking_skyrmions := findSkyrmionsCloserToNotch(notch_distance, i);
          if (abs(notch_distance(i)) - real(blocking_skyrmions) * (SKYRMION_DIAMETER + SKYRMION_MIN_DISTANCE) > delta_distance) then
            output(i)(0) := actualPosition(i)(0)+ delta_distance;
          else
            output(i)(0) := NOTCH_POSITION - real(blocking_skyrmions) * (SKYRMION_DIAMETER + SKYRMION_MIN_DISTANCE);
          end if;
        end if;
```

```
        end loop;
      else
        for i in 0 to 9 loop
          output(i)(1) := 0.0;
          output(i)(0) := actualPosition(i)(0) +
          ↪  HORIZONTAL_SPEED_HIGH*elapsedTimeNs;
        end loop;
      end if;
    end if;
    return output;
  end updatePosition;

begin

  RECEIVER: process(INPUT, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= '0';
    end if;
    if (INPUT'event and INPUT='1') then
      inputPortState <= '1';
    end if;
  end process;


  EVOLUTION:process
    variable v_TIME : time := 0 ns;
    variable skyrmion_number : integer := 0;
    variable skyrmion_number_old : integer := 0;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;
    variable results : parameters_array;
    variable timeNsReal : real := 0.0;
    variable trackBusy : boolean;
    variable write_index : integer := 0;
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
    timeNsReal := 0.01;
    trackBusy := false;
    ACK <= '0';

    if (inputPortState = '1') then
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;

```

```
    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      results := updatePosition(timeNsReal, skyrmion_position_old, CURRENT);
      for i in 0 to skyrmion_number_old-1 loop
        if (results(i)(0) > TRACK_LENGTH and not(trackBusy)) then
          skyrmion_number := skyrmion_number-1;
          emit <= '1' after 5 ps;
          trackBusy := true;
        else
          if(results(i)(0) > TRACK_LENGTH and trackBusy) then
            report "More than one skyrmion reached the output in this step; Try
             ↪ reducing the simulation step; The second skyrmion to reach the
             ↪ output will be delayed by one step";
          end if;
          write_index := write_index + 1;
          skyrmion_position(write_index) := results(i);
        end if;
      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;

      if (not(trackBusy)) then
        emit <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= '0' after 5 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
  end process;


  EMITTER: process(emit)
  begin
```

```vhdl

    if(emit'event and emit='1') then
      OUTPUT<='1';
    else
      OUTPUT<='0' after 10 ps;
    end if;
  end process;
end BLACKBOX;
```

B.1.1.6. Cross

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SKYRMIONCROSS is
  port(    A:        in std_logic;
        B:        in std_logic;
        CURRENT:    in real;
        Aout:      out std_logic;
        Bout:      out std_logic
        );
end entity SKYRMIONCROSS;

architecture BLACKBOX of SKYRMIONCROSS is
begin
  Aout <= A;
  Bout <= B;
end BLACKBOX;
```

B.1.1.7. Notch_seq

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
```

```

entity SKYRMIONNOTCHseq is
  generic (N: integer := 5);
    port( INPUT: in std_logic;
      CURRENT: in real;
      OUTPUT: out std_logic
      );
end entity SKYRMIONNOTCHseq;

architecture Structure of SKYRMIONNOTCHseq is
  component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  signal internal: std_logic_vector(N-2 downto 0);

begin

  NOTCH0: SKYRMIONNOTCH port map (INPUT => INPUT, CURRENT => CURRENT, OUTPUT =>
   ↪ internal(0));
  gen_notch: for i in 1 to N-2 generate
    NOTCH_i: SKYRMIONNOTCH port map (INPUT => internal(i-1), CURRENT => CURRENT,
     ↪ OUTPUT => internal(i));
  end generate;
  NOTCH_last: SKYRMIONNOTCH port map (INPUT => internal(N-2), CURRENT =>
   ↪ CURRENT, OUTPUT => OUTPUT);

end Structure;
```

B.1.1.8. SRlatch

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SRlatch is
port(  SET:    in std_logic;
    RST:    in std_logic;
    Q:     out std_logic
```

```
);
end entity SRlatch;

architecture Behaviour of SRlatch is
begin
  latch: process (SET, RST)
  begin
    if (RST='1') then
      Q <= '0';
    elsif(SET'event and SET='1') then
      Q <= '1';
    end if;
  end process latch;
end architecture Behaviour;
```

B.1.2. Adder (16 bit)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;

entity SKYRMIONADDER is
  generic (N: integer := 16);
  port (    A     : in std_logic_vector(N-1 downto 0);
            B     : in std_logic_vector(N-1 downto 0);
      ONE1   : in std_logic;
      ONE2   : in std_logic;
      CURRENT  : in real;
      SUM     : out std_logic_vector(N-1 downto 0);
      COUT1   : out std_logic;
      COUT2   : out std_logic;
      CTRL1   : out std_logic;
      CTRL2   : out std_logic
      );
end entity SKYRMIONADDER;

architecture BLACKBOX of SKYRMIONADDER is
  component SKYRMIONHALFADDER is
      port( A:       in std_logic;
        B:       in std_logic;
        ONE1:    in std_logic;
        ONE2:    in std_logic;
```

```
        CURRENT:   in real;
        CTRL1:     out std_logic;
        COUT1:     out std_logic;
        SUM:     out std_logic;
        CTRL2:     out std_logic;
        COUT2:     out std_logic
        );
  end component SKYRMIONHALFADDER;

  component SKYRMIONFULLADDER is
      port( A     : in std_logic;
        B     : in std_logic;
        CIN1  : in std_logic;
        CIN2  : in std_logic;
        ONE1  : in std_logic;
        ONE2  : in std_logic;
        CURRENT  : in real;
        CTRL1  : out std_logic;
        SUM    : out std_logic;
        COUT1  : out std_logic;
        COUT2  : out std_logic;
        CTRL2  : out std_logic
        );
  end component SKYRMIONFULLADDER;

  component SKYRMIONCROSS is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENT:   in real;
        Aout:     out std_logic;
        Bout:     out std_logic
      );
  end component SKYRMIONCROSS;

  component SKYRMIONNOTCHseq is
    generic (N: integer := 5);
    port( INPUT: in std_logic;
        CURRENT: in real;
        OUTPUT: out std_logic
        );
  end component SKYRMIONNOTCHseq;

  component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

```

```
  signal cross01Aout, cross03Bout: std_logic;
  signal Adelayed, Bdelayed: std_logic_vector(N-1 downto 1);
  signal CTRL1vector, COUT1vector, CTRL2vector, COUT2vector: std_logic_vector(N-2 downto 0);
  signal CIN1vector, CIN2vector, ONE1vector, ONE2vector: std_logic_vector(N-1 downto 1);
  signal crossi1Aout: std_logic_vector(N-2 downto 1);

begin

  HA: SKYRMIONHALFADDER port map (A => A(0),
                  B => B(0),
                  ONE1 => ONE1,
                  ONE2 => ONE2,
                  CURRENT => CURRENT,
                  CTRL1 => CTRL1vector(0),
                  COUT1 => COUT1vector(0),
                  SUM => SUM(0),
                  CTRL2 => CTRL2vector(0),
                  COUT2 => COUT2vector(0)
                  );

  CROSS01:  SKYRMIONCROSS port map (A => CTRL1vector(0), B => COUT1vector(0), CURRENT => CURRENT, Aout => cross01Aout, Bout => CIN1vector(1));
  CROSS02:  SKYRMIONCROSS port map (A => cross01Aout, B => cross03Bout, CURRENT => CURRENT, Aout => ONE1vector(1), Bout => CIN2vector(1));
  CROSS03:  SKYRMIONCROSS port map (A => CTRL2vector(0), B => COUT2vector(0), CURRENT => CURRENT, Aout => ONE2vector(1), Bout => cross03Bout);

  NOTCHES_1_A: SKYRMIONNOTCH port map (INPUT => A(1), CURRENT => CURRENT, OUTPUT => Adelayed(1));
  NOTCHES_1_B: SKYRMIONNOTCH port map (INPUT => B(1), CURRENT => CURRENT, OUTPUT => Bdelayed(1));
  NOTCHES_2_A: SKYRMIONNOTCHseq generic map (N => 3) port map (INPUT => A(2), CURRENT => CURRENT, OUTPUT => Adelayed(2));
  NOTCHES_2_B: SKYRMIONNOTCHseq generic map (N => 3) port map (INPUT => B(2), CURRENT => CURRENT, OUTPUT => Bdelayed(2));
  NOTCHES_3_A: SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => A(3), CURRENT => CURRENT, OUTPUT => Adelayed(3));
  NOTCHES_3_B: SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => B(3), CURRENT => CURRENT, OUTPUT => Bdelayed(3));
  NOTCHES_4_A: SKYRMIONNOTCHseq generic map (N => 7) port map (INPUT => A(4), CURRENT => CURRENT, OUTPUT => Adelayed(4));
  NOTCHES_4_B: SKYRMIONNOTCHseq generic map (N => 7) port map (INPUT => B(4), CURRENT => CURRENT, OUTPUT => Bdelayed(4));
  NOTCHES_5_A: SKYRMIONNOTCHseq generic map (N => 9) port map (INPUT => A(5), CURRENT => CURRENT, OUTPUT => Adelayed(5));
  NOTCHES_5_B: SKYRMIONNOTCHseq generic map (N => 9) port map (INPUT => B(5), CURRENT => CURRENT, OUTPUT => Bdelayed(5));
```

```
NOTCHES_6_A: SKYRMIONNOTCHseq generic map (N => 11) port map (INPUT => A(6), CURRENT => CURRENT, OUTPUT => Adelayed(6));
NOTCHES_6_B: SKYRMIONNOTCHseq generic map (N => 11) port map (INPUT => B(6), CURRENT => CURRENT, OUTPUT => Bdelayed(6));
NOTCHES_7_A: SKYRMIONNOTCHseq generic map (N => 13) port map (INPUT => A(7), CURRENT => CURRENT, OUTPUT => Adelayed(7));
NOTCHES_7_B: SKYRMIONNOTCHseq generic map (N => 13) port map (INPUT => B(7), CURRENT => CURRENT, OUTPUT => Bdelayed(7));
NOTCHES_8_A: SKYRMIONNOTCHseq generic map (N => 15) port map (INPUT => A(8), CURRENT => CURRENT, OUTPUT => Adelayed(8));
NOTCHES_8_B: SKYRMIONNOTCHseq generic map (N => 15) port map (INPUT => B(8), CURRENT => CURRENT, OUTPUT => Bdelayed(8));
NOTCHES_9_A: SKYRMIONNOTCHseq generic map (N => 17) port map (INPUT => A(9), CURRENT => CURRENT, OUTPUT => Adelayed(9));
NOTCHES_9_B: SKYRMIONNOTCHseq generic map (N => 17) port map (INPUT => B(9), CURRENT => CURRENT, OUTPUT => Bdelayed(9));
NOTCHES_10_A: SKYRMIONNOTCHseq generic map (N => 19) port map (INPUT => A(10), CURRENT => CURRENT, OUTPUT => Adelayed(10));
NOTCHES_10_B: SKYRMIONNOTCHseq generic map (N => 19) port map (INPUT => B(10), CURRENT => CURRENT, OUTPUT => Bdelayed(10));
NOTCHES_11_A: SKYRMIONNOTCHseq generic map (N => 21) port map (INPUT => A(11), CURRENT => CURRENT, OUTPUT => Adelayed(11));
NOTCHES_11_B: SKYRMIONNOTCHseq generic map (N => 21) port map (INPUT => B(11), CURRENT => CURRENT, OUTPUT => Bdelayed(11));
NOTCHES_12_A: SKYRMIONNOTCHseq generic map (N => 23) port map (INPUT => A(12), CURRENT => CURRENT, OUTPUT => Adelayed(12));
NOTCHES_12_B: SKYRMIONNOTCHseq generic map (N => 23) port map (INPUT => B(12), CURRENT => CURRENT, OUTPUT => Bdelayed(12));
NOTCHES_13_A: SKYRMIONNOTCHseq generic map (N => 25) port map (INPUT => A(13), CURRENT => CURRENT, OUTPUT => Adelayed(13));
NOTCHES_13_B: SKYRMIONNOTCHseq generic map (N => 25) port map (INPUT => B(13), CURRENT => CURRENT, OUTPUT => Bdelayed(13));
NOTCHES_14_A: SKYRMIONNOTCHseq generic map (N => 27) port map (INPUT => A(14), CURRENT => CURRENT, OUTPUT => Adelayed(14));
NOTCHES_14_B: SKYRMIONNOTCHseq generic map (N => 27) port map (INPUT => B(14), CURRENT => CURRENT, OUTPUT => Bdelayed(14));
NOTCHES_15_A: SKYRMIONNOTCHseq generic map (N => 29) port map (INPUT => A(15), CURRENT => CURRENT, OUTPUT => Adelayed(15));
NOTCHES_15_B: SKYRMIONNOTCHseq generic map (N => 29) port map (INPUT => B(15), CURRENT => CURRENT, OUTPUT => Bdelayed(15));

FA: for i in 1 to N-2 generate
  FA_i: SKYRMIONFULLADDER port map (  A => Adelayed(i),
                    B => Bdelayed(i),
                    CIN1 => CIN1vector(i),
                    CIN2 => CIN2vector(i),
                    ONE1 => ONE1vector(i),
                    ONE2 => ONE2vector(i),
                    CURRENT => CURRENT,
```

```
                      CTRL1 => CTRL1vector(i),
                      SUM  => SUM(i),
                      COUT1 => COUT1vector(i),
                      COUT2 => COUT2vector(i),
                      CTRL2 => CTRL2vector(i)
                      );
    CROSS_i1: SKYRMIONCROSS port map (A => CTRL1vector(i), B => COUT1vector(i),
    ↪  CURRENT => CURRENT, Aout => crossi1Aout(i), Bout => CIN1vector(i+1));
    CROSS_i2: SKYRMIONCROSS port map (A => crossi1Aout(i), B => COUT2vector(i),
    ↪  CURRENT => CURRENT, Aout => ONE1vector(i+1), Bout => CIN2vector(i+1));
    ONE2vector(i+1) <= CTRL2vector(i);
  end generate;

  FA_last: SKYRMIONFULLADDER port map (  A => Adelayed(N-1),
                     B => Bdelayed(N-1),
                      CIN1 => CIN1vector(N-1),
                      CIN2 => CIN2vector(N-1),
                      ONE1 => ONE1vector(N-1),
                      ONE2 => ONE2vector(N-1),
                      CURRENT => CURRENT,
                      CTRL1 => CTRL1,
                      SUM  => SUM(N-1),
                      COUT1 => COUT1,
                      COUT2 => COUT2,
                      CTRL2 => CTRL2
                      );
end BLACKBOX;
```

B.1.3. FullAdder

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;

entity SKYRMIONFULLADDER is
      port( A    : in std_logic;
            B    : in std_logic;
      CIN1  : in std_logic;
      CIN2  : in std_logic;
      ONE1  : in std_logic;
      ONE2  : in std_logic;
      CURRENT  : in real;
```

```
      CTRL1  : out std_logic;
      SUM    : out std_logic;
      COUT1  : out std_logic;
      COUT2  : out std_logic;
      CTRL2  : out std_logic
      );
end entity SKYRMIONFULLADDER;

architecture BLACKBOX of SKYRMIONFULLADDER is
  component SKYRMIONNOT is
    port( INPUT : in std_logic;
        CONTROL : in std_logic;
        CURRENT : in real;
        COPY1 : out std_logic;
        COPY2 : out std_logic;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONLINE is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONJOIN is
    port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONCROSS is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENT:   in real;
        Aout:     out std_logic;
        Bout:     out std_logic
      );
  end component;
```

```

  signal   inv1copy2, inv1out, inv1copy1,
      inv2copy2, inv2out, inv2copy1,
      line1out, line2out,
      inv3copy2, inv3out, inv3copy1,
      inv4copy2, inv4out, inv4copy1,
      join1out, join2out,
      join3out,   notch21out, notch22out,
      notch23out, notch24out, notch25out,
      notch26out, notch27out,
      inv5copy2, inv5out, inv5copy1,
      join4out,
      inv6copy2, inv6out, inv6copy1,
      notch01out, notch02out, notch03out,
      notch04out, notch05out, notch06out,
      cross1Aout, cross1Bout,
      cross2Aout, cross2Bout,
      cross3Aout, cross3Bout,
      cross4Aout, cross4Bout,
      cross5Aout, cross5Bout,
      cross6Aout, cross6Bout,
      cross7Aout, cross7Bout,
      cross8Aout, cross8Bout,
      cross9Aout, cross9Bout,
      cross10Aout, cross10Bout,
      cross11Aout, cross11Bout,
      cross12Aout, cross12Bout,
      line3out, line4out, line5out,
      line6out, line7out, line8out,
      line9out, line10out, line11out: std_logic;

begin

  NOTCH01:   SKYRMIONNOTCH port map (INPUT => CIN1, CURRENT => CURRENT, OUTPUT => notch01out);
  NOTCH02:   SKYRMIONNOTCH port map (INPUT => CIN2, CURRENT => CURRENT, OUTPUT => notch02out);
  NOTCH03:   SKYRMIONNOTCH port map (INPUT => A, CURRENT => CURRENT, OUTPUT => notch03out);
  NOTCH04:   SKYRMIONNOTCH port map (INPUT => ONE1, CURRENT => CURRENT, OUTPUT => notch04out);
  NOTCH05:   SKYRMIONNOTCH port map (INPUT => B, CURRENT => CURRENT, OUTPUT => notch05out);
  NOTCH06:   SKYRMIONNOTCH port map (INPUT => ONE2, CURRENT => CURRENT, OUTPUT => notch06out);

  INV1:   SKYRMIONNOT port map (INPUT => notch03out, CONTROL => notch04out, CURRENT => CURRENT, COPY2 => inv1copy2, OUTPUT => inv1out, COPY1 => inv1copy1);
```

```
CROSS1:  SKYRMIONCROSS port map (A => inv1copy1, B => notch05out, CURRENT => CURRENT, Aout => cross1Aout, Bout => cross1Bout);
CROSS2:  SKYRMIONCROSS port map (A => cross1Aout, B => notch06out, CURRENT => CURRENT, Aout => cross2Aout, Bout => cross2Bout);
CROSS3:  SKYRMIONCROSS port map (A => inv1copy2, B => inv1out, CURRENT => CURRENT, Aout => cross3Aout, Bout => cross3Bout);
INV2:   SKYRMIONNOT port map (INPUT => cross1Bout, CONTROL => cross2Bout, CURRENT => CURRENT, COPY2 => inv2copy2, OUTPUT => inv2out, COPY1 => inv2copy1);
LINE1:   SKYRMIONLINE port map (INPUT => cross3Bout, CURRENT => CURRENT, OUTPUT => line1out);
INV3:   SKYRMIONNOT port map (INPUT => cross3Aout, CONTROL => inv2copy2, CURRENT => CURRENT, COPY2 => inv3copy2, OUTPUT => inv3out, COPY1 => inv3copy1);
LINE2:   SKYRMIONLINE port map (INPUT => inv2out, CURRENT => CURRENT, OUTPUT => line2out);
INV4:   SKYRMIONNOT port map (INPUT => inv2copy1, CONTROL => cross2Aout, CURRENT => CURRENT, COPY2 => inv4copy2, OUTPUT => inv4out, COPY1 => inv4copy1);
CROSS4:  SKYRMIONCROSS port map (A => inv3copy1, B => cross5Bout, CURRENT => CURRENT, Aout => cross4Aout, Bout => cross4Bout);
CROSS5:  SKYRMIONCROSS port map (A => line2out, B => cross6Bout, CURRENT => CURRENT, Aout => cross5Aout, Bout => cross5Bout);
CROSS6:  SKYRMIONCROSS port map (A => inv4copy2, B => inv4out, CURRENT => CURRENT, Aout => cross6Aout, Bout => cross6Bout);

JOIN1:  SKYRMIONJOIN port map (A => line1out, B => inv3copy2, CURRENT=> CURRENT, OUTPUT => join1out);
JOIN2:  SKYRMIONJOIN port map (A => inv3out, B => cross4Bout, CURRENT=> CURRENT, OUTPUT => join2out);
JOIN3:  SKYRMIONJOIN port map (A => cross5Aout, B => cross6Aout, CURRENT=> CURRENT, OUTPUT => join3out);

NOTCH21:   SKYRMIONNOTCH port map (INPUT => notch01out, CURRENT => CURRENT, OUTPUT => notch21out);
NOTCH22:   SKYRMIONNOTCH port map (INPUT => notch02out, CURRENT => CURRENT, OUTPUT => notch22out);
NOTCH23:   SKYRMIONNOTCH port map (INPUT => join1out, CURRENT => CURRENT, OUTPUT => notch23out);
NOTCH24:   SKYRMIONNOTCH port map (INPUT => join2out, CURRENT => CURRENT, OUTPUT => notch24out);
NOTCH25:   SKYRMIONNOTCH port map (INPUT => cross4Aout, CURRENT => CURRENT, OUTPUT => notch25out);
NOTCH26:   SKYRMIONNOTCH port map (INPUT => join3out, CURRENT => CURRENT, OUTPUT => notch26out);
NOTCH27:   SKYRMIONNOTCH port map (INPUT => inv4copy1, CURRENT => CURRENT, OUTPUT => notch27out);

CROSS7:  SKYRMIONCROSS port map (A => notch26out, B => notch27out, CURRENT => CURRENT, Aout => cross7Aout, Bout => cross7Bout);
```

```
CROSS8:  SKYRMIONCROSS port map (A => notch22out, B => notch23out, CURRENT => CURRENT, Aout => cross8Aout, Bout => cross8Bout);
CROSS9:  SKYRMIONCROSS port map (A => cross8Aout, B => notch24out, CURRENT => CURRENT, Aout => cross9Aout, Bout => cross9Bout);

LINE3:   SKYRMIONLINE port map (INPUT => notch21out, CURRENT => CURRENT, OUTPUT => line3out);
LINE4:   SKYRMIONLINE port map (INPUT => cross8Bout, CURRENT => CURRENT, OUTPUT => line4out);
INV5:   SKYRMIONNOT port map (INPUT => cross9Bout, CONTROL => cross9Aout, CURRENT => CURRENT, COPY2 => inv5copy2, OUTPUT => inv5out, COPY1 => inv5copy1);
LINE5:   SKYRMIONLINE port map (INPUT => notch25out, CURRENT => CURRENT, OUTPUT => line5out);
LINE6:   SKYRMIONLINE port map (INPUT => cross7Bout, CURRENT => CURRENT, OUTPUT => line6out);
LINE7:   SKYRMIONLINE port map (INPUT => cross7Aout, CURRENT => CURRENT, OUTPUT => line7out);

CROSS10:  SKYRMIONCROSS port map (A => line3out, B => line4out, CURRENT => CURRENT, Aout => cross10Aout, Bout => cross10Bout);
CROSS11:  SKYRMIONCROSS port map (A => inv5copy1, B => line5out, CURRENT => CURRENT, Aout => cross11Aout, Bout => cross11Bout);
LINE8:     SKYRMIONLINE port map (INPUT => cross10Bout, CURRENT => CURRENT, OUTPUT => line8out);
INV6:     SKYRMIONNOT port map (INPUT => cross10Aout, CONTROL => inv5copy2, CURRENT => CURRENT, COPY2 => inv6copy2, OUTPUT => inv6out, COPY1 => inv6copy1);
LINE9:     SKYRMIONLINE port map (INPUT => inv5out, CURRENT => CURRENT, OUTPUT => line9out);
LINE10:   SKYRMIONLINE port map (INPUT => cross11Bout, CURRENT => CURRENT, OUTPUT => line10out);
JOIN4:    SKYRMIONJOIN port map (A => cross11Aout, B => line6out, CURRENT=> CURRENT, OUTPUT => join4out);
LINE11:   SKYRMIONLINE port map (INPUT => line7out, CURRENT => CURRENT, OUTPUT => line11out);
CROSS12:  SKYRMIONCROSS port map (A => inv6copy1, B => line9out, CURRENT => CURRENT, Aout => cross12Aout, Bout => cross12Bout);

LINE12: SKYRMIONLINE port map (INPUT => line8out, CURRENT => CURRENT, OUTPUT => CTRL1);
JOIN6:  SKYRMIONJOIN port map (A => inv6out, B => cross12Bout, CURRENT=> CURRENT, OUTPUT => SUM);
JOIN5:  SKYRMIONJOIN port map (A => cross12Aout, B => line10out, CURRENT=> CURRENT, OUTPUT => COUT1);
LINE13: SKYRMIONLINE port map (INPUT => join4out, CURRENT => CURRENT, OUTPUT => COUT2);
LINE14: SKYRMIONLINE port map (INPUT => line11out, CURRENT => CURRENT, OUTPUT => CTRL2);
```

```

end BLACKBOX;
```

B.1.4. HalfAdder

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;

entity SKYRMIONHALFADDER is
      port( A:       in std_logic;
            B:       in std_logic;
      ONE1:    in std_logic;
      ONE2:    in std_logic;
      CURRENT:   in real;
      CTRL1:     out std_logic;
      COUT1:     out std_logic;
      SUM:     out std_logic;
      CTRL2:     out std_logic;
      COUT2:     out std_logic
      );
end entity SKYRMIONHALFADDER;

architecture BLACKBOX of SKYRMIONHALFADDER is
  component SKYRMIONNOT is
    port( INPUT : in std_logic;
        CONTROL : in std_logic;
        CURRENT : in real;
        COPY1 : out std_logic;
        COPY2 : out std_logic;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONLINE is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONJOIN is
```

```
    port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONCROSS is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENT:   in real;
        Aout:     out std_logic;
        Bout:     out std_logic
      );
  end component;

  signal  inv1copy1, inv1out, inv1copy2,
      inv2copy1, inv2out, inv2copy2,
      line1out, line2out,
      inv3copy1, inv3out, inv3copy2,
      inv4copy1, inv4out, inv4copy2,
      notch01out, notch02out, notch03out,
      notch04out, join2out,
      cross1Aout, cross1Bout,
      cross2Aout, cross2Bout, cross3Bout: std_logic;

begin

  NOTCH01:   SKYRMIONNOTCH port map (INPUT => A, CURRENT => CURRENT, OUTPUT => notch01out);
  NOTCH02:   SKYRMIONNOTCH port map (INPUT => ONE1, CURRENT => CURRENT, OUTPUT => notch02out);
  NOTCH03:   SKYRMIONNOTCH port map (INPUT => B, CURRENT => CURRENT, OUTPUT => notch03out);
  NOTCH04:   SKYRMIONNOTCH port map (INPUT => ONE2, CURRENT => CURRENT, OUTPUT => notch04out);

  INV1:   SKYRMIONNOT port map (INPUT => notch01out, CONTROL => notch02out, CURRENT => CURRENT, COPY1 => inv1copy1, OUTPUT => inv1out, COPY2 => inv1copy2);
  CROSS1:  SKYRMIONCROSS port map (A => inv1copy2, B => inv1out, CURRENT => CURRENT, Aout => cross1Aout, Bout => cross1Bout);
```

```
  INV2:   SKYRMIONNOT port map (INPUT => notch03out, CONTROL => notch04out, CURRENT => CURRENT, COPY1 => inv2copy1, OUTPUT => inv2out, COPY2 => inv2copy2);

  LINE1:   SKYRMIONLINE port map (INPUT => cross1Bout, CURRENT => CURRENT, OUTPUT => line1out);
  INV3:   SKYRMIONNOT port map (INPUT => cross1Aout, CONTROL => inv2copy2, CURRENT => CURRENT, COPY1 => inv3copy1, OUTPUT => inv3out, COPY2 => inv3copy2);
  LINE2:   SKYRMIONLINE port map (INPUT => inv2out, CURRENT => CURRENT, OUTPUT => line2out);
  INV4:   SKYRMIONNOT port map (INPUT => inv2copy1, CONTROL => inv1copy1, CURRENT => CURRENT, COPY1 => inv4copy1, OUTPUT => inv4out, COPY2 => inv4copy2);
  CROSS2:  SKYRMIONCROSS port map (A => inv3out, B => inv3copy1, CURRENT => CURRENT, Aout => cross2Aout, Bout => cross2Bout);

  JOIN1:  SKYRMIONJOIN port map (A => line1out, B => inv3copy2, CURRENT => CURRENT, OUTPUT => CTRL1);
  LINE3:   SKYRMIONLINE port map (INPUT => cross2Bout, CURRENT => CURRENT, OUTPUT => COUT1);
  JOIN2:   SKYRMIONJOIN port map (A => line2out, B => inv4copy2, CURRENT => CURRENT, OUTPUT => join2out);
  LINE4:   SKYRMIONLINE port map (INPUT => inv4copy1, CURRENT => CURRENT, OUTPUT => COUT2);
  CROSS3: SKYRMIONCROSS port map (A => join2out, B => inv4out, CURRENT => CURRENT, Aout => CTRL2, Bout => cross3Bout);
  JOIN3:   SKYRMIONJOIN port map (A => cross2Aout, B => cross3Bout, CURRENT => CURRENT, OUTPUT => SUM);

end BLACKBOX;
```

B.1.5. Testbench Adder (16 bit)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;
use ieee.numeric_std.all;
library std;
use std.textio.all;

entity testADDER is
end entity testADDER;
```

```

architecture Structure of testADDER is
  signal CURRENT : real;
  signal A, B, SUM: std_logic_vector(15 downto 0);
  signal ONE1, ONE2, CTRL1, COUT1, COUT2, CTRL2, sum_ready: std_logic;
  signal sumDetectorOut, Aval, Bval: std_logic_vector(15 downto 0);
  signal cout1DetectorOut, cout2DetectorOut, ctrl1DetectorOut, ctrl2DetectorOut:
   ↪  std_logic := '0';
  signal RST_latch: std_logic;
  signal sum_correct: std_logic_vector(15 downto 0);
  signal Avalint, Bvalint: integer:= 0;

  component SKYRMIONADDER is
  generic (N: integer := 16);
  port (   A    : in std_logic_vector(N-1 downto 0);
            B    : in std_logic_vector(N-1 downto 0);
      ONE1  : in std_logic;
      ONE2  : in std_logic;
      CURRENT  : in real;
      SUM    : out std_logic_vector(N-1 downto 0);
      COUT1  : out std_logic;
      COUT2  : out std_logic;
      CTRL1  : out std_logic;
      CTRL2  : out std_logic
      );
  end component SKYRMIONADDER;

  component SRlatch is
  port(  SET:   in std_logic;
      RST:   in std_logic;
      Q:    out std_logic
  );
  end component SRlatch;


begin

  DUT: SKYRMIONADDER generic map (N => 16) port map ( A => A,
                              B => B,
                              ONE1 => ONE1,
                              ONE2 => ONE2,
                              CURRENT => CURRENT,
                              SUM  => SUM,
                              COUT1 => COUT1,
                              COUT2 => COUT2,
                              CTRL1 => CTRL1,
                              CTRL2 => CTRL2
                              );

```

```
GENERATOR: process
  file fp_in : text open READ_MODE is "./inputs.txt";
  variable line_in : line;
  variable xA, xB, xS : integer;
  variable in1, in2: std_logic_vector(15 downto 0);
begin
  if not endfile(fp_in) then
    readline(fp_in, line_in);
    read(line_in, xA);
    in1 := std_logic_vector(to_unsigned(xA, 16));

    readline(fp_in, line_in);
    read(line_in, xB);
    in2 := std_logic_vector(to_unsigned(xB, 16));

    xS := xA+xB;
    sum_correct <= std_logic_vector(to_unsigned(xS, 16));

    A <= in1;
    B <= in2;
    Aval <= in1;
    Bval <= in2;
    Avalint <= xA;
    Bvalint <= xB;
    ONE1 <= '1';
    ONE2 <= '1';
    sum_ready <= '0';
    RST_latch <= '1';
    wait for INPUTS_HIGH;
    A <= (others => '0');
    B <= (others => '0');
    ONE1 <= '0';
    ONE2 <= '0';
    RST_latch <= '0';
    wait for (34*CLOCK_PERIOD-2*INPUTS_HIGH);
    --wait for 11 ns;
    sum_ready <= '1';
    wait for INPUTS_HIGH;
  end if;
end process GENERATOR;

SUM_latch: for i in 0 to 15 generate
  latch_i: SRlatch port map (SET => SUM(i), RST => RST_latch, Q =>
  ↪  sumDetectorOut(i));
end generate SUM_latch;

DETECTOR: process(CTRL1,COUT1,CTRL2,COUT2,sum_ready)
begin
  if(sum_ready'event and sum_ready='1') then
```

```
      cout1DetectorOut<= '0';
      cout2DetectorOut<= '0';
      ctrl1DetectorOut<= '0';
      ctrl2DetectorOut<= '0';
    end if;
    if(COUT1'event and COUT1='1') then
      cout1DetectorOut <= '1';
    end if;
    if(COUT2'event and COUT2='1') then
      cout2DetectorOut <= '1';
    end if;
    if(CTRL1'event and CTRL1='1') then
      ctrl1DetectorOut <= '1';
    end if;
    if(CTRL2'event and CTRL2='1') then
      ctrl2DetectorOut <= '1';
    end if;
  end process;

  CHECK: process(sum_ready, cout1DetectorOut, cout2DetectorOut, ctrl1DetectorOut, ctrl2DetectorOut, CURRENT)
    variable cout1 : std_logic;
    variable cout2 : std_logic;
    variable ctrl1 : std_logic;
    variable ctrl2 : std_logic;
  begin
    cout1 := cout1DetectorOut;
    cout2 := cout2DetectorOut;
    ctrl1 := ctrl1DetectorOut;
    ctrl2 := ctrl2DetectorOut;

    if(sum_ready'event and sum_ready='1') then
        assert sumDetectorOut = sum_correct
        report "Unexpected Sum for combination"&
        integer'image(Avalint) & " " &
        integer'image(Bvalint)
        severity error;
    end if;
    if(cout1DetectorOut'event and cout1DetectorOut='1') then
      assert cout1 = (Aval(15) and Bval(15))
      report "Unexpected Reminder (cout1) for combination"&
      std_logic'image(Aval(15))&" "&
      std_logic'image(Bval(15))
      severity error;
    end if;
    if(cout2DetectorOut'event and cout2DetectorOut='1') then
      assert cout2 = (Aval(15) and Bval(15))
      report "Unexpected Reminder (cout2) for combination"&
      std_logic'image(Aval(15))&" "&
```

```
      std_logic'image(Bval(15))
      severity error;
    end if;
    if(ctrl1DetectorOut'event and ctrl1DetectorOut='1') then
      assert ctrl1 = '1'
      report "Unexpected Control (ctrl1)"
      severity error;
    end if;
    if(ctrl2DetectorOut'event and ctrl2DetectorOut='1') then
      assert ctrl2 = '1'
      report "Unexpected Control (ctrl2)"
      severity error;
    end if;
  end process CHECK;

  CURRENT_GEN : process
  begin
    CURRENT <= CURRENT_LOW;
    wait for CLOCK_LOW;
    CURRENT <= CURRENT_HIGH;
    wait for CLOCK_HIGH;
  end process CURRENT_GEN;

end architecture;
```

B.2. Adder - second version

B.2.1. Gates

B.2.1.1. Globals

```
package GLOBALS is
  type coordinates_xy is array (0 to 1) of real;
  type parameters_array is array(0 to 9) of coordinates_xy;
  type bool_array is array(integer range <>) of boolean;
  type real_array is array(integer range <>) of real;

  constant HORIZONTAL_SPEED : real := 150.0;  --m/s
  constant VERTICAL_SPEED : real :=  40.0;  --m/s
  constant DEPINNING_CURRENT : real := 260.0;  --nA
  constant NOTCH_DEPINNING_CURRENT : real := 3200.0;  --nA
  constant HORIZONTAL_SPEED_HIGH : real := 484.0;  --m/s
```

```
  constant SKYRMION_DIAMETER : real := 18.0;  --nm
  constant SKYRMION_MIN_DISTANCE : real := 22.0;  --nm
  constant CURRENT_LOW : real := 800.0;  --nA
  constant CURRENT_HIGH : real := 3200.0;  --nA
  constant CLOCK_LOW : time := 9.85 ns;
  constant CLOCK_HIGH : time := 150 ps;
  constant CLOCK_PERIOD : time := 10 ns;
  constant INPUTS_HIGH : time := 500 ps;
end package GLOBALS;
```

B.2.1.2. Join

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use work.globals.all;

entity SKYRMIONJOIN is
  port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
end entity SKYRMIONJOIN;

architecture BLACKBOX of SKYRMIONJOIN is
  ------------ CONSTANTS -------------------------------------------
  constant TRACK_LENGTH : real := 375.0;  --nm

  ------------ INTERNAL SIGNALS ------------------------------------
  signal emit : std_logic := '0';
  signal inputPortState: std_logic_vector(1 downto 0):= "00";
  signal ACK : std_logic := '0';
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer;

  ----------- FUNCTIONS --------------------------------------------
  function  updatePosition (elapsedTimeNs: real; actualPosition: parameters_array; currentValue : real ) return parameters_array is
    variable output : parameters_array;
  begin
    if(currentValue > DEPINNING_CURRENT) then
```

```
      for i in 0 to 9 loop
        output(i)(1) := 0.0;
        output(i)(0) := actualPosition(i)(0) + HORIZONTAL_SPEED*elapsedTimeNs;
      end loop;
    end if;
    return output;
  end updatePosition;

begin

  RECEIVER: process(A, B, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= "00";
    end if;
    if (B'event and B='1') then
      inputPortState(0) <= '1';
    end if;
    if (A'event and A='1') then
      inputPortState(1) <= '1';
    end if;
  end process;


  EVOLUTION:process
    variable v_TIME : time := 0 ns;
    variable skyrmion_number : integer := 0;
    variable skyrmion_number_old : integer := 0;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;
    variable results : parameters_array;
    variable timeNsReal : real := 0.0;
    variable trackBusy : boolean;
    variable write_index : integer := 0;
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
    timeNsReal := 0.01;
    trackBusy := false;
    ACK <= '0';

    if (inputPortState(0) = '1') then
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;
    if (inputPortState(1) = '1') then
      skyrmion_number := skyrmion_number +1;
```

```
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;

    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      results := updatePosition(timeNsReal, skyrmion_position_old, CURRENT);
      for i in 0 to skyrmion_number_old-1 loop
        if (results(i)(0) > TRACK_LENGTH and not(trackBusy)) then
          skyrmion_number := skyrmion_number-1;
          emit <= '1' after 5 ps;
          trackBusy := true;
        else
          if(results(i)(0) > TRACK_LENGTH and trackBusy) then
            report "More than one skyrmion reached the output in this step; Join gate does not account the skyrmion collisions yet. The skyrmions will be emitted in sequence with one step distance";
          end if;
          write_index := write_index + 1;
          skyrmion_position(write_index) := results(i);
        end if;
      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;

      if (not(trackBusy)) then
        emit <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= '0' after 5 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
```

```
  end process;


  EMITTER: process(emit)
  begin

    if(emit'event and emit='1') then
      OUTPUT<='1';
    else
      OUTPUT<='0' after 10 ps;
    end if;
  end process;
end BLACKBOX;
```

B.2.2. Adder (16 bit)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;

entity SKYRMIONADDER is
  generic (N: integer := 16);
  port (    A     : in std_logic_vector(N-1 downto 0);
             B     : in std_logic_vector(N-1 downto 0);
      ONE1   : in std_logic;
      ONE2   : in std_logic;
      CURRENT  : in real;
      SUM     : out std_logic_vector(N-1 downto 0);
      COUT1   : out std_logic;
      COUT2   : out std_logic;
      CTRL1   : out std_logic;
      CTRL2   : out std_logic
      );
end entity SKYRMIONADDER;

architecture BLACKBOX of SKYRMIONADDER is
  component SKYRMIONHALFADDER is
      port( A:       in std_logic;
        B:       in std_logic;
        ONE1:    in std_logic;
        ONE2:    in std_logic;
        CURRENT:   in real;
```

```
        CTRL1:      out std_logic;
        COUT1:      out std_logic;
        SUM:      out std_logic;
        CTRL2:      out std_logic;
        COUT2:      out std_logic
        );
end component SKYRMIONHALFADDER;

component SKYRMIONFULLADDER is
    port( A     : in std_logic;
        B     : in std_logic;
        CIN1   : in std_logic;
        CIN2   : in std_logic;
        ONE1   : in std_logic;
        ONE2   : in std_logic;
        CURRENT   : in real;
        CTRL1   : out std_logic;
        SUM     : out std_logic;
        COUT1   : out std_logic;
        COUT2   : out std_logic;
        CTRL2   : out std_logic
        );
end component SKYRMIONFULLADDER;

component SKYRMIONCROSS is
  port(   A:        in std_logic;
        B:        in std_logic;
        CURRENT:    in real;
        Aout:      out std_logic;
        Bout:      out std_logic
      );
end component SKYRMIONCROSS;

component SKYRMIONNOTCHseq is
  generic (N: integer := 5);
  port( INPUT: in std_logic;
        CURRENT: in real;
        OUTPUT: out std_logic
        );
end component SKYRMIONNOTCHseq;

component SKYRMIONNOTCH is
  port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
end component;

signal cross01Aout, cross03Bout: std_logic;
```

```
  signal Adelayed, Bdelayed: std_logic_vector(N-1 downto 1);
  signal CTRL1vector, COUT1vector, CTRL2vector, COUT2vector: std_logic_vector(N-2 downto 0);
  signal CIN1vector, CIN2vector, ONE1vector, ONE2vector: std_logic_vector(N-1 downto 1);
  signal crossi1Aout: std_logic_vector(N-2 downto 1);
  type delay is array(N-1 downto 1) of integer;
  signal delay_vector: delay;

begin

  HA: SKYRMIONHALFADDER port map (A => A(0),
                B => B(0),
                ONE1 => ONE1,
                ONE2 => ONE2,
                CURRENT => CURRENT,
                CTRL1 => CTRL1vector(0),
                COUT1 => COUT1vector(0),
                SUM => SUM(0),
                CTRL2 => CTRL2vector(0),
                COUT2 => COUT2vector(0)
                );

  CROSS01:  SKYRMIONCROSS port map (A => CTRL1vector(0), B => COUT1vector(0), CURRENT => CURRENT, Aout => cross01Aout, Bout => CIN1vector(1));
  CROSS02:  SKYRMIONCROSS port map (A => cross01Aout, B => cross03Bout, CURRENT => CURRENT, Aout => ONE1vector(1), Bout => CIN2vector(1));
  CROSS03:  SKYRMIONCROSS port map (A => CTRL2vector(0), B => COUT2vector(0), CURRENT => CURRENT, Aout => ONE2vector(1), Bout => cross03Bout);

  NOTCHES_1_A: SKYRMIONNOTCHseq generic map (N => 3) port map (INPUT => A(1), CURRENT => CURRENT, OUTPUT => Adelayed(1));
  NOTCHES_1_B: SKYRMIONNOTCHseq generic map (N => 3) port map (INPUT => B(1), CURRENT => CURRENT, OUTPUT => Bdelayed(1));
  NOTCHES_2_A: SKYRMIONNOTCHseq generic map (N => 9) port map (INPUT => A(2), CURRENT => CURRENT, OUTPUT => Adelayed(2));
  NOTCHES_2_B: SKYRMIONNOTCHseq generic map (N => 9) port map (INPUT => B(2), CURRENT => CURRENT, OUTPUT => Bdelayed(2));
  NOTCHES_3_A: SKYRMIONNOTCHseq generic map (N => 15) port map (INPUT => A(3), CURRENT => CURRENT, OUTPUT => Adelayed(3));
  NOTCHES_3_B: SKYRMIONNOTCHseq generic map (N => 15) port map (INPUT => B(3), CURRENT => CURRENT, OUTPUT => Bdelayed(3));
  NOTCHES_4_A: SKYRMIONNOTCHseq generic map (N => 21) port map (INPUT => A(4), CURRENT => CURRENT, OUTPUT => Adelayed(4));
  NOTCHES_4_B: SKYRMIONNOTCHseq generic map (N => 21) port map (INPUT => B(4), CURRENT => CURRENT, OUTPUT => Bdelayed(4));
  NOTCHES_5_A: SKYRMIONNOTCHseq generic map (N => 27) port map (INPUT => A(5), CURRENT => CURRENT, OUTPUT => Adelayed(5));
  NOTCHES_5_B: SKYRMIONNOTCHseq generic map (N => 27) port map (INPUT => B(5), CURRENT => CURRENT, OUTPUT => Bdelayed(5));
```

```
NOTCHES_6_A: SKYRMIONNOTCHseq generic map (N => 33) port map (INPUT => A(6), CURRENT => CURRENT, OUTPUT => Adelayed(6));
NOTCHES_6_B: SKYRMIONNOTCHseq generic map (N => 33) port map (INPUT => B(6), CURRENT => CURRENT, OUTPUT => Bdelayed(6));
NOTCHES_7_A: SKYRMIONNOTCHseq generic map (N => 39) port map (INPUT => A(7), CURRENT => CURRENT, OUTPUT => Adelayed(7));
NOTCHES_7_B: SKYRMIONNOTCHseq generic map (N => 39) port map (INPUT => B(7), CURRENT => CURRENT, OUTPUT => Bdelayed(7));
NOTCHES_8_A: SKYRMIONNOTCHseq generic map (N => 45) port map (INPUT => A(8), CURRENT => CURRENT, OUTPUT => Adelayed(8));
NOTCHES_8_B: SKYRMIONNOTCHseq generic map (N => 45) port map (INPUT => B(8), CURRENT => CURRENT, OUTPUT => Bdelayed(8));
NOTCHES_9_A: SKYRMIONNOTCHseq generic map (N => 51) port map (INPUT => A(9), CURRENT => CURRENT, OUTPUT => Adelayed(9));
NOTCHES_9_B: SKYRMIONNOTCHseq generic map (N => 51) port map (INPUT => B(9), CURRENT => CURRENT, OUTPUT => Bdelayed(9));
NOTCHES_10_A: SKYRMIONNOTCHseq generic map (N => 57) port map (INPUT => A(10), CURRENT => CURRENT, OUTPUT => Adelayed(10));
NOTCHES_10_B: SKYRMIONNOTCHseq generic map (N => 57) port map (INPUT => B(10), CURRENT => CURRENT, OUTPUT => Bdelayed(10));
NOTCHES_11_A: SKYRMIONNOTCHseq generic map (N => 63) port map (INPUT => A(11), CURRENT => CURRENT, OUTPUT => Adelayed(11));
NOTCHES_11_B: SKYRMIONNOTCHseq generic map (N => 63) port map (INPUT => B(11), CURRENT => CURRENT, OUTPUT => Bdelayed(11));
NOTCHES_12_A: SKYRMIONNOTCHseq generic map (N => 69) port map (INPUT => A(12), CURRENT => CURRENT, OUTPUT => Adelayed(12));
NOTCHES_12_B: SKYRMIONNOTCHseq generic map (N => 69) port map (INPUT => B(12), CURRENT => CURRENT, OUTPUT => Bdelayed(12));
NOTCHES_13_A: SKYRMIONNOTCHseq generic map (N => 75) port map (INPUT => A(13), CURRENT => CURRENT, OUTPUT => Adelayed(13));
NOTCHES_13_B: SKYRMIONNOTCHseq generic map (N => 75) port map (INPUT => B(13), CURRENT => CURRENT, OUTPUT => Bdelayed(13));
NOTCHES_14_A: SKYRMIONNOTCHseq generic map (N => 81) port map (INPUT => A(14), CURRENT => CURRENT, OUTPUT => Adelayed(14));
NOTCHES_14_B: SKYRMIONNOTCHseq generic map (N => 81) port map (INPUT => B(14), CURRENT => CURRENT, OUTPUT => Bdelayed(14));
NOTCHES_15_A: SKYRMIONNOTCHseq generic map (N => 87) port map (INPUT => A(15), CURRENT => CURRENT, OUTPUT => Adelayed(15));
NOTCHES_15_B: SKYRMIONNOTCHseq generic map (N => 87) port map (INPUT => B(15), CURRENT => CURRENT, OUTPUT => Bdelayed(15));

FA: for i in 1 to N-2 generate
  FA_i: SKYRMIONFULLADDER port map (  A => Adelayed(i),
                    B => Bdelayed(i),
                    CIN1 => CIN1vector(i),
                    CIN2 => CIN2vector(i),
                    ONE1 => ONE1vector(i),
                    ONE2 => ONE2vector(i),
                    CURRENT => CURRENT,
```

```
                      CTRL1 => CTRL1vector(i),
                      SUM  => SUM(i),
                      COUT1 => COUT1vector(i),
                      COUT2 => COUT2vector(i),
                      CTRL2 => CTRL2vector(i)
                      );
    CROSS_i1: SKYRMIONCROSS port map (A => CTRL1vector(i), B => COUT1vector(i),
    ↪  CURRENT => CURRENT, Aout => crossi1Aout(i), Bout => CIN1vector(i+1));
    CROSS_i2: SKYRMIONCROSS port map (A => crossi1Aout(i), B => COUT2vector(i),
    ↪  CURRENT => CURRENT, Aout => ONE1vector(i+1), Bout => CIN2vector(i+1));
    ONE2vector(i+1) <= CTRL2vector(i);
  end generate;

  FA_last: SKYRMIONFULLADDER port map (  A => Adelayed(N-1),
                     B => Bdelayed(N-1),
                     CIN1 => CIN1vector(N-1),
                     CIN2 => CIN2vector(N-1),
                     ONE1 => ONE1vector(N-1),
                     ONE2 => ONE2vector(N-1),
                     CURRENT => CURRENT,
                     CTRL1 => CTRL1,
                     SUM  => SUM(N-1),
                     COUT1 => COUT1,
                     COUT2 => COUT2,
                     CTRL2 => CTRL2
                     );
end BLACKBOX;
```

B.2.3. FullAdder

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;

entity SKYRMIONFULLADDER is
      port( A    : in std_logic;
            B    : in std_logic;
      CIN1  : in std_logic;
      CIN2  : in std_logic;
      ONE1  : in std_logic;
      ONE2  : in std_logic;
      CURRENT  : in real;
```

```
      CTRL1  : out std_logic;
      SUM    : out std_logic;
      COUT1  : out std_logic;
      COUT2  : out std_logic;
      CTRL2  : out std_logic
      );
end entity SKYRMIONFULLADDER;

architecture BLACKBOX of SKYRMIONFULLADDER is
  component SKYRMIONNOT is
    port( INPUT : in std_logic;
        CONTROL : in std_logic;
        CURRENT : in real;
        COPY1 : out std_logic;
        COPY2 : out std_logic;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONJOIN is
    port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONCROSS is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENT:   in real;
        Aout:     out std_logic;
        Bout:     out std_logic
      );
  end component;

  signal   inv1copy2, inv1out, inv1copy1,
      inv2copy2, inv2out, inv2copy1,
      inv3copy2, inv3out, inv3copy1,
      inv4copy2, inv4out, inv4copy1,
      notch11out, notch12out, notch13out,
      notch14out, notch15out, notch16out,
```

```
        notch17out, notch18out, notch19out,
        notch110out, join1out, join2out,
        join3out,   notch21out, notch22out,
        notch23out, notch24out, notch25out,
        notch26out, notch27out,
        inv5copy2, inv5out, inv5copy1,
        notch31out, notch32out, notch33out,
        notch34out, notch35out, notch36out,
        notch37out, notch38out, join4out,
        inv6copy2, inv6out, inv6copy1,
        notch42out,
        notch43out, notch44out, notch45out,
        notch01out, notch02out, notch03out,
        notch04out, notch05out, notch06out,
        notch51out, notch52out, notch53out,
        notch54out, notch55out, notch56out,
        notch57out, notch58out,
        cross1Aout, cross1Bout,
        cross2Aout, cross2Bout,
        cross3Aout, cross3Bout,
        cross4Aout, cross4Bout,
        cross5Aout, cross5Bout,
        cross6Aout, cross6Bout,
        cross7Aout, cross7Bout,
        cross8Aout, cross8Bout,
        cross9Aout, cross9Bout,
        cross10Aout, cross10Bout,
        cross11Aout, cross11Bout,
        cross12Aout, cross12Bout: std_logic;

begin

  NOTCH01:   SKYRMIONNOTCH port map (INPUT => CIN1, CURRENT => CURRENT, OUTPUT => notch01out);
  NOTCH02:   SKYRMIONNOTCH port map (INPUT => CIN2, CURRENT => CURRENT, OUTPUT => notch02out);
  NOTCH03:   SKYRMIONNOTCH port map (INPUT => A, CURRENT => CURRENT, OUTPUT => notch03out);
  NOTCH04:   SKYRMIONNOTCH port map (INPUT => ONE1, CURRENT => CURRENT, OUTPUT => notch04out);
  NOTCH05:   SKYRMIONNOTCH port map (INPUT => B, CURRENT => CURRENT, OUTPUT => notch05out);
  NOTCH06:   SKYRMIONNOTCH port map (INPUT => ONE2, CURRENT => CURRENT, OUTPUT => notch06out);

  INV1:   SKYRMIONNOT port map (INPUT => notch03out, CONTROL => notch04out, CURRENT => CURRENT, COPY2 => inv1copy2, OUTPUT => inv1out, COPY1 => inv1copy1);
  CROSS1:  SKYRMIONCROSS port map (A => inv1copy1, B => notch05out, CURRENT => CURRENT, Aout => cross1Aout, Bout => cross1Bout);
```

```
CROSS2:  SKYRMIONCROSS port map (A => cross1Aout, B => notch06out, CURRENT => CURRENT, Aout => cross2Aout, Bout => cross2Bout);
CROSS3:  SKYRMIONCROSS port map (A => inv1copy2, B => inv1out, CURRENT => CURRENT, Aout => cross3Aout, Bout => cross3Bout);
INV2:   SKYRMIONNOT port map (INPUT => cross1Bout, CONTROL => cross2Bout, CURRENT => CURRENT, COPY2 => inv2copy2, OUTPUT => inv2out, COPY1 => inv2copy1);

NOTCH51:   SKYRMIONNOTCH port map (INPUT => notch01out, CURRENT => CURRENT, OUTPUT => notch51out);
NOTCH52:   SKYRMIONNOTCH port map (INPUT => notch02out, CURRENT => CURRENT, OUTPUT => notch52out);
NOTCH53:   SKYRMIONNOTCH port map (INPUT => cross3Bout, CURRENT => CURRENT, OUTPUT => notch53out);
NOTCH54:   SKYRMIONNOTCH port map (INPUT => cross3Aout, CURRENT => CURRENT, OUTPUT => notch54out);
NOTCH55:   SKYRMIONNOTCH port map (INPUT => inv2copy2, CURRENT => CURRENT, OUTPUT => notch55out);
NOTCH56:   SKYRMIONNOTCH port map (INPUT => inv2out, CURRENT => CURRENT, OUTPUT => notch56out);
NOTCH57:   SKYRMIONNOTCH port map (INPUT => inv2copy1, CURRENT => CURRENT, OUTPUT => notch57out);
NOTCH58:   SKYRMIONNOTCH port map (INPUT => cross2Aout, CURRENT => CURRENT, OUTPUT => notch58out);

INV3:   SKYRMIONNOT port map (INPUT => notch54out, CONTROL => notch55out, CURRENT => CURRENT, COPY2 => inv3copy2, OUTPUT => inv3out, COPY1 => inv3copy1);
INV4:   SKYRMIONNOT port map (INPUT => notch57out, CONTROL => notch58out, CURRENT => CURRENT, COPY2 => inv4copy2, OUTPUT => inv4out, COPY1 => inv4copy1);
CROSS4:  SKYRMIONCROSS port map (A => inv3copy1, B => cross5Bout, CURRENT => CURRENT, Aout => cross4Aout, Bout => cross4Bout);
CROSS5:  SKYRMIONCROSS port map (A => notch56out, B => cross6Bout, CURRENT => CURRENT, Aout => cross5Aout, Bout => cross5Bout);
CROSS6:  SKYRMIONCROSS port map (A => inv4copy2, B => inv4out, CURRENT => CURRENT, Aout => cross6Aout, Bout => cross6Bout);

NOTCH11:   SKYRMIONNOTCH port map (INPUT => notch51out, CURRENT => CURRENT, OUTPUT => notch11out);
NOTCH12:   SKYRMIONNOTCH port map (INPUT => notch52out, CURRENT => CURRENT, OUTPUT => notch12out);
NOTCH13:   SKYRMIONNOTCH port map (INPUT => notch53out, CURRENT => CURRENT, OUTPUT => notch13out);
NOTCH14:   SKYRMIONNOTCH port map (INPUT => inv3copy2, CURRENT => CURRENT, OUTPUT => notch14out);
NOTCH15:   SKYRMIONNOTCH port map (INPUT => inv3out, CURRENT => CURRENT, OUTPUT => notch15out);
NOTCH16:   SKYRMIONNOTCH port map (INPUT => cross4Bout, CURRENT => CURRENT, OUTPUT => notch16out);
```

```
NOTCH17:   SKYRMIONNOTCH port map (INPUT => cross4Aout, CURRENT => CURRENT, OUTPUT => notch17out);
NOTCH18:   SKYRMIONNOTCH port map (INPUT => cross5Aout, CURRENT => CURRENT, OUTPUT => notch18out);
NOTCH19:   SKYRMIONNOTCH port map (INPUT => cross6Aout, CURRENT => CURRENT, OUTPUT => notch19out);
NOTCH110:   SKYRMIONNOTCH port map (INPUT => inv4copy1, CURRENT => CURRENT, OUTPUT => notch110out);

JOIN1:  SKYRMIONJOIN port map (A => notch13out, B => notch14out, CURRENT=> CURRENT, OUTPUT => join1out);
JOIN2:  SKYRMIONJOIN port map (A => notch15out, B => notch16out, CURRENT=> CURRENT, OUTPUT => join2out);
JOIN3:  SKYRMIONJOIN port map (A => notch18out, B => notch19out, CURRENT=> CURRENT, OUTPUT => join3out);

NOTCH21:   SKYRMIONNOTCH port map (INPUT => notch11out, CURRENT => CURRENT, OUTPUT => notch21out);
NOTCH22:   SKYRMIONNOTCH port map (INPUT => notch12out, CURRENT => CURRENT, OUTPUT => notch22out);
NOTCH23:   SKYRMIONNOTCH port map (INPUT => join1out, CURRENT => CURRENT, OUTPUT => notch23out);
NOTCH24:   SKYRMIONNOTCH port map (INPUT => join2out, CURRENT => CURRENT, OUTPUT => notch24out);
NOTCH25:   SKYRMIONNOTCH port map (INPUT => notch17out, CURRENT => CURRENT, OUTPUT => notch25out);
NOTCH26:   SKYRMIONNOTCH port map (INPUT => join3out, CURRENT => CURRENT, OUTPUT => notch26out);
NOTCH27:   SKYRMIONNOTCH port map (INPUT => notch110out, CURRENT => CURRENT, OUTPUT => notch27out);

CROSS7:  SKYRMIONCROSS port map (A => notch26out, B => notch27out, CURRENT => CURRENT, Aout => cross7Aout, Bout => cross7Bout);
CROSS8:  SKYRMIONCROSS port map (A => notch22out, B => notch23out, CURRENT => CURRENT, Aout => cross8Aout, Bout => cross8Bout);
CROSS9:  SKYRMIONCROSS port map (A => cross8Aout, B => notch24out, CURRENT => CURRENT, Aout => cross9Aout, Bout => cross9Bout);
INV5:   SKYRMIONNOT port map (INPUT => cross9Bout, CONTROL => cross9Aout, CURRENT => CURRENT, COPY2 => inv5copy2, OUTPUT => inv5out, COPY1 => inv5copy1);

NOTCH31:   SKYRMIONNOTCH port map (INPUT => notch21out, CURRENT => CURRENT, OUTPUT => notch31out);
NOTCH32:   SKYRMIONNOTCH port map (INPUT => cross8Bout, CURRENT => CURRENT, OUTPUT => notch32out);
NOTCH33:   SKYRMIONNOTCH port map (INPUT => inv5copy2, CURRENT => CURRENT, OUTPUT => notch33out);
NOTCH34:   SKYRMIONNOTCH port map (INPUT => inv5out, CURRENT => CURRENT, OUTPUT => notch34out);
```

```
    NOTCH35:   SKYRMIONNOTCH port map (INPUT => inv5copy1, CURRENT => CURRENT, OUTPUT => notch35out);
    NOTCH36:   SKYRMIONNOTCH port map (INPUT => notch25out, CURRENT => CURRENT, OUTPUT => notch36out);
    NOTCH37:   SKYRMIONNOTCH port map (INPUT => cross7Bout, CURRENT => CURRENT, OUTPUT => notch37out);
    NOTCH38:   SKYRMIONNOTCH port map (INPUT => cross7Aout, CURRENT => CURRENT, OUTPUT => notch38out);

    CROSS10:  SKYRMIONCROSS port map (A => notch31out, B => notch32out, CURRENT => CURRENT, Aout => cross10Aout, Bout => cross10Bout);
    CROSS11:  SKYRMIONCROSS port map (A => notch35out, B => notch36out, CURRENT => CURRENT, Aout => cross11Aout, Bout => cross11Bout);
    INV6:   SKYRMIONNOT port map (INPUT => cross10Aout, CONTROL => notch33out, CURRENT => CURRENT, COPY2 => inv6copy2, OUTPUT => inv6out, COPY1 => inv6copy1);
    JOIN4:  SKYRMIONJOIN port map (A => cross11Aout, B => notch37out, CURRENT=> CURRENT, OUTPUT => join4out);
    CROSS12:  SKYRMIONCROSS port map (A => inv6copy1, B => notch34out, CURRENT => CURRENT, Aout => cross12Aout, Bout => cross12Bout);

    NOTCH41:   SKYRMIONNOTCH port map (INPUT => cross10Bout, CURRENT => CURRENT, OUTPUT => CTRL1);
    NOTCH42:   SKYRMIONNOTCH port map (INPUT => inv6out, CURRENT => CURRENT, OUTPUT => notch42out);
    NOTCH43:   SKYRMIONNOTCH port map (INPUT => cross12Bout, CURRENT => CURRENT, OUTPUT => notch43out);
    NOTCH44:   SKYRMIONNOTCH port map (INPUT => cross12Aout, CURRENT => CURRENT, OUTPUT => notch44out);
    NOTCH45:   SKYRMIONNOTCH port map (INPUT => cross11Bout, CURRENT => CURRENT, OUTPUT => notch45out);
    NOTCH46:   SKYRMIONNOTCH port map (INPUT => join4out, CURRENT => CURRENT, OUTPUT => COUT2);
    NOTCH47:   SKYRMIONNOTCH port map (INPUT => notch38out, CURRENT => CURRENT, OUTPUT => CTRL2);

    JOIN5:  SKYRMIONJOIN port map (A => notch44out, B => notch45out, CURRENT=> CURRENT, OUTPUT => COUT1);
    JOIN6:  SKYRMIONJOIN port map (A => notch42out, B => notch43out, CURRENT=> CURRENT, OUTPUT => SUM);

end BLACKBOX;
```

B.2.4. HalfAdder

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;

entity SKYRMIONHALFADDER is
      port( A:       in std_logic;
            B:       in std_logic;
      ONE1:    in std_logic;
      ONE2:    in std_logic;
      CURRENT:   in real;
      CTRL1:     out std_logic;
      COUT1:     out std_logic;
      SUM:     out std_logic;
      CTRL2:     out std_logic;
      COUT2:     out std_logic
      );
end entity SKYRMIONHALFADDER;

architecture BLACKBOX of SKYRMIONHALFADDER is
  component SKYRMIONNOT is
    port( INPUT : in std_logic;
        CONTROL : in std_logic;
        CURRENT : in real;
        COPY1 : out std_logic;
        COPY2 : out std_logic;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONJOIN is
    port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

```

```
  component SKYRMIONCROSS is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENT:   in real;
        Aout:     out std_logic;
        Bout:     out std_logic
      );
  end component;

  signal  inv1copy1, inv1out, inv1copy2,
      inv2copy1, inv2out, inv2copy2,
      inv3copy1, inv3out, inv3copy2,
      inv4copy1, inv4out, inv4copy2,
      notch01out, notch02out, notch03out,
      notch04out, join2out,
      cross1Aout, cross1Bout,
      cross2Aout, cross2Bout, cross3Bout,
      notch11out, notch12out, notch13out,
      notch14out, notch15out, notch16out,
      notch21out, notch22out, notch24out,
      notch25out, notch26out, notch27out: std_logic;

begin

  NOTCH01:   SKYRMIONNOTCH port map (INPUT => A, CURRENT => CURRENT, OUTPUT => notch01out);
  NOTCH02:   SKYRMIONNOTCH port map (INPUT => ONE1, CURRENT => CURRENT, OUTPUT => notch02out);
  NOTCH03:   SKYRMIONNOTCH port map (INPUT => B, CURRENT => CURRENT, OUTPUT => notch03out);
  NOTCH04:   SKYRMIONNOTCH port map (INPUT => ONE2, CURRENT => CURRENT, OUTPUT => notch04out);

  INV1:   SKYRMIONNOT port map (INPUT => notch01out, CONTROL => notch02out, CURRENT => CURRENT, COPY1 => inv1copy1, OUTPUT => inv1out, COPY2 => inv1copy2);
  CROSS1:  SKYRMIONCROSS port map (A => inv1copy2, B => inv1out, CURRENT => CURRENT, Aout => cross1Aout, Bout => cross1Bout);
  INV2:   SKYRMIONNOT port map (INPUT => notch03out, CONTROL => notch04out, CURRENT => CURRENT, COPY1 => inv2copy1, OUTPUT => inv2out, COPY2 => inv2copy2);

  NOTCH11:   SKYRMIONNOTCH port map (INPUT => cross1Bout, CURRENT => CURRENT, OUTPUT => notch11out);
  NOTCH12:   SKYRMIONNOTCH port map (INPUT => cross1Aout, CURRENT => CURRENT, OUTPUT => notch12out);
  NOTCH13:   SKYRMIONNOTCH port map (INPUT => inv2copy2, CURRENT => CURRENT, OUTPUT => notch13out);
  NOTCH14:   SKYRMIONNOTCH port map (INPUT => inv2out, CURRENT => CURRENT, OUTPUT => notch14out);
```

```vhdl
    NOTCH15:   SKYRMIONNOTCH port map (INPUT => inv2copy1, CURRENT => CURRENT, OUTPUT => notch15out);
    NOTCH16:   SKYRMIONNOTCH port map (INPUT => inv1copy1, CURRENT => CURRENT, OUTPUT => notch16out);

    INV3:   SKYRMIONNOT port map (INPUT => notch12out, CONTROL => notch13out, CURRENT => CURRENT, COPY1 => inv3copy1, OUTPUT => inv3out, COPY2 => inv3copy2);
    INV4:   SKYRMIONNOT port map (INPUT => notch15out, CONTROL => notch16out, CURRENT => CURRENT, COPY1 => inv4copy1, OUTPUT => inv4out, COPY2 => inv4copy2);
    CROSS2:  SKYRMIONCROSS port map (A => inv3out, B => inv3copy1, CURRENT => CURRENT, Aout => cross2Aout, Bout => cross2Bout);

    NOTCH21:   SKYRMIONNOTCH port map (INPUT => notch11out, CURRENT => CURRENT, OUTPUT => notch21out);
    NOTCH22:   SKYRMIONNOTCH port map (INPUT => inv3copy2, CURRENT => CURRENT, OUTPUT => notch22out);
    NOTCH23:   SKYRMIONNOTCH port map (INPUT => cross2Bout, CURRENT => CURRENT, OUTPUT => COUT1);
    NOTCH24:   SKYRMIONNOTCH port map (INPUT => cross2Aout, CURRENT => CURRENT, OUTPUT => notch24out);
    NOTCH25:   SKYRMIONNOTCH port map (INPUT => notch14out, CURRENT => CURRENT, OUTPUT => notch25out);
    NOTCH26:   SKYRMIONNOTCH port map (INPUT => inv4copy2, CURRENT => CURRENT, OUTPUT => notch26out);
    NOTCH27:   SKYRMIONNOTCH port map (INPUT => inv4out, CURRENT => CURRENT, OUTPUT => notch27out);
    NOTCH28:   SKYRMIONNOTCH port map (INPUT => inv4copy1, CURRENT => CURRENT, OUTPUT => COUT2);

    JOIN1:  SKYRMIONJOIN port map (A => notch21out, B => notch22out, CURRENT => CURRENT, OUTPUT => CTRL1);
    JOIN2:   SKYRMIONJOIN port map (A => notch25out, B => notch26out, CURRENT => CURRENT, OUTPUT => join2out);
    CROSS3: SKYRMIONCROSS port map (A => join2out, B => notch27out, CURRENT => CURRENT, Aout => CTRL2, Bout => cross3Bout);
    JOIN3:   SKYRMIONJOIN port map (A => notch24out, B => cross3Bout, CURRENT => CURRENT, OUTPUT => SUM);

end BLACKBOX;
```

B.2.5. Testbench Adder (16 bit)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;
use ieee.numeric_std.all;
library std;
use std.textio.all;

entity testADDER is
end entity testADDER;

architecture Structure of testADDER is
  signal CURRENT : real;
  signal A, B, SUM: std_logic_vector(15 downto 0);
  signal ONE1, ONE2, CTRL1, COUT1, COUT2, CTRL2, sum_ready: std_logic;
  signal sumDetectorOut, Aval, Bval: std_logic_vector(15 downto 0);
  signal cout1DetectorOut, cout2DetectorOut, ctrl1DetectorOut, ctrl2DetectorOut:
   ↪ std_logic := '0';
  signal RST_latch: std_logic;
  signal sum_correct: std_logic_vector(15 downto 0);
  signal Avalint, Bvalint: integer:= 0;

  component SKYRMIONADDER is
  generic (N: integer := 16);
  port (    A     : in std_logic_vector(N-1 downto 0);
             B     : in std_logic_vector(N-1 downto 0);
      ONE1  : in std_logic;
      ONE2  : in std_logic;
      CURRENT  : in real;
      SUM    : out std_logic_vector(N-1 downto 0);
      COUT1  : out std_logic;
      COUT2  : out std_logic;
      CTRL1  : out std_logic;
      CTRL2  : out std_logic
      );
  end component SKYRMIONADDER;

  component SRlatch is
  port(  SET:   in std_logic;
      RST:   in std_logic;
      Q:    out std_logic
  );
  end component SRlatch;

begin

```

```vhdl
DUT: SKYRMIONADDER generic map (N => 16) port map ( A => A,
                          B => B,
                          ONE1 => ONE1,
                          ONE2 => ONE2,
                          CURRENT => CURRENT,
                          SUM  => SUM,
                          COUT1 => COUT1,
                          COUT2 => COUT2,
                          CTRL1 => CTRL1,
                          CTRL2 => CTRL2
                          );

GENERATOR: process
  file fp_in : text open READ_MODE is "./inputs.txt";
  variable line_in : line;
  variable xA, xB, xS : integer;
  variable in1, in2: std_logic_vector(15 downto 0);
begin
  if not endfile(fp_in) then
    readline(fp_in, line_in);
    read(line_in, xA);
    in1 := std_logic_vector(to_unsigned(xA, 16));

    readline(fp_in, line_in);
    read(line_in, xB);
    in2 := std_logic_vector(to_unsigned(xB, 16));

    xS := xA+xB;
    sum_correct <= std_logic_vector(to_unsigned(xS, 16));

    A <= in1;
    B <= in2;
    Aval <= in1;
    Bval <= in2;
    Avalint <= xA;
    Bvalint <= xB;
    ONE1 <= '1';
    ONE2 <= '1';
    sum_ready <= '0';
    RST_latch <= '1';
    wait for INPUTS_HIGH;
    A <= (others => '0');
    B <= (others => '0');
    ONE1 <= '0';
    ONE2 <= '0';
    RST_latch <= '0';
    wait for (100*CLOCK_PERIOD-2*INPUTS_HIGH);
    --wait for 11 ns;
    sum_ready <= '1';
```

```
      wait for INPUTS_HIGH;
    end if;
  end process GENERATOR;

  SUM_latch: for i in 0 to 15 generate
    latch_i: SRlatch port map (SET => SUM(i), RST => RST_latch, Q => sumDetectorOut(i));
  end generate SUM_latch;

  DETECTOR: process(CTRL1,COUT1,CTRL2,COUT2,sum_ready)
  begin
    if(sum_ready'event and sum_ready='1') then
      cout1DetectorOut<= '0';
      cout2DetectorOut<= '0';
      ctrl1DetectorOut<= '0';
      ctrl2DetectorOut<= '0';
    end if;
    if(COUT1'event and COUT1='1') then
      cout1DetectorOut <= '1';
    end if;
    if(COUT2'event and COUT2='1') then
      cout2DetectorOut <= '1';
    end if;
    if(CTRL1'event and CTRL1='1') then
      ctrl1DetectorOut <= '1';
    end if;
    if(CTRL2'event and CTRL2='1') then
      ctrl2DetectorOut <= '1';
    end if;
  end process;

  CHECK: process(sum_ready, cout1DetectorOut, cout2DetectorOut, ctrl1DetectorOut, ctrl2DetectorOut, CURRENT)
    variable cout1 : std_logic;
    variable cout2 : std_logic;
    variable ctrl1 : std_logic;
    variable ctrl2 : std_logic;
  begin
    cout1 := cout1DetectorOut;
    cout2 := cout2DetectorOut;
    ctrl1 := ctrl1DetectorOut;
    ctrl2 := ctrl2DetectorOut;

    if(sum_ready'event and sum_ready='1') then
        assert sumDetectorOut = sum_correct        --assert condition report string severity severity_level;  --The assert statement tests the boolean condition. If this is false, it outputs a message containing the report string to the simulator screen:
        report "Unexpected Sum for combination"&
```

```
        integer'image(Avalint) & " " &
        integer'image(Bvalint)
        severity error;
    end if;
    if(cout1DetectorOut'event and cout1DetectorOut='1') then
      assert cout1 = (Aval(15) and Bval(15))
      report "Unexpected Reminder (cout1) for combination"&
      std_logic'image(Aval(15))&" "&
      std_logic'image(Bval(15))
      severity error;
    end if;
    if(cout2DetectorOut'event and cout2DetectorOut='1') then
      assert cout2 = (Aval(15) and Bval(15))
      report "Unexpected Reminder (cout2) for combination"&
      std_logic'image(Aval(15))&" "&
      std_logic'image(Bval(15))
      severity error;
    end if;
    if(ctrl1DetectorOut'event and ctrl1DetectorOut='1') then
      assert ctrl1 = '1'
      report "Unexpected Control (ctrl1)"
      severity error;
    end if;
    if(ctrl2DetectorOut'event and ctrl2DetectorOut='1') then
      assert ctrl2 = '1'
      report "Unexpected Control (ctrl2)"
      severity error;
    end if;
  end process CHECK;

  CURRENT_GEN : process
  begin
    CURRENT <= CURRENT_LOW;
    wait for CLOCK_LOW;
    CURRENT <= CURRENT_HIGH;
    wait for CLOCK_HIGH;
  end process CURRENT_GEN;

end architecture;
```

B.3. Adder - pipelined version

B.3.1. Adder (16 bit)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;

entity SKYRMIONADDER is
  generic (N: integer := 16);
  port (   A     : in std_logic_vector(N-1 downto 0);
            B     : in std_logic_vector(N-1 downto 0);
      ONE1   : in std_logic;
      ONE2   : in std_logic;
      CURRENT  : in real;
      SUM     : out std_logic_vector(N-1 downto 0);
      COUT1  : out std_logic;
      COUT2  : out std_logic;
      CTRL1  : out std_logic;
      CTRL2  : out std_logic
      );
end entity SKYRMIONADDER;

architecture BLACKBOX of SKYRMIONADDER is
  component SKYRMIONHALFADDER is
      port( A:       in std_logic;
        B:       in std_logic;
        ONE1:    in std_logic;
        ONE2:    in std_logic;
        CURRENT:   in real;
        CTRL1:     out std_logic;
        COUT1:     out std_logic;
        SUM:     out std_logic;
        CTRL2:     out std_logic;
        COUT2:     out std_logic
        );
  end component SKYRMIONHALFADDER;

  component SKYRMIONFULLADDER is
      port( A     : in std_logic;
        B     : in std_logic;
        CIN1   : in std_logic;
        CIN2   : in std_logic;
        ONE1   : in std_logic;
        ONE2   : in std_logic;
```

```
        CURRENT  : in real;
        CTRL1   : out std_logic;
        SUM     : out std_logic;
        COUT1   : out std_logic;
        COUT2   : out std_logic;
        CTRL2   : out std_logic
        );
  end component SKYRMIONFULLADDER;

  component SKYRMIONCROSS is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENT:   in real;
        Aout:      out std_logic;
        Bout:      out std_logic
      );
  end component SKYRMIONCROSS;

  component SKYRMIONNOTCHseq is
    generic (N: integer := 5);
    port( INPUT: in std_logic;
        CURRENT: in real;
        OUTPUT: out std_logic
        );
  end component SKYRMIONNOTCHseq;

  component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component;

  signal cross01Aout, cross03Bout: std_logic;
  signal Adelayed, Bdelayed: std_logic_vector(N-1 downto 1);
  signal CTRL1vector, COUT1vector, CTRL2vector, COUT2vector, SUMpredelay:
  ↪  std_logic_vector(N-2 downto 0);
  signal CIN1vector, CIN2vector, ONE1vector, ONE2vector: std_logic_vector(N-1
  ↪  downto 1);
  signal crossi1Aout: std_logic_vector(N-2 downto 1);
  type delay is array(N-1 downto 1) of integer;
  signal delay_vector: delay;

begin

  HA: SKYRMIONHALFADDER port map (A => A(0),
                  B => B(0),
                  ONE1 => ONE1,
                  ONE2 => ONE2,
```

```
                CURRENT => CURRENT,
                CTRL1 => CTRL1vector(0),
                COUT1 => COUT1vector(0),
                SUM => SUMpredelay(0),
                CTRL2 => CTRL2vector(0),
                COUT2 => COUT2vector(0)
                );

CROSS01:  SKYRMIONCROSS port map (A => CTRL1vector(0), B => COUT1vector(0), CURRENT => CURRENT, Aout => cross01Aout, Bout => CIN1vector(1));
CROSS02:  SKYRMIONCROSS port map (A => cross01Aout, B => cross03Bout, CURRENT => CURRENT, Aout => ONE1vector(1), Bout => CIN2vector(1));
CROSS03:  SKYRMIONCROSS port map (A => CTRL2vector(0), B => COUT2vector(0), CURRENT => CURRENT, Aout => ONE2vector(1), Bout => cross03Bout);

NOTCHES_1_A: SKYRMIONNOTCHseq generic map (N => 3) port map (INPUT => A(1), CURRENT => CURRENT, OUTPUT => Adelayed(1));
NOTCHES_1_B: SKYRMIONNOTCHseq generic map (N => 3) port map (INPUT => B(1), CURRENT => CURRENT, OUTPUT => Bdelayed(1));
NOTCHES_2_A: SKYRMIONNOTCHseq generic map (N => 9) port map (INPUT => A(2), CURRENT => CURRENT, OUTPUT => Adelayed(2));
NOTCHES_2_B: SKYRMIONNOTCHseq generic map (N => 9) port map (INPUT => B(2), CURRENT => CURRENT, OUTPUT => Bdelayed(2));
NOTCHES_3_A: SKYRMIONNOTCHseq generic map (N => 15) port map (INPUT => A(3), CURRENT => CURRENT, OUTPUT => Adelayed(3));
NOTCHES_3_B: SKYRMIONNOTCHseq generic map (N => 15) port map (INPUT => B(3), CURRENT => CURRENT, OUTPUT => Bdelayed(3));
NOTCHES_4_A: SKYRMIONNOTCHseq generic map (N => 21) port map (INPUT => A(4), CURRENT => CURRENT, OUTPUT => Adelayed(4));
NOTCHES_4_B: SKYRMIONNOTCHseq generic map (N => 21) port map (INPUT => B(4), CURRENT => CURRENT, OUTPUT => Bdelayed(4));
NOTCHES_5_A: SKYRMIONNOTCHseq generic map (N => 27) port map (INPUT => A(5), CURRENT => CURRENT, OUTPUT => Adelayed(5));
NOTCHES_5_B: SKYRMIONNOTCHseq generic map (N => 27) port map (INPUT => B(5), CURRENT => CURRENT, OUTPUT => Bdelayed(5));
NOTCHES_6_A: SKYRMIONNOTCHseq generic map (N => 33) port map (INPUT => A(6), CURRENT => CURRENT, OUTPUT => Adelayed(6));
NOTCHES_6_B: SKYRMIONNOTCHseq generic map (N => 33) port map (INPUT => B(6), CURRENT => CURRENT, OUTPUT => Bdelayed(6));
NOTCHES_7_A: SKYRMIONNOTCHseq generic map (N => 39) port map (INPUT => A(7), CURRENT => CURRENT, OUTPUT => Adelayed(7));
NOTCHES_7_B: SKYRMIONNOTCHseq generic map (N => 39) port map (INPUT => B(7), CURRENT => CURRENT, OUTPUT => Bdelayed(7));
NOTCHES_8_A: SKYRMIONNOTCHseq generic map (N => 45) port map (INPUT => A(8), CURRENT => CURRENT, OUTPUT => Adelayed(8));
NOTCHES_8_B: SKYRMIONNOTCHseq generic map (N => 45) port map (INPUT => B(8), CURRENT => CURRENT, OUTPUT => Bdelayed(8));
NOTCHES_9_A: SKYRMIONNOTCHseq generic map (N => 51) port map (INPUT => A(9), CURRENT => CURRENT, OUTPUT => Adelayed(9));
```

```
  NOTCHES_9_B: SKYRMIONNOTCHseq generic map (N => 51) port map (INPUT => B(9), CURRENT => CURRENT, OUTPUT => Bdelayed(9));
  NOTCHES_10_A: SKYRMIONNOTCHseq generic map (N => 57) port map (INPUT => A(10), CURRENT => CURRENT, OUTPUT => Adelayed(10));
  NOTCHES_10_B: SKYRMIONNOTCHseq generic map (N => 57) port map (INPUT => B(10), CURRENT => CURRENT, OUTPUT => Bdelayed(10));
  NOTCHES_11_A: SKYRMIONNOTCHseq generic map (N => 63) port map (INPUT => A(11), CURRENT => CURRENT, OUTPUT => Adelayed(11));
  NOTCHES_11_B: SKYRMIONNOTCHseq generic map (N => 63) port map (INPUT => B(11), CURRENT => CURRENT, OUTPUT => Bdelayed(11));
  NOTCHES_12_A: SKYRMIONNOTCHseq generic map (N => 69) port map (INPUT => A(12), CURRENT => CURRENT, OUTPUT => Adelayed(12));
  NOTCHES_12_B: SKYRMIONNOTCHseq generic map (N => 69) port map (INPUT => B(12), CURRENT => CURRENT, OUTPUT => Bdelayed(12));
  NOTCHES_13_A: SKYRMIONNOTCHseq generic map (N => 75) port map (INPUT => A(13), CURRENT => CURRENT, OUTPUT => Adelayed(13));
  NOTCHES_13_B: SKYRMIONNOTCHseq generic map (N => 75) port map (INPUT => B(13), CURRENT => CURRENT, OUTPUT => Bdelayed(13));
  NOTCHES_14_A: SKYRMIONNOTCHseq generic map (N => 81) port map (INPUT => A(14), CURRENT => CURRENT, OUTPUT => Adelayed(14));
  NOTCHES_14_B: SKYRMIONNOTCHseq generic map (N => 81) port map (INPUT => B(14), CURRENT => CURRENT, OUTPUT => Bdelayed(14));
  NOTCHES_15_A: SKYRMIONNOTCHseq generic map (N => 87) port map (INPUT => A(15), CURRENT => CURRENT, OUTPUT => Adelayed(15));
  NOTCHES_15_B: SKYRMIONNOTCHseq generic map (N => 87) port map (INPUT => B(15), CURRENT => CURRENT, OUTPUT => Bdelayed(15));

  FA: for i in 1 to N-2 generate
    FA_i: SKYRMIONFULLADDER port map (  A => Adelayed(i),
                      B => Bdelayed(i),
                      CIN1 => CIN1vector(i),
                      CIN2 => CIN2vector(i),
                      ONE1 => ONE1vector(i),
                      ONE2 => ONE2vector(i),
                      CURRENT => CURRENT,
                      CTRL1 => CTRL1vector(i),
                      SUM  => SUMpredelay(i),
                      COUT1 => COUT1vector(i),
                      COUT2 => COUT2vector(i),
                      CTRL2 => CTRL2vector(i)
                      );
    CROSS_i1: SKYRMIONCROSS port map (A => CTRL1vector(i), B => COUT1vector(i), CURRENT => CURRENT, Aout => crossi1Aout(i), Bout => CIN1vector(i+1));
    CROSS_i2: SKYRMIONCROSS port map (A => crossi1Aout(i), B => COUT2vector(i), CURRENT => CURRENT, Aout => ONE1vector(i+1), Bout => CIN2vector(i+1));
    ONE2vector(i+1) <= CTRL2vector(i);
  end generate;

  FA_last: SKYRMIONFULLADDER port map (  A => Adelayed(N-1),
```

```
                        B => Bdelayed(N-1),
                        CIN1 => CIN1vector(N-1),
                        CIN2 => CIN2vector(N-1),
                        ONE1 => ONE1vector(N-1),
                        ONE2 => ONE2vector(N-1),
                        CURRENT => CURRENT,
                        CTRL1 => CTRL1,
                        SUM  => SUM(N-1),
                        COUT1 => COUT1,
                        COUT2 => COUT2,
                        CTRL2 => CTRL2
                        );

  NOTCHES_14_SUM: SKYRMIONNOTCHseq generic map (N => 6) port map (INPUT => SUMpredelay(14), CURRENT => CURRENT, OUTPUT => SUM(14));
  NOTCHES_13_SUM: SKYRMIONNOTCHseq generic map (N => 12) port map (INPUT => SUMpredelay(13), CURRENT => CURRENT, OUTPUT => SUM(13));
  NOTCHES_12_SUM: SKYRMIONNOTCHseq generic map (N => 18) port map (INPUT => SUMpredelay(12), CURRENT => CURRENT, OUTPUT => SUM(12));
  NOTCHES_11_SUM: SKYRMIONNOTCHseq generic map (N => 24) port map (INPUT => SUMpredelay(11), CURRENT => CURRENT, OUTPUT => SUM(11));
  NOTCHES_10_SUM: SKYRMIONNOTCHseq generic map (N => 30) port map (INPUT => SUMpredelay(10), CURRENT => CURRENT, OUTPUT => SUM(10));
  NOTCHES_9_SUM: SKYRMIONNOTCHseq generic map (N => 36) port map (INPUT => SUMpredelay(9), CURRENT => CURRENT, OUTPUT => SUM(9));
  NOTCHES_8_SUM: SKYRMIONNOTCHseq generic map (N => 42) port map (INPUT => SUMpredelay(8), CURRENT => CURRENT, OUTPUT => SUM(8));
  NOTCHES_7_SUM: SKYRMIONNOTCHseq generic map (N => 48) port map (INPUT => SUMpredelay(7), CURRENT => CURRENT, OUTPUT => SUM(7));
  NOTCHES_6_SUM: SKYRMIONNOTCHseq generic map (N => 54) port map (INPUT => SUMpredelay(6), CURRENT => CURRENT, OUTPUT => SUM(6));
  NOTCHES_5_SUM: SKYRMIONNOTCHseq generic map (N => 60) port map (INPUT => SUMpredelay(5), CURRENT => CURRENT, OUTPUT => SUM(5));
  NOTCHES_4_SUM: SKYRMIONNOTCHseq generic map (N => 66) port map (INPUT => SUMpredelay(4), CURRENT => CURRENT, OUTPUT => SUM(4));
  NOTCHES_3_SUM: SKYRMIONNOTCHseq generic map (N => 72) port map (INPUT => SUMpredelay(3), CURRENT => CURRENT, OUTPUT => SUM(3));
  NOTCHES_2_SUM: SKYRMIONNOTCHseq generic map (N => 78) port map (INPUT => SUMpredelay(2), CURRENT => CURRENT, OUTPUT => SUM(2));
  NOTCHES_1_SUM: SKYRMIONNOTCHseq generic map (N => 84) port map (INPUT => SUMpredelay(1), CURRENT => CURRENT, OUTPUT => SUM(1));
  NOTCHES_0_SUM: SKYRMIONNOTCHseq generic map (N => 90) port map (INPUT => SUMpredelay(0), CURRENT => CURRENT, OUTPUT => SUM(0));

end BLACKBOX;
```

B.3.2. Testbench Adder (16 bit)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;
library std;
use std.textio.all;

entity testADDER is
end entity testADDER;

architecture Structure of testADDER is
  signal CURRENT : real;
  signal A, B, SUM: std_logic_vector(15 downto 0);
  signal ONE1, ONE2, CTRL1, COUT1, COUT2, CTRL2: std_logic;
  signal sum_ready: std_logic := '0';
  signal Aval, Bval: std_logic_vector(15 downto 0);
  signal sumDetectorOut: std_logic_vector(15 downto 0);
  signal cout1DetectorOut, cout2DetectorOut, ctrl1DetectorOut, ctrl2DetectorOut:
  ↪ std_logic := '0';
  signal RST_latch: std_logic;
  signal Avalint, Bvalint: integer:= 0;
  signal notch_in, notch_out, latchOut: std_logic_vector (19 downto 0);

  component SKYRMIONADDER is
  generic (N: integer := 16);
  port (    A     : in std_logic_vector(N-1 downto 0);
            B     : in std_logic_vector(N-1 downto 0);
      ONE1   : in std_logic;
      ONE2   : in std_logic;
      CURRENT  : in real;
      SUM     : out std_logic_vector(N-1 downto 0);
      COUT1   : out std_logic;
      COUT2   : out std_logic;
      CTRL1   : out std_logic;
      CTRL2   : out std_logic
      );
  end component SKYRMIONADDER;

  component SKYRMIONNOTCH is
  port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
  end component SKYRMIONNOTCH;
```

```

  component SRlatch is
  port(  SET:   in std_logic;
      RST:   in std_logic;
      Q:     out std_logic
  );
  end component SRlatch;

begin

  DUT: SKYRMIONADDER generic map (N => 16) port map (   A => A,
                          B => B,
                          ONE1 => ONE1,
                          ONE2 => ONE2,
                          CURRENT => CURRENT,
                          SUM  => SUM,
                          COUT1 => COUT1,
                          COUT2 => COUT2,
                          CTRL1 => CTRL1,
                          CTRL2 => CTRL2
                          );

  notch_in <= CTRL2 & CTRL1 & COUT2 & COUT1 & SUM;
  SYNC: for i in 0 to 19 generate
    notch_i: SKYRMIONNOTCH port map (INPUT => notch_in(i), CURRENT => CURRENT,
    ↪  OUTPUT => notch_out(i));
  end generate;

  GENERATOR: process
    file fp_in : text open READ_MODE is "./inputs.txt";
    file sum_corr_fp : text open WRITE_MODE is "./sum_corr.txt";
    file cout_corr_fp : text open WRITE_MODE is "./cout_corr.txt";
    file ctrl_corr_fp : text open WRITE_MODE is "./ctrl_corr.txt";
    variable line_in : line;
    variable line_out_sum, line_out_cout, line_out_ctrl: line;
    variable xA, xB, xS, xcout: integer;
    variable in1, in2: std_logic_vector(15 downto 0);
    variable sum_full: std_logic_vector(16 downto 0);
  begin
    if not endfile(fp_in) then
      readline(fp_in, line_in);
      read(line_in, xA);
      in1 := std_logic_vector(to_unsigned(xA, 16));

      readline(fp_in, line_in);
      read(line_in, xB);
      in2 := std_logic_vector(to_unsigned(xB, 16));

      xS := xA+xB;
```

```
      sum_full := std_logic_vector(to_unsigned(xS, 17));

      write(line_out_sum, to_integer(unsigned(sum_full(15 downto 0))));
      writeline(sum_corr_fp, line_out_sum);

      write(line_out_cout, to_bit(sum_full(16)));
      writeline(cout_corr_fp, line_out_cout);

      write(line_out_ctrl, to_bit('1'));
      writeline(ctrl_corr_fp, line_out_ctrl);

      A <= in1;
      B <= in2;
      Aval <= in1;
      Bval <= in2;
      Avalint <= xA;
      Bvalint <= xB;
      ONE1 <= '1';
      ONE2 <= '1';
      wait for INPUTS_HIGH;
      A <= (others => '0');
      B <= (others => '0');
      ONE1 <= '0';
      ONE2 <= '0';
      wait for (CLOCK_PERIOD-INPUTS_HIGH);
    end if;
  end process GENERATOR;

  sum_ready_proc: process (notch_out(19))
  begin
    if (notch_out(19)'event and notch_out(19)='1') then
      sum_ready <= '1', '0' after 500 ps;
    end if;
  end process;

  OUT_latch: for i in 0 to 19 generate
    latch_i: SRlatch port map (SET => notch_out(i), RST => RST_latch, Q =>
     ↪ latchOut(i));
  end generate OUT_latch;

  sumDetectorOut <= latchOut(15 downto 0);
  cout1DetectorOut <= latchOut(16);
  cout2DetectorOut <= latchOut(17);
  ctrl1DetectorOut <= latchOut(18);
  ctrl2DetectorOut <= latchOut(19);

  CHECK: process(sumDetectorOut, cout1DetectorOut, cout2DetectorOut,
   ↪ ctrl1DetectorOut, ctrl2DetectorOut, CURRENT)
    variable sum: std_logic_vector(15 downto 0);
```

```
    variable cout : std_logic;
    variable ctrl : std_logic;
    variable xS: integer;
    variable xcout, xctrl: boolean;

    file sum_corr_fp : text open READ_MODE is "./sum_corr.txt";
    file cout_corr_fp : text open READ_MODE is "./cout_corr.txt";
    file ctrl_corr_fp : text open READ_MODE is "./ctrl_corr.txt";
    variable line_in : line;
  begin
    if(sumDetectorOut'event) then
      if not endfile(sum_corr_fp) then
        readline(sum_corr_fp, line_in);
        read(line_in, xS);
        sum := std_logic_vector(to_unsigned(xS, 16));

        assert sumDetectorOut = sum
        report "Unexpected Sum"
        severity error;
      end if;
    end if;
    if(cout1DetectorOut'event or cout2DetectorOut'event) then
      if not endfile(cout_corr_fp) then
        readline(cout_corr_fp, line_in);
        read(line_in, xcout);
        if (xcout) then cout := '1'; else cout := '0'; end if;

        assert cout1DetectorOut = cout
        report "Unexpected Reminder (cout1)"
        severity error;

        assert cout2DetectorOut = cout
        report "Unexpected Reminder (cout2)"
        severity error;
      end if;
    end if;
    if(ctrl1DetectorOut'event or ctrl2DetectorOut'event) then
      if not endfile(ctrl_corr_fp) then
        readline(ctrl_corr_fp, line_in);
        read(line_in, xctrl);
        if (xctrl) then ctrl := '1'; else ctrl := '0'; end if;

        assert ctrl1DetectorOut = ctrl
        report "Unexpected Control (ctrl1)"
        severity error;

        assert ctrl2DetectorOut = ctrl
        report "Unexpected Control (ctrl2)"
        severity error;
```

```
      end if;
    end if;
  end process CHECK;

  CURRENT_GEN : process
  begin
    CURRENT <= CURRENT_LOW;
    RST_latch <= '0';
    wait for CLOCK_LOW;
    CURRENT <= CURRENT_HIGH;
    RST_latch <= '1';
    wait for CLOCK_HIGH;
  end process CURRENT_GEN;

end architecture;
```

C. Logic in memory VHDL code - architecture 1

C.1. Shared components

C.1.1. Globals

```
package GLOBALS is
  type coordinates_xy is array (0 to 1) of real;
  type parameters_array is array(0 to 9) of coordinates_xy;
  type bool_array is array(integer range <>) of boolean;
  type real_array is array(integer range <>) of real;

  constant HORIZONTAL_SPEED : real := 150.0;  --m/s
  constant VERTICAL_SPEED : real :=  40.0;  --m/s
  constant DEPINNING_CURRENT : real := 260.0;  --nA
  constant NOTCH_DEPINNING_CURRENT : real := 3200.0;  --nA
  constant HORIZONTAL_SPEED_HIGH : real := 484.0;   --m/2
  constant SKYRMION_DIAMETER : real := 18.0;      --nm
  constant SKYRMION_MIN_DISTANCE : real := 22.0;    --nm
  constant CURRENT_LOW : real := 800.0;  --nA
  constant CURRENT_HIGH : real := 3200.0;  --nA
  constant CLOCK_LOW : time := 5.35 ns;
  constant CLOCK_HIGH : time := 150 ps;
  constant CLOCK_PERIOD : time := 5.5 ns;
  constant INPUTS_HIGH : time := 500 ps;

end package GLOBALS;
```

C.1.2. AND/OR gate (from a previous work)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SKYRMIONH is
port (  INPUTA : in std_logic;
    INPUTB : in std_logic;
    CURRENT : in real;
    OUTPUTAND : out std_logic;
    OUTPUTOR : out std_logic
);
end entity SKYRMIONH;

architecture CENTRALLOGIC of skyrmionh is
  ------------ CONSTANTS -------------------------------------------
  constant TRACK_LENGTH : real := 256.0;    --nm
  constant HOLE_X_START : real := 113.0;    --nm
  constant HOLE_X_END : real := 143.0;    --nm
  constant HOLE_X_POSITION : real := 128.0;  --nm
  constant TRACK_0_Y : real := 10.0;      --nm
  constant TRACK_1_Y : real := 50.0;      --nm
  constant HOLE_Y_BOTTOM : real := 20.0;    --nm
  constant HOLE_Y_TOP : real := 40.0;      --nm

  ----------- FUNCTIONS --------------------------------------------

  function  updatePosition (elapsedTimeNs: real; actualPosition:
  ↪ parameters_array; currentValue : real; index: integer) return
  ↪ coordinates_xy is
    variable speed : coordinates_xy;
    variable output : coordinates_xy;
    variable changeTrack : boolean;
  begin
    changeTrack := false;
    output(0) := 0.0;
    output(1) := 0.0;
    if(currentValue > DEPINNING_CURRENT) then
      speed(1) := 0.0;
      speed(0) := 0.0;
      if(actualPosition(index)(0) > HOLE_X_START and  actualPosition(index)(0) <
      ↪ HOLE_X_END and actualPosition(index)(1) <= HOLE_Y_BOTTOM) then
        changeTrack := true;
        for i in 0 to 9 loop
          if(index /= i and actualPosition(i)(0) > HOLE_X_START and
          ↪ actualPosition(i)(0) < HOLE_X_END and actualPosition(i)(1) > 0.0)
          ↪ then
```

```
            changeTrack := false;
          end if;
        end loop;
      end if;
      if (changeTrack or (actualPosition(index)(1) > HOLE_Y_BOTTOM and
      ↪  actualPosition(index)(1) < HOLE_Y_TOP)) then
        speed(1) := VERTICAL_SPEED;
        speed(0) := 0.0;
      else
        speed(1) := 0.0;
        speed(0) := HORIZONTAL_SPEED;
      end if;
      output(0) := actualPosition(index)(0)+speed(0)*elapsedTimeNs;
      output(1) := actualPosition(index)(1)+speed(1)*elapsedTimeNs;
    end if;
    return output;
  end updatePosition;


  ------------ SIGNALS ---------------------------------
  signal ACK : std_logic := '0';
  signal inputPortState, emit : std_logic_vector(1 downto 0) := "00";
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer := 0;
begin

  RECEIVER: process(INPUTA, INPUTB, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= "00";
    end if;
    if (INPUTA'event and INPUTA='1') then
      inputPortState(1) <= '1';
    end if;
    if (INPUTB'event and INPUTB='1') then
      inputPortState(0) <= '1';
    end if;
  end process;


  EMITTER: process(emit)
  begin
    if(emit(0)'event and emit(0)='1') then
      OUTPUTAND<='1';
    else
      OUTPUTAND<='0' after 1 ns;
    end if;
    if(emit(1)'event and emit(1)='1') then
      OUTPUTOR<='1';
```

```
    else
      OUTPUTOR<='0' after 1 ns;
    end if;
  end process;

  EVOLUTION:process
    variable v_TIME : time := 0 ns;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;
    variable skyrmion_number, skyrmion_number_old, write_index : integer := 0;
    variable result : coordinates_xy;
    variable timeNsReal : real := 0.0;
    variable trackBusy : bool_array(1 downto 0);
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
    timeNsReal := 0.01;
    trackBusy(0) := false;
    trackBusy(1) := false;
    ACK <= '0';
    if (inputPortState(0) = '1') then --skyrmion detected on B
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := TRACK_0_Y;
      ACK <= '1';
    end if;

    if (inputPortState(1) = '1') then --skyrmion detected on A
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := TRACK_1_Y;
      ACK <= '1';
    end if;

    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      for i in 0 to skyrmion_number_old-1 loop
        result := updatePosition(timeNsReal,skyrmion_position_old,CURRENT,i);
        if (result(0) > TRACK_LENGTH ) then
          skyrmion_number := skyrmion_number-1;
          if result(1) > HOLE_Y_TOP then
            emit(1) <= '1' after 5 ps;  --OR=1
            trackBusy(1) := true;
          else
            emit(0) <= '1' after 5 ps;  --AND=1
            trackBusy(0) := true;
          end if;
```

```
        else
          write_index := write_index + 1;
          skyrmion_position(write_index) := result;
        end if;

      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;

      if (not(trackBusy(0))) then
        emit(0) <= '0' after 5 ps;
      end if;
      if (not(trackBusy(1))) then
        emit(1) <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= "00" after 15 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
  end process;
end CENTRALLOGIC;
```

C.1.3. Read/Write heads

C.1.3.1. MTJ_R

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
use WORK.all;
use work.globals.all;


entity MTJ_R is
port(  IN_SK:     in std_logic;
    CURRENT:  in real;
    OUT_SIGN:  out std_logic);
end entity MTJ_R;

architecture Behavioural of MTJ_R is
begin
  SK_DETECT: process (IN_SK, CURRENT)
    variable Nsk: std_logic := '0';
  begin
    if (IN_SK'event and IN_SK='1') then
      Nsk := '1';
    end if;
    if (CURRENT /= 0.0) then
      if (Nsk='1') then
        OUT_SIGN <= '1', '0' after 10 ps;
        Nsk:='0';
      else
        OUT_SIGN <= '0';
      end if;
    else
      OUT_SIGN <= '0';
    end if;
  end process SK_DETECT;
end architecture Behavioural;
```

C.1.3.2. MTJ_W

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity MTJ_W is
port(  CTRL: in std_logic;
    OUT_SK:  out std_logic);
```

```
end entity MTJ_W;

architecture Behavioural of MTJ_W is
begin
  SK_GEN: process (CTRL)
  begin
    if (CTRL'event and CTRL='1') then
      OUT_SK <= '1', '0' after 10 ps;
    else
      OUT_SK <= '0';
    end if;
  end process SK_GEN;
end architecture Behavioural;
```

C.1.3.3. MTJ_CONV

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity MTJ_CONV is
port(  IN_SK: in std_logic;
    CURRENT: in real;
    OUT_SK:  out std_logic);
end entity MTJ_CONV;

architecture Behavioural of MTJ_CONV is
  component MTJ_R is
  port(  IN_SK:     in std_logic;
      CURRENT:  in real;
      OUT_SIGN:  out std_logic);
  end component MTJ_R;

  signal SK_DETECT_SIGN: std_logic;

begin
  SK_DETECT: MTJ_R port map (IN_SK => IN_SK, CURRENT => CURRENT, OUT_SIGN =>
  ↪  SK_DETECT_SIGN);

  SK_CONV: process (SK_DETECT_SIGN)
```

```
  begin
    if (SK_DETECT_SIGN'event and SK_DETECT_SIGN='1') then
      OUT_SK <= '1', '0' after 10 ps;
    else
      OUT_SK <= '0';
    end if;
  end process SK_CONV;
end architecture Behavioural;
```

C.1.4. Crosses

C.1.4.1. CROSS with Magnus force

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SKYRMIONCROSS_Magn is
port(   A:        in std_logic;
    B:        in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end entity SKYRMIONCROSS_Magn;

architecture BLACKBOX of SKYRMIONCROSS_Magn is

begin
  process (A, B, CURRENTA, CURRENTB) is
    variable NskA, NskB: integer := 0;
  begin
    if (A'event and A='1') then
      NskA := NskA+1;
    end if;
    if (B'event and B='1') then
      NskB := NskB+1;
    end if;

    if (CURRENTA /= 0.0) then
      if (NskA = 1) then
        Aout <= '1', '0' after 9 ps;
```

```
          NskA := 0;
        else
          Aout <= '0';
        end if;
      end if;
      if (CURRENTB /= 0.0) then
        if (NskB = 1) then
          Bout <= '1', '0' after 9 ps;
          NskB := 0;
        else
          Bout <= '0';
        end if;
      end if;
    end process;
end BLACKBOX;
```

C.1.4.2. CROSS without Magnus force

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SKYRMIONCROSS_noMagn is
port(    A:        in std_logic;
    B:        in std_logic;
    CURRENTA:    in real;
    CURRENTB:   in real;
    Aout:      out std_logic;
    Bout:      out std_logic);
end entity SKYRMIONCROSS_noMagn;

architecture BLACKBOX of SKYRMIONCROSS_noMagn is
begin
  process (A, B, CURRENTA, CURRENTB)
    variable NskA, NskB: integer := 0;
  begin
    if (A'event and A='1') then
      NskA := NskA+1;
    end if;
    if (B'event and B='1') then
      NskB := NskB+1;
    end if;
```

```

    if (CURRENTA /= 0.0) then
      if (NskA = 1) then
        Aout <= '1', '0' after 10 ps;
        NskA := 0;
      else
        Aout <= '0';
      end if;
    end if;
    if (CURRENTB /= 0.0) then
      if (NskB = 1) then
        Bout <= '1', '0' after 10 ps;
        NskB := 0;
      else
        Bout <= '0';
      end if;
    end if;
  end process;
end BLACKBOX;
```

C.1.5. Duplication/Merging elements

C.1.5.1. Duplication element

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity skyrmionDUPLICATE is
port(  IN_SK:       in std_logic;
    CURRENT:    in real;
    OUT_SK_TOP:    out std_logic;
    OUT_SK_BOTTOM:  out std_logic);
end entity skyrmionDUPLICATE;

architecture Behavioural of skyrmionDUPLICATE is
begin
  SK_DUPL: process (IN_SK, CURRENT)
    variable Nsk: integer := 0;
  begin
```

```
    if (IN_SK'event and IN_SK='1') then
      Nsk := Nsk+1;
    end if;

    if (CURRENT /= 0.0) then
      if (Nsk /= 0) then
        OUT_SK_TOP <= '1', '0' after 10 ps;
        OUT_SK_BOTTOM <= '1', '0' after 10 ps;
        Nsk:=Nsk-1;
      else
        OUT_SK_TOP <= '0';
        OUT_SK_BOTTOM <= '0';
      end if;
    else
      OUT_SK_TOP <= '0';
      OUT_SK_BOTTOM <= '0';
    end if;
  end process SK_DUPL;
end architecture Behavioural;
```

C.1.5.2. Merging element

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity skyrmionMERGE is
port(  IN_SK_TOP:     in std_logic;
    IN_SK_BOTTOM:  in std_logic;
    CURRENT:    in real;
    OUT_SK:      out std_logic);
end entity skyrmionMERGE;

architecture Behavioural of skyrmionMERGE is
begin
  SK_MERGE: process (IN_SK_TOP, IN_SK_BOTTOM, CURRENT)
    variable Nsk: integer := 0;
  begin
    if (IN_SK_TOP'event and IN_SK_TOP='1') then
      Nsk := Nsk+1;
```

```
    end if;
    if (IN_SK_BOTTOM'event and IN_SK_BOTTOM='1') then
      Nsk := Nsk+1;
    end if;

    if (CURRENT /= 0.0) then
      if (Nsk=1 or Nsk=2) then
        OUT_SK <= '1', '0' after 10 ps;
        Nsk := 0;
      else
        OUT_SK <= '0';
      end if;
    else
      OUT_SK <= '0';
    end if;
  end process SK_MERGE;
end architecture Behavioural;
```

C.1.6. Deviation element

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity skyrmionDEVIATION is
port(  IN_SK:      in std_logic;
    CURRENT:   in real;
    CURRENTDEV:  in real;
    OUT_SK:     out std_logic;
    OUT_SK_DEV:  out std_logic);
end entity skyrmionDEVIATION;

architecture Behavioural of skyrmionDEVIATION is
signal Nsks: integer;
signal outens: std_logic;
begin
  SK_DEV: process (IN_SK, CURRENT, CURRENTDEV)
    variable Nsk: integer := 0;
    variable out_en: std_logic := '0';
  begin
```

```
    if (IN_SK'event and IN_SK='1') then
      Nsk := Nsk+1;
    end if;

    if (CURRENT = 0.0 and CURRENTDEV = 0.0) then
      OUT_SK <= '0';
      OUT_SK_DEV <= '0';
    end if;

    if(out_en = '1') then
      if (CURRENTDEV'event and CURRENTDEV /= 0.0) then
        if (Nsk=1) then
          OUT_SK <= '0';
          OUT_SK_DEV <= '1', '0' after 10 ps;
          Nsk := Nsk-1;
          out_en := '0';
        else
          OUT_SK <= '0';
          OUT_SK_DEV <= '0';
          out_en := '0';
        end if;
      elsif (CURRENT'event and CURRENT /= 0.0) then
        if (Nsk=1) then
          OUT_SK <= '1', '0' after 10 ps;
          OUT_SK_DEV <= '0';
          Nsk := Nsk-1;
          out_en := '0';
        else
          OUT_SK <= '0';
          OUT_SK_DEV <= '0';
          out_en := '0';
        end if;
      end if;
    end if;

    if (CURRENT'event and CURRENT /= 0.0) then
      if (out_en = '0' and Nsk /= 0) then
        out_en:='1';
      end if;
    end if;

    Nsks <= Nsk;
    outens <= out_en;

  end process SK_DEV;
end architecture Behavioural;
```

C.1.7. Voltage generators

C.1.7.1. Voltage_genL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity voltage_genL is
port(  CTRL: in std_logic;
    CURRENT: out real);
end entity voltage_genL;

architecture Behavioural of voltage_genL is
begin
  CURR_GEN: process (CTRL)
  begin
    if (CTRL='1') then
      CURRENT <= CURRENT_LOW;
    else
      CURRENT <= 0.0;
    end if;
  end process CURR_GEN;
end architecture Behavioural;
```

C.1.7.2. Voltage_genH

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity voltage_genH is
port(  CTRL: in std_logic;
    CURRENT: out real);
end entity voltage_genH;
```

```vhdl

architecture Behavioural of voltage_genH is
begin
  CURR_GEN: process (CTRL)
  begin
    if (CTRL='1') then
      CURRENT <= CURRENT_HIGH;
    else
      CURRENT <= 0.0;
    end if;
  end process CURR_GEN;
end architecture Behavioural;
```

C.1.7.3. Vclock_gen

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity vclock_gen is
port(  CTRL: in std_logic;
    CURRENTclk: in real;
    CURRENT:  out real);
end entity vclock_gen;

architecture Behavioural of vclock_gen is
begin
  CURR_GEN: process (CTRL, CURRENTclk)
  begin
    if (CTRL='1') then
      CURRENT <= CURRENTclk;
    else
      CURRENT <= 0.0;
    end if;
  end process CURR_GEN;
end architecture Behavioural;
```

C.1.8. Tanks

C.1.8.1. Bottom tank

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity tank_bottom is
port(  IN1, IN2, IN3, IN4, IN5, IN6, IN7: in std_logic;  --from left to right
    CURRENT: in real;
    tankOUT1, tankOUT2, tankOUT3: out std_logic);    --priority order: from right to left
end entity tank_bottom;

architecture Behavioural of tank_bottom is
  component SKYRMIONNOTCH is
  port( INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  signal OUT1, OUT2, OUT3: std_logic;
  signal Nsks: integer;

begin
  process (IN1, IN2, IN3, IN4, IN5, IN6, IN7, CURRENT)
    variable Nsk: integer := 0;
    variable out_en, OUT1_var, OUT2_var, OUT3_var: std_logic := '0';
  begin
    if (IN1'event and IN1='1') then
      Nsk := Nsk+1;
    end if;
    if (IN2'event and IN2='1') then
      Nsk := Nsk+1;
    end if;
    if (IN3'event and IN3='1') then
      Nsk := Nsk+1;
    end if;
    if (IN4'event and IN4='1') then
      Nsk := Nsk+1;
    end if;
    if (IN5'event and IN5='1') then
      Nsk := Nsk+1;
```

```
    end if;
    if (IN6'event and IN6='1') then
      Nsk := Nsk+1;
    end if;
    if (IN7'event and IN7='1') then
      Nsk := Nsk+1;
    end if;

    Nsks <= Nsk;

    if (CURRENT'event and CURRENT = CURRENT_LOW) then
      if (out_en='0') then
        out_en := '1';
        case Nsk is
          when 1 =>   OUT1_var := '1';
                OUT2_var := '0';
                OUT3_var := '0';
                Nsk := 0;
          when 2 =>   OUT1_var := '1';
                OUT2_var := '1';
                OUT3_var := '0';
                Nsk := 0;
          when 3 =>   OUT1_var := '1';
                OUT2_var := '1';
                OUT3_var := '1';
                Nsk := 0;
          when 0 =>   OUT1_var := '0';
                OUT2_var := '0';
                OUT3_var := '0';
                Nsk := 0;
          when others =>   OUT1_var := '1';
                  OUT2_var := '1';
                  OUT3_var := '1';
                  Nsk := Nsk-3;
        end case;
      end if;
    end if;

    if (CURRENT'event and CURRENT = 0.0) then
      out_en := '0';
    end if;

    if (OUT1_var = '1' or OUT2_var = '1' or OUT3_var = '1') then
      if (OUT1_var = '1') then
        OUT1_var := '0';
        OUT1 <= '1', '0' after 10 ps;
      else
        OUT1 <= '0';
      end if;
```

```
      if (OUT2_var = '1') then
        OUT2_var := '0';
        OUT2 <= '1', '0' after 10 ps;
      else
        OUT2 <= '0';
      end if;
      if (OUT3_var = '1') then
        OUT3_var := '0';
        OUT3 <= '1', '0' after 10 ps;
      else
        OUT3 <= '0';
      end if;
    else
      OUT1 <= '0';
      OUT2 <= '0';
      OUT3 <= '0';
    end if;
  end process;

  notch1: SKYRMIONNOTCH port map (INPUT => OUT1, CURRENT => CURRENT, OUTPUT =>
  ↪ tankOUT1);
  notch2: SKYRMIONNOTCH port map (INPUT => OUT2, CURRENT => CURRENT, OUTPUT =>
  ↪ tankOUT2);
  notch3: SKYRMIONNOTCH port map (INPUT => OUT3, CURRENT => CURRENT, OUTPUT =>
  ↪ tankOUT3);

end architecture Behavioural;
```

C.1.8.2. Top tank

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity tank_top is
port(  IN1, IN2, IN3: in std_logic; --from right to left
    CURRENT: in real;
    tankOUT: out std_logic);
end entity tank_top;

```

```
architecture Behavioural of tank_top is
  component SKYRMIONNOTCH is
  port( INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  signal OUT1: std_logic;
  signal Nsks: integer;

begin
  process (IN1, IN2, IN3, CURRENT)
    variable Nsk: integer := 0;
    variable out_en, OUT1_var: std_logic := '0';
  begin
    if (IN1'event and IN1='1') then
      Nsk := Nsk+1;
    end if;
    if (IN2'event and IN2='1') then
      Nsk := Nsk+1;
    end if;
    if (IN3'event and IN3='1') then
      Nsk := Nsk+1;
    end if;

    Nsks <= Nsk;

    if (CURRENT'event and CURRENT = CURRENT_LOW) then
      if (out_en='0') then
        out_en := '1';
        case Nsk is
          when 1 =>   OUT1_var := '1';
                Nsk := 0;
          when 0 =>   OUT1_var := '0';
                Nsk := 0;
          when others =>   OUT1_var := '1';
                Nsk := Nsk-1;
        end case;
      end if;
    end if;

    if (CURRENT'event and CURRENT = 0.0) then
      out_en := '0';
    end if;

    if (OUT1_var = '1') then
      OUT1_var := '0';
      OUT1 <= '1', '0' after 10 ps;
    else
```

```
      OUT1 <= '0';
    end if;

  end process;

  notch1: SKYRMIONNOTCH port map (INPUT => OUT1, CURRENT => CURRENT, OUTPUT =>
  ↪  tankOUT);

end architecture Behavioural;
```

C.1.9. Multiplexers

C.1.9.1. Results multiplexer

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity mux1 is
port(  inT, inM, inB: in std_logic;
    CURRENT: in real;
    CURRENT_V1, CURRENT_V2, CURRENT_V3: in real;
    selection: in std_logic_vector(1 downto 0);
    muxOUT, V1OUT, V3OUT, V2OUTt, V2OUTb: out std_logic);
end entity mux1;

architecture Behavioural of mux1 is
begin
  process (inT, inM, inB, selection, CURRENT, CURRENT_V1, CURRENT_V2,
  ↪  CURRENT_V3)
  variable skT, skM, skB: std_logic;
  variable out_en, muxOUT_var: std_logic;
  variable totSk: integer := 0;
  begin
    if (inT'event and inT='1') then
      skT := '1';
      totSk := totSk+1;
    end if;
    if (inM'event and inM='1') then
      skM := '1';
```

```
      totSk := totSk+1;
    end if;
    if (inB'event and inB='1') then
      skB := '1';
      totSk := totSk+1;
    end if;

    if (CURRENT_V1 /= 0.0 or CURRENT_V2 /= 0.0 or CURRENT_V3 /= 0.0 ) then --CURRENT /= 0.0 or
      if (selection = "01") then  --OR, V1 si attiva
        if (skT = '1') then
          muxOUT_var := '1';
          out_en := '1';
          totSk := totSk-1;
        else
          muxOUT_var := '0';
          out_en := '1';
        end if;
        case totSk is
          when 1 =>   V1OUT <= '1', '0' after 10 ps;
                V3OUT <= '0';
                V2OUTt <= '0';
                V2OUTb <= '0';
          when 2 =>   V1OUT <= '1', '0' after 10 ps, '1' after 20 ps, '0' after 30 ps;
                V3OUT <= '0';
                V2OUTt <= '0';
                V2OUTb <= '0';
          when others =>   V1OUT <= '0';
                  V3OUT <= '0';
                  V2OUTt <= '0';
                  V2OUTb <= '0';
        end case;
        totSk := 0;
        skT := '0';
        skM := '0';
        skB := '0';
      elsif (selection = "11") then  --SUM, V3 si attiva
        if (skB = '1') then
          muxOUT_var := '1';
          out_en := '1';
          totSk := totSk-1;
        else
          muxOUT_var := '0';
          out_en := '1';
        end if;
        case totSk is
          when 1 =>   V3OUT <= '1', '0' after 10 ps;
                V1OUT <= '0';
```

```
        V2OUTt <= '0';
        V2OUTb <= '0';
     when 2 =>   V3OUT <= '1', '0' after 10 ps, '1' after 20 ps, '0' after
     ↪  30 ps;
        V1OUT <= '0';
        V2OUTt <= '0';
        V2OUTb <= '0';
     when others =>   V3OUT <= '0';
         V1OUT <= '0';
         V2OUTt <= '0';
         V2OUTb <= '0';
    end case;
    totSk := 0;
    skT := '0';
    skM := '0';
    skB := '0';
   elsif (selection = "10") then  --AND, V2 si attiva
    if (skM = '1') then
     muxOUT_var := '1';
     out_en := '1';
     totSk := totSk-1;
    else
     muxOUT_var := '0';
     out_en := '1';
    end if;
    case totSk is
     when 1 =>   if (skT = '1') then
         V2OUTt <= '1', '0' after 10 ps;
         V2OUTb <= '0';
        else
         V2OUTb <= '1', '0' after 10 ps;
         V2OUTt <= '0';
        end if;
        V3OUT <= '0';
        V1OUT <= '0';

     when 2 =>   V2OUTt <= '1', '0' after 10 ps;
        V2OUTb <= '1', '0' after 10 ps;
        V3OUT <= '0';
        V1OUT <= '0';
     when others =>   V3OUT <= '0';
         V1OUT <= '0';
         V2OUTt <= '0';
         V2OUTb <= '0';
    end case;
    totSk := 0;
    skT := '0';
    skM := '0';
    skB := '0';
```

```
      else
        V3OUT <= '0';
        V1OUT <= '0';
        V2OUTt <= '0';
        V2OUTb <= '0';
        totSk := 0;
        skT := '0';
        skM := '0';
        skB := '0';
      end if;
    else
      V3OUT <= '0';
      V1OUT <= '0';
      V2OUTt <= '0';
      V2OUTb <= '0';
    end if;

    if(CURRENT /= 0.0 and out_en='1') then
      out_en := '0';
      if (muxOUT_var='1') then
        muxOUT_var := '0';
        muxOUT <= '1', '0' after 10 ps;
      else
        muxOUT <= '0';
      end if;
    else
      muxOUT <= '0';
    end if;

  end process;
end architecture Behavioural;
```

C.1.9.2. Result/Stored element multiplexer

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity mux2 is
port(  inR, inL: in std_logic;    --Right=result, Left=stored value
```

```
    CURRENT: in real;
    CURRENT_Vst, CURRENT_Vres: in real;
    selection: in std_logic;  --'1'=result, '0'=stored value
    muxOUT, VstOUT, VresOUT: out std_logic);
end entity mux2;

architecture Behavioural of mux2 is
begin
  process (inR, inL, selection, CURRENT, CURRENT_Vst, CURRENT_Vres)
  variable skR, skL: std_logic;
  variable out_en, muxOUT_var: std_logic;
  variable totSk: integer := 0;
  begin
    if (inR'event and inR='1') then
      skR := '1';
      totSk := totSk+1;
    end if;
    if (inL'event and inL='1') then
      skL := '1';
      totSk := totSk+1;
    end if;

    if (CURRENT_Vst /= 0.0 or CURRENT_Vres /= 0.0) then  --CURRENT /= 0.0
      if (selection = '1') then  --result is desired
        if (skR = '1') then
          muxOUT_var := '1';
          out_en := '1';
          totSk := totSk-1;
        else
          muxOUT_var := '0';
          out_en := '1';
        end if;
        case totSk is
          when 1 =>   VstOUT <= '1', '0' after 10 ps;
                VresOUT <= '0';
          when others =>   VstOUT <= '0';
                VresOUT <= '0';
        end case;
        totSk := 0;
        skR := '0';
        skL := '0';
      else  --stored value is desired: selection='0'
        if (skL = '1') then
          muxOUT_var := '1';
          out_en := '1';
          totSk := totSk-1;
        else
          muxOUT_var := '0';
          out_en := '1';
```

```
        end if;
        case totSk is
          when 1 =>   VresOUT <= '1', '0' after 10 ps;
                VstOUT <= '0';
          when others =>   VresOUT <= '0';
                  VstOUT <= '0';
        end case;
        totSk := 0;
        skR := '0';
        skL := '0';
      end if;
    else
      VresOUT <= '0';
      VstOUT <= '0';
    end if;

    if(CURRENT /= 0.0 and out_en='1') then
      out_en := '0';
      if (muxOUT_var='1') then
        muxOUT_var := '0';
        muxOUT <= '1', '0' after 10 ps;
      else
        muxOUT <= '0';
      end if;
    else
      muxOUT <= '0';
    end if;

  end process;
end architecture Behavioural;
```

C.1.10. SRlatch

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SRlatch is
port(  SET:   in std_logic;
    RST:   in std_logic;
    Q:    out std_logic
);
```

```
end entity SRlatch;

architecture Behaviour of SRlatch is

begin
  latch: process (SET, RST)
  begin
    if (RST='1') then
      Q <= '0';
    elsif(SET'event and SET='1') then
      Q <= '1';
    end if;
  end process latch;

end architecture Behaviour;
```

C.2. Cell00

C.2.1. Datapath

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity CELL_00 is
port(  IN_CELL, RESET_n: in std_logic;
    CURRENTclk: in real;

    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3,
    ↪  CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4,
    ↪  CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop: in
    ↪  std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
    CTRL_mux1: in std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
    SR_MTJR1_out, SR_MTJR2_out: out std_logic;
    CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

```

```
    CELL_OUT_MTJ_CONV12_out, CELL_OUT_MTJ_CONV11_out, CELL_OUT_DEV2_out,
     ↪  CELL_OUT_CROSS4_Aout, CELL_OUT_TANKT_out: out std_logic
);
end entity CELL_00;

architecture Structure of CELL_00 is
  component SKYRMIONNOTCH is
  port(  INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  component SKYRMIONNOTCHseq is
  generic (N: integer := 5);
  port(   INPUT: in std_logic;
      CURRENT: in real;
      OUTPUT: out std_logic);
  end component SKYRMIONNOTCHseq;

  component SKYRMIONJOIN is
  port( A : in std_logic;
      B : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONJOIN;

  component SKYRMIONH is
  port (  INPUTA : in std_logic;
      INPUTB : in std_logic;
      CURRENT : in real;
      OUTPUTAND : out std_logic;
      OUTPUTOR : out std_logic);
  end component SKYRMIONH;

  component SKYRMIONFULLADDER is
  port(   A     : in std_logic;
      B     : in std_logic;
      CIN1  : in std_logic;
      CIN2  : in std_logic;
      ONE1  : in std_logic;
      ONE2  : in std_logic;
      CURRENT   : in real;
      CTRL1  : out std_logic;
      SUM    : out std_logic;
      COUT1  : out std_logic;
      COUT2  : out std_logic;
      CTRL2  : out std_logic);
  end component SKYRMIONFULLADDER;

```

```
component voltage_genH is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genH;

component voltage_genL is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genL;

component vclock_gen is
port(  CTRL: in std_logic;
    CURRENTclk: in real;
    CURRENT:  out real);
end component vclock_gen;

component skyrmionDUPLICATE is
port(  IN_SK:      in std_logic;
    CURRENT:    in real;
    OUT_SK_TOP:    out std_logic;
    OUT_SK_BOTTOM:  out std_logic);
end component skyrmionDUPLICATE;

component skyrmionMERGE is
port(  IN_SK_TOP:     in std_logic;
    IN_SK_BOTTOM:  in std_logic;
    CURRENT:    in real;
    OUT_SK:      out std_logic);
end component skyrmionMERGE;

component SKYRMIONCROSS_Magn is
port(   A:       in std_logic;
    B:       in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end component SKYRMIONCROSS_Magn;

component SKYRMIONCROSS_noMagn is
port(   A:       in std_logic;
    B:       in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end component SKYRMIONCROSS_noMagn;

component skyrmionDEVIATION is
```

```
port(  IN_SK:     in std_logic;
    CURRENT:  in real;
    CURRENTDEV:  in real;
    OUT_SK:    out std_logic;
    OUT_SK_DEV:  out std_logic);
end component skyrmionDEVIATION;

component SRlatch is
port(  SET:   in std_logic;
    RST:   in std_logic;
    Q:    buffer std_logic);
end component SRlatch;

component MTJ_R is
port(  IN_SK:     in std_logic;
    CURRENT:  in real;
    OUT_SIGN:  out std_logic);
end component MTJ_R;

component MTJ_W is
port(  CTRL: in std_logic;
    OUT_SK:  out std_logic);
end component MTJ_W;

component MTJ_CONV is
port(  IN_SK: in std_logic;
    CURRENT: in real;
    OUT_SK:  out std_logic);
end component MTJ_CONV;

component tank_bottom is
port(  IN1, IN2, IN3, IN4, IN5, IN6, IN7: in std_logic;  --from left to right
    CURRENT: in real;
    tankOUT1, tankOUT2, tankOUT3: out std_logic);    --priority order: from
    ↪ right to left
end component tank_bottom;

component tank_top is
port(  IN1, IN2, IN3: in std_logic; --from right to left
    CURRENT: in real;
    tankOUT: out std_logic);
end component tank_top;

component mux1 is
port(  inT, inM, inB: in std_logic;
    CURRENT: in real;
    CURRENT_V1, CURRENT_V2, CURRENT_V3: in real;
    selection: in std_logic_vector(1 downto 0);
    muxOUT, V1OUT, V3OUT, V2OUTt, V2OUTb: out std_logic);
```

```
  end component mux1;

  component mux2 is
  port(  inR, inL: in std_logic;    --Right=result, Left=stored value
      CURRENT: in real;
      CURRENT_Vst, CURRENT_Vres: in real;
      selection: in std_logic;  --'1'=result, '0'=stored value
      muxOUT, VstOUT, VresOUT: out std_logic);
  end component mux2;



  signal CURRENT_Vstart, CURRENT_Vmove1, CURRENT_Vop, CURRENT_Vmove2, CURRENT_Vmove2C, CURRENT_Vmove3, CURRENT_MUX1_V1, CURRENT_MUX1_V2, CURRENT_MUX1_V3, CURRENT_Vmovecout, CURRENT_Vtankbot, CURRENT_Vnoext, CURRENT_Vmoveres, CURRENT_Vtanktop, CURRENT_Vmove4, CURRENT_Vnewdata, CURRENT_Vdetect, CURRENT_Vrout, CURRENT_Vsout: real;

  signal MEM_out: std_logic;
  signal x2_1_outtop, x2_1_outbottom, x2_2_outtop, x2_2_outbottom, x2_3_outtop, x2_3_outbottom, x2_4_outtop, x2_4_outbottom, x2_5_outright, x2_5_outleft, x1_1_out: std_logic;
  signal CROSSM1_outA, CROSSM1_outB, CROSSM2_outA, CROSSM2_outB, CROSSM3_outA, CROSSM3_outB: std_logic;
  signal CROSS11_Aout, CROSS11_Bout, CROSS12_Aout, CROSS12_Bout, CROSS21_Aout, CROSS21_Bout, CROSS22_Aout, CROSS22_Bout, CROSS4_Aout, CROSS4_Bout: std_logic;
  signal SYNC_LOG_TOP_out, SYNC_LOG_BOT_out, AND_out, OR_out, SKEW_OR_out, SKEW_AND_out: std_logic;
  signal FA_CTRL1_out, FA_CTRL2_out, FA_COUT1_out, FA_COUT2_out, FA_SUM_out: std_logic;
  signal RST_SR_MTJR1, RST_SR_MTJR2: std_logic;
  signal MTJ_R1_out, MTJ_R2_out: std_logic;
  signal MTJ_W1_out, MTJ_W2_out, MTJ_W3_out: std_logic;
  signal MTJ_CONV1_out, MTJ_CONV2_out, MTJ_CONV3_out, MTJ_CONV4_out, MTJ_CONV5_out, MTJ_CONV6_out, MTJ_CONV7_out, MTJ_CONV8_out, MTJ_CONV9_out, MTJ_CONV10_out, MTJ_CONV11_out, MTJ_CONV12_out: std_logic;
  signal MUX1_CTRL_V1, MUX1_CTRL_V2, MUX1_CTRL_V3: std_logic;
  signal MUX1_out, MUX1_V1out, MUX1_V3out, MUX1_V2Tout, MUX1_V2Bout: std_logic;
  signal CTRL_mux2, MUX2_Vstout, MUX2_Vresout, MUX2_out: std_logic;
  signal JOIN1_out, JOIN2_out, JOIN3_out: std_logic;
  signal DEV1_out, DEV1_devout, DEV2_out, DEV2_devout: std_logic;
  signal TANKB_out1, TANKB_out2, TANKB_out3: std_logic;
  signal TANKT_out: std_logic;


begin
  Vstart:  voltage_genH port map (CTRL => CTRL_Vstart, CURRENT => CURRENT_Vstart);
```

```
MEM:   SKYRMIONNOTCH port map (INPUT => IN_CELL, CURRENT => CURRENT_Vstart, OUTPUT => MEM_out);
Vmove1:  voltage_genL port map (CTRL => CTRL_Vmove1, CURRENT => CURRENT_Vmove1);
x2_1:  skyrmionDUPLICATE port map (IN_SK => MEM_out, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_1_outtop, OUT_SK_BOTTOM => x2_1_outbottom);
x2_2:  skyrmionDUPLICATE port map (IN_SK => x2_1_outtop, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_2_outtop, OUT_SK_BOTTOM => x2_2_outbottom);

CROSSM1:       SKYRMIONCROSS_Magn port map (A => x2_2_outbottom, B => x2_3_outtop, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM1_outA, Bout => CROSSM1_outB);
Vop:         vclock_gen port map (CTRL => CTRL_Vop, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vop);
SYNCNET_LOG_TOP:  SKYRMIONNOTCH port map (INPUT => x2_2_outtop, CURRENT => CURRENT_Vop, OUTPUT => SYNC_LOG_TOP_out);
SYNCNET_LOG_BOT:  SKYRMIONNOTCH port map (INPUT => CROSSM1_outB, CURRENT => CURRENT_Vop, OUTPUT => SYNC_LOG_BOT_out);
LOGIC:         SKYRMIONH port map (INPUTA => SYNC_LOG_TOP_out, INPUTB => SYNC_LOG_BOT_out, CURRENT => CURRENT_Vop, OUTPUTAND => AND_OUT, OUTPUTOR => OR_OUT);
SKEW_OR:       SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => OR_OUT, CURRENT => CURRENT_Vop, OUTPUT => SKEW_OR_out);
SKEW_AND:       SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => AND_OUT, CURRENT => CURRENT_Vop, OUTPUT => SKEW_AND_out);
FA:            SKYRMIONFULLADDER port map (A => CROSSM1_outA, B => x2_3_outbottom, CIN1 => '0', CIN2 => '0', ONE1 => CROSSM2_outB, ONE2 => CROSSM3_outB, CURRENT => CURRENT_Vop, CTRL1 => FA_CTRL1_out, SUM => FA_SUM_out, COUT1 => FA_COUT1_out, COUT2 => FA_COUT2_out, CTRL2 => FA_CTRL2_out);

MTJ_CONV_1:  MTJ_CONV port map (IN_SK => SKEW_OR_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV1_out);
MTJ_CONV_2:  MTJ_CONV port map (IN_SK => SKEW_AND_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV2_out);
MTJ_CONV_3:  MTJ_CONV port map (IN_SK => FA_SUM_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV3_out);
MTJ_CONV_4:  MTJ_CONV port map (IN_SK => FA_COUT1_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV4_out);
MTJ_CONV_5:  MTJ_CONV port map (IN_SK => FA_COUT2_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV5_out);
Vmove2:    voltage_genL port map (CTRL => CTRL_Vmove2, CURRENT => CURRENT_Vmove2);
Vmove2C:  voltage_genL port map (CTRL => CTRL_Vmove2C, CURRENT => CURRENT_Vmove2C);

MUX1_CTRL_V1 <= CTRL_mux1(0) and (not CTRL_mux1(1));
```

```
MUX1_CTRL_V2 <= CTRL_mux1(1) and (not CTRL_mux1(0));
MUX1_CTRL_V3 <= CTRL_mux1(1) and CTRL_mux1(0);
Vmux1_1:  voltage_genL port map (CTRL => MUX1_CTRL_V1, CURRENT => CURRENT_MUX1_V1);
Vmux1_2:  voltage_genL port map (CTRL => MUX1_CTRL_V2, CURRENT => CURRENT_MUX1_V2);
Vmux1_3:  voltage_genL port map (CTRL => MUX1_CTRL_V3, CURRENT => CURRENT_MUX1_V3);

MUX1_COM:     mux1 port map (inT => MTJ_CONV1_out, inM => MTJ_CONV2_out, inB => MTJ_CONV3_out, CURRENT => CURRENT_Vmove2, CURRENT_V1 => CURRENT_MUX1_V1, CURRENT_V2 => CURRENT_MUX1_V2, CURRENT_V3 => CURRENT_MUX1_V3, selection => CTRL_mux1, muxOUT => MUX1_out, V1OUT => MUX1_V1out, V3OUT => MUX1_V3out, V2OUTt => MUX1_V2Tout, V2OUTb => MUX1_V2Bout);
Vmove3:    voltage_genL port map (CTRL => CTRL_Vmove3, CURRENT => CURRENT_Vmove3);
x2_4:    skyrmionDUPLICATE port map (IN_SK => MUX1_out, CURRENT => CURRENT_Vmove3, OUT_SK_TOP => x2_4_outtop, OUT_SK_BOTTOM => x2_4_outbottom);
MTJ_CONV_7:  MTJ_CONV port map (IN_SK => x2_4_outbottom, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV7_out);

CROSS11:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV4_out, B => MUX1_V1out, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout => CROSS11_Aout, Bout => CROSS11_Bout);
CROSS12:  SKYRMIONCROSS_noMagn port map (A => CROSS11_Aout, B => MUX1_V2Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS12_Aout, Bout => CROSS12_Bout);
CROSS21:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV5_out, B => CROSS11_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout => CROSS21_Aout, Bout => CROSS21_Bout);
CROSS22:  SKYRMIONCROSS_noMagn port map (A => CROSS21_Aout, B => CROSS12_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS22_Aout, Bout => CROSS22_Bout);

Vmovecout:  voltage_genL port map (CTRL => CTRL_Vmovecout, CURRENT => CURRENT_Vmovecout);
x1_1:    skyrmionMERGE port map (IN_SK_TOP => CROSS12_Aout, IN_SK_BOTTOM => CROSS22_Aout, CURRENT => CURRENT_Vmovecout, OUT_SK => x1_1_out);
MTJ_CONV_6:  MTJ_CONV port map (IN_SK => x1_1_out, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV6_out);
CROSS4:   SKYRMIONCROSS_noMagn port map (A => MTJ_CONV6_out, B => MTJ_CONV7_out, CURRENTA => CURRENT_Vmovecout, CURRENTB => CURRENT_Vmove3, Aout => CROSS4_Aout, Bout => CROSS4_Bout);

Vtankbot:  vclock_gen port map (CTRL => CTRL_Vtankbot, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vtankbot);

```

```
TANK_BOT:  tank_bottom port map (IN1 => FA_CTRL2_out, IN2 => FA_CTRL1_out, IN3 => CROSS21_Bout, IN4 => CROSS22_Bout, IN5 => DEV1_devout, IN6 => MUX2_Vresout, IN7 => MUX2_Vstout, CURRENT => CURRENT_Vtankbot, tankOUT1 => TANKB_out1, tankOUT2 => TANKB_out2, tankOUT3 => TANKB_out3);

MTJ_W_1:  MTJ_W port map (CTRL => CTRL_MTJ_W1, OUT_SK => MTJ_W1_out);
MTJ_W_2:  MTJ_W port map (CTRL => CTRL_MTJ_W2, OUT_SK => MTJ_W2_out);
JOIN2:    SKYRMIONJOIN port map (A => MTJ_W1_out, B => TANKB_out2, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN2_out);
JOIN3:    SKYRMIONJOIN port map (A => MTJ_W2_out, B => TANKB_out1, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN3_out);
CROSSM2:   SKYRMIONCROSS_Magn port map (A => TANKB_out1, B => JOIN2_out, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM2_outA, Bout => CROSSM2_outB);
CROSSM3:   SKYRMIONCROSS_Magn port map (A => CROSSM2_outA, B => JOIN3_out, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM3_outA, Bout => CROSSM3_outB);

MTJ_R_1:  MTJ_R port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vmove1, OUT_SIGN => MTJ_R1_out);
RST_SR_MTJR1 <= ACK_DETECT_MTJ_R1 or (not RESET_n);
SR_MTJR1:  SRlatch port map (SET => MTJ_R1_out, RST => RST_SR_MTJR1, Q => SR_MTJR1_out);
MTJ_CONV_8:  MTJ_CONV port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vmove1, OUT_SK => MTJ_CONV8_out);
Vnoext:    voltage_genL port map (CTRL => CTRL_Vnoext, CURRENT => CURRENT_Vnoext);
DEV1:    skyrmionDEVIATION port map (IN_SK => MTJ_CONV8_out, CURRENT => CURRENT_Vmove1, CURRENTDEV => CURRENT_Vnoext, OUT_SK => DEV1_out, OUT_SK_DEV => DEV1_devout);
MTJ_W_3:  MTJ_W port map (CTRL => CTRL_MTJ_W3, OUT_SK => MTJ_W3_out);
JOIN1:    SKYRMIONJOIN port map (A => MTJ_W3_out, B => DEV1_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN1_out);
x2_3:  skyrmionDUPLICATE port map (IN_SK => JOIN1_out, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_3_outtop, OUT_SK_BOTTOM => x2_3_outbottom);

MTJ_CONV_10:  MTJ_CONV port map (IN_SK => x2_1_outbottom, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV10_out);
CTRL_mux2 <= CTRL_Vrout;
Vmux2_Vsout:  voltage_genL port map (CTRL => CTRL_Vsout, CURRENT => CURRENT_Vsout);
Vmux2_Vrout:  voltage_genL port map (CTRL => CTRL_Vrout, CURRENT => CURRENT_Vrout);
MUX2_COM:     mux2 port map (inR => CROSS4_Bout, inL => MTJ_CONV10_out, CURRENT => CURRENT_Vmove3, CURRENT_Vst => CURRENT_Vsout, CURRENT_Vres => CURRENT_Vrout, selection => CTRL_mux2, muxOUT => MUX2_out, VstOUT => MUX2_Vstout, VresOUT => MUX2_Vresout);

```

```
  Vmoveres:    voltage_genL port map (CTRL => CTRL_Vmoveres, CURRENT => CURRENT_Vmoveres);
  x2_5:    skyrmionDUPLICATE port map (IN_SK => MUX2_out, CURRENT => CURRENT_Vmoveres, OUT_SK_TOP => x2_5_outright, OUT_SK_BOTTOM => x2_5_outleft);
  MTJ_CONV_11:  MTJ_CONV port map (IN_SK => x2_5_outright, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV11_out);
  MTJ_CONV_12:  MTJ_CONV port map (IN_SK => x2_5_outleft, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV12_out);

  Vtanktop:  vclock_gen port map (CTRL => CTRL_Vtanktop, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vtanktop);
  TANK_TOP_C:  tank_top port map (IN1 => MUX1_V2Tout, IN2 => MUX1_V3out, IN3 => DEV2_devout, CURRENT => CURRENT_Vtanktop, tankOUT => TANKT_out);
  MTJ_CONV_9:  MTJ_CONV port map (IN_SK => x2_4_outtop, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV9_out);
  Vmove4:    voltage_genL port map (CTRL => CTRL_Vmove4, CURRENT => CURRENT_Vmove4);
  Vnewdata:  voltage_genL port map (CTRL => CTRL_Vnewdata, CURRENT => CURRENT_Vnewdata);
  DEV2:    skyrmionDEVIATION port map (IN_SK => MTJ_CONV9_out, CURRENT => CURRENT_Vmove4, CURRENTDEV => CURRENT_Vnewdata, OUT_SK => DEV2_out, OUT_SK_DEV => DEV2_devout);
  Vdetect:  voltage_genL port map (CTRL => CTRL_Vdetect, CURRENT => CURRENT_Vdetect);
  MTJ_R_2:  MTJ_R port map (IN_SK => TANKT_out, CURRENT => CURRENT_Vdetect, OUT_SIGN => MTJ_R2_out);
  RST_SR_MTJR2 <= ACK_DETECT_MTJ_R2 or (not RESET_n);
  SR_MTJR2:  SRlatch port map (SET => MTJ_R2_out, RST => RST_SR_MTJR2, Q => SR_MTJR2_out);

  CELL_OUT_MTJ_CONV12_out   <= MTJ_CONV12_out;
  CELL_OUT_MTJ_CONV11_out   <= MTJ_CONV11_out;
  CELL_OUT_DEV2_out       <= DEV2_out;
  CELL_OUT_CROSS4_Aout     <= CROSS4_Aout;
  CELL_OUT_TANKT_out      <= TANKT_out;

  CELL_OUT_CURRENT_Vmoveres   <= CURRENT_Vmoveres;
  CELL_OUT_CURRENT_Vmovecout  <= CURRENT_Vmovecout;


end Structure;
```

C.2.2. FSM

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity FSM_CELL_00 is
port(  CURRENTclk: in real;
    RESET_n: in std_logic;

    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop: out std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
    CTRL_mux1: out std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
    SR_MTJR2_out, SR_MTJR1_out: in std_logic;

    START, ACK_RES_AVAILABLE: in std_logic;
    DES_EXTIN_00, DES_OUT_00, DES_STORE_00, DES_NEWDATA_00: in std_logic;
    DES_RES_00: in std_logic_vector(1 downto 0);
    RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE: out std_logic;
    READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic
);
end entity FSM_CELL_00;

architecture Behaviour of FSM_CELL_00 is

  type state_type is (
    reset, S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10,
    S4_1, S4_2, S4_3, S4_4, S4_5, S4_6,
    S11, S12, S13, S14, S15, S16, S17, S18, S19, S20,
    S21, S22, S23, S24, S24_1, S25, S26, S27, S28, S29, S30,
    S31, S32, S33, S34, S35, S36
  );

  signal pstate, nstate: state_type;

begin

  state_register: process (CURRENTclk)
  begin
```

```
    if (CURRENTclk'event and CURRENTclk=CURRENT_HIGH) then
      if (RESET_n = '0') then
        pstate <= reset;
      else
        pstate <= nstate;
      end if;
    end if;
  end process state_register;

  state_transition: process (pstate, CURRENTclk)
  begin
    case pstate is
      when reset  => nstate <= S0;
      when S0    => if (START='0') then nstate <= S0; else if (FIRST_RUN='1') then nstate <= S1; else nstate <= S29; end if; end if;
      when S1    => if(DES_EXTIN_00='1') then nstate <= S2; else nstate <= S3; end if;
      when S2    => nstate <= S4;
      when S3    => nstate <= S4;
      when S4    => nstate <= S4_1;
      when S4_1   => nstate <= S4_2;
      when S4_2   => nstate <= S4_3;
      when S4_3   => nstate <= S4_4;
      when S4_4   => nstate <= S4_5;
      when S4_5   => nstate <= S4_6;
      when S4_6   => nstate <= S5;
      when S5    => nstate <= S6;
      when S6    => if(DES_RES_00="10") then nstate <= S7; elsif(DES_RES_00="01") then nstate <= S8; else nstate <= S9; end if;
      when S7    => nstate <= S10;
      when S8    => nstate <= S10;
      when S9    => nstate <= S10;
      when S10    => nstate <= S11;
      when S11    => if(DES_OUT_00='1') then nstate <= S12; else nstate <= S13; end if;
      when S12    => nstate <= S14;
      when S13    => nstate <= S14;
      when S14    => nstate <= S15;
      when S15    => if(DES_STORE_00='1') then nstate <= S16; else nstate <= S17; end if;
      when S16    => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S16; else nstate <= S18; end if;
      when S17   => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S17; else if(DES_NEWDATA_00='1') then nstate <= S19; else nstate <= S20; end if; end if;
      when S18    => nstate <= S22;
      when S19    => nstate <= S21;
      when S20    => nstate <= S24;
      when S21    => nstate <= S23;
```

```
      when S22   => if(ACK_RES_AVAILABLE='0') then nstate <= S22; else nstate <= S26; end if;
      when S23  => if(ACK_RES_AVAILABLE='0') then nstate <= S23; else nstate <= S25; end if;
      when S24  => if(ACK_RES_AVAILABLE='0') then nstate <= S24; else nstate <= S28; end if;
      when S25   => if(SR_MTJR2_out='0') then nstate <= S27; else nstate <= S28; end if;
      when S26   => if (START='0') then nstate <= S26; else nstate <= S29; end if;
      when S27   => if (START='0') then nstate <= S27; else nstate <= S29; end if;
      when S28   => if (START='0') then nstate <= S28; else nstate <= S29; end if;
      when S29   => nstate <= S30;
      when S30   => if(DES_EXTIN_00='0') then nstate <= S31; else nstate <= S32; end if;
      when S31   => if(SR_MTJR1_out='0') then nstate <= S35; else nstate <= S33; end if;
      when S32   => if(SR_MTJR1_out='1') then nstate <= S36; else nstate <= S34; end if;
      when S33   => nstate <= S35;
      when S34   => nstate <= S36;
      when S35   => nstate <= S4;
      when S36   => nstate <= S4;
      when others => nstate <= S0;
    end case;
  end process state_transition;

  output: process (pstate)
  begin
    CTRL_Vstart <= '0';
    CTRL_Vmove1 <= '0';
    CTRL_Vop <= '0';
    CTRL_Vmove2 <= '0';
    CTRL_Vmove2C <= '0';
    CTRL_Vmove3 <= '0';
    CTRL_Vmovecout <= '0';
    CTRL_Vtankbot <= '0';
    CTRL_Vnoext <= '0';
    CTRL_Vmoveres <= '0';
    CTRL_Vmove4 <= '0';
    CTRL_Vnewdata <= '0';
    CTRL_Vdetect <= '0';
    CTRL_Vrout <= '0';
    CTRL_Vsout <= '0';
    CTRL_Vtanktop <= '0';
    CTRL_MTJ_W1 <= '0';
    CTRL_MTJ_W2 <= '0';
```

```
    CTRL_MTJ_W3 <= '0';
    CTRL_mux1 <= "00";
    ACK_DETECT_MTJ_R1 <= '0';
    ACK_DETECT_MTJ_R2 <= '0';
    RES_AVAILABLE <= '0';
    REQUEST_NEW_START <= '0';
    REQUEST_NEW_START_W <= '0';
    REQUEST_NEW_STORE <= '0';

    case pstate is
      when S0    =>  CTRL_Vstart <= '0';
              CTRL_Vmove1 <= '0';
              CTRL_Vop <= '0';
              CTRL_Vmove2 <= '0';
              CTRL_Vmove2C <= '0';
              CTRL_Vmove3 <= '0';
              CTRL_Vmovecout <= '0';
              CTRL_Vtankbot <= '0';
              CTRL_Vnoext <= '0';
              CTRL_Vmoveres <= '0';
              CTRL_Vmove4 <= '0';
              CTRL_Vnewdata <= '0';
              CTRL_Vdetect <= '0';
              CTRL_Vrout <= '0';
              CTRL_Vsout <= '0';
              CTRL_Vtanktop <= '0';
              CTRL_MTJ_W1 <= '0';
              CTRL_MTJ_W2 <= '0';
              CTRL_MTJ_W3 <= '0';
              CTRL_mux1 <= "00";
              ACK_DETECT_MTJ_R1 <= '0';
              ACK_DETECT_MTJ_R2 <= '0';
              RES_AVAILABLE <= '0';
              REQUEST_NEW_START <= '0';
              REQUEST_NEW_START_W <= '0';
              REQUEST_NEW_STORE <= '0';
      when S1    =>  CTRL_Vstart <= '1';
      when S2    =>  CTRL_Vmove1 <= '1';
              CTRL_MTJ_W1 <= '1';
              CTRL_MTJ_W2 <= '1';
              CTRL_MTJ_W3 <= '1';
      when S3    =>  CTRL_Vmove1 <= '1';
              CTRL_MTJ_W1 <= '1';
              CTRL_MTJ_W2 <= '1';
      when S4    =>  CTRL_Vop <= '1';
      when S4_1  =>  CTRL_Vop <= '1';
      when S4_2  =>  CTRL_Vop <= '1';
      when S4_3  =>  CTRL_Vop <= '1';
      when S4_4  =>  CTRL_Vop <= '1';
```

```
        when S4_5  =>  CTRL_Vop <= '1';
        when S4_6  =>  CTRL_Vop <= '1';
        when S5    =>  CTRL_Vmove2 <= '1';
                CTRL_Vmove2C <= '1';
        when S6    =>  CTRL_Vmove2C <= '1';
        when S7    =>  CTRL_mux1 <= "10";
        when S8    =>  CTRL_mux1 <= "01";
        when S9    =>  CTRL_mux1 <= "11";
        when S10  =>  CTRL_Vmove2 <= '1';
        when S11  =>  CTRL_Vmove3 <= '1';
        when S12  =>  CTRL_Vrout <= '1';
        when S13  =>  CTRL_Vsout <= '1';
        when S14  =>  CTRL_Vmove3 <= '1';
        when S15  =>  CTRL_Vmove4 <= '1', '0' after CLOCK_PERIOD/2;
        when S16  =>  CTRL_Vmove4 <= '1';
        when S17  =>  CTRL_Vnewdata <= '1';
        when S18  =>  CTRL_Vmoveres <= '1';
                CTRL_Vmovecout <= '1';
        when S19  =>  CTRL_Vtanktop <= '1';
                CTRL_Vmoveres <= '1';
                CTRL_Vmovecout <= '1';
        when S20  =>  CTRL_Vmoveres <= '1';
                CTRL_Vmovecout <= '1';
        when S21  =>  CTRL_Vtanktop <= '1';
        when S22  =>  RES_AVAILABLE <= '1';
        when S23  =>  RES_AVAILABLE <= '1';
        when S24  =>  RES_AVAILABLE <= '1';
        when S25  =>  CTRL_Vdetect <= '1';
        when S26  =>  REQUEST_NEW_STORE <= '1';
        when S27  =>  REQUEST_NEW_START_W <= '1';
                ACK_DETECT_MTJ_R2 <= '1';
        when S28  =>  REQUEST_NEW_START <= '1';
                ACK_DETECT_MTJ_R2 <= '1';
        when S29  =>  CTRL_Vstart <= '1';
                CTRL_Vtankbot <= '1';
        when S30  =>  CTRL_Vtankbot <= '1';
        when S31  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
        when S32  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
        when S33  =>  CTRL_Vnoext <= '1';
                ACK_DETECT_MTJ_R1 <= '1';
        when S34  =>  CTRL_MTJ_W3 <= '1';
        when S35  =>  CTRL_Vmove1 <= '1';
        when S36  =>  CTRL_Vmove1 <= '1';
                ACK_DETECT_MTJ_R1 <= '1';

        when others =>  CTRL_Vstart <= '0';
                CTRL_Vmove1 <= '0';
                CTRL_Vop <= '0';
                CTRL_Vmove2 <= '0';
```

```
                CTRL_Vmove2C <= '0';
                CTRL_Vmove3 <= '0';
                CTRL_Vmovecout <= '0';
                CTRL_Vtankbot <= '0';
                CTRL_Vnoext <= '0';
                CTRL_Vmoveres <= '0';
                CTRL_Vmove4 <= '0';
                CTRL_Vnewdata <= '0';
                CTRL_Vdetect <= '0';
                CTRL_Vrout <= '0';
                CTRL_Vsout <= '0';
                CTRL_Vtanktop <= '0';
                CTRL_MTJ_W1 <= '0';
                CTRL_MTJ_W2 <= '0';
                CTRL_MTJ_W3 <= '0';
                CTRL_mux1 <= "00";
                ACK_DETECT_MTJ_R1 <= '0';
                ACK_DETECT_MTJ_R2 <= '0';
                RES_AVAILABLE <= '0';
                REQUEST_NEW_START <= '0';
                REQUEST_NEW_START_W <= '0';
                REQUEST_NEW_STORE <= '0';
    end case;
  end process output;
end Behaviour;
```

C.3. Cell10

C.3.1. Datapath

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity CELL_X0 is
port(  IN_CELL, RESET_n: in std_logic;
    CURRENTclk: in real;

```

```
    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop: in std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
    CTRL_mux1: in std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
    SR_MTJR1_out, SR_MTJR2_out: out std_logic;
    CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

    CELL_OUT_MTJ_CONV12_out, CELL_OUT_CROSS5_Aout, CELL_OUT_DEV2_out, CELL_OUT_CROSS6_Aout, CELL_OUT_CROSS6_Bout, CELL_OUT_TANKT_out: out std_logic;

    CTRL_Vrestop: in std_logic;
    TOP_RESULT:  in std_logic;
    TOP_RESULT_CURRENT: in real
);
end entity CELL_X0;

architecture Structure of CELL_X0 is
  component SKYRMIONNOTCH is
  port(  INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  component SKYRMIONNOTCHseq is
  generic (N: integer := 5);
  port(   INPUT: in std_logic;
      CURRENT: in real;
      OUTPUT: out std_logic);
  end component SKYRMIONNOTCHseq;

  component SKYRMIONJOIN is
  port( A : in std_logic;
      B : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONJOIN;

  component SKYRMIONH is
  port (  INPUTA : in std_logic;
      INPUTB : in std_logic;
      CURRENT : in real;
      OUTPUTAND : out std_logic;
      OUTPUTOR : out std_logic);
  end component SKYRMIONH;

```

```
component SKYRMIONFULLADDER is
port(   A     : in std_logic;
    B     : in std_logic;
    CIN1  : in std_logic;
    CIN2  : in std_logic;
    ONE1  : in std_logic;
    ONE2  : in std_logic;
    CURRENT  : in real;
    CTRL1  : out std_logic;
    SUM    : out std_logic;
    COUT1  : out std_logic;
    COUT2  : out std_logic;
    CTRL2  : out std_logic);
end component SKYRMIONFULLADDER;

component voltage_genH is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genH;

component voltage_genL is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genL;

component vclock_gen is
port(  CTRL: in std_logic;
    CURRENTclk: in real;
    CURRENT:  out real);
end component vclock_gen;

component skyrmionDUPLICATE is
port(  IN_SK:       in std_logic;
    CURRENT:    in real;
    OUT_SK_TOP:    out std_logic;
    OUT_SK_BOTTOM:  out std_logic);
end component skyrmionDUPLICATE;

component skyrmionMERGE is
port(  IN_SK_TOP:     in std_logic;
    IN_SK_BOTTOM:  in std_logic;
    CURRENT:    in real;
    OUT_SK:      out std_logic);
end component skyrmionMERGE;

component SKYRMIONCROSS_Magn is
port(   A:       in std_logic;
    B:       in std_logic;
    CURRENTA:   in real;
```

```
        CURRENTB:  in real;
        Aout:     out std_logic;
        Bout:     out std_logic);
    end component SKYRMIONCROSS_Magn;

    component SKYRMIONCROSS_noMagn is
    port(   A:        in std_logic;
        B:        in std_logic;
        CURRENTA:   in real;
        CURRENTB:  in real;
        Aout:     out std_logic;
        Bout:     out std_logic);
    end component SKYRMIONCROSS_noMagn;

    component skyrmionDEVIATION is
    port(  IN_SK:      in std_logic;
        CURRENT:  in real;
        CURRENTDEV:  in real;
        OUT_SK:    out std_logic;
        OUT_SK_DEV:  out std_logic);
    end component skyrmionDEVIATION;

    component SRlatch is
    port(  SET:   in std_logic;
        RST:   in std_logic;
        Q:    buffer std_logic);
    end component SRlatch;

    component MTJ_R is
    port(  IN_SK:      in std_logic;
        CURRENT:  in real;
        OUT_SIGN:  out std_logic);
    end component MTJ_R;

    component MTJ_W is
    port(  CTRL: in std_logic;
        OUT_SK:  out std_logic);
    end component MTJ_W;

    component MTJ_CONV is
    port(  IN_SK: in std_logic;
        CURRENT: in real;
        OUT_SK:  out std_logic);
    end component MTJ_CONV;

    component tank_bottom is
    port(  IN1, IN2, IN3, IN4, IN5, IN6, IN7: in std_logic;  --from left to right
        CURRENT: in real;
        tankOUT1, tankOUT2, tankOUT3: out std_logic);    --priority order: from
     ↪  right to left
```

```
end component tank_bottom;

component tank_top is
port(  IN1, IN2, IN3: in std_logic; --from right to left
    CURRENT: in real;
    tankOUT: out std_logic);
end component tank_top;

component mux1 is
port(  inT, inM, inB: in std_logic;
    CURRENT: in real;
    CURRENT_V1, CURRENT_V2, CURRENT_V3: in real;
    selection: in std_logic_vector(1 downto 0);
    muxOUT, V1OUT, V3OUT, V2OUTt, V2OUTb: out std_logic);
end component mux1;

component mux2 is
port(  inR, inL: in std_logic;    --Right=result, Left=stored value
    CURRENT: in real;
    CURRENT_Vst, CURRENT_Vres: in real;
    selection: in std_logic;  --'1'=result, '0'=stored value
    muxOUT, VstOUT, VresOUT: out std_logic);
end component mux2;

signal CURRENT_Vstart, CURRENT_Vmove1, CURRENT_Vop, CURRENT_Vmove2,
↪  CURRENT_Vmove2C, CURRENT_Vmove3, CURRENT_MUX1_V1, CURRENT_MUX1_V2,
↪  CURRENT_MUX1_V3, CURRENT_Vmovecout, CURRENT_Vtankbot, CURRENT_Vnoext,
↪  CURRENT_Vmoveres, CURRENT_Vtanktop, CURRENT_Vmove4, CURRENT_Vnewdata,
↪  CURRENT_Vdetect, CURRENT_Vrout, CURRENT_Vsout, CURRENT_Vrestop: real;

signal MEM_out: std_logic;
signal x2_1_outtop, x2_1_outbottom, x2_2_outtop, x2_2_outbottom, x2_3_outtop,
↪  x2_3_outbottom, x2_4_outtop, x2_4_outbottom, x2_5_outright, x2_5_outleft,
↪  x2_6_outtop, x2_6_outbottom, x1_1_out: std_logic;
signal CROSSM1_outA, CROSSM1_outB, CROSSM2_outA, CROSSM2_outB, CROSSM3_outA,
↪  CROSSM3_outB: std_logic;
signal CROSS11_Aout, CROSS11_Bout, CROSS12_Aout, CROSS12_Bout, CROSS21_Aout,
↪  CROSS21_Bout, CROSS22_Aout, CROSS22_Bout, CROSS3_Aout, CROSS3_Bout,
↪  CROSS4_Aout, CROSS4_Bout, CROSS5_Aout, CROSS5_Bout, CROSS6_Aout,
↪  CROSS6_Bout: std_logic;
signal SYNC_LOG_TOP_out, SYNC_LOG_BOT_out, AND_out, OR_out, SKEW_OR_out,
↪  SKEW_AND_out: std_logic;
signal FA_CTRL1_out, FA_CTRL2_out, FA_COUT1_out, FA_COUT2_out, FA_SUM_out:
↪  std_logic;
signal RST_SR_MTJR1, RST_SR_MTJR2: std_logic;
signal MTJ_R1_out, MTJ_R2_out: std_logic;
signal MTJ_W1_out, MTJ_W2_out, MTJ_W3_out: std_logic;
signal MTJ_CONV1_out, MTJ_CONV2_out, MTJ_CONV3_out, MTJ_CONV4_out,
↪  MTJ_CONV5_out, MTJ_CONV6_out, MTJ_CONV7_out, MTJ_CONV8_out, MTJ_CONV9_out,
↪  MTJ_CONV10_out, MTJ_CONV11_out, MTJ_CONV12_out, MTJ_CONV13_out: std_logic;
```

```
  signal MUX1_CTRL_V1, MUX1_CTRL_V2, MUX1_CTRL_V3: std_logic;
  signal MUX1_out, MUX1_V1out, MUX1_V3out, MUX1_V2Tout, MUX1_V2Bout: std_logic;
  signal CTRL_mux2, MUX2_Vstout, MUX2_Vresout, MUX2_out: std_logic;
  signal JOIN1_out, JOIN2_out, JOIN3_out, JOIN5_out, JOIN6_out: std_logic;
  signal DEV1_out, DEV1_devout, DEV2_out, DEV2_devout, DEV3_out, DEV3_devout:
  ↪  std_logic;
  signal TANKB_out1, TANKB_out2, TANKB_out3: std_logic;
  signal TANKT_out: std_logic;


begin
  Vstart:  voltage_genH port map (CTRL => CTRL_Vstart, CURRENT =>
  ↪  CURRENT_Vstart);
  MEM:   SKYRMIONNOTCH port map (INPUT => IN_CELL, CURRENT => CURRENT_Vstart,
  ↪  OUTPUT => MEM_out);
  Vmove1:  voltage_genL port map (CTRL => CTRL_Vmove1, CURRENT =>
  ↪  CURRENT_Vmove1);
  x2_1:  skyrmionDUPLICATE port map (IN_SK => MEM_out, CURRENT =>
  ↪  CURRENT_Vmove1, OUT_SK_TOP => x2_1_outtop, OUT_SK_BOTTOM =>
  ↪  x2_1_outbottom);
  x2_2:  skyrmionDUPLICATE port map (IN_SK => x2_1_outtop, CURRENT =>
  ↪  CURRENT_Vmove1, OUT_SK_TOP => x2_2_outtop, OUT_SK_BOTTOM =>
  ↪  x2_2_outbottom);

  CROSSM1:       SKYRMIONCROSS_Magn port map (A => x2_2_outbottom, B =>
  ↪  x2_3_outtop, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout
  ↪  => CROSSM1_outA, Bout => CROSSM1_outB);
  Vop:         vclock_gen port map (CTRL => CTRL_Vop, CURRENTclk => CURRENTclk,
  ↪  CURRENT => CURRENT_Vop);
  SYNCNET_LOG_TOP:  SKYRMIONNOTCH port map (INPUT => x2_2_outtop, CURRENT =>
  ↪  CURRENT_Vop, OUTPUT => SYNC_LOG_TOP_out);
  SYNCNET_LOG_BOT:  SKYRMIONNOTCH port map (INPUT => CROSSM1_outB, CURRENT =>
  ↪  CURRENT_Vop, OUTPUT => SYNC_LOG_BOT_out);
  LOGIC:         SKYRMIONH port map (INPUTA => SYNC_LOG_TOP_out, INPUTB =>
  ↪  SYNC_LOG_BOT_out, CURRENT => CURRENT_Vop, OUTPUTAND => AND_OUT, OUTPUTOR
  ↪  => OR_OUT);
  SKEW_OR:       SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => OR_OUT,
  ↪  CURRENT => CURRENT_Vop, OUTPUT => SKEW_OR_out);
  SKEW_AND:       SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT =>
  ↪  AND_OUT, CURRENT => CURRENT_Vop, OUTPUT => SKEW_AND_out);
  FA:          SKYRMIONFULLADDER port map (A => CROSSM1_outA, B =>
  ↪  x2_3_outbottom, CIN1 => '0', CIN2 => '0', ONE1 => CROSSM2_outB, ONE2 =>
  ↪  CROSSM3_outB, CURRENT => CURRENT_Vop, CTRL1 => FA_CTRL1_out, SUM =>
  ↪  FA_SUM_out, COUT1 => FA_COUT1_out, COUT2 => FA_COUT2_out, CTRL2 =>
  ↪  FA_CTRL2_out);

  MTJ_CONV_1:  MTJ_CONV port map (IN_SK => SKEW_OR_out, CURRENT => CURRENT_Vop,
  ↪  OUT_SK => MTJ_CONV1_out);
  MTJ_CONV_2:  MTJ_CONV port map (IN_SK => SKEW_AND_out, CURRENT => CURRENT_Vop,
  ↪  OUT_SK => MTJ_CONV2_out);
```

```
MTJ_CONV_3:  MTJ_CONV port map (IN_SK => FA_SUM_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV3_out);
MTJ_CONV_4:  MTJ_CONV port map (IN_SK => FA_COUT1_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV4_out);
MTJ_CONV_5:  MTJ_CONV port map (IN_SK => FA_COUT2_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV5_out);
Vmove2:    voltage_genL port map (CTRL => CTRL_Vmove2, CURRENT => CURRENT_Vmove2);
Vmove2C:  voltage_genL port map (CTRL => CTRL_Vmove2C, CURRENT => CURRENT_Vmove2C);

MUX1_CTRL_V1 <= CTRL_mux1(0) and (not CTRL_mux1(1));
MUX1_CTRL_V2 <= CTRL_mux1(1) and (not CTRL_mux1(0));
MUX1_CTRL_V3 <= CTRL_mux1(1) and CTRL_mux1(0);
Vmux1_1:  voltage_genL port map (CTRL => MUX1_CTRL_V1, CURRENT => CURRENT_MUX1_V1);
Vmux1_2:  voltage_genL port map (CTRL => MUX1_CTRL_V2, CURRENT => CURRENT_MUX1_V2);
Vmux1_3:  voltage_genL port map (CTRL => MUX1_CTRL_V3, CURRENT => CURRENT_MUX1_V3);

MUX1_COM:     mux1 port map (inT => MTJ_CONV1_out, inM => MTJ_CONV2_out, inB => MTJ_CONV3_out, CURRENT => CURRENT_Vmove2, CURRENT_V1 => CURRENT_MUX1_V1, CURRENT_V2 => CURRENT_MUX1_V2, CURRENT_V3 => CURRENT_MUX1_V3, selection => CTRL_mux1, muxOUT => MUX1_out, V1OUT => MUX1_V1out, V3OUT => MUX1_V3out, V2OUTt => MUX1_V2Tout, V2OUTb => MUX1_V2Bout);
Vmove3:    voltage_genL port map (CTRL => CTRL_Vmove3, CURRENT => CURRENT_Vmove3);
x2_4:    skyrmionDUPLICATE port map (IN_SK => MUX1_out, CURRENT => CURRENT_Vmove3, OUT_SK_TOP => x2_4_outtop, OUT_SK_BOTTOM => x2_4_outbottom);
MTJ_CONV_7:  MTJ_CONV port map (IN_SK => x2_4_outbottom, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV7_out);

CROSS11:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV4_out, B => MUX1_V1out, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout => CROSS11_Aout, Bout => CROSS11_Bout);
CROSS12:  SKYRMIONCROSS_noMagn port map (A => CROSS11_Aout, B => MUX1_V2Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS12_Aout, Bout => CROSS12_Bout);
CROSS21:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV5_out, B => CROSS11_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout => CROSS21_Aout, Bout => CROSS21_Bout);
CROSS22:  SKYRMIONCROSS_noMagn port map (A => CROSS21_Aout, B => CROSS12_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS22_Aout, Bout => CROSS22_Bout);

Vmovecout:  voltage_genL port map (CTRL => CTRL_Vmovecout, CURRENT => CURRENT_Vmovecout);
```

```
x1_1:    skyrmionMERGE port map (IN_SK_TOP => CROSS12_Aout, IN_SK_BOTTOM => CROSS22_Aout, CURRENT => CURRENT_Vmovecout, OUT_SK => x1_1_out);
MTJ_CONV_6:  MTJ_CONV port map (IN_SK => x1_1_out, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV6_out);
CROSS4:   SKYRMIONCROSS_noMagn port map (A => MTJ_CONV6_out, B => MTJ_CONV7_out, CURRENTA => CURRENT_Vmovecout, CURRENTB => CURRENT_Vmove3, Aout => CROSS4_Aout, Bout => CROSS4_Bout);
CROSS6:   SKYRMIONCROSS_noMagn port map (A => CROSS4_Aout, B => CROSS5_Bout, CURRENTA => CURRENT_Vmovecout, CURRENTB => CURRENT_Vmoveres, Aout => CROSS6_Aout, Bout => CROSS6_Bout);

Vtankbot:  vclock_gen port map (CTRL => CTRL_Vtankbot, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vtankbot);
TANK_BOT:  tank_bottom port map (IN1 => FA_CTRL2_out, IN2 => FA_CTRL1_out, IN3 => CROSS21_Bout, IN4 => CROSS22_Bout, IN5 => JOIN5_out, IN6 => MUX2_Vresout, IN7 => MUX2_Vstout, CURRENT => CURRENT_Vtankbot, tankOUT1 => TANKB_out1, tankOUT2 => TANKB_out2, tankOUT3 => TANKB_out3);

MTJ_W_1:  MTJ_W port map (CTRL => CTRL_MTJ_W1, OUT_SK => MTJ_W1_out);
MTJ_W_2:  MTJ_W port map (CTRL => CTRL_MTJ_W2, OUT_SK => MTJ_W2_out);
JOIN2:    SKYRMIONJOIN port map (A => MTJ_W1_out, B => TANKB_out2, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN2_out);
JOIN3:    SKYRMIONJOIN port map (A => MTJ_W2_out, B => TANKB_out1, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN3_out);
CROSSM2:   SKYRMIONCROSS_Magn port map (A => TANKB_out1, B => JOIN2_out, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM2_outA, Bout => CROSSM2_outB);
CROSSM3:   SKYRMIONCROSS_Magn port map (A => CROSSM2_outA, B => JOIN3_out, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM3_outA, Bout => CROSSM3_outB);

MTJ_R_1:  MTJ_R port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vmove1, OUT_SIGN => MTJ_R1_out);
RST_SR_MTJR1 <= ACK_DETECT_MTJ_R1 or (not(RESET_n));
SR_MTJR1:  SRlatch port map (SET => MTJ_R1_out, RST => RST_SR_MTJR1, Q => SR_MTJR1_out);
MTJ_CONV_8:  MTJ_CONV port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vmove1, OUT_SK => MTJ_CONV8_out);
Vnoext:    voltage_genL port map (CTRL => CTRL_Vnoext, CURRENT => CURRENT_Vnoext);
DEV1:    skyrmionDEVIATION port map (IN_SK => MTJ_CONV8_out, CURRENT => CURRENT_Vmove1, CURRENTDEV => CURRENT_Vnoext, OUT_SK => DEV1_out, OUT_SK_DEV => DEV1_devout);
JOIN5:    SKYRMIONJOIN port map (A => DEV1_devout, B => DEV3_out, CURRENT => CURRENT_Vnoext, OUTPUT => JOIN5_out);
JOIN6:    SKYRMIONJOIN port map (A => DEV1_out, B => DEV3_devout, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN6_out);
Vrestop:  voltage_genL port map (CTRL => CTRL_Vrestop, CURRENT => CURRENT_Vrestop);
```

```
DEV3:    skyrmionDEVIATION port map (IN_SK => CROSS3_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vrestop, OUT_SK => DEV3_out, OUT_SK_DEV => DEV3_devout);

MTJ_W_3:  MTJ_W port map (CTRL => CTRL_MTJ_W3, OUT_SK => MTJ_W3_out);
JOIN1:    SKYRMIONJOIN port map (A => MTJ_W3_out, B => JOIN6_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN1_out);
x2_3:  skyrmionDUPLICATE port map (IN_SK => JOIN1_out, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_3_outtop, OUT_SK_BOTTOM => x2_3_outbottom);

MTJ_CONV_10:  MTJ_CONV port map (IN_SK => x2_1_outbottom, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV10_out);
CROSS3:     SKYRMIONCROSS_noMagn port map (A => TOP_RESULT, B => MTJ_CONV10_out, CURRENTA => TOP_RESULT_CURRENT, CURRENTB => CURRENT_Vmove3, Aout => CROSS3_Aout, Bout => CROSS3_Bout);
CTRL_mux2 <= CTRL_Vrout;
Vmux2_Vsout:  voltage_genL port map (CTRL => CTRL_Vsout, CURRENT => CURRENT_Vsout);
Vmux2_Vrout:  voltage_genL port map (CTRL => CTRL_Vrout, CURRENT => CURRENT_Vrout);
MUX2_COM:     mux2 port map (inR => CROSS4_Bout, inL => CROSS3_Bout, CURRENT => CURRENT_Vmove3, CURRENT_Vst => CURRENT_Vsout, CURRENT_Vres => CURRENT_Vrout, selection => CTRL_mux2, muxOUT => MUX2_out, VstOUT => MUX2_Vstout, VresOUT => MUX2_Vresout);

Vmoveres:    voltage_genL port map (CTRL => CTRL_Vmoveres, CURRENT => CURRENT_Vmoveres);
x2_5:       skyrmionDUPLICATE port map (IN_SK => MUX2_out, CURRENT => CURRENT_Vmoveres, OUT_SK_TOP => x2_5_outright, OUT_SK_BOTTOM => x2_5_outleft);
MTJ_CONV_12:  MTJ_CONV port map (IN_SK => x2_5_outleft, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV12_out);
x2_6:       skyrmionDUPLICATE port map (IN_SK => x2_5_outright, CURRENT => CURRENT_Vmoveres, OUT_SK_TOP => x2_6_outtop, OUT_SK_BOTTOM => x2_6_outbottom);
MTJ_CONV_11:  MTJ_CONV port map (IN_SK => x2_6_outtop, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV11_out);
MTJ_CONV_13:  MTJ_CONV port map (IN_SK => x2_6_outbottom, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV13_out);
CROSS5:     SKYRMIONCROSS_noMagn port map (A => MTJ_CONV11_out, B => MTJ_CONV13_out, CURRENTA => CURRENT_Vmoveres, CURRENTB => CURRENT_Vmoveres, Aout => CROSS5_Aout, Bout => CROSS5_Bout);

Vtanktop:  vclock_gen port map (CTRL => CTRL_Vtanktop, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vtanktop);
TANK_TOP_C:  tank_top port map (IN1 => MUX1_V2Tout, IN2 => MUX1_V3out, IN3 => DEV2_devout, CURRENT => CURRENT_Vtanktop, tankOUT => TANKT_out);
MTJ_CONV_9:  MTJ_CONV port map (IN_SK => x2_4_outtop, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV9_out);
```

```
  Vmove4:    voltage_genL port map (CTRL => CTRL_Vmove4, CURRENT => CURRENT_Vmove4);
  Vnewdata:  voltage_genL port map (CTRL => CTRL_Vnewdata, CURRENT => CURRENT_Vnewdata);
  DEV2:    skyrmionDEVIATION port map (IN_SK => MTJ_CONV9_out, CURRENT => CURRENT_Vmove4, CURRENTDEV => CURRENT_Vnewdata, OUT_SK => DEV2_out, OUT_SK_DEV => DEV2_devout);
  Vdetect:  voltage_genL port map (CTRL => CTRL_Vdetect, CURRENT => CURRENT_Vdetect);
  MTJ_R_2:  MTJ_R port map (IN_SK => TANKT_out, CURRENT => CURRENT_Vdetect, OUT_SIGN => MTJ_R2_out);
  RST_SR_MTJR2 <= ACK_DETECT_MTJ_R2 or (not RESET_n);
  SR_MTJR2:  SRlatch port map (SET => MTJ_R2_out, RST => RST_SR_MTJR2, Q => SR_MTJR2_out);

  CELL_OUT_MTJ_CONV12_out   <= MTJ_CONV12_out;
  CELL_OUT_CROSS5_Aout     <= CROSS5_Aout;
  CELL_OUT_CROSS6_Aout     <= CROSS6_Aout;
  CELL_OUT_CROSS6_Bout     <= CROSS6_Bout;
  CELL_OUT_DEV2_out       <= DEV2_out;
  CELL_OUT_TANKT_out      <= TANKT_out;

  CELL_OUT_CURRENT_Vmoveres   <= CURRENT_Vmoveres;
  CELL_OUT_CURRENT_Vmovecout      <= CURRENT_Vmovecout;


end Structure;
```

C.3.2. FSM

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;

entity FSM_CELL_X0 is
port(  CURRENTclk: in real;
    RESET_n: in std_logic;

    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop: out std_logic;
```

```
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
    CTRL_mux1: out std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
    SR_MTJR2_out, SR_MTJR1_out: in std_logic;

    START, ACK_RES_AVAILABLE: in std_logic;
    DES_EXTIN_X0, DES_OUT_X0, DES_STORE_X0, DES_NEWDATA_X0: in std_logic;
    DES_RES_X0: in std_logic_vector(1 downto 0);
    RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE: out std_logic;
    READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic;

    CTRL_Vrestop: out std_logic;
    READY_FOR_NEWDATA: out std_logic;
    DES_DATA_X0: in std_logic
);
end entity FSM_CELL_X0;

architecture Behaviour of FSM_CELL_X0 is

  type state_type is (
    reset, S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10,
    S6_1, S6_2, S6_3, S6_4, S6_5, S6_6,
    S11, S12, S13, S14, S15, S16, S17, S18, S19, S20,
    S21, S22, S23, S24, S25, S26, S27, S28, S29, S30,
    S31, S32, S33, S34, S35, S36, S37, S38, S39, S40
  );

  signal pstate, nstate: state_type;

begin

  state_register: process (CURRENTclk)
  begin
    if (CURRENTclk'event and CURRENTclk=CURRENT_HIGH) then
      if (RESET_n = '0') then
        pstate <= reset;
      else
        pstate <= nstate;
      end if;
    end if;
  end process state_register;

  state_transition: process (pstate, CURRENTclk)
  begin
    case pstate is
      when reset  => nstate <= S0;
      when S0   => if (START='0') then nstate <= S0; else if(FIRST_RUN='1') then nstate <= S1; else nstate <= S31; end if; end if;
```

```
        when S1   => if(DES_DATA_X0='1') then nstate <= S2; else nstate <= S3; end if;
        when S2   => if(DES_EXTIN_X0='1') then nstate <= S4; else nstate <= S5; end if;
        when S3   => nstate <= S5;
        when S4   => nstate <= S6;
        when S5   => nstate <= S6;
        when S6   => nstate <= S6_1;
        when S6_1   => nstate <= S6_2;
        when S6_2   => nstate <= S6_3;
        when S6_3   => nstate <= S6_4;
        when S6_4   => nstate <= S6_5;
        when S6_5   => nstate <= S6_6;
        when S6_6   => nstate <= S7;
        when S7   => nstate <= S8;
        when S8   => if (DES_RES_X0="10") then nstate <= S9; elsif (DES_RES_X0="01") then nstate <= S10; else nstate <= S11; end if;
        when S9   => nstate <= S12;
        when S10   => nstate <= S12;
        when S11   => nstate <= S12;
        when S12   => nstate <= S13;
        when S13   => if (DES_OUT_X0='1') then nstate <= S14; else nstate <= S15; end if;
        when S14   => nstate <= S16;
        when S15   => nstate <= S16;
        when S16   => nstate <= S17;
        when S17   => if(DES_STORE_X0='1') then nstate <= S18; else nstate <= S19; end if;
        when S18   => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S18; else nstate <= S20; end if;
        when S19   => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S19; else if(DES_NEWDATA_X0='1') then nstate <= S21; else nstate <= S22; end if; end if;
        when S20   => nstate <= S24;
        when S21   => nstate <= S23;
        when S22   => nstate <= S26;
        when S23  => nstate <= S25;
        when S24   => if (ACK_RES_AVAILABLE='0') then nstate <= S24; else nstate <= S28; end if;
        when S25   => if (ACK_RES_AVAILABLE='0') then nstate <= S25; else nstate <= S27; end if;
        when S26   => if (ACK_RES_AVAILABLE='0') then nstate <= S26; else nstate <= S30; end if;
        when S27  => if (SR_MTJR2_out='0') then nstate <= S29; else nstate <= S30; end if;
        when S28   => if(START='0') then nstate <= S28; else nstate <= S31; end if;
        when S29   => if(START='0') then nstate <= S29; else nstate <= S31; end if;
```

```
      when S30   => if(START='0') then nstate <= S30; else nstate <= S31; end if;
      when S31   => nstate <= S32;
      when S32   => if(DES_DATA_X0='0') then nstate <= S33; else nstate <= S34; end if;
      when S33   => nstate <= S35;
      when S34   => if(DES_EXTIN_X0='0') then nstate <= S35; else nstate <= S36; end if;
      when S35   => if(SR_MTJR1_out='0') then nstate <= S39; else nstate <= S37; end if;
      when S36   => if(SR_MTJR1_out='1') then nstate <= S40; else nstate <= S38; end if;
      when S37   => nstate <= S39;
      when S38   => nstate <= S40;
      when S39   => nstate <= S6;
      when S40   => nstate <= S6;
      when others => nstate <= S0;
    end case;
  end process state_transition;

  output: process (pstate)
  begin
    CTRL_Vstart <= '0';
    CTRL_Vmove1 <= '0';
    CTRL_Vop <= '0';
    CTRL_Vmove2 <= '0';
    CTRL_Vmove2C <= '0';
    CTRL_Vmove3 <= '0';
    CTRL_Vmovecout <= '0';
    CTRL_Vtankbot <= '0';
    CTRL_Vnoext <= '0';
    CTRL_Vmoveres <= '0';
    CTRL_Vmove4 <= '0';
    CTRL_Vnewdata <= '0';
    CTRL_Vdetect <= '0';
    CTRL_Vrout <= '0';
    CTRL_Vsout <= '0';
    CTRL_Vtanktop <= '0';
    CTRL_MTJ_W1 <= '0';
    CTRL_MTJ_W2 <= '0';
    CTRL_MTJ_W3 <= '0';
    CTRL_mux1 <= "00";
    ACK_DETECT_MTJ_R1 <= '0';
    ACK_DETECT_MTJ_R2 <= '0';
    RES_AVAILABLE <= '0';
    REQUEST_NEW_START <= '0';
    REQUEST_NEW_START_W <= '0';
    REQUEST_NEW_STORE <= '0';
    ---
```

```
    CTRL_Vrestop <= '0';
    READY_FOR_NEWDATA <= '0';

    case pstate is
      when S0    =>  CTRL_Vstart <= '0';
              CTRL_Vmove1 <= '0';
              CTRL_Vop <= '0';
              CTRL_Vmove2 <= '0';
              CTRL_Vmove2C <= '0';
              CTRL_Vmove3 <= '0';
              CTRL_Vmovecout <= '0';
              CTRL_Vtankbot <= '0';
              CTRL_Vnoext <= '0';
              CTRL_Vmoveres <= '0';
              CTRL_Vmove4 <= '0';
              CTRL_Vnewdata <= '0';
              CTRL_Vdetect <= '0';
              CTRL_Vrout <= '0';
              CTRL_Vsout <= '0';
              CTRL_Vtanktop <= '0';
              CTRL_MTJ_W1 <= '0';
              CTRL_MTJ_W2 <= '0';
              CTRL_MTJ_W3 <= '0';
              CTRL_mux1 <= "00";
              ACK_DETECT_MTJ_R1 <= '0';
              ACK_DETECT_MTJ_R2 <= '0';
              RES_AVAILABLE <= '0';
              REQUEST_NEW_START <= '0';
              REQUEST_NEW_START_W <= '0';
              REQUEST_NEW_STORE <= '0';
              ---
              CTRL_Vrestop <= '0';
      when S1    =>  CTRL_Vnoext <= '1', '0' after CLOCK_PERIOD/2;
              READY_FOR_NEWDATA <= '1';
      when S2    =>  CTRL_Vnoext <= '1';
              CTRL_Vstart <= '1';
      when S3    =>  CTRL_Vrestop <= '1';
              CTRL_Vstart <= '1';
      when S4    =>  CTRL_Vmove1 <= '1';
              CTRL_MTJ_W1 <= '1';
              CTRL_MTJ_W2 <= '1';
              CTRL_MTJ_W3 <= '1';
      when S5    =>  CTRL_Vmove1 <= '1';
              CTRL_MTJ_W1 <= '1';
              CTRL_MTJ_W2 <= '1';
      when S6    =>  CTRL_Vop <= '1';
      when S6_1  =>  CTRL_Vop <= '1';
      when S6_2  =>  CTRL_Vop <= '1';
      when S6_3  =>  CTRL_Vop <= '1';
```

```
      when S6_4  =>  CTRL_Vop <= '1';
      when S6_5  =>  CTRL_Vop <= '1';
      when S6_6  =>  CTRL_Vop <= '1';
      when S7    =>  CTRL_Vmove2 <= '1';
              CTRL_Vmove2C <= '1';
      when S8    =>  CTRL_Vmove2C <= '1';
      when S9    =>  CTRL_mux1 <= "10";
      when S10  =>  CTRL_mux1 <= "01";
      when S11  =>  CTRL_mux1 <= "11";
      when S12  =>  CTRL_Vmove2 <= '1';
      when S13  =>  CTRL_Vmove3 <= '1';
      when S14  =>  CTRL_Vrout <= '1';
      when S15  =>  CTRL_Vsout <= '1';
      when S16  =>  CTRL_Vmove3 <= '1';
      when S17  =>  CTRL_Vmove4 <= '1', '0' after CLOCK_PERIOD/2;
      when S18  =>  CTRL_Vmove4 <= '1';
      when S19  =>  CTRL_Vnewdata <= '1';
      when S20  =>  CTRL_Vmoveres <= '1';
              CTRL_Vmovecout <= '1';
      when S21  =>  CTRL_Vtanktop <= '1';
              CTRL_Vmoveres <= '1';
              CTRL_Vmovecout <= '1';
      when S22  =>  CTRL_Vmoveres <= '1';
              CTRL_Vmovecout <= '1';
      when S23  =>  CTRL_Vtanktop <= '1';
      when S24  =>  RES_AVAILABLE  <= '1';
      when S25  =>  RES_AVAILABLE  <= '1';
      when S26  =>  RES_AVAILABLE  <= '1';
      when S27  =>  CTRL_Vdetect <= '1';
      when S28  =>  REQUEST_NEW_STORE <= '1';
      when S29  =>  REQUEST_NEW_START_W <= '1';
              ACK_DETECT_MTJ_R2 <= '1';
      when S30  =>  REQUEST_NEW_START <= '1';
              ACK_DETECT_MTJ_R2 <= '1';
      when S31  =>  CTRL_Vstart <= '1';
              CTRL_Vtankbot <= '1';
      when S32  =>  CTRL_Vtankbot <= '1';
              CTRL_Vnoext <= '1', '0' after CLOCK_PERIOD/2;
              READY_FOR_NEWDATA <= '1';
      when S33  =>  CTRL_Vrestop <= '1';
      when S34  =>  CTRL_Vnoext <= '1';
      when S35  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
      when S36  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
      when S37  =>  CTRL_Vnoext <= '1';
              ACK_DETECT_MTJ_R1 <= '1';
      when S38  =>  CTRL_MTJ_W3 <= '1';
      when S39  =>  CTRL_Vmove1 <= '1';
      when S40  =>  CTRL_Vmove1 <= '1';
              ACK_DETECT_MTJ_R1 <= '1';
```

```

      when others =>  CTRL_Vstart <= '0';
              CTRL_Vmove1 <= '0';
              CTRL_Vop <= '0';
              CTRL_Vmove2 <= '0';
              CTRL_Vmove2C <= '0';
              CTRL_Vmove3 <= '0';
              CTRL_Vmovecout <= '0';
              CTRL_Vtankbot <= '0';
              CTRL_Vnoext <= '0';
              CTRL_Vmoveres <= '0';
              CTRL_Vmove4 <= '0';
              CTRL_Vnewdata <= '0';
              CTRL_Vdetect <= '0';
              CTRL_Vrout <= '0';
              CTRL_Vsout <= '0';
              CTRL_Vtanktop <= '0';
              CTRL_MTJ_W1 <= '0';
              CTRL_MTJ_W2 <= '0';
              CTRL_MTJ_W3 <= '0';
              CTRL_mux1 <= "00";
              ACK_DETECT_MTJ_R1 <= '0';
              ACK_DETECT_MTJ_R2 <= '0';
              RES_AVAILABLE <= '0';
              REQUEST_NEW_START <= '0';
              REQUEST_NEW_START_W <= '0';
              REQUEST_NEW_STORE <= '0';
              ---
              CTRL_Vrestop <= '0';
    end case;
  end process output;
end Behaviour;
```

C.4. Cell01

C.4.1. Datapath

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
```

```
use work.globals.all;


entity CELL_OX is
port(  IN_CELL, RESET_n: in std_logic;
    CURRENTclk: in real;

    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3,
    ↪  CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4,
    ↪  CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop,
    ↪  CTRL_Vouttank: in std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
    CTRL_mux1: in std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
    SR_MTJR1_out, SR_MTJR2_out: out std_logic;
    CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

    CELL_OUT_MTJ_CONV14_out, CELL_OUT_CROSS9_Bout, CELL_OUT_CROSS9_Aout,
    ↪  CELL_OUT_MTJ_CONV12_out, CELL_OUT_DEV2_out, CELL_OUT_TANKT_out: out
    ↪  std_logic;

    CTRL_Vresbotleft, CTRL_Vresleft: in std_logic;
    LEFT_COUT, LEFT_RESULT, BOTLEFT_RESULT:  in std_logic;
    LEFT_COUT_CURRENT, LEFT_RESULT_CURRENT, BOTLEFT_RESULT_CURRENT: in real
);
end entity CELL_OX;

architecture Structure of CELL_OX is
  component SKYRMIONNOTCH is
  port(  INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  component SKYRMIONNOTCHseq is
  generic (N: integer := 5);
  port(   INPUT: in std_logic;
      CURRENT: in real;
      OUTPUT: out std_logic);
  end component SKYRMIONNOTCHseq;

  component SKYRMIONJOIN is
  port( A : in std_logic;
      B : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONJOIN;

  component SKYRMIONH is
```

```
port (  INPUTA : in std_logic;
    INPUTB : in std_logic;
    CURRENT : in real;
    OUTPUTAND : out std_logic;
    OUTPUTOR : out std_logic);
end component SKYRMIONH;

component SKYRMIONFULLADDER is
port(   A     : in std_logic;
    B     : in std_logic;
    CIN1   : in std_logic;
    CIN2   : in std_logic;
    ONE1   : in std_logic;
    ONE2   : in std_logic;
    CURRENT  : in real;
    CTRL1   : out std_logic;
    SUM     : out std_logic;
    COUT1   : out std_logic;
    COUT2   : out std_logic;
    CTRL2   : out std_logic);
end component SKYRMIONFULLADDER;

component voltage_genH is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genH;

component voltage_genL is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genL;

component vclock_gen is
port(  CTRL: in std_logic;
    CURRENTclk: in real;
    CURRENT:  out real);
end component vclock_gen;

component skyrmionDUPLICATE is
port(  IN_SK:       in std_logic;
    CURRENT:    in real;
    OUT_SK_TOP:    out std_logic;
    OUT_SK_BOTTOM:  out std_logic);
end component skyrmionDUPLICATE;

component skyrmionMERGE is
port(  IN_SK_TOP:     in std_logic;
    IN_SK_BOTTOM:  in std_logic;
    CURRENT:    in real;
```

```
        OUT_SK:      out std_logic);
    end component skyrmionMERGE;

    component SKYRMIONCROSS_Magn is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENTA:   in real;
        CURRENTB:  in real;
        Aout:     out std_logic;
        Bout:     out std_logic);
    end component SKYRMIONCROSS_Magn;

    component SKYRMIONCROSS_noMagn is
    port(   A:       in std_logic;
        B:       in std_logic;
        CURRENTA:   in real;
        CURRENTB:  in real;
        Aout:     out std_logic;
        Bout:     out std_logic);
    end component SKYRMIONCROSS_noMagn;

    component skyrmionDEVIATION is
    port(  IN_SK:     in std_logic;
        CURRENT:  in real;
        CURRENTDEV:  in real;
        OUT_SK:    out std_logic;
        OUT_SK_DEV:  out std_logic);
    end component skyrmionDEVIATION;

    component SRlatch is
    port(  SET:   in std_logic;
        RST:   in std_logic;
        Q:    buffer std_logic);
    end component SRlatch;

    component MTJ_R is
    port(  IN_SK:     in std_logic;
        CURRENT:  in real;
        OUT_SIGN:  out std_logic);
    end component MTJ_R;

    component MTJ_W is
    port(  CTRL: in std_logic;
        OUT_SK:  out std_logic);
    end component MTJ_W;

    component MTJ_CONV is
    port(  IN_SK: in std_logic;
        CURRENT: in real;
```

```
        OUT_SK:  out std_logic);
    end component MTJ_CONV;

    component tank_bottom is
    port(  IN1, IN2, IN3, IN4, IN5, IN6, IN7: in std_logic;  --from left to right
        CURRENT: in real;
        tankOUT1, tankOUT2, tankOUT3: out std_logic);    --priority order: from right to left
    end component tank_bottom;

    component tank_top is
    port(  IN1, IN2, IN3: in std_logic; --from right to left
        CURRENT: in real;
        tankOUT: out std_logic);
    end component tank_top;

    component mux1 is
    port(  inT, inM, inB: in std_logic;
        CURRENT: in real;
        CURRENT_V1, CURRENT_V2, CURRENT_V3: in real;
        selection: in std_logic_vector(1 downto 0);
        muxOUT, V1OUT, V3OUT, V2OUTt, V2OUTb: out std_logic);
    end component mux1;

    component mux2 is
    port(  inR, inL: in std_logic;    --Right=result, Left=stored value
        CURRENT: in real;
        CURRENT_Vst, CURRENT_Vres: in real;
        selection: in std_logic;  --'1'=result, '0'=stored value
        muxOUT, VstOUT, VresOUT: out std_logic);
    end component mux2;

    signal CURRENT_Vstart, CURRENT_Vmove1, CURRENT_Vop, CURRENT_Vmove2, CURRENT_Vmove2C, CURRENT_Vmove3, CURRENT_MUX1_V1, CURRENT_MUX1_V2, CURRENT_MUX1_V3, CURRENT_Vmovecout, CURRENT_Vtankbot, CURRENT_Vnoext, CURRENT_Vmoveres, CURRENT_Vtanktop, CURRENT_Vmove4, CURRENT_Vnewdata, CURRENT_Vdetect, CURRENT_Vrout, CURRENT_Vsout, CURRENT_Vrestop, CURRENT_Vresbotleft, CURRENT_Vresleft, CURRENT_Vouttank: real;

    signal MEM_out: std_logic;
    signal x2_1_outtop, x2_1_outbottom, x2_2_outtop, x2_2_outbottom, x2_3_outtop, x2_3_outbottom, x2_4_outtop, x2_4_outbottom, x2_5_outright, x2_5_outleft, x2_7_outtop, x2_7_outbottom, x2_8_outtop, x2_8_outbottom, x1_1_out: std_logic;
    signal CROSSM1_outA, CROSSM1_outB, CROSSM2_outA, CROSSM2_outB, CROSSM3_outA, CROSSM3_outB, CROSSM4_outA, CROSSM4_outB: std_logic;
    signal CROSS11_Aout, CROSS11_Bout, CROSS12_Aout, CROSS12_Bout, CROSS21_Aout, CROSS21_Bout, CROSS22_Aout, CROSS22_Bout, CROSS4_Aout, CROSS4_Bout, CROSS5_Aout, CROSS5_Bout, CROSS6_Aout, CROSS6_Bout, CROSS7_Aout, CROSS7_Bout, CROSS8_Aout, CROSS8_Bout, CROSS9_Aout, CROSS9_Bout: std_logic;
```

```
  signal SYNC_LOG_TOP_out, SYNC_LOG_BOT_out, AND_out, OR_out, SKEW_OR_out, SKEW_AND_out: std_logic;
  signal FA_CTRL1_out, FA_CTRL2_out, FA_COUT1_out, FA_COUT2_out, FA_SUM_out: std_logic;
  signal RST_SR_MTJR1, RST_SR_MTJR2: std_logic;
  signal MTJ_R1_out, MTJ_R2_out: std_logic;
  signal MTJ_W1_out, MTJ_W2_out, MTJ_W3_out: std_logic;
  signal MTJ_CONV1_out, MTJ_CONV2_out, MTJ_CONV3_out, MTJ_CONV4_out, MTJ_CONV5_out, MTJ_CONV6_out, MTJ_CONV7_out, MTJ_CONV8_out, MTJ_CONV9_out, MTJ_CONV10_out, MTJ_CONV11_out, MTJ_CONV12_out, MTJ_CONV14_out, MTJ_CONV15_out: std_logic;
  signal MUX1_CTRL_V1, MUX1_CTRL_V2, MUX1_CTRL_V3: std_logic;
  signal MUX1_out, MUX1_V1out, MUX1_V3out, MUX1_V2Tout, MUX1_V2Bout: std_logic;
  signal CTRL_mux2, MUX2_Vstout, MUX2_Vresout, MUX2_out: std_logic;
  signal JOIN1_out, JOIN2_out, JOIN3_out, JOIN7_out, JOIN8_out, JOIN9_out, JOIN10_out: std_logic;
  signal DEV1_out, DEV1_devout, DEV2_out, DEV2_devout, DEV4_out, DEV4_devout, DEV5_out, DEV5_devout: std_logic;
  signal TANKB_out1, TANKB_out2, TANKB_out3: std_logic;
  signal TANKT_out: std_logic;


begin
  Vstart:  voltage_genH port map (CTRL => CTRL_Vstart, CURRENT => CURRENT_Vstart);
  MEM:   SKYRMIONNOTCH port map (INPUT => IN_CELL, CURRENT => CURRENT_Vstart, OUTPUT => MEM_out);
  Vmove1:  voltage_genL port map (CTRL => CTRL_Vmove1, CURRENT => CURRENT_Vmove1);
  x2_1:  skyrmionDUPLICATE port map (IN_SK => MEM_out, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_1_outtop, OUT_SK_BOTTOM => x2_1_outbottom);
  x2_2:  skyrmionDUPLICATE port map (IN_SK => x2_1_outtop, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_2_outtop, OUT_SK_BOTTOM => x2_2_outbottom);

  CROSSM1:       SKYRMIONCROSS_Magn port map (A => x2_2_outbottom, B => CROSSM4_outB, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM1_outA, Bout => CROSSM1_outB);
  Vop:         vclock_gen port map (CTRL => CTRL_Vop, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vop);
  SYNCNET_LOG_TOP:  SKYRMIONNOTCH port map (INPUT => x2_2_outtop, CURRENT => CURRENT_Vop, OUTPUT => SYNC_LOG_TOP_out);
  SYNCNET_LOG_BOT:  SKYRMIONNOTCH port map (INPUT => CROSSM1_outB, CURRENT => CURRENT_Vop, OUTPUT => SYNC_LOG_BOT_out);
  LOGIC:        SKYRMIONH port map (INPUTA => SYNC_LOG_TOP_out, INPUTB => SYNC_LOG_BOT_out, CURRENT => CURRENT_Vop, OUTPUTAND => AND_OUT, OUTPUTOR => OR_OUT);
  SKEW_OR:      SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => OR_OUT, CURRENT => CURRENT_Vop, OUTPUT => SKEW_OR_out);
```

```
SKEW_AND:      SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT =>
↪  AND_OUT, CURRENT => CURRENT_Vop, OUTPUT => SKEW_AND_out);
FA:           SKYRMIONFULLADDER port map (A => CROSSM1_outA, B =>
↪  x2_3_outbottom, CIN1 => x2_7_outtop, CIN2 => x2_7_outbottom, ONE1 =>
↪  CROSSM2_outB, ONE2 => CROSSM3_outB, CURRENT => CURRENT_Vop, CTRL1 =>
↪  FA_CTRL1_out, SUM => FA_SUM_out, COUT1 => FA_COUT1_out, COUT2 =>
↪  FA_COUT2_out, CTRL2 => FA_CTRL2_out);

MTJ_CONV_1:  MTJ_CONV port map (IN_SK => SKEW_OR_out, CURRENT => CURRENT_Vop,
↪  OUT_SK => MTJ_CONV1_out);
MTJ_CONV_2:  MTJ_CONV port map (IN_SK => SKEW_AND_out, CURRENT => CURRENT_Vop,
↪  OUT_SK => MTJ_CONV2_out);
MTJ_CONV_3:  MTJ_CONV port map (IN_SK => FA_SUM_out, CURRENT => CURRENT_Vop,
↪  OUT_SK => MTJ_CONV3_out);
MTJ_CONV_4:  MTJ_CONV port map (IN_SK => FA_COUT1_out, CURRENT => CURRENT_Vop,
↪  OUT_SK => MTJ_CONV4_out);
MTJ_CONV_5:  MTJ_CONV port map (IN_SK => FA_COUT2_out, CURRENT => CURRENT_Vop,
↪  OUT_SK => MTJ_CONV5_out);
Vmove2:     voltage_genL port map (CTRL => CTRL_Vmove2, CURRENT =>
↪  CURRENT_Vmove2);
Vmove2C:  voltage_genL port map (CTRL => CTRL_Vmove2C, CURRENT =>
↪  CURRENT_Vmove2C);

MUX1_CTRL_V1 <= CTRL_mux1(0) and (not CTRL_mux1(1));
MUX1_CTRL_V2 <= CTRL_mux1(1) and (not CTRL_mux1(0));
MUX1_CTRL_V3 <= CTRL_mux1(1) and CTRL_mux1(0);
Vmux1_1:  voltage_genL port map (CTRL => MUX1_CTRL_V1, CURRENT =>
↪  CURRENT_MUX1_V1);
Vmux1_2:  voltage_genL port map (CTRL => MUX1_CTRL_V2, CURRENT =>
↪  CURRENT_MUX1_V2);
Vmux1_3:  voltage_genL port map (CTRL => MUX1_CTRL_V3, CURRENT =>
↪  CURRENT_MUX1_V3);

MUX1_COM:      mux1 port map (inT => MTJ_CONV1_out, inM => MTJ_CONV2_out, inB
↪  => MTJ_CONV3_out, CURRENT => CURRENT_Vmove2, CURRENT_V1 =>
↪  CURRENT_MUX1_V1, CURRENT_V2 => CURRENT_MUX1_V2, CURRENT_V3 =>
↪  CURRENT_MUX1_V3, selection => CTRL_mux1, muxOUT => MUX1_out, V1OUT =>
↪  MUX1_V1out, V3OUT => MUX1_V3out, V2OUTt => MUX1_V2Tout, V2OUTb =>
↪  MUX1_V2Bout);
Vmove3:     voltage_genL port map (CTRL => CTRL_Vmove3, CURRENT =>
↪  CURRENT_Vmove3);
x2_4:    skyrmionDUPLICATE port map (IN_SK => MUX1_out, CURRENT =>
↪  CURRENT_Vmove3, OUT_SK_TOP => x2_4_outtop, OUT_SK_BOTTOM =>
↪  x2_4_outbottom);
MTJ_CONV_7:  MTJ_CONV port map (IN_SK => x2_4_outbottom, CURRENT =>
↪  CURRENT_Vmove3, OUT_SK => MTJ_CONV7_out);

CROSS11:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV4_out, B => MUX1_V1out,
↪  CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout =>
↪  CROSS11_Aout, Bout => CROSS11_Bout);
```

```
CROSS12:  SKYRMIONCROSS_noMagn port map (A => CROSS11_Aout, B => MUX1_V2Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS12_Aout, Bout => CROSS12_Bout);
CROSS21:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV5_out, B => CROSS11_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout => CROSS21_Aout, Bout => CROSS21_Bout);
CROSS22:  SKYRMIONCROSS_noMagn port map (A => CROSS21_Aout, B => CROSS12_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS22_Aout, Bout => CROSS22_Bout);

Vmovecout:  voltage_genL port map (CTRL => CTRL_Vmovecout, CURRENT => CURRENT_Vmovecout);
x1_1:    skyrmionMERGE port map (IN_SK_TOP => CROSS12_Aout, IN_SK_BOTTOM => CROSS22_Aout, CURRENT => CURRENT_Vmovecout, OUT_SK => x1_1_out);
MTJ_CONV_6:  MTJ_CONV port map (IN_SK => x1_1_out, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV6_out);
CROSS4:   SKYRMIONCROSS_noMagn port map (A => MTJ_CONV6_out, B => MTJ_CONV7_out, CURRENTA => CURRENT_Vmovecout, CURRENTB => CURRENT_Vmove3, Aout => CROSS4_Aout, Bout => CROSS4_Bout);

x2_8:      skyrmionDUPLICATE port map (IN_SK => CROSS4_Aout, CURRENT => CURRENT_Vmovecout, OUT_SK_TOP => x2_8_outtop, OUT_SK_BOTTOM => x2_8_outbottom);
MTJ_CONV_14:  MTJ_CONV port map (IN_SK => x2_8_outtop, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV14_out);
MTJ_CONV_15:  MTJ_CONV port map (IN_SK => x2_8_outbottom, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV15_out);
CROSS9:     SKYRMIONCROSS_noMagn port map (A => MTJ_CONV11_out, B => MTJ_CONV15_out, CURRENTA => CURRENT_Vmoveres, CURRENTB => CURRENT_Vmovecout, Aout => CROSS9_Aout, Bout => CROSS9_Bout);

Vtankbot:  vclock_gen port map (CTRL => CTRL_Vtankbot, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vtankbot);
TANK_BOT:  tank_bottom port map (IN1 => FA_CTRL2_out, IN2 => FA_CTRL1_out, IN3 => CROSS21_Bout, IN4 => CROSS22_Bout, IN5 => JOIN10_out, IN6 => MUX2_Vresout, IN7 => MUX2_Vstout, CURRENT => CURRENT_Vtankbot, tankOUT1 => TANKB_out1, tankOUT2 => TANKB_out2, tankOUT3 => TANKB_out3);

MTJ_W_1:  MTJ_W port map (CTRL => CTRL_MTJ_W1, OUT_SK => MTJ_W1_out);
MTJ_W_2:  MTJ_W port map (CTRL => CTRL_MTJ_W2, OUT_SK => MTJ_W2_out);
JOIN2:    SKYRMIONJOIN port map (A => MTJ_W1_out, B => TANKB_out2, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN2_out);
JOIN3:    SKYRMIONJOIN port map (A => MTJ_W2_out, B => TANKB_out1, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN3_out);
Vouttank:  voltage_genL port map (CTRL => CTRL_Vouttank, CURRENT => CURRENT_Vouttank);
CROSSM2:   SKYRMIONCROSS_Magn port map (A => TANKB_out1, B => JOIN2_out, CURRENTA => CURRENT_Vouttank, CURRENTB => CURRENT_Vmove1, Aout => CROSSM2_outA, Bout => CROSSM2_outB);
```

```
CROSSM3:   SKYRMIONCROSS_Magn port map (A => CROSSM2_outA, B => JOIN3_out, CURRENTA => CURRENT_Vouttank, CURRENTB => CURRENT_Vmove1, Aout => CROSSM3_outA, Bout => CROSSM3_outB);

MTJ_R_1:  MTJ_R port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vouttank, OUT_SIGN => MTJ_R1_out);
RST_SR_MTJR1 <= ACK_DETECT_MTJ_R1 or (not(RESET_n));
SR_MTJR1:  SRlatch port map (SET => MTJ_R1_out, RST => RST_SR_MTJR1, Q => SR_MTJR1_out);
MTJ_CONV_8:  MTJ_CONV port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vouttank, OUT_SK => MTJ_CONV8_out);
Vnoext:    voltage_genL port map (CTRL => CTRL_Vnoext, CURRENT => CURRENT_Vnoext);
DEV1:     skyrmionDEVIATION port map (IN_SK => MTJ_CONV8_out, CURRENT => CURRENT_Vmove1, CURRENTDEV => CURRENT_Vnoext, OUT_SK => DEV1_out, OUT_SK_DEV => DEV1_devout);
JOIN9:    SKYRMIONJOIN port map (A => DEV1_devout, B => CROSS8_Aout, CURRENT => CURRENT_Vnoext, OUTPUT => JOIN9_out);
JOIN10:    SKYRMIONJOIN port map (A => JOIN9_out, B => DEV5_out, CURRENT => CURRENT_Vnoext, OUTPUT => JOIN10_out);

Vresleft:    voltage_genL port map (CTRL => CTRL_Vresleft, CURRENT => CURRENT_Vresleft);
Vresbotleft:  voltage_genL port map (CTRL => CTRL_Vresbotleft, CURRENT => CURRENT_Vresbotleft);
DEV4:       skyrmionDEVIATION port map (IN_SK => CROSS6_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vresleft, OUT_SK => DEV4_out, OUT_SK_DEV => DEV4_devout);
DEV5:       skyrmionDEVIATION port map (IN_SK => CROSS7_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vresbotleft, OUT_SK => DEV5_out, OUT_SK_DEV => DEV5_devout);
CROSS8:     SKYRMIONCROSS_noMagn port map (A => DEV4_out, B => DEV5_devout, CURRENTA => CURRENT_Vnoext, CURRENTB => CURRENT_Vresbotleft, Aout => CROSS8_Aout, Bout => CROSS8_Bout);
JOIN8:    SKYRMIONJOIN port map (A => CROSS8_Bout, B => DEV1_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN8_out);
JOIN7:    SKYRMIONJOIN port map (A => DEV4_devout, B => JOIN8_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN7_out);

MTJ_W_3:  MTJ_W port map (CTRL => CTRL_MTJ_W3, OUT_SK => MTJ_W3_out);
JOIN1:    SKYRMIONJOIN port map (A => MTJ_W3_out, B => JOIN7_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN1_out);
x2_3:     skyrmionDUPLICATE port map (IN_SK => JOIN1_out, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_3_outtop, OUT_SK_BOTTOM => x2_3_outbottom);
CROSSM4:   SKYRMIONCROSS_Magn port map (A => CROSS5_Aout, B => x2_3_outtop, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM4_outA, Bout => CROSSM4_outB);
x2_7:     skyrmionDUPLICATE port map (IN_SK => CROSSM4_outA, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_7_outtop, OUT_SK_BOTTOM => x2_7_outbottom);
```

```

MTJ_CONV_10:  MTJ_CONV port map (IN_SK => x2_1_outbottom, CURRENT =>
↪  CURRENT_Vmove3, OUT_SK => MTJ_CONV10_out);
CROSS5:      SKYRMIONCROSS_noMagn port map (A => LEFT_COUT, B =>
↪  MTJ_CONV10_out, CURRENTA => LEFT_COUT_CURRENT, CURRENTB => CURRENT_Vmove3,
↪  Aout => CROSS5_Aout, Bout => CROSS5_Bout);
CROSS6:      SKYRMIONCROSS_noMagn port map (A => LEFT_RESULT, B => CROSS5_Bout,
↪  CURRENTA => LEFT_RESULT_CURRENT, CURRENTB => CURRENT_Vmove3, Aout =>
↪  CROSS6_Aout, Bout => CROSS6_Bout);
CROSS7:      SKYRMIONCROSS_noMagn port map (A => BOTLEFT_RESULT, B =>
↪  CROSS6_Bout, CURRENTA => BOTLEFT_RESULT_CURRENT, CURRENTB =>
↪  CURRENT_Vmove3, Aout => CROSS7_Aout, Bout => CROSS7_Bout);
CTRL_mux2 <= CTRL_Vrout;
Vmux2_Vsout:  voltage_genL port map (CTRL => CTRL_Vsout, CURRENT =>
↪  CURRENT_Vsout);
Vmux2_Vrout:  voltage_genL port map (CTRL => CTRL_Vrout, CURRENT =>
↪  CURRENT_Vrout);
MUX2_COM:     mux2 port map (inR => CROSS4_Bout, inL => CROSS7_Bout, CURRENT
↪  => CURRENT_Vmove3, CURRENT_Vst => CURRENT_Vsout, CURRENT_Vres =>
↪  CURRENT_Vrout, selection => CTRL_mux2, muxOUT => MUX2_out, VstOUT =>
↪  MUX2_Vstout, VresOUT => MUX2_Vresout);

Vmoveres:    voltage_genL port map (CTRL => CTRL_Vmoveres, CURRENT =>
↪  CURRENT_Vmoveres);
x2_5:       skyrmionDUPLICATE port map (IN_SK => MUX2_out, CURRENT =>
↪  CURRENT_Vmoveres, OUT_SK_TOP => x2_5_outright, OUT_SK_BOTTOM =>
↪  x2_5_outleft);
MTJ_CONV_12:  MTJ_CONV port map (IN_SK => x2_5_outleft, CURRENT =>
↪  CURRENT_Vmoveres, OUT_SK => MTJ_CONV12_out);
MTJ_CONV_11:  MTJ_CONV port map (IN_SK => x2_5_outright, CURRENT =>
↪  CURRENT_Vmoveres, OUT_SK => MTJ_CONV11_out);

Vtanktop:  vclock_gen port map (CTRL => CTRL_Vtanktop, CURRENTclk =>
↪  CURRENTclk, CURRENT => CURRENT_Vtanktop);
TANK_TOP_C:  tank_top port map (IN1 => MUX1_V2Tout, IN2 => MUX1_V3out, IN3 =>
↪  DEV2_devout, CURRENT => CURRENT_Vtanktop, tankOUT => TANKT_out);
MTJ_CONV_9:  MTJ_CONV port map (IN_SK => x2_4_outtop, CURRENT =>
↪  CURRENT_Vmove3, OUT_SK => MTJ_CONV9_out);
Vmove4:     voltage_genL port map (CTRL => CTRL_Vmove4, CURRENT =>
↪  CURRENT_Vmove4);
Vnewdata:  voltage_genL port map (CTRL => CTRL_Vnewdata, CURRENT =>
↪  CURRENT_Vnewdata);
DEV2:     skyrmionDEVIATION port map (IN_SK => MTJ_CONV9_out, CURRENT =>
↪  CURRENT_Vmove4, CURRENTDEV => CURRENT_Vnewdata, OUT_SK => DEV2_out,
↪  OUT_SK_DEV => DEV2_devout);
Vdetect:  voltage_genL port map (CTRL => CTRL_Vdetect, CURRENT =>
↪  CURRENT_Vdetect);
MTJ_R_2:  MTJ_R port map (IN_SK => TANKT_out, CURRENT => CURRENT_Vdetect,
↪  OUT_SIGN => MTJ_R2_out);
```

```
  RST_SR_MTJR2 <= ACK_DETECT_MTJ_R2 or (not RESET_n);
  SR_MTJR2:  SRlatch port map (SET => MTJ_R2_out, RST => RST_SR_MTJR2, Q =>
  ↪  SR_MTJR2_out);

  CELL_OUT_MTJ_CONV14_out   <= MTJ_CONV14_out;
  CELL_OUT_CROSS9_Bout     <= CROSS9_Bout;
  CELL_OUT_CROSS9_Aout     <= CROSS9_Aout;
  CELL_OUT_MTJ_CONV12_out   <= MTJ_CONV12_out;
  CELL_OUT_DEV2_out        <= DEV2_out;
  CELL_OUT_TANKT_out       <= TANKT_out;

  CELL_OUT_CURRENT_Vmoveres   <= CURRENT_Vmoveres;
  CELL_OUT_CURRENT_Vmovecout      <= CURRENT_Vmovecout;


end Structure;
```

C.4.2. FSM

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity FSM_CELL_OX is
port(  CURRENTclk: in real;
    RESET_n: in std_logic;

    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3,
    ↪  CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4,
    ↪  CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop,
    ↪  CTRL_Vouttank: out std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
    CTRL_mux1: out std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
    SR_MTJR2_out, SR_MTJR1_out: in std_logic;

    START, ACK_RES_AVAILABLE: in std_logic;
    DES_EXTIN_OX, DES_OUT_OX, DES_STORE_OX, DES_NEWDATA_OX: in std_logic;
    DES_RES_OX: in std_logic_vector(1 downto 0);
    RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE:
    ↪  out std_logic;
```

```
    READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic;

    CTRL_Vresleft, CTRL_Vresbotleft: out std_logic;
    READY_FOR_NEWDATA: out std_logic;
    DES_DATA_0X: in std_logic_vector(1 downto 0)
);
end entity FSM_CELL_0X;

architecture Behaviour of FSM_CELL_0X is

  type state_type is (
    reset, S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10,
    S8_1, S8_2, S8_3, S8_4, S8_5, S8_6,
    S11, S12, S13, S14, S15, S16, S17, S18, S19, S20,
    S21, S22, S23, S24, S25, S26, S27, S28, S29, S30,
    S31, S32, S33, S34, S35, S36, S37, S38, S39, S40,
    S41, S42, S43, S44
  );

  signal pstate, nstate: state_type;

begin

  state_register: process (CURRENTclk)
  begin
    if (CURRENTclk'event and CURRENTclk=CURRENT_HIGH) then
      if (RESET_n = '0') then
        pstate <= reset;
      else
        pstate <= nstate;
      end if;
    end if;
  end process state_register;

  state_transition: process (pstate, CURRENTclk)
  begin
    case pstate is
      when reset  => nstate <= S0;
      when S0    => if (START='0') then nstate <= S0; else if(FIRST_RUN='1') then nstate <= S1; else nstate <= S33; end if; end if;
      when S1    => if(DES_DATA_0X="01") then nstate <= S2; elsif (DES_DATA_0X="10")  then nstate <= S3; else nstate <= S4; end if;
      when S2    => nstate <= S5;
      when S3    => nstate <= S5;
      when S4    => if(DES_EXTIN_0X='1') then nstate <= S6; else nstate <= S7; end if;
      when S5    => nstate <= S8;
      when S6    => nstate <= S8;
      when S7    => nstate <= S8;
```

```
        when S8   => nstate <= S8_1;
        when S8_1   => nstate <= S8_2;
        when S8_2   => nstate <= S8_3;
        when S8_3   => nstate <= S8_4;
        when S8_4   => nstate <= S8_5;
        when S8_5   => nstate <= S8_6;
        when S8_6   => nstate <= S9;
        when S9   => nstate <= S10;
        when S10   => if (DES_RES_OX="10") then nstate <= S11; elsif (DES_RES_OX="01") then nstate <= S12; else nstate <= S13; end if;
        when S11   => nstate <= S14;
        when S12   => nstate <= S14;
        when S13   => nstate <= S14;
        when S14   => nstate <= S15;
        when S15   => if (DES_OUT_OX='1') then nstate <= S16; else nstate <= S17; end if;
        when S16   => nstate <= S18;
        when S17   => nstate <= S18;
        when S18   => nstate <= S19;
        when S19   => if(DES_STORE_OX='1') then nstate <= S20; else nstate <= S21; end if;
        when S20   => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S20; else nstate <= S22; end if;
        when S21   => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S21; else if(DES_NEWDATA_OX='1') then nstate <= S23; else nstate <= S24; end if; end if;
        when S22   => nstate <= S26;
        when S23   => nstate <= S25;
        when S24  => nstate <= S28;
        when S25  => nstate <= S27;
        when S26   => if (ACK_RES_AVAILABLE='0') then nstate <= S26; else nstate <= S30; end if;
        when S27   => if (ACK_RES_AVAILABLE='0') then nstate <= S27; else nstate <= S29; end if;
        when S28   => if (ACK_RES_AVAILABLE='0') then nstate <= S28; else nstate <= S32; end if;
        when S29  => if (SR_MTJR2_out='0') then nstate <= S31; else nstate <= S32; end if;
        when S30   => if(START='0') then nstate <= S30; else nstate <= S33; end if;
        when S31   => if(START='0') then nstate <= S31; else nstate <= S33; end if;
        when S32   => if(START='0') then nstate <= S32; else nstate <= S33; end if;
        when S33   => nstate <= S34;
        when S34   => if (DES_DATA_OX="01") then nstate <= S35; elsif (DES_DATA_OX="10") then nstate <= S36; else nstate <= S37; end if;
        when S35   => nstate <= S38;
        when S36   => nstate <= S38;
```

```
      when S37   => if (DES_EXTIN_OX='0') then nstate <= S39; else nstate <= S40; end if;
      when S38   => nstate <= S39;
      when S39   => if (SR_MTJR1_out='0') then nstate <= S43; else nstate <= S41; end if;
      when S40   => if (SR_MTJR1_out='1') then nstate <= S44; else nstate <= S42; end if;
      when S41   => nstate <= S43;
      when S42   => nstate <= S44;
      when S43   => nstate <= S8;
      when S44   => nstate <= S8;
      when others => nstate <= S0;
    end case;
  end process state_transition;

  output: process (pstate)
  begin

    CTRL_Vstart <= '0';
    CTRL_Vmove1 <= '0';
    CTRL_Vop <= '0';
    CTRL_Vmove2 <= '0';
    CTRL_Vmove2C <= '0';
    CTRL_Vmove3 <= '0';
    CTRL_Vmovecout <= '0';
    CTRL_Vtankbot <= '0';
    CTRL_Vnoext <= '0';
    CTRL_Vmoveres <= '0';
    CTRL_Vmove4 <= '0';
    CTRL_Vnewdata <= '0';
    CTRL_Vdetect <= '0';
    CTRL_Vrout <= '0';
    CTRL_Vsout <= '0';
    CTRL_Vtanktop <= '0';
    CTRL_Vouttank <= '0';
    CTRL_MTJ_W1 <= '0';
    CTRL_MTJ_W2 <= '0';
    CTRL_MTJ_W3 <= '0';
    CTRL_mux1 <= "00";
    ACK_DETECT_MTJ_R1 <= '0';
    ACK_DETECT_MTJ_R2 <= '0';
    RES_AVAILABLE <= '0';
    REQUEST_NEW_START <= '0';
    REQUEST_NEW_START_W <= '0';
    REQUEST_NEW_STORE <= '0';
    ---
    CTRL_Vresleft <= '0';
    CTRL_Vresbotleft <= '0';
    READY_FOR_NEWDATA <= '0';
```

```
    case pstate is
      when S0     =>  CTRL_Vstart <= '0';
              CTRL_Vmove1 <= '0';
              CTRL_Vop <= '0';
              CTRL_Vmove2 <= '0';
              CTRL_Vmove2C <= '0';
              CTRL_Vmove3 <= '0';
              CTRL_Vmovecout <= '0';
              CTRL_Vtankbot <= '0';
              CTRL_Vnoext <= '0';
              CTRL_Vmoveres <= '0';
              CTRL_Vmove4 <= '0';
              CTRL_Vnewdata <= '0';
              CTRL_Vdetect <= '0';
              CTRL_Vrout <= '0';
              CTRL_Vsout <= '0';
              CTRL_Vtanktop <= '0';
              CTRL_Vouttank <= '0';
              CTRL_MTJ_W1 <= '0';
              CTRL_MTJ_W2 <= '0';
              CTRL_MTJ_W3 <= '0';
              CTRL_mux1 <= "00";
              ACK_DETECT_MTJ_R1 <= '0';
              ACK_DETECT_MTJ_R2 <= '0';
              RES_AVAILABLE <= '0';
              REQUEST_NEW_START <= '0';
              REQUEST_NEW_START_W <= '0';
              REQUEST_NEW_STORE <= '0';
              ---
              CTRL_Vresleft <= '0';
              CTRL_Vresbotleft <= '0';
              READY_FOR_NEWDATA <= '0';
      when S1     =>  CTRL_Vnoext <= '1', '0' after CLOCK_PERIOD/2;
              READY_FOR_NEWDATA <= '1';
              CTRL_Vstart <= '1';
      when S2     =>  CTRL_Vresleft <= '1';
      when S3     =>  CTRL_Vresbotleft <= '1';
      when S4     =>  CTRL_Vnoext <= '1';
      when S5     =>  CTRL_Vnoext <= '1';
              CTRL_Vmove1 <= '1';
              CTRL_MTJ_W1 <= '1';
              CTRL_MTJ_W2 <= '1';
      when S6     =>  CTRL_Vmove1 <= '1';
              CTRL_MTJ_W1 <= '1';
              CTRL_MTJ_W2 <= '1';
              CTRL_MTJ_W3 <= '1';
      when S7     =>  CTRL_Vmove1 <= '1';
              CTRL_MTJ_W1 <= '1';
```

```
              CTRL_MTJ_W2 <= '1';
      when S8    =>  CTRL_Vop <= '1';
      when S8_1  =>  CTRL_Vop <= '1';
      when S8_2  =>  CTRL_Vop <= '1';
      when S8_3  =>  CTRL_Vop <= '1';
      when S8_4  =>  CTRL_Vop <= '1';
      when S8_5  =>  CTRL_Vop <= '1';
      when S8_6  =>  CTRL_Vop <= '1';
      when S9    =>  CTRL_Vmove2 <= '1';
              CTRL_Vmove2C <= '1';
      when S10  =>  CTRL_Vmove2C <= '1';
      when S11  =>  CTRL_mux1 <= "10";
      when S12  =>  CTRL_mux1 <= "01";
      when S13  =>  CTRL_mux1 <= "11";
      when S14  =>  CTRL_Vmove2 <= '1';
      when S15  =>  CTRL_Vmove3 <= '1';
      when S16  =>  CTRL_Vrout <= '1';
      when S17  =>  CTRL_Vsout <= '1';
      when S18  =>  CTRL_Vmove3 <= '1';
      when S19  =>  CTRL_Vmove4 <= '1', '0' after CLOCK_PERIOD/2;
      when S20  =>  CTRL_Vmove4 <= '1';
      when S21  =>  CTRL_Vnewdata <= '1';
      when S22  =>  CTRL_Vmoveres <= '1';
              CTRL_Vmovecout <= '1';
      when S23  =>  CTRL_Vtanktop <= '1';
              CTRL_Vmoveres <= '1';
              CTRL_Vmovecout <= '1';
      when S24  =>  CTRL_Vmoveres <= '1';
              CTRL_Vmovecout <= '1';
      when S25  =>  CTRL_Vtanktop <= '1';
      when S26  =>  RES_AVAILABLE <= '1';
      when S27  =>  RES_AVAILABLE <= '1';
      when S28  =>  RES_AVAILABLE <= '1';
      when S29  =>  CTRL_Vdetect <= '1';
      when S30  =>  REQUEST_NEW_STORE <= '1';
      when S31  =>  REQUEST_NEW_START_W <= '1';
              ACK_DETECT_MTJ_R2 <= '1';
      when S32  =>  REQUEST_NEW_START <= '1';
              ACK_DETECT_MTJ_R2 <= '1';
      when S33  =>  CTRL_Vstart <= '1';
              CTRL_Vtankbot <= '1';
      when S34  =>  CTRL_Vtankbot <= '1';
              CTRL_Vnoext <= '1', '0' after CLOCK_PERIOD/2;
              READY_FOR_NEWDATA <= '1';
              CTRL_Vouttank <= '1';
      when S35  =>  CTRL_Vresleft <= '1';
      when S36  =>  CTRL_Vresbotleft <= '1';
      when S37  =>  CTRL_Vnoext <= '1';
      when S38  =>  CTRL_Vnoext <= '1';
```

```
        when S39  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
        when S40  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
        when S41  =>  CTRL_Vnoext <= '1';
                ACK_DETECT_MTJ_R1 <= '1';
        when S42  =>  CTRL_MTJ_W3 <= '1';
        when S43  =>  CTRL_Vmove1 <= '1';
        when S44  =>  CTRL_Vmove1 <= '1';
                ACK_DETECT_MTJ_R1 <= '1';

        when others =>  CTRL_Vstart <= '0';
                CTRL_Vmove1 <= '0';
                CTRL_Vop <= '0';
                CTRL_Vmove2 <= '0';
                CTRL_Vmove2C <= '0';
                CTRL_Vmove3 <= '0';
                CTRL_Vmovecout <= '0';
                CTRL_Vtankbot <= '0';
                CTRL_Vnoext <= '0';
                CTRL_Vmoveres <= '0';
                CTRL_Vmove4 <= '0';
                CTRL_Vnewdata <= '0';
                CTRL_Vdetect <= '0';
                CTRL_Vrout <= '0';
                CTRL_Vsout <= '0';
                CTRL_Vtanktop <= '0';
                CTRL_Vouttank <= '0';
                CTRL_MTJ_W1 <= '0';
                CTRL_MTJ_W2 <= '0';
                CTRL_MTJ_W3 <= '0';
                CTRL_mux1 <= "00";
                ACK_DETECT_MTJ_R1 <= '0';
                ACK_DETECT_MTJ_R2 <= '0';
                RES_AVAILABLE <= '0';
                REQUEST_NEW_START <= '0';
                REQUEST_NEW_START_W <= '0';
                REQUEST_NEW_STORE <= '0';
                ---
                CTRL_Vresleft <= '0';
                CTRL_Vresbotleft <= '0';
                READY_FOR_NEWDATA <= '0';
      end case;
    end process output;
end Behaviour;
```

C.5. Cell11

C.5.1. Datapath

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;

entity CELL_XX is
port(  IN_CELL, RESET_n: in std_logic;
    CURRENTclk: in real;

    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3,
    ↪ CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4,
    ↪ CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop,
    ↪ CTRL_Vouttank: in std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
    CTRL_mux1: in std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
    SR_MTJR1_out, SR_MTJR2_out: out std_logic;
    CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

    CELL_OUT_MTJ_CONV14_out, CELL_OUT_MTJ_CONV15_out, CELL_OUT_MTJ_CONV11_out,
    ↪ CELL_OUT_MTJ_CONV13_out, CELL_OUT_MTJ_CONV12_out, CELL_OUT_DEV2_out,
    ↪ CELL_OUT_TANKT_out: out std_logic;

    CTRL_Vcoutleft, CTRL_Vcouttop, CTRL_Vrestop, CTRL_Vresbotleft,
    ↪ CTRL_Vresleft: in std_logic;
    TOP_COUT, LEFT_COUT, TOP_RESULT, LEFT_RESULT, BOTLEFT_RESULT:  in std_logic;
    TOP_COUT_CURRENT, LEFT_COUT_CURRENT, TOP_RESULT_CURRENT,
    ↪ LEFT_RESULT_CURRENT, BOTLEFT_RESULT_CURRENT: in real
);
end entity CELL_XX;

architecture Structure of CELL_XX is
  component SKYRMIONNOTCH is
  port(  INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  component SKYRMIONNOTCHseq is
  generic (N: integer := 5);
  port(   INPUT: in std_logic;
```

```
        CURRENT: in real;
        OUTPUT: out std_logic);
    end component SKYRMIONNOTCHseq;

    component SKYRMIONJOIN is
    port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic);
    end component SKYRMIONJOIN;

    component SKYRMIONH is
    port (  INPUTA : in std_logic;
        INPUTB : in std_logic;
        CURRENT : in real;
        OUTPUTAND : out std_logic;
        OUTPUTOR : out std_logic);
    end component SKYRMIONH;

    component SKYRMIONFULLADDER is
    port(   A     : in std_logic;
        B     : in std_logic;
        CIN1  : in std_logic;
        CIN2  : in std_logic;
        ONE1  : in std_logic;
        ONE2  : in std_logic;
        CURRENT  : in real;
        CTRL1  : out std_logic;
        SUM    : out std_logic;
        COUT1  : out std_logic;
        COUT2  : out std_logic;
        CTRL2  : out std_logic);
    end component SKYRMIONFULLADDER;

    component voltage_genH is
    port(  CTRL: in std_logic;
        CURRENT: out real);
    end component voltage_genH;

    component voltage_genL is
    port(  CTRL: in std_logic;
        CURRENT: out real);
    end component voltage_genL;

    component vclock_gen is
    port(  CTRL: in std_logic;
        CURRENTclk: in real;
        CURRENT:  out real);
    end component vclock_gen;
```

```

component skyrmionDUPLICATE is
port(  IN_SK:       in std_logic;
    CURRENT:    in real;
    OUT_SK_TOP:    out std_logic;
    OUT_SK_BOTTOM:  out std_logic);
end component skyrmionDUPLICATE;

component skyrmionMERGE is
port(  IN_SK_TOP:     in std_logic;
    IN_SK_BOTTOM:  in std_logic;
    CURRENT:    in real;
    OUT_SK:      out std_logic);
end component skyrmionMERGE;

component SKYRMIONCROSS_Magn is
port(   A:       in std_logic;
    B:       in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end component SKYRMIONCROSS_Magn;

component SKYRMIONCROSS_noMagn is
port(   A:       in std_logic;
    B:       in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end component SKYRMIONCROSS_noMagn;

component skyrmionDEVIATION is
port(  IN_SK:     in std_logic;
    CURRENT:  in real;
    CURRENTDEV:  in real;
    OUT_SK:    out std_logic;
    OUT_SK_DEV:  out std_logic);
end component skyrmionDEVIATION;

component SRlatch is
port(  SET:   in std_logic;
    RST:   in std_logic;
    Q:    buffer std_logic);
end component SRlatch;

component MTJ_R is
port(  IN_SK:     in std_logic;
```

```
        CURRENT:  in real;
        OUT_SIGN:  out std_logic);
    end component MTJ_R;

    component MTJ_W is
    port(  CTRL: in std_logic;
        OUT_SK:  out std_logic);
    end component MTJ_W;

    component MTJ_CONV is
    port(  IN_SK: in std_logic;
        CURRENT: in real;
        OUT_SK:  out std_logic);
    end component MTJ_CONV;

    component tank_bottom is
    port(  IN1, IN2, IN3, IN4, IN5, IN6, IN7: in std_logic;  --from left to right
        CURRENT: in real;
        tankOUT1, tankOUT2, tankOUT3: out std_logic);    --priority order: from
        ↪  right to left
    end component tank_bottom;

    component tank_top is
    port(  IN1, IN2, IN3: in std_logic; --from right to left
        CURRENT: in real;
        tankOUT: out std_logic);
    end component tank_top;

    component tank_topXX is
    port(  IN1, IN2, IN3, IN4, IN5: in std_logic; --from right to left
        CURRENT: in real;
        tankOUT: out std_logic);
    end component tank_topXX;

    component mux1 is
    port(  inT, inM, inB: in std_logic;
        CURRENT: in real;
        CURRENT_V1, CURRENT_V2, CURRENT_V3: in real;
        selection: in std_logic_vector(1 downto 0);
        muxOUT, V1OUT, V3OUT, V2OUTt, V2OUTb: out std_logic);
    end component mux1;

    component mux2 is
    port(  inR, inL: in std_logic;    --Right=result, Left=stored value
        CURRENT: in real;
        CURRENT_Vst, CURRENT_Vres: in real;
        selection: in std_logic;  --'1'=result, '0'=stored value
        muxOUT, VstOUT, VresOUT: out std_logic);
    end component mux2;
```

```

  signal CURRENT_Vstart, CURRENT_Vmove1, CURRENT_Vop, CURRENT_Vmove2,
  ↪  CURRENT_Vmove2C, CURRENT_Vmove3, CURRENT_MUX1_V1, CURRENT_MUX1_V2,
  ↪  CURRENT_MUX1_V3, CURRENT_Vmovecout, CURRENT_Vtankbot, CURRENT_Vnoext,
  ↪  CURRENT_Vmoveres, CURRENT_Vtanktop, CURRENT_Vouttank, CURRENT_Vmove4,
  ↪  CURRENT_Vnewdata, CURRENT_Vdetect, CURRENT_Vrout, CURRENT_Vsout,
  ↪  CURRENT_Vrestop, CURRENT_Vresbotleft, CURRENT_Vresleft,
  ↪  CURRENT_Vcoutleft,CURRENT_Vcouttop: real;

  signal MEM_out: std_logic;
  signal x2_1_outtop, x2_1_outbottom, x2_2_outtop, x2_2_outbottom, x2_3_outtop,
  ↪  x2_3_outbottom, x2_4_outtop, x2_4_outbottom, x2_5_outright, x2_5_outleft,
  ↪  x2_6_outtop, x2_6_outbottom, x2_7_outtop, x2_7_outbottom, x2_8_outtop,
  ↪  x2_8_outbottom, x1_1_out: std_logic;
  signal CROSSM1_outA, CROSSM1_outB, CROSSM2_outA, CROSSM2_outB, CROSSM3_outA,
  ↪  CROSSM3_outB, CROSSM4_outA, CROSSM4_outB: std_logic;
  signal CROSS11_Aout, CROSS11_Bout, CROSS12_Aout, CROSS12_Bout, CROSS21_Aout,
  ↪  CROSS21_Bout, CROSS22_Aout, CROSS22_Bout, CROSS3_Aout, CROSS3_Bout,
  ↪  CROSS4_Aout, CROSS4_Bout, CROSS6_Aout, CROSS6_Bout, CROSS7_Aout,
  ↪  CROSS7_Bout, CROSS8_Aout, CROSS8_Bout, CROSS9_Aout, CROSS9_Bout,
  ↪  CROSS10_Aout, CROSS10_Bout, CROSS13_Aout, CROSS13_Bout, CROSS14_Aout,
  ↪  CROSS14_Bout, CROSS15_Aout, CROSS15_Bout, CROSS16_Aout, CROSS16_Bout,
  ↪  CROSS17_Aout, CROSS17_Bout: std_logic;
  signal SYNC_LOG_TOP_out, SYNC_LOG_BOT_out, AND_out, OR_out, SKEW_OR_out,
  ↪  SKEW_AND_out: std_logic;
  signal FA_CTRL1_out, FA_CTRL2_out, FA_COUT1_out, FA_COUT2_out, FA_SUM_out:
  ↪  std_logic;
  signal RST_SR_MTJR1, RST_SR_MTJR2: std_logic;
  signal MTJ_R1_out, MTJ_R2_out: std_logic;
  signal MTJ_W1_out, MTJ_W2_out, MTJ_W3_out: std_logic;
  signal MTJ_CONV1_out, MTJ_CONV2_out, MTJ_CONV3_out, MTJ_CONV4_out,
  ↪  MTJ_CONV5_out, MTJ_CONV6_out, MTJ_CONV7_out, MTJ_CONV8_out, MTJ_CONV9_out,
  ↪  MTJ_CONV10_out, MTJ_CONV11_out, MTJ_CONV12_out, MTJ_CONV13_out,
  ↪  MTJ_CONV14_out, MTJ_CONV15_out, MTJ_CONV16_out: std_logic;
  signal MUX1_CTRL_V1, MUX1_CTRL_V2, MUX1_CTRL_V3: std_logic;
  signal MUX1_out, MUX1_V1out, MUX1_V3out, MUX1_V2Tout, MUX1_V2Bout: std_logic;
  signal CTRL_mux2, MUX2_Vstout, MUX2_Vresout, MUX2_out: std_logic;
  signal JOIN1_out, JOIN2_out, JOIN3_out, JOIN5_out, JOIN6_out, JOIN7_out,
  ↪  JOIN8_out, JOIN9_out, JOIN10_out, JOIN11_out: std_logic;
  signal DEV1_out, DEV1_devout, DEV2_out, DEV2_devout, DEV3_out, DEV3_devout,
  ↪  DEV4_out, DEV4_devout, DEV5_out, DEV5_devout, DEV6_out, DEV6_devout,
  ↪  DEV7_out, DEV7_devout: std_logic;
  signal TANKB_out1, TANKB_out2, TANKB_out3: std_logic;
  signal TANKT_out: std_logic;


begin
  Vstart:      voltage_genH port map (CTRL => CTRL_Vstart, CURRENT =>
  ↪  CURRENT_Vstart);
```

```
MEM:       SKYRMIONNOTCH port map (INPUT => IN_CELL, CURRENT => CURRENT_Vstart, OUTPUT => MEM_out);
Vmove1:      voltage_genL port map (CTRL => CTRL_Vmove1, CURRENT => CURRENT_Vmove1);
x2_1:      skyrmionDUPLICATE port map (IN_SK => MEM_out, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_1_outtop, OUT_SK_BOTTOM => x2_1_outbottom);
MTJ_CONV_16:  MTJ_CONV port map (IN_SK => x2_1_outtop, CURRENT => CURRENT_Vmove1, OUT_SK => MTJ_CONV16_out);
Vcoutleft:    voltage_genL port map (CTRL => CTRL_Vcoutleft, CURRENT => CURRENT_Vcoutleft);
Vcouttop:    voltage_genL port map (CTRL => CTRL_Vcouttop, CURRENT => CURRENT_Vcouttop);
CROSS14:     SKYRMIONCROSS_noMagn port map (A => MTJ_CONV16_out, B => DEV6_devout, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vcoutleft, Aout => CROSS14_Aout, Bout => CROSS14_Bout);
CROSS15:     SKYRMIONCROSS_noMagn port map (A => CROSS14_Aout, B => CROSS13_Bout, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vcouttop, Aout => CROSS15_Aout, Bout => CROSS15_Bout);
x2_2:      skyrmionDUPLICATE port map (IN_SK => CROSS15_Aout, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_2_outtop, OUT_SK_BOTTOM => x2_2_outbottom);

Vmove3:      voltage_genL port map (CTRL => CTRL_Vmove3, CURRENT => CURRENT_Vmove3);
MTJ_CONV_10:  MTJ_CONV port map (IN_SK => x2_1_outbottom, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV10_out);
CROSS16:     SKYRMIONCROSS_noMagn port map (A => TOP_COUT, B => MTJ_CONV10_out, CURRENTA => TOP_COUT_CURRENT, CURRENTB => CURRENT_Vmove3, Aout => CROSS16_Aout, Bout => CROSS16_Bout);
CROSS17:     SKYRMIONCROSS_noMagn port map (A => LEFT_COUT, B => CROSS16_Bout, CURRENTA => LEFT_COUT_CURRENT, CURRENTB => CURRENT_Vmove3, Aout => CROSS17_Aout, Bout => CROSS17_Bout);

Vnoext:    voltage_genL port map (CTRL => CTRL_Vnoext, CURRENT => CURRENT_Vnoext);
DEV6:    skyrmionDEVIATION port map (IN_SK => CROSS16_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vcoutleft, OUT_SK => DEV6_out, OUT_SK_DEV => DEV6_devout);
DEV7:    skyrmionDEVIATION port map (IN_SK => CROSS17_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vcouttop, OUT_SK => DEV7_out, OUT_SK_DEV => DEV7_devout);
CROSS13:   SKYRMIONCROSS_noMagn port map (A => DEV6_out, B => DEV7_devout, CURRENTA => CURRENT_Vnoext, CURRENTB => CURRENT_Vcouttop, Aout => CROSS13_Aout, Bout => CROSS13_Bout);
JOIN11:    SKYRMIONJOIN port map (A => CROSS13_Aout, B => DEV7_out, CURRENT => CURRENT_Vnoext, OUTPUT => JOIN11_out);

CROSSM1:      SKYRMIONCROSS_Magn port map (A => x2_2_outbottom, B => CROSSM4_outB, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM1_outA, Bout => CROSSM1_outB);
```

```
Vop:        vclock_gen port map (CTRL => CTRL_Vop, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vop);
SYNCNET_LOG_TOP:  SKYRMIONNOTCH port map (INPUT => x2_2_outtop, CURRENT => CURRENT_Vop, OUTPUT => SYNC_LOG_TOP_out);
SYNCNET_LOG_BOT:  SKYRMIONNOTCH port map (INPUT => CROSSM1_outB, CURRENT => CURRENT_Vop, OUTPUT => SYNC_LOG_BOT_out);
LOGIC:        SKYRMIONH port map (INPUTA => SYNC_LOG_TOP_out, INPUTB => SYNC_LOG_BOT_out, CURRENT => CURRENT_Vop, OUTPUTAND => AND_OUT, OUTPUTOR => OR_OUT);
SKEW_OR:      SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => OR_OUT, CURRENT => CURRENT_Vop, OUTPUT => SKEW_OR_out);
SKEW_AND:      SKYRMIONNOTCHseq generic map (N => 5) port map (INPUT => AND_OUT, CURRENT => CURRENT_Vop, OUTPUT => SKEW_AND_out);
FA:           SKYRMIONFULLADDER port map (A => CROSSM1_outA, B => x2_3_outbottom, CIN1 => x2_7_outtop, CIN2 => x2_7_outbottom, ONE1 => CROSSM2_outB, ONE2 => CROSSM3_outB, CURRENT => CURRENT_Vop, CTRL1 => FA_CTRL1_out, SUM => FA_SUM_out, COUT1 => FA_COUT1_out, COUT2 => FA_COUT2_out, CTRL2 => FA_CTRL2_out);

MTJ_CONV_1:  MTJ_CONV port map (IN_SK => SKEW_OR_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV1_out);
MTJ_CONV_2:  MTJ_CONV port map (IN_SK => SKEW_AND_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV2_out);
MTJ_CONV_3:  MTJ_CONV port map (IN_SK => FA_SUM_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV3_out);
MTJ_CONV_4:  MTJ_CONV port map (IN_SK => FA_COUT1_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV4_out);
MTJ_CONV_5:  MTJ_CONV port map (IN_SK => FA_COUT2_out, CURRENT => CURRENT_Vop, OUT_SK => MTJ_CONV5_out);
Vmove2:    voltage_genL port map (CTRL => CTRL_Vmove2, CURRENT => CURRENT_Vmove2);
Vmove2C:  voltage_genL port map (CTRL => CTRL_Vmove2C, CURRENT => CURRENT_Vmove2C);

MUX1_CTRL_V1 <= CTRL_mux1(0) and (not CTRL_mux1(1));
MUX1_CTRL_V2 <= CTRL_mux1(1) and (not CTRL_mux1(0));
MUX1_CTRL_V3 <= CTRL_mux1(1) and CTRL_mux1(0);
Vmux1_1:  voltage_genL port map (CTRL => MUX1_CTRL_V1, CURRENT => CURRENT_MUX1_V1);
Vmux1_2:  voltage_genL port map (CTRL => MUX1_CTRL_V2, CURRENT => CURRENT_MUX1_V2);
Vmux1_3:  voltage_genL port map (CTRL => MUX1_CTRL_V3, CURRENT => CURRENT_MUX1_V3);

MUX1_COM:     mux1 port map (inT => MTJ_CONV1_out, inM => MTJ_CONV2_out, inB => MTJ_CONV3_out, CURRENT => CURRENT_Vmove2, CURRENT_V1 => CURRENT_MUX1_V1, CURRENT_V2 => CURRENT_MUX1_V2, CURRENT_V3 => CURRENT_MUX1_V3, selection => CTRL_mux1, muxOUT => MUX1_out, V1OUT => MUX1_V1out, V3OUT => MUX1_V3out, V2OUTt => MUX1_V2Tout, V2OUTb => MUX1_V2Bout);
```

```
x2_4:    skyrmionDUPLICATE port map (IN_SK => MUX1_out, CURRENT => CURRENT_Vmove3, OUT_SK_TOP => x2_4_outtop, OUT_SK_BOTTOM => x2_4_outbottom);
MTJ_CONV_7:  MTJ_CONV port map (IN_SK => x2_4_outbottom, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV7_out);

CROSS11:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV4_out, B => MUX1_V1out, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout => CROSS11_Aout, Bout => CROSS11_Bout);
CROSS12:  SKYRMIONCROSS_noMagn port map (A => CROSS11_Aout, B => MUX1_V2Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS12_Aout, Bout => CROSS12_Bout);
CROSS21:  SKYRMIONCROSS_noMagn port map (A => MTJ_CONV5_out, B => CROSS11_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V1, Aout => CROSS21_Aout, Bout => CROSS21_Bout);
CROSS22:  SKYRMIONCROSS_noMagn port map (A => CROSS21_Aout, B => CROSS12_Bout, CURRENTA => CURRENT_Vmove2C, CURRENTB => CURRENT_MUX1_V2, Aout => CROSS22_Aout, Bout => CROSS22_Bout);

Vmovecout:  voltage_genL port map (CTRL => CTRL_Vmovecout, CURRENT => CURRENT_Vmovecout);
x1_1:    skyrmionMERGE port map (IN_SK_TOP => CROSS12_Aout, IN_SK_BOTTOM => CROSS22_Aout, CURRENT => CURRENT_Vmovecout, OUT_SK => x1_1_out);
MTJ_CONV_6:  MTJ_CONV port map (IN_SK => x1_1_out, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV6_out);
CROSS4:   SKYRMIONCROSS_noMagn port map (A => MTJ_CONV6_out, B => MTJ_CONV7_out, CURRENTA => CURRENT_Vmovecout, CURRENTB => CURRENT_Vmove3, Aout => CROSS4_Aout, Bout => CROSS4_Bout);

x2_8:      skyrmionDUPLICATE port map (IN_SK => CROSS4_Aout, CURRENT => CURRENT_Vmovecout, OUT_SK_TOP => x2_8_outtop, OUT_SK_BOTTOM => x2_8_outbottom);
MTJ_CONV_14:  MTJ_CONV port map (IN_SK => x2_8_outtop, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV14_out);
MTJ_CONV_15:  MTJ_CONV port map (IN_SK => x2_8_outbottom, CURRENT => CURRENT_Vmovecout, OUT_SK => MTJ_CONV15_out);
--CROSS9:     SKYRMIONCROSS_noMagn port map (A => MTJ_CONV11_out, B => MTJ_CONV15_out, CURRENTA => CURRENT_Vmoveres, CURRENTB => CURRENT_Vmovecout, Aout => CROSS9_Aout, Bout => CROSS9_Bout);

Vtankbot:  vclock_gen port map (CTRL => CTRL_Vtankbot, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vtankbot);
TANK_BOT:  tank_bottom port map (IN1 => FA_CTRL2_out, IN2 => FA_CTRL1_out, IN3 => CROSS21_Bout, IN4 => CROSS22_Bout, IN5 => JOIN10_out, IN6 => MUX2_Vresout, IN7 => MUX2_Vstout, CURRENT => CURRENT_Vtankbot, tankOUT1 => TANKB_out1, tankOUT2 => TANKB_out2, tankOUT3 => TANKB_out3);

MTJ_W_1:  MTJ_W port map (CTRL => CTRL_MTJ_W1, OUT_SK => MTJ_W1_out);
MTJ_W_2:  MTJ_W port map (CTRL => CTRL_MTJ_W2, OUT_SK => MTJ_W2_out);
```

```
JOIN2:    SKYRMIONJOIN port map (A => MTJ_W1_out, B => TANKB_out2, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN2_out);
JOIN3:    SKYRMIONJOIN port map (A => MTJ_W2_out, B => TANKB_out1, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN3_out);
Vouttank:  voltage_genL port map (CTRL => CTRL_Vouttank, CURRENT => CURRENT_Vouttank);
CROSSM2:   SKYRMIONCROSS_Magn port map (A => TANKB_out1, B => JOIN2_out, CURRENTA => CURRENT_Vouttank, CURRENTB => CURRENT_Vmove1, Aout => CROSSM2_outA, Bout => CROSSM2_outB);
CROSSM3:   SKYRMIONCROSS_Magn port map (A => CROSSM2_outA, B => JOIN3_out, CURRENTA => CURRENT_Vouttank, CURRENTB => CURRENT_Vmove1, Aout => CROSSM3_outA, Bout => CROSSM3_outB);

MTJ_R_1:  MTJ_R port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vouttank, OUT_SIGN => MTJ_R1_out);
RST_SR_MTJR1 <= ACK_DETECT_MTJ_R1 or (not(RESET_n));
SR_MTJR1:  SRlatch port map (SET => MTJ_R1_out, RST => RST_SR_MTJR1, Q => SR_MTJR1_out);
MTJ_CONV_8:  MTJ_CONV port map (IN_SK => CROSSM3_outA, CURRENT => CURRENT_Vouttank, OUT_SK => MTJ_CONV8_out);
DEV1:    skyrmionDEVIATION port map (IN_SK => MTJ_CONV8_out, CURRENT => CURRENT_Vmove1, CURRENTDEV => CURRENT_Vnoext, OUT_SK => DEV1_out, OUT_SK_DEV => DEV1_devout);
JOIN5:    SKYRMIONJOIN port map (A => DEV1_devout, B => CROSS9_Aout, CURRENT => CURRENT_Vnoext, OUTPUT => JOIN5_out);
JOIN9:    SKYRMIONJOIN port map (A => JOIN5_out, B => CROSS8_Aout, CURRENT => CURRENT_Vnoext, OUTPUT => JOIN9_out);
JOIN10:    SKYRMIONJOIN port map (A => JOIN9_out, B => DEV5_out, CURRENT => CURRENT_Vnoext, OUTPUT => JOIN10_out);

Vrestop:    voltage_genL port map (CTRL => CTRL_Vrestop, CURRENT => CURRENT_Vrestop);
Vresleft:    voltage_genL port map (CTRL => CTRL_Vresleft, CURRENT => CURRENT_Vresleft);
Vresbotleft:  voltage_genL port map (CTRL => CTRL_Vresbotleft, CURRENT => CURRENT_Vresbotleft);
CROSS3:      SKYRMIONCROSS_noMagn port map (A => TOP_RESULT, B => CROSS17_Bout, CURRENTA => TOP_RESULT_CURRENT, CURRENTB => CURRENT_Vmove3, Aout => CROSS3_Aout, Bout => CROSS3_Bout);
CROSS6:     SKYRMIONCROSS_noMagn port map (A => LEFT_RESULT, B => CROSS3_Bout, CURRENTA => LEFT_RESULT_CURRENT, CURRENTB => CURRENT_Vmove3, Aout => CROSS6_Aout, Bout => CROSS6_Bout);
CROSS7:     SKYRMIONCROSS_noMagn port map (A => BOTLEFT_RESULT, B => CROSS6_Bout, CURRENTA => BOTLEFT_RESULT_CURRENT, CURRENTB => CURRENT_Vmove3, Aout => CROSS7_Aout, Bout => CROSS7_Bout);

DEV3:    skyrmionDEVIATION port map (IN_SK => CROSS3_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vrestop, OUT_SK => DEV3_out, OUT_SK_DEV => DEV3_devout);
```

```
DEV4:    skyrmionDEVIATION port map (IN_SK => CROSS6_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vresleft, OUT_SK => DEV4_out, OUT_SK_DEV => DEV4_devout);
DEV5:    skyrmionDEVIATION port map (IN_SK => CROSS7_Aout, CURRENT => CURRENT_Vnoext, CURRENTDEV => CURRENT_Vresbotleft, OUT_SK => DEV5_out, OUT_SK_DEV => DEV5_devout);
CROSS8:   SKYRMIONCROSS_noMagn port map (A => DEV4_out, B => DEV5_devout, CURRENTA => CURRENT_Vnoext, CURRENTB => CURRENT_Vresbotleft, Aout => CROSS8_Aout, Bout => CROSS8_Bout);
CROSS10:   SKYRMIONCROSS_noMagn port map (A => DEV3_out, B => DEV4_devout, CURRENTA => CURRENT_Vnoext, CURRENTB => CURRENT_Vresleft, Aout => CROSS10_Aout, Bout => CROSS10_Bout);
CROSS9:   SKYRMIONCROSS_noMagn port map (A => CROSS10_Aout, B => CROSS8_Bout, CURRENTA => CURRENT_Vnoext, CURRENTB => CURRENT_Vresbotleft, Aout => CROSS9_Aout, Bout => CROSS9_Bout);

JOIN8:    SKYRMIONJOIN port map (A => CROSS9_Bout, B => DEV1_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN8_out);
JOIN7:    SKYRMIONJOIN port map (A => CROSS10_Bout, B => JOIN8_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN7_out);
JOIN6:    SKYRMIONJOIN port map (A => DEV3_devout, B => JOIN7_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN6_out);
MTJ_W_3:  MTJ_W port map (CTRL => CTRL_MTJ_W3, OUT_SK => MTJ_W3_out);
JOIN1:    SKYRMIONJOIN port map (A => MTJ_W3_out, B => JOIN6_out, CURRENT => CURRENT_Vmove1, OUTPUT => JOIN1_out);
x2_3:    skyrmionDUPLICATE port map (IN_SK => JOIN1_out, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_3_outtop, OUT_SK_BOTTOM => x2_3_outbottom);
CROSSM4:   SKYRMIONCROSS_Magn port map (A => JOIN11_out, B => x2_3_outtop, CURRENTA => CURRENT_Vmove1, CURRENTB => CURRENT_Vmove1, Aout => CROSSM4_outA, Bout => CROSSM4_outB);
x2_7:    skyrmionDUPLICATE port map (IN_SK => CROSSM4_outA, CURRENT => CURRENT_Vmove1, OUT_SK_TOP => x2_7_outtop, OUT_SK_BOTTOM => x2_7_outbottom);

CTRL_mux2 <= CTRL_Vrout;
Vmux2_Vsout:  voltage_genL port map (CTRL => CTRL_Vsout, CURRENT => CURRENT_Vsout);
Vmux2_Vrout:  voltage_genL port map (CTRL => CTRL_Vrout, CURRENT => CURRENT_Vrout);
MUX2_COM:     mux2 port map (inR => CROSS4_Bout, inL => CROSS7_Bout, CURRENT => CURRENT_Vmove3, CURRENT_Vst => CURRENT_Vsout, CURRENT_Vres => CURRENT_Vrout, selection => CTRL_mux2, muxOUT => MUX2_out, VstOUT => MUX2_Vstout, VresOUT => MUX2_Vresout);

Vmoveres:    voltage_genL port map (CTRL => CTRL_Vmoveres, CURRENT => CURRENT_Vmoveres);
x2_5:      skyrmionDUPLICATE port map (IN_SK => MUX2_out, CURRENT => CURRENT_Vmoveres, OUT_SK_TOP => x2_5_outright, OUT_SK_BOTTOM => x2_5_outleft);
```

```
  MTJ_CONV_12:  MTJ_CONV port map (IN_SK => x2_5_outleft, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV12_out);
  x2_6:       skyrmionDUPLICATE port map (IN_SK => x2_5_outright, CURRENT => CURRENT_Vmoveres, OUT_SK_TOP => x2_6_outtop, OUT_SK_BOTTOM => x2_6_outbottom);
  MTJ_CONV_11:  MTJ_CONV port map (IN_SK => x2_6_outtop, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV11_out);
  MTJ_CONV_13:  MTJ_CONV port map (IN_SK => x2_6_outbottom, CURRENT => CURRENT_Vmoveres, OUT_SK => MTJ_CONV13_out);

  Vtanktop:  vclock_gen port map (CTRL => CTRL_Vtanktop, CURRENTclk => CURRENTclk, CURRENT => CURRENT_Vtanktop);
  TANK_TOP_C:  tank_topXX port map (IN1 => MUX1_V2Tout, IN2 => MUX1_V3out, IN3 => DEV2_devout, IN4 => CROSS15_Bout, IN5 => CROSS14_Bout, CURRENT => CURRENT_Vtanktop, tankOUT => TANKT_out);
  MTJ_CONV_9:  MTJ_CONV port map (IN_SK => x2_4_outtop, CURRENT => CURRENT_Vmove3, OUT_SK => MTJ_CONV9_out);
  Vmove4:    voltage_genL port map (CTRL => CTRL_Vmove4, CURRENT => CURRENT_Vmove4);
  Vnewdata:  voltage_genL port map (CTRL => CTRL_Vnewdata, CURRENT => CURRENT_Vnewdata);
  DEV2:    skyrmionDEVIATION port map (IN_SK => MTJ_CONV9_out, CURRENT => CURRENT_Vmove4, CURRENTDEV => CURRENT_Vnewdata, OUT_SK => DEV2_out, OUT_SK_DEV => DEV2_devout);
  Vdetect:  voltage_genL port map (CTRL => CTRL_Vdetect, CURRENT => CURRENT_Vdetect);
  MTJ_R_2:  MTJ_R port map (IN_SK => TANKT_out, CURRENT => CURRENT_Vdetect, OUT_SIGN => MTJ_R2_out);
  RST_SR_MTJR2 <= ACK_DETECT_MTJ_R2 or (not RESET_n);
  SR_MTJR2:  SRlatch port map (SET => MTJ_R2_out, RST => RST_SR_MTJR2, Q => SR_MTJR2_out);

  CELL_OUT_MTJ_CONV14_out   <= MTJ_CONV14_out;
  CELL_OUT_MTJ_CONV15_out   <= MTJ_CONV15_out;
  CELL_OUT_MTJ_CONV11_out   <= MTJ_CONV11_out;
  CELL_OUT_MTJ_CONV13_out   <= MTJ_CONV13_out;
  CELL_OUT_MTJ_CONV12_out   <= MTJ_CONV12_out;
  CELL_OUT_DEV2_out       <= DEV2_out;
  CELL_OUT_TANKT_out      <= TANKT_out;

  CELL_OUT_CURRENT_Vmoveres   <= CURRENT_Vmoveres;
  CELL_OUT_CURRENT_Vmovecout  <= CURRENT_Vmovecout;


end Structure;
```

C.5.1.1. Top tank (only cell 11)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity tank_topXX is
port(  IN1, IN2, IN3, IN4, IN5: in std_logic; --from right to left
    CURRENT: in real;
    tankOUT: out std_logic);
end entity tank_topXX;

architecture Behavioural of tank_topXX is
  component SKYRMIONNOTCH is
  port( INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  signal OUT1: std_logic;

begin
  process (IN1, IN2, IN3, IN4, IN5, CURRENT)
    variable Nsk: integer := 0;
    variable out_en, OUT1_var: std_logic := '0';
  begin
    if (IN1'event and IN1='1') then
      Nsk := Nsk+1;
    end if;
    if (IN2'event and IN2='1') then
      Nsk := Nsk+1;
    end if;
    if (IN3'event and IN3='1') then
      Nsk := Nsk+1;
    end if;
    if (IN4'event and IN4='1') then
      Nsk := Nsk+1;
    end if;
    if (IN5'event and IN5='1') then
      Nsk := Nsk+1;
    end if;

    if (CURRENT'event and CURRENT = CURRENT_LOW) then
      if (out_en='0') then
```

```
        out_en := '1';
        case Nsk is
          when 1 =>   OUT1_var := '1';
                Nsk := 0;
          when 0 =>   OUT1_var := '0';
                Nsk := 0;
          when others =>   OUT1_var := '1';
                  Nsk := Nsk-1;
        end case;
      end if;
    end if;

    if (CURRENT'event and CURRENT = 0.0) then
      out_en := '0';
    end if;

    if (OUT1_var = '1') then
      OUT1_var := '0';
      OUT1 <= '1', '0' after 10 ps;
    else
      OUT1 <= '0';
    end if;

  end process;

  notch1: SKYRMIONNOTCH port map (INPUT => OUT1, CURRENT => CURRENT, OUTPUT =>
  ↪ tankOUT);

end architecture Behavioural;
```

C.5.2. FSM

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity FSM_CELL_XX is
port(  CURRENTclk: in real;
    RESET_n: in std_logic;

```

```
    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3,
    ↪  CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4,
    ↪  CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop,
    ↪  CTRL_Vouttank: out std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
    CTRL_mux1: out std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
    SR_MTJR2_out, SR_MTJR1_out: in std_logic;

    START, ACK_RES_AVAILABLE: in std_logic;
    DES_EXTIN_XX, DES_OUT_XX, DES_STORE_XX, DES_NEWDATA_XX: in std_logic;
    DES_RES_XX: in std_logic_vector(1 downto 0);
    RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE:
    ↪  out std_logic;
    READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic;

    CTRL_Vcouttop, CTRL_Vcoutleft, CTRL_Vrestop, CTRL_Vresleft,
    ↪  CTRL_Vresbotleft: out std_logic;
    READY_FOR_NEWDATA: out std_logic;
    DES_COUT_XX: in std_logic;
    DES_DATA_XX: in std_logic_vector(1 downto 0)
);
end entity FSM_CELL_XX;

architecture Behaviour of FSM_CELL_XX is

  type state_type is (
    reset, S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10,
    S11_1, S11_2, S11_3, S11_4, S11_5, S11_6,
    S11, S12, S13, S14, S15, S16, S17, S18, S19, S20,
    S21, S22, S23, S24, S25, S26, S27, S28, S29, S30,
    S31, S32, S33, S34, S35, S36, S37, S38, S39, S40,
    S41, S42, S43, S44, S45, S46, S47, S48, S49, S50
  );

  signal pstate, nstate: state_type;

begin

  state_register: process (CURRENTclk)
  begin
    if (CURRENTclk'event and CURRENTclk=CURRENT_HIGH) then
      if (RESET_n = '0') then
        pstate <= reset;
      else
        pstate <= nstate;
      end if;
    end if;
  end process state_register;
```

```

state_transition: process (pstate, CURRENTclk)
begin
  case pstate is
    when reset  => nstate <= S0;
    when S0   => if (START='0') then nstate <= S0; else if(FIRST_RUN='1') then nstate <= S1; else nstate <= S36; end if; end if;
    when S1   => if(DES_COUT_XX='0') then nstate <= S2; else nstate <= S3; end if;
    when S2   => if (DES_DATA_XX="00") then nstate <= S4; elsif(DES_DATA_XX="01") then nstate <= S5; elsif (DES_DATA_XX="10")  then nstate <= S6; else nstate <= S7; end if;
    when S3   => if (DES_DATA_XX="00") then nstate <= S4; elsif(DES_DATA_XX="01") then nstate <= S5; elsif (DES_DATA_XX="10")  then nstate <= S6; else nstate <= S7; end if;
    when S4   => nstate <= S8;
    when S5   => nstate <= S8;
    when S6   => nstate <= S8;
    when S7   => if(DES_EXTIN_XX='1') then nstate <= S9; else nstate <= S10; end if;
    when S8   => nstate <= S11;
    when S9   => nstate <= S11;
    when S10   => nstate <= S11;
    when S11   => nstate <= S11_1;
    when S11_1   => nstate <= S11_2;
    when S11_2   => nstate <= S11_3;
    when S11_3   => nstate <= S11_4;
    when S11_4   => nstate <= S11_5;
    when S11_5   => nstate <= S11_6;
    when S11_6   => nstate <= S12;
    when S12   => nstate <= S13;
    when S13   => if (DES_RES_XX="10") then nstate <= S14; elsif (DES_RES_XX="01") then nstate <= S15; else nstate <= S16; end if;
    when S14   => nstate <= S17;
    when S15   => nstate <= S17;
    when S16   => nstate <= S17;
    when S17   => nstate <= S18;
    when S18   => if (DES_OUT_XX='1') then nstate <= S19; else nstate <= S20; end if;
    when S19   => nstate <= S21;
    when S20   => nstate <= S21;
    when S21   => nstate <= S22;
    when S22   => if(DES_STORE_XX='1') then nstate <= S23; else nstate <= S24; end if;
    when S23   => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S23; else nstate <= S25; end if;
    when S24   => if(READY_FOR_NEWDATA_RIGHT='0') then nstate <= S24; else if(DES_NEWDATA_XX='1') then nstate <= S26; else nstate <= S27; end if; end if;
```

```
      when S25   => nstate <= S29;
      when S26   => nstate <= S28;
      when S27  => nstate <= S31;
      when S28   => nstate <= S30;
      when S29   => if (ACK_RES_AVAILABLE='0') then nstate <= S29; else nstate <= S33; end if;
      when S30   => if (ACK_RES_AVAILABLE='0') then nstate <= S30; else nstate <= S32; end if;
      when S31   => if (ACK_RES_AVAILABLE='0') then nstate <= S31; else nstate <= S35; end if;
      when S32  => if (SR_MTJR2_out='0') then nstate <= S34; else nstate <= S35; end if;
      when S33   => if(START='0') then nstate <= S33; else nstate <= S36; end if;
      when S34   => if(START='0') then nstate <= S34; else nstate <= S36; end if;
      when S35   => if(START='0') then nstate <= S35; else nstate <= S36; end if;
      when S36   => nstate <= S37;
      when S37   => if (DES_COUT_XX='0') then nstate <= S38; else nstate <= S39; end if;
      when S38   => if (DES_DATA_XX="00") then nstate <= S40; elsif (DES_DATA_XX="01") then nstate <= S41; elsif (DES_DATA_XX="10") then nstate <= S42; else nstate <= S43; end if;
      when S39   => if (DES_DATA_XX="00") then nstate <= S40; elsif (DES_DATA_XX="01") then nstate <= S41; elsif (DES_DATA_XX="10") then nstate <= S42; else nstate <= S43; end if;
      when S40   => nstate <= S44;
      when S41   => nstate <= S44;
      when S42   => nstate <= S44;
      when S43   => if (DES_EXTIN_XX='0') then nstate <= S45; else nstate <= S46; end if;
      when S44   => nstate <= S45;
      when S45   => if (SR_MTJR1_out='0') then nstate <= S49; else nstate <= S47; end if;
      when S46   => if (SR_MTJR1_out='1') then nstate <= S50; else nstate <= S48; end if;
      when S47   => nstate <= S49;
      when S48   => nstate <= S50;
      when S49   => nstate <= S11;
      when S50   => nstate <= S11;
      when others => nstate <= S0;
    end case;
  end process state_transition;

  output: process (pstate)
  begin

    CTRL_Vstart <= '0';
```

```
    CTRL_Vmove1 <= '0';
    CTRL_Vop <= '0';
    CTRL_Vmove2 <= '0';
    CTRL_Vmove2C <= '0';
    CTRL_Vmove3 <= '0';
    CTRL_Vmovecout <= '0';
    CTRL_Vtankbot <= '0';
    CTRL_Vnoext <= '0';
    CTRL_Vmoveres <= '0';
    CTRL_Vmove4 <= '0';
    CTRL_Vnewdata <= '0';
    CTRL_Vdetect <= '0';
    CTRL_Vrout <= '0';
    CTRL_Vsout <= '0';
    CTRL_Vtanktop <= '0';
    CTRL_Vouttank <= '0';
    CTRL_MTJ_W1 <= '0';
    CTRL_MTJ_W2 <= '0';
    CTRL_MTJ_W3 <= '0';
    CTRL_mux1 <= "00";
    ACK_DETECT_MTJ_R1 <= '0';
    ACK_DETECT_MTJ_R2 <= '0';
    RES_AVAILABLE <= '0';
    REQUEST_NEW_START <= '0';
    REQUEST_NEW_START_W <= '0';
    REQUEST_NEW_STORE <= '0';
    ---
    CTRL_Vcouttop <= '0';
    CTRL_Vcoutleft <= '0';
    CTRL_Vrestop <= '0';
    CTRL_Vresleft <= '0';
    CTRL_Vresbotleft <= '0';
    READY_FOR_NEWDATA <= '0';

    case pstate is
      when S0    =>  CTRL_Vstart <= '0';
              CTRL_Vmove1 <= '0';
              CTRL_Vop <= '0';
              CTRL_Vmove2 <= '0';
              CTRL_Vmove2C <= '0';
              CTRL_Vmove3 <= '0';
              CTRL_Vmovecout <= '0';
              CTRL_Vtankbot <= '0';
              CTRL_Vnoext <= '0';
              CTRL_Vmoveres <= '0';
              CTRL_Vmove4 <= '0';
              CTRL_Vnewdata <= '0';
              CTRL_Vdetect <= '0';
              CTRL_Vrout <= '0';
```

```
                CTRL_Vsout <= '0';
                CTRL_Vtanktop <= '0';
                CTRL_Vouttank <= '0';
                CTRL_MTJ_W1 <= '0';
                CTRL_MTJ_W2 <= '0';
                CTRL_MTJ_W3 <= '0';
                CTRL_mux1 <= "00";
                ACK_DETECT_MTJ_R1 <= '0';
                ACK_DETECT_MTJ_R2 <= '0';
                RES_AVAILABLE <= '0';
                REQUEST_NEW_START <= '0';
                REQUEST_NEW_START_W <= '0';
                REQUEST_NEW_STORE <= '0';
                ---
                CTRL_Vcouttop <= '0';
                CTRL_Vcoutleft <= '0';
                CTRL_Vrestop <= '0';
                CTRL_Vresleft <= '0';
                CTRL_Vresbotleft <= '0';
                READY_FOR_NEWDATA <= '0';
        when S1    =>  CTRL_Vnoext <= '1', '0' after CLOCK_PERIOD/2;
                READY_FOR_NEWDATA <= '1';
                CTRL_Vstart <= '1';
        when S2    =>  CTRL_Vcouttop <= '1';
        when S3    =>  CTRL_Vcoutleft <= '1';
        when S4    =>  CTRL_Vrestop <= '1';
        when S5    =>  CTRL_Vresleft <= '1';
        when S6    =>  CTRL_Vresbotleft <= '1';
        when S7    =>  CTRL_Vnoext <= '1';
        when S8    =>  CTRL_Vnoext <= '1';
                CTRL_Vmove1 <= '1';
                CTRL_MTJ_W1 <= '1';
                CTRL_MTJ_W2 <= '1';
        when S9    =>  CTRL_Vmove1 <= '1';
                CTRL_MTJ_W1 <= '1';
                CTRL_MTJ_W2 <= '1';
                CTRL_MTJ_W3 <= '1';
        when S10  =>  CTRL_Vmove1 <= '1';
                CTRL_MTJ_W1 <= '1';
                CTRL_MTJ_W2 <= '1';
        when S11  =>  CTRL_Vop <= '1';
        when S11_1  =>  CTRL_Vop <= '1';
        when S11_2  =>  CTRL_Vop <= '1';
        when S11_3  =>  CTRL_Vop <= '1';
        when S11_4  =>  CTRL_Vop <= '1';
        when S11_5  =>  CTRL_Vop <= '1';
        when S11_6  =>  CTRL_Vop <= '1';
        when S12  =>  CTRL_Vmove2 <= '1';
                CTRL_Vmove2C <= '1';
```

```
        when S13  =>  CTRL_Vmove2C <= '1';
        when S14  =>  CTRL_mux1 <= "10";
        when S15  =>  CTRL_mux1 <= "01";
        when S16  =>  CTRL_mux1 <= "11";
        when S17  =>  CTRL_Vmove2 <= '1';
        when S18  =>  CTRL_Vmove3 <= '1';
        when S19  =>  CTRL_Vrout <= '1';
        when S20  =>  CTRL_Vsout <= '1';
        when S21  =>  CTRL_Vmove3 <= '1';
        when S22  =>  CTRL_Vmove4 <= '1', '0' after CLOCK_PERIOD/2;
        when S23  =>  CTRL_Vmove4 <= '1';
        when S24  =>  CTRL_Vnewdata <= '1';
        when S25  =>  CTRL_Vmoveres <= '1';
                CTRL_Vmovecout <= '1';
        when S26  =>  CTRL_Vtanktop <= '1';
                CTRL_Vmoveres <= '1';
                CTRL_Vmovecout <= '1';
        when S27  =>  CTRL_Vmoveres <= '1';
                CTRL_Vmovecout <= '1';
        when S28  =>  CTRL_Vtanktop <= '1';
        when S29  =>  RES_AVAILABLE <= '1';
        when S30  =>  RES_AVAILABLE <= '1';
        when S31  =>  RES_AVAILABLE <= '1';
        when S32  =>  CTRL_Vdetect <= '1';
        when S33  =>  REQUEST_NEW_STORE <= '1';
        when S34  =>  REQUEST_NEW_START_W <= '1';
                ACK_DETECT_MTJ_R2 <= '1';
        when S35  =>  REQUEST_NEW_START <= '1';
                ACK_DETECT_MTJ_R2 <= '1';
        when S36  =>  CTRL_Vstart <= '1';
                CTRL_Vtankbot <= '1';
        when S37  =>  CTRL_Vtankbot <= '1';
                CTRL_Vnoext <= '1', '0' after CLOCK_PERIOD/2;
                READY_FOR_NEWDATA <= '1';
                CTRL_Vouttank <= '1';
        when S38  =>  CTRL_Vcouttop <= '1';
        when S39  =>  CTRL_Vcoutleft <= '1';
        when S40  =>  CTRL_Vrestop <= '1';
        when S41  =>  CTRL_Vresleft <= '1';
        when S42  =>  CTRL_Vresbotleft <= '1';
        when S43  =>  CTRL_Vnoext <= '1';
        when S44  =>  CTRL_Vnoext <= '1';
        when S45  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
        when S46  =>  CTRL_Vmove1 <= '1', '0' after CLOCK_PERIOD/2;
        when S47  =>  CTRL_Vnoext <= '1';
                ACK_DETECT_MTJ_R1 <= '1';
        when S48  =>  CTRL_MTJ_W3 <= '1';
        when S49  =>  CTRL_Vmove1 <= '1';
        when S50  =>  CTRL_Vmove1 <= '1';
```

```
                ACK_DETECT_MTJ_R1 <= '1';
        when others =>  CTRL_Vstart <= '0';
                CTRL_Vmove1 <= '0';
                CTRL_Vop <= '0';
                CTRL_Vmove2 <= '0';
                CTRL_Vmove2C <= '0';
                CTRL_Vmove3 <= '0';
                CTRL_Vmovecout <= '0';
                CTRL_Vtankbot <= '0';
                CTRL_Vnoext <= '0';
                CTRL_Vmoveres <= '0';
                CTRL_Vmove4 <= '0';
                CTRL_Vnewdata <= '0';
                CTRL_Vdetect <= '0';
                CTRL_Vrout <= '0';
                CTRL_Vsout <= '0';
                CTRL_Vtanktop <= '0';
                CTRL_Vouttank <= '0';
                CTRL_MTJ_W1 <= '0';
                CTRL_MTJ_W2 <= '0';
                CTRL_MTJ_W3 <= '0';
                CTRL_mux1 <= "00";
                ACK_DETECT_MTJ_R1 <= '0';
                ACK_DETECT_MTJ_R2 <= '0';
                RES_AVAILABLE <= '0';
                REQUEST_NEW_START <= '0';
                REQUEST_NEW_START_W <= '0';
                REQUEST_NEW_STORE <= '0';
                ---
                CTRL_Vcouttop <= '0';
                CTRL_Vcoutleft <= '0';
                CTRL_Vrestop <= '0';
                CTRL_Vresleft <= '0';
                CTRL_Vresbotleft <= '0';
                READY_FOR_NEWDATA <= '0';
    end case;
  end process output;
end Behaviour;
```

C.6. Memory array

C.6.1. Datapath

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity memory is
port(  RESET_n: in std_logic;
    CURRENTclk: in real;

    DES_EXTIN_00, DES_OUT_00, DES_STORE_00, DES_NEWDATA_00: in std_logic;
    DES_RES_00: in std_logic_vector(1 downto 0);

    DES_EXTIN_10, DES_OUT_10, DES_STORE_10, DES_NEWDATA_10, DES_DATA_10: in
    ↪  std_logic;
    DES_RES_10: in std_logic_vector(1 downto 0);

    DES_EXTIN_01, DES_OUT_01, DES_STORE_01, DES_NEWDATA_01: in std_logic;
    DES_DATA_01, DES_RES_01: in std_logic_vector(1 downto 0);

    DES_EXTIN_11, DES_OUT_11, DES_STORE_11, DES_NEWDATA_11, DES_COUT_11: in
    ↪  std_logic;
    DES_DATA_11, DES_RES_11: in std_logic_vector(1 downto 0)
);
end entity memory;

architecture Structure of memory is
  component MTJ_W is
  port(  CTRL: in std_logic;
      OUT_SK:  out std_logic);
  end component MTJ_W;

  component voltage_genL is
  port(  CTRL: in std_logic;
      CURRENT: out real);
  end component voltage_genL;

  component cell_input is
  port(  IN_SK_L:  in std_logic;
      IN_SK_T:   in std_logic;
      CURRENT_L:  in real;
```

```
        CURRENT_T:  in real;
        OUT_SK_R:  out std_logic;
        OUT_SK_B:  out std_logic);
    end component cell_input;

    component SKYRMIONJOIN is
    port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic);
    end component SKYRMIONJOIN;

    component SRlatch_H is
    port(  SET:   in std_logic;
        RST:   in std_logic;
        Q:    buffer std_logic
    );
    end component SRlatch_H;

    component FSM_MAIN is
    port(  CURRENTclk: in real;
        RESET_n: in std_logic;

        --from CELL 00
        INTERRUPT_00, RES_00_AVAILABLE, REQUEST_NEW_START_00, REQUEST_NEW_START_W_00, REQUEST_NEW_STORE_00, FIRST_RUN_00: in std_logic;
        --to CELL 00
        CTRL_MTJ_W4_00, CTRL_Vbl_00, CTRL_Vstore_00, START_00, ACK_RES00_AVAILABLE, RST_FIRST_RUN_00: out std_logic;

        --from CELL 10
        INTERRUPT_10, RES_10_AVAILABLE, REQUEST_NEW_START_10, REQUEST_NEW_START_W_10, REQUEST_NEW_STORE_10, FIRST_RUN_10, ALL_INPUTS_AVAILABLE_10: in std_logic;
        --to CELL 10
        CTRL_MTJ_W4_10, CTRL_Vbl_10, CTRL_Vstore_10, FIRST_START_10, START_10, ACK_RES10_AVAILABLE, RST_FIRST_RUN_10: out std_logic;
        RST_READY_10_00: out std_logic;

        --from CELL 01
        INTERRUPT_01, RES_01_AVAILABLE, REQUEST_NEW_START_01, REQUEST_NEW_START_W_01, REQUEST_NEW_STORE_01, FIRST_RUN_01, ALL_INPUTS_AVAILABLE_01: in std_logic;
        --to CELL 01
        CTRL_MTJ_W4_01, CTRL_Vbl_01, CTRL_Vstore_01, FIRST_START_01, START_01, ACK_RES01_AVAILABLE, RST_FIRST_RUN_01: out std_logic;
        RST_READY_01_00, RST_READY_01_10: out std_logic;

```

```
        --from CELL 11
        INTERRUPT_11, RES_11_AVAILABLE, REQUEST_NEW_START_11,
        ↪  REQUEST_NEW_START_W_11, REQUEST_NEW_STORE_11, FIRST_RUN_11,
        ↪  ALL_INPUTS_AVAILABLE_11: in std_logic;
        --to CELL 11
        CTRL_MTJ_W4_11, CTRL_Vbl_11, CTRL_Vstore_11, FIRST_START_11, START_11,
        ↪  ACK_RES11_AVAILABLE, RST_FIRST_RUN_11: out std_logic;
        RST_READY_11_10, RST_READY_11_01: out std_logic
    );
    end component FSM_MAIN;

--*************************************************
--CELL00--*****************************************
--*************************************************
    component CELL_00 is
    port(  IN_CELL, RESET_n: in std_logic;
        CURRENTclk: in real;

        CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C,
        ↪  CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext,
        ↪  CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout,
        ↪  CTRL_Vsout, CTRL_Vtanktop: in std_logic;
        CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
        CTRL_mux1: in std_logic_vector(1 downto 0);
        ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
        SR_MTJR1_out, SR_MTJR2_out: out std_logic;
        CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

        CELL_OUT_MTJ_CONV12_out, CELL_OUT_MTJ_CONV11_out, CELL_OUT_DEV2_out,
        ↪  CELL_OUT_CROSS4_Aout, CELL_OUT_TANKT_out: out std_logic
    );
    end component CELL_00;

    component FSM_CELL_00 is
    port(  CURRENTclk: in real;
        RESET_n: in std_logic;

        CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C,
        ↪  CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext,
        ↪  CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout,
        ↪  CTRL_Vsout, CTRL_Vtanktop: out std_logic;
        CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
        CTRL_mux1: out std_logic_vector(1 downto 0);
        ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
        SR_MTJR2_out, SR_MTJR1_out: in std_logic;

        START, ACK_RES_AVAILABLE: in std_logic;
        DES_EXTIN_00, DES_OUT_00, DES_STORE_00, DES_NEWDATA_00: in std_logic;
        DES_RES_00: in std_logic_vector(1 downto 0);
```

```
      RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE:
      ↪  out std_logic;
      READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic
  );
  end component FSM_CELL_00;

  --CELL00
  signal IN_CELL_00: std_logic;
  signal CTRL_Vstart_00, CTRL_Vmove1_00, CTRL_Vop_00, CTRL_Vmove2_00,
  ↪  CTRL_Vmove2C_00, CTRL_Vmove3_00, CTRL_Vmovecout_00, CTRL_Vtankbot_00,
  ↪  CTRL_Vnoext_00, CTRL_Vmoveres_00, CTRL_Vmove4_00, CTRL_Vnewdata_00,
  ↪  CTRL_Vdetect_00, CTRL_Vrout_00, CTRL_Vsout_00, CTRL_Vtanktop_00:
  ↪  std_logic;
  signal CTRL_MTJ_W1_00, CTRL_MTJ_W2_00, CTRL_MTJ_W3_00: std_logic;
  signal CTRL_mux1_00: std_logic_vector(1 downto 0);
  signal ACK_DETECT_MTJ_R1_00, ACK_DETECT_MTJ_R2_00: std_logic;
  signal SR_MTJR1_out_00, SR_MTJR2_out_00: std_logic;
  signal CELL_OUT_CURRENT_Vmoveres_00, CELL_OUT_CURRENT_Vmovecout_00: real;
  ---
  signal CELL_OUT_MTJ_CONV12_out_00, CELL_OUT_MTJ_CONV11_out_00,
  ↪  CELL_OUT_DEV2_out_00, CELL_OUT_CROSS4_Aout_00, CELL_OUT_TANKT_out_00:
  ↪  std_logic;
  ---
  signal START_00, ACK_RES00_AVAILABLE: std_logic;
  signal RES_00_AVAILABLE, REQUEST_NEW_START_00,REQUEST_NEW_START_W_00,
  ↪  REQUEST_NEW_STORE_00: std_logic;
  signal READY_FOR_NEWDATA_RIGHT_00: std_logic;
  ---
  signal CTRL_MTJ_W4_00, CTRL_Vbl_00, CTRL_Vstore_00: std_logic;
  signal MTJ_W4_out_00, BL_00_bottom, JOIN4_00_out: std_logic;
  signal CURRENT_Vbl_00, CURRENT_Vstore_00: real;
  ---
  signal START_00_AND: std_logic;
  signal INTERRUPT_00, FIRST_RUN_00, RST_FIRST_RUN_00: std_logic;


--************************************************
--CELLX0--****************************************
--************************************************
  component CELL_X0 is
  port(  IN_CELL, RESET_n: in std_logic;
      CURRENTclk: in real;

      CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C,
      ↪  CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext,
      ↪  CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout,
      ↪  CTRL_Vsout, CTRL_Vtanktop: in std_logic;
      CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
      CTRL_mux1: in std_logic_vector(1 downto 0);
```

```
        ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
        SR_MTJR1_out, SR_MTJR2_out: out std_logic;
        CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

        CELL_OUT_MTJ_CONV12_out, CELL_OUT_CROSS5_Aout, CELL_OUT_DEV2_out,
        ↪  CELL_OUT_CROSS6_Aout, CELL_OUT_CROSS6_Bout, CELL_OUT_TANKT_out: out
        ↪  std_logic;

        CTRL_Vrestop: in std_logic;
        TOP_RESULT:  in std_logic;
        TOP_RESULT_CURRENT: in real
    );
    end component CELL_X0;

    component FSM_CELL_X0 is
    port(  CURRENTclk: in real;
        RESET_n: in std_logic;

        CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C,
        ↪  CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext,
        ↪  CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout,
        ↪  CTRL_Vsout, CTRL_Vtanktop: out std_logic;
        CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
        CTRL_mux1: out std_logic_vector(1 downto 0);
        ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
        SR_MTJR2_out, SR_MTJR1_out: in std_logic;

        START, ACK_RES_AVAILABLE: in std_logic;
        DES_EXTIN_X0, DES_OUT_X0, DES_STORE_X0, DES_NEWDATA_X0: in std_logic;
        DES_RES_X0: in std_logic_vector(1 downto 0);
        RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE:
        ↪  out std_logic;
        READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic;

        CTRL_Vrestop: out std_logic;
        READY_FOR_NEWDATA: out std_logic;
        DES_DATA_X0: in std_logic
    );
    end component FSM_CELL_X0;

    --CELL10
    signal IN_CELL_10: std_logic;
    signal CTRL_Vstart_10, CTRL_Vmove1_10, CTRL_Vop_10, CTRL_Vmove2_10,
    ↪  CTRL_Vmove2C_10, CTRL_Vmove3_10, CTRL_Vmovecout_10, CTRL_Vtankbot_10,
    ↪  CTRL_Vnoext_10, CTRL_Vmoveres_10, CTRL_Vmove4_10, CTRL_Vnewdata_10,
    ↪  CTRL_Vdetect_10, CTRL_Vrout_10, CTRL_Vsout_10, CTRL_Vtanktop_10:
    ↪  std_logic;
    signal CTRL_MTJ_W1_10, CTRL_MTJ_W2_10, CTRL_MTJ_W3_10: std_logic;
    signal CTRL_mux1_10: std_logic_vector(1 downto 0);
```

```
  signal ACK_DETECT_MTJ_R1_10, ACK_DETECT_MTJ_R2_10: std_logic;
  signal SR_MTJR1_out_10, SR_MTJR2_out_10: std_logic;
  signal CELL_OUT_CURRENT_Vmoveres_10, CELL_OUT_CURRENT_Vmovecout_10: real;
  ---
  signal CELL_OUT_MTJ_CONV12_out_10, CELL_OUT_CROSS5_Aout_10, CELL_OUT_DEV2_out_10, CELL_OUT_CROSS6_Aout_10, CELL_OUT_CROSS6_Bout_10, CELL_OUT_TANKT_out_10: std_logic;
  ---
  signal CTRL_Vrestop_10: std_logic;
  signal TOP_RESULT_10: std_logic;
  signal TOP_RESULT_CURRENT_10: real;
  ---
  signal FIRST_START_10, ALL_INPUTS_AVAILABLE_10, START_10, ACK_RES10_AVAILABLE: std_logic;
  signal RES_10_AVAILABLE, REQUEST_NEW_START_10, REQUEST_NEW_START_W_10, REQUEST_NEW_STORE_10: std_logic;
  signal READY_FOR_NEWDATA_RIGHT_10: std_logic;
  ---
  signal READY_FOR_NEWDATA_10: std_logic;
  ---
  signal CTRL_MTJ_W4_10, CTRL_Vbl_10, CTRL_Vstore_10: std_logic;
  signal MTJ_W4_out_10, BL_10_bottom, JOIN4_10_out: std_logic;
  signal CURRENT_Vbl_10, CURRENT_Vstore_10: real;
  ---
  signal START_10_AND, LATCH_10_00_OUT: std_logic;
  signal INTERRUPT_10, FIRST_RUN_10, RST_FIRST_RUN_10, RST_READY_10_00: std_logic;

--*************************************************
--CELL0X--*****************************************
--*************************************************
  component CELL_0X is
  port(  IN_CELL, RESET_n: in std_logic;
      CURRENTclk: in real;

      CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop, CTRL_Vouttank: in std_logic;
      CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
      CTRL_mux1: in std_logic_vector(1 downto 0);
      ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
      SR_MTJR1_out, SR_MTJR2_out: out std_logic;
      CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

      CELL_OUT_MTJ_CONV14_out, CELL_OUT_CROSS9_Bout, CELL_OUT_CROSS9_Aout, CELL_OUT_MTJ_CONV12_out, CELL_OUT_DEV2_out, CELL_OUT_TANKT_out: out std_logic;

```

```
        CTRL_Vresbotleft, CTRL_Vresleft: in std_logic;
        LEFT_COUT, LEFT_RESULT, BOTLEFT_RESULT:  in std_logic;
        LEFT_COUT_CURRENT, LEFT_RESULT_CURRENT, BOTLEFT_RESULT_CURRENT: in real
    );
    end component CELL_0X;

    component FSM_CELL_0X is
    port(  CURRENTclk: in real;
        RESET_n: in std_logic;

        CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C,
        ↪  CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext,
        ↪  CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout,
        ↪  CTRL_Vsout, CTRL_Vtanktop, CTRL_Vouttank: out std_logic;
        CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
        CTRL_mux1: out std_logic_vector(1 downto 0);
        ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
        SR_MTJR2_out, SR_MTJR1_out: in std_logic;

        START, ACK_RES_AVAILABLE: in std_logic;
        DES_EXTIN_0X, DES_OUT_0X, DES_STORE_0X, DES_NEWDATA_0X: in std_logic;
        DES_RES_0X: in std_logic_vector(1 downto 0);
        RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE:
        ↪  out std_logic;
        READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic;

        CTRL_Vresleft, CTRL_Vresbotleft: out std_logic;
        READY_FOR_NEWDATA: out std_logic;
        DES_DATA_0X: in std_logic_vector(1 downto 0)
    );
    end component FSM_CELL_0X;

    --CELL01
    signal IN_CELL_01: std_logic;
    signal CTRL_Vstart_01, CTRL_Vmove1_01, CTRL_Vop_01, CTRL_Vmove2_01,
    ↪  CTRL_Vmove2C_01, CTRL_Vmove3_01, CTRL_Vmovecout_01, CTRL_Vtankbot_01,
    ↪  CTRL_Vnoext_01, CTRL_Vmoveres_01, CTRL_Vmove4_01, CTRL_Vnewdata_01,
    ↪  CTRL_Vdetect_01, CTRL_Vrout_01, CTRL_Vsout_01, CTRL_Vtanktop_01,
    ↪  CTRL_Vouttank_01: std_logic;
    signal CTRL_MTJ_W1_01, CTRL_MTJ_W2_01, CTRL_MTJ_W3_01: std_logic;
    signal CTRL_mux1_01: std_logic_vector(1 downto 0);
    signal ACK_DETECT_MTJ_R1_01, ACK_DETECT_MTJ_R2_01: std_logic;
    signal SR_MTJR1_out_01, SR_MTJR2_out_01: std_logic;
    signal CELL_OUT_CURRENT_Vmoveres_01, CELL_OUT_CURRENT_Vmovecout_01: real;
    ---
    signal CELL_OUT_MTJ_CONV14_out_01, CELL_OUT_CROSS9_Bout_01,
    ↪  CELL_OUT_CROSS9_Aout_01, CELL_OUT_MTJ_CONV12_out_01, CELL_OUT_DEV2_out_01,
    ↪  CELL_OUT_TANKT_out_01: std_logic;
    ---
```

```
  signal CTRL_Vresbotleft_01, CTRL_Vresleft_01: std_logic;
  signal LEFT_COUT_01, LEFT_RESULT_01, BOTLEFT_RESULT_01: std_logic;
  signal LEFT_COUT_CURRENT_01, LEFT_RESULT_CURRENT_01, BOTLEFT_RESULT_CURRENT_01: real;
  ---
  signal FIRST_START_01, ALL_INPUTS_AVAILABLE_01, START_01, ACK_RES01_AVAILABLE: std_logic;
  signal RES_01_AVAILABLE, REQUEST_NEW_START_01, REQUEST_NEW_START_W_01, REQUEST_NEW_STORE_01: std_logic;
  signal READY_FOR_NEWDATA_RIGHT_01: std_logic;
  ---
  signal READY_FOR_NEWDATA_01: std_logic;
  ---
  signal CTRL_MTJ_W4_01, CTRL_Vbl_01, CTRL_Vstore_01: std_logic;
  signal MTJ_W4_out_01, BL_01_bottom, JOIN4_01_out: std_logic;
  signal CURRENT_Vbl_01, CURRENT_Vstore_01: real;
  ---
  signal START_01_AND, LATCH_01_00_OUT, LATCH_01_10_OUT: std_logic;
  signal INTERRUPT_01, FIRST_RUN_01, RST_FIRST_RUN_01, RST_READY_01_00, RST_READY_01_10: std_logic;

--************************************************
--CELLXX--****************************************
--************************************************
  component CELL_XX is
  port(  IN_CELL, RESET_n: in std_logic;
      CURRENTclk: in real;

      CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop, CTRL_Vouttank: in std_logic;
      CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: in std_logic;
      CTRL_mux1: in std_logic_vector(1 downto 0);
      ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: in std_logic;
      SR_MTJR1_out, SR_MTJR2_out: out std_logic;
      CELL_OUT_CURRENT_Vmoveres, CELL_OUT_CURRENT_Vmovecout: out real;

      CELL_OUT_MTJ_CONV14_out, CELL_OUT_MTJ_CONV15_out, CELL_OUT_MTJ_CONV11_out, CELL_OUT_MTJ_CONV13_out, CELL_OUT_MTJ_CONV12_out, CELL_OUT_DEV2_out, CELL_OUT_TANKT_out: out std_logic;

      CTRL_Vcoutleft, CTRL_Vcouttop, CTRL_Vrestop, CTRL_Vresbotleft, CTRL_Vresleft: in std_logic;
      TOP_COUT, LEFT_COUT, TOP_RESULT, LEFT_RESULT, BOTLEFT_RESULT:  in std_logic;
      TOP_COUT_CURRENT, LEFT_COUT_CURRENT, TOP_RESULT_CURRENT, LEFT_RESULT_CURRENT, BOTLEFT_RESULT_CURRENT: in real
  );
```

```
end component CELL_XX;

component FSM_CELL_XX is
port(  CURRENTclk: in real;
    RESET_n: in std_logic;

    CTRL_Vstart, CTRL_Vmove1, CTRL_Vop, CTRL_Vmove2, CTRL_Vmove2C, CTRL_Vmove3, CTRL_Vmovecout, CTRL_Vtankbot, CTRL_Vnoext, CTRL_Vmoveres, CTRL_Vmove4, CTRL_Vnewdata, CTRL_Vdetect, CTRL_Vrout, CTRL_Vsout, CTRL_Vtanktop, CTRL_Vouttank: out std_logic;
    CTRL_MTJ_W1, CTRL_MTJ_W2, CTRL_MTJ_W3: out std_logic;
    CTRL_mux1: out std_logic_vector(1 downto 0);
    ACK_DETECT_MTJ_R1, ACK_DETECT_MTJ_R2: out std_logic;
    SR_MTJR2_out, SR_MTJR1_out: in std_logic;

    START, ACK_RES_AVAILABLE: in std_logic;
    DES_EXTIN_XX, DES_OUT_XX, DES_STORE_XX, DES_NEWDATA_XX: in std_logic;
    DES_RES_XX: in std_logic_vector(1 downto 0);
    RES_AVAILABLE, REQUEST_NEW_START, REQUEST_NEW_START_W, REQUEST_NEW_STORE: out std_logic;
    READY_FOR_NEWDATA_RIGHT, FIRST_RUN: in std_logic;

    CTRL_Vcouttop, CTRL_Vcoutleft, CTRL_Vrestop, CTRL_Vresleft, CTRL_Vresbotleft: out std_logic;
    READY_FOR_NEWDATA: out std_logic;
    DES_COUT_XX: in std_logic;
    DES_DATA_XX: in std_logic_vector(1 downto 0)
);
end component FSM_CELL_XX;

--CELL11
signal IN_CELL_11: std_logic;
signal CTRL_Vstart_11, CTRL_Vmove1_11, CTRL_Vop_11, CTRL_Vmove2_11, CTRL_Vmove2C_11, CTRL_Vmove3_11, CTRL_Vmovecout_11, CTRL_Vtankbot_11, CTRL_Vnoext_11, CTRL_Vmoveres_11, CTRL_Vmove4_11, CTRL_Vnewdata_11, CTRL_Vdetect_11, CTRL_Vrout_11, CTRL_Vsout_11, CTRL_Vtanktop_11, CTRL_Vouttank_11: std_logic;
signal CTRL_MTJ_W1_11, CTRL_MTJ_W2_11, CTRL_MTJ_W3_11: std_logic;
signal CTRL_mux1_11: std_logic_vector(1 downto 0);
signal ACK_DETECT_MTJ_R1_11, ACK_DETECT_MTJ_R2_11: std_logic;
signal SR_MTJR1_out_11, SR_MTJR2_out_11: std_logic;
signal CELL_OUT_CURRENT_Vmoveres_11, CELL_OUT_CURRENT_Vmovecout_11: real;
---
signal CELL_OUT_MTJ_CONV14_out_11, CELL_OUT_MTJ_CONV15_out_11, CELL_OUT_MTJ_CONV11_out_11, CELL_OUT_MTJ_CONV13_out_11, CELL_OUT_MTJ_CONV12_out_11, CELL_OUT_DEV2_out_11, CELL_OUT_TANKT_out_11: std_logic;
---
signal CTRL_Vcoutleft_11, CTRL_Vcouttop_11, CTRL_Vrestop_11, CTRL_Vresbotleft_11, CTRL_Vresleft_11: std_logic;
```

```
  signal TOP_COUT_11, LEFT_COUT_11, TOP_RESULT_11, LEFT_RESULT_11, BOTLEFT_RESULT_11: std_logic;
  signal TOP_COUT_CURRENT_11, LEFT_COUT_CURRENT_11, TOP_RESULT_CURRENT_11, LEFT_RESULT_CURRENT_11, BOTLEFT_RESULT_CURRENT_11: real;
  ---
  signal FIRST_START_11, ALL_INPUTS_AVAILABLE_11, START_11, ACK_RES11_AVAILABLE: std_logic;
  signal RES_11_AVAILABLE, REQUEST_NEW_START_11, REQUEST_NEW_START_W_11, REQUEST_NEW_STORE_11: std_logic;
  signal READY_FOR_NEWDATA_RIGHT_11: std_logic;
  ---
  signal READY_FOR_NEWDATA_11: std_logic;
  ---
  signal CTRL_MTJ_W4_11, CTRL_Vbl_11, CTRL_Vstore_11: std_logic;
  signal MTJ_W4_out_11, BL_11_bottom, JOIN4_11_out: std_logic;
  signal CURRENT_Vbl_11, CURRENT_Vstore_11: real;
  ---
  signal START_11_AND, LATCH_11_10_OUT, LATCH_11_01_OUT: std_logic;
  signal INTERRUPT_11, FIRST_RUN_11, RST_FIRST_RUN_11, RST_READY_11_10, RST_READY_11_01: std_logic;

--************************************************
----------****************************************
--************************************************


begin

  INTERRUPT_00 <= RES_00_AVAILABLE or REQUEST_NEW_START_00 or REQUEST_NEW_START_W_00 or REQUEST_NEW_STORE_00;
  INTERRUPT_10 <= RES_10_AVAILABLE or REQUEST_NEW_START_10 or REQUEST_NEW_START_W_10 or REQUEST_NEW_STORE_10;
  INTERRUPT_01 <= RES_01_AVAILABLE or REQUEST_NEW_START_01 or REQUEST_NEW_START_W_01 or REQUEST_NEW_STORE_01;
  INTERRUPT_11 <= RES_11_AVAILABLE or REQUEST_NEW_START_11 or REQUEST_NEW_START_W_11 or REQUEST_NEW_STORE_11;

  MAINFSM: FSM_MAIN port map (
    CURRENTclk => CURRENTclk,
    RESET_n => RESET_n,
    ---
    INTERRUPT_00 => INTERRUPT_00,
    RES_00_AVAILABLE => RES_00_AVAILABLE,
    REQUEST_NEW_START_00 => REQUEST_NEW_START_00,
    REQUEST_NEW_START_W_00 => REQUEST_NEW_START_W_00,
    REQUEST_NEW_STORE_00 => REQUEST_NEW_STORE_00,
    FIRST_RUN_00 => FIRST_RUN_00,
    CTRL_MTJ_W4_00 => CTRL_MTJ_W4_00,
    CTRL_Vbl_00 => CTRL_Vbl_00,
```

```
    CTRL_Vstore_00 => CTRL_Vstore_00,
    START_00 => START_00,
    ACK_RES00_AVAILABLE => ACK_RES00_AVAILABLE,
    RST_FIRST_RUN_00 => RST_FIRST_RUN_00,
    ---
    INTERRUPT_10 => INTERRUPT_10,
    RES_10_AVAILABLE => RES_10_AVAILABLE,
    REQUEST_NEW_START_10 => REQUEST_NEW_START_10,
    REQUEST_NEW_START_W_10 => REQUEST_NEW_START_W_10,
    REQUEST_NEW_STORE_10 => REQUEST_NEW_STORE_10,
    FIRST_RUN_10 => FIRST_RUN_10,
    ALL_INPUTS_AVAILABLE_10 => ALL_INPUTS_AVAILABLE_10,
    CTRL_MTJ_W4_10 => CTRL_MTJ_W4_10,
    CTRL_Vbl_10 => CTRL_Vbl_10,
    CTRL_Vstore_10 => CTRL_Vstore_10,
    FIRST_START_10 => FIRST_START_10,
    START_10 => START_10,
    ACK_RES10_AVAILABLE => ACK_RES10_AVAILABLE,
    RST_FIRST_RUN_10 => RST_FIRST_RUN_10,
    RST_READY_10_00 => RST_READY_10_00,
    ---
    INTERRUPT_01 => INTERRUPT_01,
    RES_01_AVAILABLE => RES_01_AVAILABLE,
    REQUEST_NEW_START_01 => REQUEST_NEW_START_01,
    REQUEST_NEW_START_W_01 => REQUEST_NEW_START_W_01,
    REQUEST_NEW_STORE_01 => REQUEST_NEW_STORE_01,
    FIRST_RUN_01 => FIRST_RUN_01,
    ALL_INPUTS_AVAILABLE_01 => ALL_INPUTS_AVAILABLE_01,
    CTRL_MTJ_W4_01 => CTRL_MTJ_W4_01,
    CTRL_Vbl_01 => CTRL_Vbl_01,
    CTRL_Vstore_01 => CTRL_Vstore_01,
    FIRST_START_01 => FIRST_START_01,
    START_01 => START_01,
    ACK_RES01_AVAILABLE => ACK_RES01_AVAILABLE,
    RST_FIRST_RUN_01 => RST_FIRST_RUN_01,
    RST_READY_01_00 => RST_READY_01_00,
    RST_READY_01_10 => RST_READY_01_10,
    ---
    INTERRUPT_11 => INTERRUPT_11,
    RES_11_AVAILABLE => RES_11_AVAILABLE,
    REQUEST_NEW_START_11 => REQUEST_NEW_START_11,
    REQUEST_NEW_START_W_11 => REQUEST_NEW_START_W_11,
    REQUEST_NEW_STORE_11 => REQUEST_NEW_STORE_11,
    FIRST_RUN_11 => FIRST_RUN_11,
    ALL_INPUTS_AVAILABLE_11 => ALL_INPUTS_AVAILABLE_11,
    CTRL_MTJ_W4_11 => CTRL_MTJ_W4_11,
    CTRL_Vbl_11 => CTRL_Vbl_11,
    CTRL_Vstore_11 => CTRL_Vstore_11,
    FIRST_START_11 => FIRST_START_11,
```

```
    START_11 => START_11,
    ACK_RES11_AVAILABLE => ACK_RES11_AVAILABLE,
    RST_FIRST_RUN_11 => RST_FIRST_RUN_11,
    RST_READY_11_10 => RST_READY_11_10,
    RST_READY_11_01 => RST_READY_11_01
  );

--************************************************
--CELL00--****************************************
--************************************************

  --BL00
  MTJ4_00:   MTJ_W port map (CTRL => CTRL_MTJ_W4_00, OUT_SK => MTJ_W4_out_00);
  Vbl_00:    voltage_genL port map (CTRL => CTRL_Vbl_00, CURRENT => CURRENT_Vbl_00);
  Vstore_00:  voltage_genL port map (CTRL => CTRL_Vstore_00, CURRENT => CURRENT_Vstore_00);
  join4_00:  SKYRMIONJOIN port map (A => MTJ_W4_out_00, B => CELL_OUT_TANKT_out_00, CURRENT => CURRENT_Vbl_00, OUTPUT => JOIN4_00_out);
  cell00_in:  cell_input  port map (IN_SK_L => CELL_OUT_DEV2_out_00, IN_SK_T => JOIN4_00_out, CURRENT_L => CURRENT_Vstore_00, CURRENT_T => CURRENT_Vbl_00, OUT_SK_R => IN_CELL_00, OUT_SK_B => BL_00_bottom);

  START_00_AND <= START_00;
  READY_FOR_NEWDATA_RIGHT_00 <= LATCH_10_00_OUT and LATCH_01_00_OUT;

  LATCH_FIRSTRUN_00: SRlatch_H port map (SET => '0', RST => RST_FIRST_RUN_00, Q => FIRST_RUN_00);

  CELL00: CELL_00 port map (
    IN_CELL => IN_CELL_00,
    RESET_n => RESET_n,
    CURRENTclk => CURRENTclk,
    ---
    CTRL_Vstart => CTRL_Vstart_00,
    CTRL_Vmove1 => CTRL_Vmove1_00,
    CTRL_Vop => CTRL_Vop_00,
    CTRL_Vmove2 => CTRL_Vmove2_00,
    CTRL_Vmove2C => CTRL_Vmove2C_00,
    CTRL_Vmove3 => CTRL_Vmove3_00,
    CTRL_Vmovecout => CTRL_Vmovecout_00,
    CTRL_Vtankbot => CTRL_Vtankbot_00,
    CTRL_Vnoext => CTRL_Vnoext_00,
    CTRL_Vmoveres => CTRL_Vmoveres_00,
    CTRL_Vmove4 => CTRL_Vmove4_00,
    CTRL_Vnewdata => CTRL_Vnewdata_00,
    CTRL_Vdetect => CTRL_Vdetect_00,
    CTRL_Vrout => CTRL_Vrout_00,
    CTRL_Vsout => CTRL_Vsout_00,
```

```
    CTRL_Vtanktop => CTRL_Vtanktop_00,
    CTRL_MTJ_W1 => CTRL_MTJ_W1_00,
    CTRL_MTJ_W2 => CTRL_MTJ_W2_00,
    CTRL_MTJ_W3 => CTRL_MTJ_W3_00,
    CTRL_mux1 => CTRL_mux1_00,
    ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_00,
    ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_00,
    SR_MTJR1_out => SR_MTJR1_out_00,
    SR_MTJR2_out => SR_MTJR2_out_00,
    CELL_OUT_CURRENT_Vmoveres => CELL_OUT_CURRENT_Vmoveres_00,
    CELL_OUT_CURRENT_Vmovecout => CELL_OUT_CURRENT_Vmovecout_00,
    ---
    CELL_OUT_MTJ_CONV12_out => CELL_OUT_MTJ_CONV12_out_00,
    CELL_OUT_MTJ_CONV11_out => CELL_OUT_MTJ_CONV11_out_00,
    CELL_OUT_DEV2_out => CELL_OUT_DEV2_out_00,
    CELL_OUT_CROSS4_Aout => CELL_OUT_CROSS4_Aout_00,
    CELL_OUT_TANKT_out => CELL_OUT_TANKT_out_00
  );

  FSM00: FSM_CELL_00 port map (
    CURRENTclk => CURRENTclk,
    RESET_n => RESET_n,
    ---
    CTRL_Vstart => CTRL_Vstart_00,
    CTRL_Vmove1 => CTRL_Vmove1_00,
    CTRL_Vop => CTRL_Vop_00,
    CTRL_Vmove2 => CTRL_Vmove2_00,
    CTRL_Vmove2C => CTRL_Vmove2C_00,
    CTRL_Vmove3 => CTRL_Vmove3_00,
    CTRL_Vmovecout => CTRL_Vmovecout_00,
    CTRL_Vtankbot => CTRL_Vtankbot_00,
    CTRL_Vnoext => CTRL_Vnoext_00,
    CTRL_Vmoveres => CTRL_Vmoveres_00,
    CTRL_Vmove4 => CTRL_Vmove4_00,
    CTRL_Vnewdata => CTRL_Vnewdata_00,
    CTRL_Vdetect => CTRL_Vdetect_00,
    CTRL_Vrout => CTRL_Vrout_00,
    CTRL_Vsout => CTRL_Vsout_00,
    CTRL_Vtanktop => CTRL_Vtanktop_00,
    CTRL_MTJ_W1 => CTRL_MTJ_W1_00,
    CTRL_MTJ_W2 => CTRL_MTJ_W2_00,
    CTRL_MTJ_W3 => CTRL_MTJ_W3_00,
    CTRL_mux1 => CTRL_mux1_00,
    ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_00,
    ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_00,
    SR_MTJR2_out => SR_MTJR2_out_00,
    SR_MTJR1_out => SR_MTJR1_out_00,
    ---
    START => START_00_AND,
```

```
    ACK_RES_AVAILABLE => ACK_RES00_AVAILABLE,
    DES_EXTIN_00 => DES_EXTIN_00,
    DES_OUT_00 => DES_OUT_00,
    DES_STORE_00 => DES_STORE_00,
    DES_NEWDATA_00 => DES_NEWDATA_00,
    DES_RES_00 => DES_RES_00,
    RES_AVAILABLE => RES_00_AVAILABLE,
    REQUEST_NEW_START => REQUEST_NEW_START_00,
    REQUEST_NEW_START_W => REQUEST_NEW_START_W_00,
    REQUEST_NEW_STORE => REQUEST_NEW_STORE_00,
    READY_FOR_NEWDATA_RIGHT => READY_FOR_NEWDATA_RIGHT_00,
    FIRST_RUN => FIRST_RUN_00
  );

--************************************************
--CELL10--****************************************
--************************************************

  --BL10
  MTJ4_10:   MTJ_W port map (CTRL => CTRL_MTJ_W4_10, OUT_SK => MTJ_W4_out_10);
  Vbl_10:    voltage_genL port map (CTRL => CTRL_Vbl_10, CURRENT => CURRENT_Vbl_10);
  Vstore_10:  voltage_genL port map (CTRL => CTRL_Vstore_10, CURRENT => CURRENT_Vstore_10);
  join4_10:  SKYRMIONJOIN port map (A => MTJ_W4_out_10, B => CELL_OUT_TANKT_out_10, CURRENT => CURRENT_Vbl_10, OUTPUT => JOIN4_10_out);
  cell10_in:  cell_input  port map (IN_SK_L => CELL_OUT_DEV2_out_10, IN_SK_T => JOIN4_10_out, CURRENT_L => CURRENT_Vstore_10, CURRENT_T => CURRENT_Vbl_10, OUT_SK_R => IN_CELL_10, OUT_SK_B => BL_10_bottom);

  LATCH_10_00: SRlatch_H port map (SET => READY_FOR_NEWDATA_10, RST => RST_READY_10_00, Q => LATCH_10_00_OUT);
  START_10_AND <= ((FIRST_START_10 and FIRST_RUN_10) or START_10) and ALL_INPUTS_AVAILABLE_10;
  ALL_INPUTS_AVAILABLE_10 <= not(LATCH_10_00_OUT);
  READY_FOR_NEWDATA_RIGHT_10 <= LATCH_01_10_OUT and LATCH_11_10_OUT; --and LATCH_20_10_OUT

  LATCH_FIRSTRUN_10: SRlatch_H port map (SET => '0', RST => RST_FIRST_RUN_10, Q => FIRST_RUN_10);

  TOP_RESULT_10 <= CELL_OUT_MTJ_CONV12_out_00;
  TOP_RESULT_CURRENT_10 <= CELL_OUT_CURRENT_Vmoveres_00;

  CELL10:  CELL_X0 port map (
    IN_CELL => IN_CELL_10,
    RESET_n => RESET_n,
    CURRENTclk => CURRENTclk,
    ---
```

```
    CTRL_Vstart => CTRL_Vstart_10,
    CTRL_Vmove1 => CTRL_Vmove1_10,
    CTRL_Vop => CTRL_Vop_10,
    CTRL_Vmove2 => CTRL_Vmove2_10,
    CTRL_Vmove2C => CTRL_Vmove2C_10,
    CTRL_Vmove3 => CTRL_Vmove3_10,
    CTRL_Vmovecout => CTRL_Vmovecout_10,
    CTRL_Vtankbot => CTRL_Vtankbot_10,
    CTRL_Vnoext => CTRL_Vnoext_10,
    CTRL_Vmoveres => CTRL_Vmoveres_10,
    CTRL_Vmove4 => CTRL_Vmove4_10,
    CTRL_Vnewdata => CTRL_Vnewdata_10,
    CTRL_Vdetect => CTRL_Vdetect_10,
    CTRL_Vrout => CTRL_Vrout_10,
    CTRL_Vsout => CTRL_Vsout_10,
    CTRL_Vtanktop => CTRL_Vtanktop_10,
    CTRL_MTJ_W1 => CTRL_MTJ_W1_10,
    CTRL_MTJ_W2 => CTRL_MTJ_W2_10,
    CTRL_MTJ_W3 => CTRL_MTJ_W3_10,
    CTRL_mux1 => CTRL_mux1_10,
    ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_10,
    ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_10,
    SR_MTJR1_out => SR_MTJR1_out_10,
    SR_MTJR2_out => SR_MTJR2_out_10,
    CELL_OUT_CURRENT_Vmoveres => CELL_OUT_CURRENT_Vmoveres_10,
    CELL_OUT_CURRENT_Vmovecout => CELL_OUT_CURRENT_Vmovecout_10,
    ---
    CELL_OUT_MTJ_CONV12_out => CELL_OUT_MTJ_CONV12_out_10,
    CELL_OUT_CROSS5_Aout => CELL_OUT_CROSS5_Aout_10,
    CELL_OUT_DEV2_out => CELL_OUT_DEV2_out_10,
    CELL_OUT_CROSS6_Aout => CELL_OUT_CROSS6_Aout_10,
    CELL_OUT_CROSS6_Bout => CELL_OUT_CROSS6_Bout_10,
    CELL_OUT_TANKT_out => CELL_OUT_TANKT_out_10,
    ---
    CTRL_Vrestop => CTRL_Vrestop_10,
    TOP_RESULT => TOP_RESULT_10,
    TOP_RESULT_CURRENT => TOP_RESULT_CURRENT_10
  );

  FSM10: FSM_CELL_X0 port map (
    CURRENTclk => CURRENTclk,
    RESET_n => RESET_n,
    ---
    CTRL_Vstart => CTRL_Vstart_10,
    CTRL_Vmove1 => CTRL_Vmove1_10,
    CTRL_Vop => CTRL_Vop_10,
    CTRL_Vmove2 => CTRL_Vmove2_10,
    CTRL_Vmove2C => CTRL_Vmove2C_10,
    CTRL_Vmove3 => CTRL_Vmove3_10,
```

```
    CTRL_Vmovecout => CTRL_Vmovecout_10,
    CTRL_Vtankbot => CTRL_Vtankbot_10,
    CTRL_Vnoext => CTRL_Vnoext_10,
    CTRL_Vmoveres => CTRL_Vmoveres_10,
    CTRL_Vmove4 => CTRL_Vmove4_10,
    CTRL_Vnewdata => CTRL_Vnewdata_10,
    CTRL_Vdetect => CTRL_Vdetect_10,
    CTRL_Vrout => CTRL_Vrout_10,
    CTRL_Vsout => CTRL_Vsout_10,
    CTRL_Vtanktop => CTRL_Vtanktop_10,
    CTRL_MTJ_W1 => CTRL_MTJ_W1_10,
    CTRL_MTJ_W2 => CTRL_MTJ_W2_10,
    CTRL_MTJ_W3 => CTRL_MTJ_W3_10,
    CTRL_mux1 => CTRL_mux1_10,
    ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_10,
    ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_10,
    SR_MTJR2_out => SR_MTJR2_out_10,
    SR_MTJR1_out => SR_MTJR1_out_10,
    ---
    START => START_10_AND,
    ACK_RES_AVAILABLE => ACK_RES10_AVAILABLE,
    DES_EXTIN_X0 => DES_EXTIN_10,
    DES_OUT_X0 => DES_OUT_10,
    DES_STORE_X0 => DES_STORE_10,
    DES_NEWDATA_X0 => DES_NEWDATA_10,
    DES_RES_X0 => DES_RES_10,
    RES_AVAILABLE => RES_10_AVAILABLE,
    REQUEST_NEW_START => REQUEST_NEW_START_10,
    REQUEST_NEW_START_W => REQUEST_NEW_START_W_10,
    REQUEST_NEW_STORE => REQUEST_NEW_STORE_10,
    READY_FOR_NEWDATA_RIGHT => READY_FOR_NEWDATA_RIGHT_10,
    FIRST_RUN => FIRST_RUN_10,
    ---
    CTRL_Vrestop => CTRL_Vrestop_10,
    READY_FOR_NEWDATA => READY_FOR_NEWDATA_10,
    DES_DATA_X0 => DES_DATA_10
  );

--*************************************************
--CELL01--****************************************
--*************************************************

  --BL01
  MTJ4_01:   MTJ_W port map (CTRL => CTRL_MTJ_W4_01, OUT_SK => MTJ_W4_out_01);
  Vbl_01:    voltage_genL port map (CTRL => CTRL_Vbl_01, CURRENT =>
   ↪  CURRENT_Vbl_01);
  Vstore_01:  voltage_genL port map (CTRL => CTRL_Vstore_01, CURRENT =>
   ↪  CURRENT_Vstore_01);
  join4_01:  SKYRMIONJOIN port map (A => MTJ_W4_out_01, B =>
   ↪  CELL_OUT_TANKT_out_01, CURRENT => CURRENT_Vbl_01, OUTPUT => JOIN4_01_out);
```

```
cell01_in:  cell_input  port map (IN_SK_L => CELL_OUT_DEV2_out_01, IN_SK_T => JOIN4_01_out, CURRENT_L => CURRENT_Vstore_01, CURRENT_T => CURRENT_Vbl_01, OUT_SK_R => IN_CELL_01, OUT_SK_B => BL_01_bottom);

LATCH_01_00: SRlatch_H port map (SET => READY_FOR_NEWDATA_01, RST => RST_READY_01_00, Q => LATCH_01_00_OUT);
LATCH_01_10: SRlatch_H port map (SET => READY_FOR_NEWDATA_01, RST => RST_READY_01_10, Q => LATCH_01_10_OUT);
START_01_AND <= ((FIRST_START_01 and FIRST_RUN_01) or START_01) and ALL_INPUTS_AVAILABLE_01;
ALL_INPUTS_AVAILABLE_01 <= not(LATCH_01_00_OUT) and not (LATCH_01_10_OUT);
READY_FOR_NEWDATA_RIGHT_01 <= LATCH_11_01_OUT; --and LATCH_02_01_OUT

LATCH_FIRSTRUN_01: SRlatch_H port map (SET => '0', RST => RST_FIRST_RUN_01, Q => FIRST_RUN_01);

LEFT_COUT_01        <= CELL_OUT_CROSS4_Aout_00;
LEFT_RESULT_01        <= CELL_OUT_MTJ_CONV11_out_00;
BOTLEFT_RESULT_01      <= CELL_OUT_CROSS6_Bout_10;
LEFT_COUT_CURRENT_01    <= CELL_OUT_CURRENT_Vmovecout_00;
LEFT_RESULT_CURRENT_01    <= CELL_OUT_CURRENT_Vmoveres_00;
BOTLEFT_RESULT_CURRENT_01  <= CELL_OUT_CURRENT_Vmoveres_10;

CELL01:  CELL_0X port map (
  IN_CELL => IN_CELL_01,
  RESET_n => RESET_n,
  CURRENTclk => CURRENTclk,
  ---
  CTRL_Vstart => CTRL_Vstart_01,
  CTRL_Vmove1 => CTRL_Vmove1_01,
  CTRL_Vop => CTRL_Vop_01,
  CTRL_Vmove2 => CTRL_Vmove2_01,
  CTRL_Vmove2C => CTRL_Vmove2C_01,
  CTRL_Vmove3 => CTRL_Vmove3_01,
  CTRL_Vmovecout => CTRL_Vmovecout_01,
  CTRL_Vtankbot => CTRL_Vtankbot_01,
  CTRL_Vnoext => CTRL_Vnoext_01,
  CTRL_Vmoveres => CTRL_Vmoveres_01,
  CTRL_Vmove4 => CTRL_Vmove4_01,
  CTRL_Vnewdata => CTRL_Vnewdata_01,
  CTRL_Vdetect => CTRL_Vdetect_01,
  CTRL_Vrout => CTRL_Vrout_01,
  CTRL_Vsout => CTRL_Vsout_01,
  CTRL_Vtanktop => CTRL_Vtanktop_01,
  CTRL_Vouttank => CTRL_Vouttank_01,
  CTRL_MTJ_W1 => CTRL_MTJ_W1_01,
  CTRL_MTJ_W2 => CTRL_MTJ_W2_01,
  CTRL_MTJ_W3 => CTRL_MTJ_W3_01,
  CTRL_mux1 => CTRL_mux1_01,
```

```
    ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_01,
    ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_01,
    SR_MTJR1_out => SR_MTJR1_out_01,
    SR_MTJR2_out => SR_MTJR2_out_01,
    CELL_OUT_CURRENT_Vmoveres => CELL_OUT_CURRENT_Vmoveres_01,
    CELL_OUT_CURRENT_Vmovecout => CELL_OUT_CURRENT_Vmovecout_01,
    ---
    CELL_OUT_MTJ_CONV14_out => CELL_OUT_MTJ_CONV14_out_01,
    CELL_OUT_CROSS9_Bout => CELL_OUT_CROSS9_Bout_01,
    CELL_OUT_CROSS9_Aout => CELL_OUT_CROSS9_Aout_01,
    CELL_OUT_MTJ_CONV12_out => CELL_OUT_MTJ_CONV12_out_01,
    CELL_OUT_DEV2_out => CELL_OUT_DEV2_out_01,
    CELL_OUT_TANKT_out => CELL_OUT_TANKT_out_01,
    ---
    CTRL_Vresbotleft => CTRL_Vresbotleft_01,
    CTRL_Vresleft => CTRL_Vresleft_01,
    LEFT_COUT => LEFT_COUT_01,
    LEFT_RESULT => LEFT_RESULT_01,
    BOTLEFT_RESULT => BOTLEFT_RESULT_01,
    LEFT_COUT_CURRENT => LEFT_COUT_CURRENT_01,
    LEFT_RESULT_CURRENT  => LEFT_RESULT_CURRENT_01,
    BOTLEFT_RESULT_CURRENT => BOTLEFT_RESULT_CURRENT_01
  );

  FSM01: FSM_CELL_0X port map (
    CURRENTclk => CURRENTclk,
    RESET_n => RESET_n,
    ---
    CTRL_Vstart => CTRL_Vstart_01,
    CTRL_Vmove1 => CTRL_Vmove1_01,
    CTRL_Vop => CTRL_Vop_01,
    CTRL_Vmove2 => CTRL_Vmove2_01,
    CTRL_Vmove2C => CTRL_Vmove2C_01,
    CTRL_Vmove3 => CTRL_Vmove3_01,
    CTRL_Vmovecout => CTRL_Vmovecout_01,
    CTRL_Vtankbot => CTRL_Vtankbot_01,
    CTRL_Vnoext => CTRL_Vnoext_01,
    CTRL_Vmoveres => CTRL_Vmoveres_01,
    CTRL_Vmove4 => CTRL_Vmove4_01,
    CTRL_Vnewdata => CTRL_Vnewdata_01,
    CTRL_Vdetect => CTRL_Vdetect_01,
    CTRL_Vrout => CTRL_Vrout_01,
    CTRL_Vsout => CTRL_Vsout_01,
    CTRL_Vtanktop => CTRL_Vtanktop_01,
    CTRL_Vouttank => CTRL_Vouttank_01,
    CTRL_MTJ_W1 => CTRL_MTJ_W1_01,
    CTRL_MTJ_W2 => CTRL_MTJ_W2_01,
    CTRL_MTJ_W3 => CTRL_MTJ_W3_01,
    CTRL_mux1 => CTRL_mux1_01,
```

```
    ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_01,
    ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_01,
    SR_MTJR2_out => SR_MTJR2_out_01,
    SR_MTJR1_out => SR_MTJR1_out_01,
    ---
    START => START_01_AND,
    ACK_RES_AVAILABLE => ACK_RES01_AVAILABLE,
    DES_EXTIN_0X => DES_EXTIN_01,
    DES_OUT_0X => DES_OUT_01,
    DES_STORE_0X => DES_STORE_01,
    DES_NEWDATA_0X => DES_NEWDATA_01,
    DES_RES_0X => DES_RES_01,
    RES_AVAILABLE => RES_01_AVAILABLE,
    REQUEST_NEW_START => REQUEST_NEW_START_01,
    REQUEST_NEW_START_W => REQUEST_NEW_START_W_01,
    REQUEST_NEW_STORE => REQUEST_NEW_STORE_01,
    READY_FOR_NEWDATA_RIGHT => READY_FOR_NEWDATA_RIGHT_01,
    FIRST_RUN => FIRST_RUN_01,
    ---
    CTRL_Vresleft => CTRL_Vresleft_01,
    CTRL_Vresbotleft => CTRL_Vresbotleft_01,
    READY_FOR_NEWDATA => READY_FOR_NEWDATA_01,
    DES_DATA_0X => DES_DATA_01
  );

--*************************************************
--CELL11--****************************************
--*************************************************

  --BL11
  MTJ4_11:   MTJ_W port map (CTRL => CTRL_MTJ_W4_11, OUT_SK => MTJ_W4_out_11);
  Vbl_11:    voltage_genL port map (CTRL => CTRL_Vbl_11, CURRENT =>
    CURRENT_Vbl_11);
  Vstore_11:  voltage_genL port map (CTRL => CTRL_Vstore_11, CURRENT =>
    CURRENT_Vstore_11);
  join4_11:  SKYRMIONJOIN port map (A => MTJ_W4_out_11, B =>
    CELL_OUT_TANKT_out_11, CURRENT => CURRENT_Vbl_11, OUTPUT => JOIN4_11_out);
  cell11_in:  cell_input  port map (IN_SK_L => CELL_OUT_DEV2_out_11, IN_SK_T =>
    JOIN4_11_out, CURRENT_L => CURRENT_Vstore_11, CURRENT_T => CURRENT_Vbl_11,
    OUT_SK_R => IN_CELL_11, OUT_SK_B => BL_11_bottom);

  LATCH_11_10: SRlatch_H port map (SET => READY_FOR_NEWDATA_11, RST =>
    RST_READY_11_10, Q => LATCH_11_10_OUT);
  LATCH_11_01: SRlatch_H port map (SET => READY_FOR_NEWDATA_11, RST =>
    RST_READY_11_01, Q => LATCH_11_01_OUT);
  START_11_AND <= ((FIRST_START_11 and FIRST_RUN_11) or START_11) and
    ALL_INPUTS_AVAILABLE_11;
  ALL_INPUTS_AVAILABLE_11 <= not(LATCH_11_10_OUT) and not (LATCH_11_01_OUT);
  READY_FOR_NEWDATA_RIGHT_11 <= '0', '1' after 520 ns, '0' after 530 ns;
    --PROVVISORIO, sarebbe LATCH_21_11_OUT and LATCH_02_11_OUT and
    LATCH_12_11_OUT;
```

```

LATCH_FIRSTRUN_11: SRlatch_H port map (SET => '0', RST => RST_FIRST_RUN_11, Q => FIRST_RUN_11);

TOP_COUT_11          <= CELL_OUT_CROSS9_Bout_01;
LEFT_COUT_11        <= CELL_OUT_CROSS6_Aout_10;
TOP_RESULT_11        <= CELL_OUT_MTJ_CONV12_out_01;
LEFT_RESULT_11        <= CELL_OUT_CROSS5_Aout_10;
BOTLEFT_RESULT_11      <= '0';              --PROVVISORIO
TOP_COUT_CURRENT_11      <= CELL_OUT_CURRENT_Vmovecout_01;
LEFT_COUT_CURRENT_11    <= CELL_OUT_CURRENT_Vmovecout_10;
TOP_RESULT_CURRENT_11    <= CELL_OUT_CURRENT_Vmoveres_01;
LEFT_RESULT_CURRENT_11    <= CELL_OUT_CURRENT_Vmoveres_10;
BOTLEFT_RESULT_CURRENT_11  <= 0.0;              --PROVVISORIO

CELL11:  CELL_XX port map (
  IN_CELL => IN_CELL_11,
  RESET_n => RESET_n,
  CURRENTclk => CURRENTclk,
  ---
  CTRL_Vstart => CTRL_Vstart_11,
  CTRL_Vmove1 => CTRL_Vmove1_11,
  CTRL_Vop => CTRL_Vop_11,
  CTRL_Vmove2 => CTRL_Vmove2_11,
  CTRL_Vmove2C => CTRL_Vmove2C_11,
  CTRL_Vmove3 => CTRL_Vmove3_11,
  CTRL_Vmovecout => CTRL_Vmovecout_11,
  CTRL_Vtankbot => CTRL_Vtankbot_11,
  CTRL_Vnoext => CTRL_Vnoext_11,
  CTRL_Vmoveres => CTRL_Vmoveres_11,
  CTRL_Vmove4 => CTRL_Vmove4_11,
  CTRL_Vnewdata => CTRL_Vnewdata_11,
  CTRL_Vdetect => CTRL_Vdetect_11,
  CTRL_Vrout => CTRL_Vrout_11,
  CTRL_Vsout => CTRL_Vsout_11,
  CTRL_Vtanktop => CTRL_Vtanktop_11,
  CTRL_Vouttank => CTRL_Vouttank_11,
  CTRL_MTJ_W1 => CTRL_MTJ_W1_11,
  CTRL_MTJ_W2 => CTRL_MTJ_W2_11,
  CTRL_MTJ_W3 => CTRL_MTJ_W3_11,
  CTRL_mux1 => CTRL_mux1_11,
  ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_11,
  ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_11,
  SR_MTJR1_out => SR_MTJR1_out_11,
  SR_MTJR2_out => SR_MTJR2_out_11,
  CELL_OUT_CURRENT_Vmoveres => CELL_OUT_CURRENT_Vmoveres_11,
  CELL_OUT_CURRENT_Vmovecout => CELL_OUT_CURRENT_Vmovecout_11,
  ---
  CELL_OUT_MTJ_CONV14_out => CELL_OUT_MTJ_CONV14_out_11,
```

```
    CELL_OUT_MTJ_CONV15_out => CELL_OUT_MTJ_CONV15_out_11,
    CELL_OUT_MTJ_CONV11_out => CELL_OUT_MTJ_CONV11_out_11,
    CELL_OUT_MTJ_CONV13_out => CELL_OUT_MTJ_CONV13_out_11,
    CELL_OUT_MTJ_CONV12_out => CELL_OUT_MTJ_CONV12_out_11,
    CELL_OUT_DEV2_out => CELL_OUT_DEV2_out_11,
    CELL_OUT_TANKT_out => CELL_OUT_TANKT_out_11,
    ---
    CTRL_Vcoutleft => CTRL_Vcoutleft_11,
    CTRL_Vcouttop => CTRL_Vcouttop_11,
    CTRL_Vrestop => CTRL_Vrestop_11,
    CTRL_Vresbotleft => CTRL_Vresbotleft_11,
    CTRL_Vresleft => CTRL_Vresleft_11,
    TOP_COUT => TOP_COUT_11,
    LEFT_COUT => LEFT_COUT_11,
    TOP_RESULT => TOP_RESULT_11,
    LEFT_RESULT => LEFT_RESULT_11,
    BOTLEFT_RESULT => BOTLEFT_RESULT_11,
    TOP_COUT_CURRENT => TOP_COUT_CURRENT_11,
    LEFT_COUT_CURRENT => LEFT_COUT_CURRENT_11,
    TOP_RESULT_CURRENT => TOP_RESULT_CURRENT_11,
    LEFT_RESULT_CURRENT  => LEFT_RESULT_CURRENT_11,
    BOTLEFT_RESULT_CURRENT => BOTLEFT_RESULT_CURRENT_11
  );

  FSM11: FSM_CELL_XX port map (
    CURRENTclk => CURRENTclk,
    RESET_n => RESET_n,
    ---
    CTRL_Vstart => CTRL_Vstart_11,
    CTRL_Vmove1 => CTRL_Vmove1_11,
    CTRL_Vop => CTRL_Vop_11,
    CTRL_Vmove2 => CTRL_Vmove2_11,
    CTRL_Vmove2C => CTRL_Vmove2C_11,
    CTRL_Vmove3 => CTRL_Vmove3_11,
    CTRL_Vmovecout => CTRL_Vmovecout_11,
    CTRL_Vtankbot => CTRL_Vtankbot_11,
    CTRL_Vnoext => CTRL_Vnoext_11,
    CTRL_Vmoveres => CTRL_Vmoveres_11,
    CTRL_Vmove4 => CTRL_Vmove4_11,
    CTRL_Vnewdata => CTRL_Vnewdata_11,
    CTRL_Vdetect => CTRL_Vdetect_11,
    CTRL_Vrout => CTRL_Vrout_11,
    CTRL_Vsout => CTRL_Vsout_11,
    CTRL_Vtanktop => CTRL_Vtanktop_11,
    CTRL_Vouttank => CTRL_Vouttank_11,
    CTRL_MTJ_W1 => CTRL_MTJ_W1_11,
    CTRL_MTJ_W2 => CTRL_MTJ_W2_11,
    CTRL_MTJ_W3 => CTRL_MTJ_W3_11,
    CTRL_mux1 => CTRL_mux1_11,
```

```
    ACK_DETECT_MTJ_R1 => ACK_DETECT_MTJ_R1_11,
    ACK_DETECT_MTJ_R2 => ACK_DETECT_MTJ_R2_11,
    SR_MTJR2_out => SR_MTJR2_out_11,
    SR_MTJR1_out => SR_MTJR1_out_11,
    ---
    START => START_11_AND,
    ACK_RES_AVAILABLE => ACK_RES11_AVAILABLE,
    DES_EXTIN_XX => DES_EXTIN_11,
    DES_OUT_XX => DES_OUT_11,
    DES_STORE_XX => DES_STORE_11,
    DES_NEWDATA_XX => DES_NEWDATA_11,
    DES_RES_XX => DES_RES_11,
    RES_AVAILABLE => RES_11_AVAILABLE,
    REQUEST_NEW_START => REQUEST_NEW_START_11,
    REQUEST_NEW_START_W => REQUEST_NEW_START_W_11,
    REQUEST_NEW_STORE => REQUEST_NEW_STORE_11,
    READY_FOR_NEWDATA_RIGHT => READY_FOR_NEWDATA_RIGHT_11,
    FIRST_RUN => FIRST_RUN_11,
    ---
    CTRL_Vcouttop => CTRL_Vcouttop_11,
    CTRL_Vcoutleft => CTRL_Vcoutleft_11,
    CTRL_Vrestop => CTRL_Vrestop_11,
    CTRL_Vresleft => CTRL_Vresleft_11,
    CTRL_Vresbotleft => CTRL_Vresbotleft_11,
    READY_FOR_NEWDATA => READY_FOR_NEWDATA_11,
    DES_COUT_XX => DES_COUT_11,
    DES_DATA_XX => DES_DATA_11
  );

end architecture Structure;
```

C.6.1.1. SRlatch_H

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SRlatch_H is
port(  SET:   in std_logic;
    RST:   in std_logic;
    Q:     out std_logic
);
```

```
end entity SRlatch_H;

architecture Behaviour of SRlatch_H is

begin
  latch: process (SET, RST)
    variable first: std_logic := '1';
  begin
    if (first='1') then
      first := '0';
      Q <= '1';
    end if;
    if (RST='1') then
      Q <= '0';
    elsif(SET'event and SET='1') then
      Q <= '1';
    end if;
  end process latch;

end architecture Behaviour;
```

C.6.1.2. Cell_input

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity cell_input is
port(  IN_SK_L:  in std_logic;
    IN_SK_T:   in std_logic;
    CURRENT_L:  in real;
    CURRENT_T:  in real;
    OUT_SK_R:  out std_logic;
    OUT_SK_B:  out std_logic);
end entity cell_input;

architecture Behavioural of cell_input is
begin
  process (IN_SK_L, IN_SK_T, CURRENT_L, CURRENT_T)
    variable NskL, NskT: integer := 0;
```

```
    variable st_T: integer := 0;
  begin
    if (IN_SK_L'event and IN_SK_L='1') then
      NskL := NskL+1;
    end if;
    if (IN_SK_T'event and IN_SK_T='1') then
      NskT := NskT+1;
    end if;

    if (CURRENT_T /= 0.0) then
      st_T := 1;
      OUT_SK_R <= '0';
      OUT_SK_B <= '0';    --suppongo che lo sk si fermi esattamente
      ↪  all'intersezione, e non proceda oltre
    end if;
    if (CURRENT_L /= 0.0) then
      if (st_T=1) then
        st_T := 0;
        if (NskT=1) then
          OUT_SK_R <= '1', '0' after 10 ps;
          OUT_SK_B <= '0';
          NskT := 0;
        else
          OUT_SK_R <= '0';
          OUT_SK_B <= '0';
        end if;
      else
        if (NskL=1) then
          OUT_SK_R <= '1', '0' after 10 ps;
          OUT_SK_B <= '0';
          NskL := 0;
        else
          OUT_SK_R <= '0';
          OUT_SK_B <= '0';
        end if;
      end if;
    else
      OUT_SK_R <= '0';
      OUT_SK_B <= '0';
    end if;
  end process;
end architecture Behavioural;
```

C.6.2. Master FSM

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity FSM_MAIN is
port(  CURRENTclk: in real;
    RESET_n: in std_logic;

    --from CELL 00
    INTERRUPT_00, RES_00_AVAILABLE, REQUEST_NEW_START_00, REQUEST_NEW_START_W_00, REQUEST_NEW_STORE_00, FIRST_RUN_00: in std_logic;
    --to CELL 00
    CTRL_MTJ_W4_00, CTRL_Vbl_00, CTRL_Vstore_00, START_00, ACK_RES00_AVAILABLE, RST_FIRST_RUN_00: out std_logic;

    --from CELL 10
    INTERRUPT_10, RES_10_AVAILABLE, REQUEST_NEW_START_10, REQUEST_NEW_START_W_10, REQUEST_NEW_STORE_10, FIRST_RUN_10, ALL_INPUTS_AVAILABLE_10: in std_logic;
    --to CELL 10
    CTRL_MTJ_W4_10, CTRL_Vbl_10, CTRL_Vstore_10, FIRST_START_10, START_10, ACK_RES10_AVAILABLE, RST_FIRST_RUN_10: out std_logic;
    RST_READY_10_00: out std_logic;

    --from CELL 01
    INTERRUPT_01, RES_01_AVAILABLE, REQUEST_NEW_START_01, REQUEST_NEW_START_W_01, REQUEST_NEW_STORE_01, FIRST_RUN_01, ALL_INPUTS_AVAILABLE_01: in std_logic;
    --to CELL 01
    CTRL_MTJ_W4_01, CTRL_Vbl_01, CTRL_Vstore_01, FIRST_START_01, START_01, ACK_RES01_AVAILABLE, RST_FIRST_RUN_01: out std_logic;
    RST_READY_01_00, RST_READY_01_10: out std_logic;

    --from CELL 11
    INTERRUPT_11, RES_11_AVAILABLE, REQUEST_NEW_START_11, REQUEST_NEW_START_W_11, REQUEST_NEW_STORE_11, FIRST_RUN_11, ALL_INPUTS_AVAILABLE_11: in std_logic;
    --to CELL 11
    CTRL_MTJ_W4_11, CTRL_Vbl_11, CTRL_Vstore_11, FIRST_START_11, START_11, ACK_RES11_AVAILABLE, RST_FIRST_RUN_11: out std_logic;
    RST_READY_11_10, RST_READY_11_01: out std_logic
```

```
);
end entity FSM_MAIN;

architecture Behaviour of FSM_MAIN is

  type state_type is (
    S0, S1, S2, S3, idle, ISR_00, ISR_10, ISR_01, ISR_11,
    ISR00_S1, ISR00_S2, ISR00_S3, ISR00_S4, ISR00_S5,
    ISR10_S1, ISR10_S2, ISR10_S3, ISR10_S4, ISR10_S5,
    ISR01_S1, ISR01_S2, ISR01_S3, ISR01_S4, ISR01_S5,
    ISR11_S1, ISR11_S2, ISR11_S3, ISR11_S4, ISR11_S5
  );

  signal pstate, nstate: state_type;

begin

  state_register: process (CURRENTclk)
  begin
    if (CURRENTclk'event and CURRENTclk=CURRENT_HIGH) then
      if (RESET_n = '0') then
        pstate <= S0;
      else
        pstate <= nstate;
      end if;
    end if;
  end process state_register;

  state_transition: process (pstate, CURRENTclk)
  begin
    case pstate is
      when S0    =>   nstate <= S1;
      when S1    =>   nstate <= S2;
      when S2    =>   nstate <= S3;
      when S3   =>   if (INTERRUPT_00='1') then nstate <= ISR_00; elsif (INTERRUPT_10='1') then nstate <= ISR_10; elsif (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      when idle  =>  if (INTERRUPT_00='1') then nstate <= ISR_00; elsif (INTERRUPT_10='1') then nstate <= ISR_10; elsif (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      ---
      when ISR_00    => if (RES_00_AVAILABLE='1') then nstate <=  ISR00_S1; elsif (REQUEST_NEW_START_00='1') then nstate <= ISR00_S3; elsif (REQUEST_NEW_START_W_00='1') then nstate <= ISR00_S4; elsif (REQUEST_NEW_STORE_00='1') then nstate <= ISR00_S5; end if;
      when ISR00_S1  => if (FIRST_RUN_00='1') then nstate <= ISR00_S2; else if (INTERRUPT_10='1') then nstate <= ISR_10; elsif (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if; end if;
```

```
      when ISR00_S2  => if (INTERRUPT_10='1') then nstate <= ISR_10; elsif (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      when ISR00_S3  => nstate <= ISR00_S5;
      when ISR00_S4  => nstate <= ISR00_S5;
      when ISR00_S5  => if (INTERRUPT_10='1') then nstate <= ISR_10; elsif (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      ---
      when ISR_10    =>   if (RES_10_AVAILABLE='1') then
                  nstate <=  ISR10_S1;
                elsif (ALL_INPUTS_AVAILABLE_10='0') then
                  if (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
                else
                  if (REQUEST_NEW_START_10='1') then nstate <= ISR10_S3; elsif (REQUEST_NEW_START_W_10='1') then nstate <= ISR10_S4; elsif (REQUEST_NEW_STORE_10='1') then nstate <= ISR10_S5; end if;
                end if;
      when ISR10_S1  => if (FIRST_RUN_10='1') then nstate <= ISR10_S2; else if (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if; end if;
      when ISR10_S2  => if (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      when ISR10_S3  => nstate <= ISR10_S5;
      when ISR10_S4  => nstate <= ISR10_S5;
      when ISR10_S5  => if (INTERRUPT_01='1') then nstate <= ISR_01; elsif (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      ---
      when ISR_01    =>   if (RES_01_AVAILABLE='1') then
                  nstate <=  ISR01_S1;
                elsif (ALL_INPUTS_AVAILABLE_01='0') then
                  if (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
                else
                  if (REQUEST_NEW_START_01='1') then nstate <= ISR01_S3; elsif (REQUEST_NEW_START_W_01='1') then nstate <= ISR01_S4; elsif (REQUEST_NEW_STORE_01='1') then nstate <= ISR01_S5; end if;
                end if;
      when ISR01_S1  => if (FIRST_RUN_01='1') then nstate <= ISR01_S2; else if (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if; end if;
      when ISR01_S2  => if (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      when ISR01_S3  => nstate <= ISR01_S5;
      when ISR01_S4  => nstate <= ISR01_S5;
```

```
      when ISR01_S5  => if (INTERRUPT_11='1') then nstate <= ISR_11; else nstate <= idle; end if;
      ---
      when ISR_11    =>   if (RES_11_AVAILABLE='1') then
                  nstate <=  ISR11_S1;
                elsif (ALL_INPUTS_AVAILABLE_11='0') then
                  nstate <= idle;
                else
                  if (REQUEST_NEW_START_11='1') then nstate <= ISR11_S3; elsif (REQUEST_NEW_START_W_11='1') then nstate <= ISR11_S4; elsif (REQUEST_NEW_STORE_11='1') then nstate <= ISR11_S5; end if;
                end if;
      when ISR11_S1  => if (FIRST_RUN_11='1') then nstate <= ISR11_S2; else nstate <= idle; end if;
      when ISR11_S2  => nstate <= idle;
      when ISR11_S3  => nstate <= ISR11_S5;
      when ISR11_S4  => nstate <= ISR11_S5;
      when ISR11_S5  => nstate <= idle;
      ---
      when others   => nstate <= idle;
    end case;
  end process state_transition;

  output: process (pstate)
  begin
    CTRL_MTJ_W4_00 <= '0';
    CTRL_Vbl_00 <= '0';
    CTRL_Vstore_00 <= '0';
    START_00 <= '0';
    ACK_RES00_AVAILABLE <= '0';
    RST_FIRST_RUN_00 <= '0';
    ---
    CTRL_MTJ_W4_10 <= '0';
    CTRL_Vbl_10 <= '0';
    CTRL_Vstore_10 <= '0';
    FIRST_START_10 <= '0';
    START_10 <= '0';
    ACK_RES10_AVAILABLE <= '0';
    RST_FIRST_RUN_10 <= '0';
    RST_READY_10_00 <= '0';
    ---
    CTRL_MTJ_W4_01 <= '0';
    CTRL_Vbl_01 <= '0';
    CTRL_Vstore_01 <= '0';
    FIRST_START_01 <= '0';
    START_01 <= '0';
    ACK_RES01_AVAILABLE <= '0';
    RST_FIRST_RUN_01 <= '0';
```

```
    RST_READY_01_00 <= '0';
    RST_READY_01_10 <= '0';
    ---
    CTRL_MTJ_W4_11 <= '0';
    CTRL_Vbl_11 <= '0';
    CTRL_Vstore_11 <= '0';
    FIRST_START_11 <= '0';
    START_11 <= '0';
    ACK_RES11_AVAILABLE <= '0';
    RST_FIRST_RUN_11 <= '0';
    RST_READY_11_10 <= '0';
    RST_READY_11_01 <= '0';

    case pstate is
      when S0    =>  CTRL_MTJ_W4_00 <= '0';
              CTRL_Vbl_00 <= '0';
              CTRL_Vstore_00 <= '0';
              START_00 <= '0';
              ACK_RES00_AVAILABLE <= '0';
              RST_FIRST_RUN_00 <= '0';
              ---
              CTRL_MTJ_W4_10 <= '0';
              CTRL_Vbl_10 <= '0';
              CTRL_Vstore_10 <= '0';
              FIRST_START_10 <= '0';
              START_10 <= '0';
              ACK_RES10_AVAILABLE <= '0';
              RST_FIRST_RUN_10 <= '0';
              RST_READY_10_00 <= '0';
              ---
              CTRL_MTJ_W4_01 <= '0';
              CTRL_Vbl_01 <= '0';
              CTRL_Vstore_01 <= '0';
              FIRST_START_01 <= '0';
              START_01 <= '0';
              ACK_RES01_AVAILABLE <= '0';
              RST_FIRST_RUN_01 <= '0';
              RST_READY_01_00 <= '0';
              RST_READY_01_10 <= '0';
              ---
              CTRL_MTJ_W4_11 <= '0';
              CTRL_Vbl_11 <= '0';
              CTRL_Vstore_11 <= '0';
              FIRST_START_11 <= '0';
              START_11 <= '0';
              ACK_RES11_AVAILABLE <= '0';
              RST_FIRST_RUN_11 <= '0';
              RST_READY_11_10 <= '0';
              RST_READY_11_01 <= '0';
```

```
        when S1    =>  CTRL_MTJ_W4_00 <= '1';        --PROVVISORIO
                CTRL_MTJ_W4_10 <= '1';        --PROVVISORIO
                CTRL_MTJ_W4_01 <= '1';        --PROVVISORIO
                CTRL_MTJ_W4_11 <= '1';        --PROVVISORIO
                CTRL_Vbl_00 <= '1';
                CTRL_Vbl_10 <= '1';
                CTRL_Vbl_01 <= '1';
                CTRL_Vbl_11 <= '1';
        when S2    =>  CTRL_Vstore_00 <= '1';
                CTRL_Vstore_10 <= '1';
                CTRL_Vstore_01 <= '1';
                CTRL_Vstore_11 <= '1';
        when S3    =>   START_00 <= '1';

        when idle  =>  null;
        ---
        when ISR_00    =>   null;
        when ISR00_S1  =>   ACK_RES00_AVAILABLE <= '1';
                  RST_READY_10_00 <= '1';
                  RST_READY_01_00 <= '1';
                  FIRST_START_10 <= '1';
        when ISR00_S2  =>   RST_FIRST_RUN_00 <= '1';
        when ISR00_S3  =>   CTRL_Vbl_00 <= '1';
        when ISR00_S4  =>   CTRL_MTJ_W4_00 <= '1';
                  CTRL_Vbl_00 <= '1';
        when ISR00_S5  =>  CTRL_Vstore_00 <= '1';
                  START_00 <= '1';
        ---
        when ISR_10    =>   null;
        when ISR10_S1  =>   ACK_RES10_AVAILABLE <= '1';
                  --RST_READY_20_10 <= '1';
                  RST_READY_01_10 <= '1';
                  RST_READY_11_10 <= '1';
                  --FIRST_START20 <= '1';
                  FIRST_START_01 <= '1';
        when ISR10_S2  =>   RST_FIRST_RUN_10 <= '1';
        when ISR10_S3  =>   CTRL_Vbl_10 <= '1';
        when ISR10_S4  =>   CTRL_MTJ_W4_10 <= '1';
                  CTRL_Vbl_10 <= '1';
        when ISR10_S5  =>   CTRL_Vstore_10 <= '1';
                  START_10 <= '1';
        ---
        when ISR_01    =>   null;
        when ISR01_S1  =>   ACK_RES01_AVAILABLE <= '1';
                  RST_READY_11_01 <= '1';
                  --RST_READY_02_01 <= '1';
                  FIRST_START_11 <= '1';
                  --FIRST_START_30 <= '1';
        when ISR01_S2  =>   RST_FIRST_RUN_01 <= '1';
```

```
      when ISR01_S3   =>   CTRL_Vbl_01 <= '1';
      when ISR01_S4   =>   CTRL_MTJ_W4_01 <= '1';
                 CTRL_Vbl_01 <= '1';
      when ISR01_S5   =>   CTRL_Vstore_01 <= '1';
                 START_01 <= '1';
      ---
      when ISR_11     =>   null;
      when ISR11_S1   =>   ACK_RES11_AVAILABLE <= '1';
                 --RST_READY_21_11 <= '1';
                 --RST_READY_02_11 <= '1';
                 --RST_READY_12_11 <= '1';
                 --FIRST_START_40 <= '1';
                 --FIRST_START_21 <= '1';
                 --FIRST_START_02 <= '1';
      when ISR11_S2   =>   RST_FIRST_RUN_11 <= '1';
      when ISR11_S3   =>   CTRL_Vbl_11 <= '1';
      when ISR11_S4   =>   CTRL_MTJ_W4_11 <= '1';
                 CTRL_Vbl_11 <= '1';
      when ISR11_S5   =>   CTRL_Vstore_11 <= '1';
                 START_11 <= '1';
      ---
      when others  => null;
    end case;
  end process output;
end Behaviour;
```

C.6.3. Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity TB_memory is
end entity TB_memory;

architecture Structure of TB_memory is
  component memory is
  port(  RESET_n: in std_logic;
      CURRENTclk: in real;

```

```
      DES_EXTIN_00, DES_OUT_00, DES_STORE_00, DES_NEWDATA_00: in std_logic;
      DES_RES_00: in std_logic_vector(1 downto 0);

      DES_EXTIN_10, DES_OUT_10, DES_STORE_10, DES_NEWDATA_10, DES_DATA_10: in std_logic;
      DES_RES_10: in std_logic_vector(1 downto 0);

      DES_EXTIN_01, DES_OUT_01, DES_STORE_01, DES_NEWDATA_01: in std_logic;
      DES_DATA_01, DES_RES_01: in std_logic_vector(1 downto 0);

      DES_EXTIN_11, DES_OUT_11, DES_STORE_11, DES_NEWDATA_11, DES_COUT_11: in std_logic;
      DES_DATA_11, DES_RES_11: in std_logic_vector(1 downto 0)
  );
  end component memory;

  signal RESET_n: std_logic;
  signal CURRENTclk: real;

  signal DES_EXTIN_00, DES_OUT_00, DES_STORE_00, DES_NEWDATA_00: std_logic;
  signal DES_RES_00: std_logic_vector(1 downto 0);

  signal DES_EXTIN_10, DES_OUT_10, DES_STORE_10, DES_NEWDATA_10, DES_DATA_10: std_logic;
  signal DES_RES_10: std_logic_vector(1 downto 0);

  signal DES_EXTIN_01, DES_OUT_01, DES_STORE_01, DES_NEWDATA_01: std_logic;
  signal DES_DATA_01, DES_RES_01: std_logic_vector(1 downto 0);

  signal DES_EXTIN_11, DES_OUT_11, DES_STORE_11, DES_NEWDATA_11, DES_COUT_11: std_logic;
  signal DES_DATA_11, DES_RES_11: std_logic_vector(1 downto 0);


begin

  DUT: memory port map (
    RESET_n => RESET_n,
    CURRENTclk => CURRENTclk,
    ---
    DES_EXTIN_00 => DES_EXTIN_00,
    DES_OUT_00 => DES_OUT_00,
    DES_STORE_00 => DES_STORE_00,
    DES_NEWDATA_00 => DES_NEWDATA_00,
    DES_RES_00 => DES_RES_00,
    ---
    DES_EXTIN_10 => DES_EXTIN_10,
    DES_OUT_10 => DES_OUT_10,
    DES_STORE_10 => DES_STORE_10,
```

```
    DES_NEWDATA_10 => DES_NEWDATA_10,
    DES_DATA_10 => DES_DATA_10,
    DES_RES_10 => DES_RES_10,
    ---
    DES_EXTIN_01 => DES_EXTIN_01,
    DES_OUT_01 => DES_OUT_01,
    DES_STORE_01 => DES_STORE_01,
    DES_NEWDATA_01 => DES_NEWDATA_01,
    DES_DATA_01 => DES_DATA_01,
    DES_RES_01 => DES_RES_01,
    ---
    DES_EXTIN_11 => DES_EXTIN_11,
    DES_OUT_11 => DES_OUT_11,
    DES_STORE_11 => DES_STORE_11,
    DES_NEWDATA_11 => DES_NEWDATA_11,
    DES_COUT_11 => DES_COUT_11,
    DES_DATA_11 => DES_DATA_11,
    DES_RES_11 => DES_RES_11
  );

  RESET_n <= '0', '1' after 50 ps;

  DES_EXTIN_00 <= '1';
  DES_OUT_00 <= '1';
  DES_STORE_00 <= '0';
  DES_NEWDATA_00 <= '1';
  DES_RES_00 <= "01";

  DES_EXTIN_10 <= '0';
  DES_OUT_10 <= '1';
  DES_STORE_10 <= '1';
  DES_NEWDATA_10 <= '0';
  DES_DATA_10 <= '0';
  DES_RES_10 <= "10";

  DES_EXTIN_01 <= '1';
  DES_OUT_01 <= '0';
  DES_STORE_01 <= '1';
  DES_NEWDATA_01 <= '1';
  DES_DATA_01 <= "10", "11" after 480 ns, "00" after 530 ns;
  --DES_DATA_01 <= "10";
  DES_RES_01 <= "10";

  DES_EXTIN_11 <= '0';
  DES_OUT_11 <= '1';
  DES_STORE_11 <= '1';
  DES_NEWDATA_11 <= '0';
  DES_COUT_11 <= '1', '0' after 640 ns;
  DES_DATA_11 <= "10", "11" after 640 ns;
```

```
  DES_RES_11 <= "01";

  CURRENT_GEN : process
  begin
    CURRENTclk <= CURRENT_LOW;
    wait for CLOCK_LOW;
    CURRENTclk <= CURRENT_HIGH;
    wait for CLOCK_HIGH;
  end process;

end architecture Structure;
```

D. Logic in memory VHDL code - architecture 2

D.1. Array components and related files

D.1.1. Components

D.1.1.1. Standard voltage generator

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity voltage_genL is
port(  CTRL: in std_logic;
    CURRENT: out real);
end entity voltage_genL;

architecture Behavioural of voltage_genL is
begin
  CURR_GEN: process (CTRL)
  begin
    if (CTRL='1') then
      CURRENT <= CURRENT_LOW;
    else
      CURRENT <= 0.0;
    end if;
  end process CURR_GEN;
end architecture Behavioural;
```

D.1.1.2. V_{op} generator

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity voltage_genPEAK is
port(  CTRL: in std_logic;
    CURRENT: out real);
end entity voltage_genPEAK;

architecture Behavioural of voltage_genPEAK is
begin
  CURR_GEN: process (CTRL)
  begin
    if (CTRL='1') then
      CURRENT <= CURRENT_LOW, CURRENT_HIGH after CLOCK_PERIOD/2, CURRENT_LOW after CLOCK_PERIOD/2+CLOCK_HIGH;
    else
      CURRENT <= 0.0;
    end if;
  end process CURR_GEN;
end architecture Behavioural;
```

D.1.1.3. And/Or gate

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SKYRMIONH is
port (  INPUTA : in std_logic;
    INPUTB : in std_logic;
    CURRENT : in real;
    OUTPUTAND : out std_logic;
```

```vhdl
    OUTPUTOR : out std_logic
);
end entity SKYRMIONH;

architecture CENTRALLOGIC of skyrmionh is
  ------------ CONSTANTS --------------------------------------------
  constant TRACK_LENGTH : real := 256.0;    --nm
  constant HOLE_X_START : real := 113.0;    --nm
  constant HOLE_X_END : real := 143.0;    --nm
  constant HOLE_X_POSITION : real := 128.0;  --nm
  constant TRACK_0_Y : real := 10.0;      --nm
  constant TRACK_1_Y : real := 50.0;      --nm
  constant HOLE_Y_BOTTOM : real := 20.0;    --nm
  constant HOLE_Y_TOP : real := 40.0;      --nm

  ----------- FUNCTIONS ---------------------------------------------

  function  updatePosition (elapsedTimeNs: real; actualPosition: parameters_array; currentValue : real; index: integer) return coordinates_xy is
    variable speed : coordinates_xy;
    variable output : coordinates_xy;
    variable changeTrack : boolean;
  begin
    changeTrack := false;
    output(0) := 0.0;
    output(1) := 0.0;
    if(currentValue > DEPINNING_CURRENT) then
      speed(1) := 0.0;
      speed(0) := 0.0;
      if(actualPosition(index)(0) > HOLE_X_START and  actualPosition(index)(0) < HOLE_X_END and actualPosition(index)(1) <= HOLE_Y_BOTTOM) then
        changeTrack := true;
        for i in 0 to 9 loop
          if(index /= i and actualPosition(i)(0) > HOLE_X_START and actualPosition(i)(0) < HOLE_X_END and actualPosition(i)(1) > 0.0) then
            changeTrack := false;
          end if;
        end loop;
      end if;
      if (changeTrack or (actualPosition(index)(1) > HOLE_Y_BOTTOM and actualPosition(index)(1) < HOLE_Y_TOP)) then
        speed(1) := VERTICAL_SPEED;
        speed(0) := 0.0;
      else
        speed(1) := 0.0;
        speed(0) := HORIZONTAL_SPEED;
      end if;
```

```
      output(0) := actualPosition(index)(0)+speed(0)*elapsedTimeNs;
      output(1) := actualPosition(index)(1)+speed(1)*elapsedTimeNs;
    end if;
    return output;
  end updatePosition;


  ------------ SIGNALS ---------------------------------
  signal ACK : std_logic := '0';
  signal inputPortState, emit : std_logic_vector(1 downto 0) := "00";
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer := 0;
begin

  RECEIVER: process(INPUTA, INPUTB, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= "00";
    end if;
    if (INPUTA'event and INPUTA='1') then
      inputPortState(1) <= '1';
    end if;
    if (INPUTB'event and INPUTB='1') then
      inputPortState(0) <= '1';
    end if;
  end process;


  EMITTER: process(emit)
  begin
    if(emit(0)'event and emit(0)='1') then
      OUTPUTAND<='1';
    else
      OUTPUTAND<='0' after 1 ns;
    end if;
    if(emit(1)'event and emit(1)='1') then
      OUTPUTOR<='1';
    else
      OUTPUTOR<='0' after 1 ns;
    end if;
  end process;

  EVOLUTION:process
    variable v_TIME : time := 0 ns;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;
    variable skyrmion_number, skyrmion_number_old, write_index : integer := 0;
    variable result : coordinates_xy;
    variable timeNsReal : real := 0.0;
```

```vhdl
    variable trackBusy : bool_array(1 downto 0);
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
    timeNsReal := 0.01;
    trackBusy(0) := false;
    trackBusy(1) := false;
    ACK <= '0';
    if (inputPortState(0) = '1') then --skyrmion detected on B
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := TRACK_0_Y;
      ACK <= '1';
    end if;

    if (inputPortState(1) = '1') then --skyrmion detected on A
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := TRACK_1_Y;
      ACK <= '1';
    end if;

    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      for i in 0 to skyrmion_number_old-1 loop
        result := updatePosition(timeNsReal,skyrmion_position_old,CURRENT,i);
        if (result(0) > TRACK_LENGTH ) then
          skyrmion_number := skyrmion_number-1;
          if result(1) > HOLE_Y_TOP then
            emit(1) <= '1' after 5 ps;  --OR=1
            trackBusy(1) := true;
          else
            emit(0) <= '1' after 5 ps;  --AND=1
            trackBusy(0) := true;
          end if;
        else
          write_index := write_index + 1;
          skyrmion_position(write_index) := result;
        end if;

      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
```

```
      end if;

      if (not(trackBusy(0))) then
        emit(0) <= '0' after 5 ps;
      end if;
      if (not(trackBusy(1))) then
        emit(1) <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= "00" after 15 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
  end process;
end CENTRALLOGIC;
```

D.1.1.4. Join element

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use work.globals.all;

entity SKYRMIONJOIN is
  port( A : in std_logic;
        B : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic
      );
end entity SKYRMIONJOIN;

architecture BLACKBOX of SKYRMIONJOIN is

  ------------ CONSTANTS -------------------------------------------
```

```
  constant TRACK_LENGTH : real := 256.0;  --nm

  ------------ INTERNAL SIGNALS ------------------------------------
  signal emit : std_logic := '0';
  signal inputPortState: std_logic_vector(1 downto 0):= "00";
  signal ACK : std_logic := '0';
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer;

  ----------- FUNCTIONS --------------------------------------------
  function  updatePosition (elapsedTimeNs: real; actualPosition:
  ↪  parameters_array; currentValue : real ) return parameters_array is
    variable output : parameters_array;
  begin
    if(currentValue > DEPINNING_CURRENT) then
      for i in 0 to 9 loop
        output(i)(1) := 0.0;
        output(i)(0) := actualPosition(i)(0) + HORIZONTAL_SPEED*elapsedTimeNs;
      end loop;
    end if;
    return output;
  end updatePosition;

begin

  RECEIVER: process(A, B, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= "00";
    end if;
    if (B'event and B='1') then
      inputPortState(0) <= '1';
    end if;
    if (A'event and A='1') then
      inputPortState(1) <= '1';
    end if;
  end process;


  EVOLUTION:process
    variable v_TIME : time := 0 ns;

    variable skyrmion_number : integer := 0;
    variable skyrmion_number_old : integer := 0;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;

    variable results : parameters_array;
    variable timeNsReal : real := 0.0;
```

```
    variable trackBusy : boolean;
    variable write_index : integer := 0;
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
    timeNsReal := 0.01;
    trackBusy := false;
    ACK <= '0';

    if (inputPortState(0) = '1') then
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;
    if (inputPortState(1) = '1') then
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;

    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      results := updatePosition(timeNsReal, skyrmion_position_old, CURRENT);
      for i in 0 to skyrmion_number_old-1 loop
        if (results(i)(0) > TRACK_LENGTH and not(trackBusy)) then
          skyrmion_number := skyrmion_number-1;
          emit <= '1' after 5 ps;
          trackBusy := true;
        else
          if(results(i)(0) > TRACK_LENGTH and trackBusy) then
            report "More than one skyrmion reached the output in this step; Join
            ↪ gate does not account the skyrmion collisions yet. The skyrmions
            ↪ will be emitted in sequence with one step distance";
          end if;
          write_index := write_index + 1;
          skyrmion_position(write_index) := results(i);
        end if;
      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;
```

```

      if (not(trackBusy)) then
        emit <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= '0' after 5 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
    wait for 5 ps;
  end process;


  EMITTER: process(emit)
  begin

    if(emit'event and emit='1') then
      OUTPUT<='1';
    else
      OUTPUT<='0' after 10 ps;
    end if;
  end process;
end BLACKBOX;
```

D.1.1.5. Notch element

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use work.globals.all;
use work.all;

entity SKYRMIONNOTCH is
  port( INPUT : in std_logic;
        CURRENT : in real;
```

```
        OUTPUT : out std_logic
      );
end entity SKYRMIONNOTCH;

architecture BLACKBOX of SKYRMIONNOTCH is
  ------------ CONSTANTS --------------------------------------------
  constant TRACK_LENGTH : real := 256.0;  --nm
  constant NOTCH_POSITION: real := 113.0;  --nm

  ------------ INTERNAL SIGNALS ------------------------------------
  signal emit : std_logic := '0';
  signal inputPortState: std_logic:= '0';
  signal ACK : std_logic := '0';
  signal skyrmion_position_debug : parameters_array;
  signal skyrmion_number_debug : integer;

  ----------- FUNCTIONS --------------------------------------------
  function findSkyrmionsCloserToNotch (notch_distance: real_array(9 downto 0);
  ↪  index: integer) return integer is
  variable output : integer := 0;
  begin
    for i in 0 to 9 loop
      if(notch_distance(index) < notch_distance(i)) then
        output := output + 1;
      end if;
    end loop;
    return output;
  end findSkyrmionsCloserToNotch;


  function  updatePosition (elapsedTimeNs: real; actualPosition:
  ↪  parameters_array; currentValue : real ) return parameters_array is
    variable speed : coordinates_xy;
    variable output : parameters_array;
    variable blocking_skyrmions : integer;
    variable notch_distance : real_array(9 downto 0);
    variable delta_distance : real;
  begin
    if(currentValue > DEPINNING_CURRENT) then
      if (currentValue < NOTCH_DEPINNING_CURRENT) then
        for i in 0 to 9 loop
          output(i)(1) := 0.0;
          notch_distance(i) := actualPosition(i)(0)-NOTCH_POSITION;
          delta_distance := HORIZONTAL_SPEED*elapsedTimeNs;
          if(notch_distance(i) > 0.0) then
            output(i)(0) := actualPosition(i)(0)+delta_distance;
          else
            blocking_skyrmions := findSkyrmionsCloserToNotch(notch_distance, i);
            if (abs(notch_distance(i)) - real(blocking_skyrmions) *
            ↪  (SKYRMION_DIAMETER + SKYRMION_MIN_DISTANCE) > delta_distance)
            ↪  then
```

```
              output(i)(0) := actualPosition(i)(0)+ delta_distance;
            else
              output(i)(0) := NOTCH_POSITION - real(blocking_skyrmions) * (SKYRMION_DIAMETER + SKYRMION_MIN_DISTANCE);
            end if;
          end if;
        end loop;
      else
        for i in 0 to 9 loop
          output(i)(1) := 0.0;
          output(i)(0) := actualPosition(i)(0) + HORIZONTAL_SPEED_HIGH*elapsedTimeNs;
        end loop;
      end if;
    end if;
    return output;
  end updatePosition;

begin

  RECEIVER: process(INPUT, ACK)
  begin
    if (ACK'event and ACK='1') then
      inputPortState <= '0';
    end if;
    if (INPUT'event and INPUT='1') then
      inputPortState <= '1';
    end if;
  end process;


  EVOLUTION:process
    variable v_TIME : time := 0 ns;
    variable skyrmion_number : integer := 0;
    variable skyrmion_number_old : integer := 0;
    variable skyrmion_position : parameters_array;
    variable skyrmion_position_old : parameters_array;
    variable results : parameters_array;
    variable timeNsReal : real := 0.0;
    variable trackBusy : boolean;
    variable write_index : integer := 0;
  begin
    wait for 5 ps;
    v_TIME := now - v_TIME;
    timeNsReal := 0.01;
    trackBusy := false;
    ACK <= '0';

    if (inputPortState = '1') then
```

```
      skyrmion_number := skyrmion_number +1;
      skyrmion_position(skyrmion_number-1)(0) := 0.0;
      skyrmion_position(skyrmion_number-1)(1) := 0.0;
      ACK <= '1';
    end if;

    if (skyrmion_number>0 and CURRENT>DEPINNING_CURRENT) then
      skyrmion_position_old := skyrmion_position;
      skyrmion_number_old := skyrmion_number;
      write_index := -1;
      results := updatePosition(timeNsReal, skyrmion_position_old, CURRENT);
      for i in 0 to skyrmion_number_old-1 loop
        if (results(i)(0) > TRACK_LENGTH and not(trackBusy)) then
          skyrmion_number := skyrmion_number-1;
          emit <= '1' after 5 ps;
          trackBusy := true;
        else
          if(results(i)(0) > TRACK_LENGTH and trackBusy) then
            report "More than one skyrmion reached the output in this step; Try
            ↪ reducing the simulation step; The second skyrmion to reach the
            ↪ output will be delayed by one step";
          end if;
          write_index := write_index + 1;
          skyrmion_position(write_index) := results(i);
        end if;
      end loop;
      if (write_index < 9) then
        write_index := write_index+1;
        for i in write_index to 9 loop
          skyrmion_position(i)(0) := 0.0;
          skyrmion_position(i)(1) := 0.0;
        end loop;
      end if;

      if (not(trackBusy)) then
        emit <= '0' after 5 ps;
      end if;
    elsif (skyrmion_number=0) then
      emit <= '0' after 5 ps;
      for i in 0 to 9 loop
        skyrmion_position(i)(0) := 0.0;
        skyrmion_position(i)(1) := 0.0;
      end loop;
    else
      report "Skyrmion number exceeded maximum admitted";
    end if;

    skyrmion_position_debug <= skyrmion_position after 5 ps;
    skyrmion_number_debug <= skyrmion_number after 5 ps;
```

```
    wait for 5 ps;
  end process;


  EMITTER: process(emit)
  begin

    if(emit'event and emit='1') then
      OUTPUT<='1';
    else
      OUTPUT<='0' after 10 ps;
    end if;
  end process;
end BLACKBOX;
```

D.1.1.6. Duplication element

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity skyrmionDUPLICATE is
port(  IN_SK:       in std_logic;
    CURRENT:    in real;
    OUT_SK_TOP:     out std_logic;
    OUT_SK_BOTTOM:  out std_logic);
end entity skyrmionDUPLICATE;

architecture Behavioural of skyrmionDUPLICATE is
begin
  SK_DUPL: process (IN_SK, CURRENT)
    variable Nsk: integer := 0;
  begin
    if (IN_SK'event and IN_SK='0') then
      Nsk := Nsk+1;
    end if;

    if (CURRENT /= 0.0) then
      if (Nsk /= 0) then
        OUT_SK_TOP <= '1' after 1 ps, '0' after 11 ps;
```

```
        OUT_SK_BOTTOM <= '1' after 1 ps, '0' after 11 ps;
        Nsk:=Nsk-1;
      else
        OUT_SK_TOP <= '0';
        OUT_SK_BOTTOM <= '0';
      end if;
    else
      OUT_SK_TOP <= '0';
      OUT_SK_BOTTOM <= '0';
    end if;
  end process SK_DUPL;
end architecture Behavioural;
```

D.1.1.7. Cross element

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity SKYRMIONCROSS is
port(    A:        in std_logic;
    B:        in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end entity SKYRMIONCROSS;

architecture BLACKBOX of SKYRMIONCROSS is

begin
  process (A, B, CURRENTA, CURRENTB) is
    variable NskA, NskB: integer := 0;
  begin
    if (A'event and A='1') then
      NskA := NskA+1;
    end if;
    if (B'event and B='1') then
      NskB := NskB+1;
    end if;

    if (CURRENTA /= 0.0) then
```

```
      if (NskA = 1) then
        Aout <= '1', '0' after 9 ps;
        NskA := 0;
      else
        Aout <= '0';
      end if;
    end if;
    if (CURRENTB /= 0.0) then
      if (NskB = 1) then
        Bout <= '1', '0' after 9 ps;
        NskB := 0;
      else
        Bout <= '0';
      end if;
    end if;
  end process;
end BLACKBOX;
```

D.1.1.8. Read head

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity MTJ_R is
port(  IN_SK:     in std_logic;
    CURRENT:  in real;
    OUT_SIGN:  out std_logic);
end entity MTJ_R;

architecture Behavioural of MTJ_R is
begin
  SK_DETECT: process (IN_SK, CURRENT)
    variable Nsk: std_logic := '0';
  begin
    if (IN_SK'event and IN_SK='1') then
      Nsk := '1';
    end if;
    if (CURRENT /= 0.0) then
      if (Nsk='1') then
```

```
      --OUT_SIGN <= '0', '1' after 10 ps, '0' after 20 ps;
      OUT_SIGN <= '1', '0' after 10 ps;
      Nsk:='0';
    else
      OUT_SIGN <= '0';
    end if;
  else
    OUT_SIGN <= '0';
  end if;
 end process SK_DETECT;
end architecture Behavioural;
```

D.1.1.9. Write head

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity MTJ_W is
port(  CTRL: in std_logic;
    OUT_SK:  out std_logic);
end entity MTJ_W;

architecture Behavioural of MTJ_W is
begin
  SK_GEN: process (CTRL)
  begin
    if (CTRL'event and CTRL='1') then
      OUT_SK <= '1', '0' after 10 ps;
    else
      OUT_SK <= '0';
    end if;
  end process SK_GEN;
end architecture Behavioural;
```

D.1.1.10. *Dxx* component

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;

entity cell_input is
port(  IN_SK_L:   in std_logic;
    IN_SK_R:   in std_logic;
    IN_SK_T:   in std_logic;
    ENABLE:    in std_logic;
    CURRENT_T:  in real;
    CURRENT_L:  in real;
    CURRENT_R:  in real;
    OUT_SK_L:  out std_logic;
    OUT_SK_R:  out std_logic;
    OUT_SK_B:  out std_logic);
end entity cell_input;

architecture Behavioural of cell_input is
signal Nsks: integer;
signal outens: std_logic;
begin
  SK_DEV: process (IN_SK_L, IN_SK_R, IN_SK_T, CURRENT_T, CURRENT_L, CURRENT_R)
    variable Nsk: integer := 0;
    variable out_en: std_logic := '0';
    variable outr_en: std_logic := '1';
  begin
    if (ENABLE='1') then
      if (IN_SK_L'event and IN_SK_L='1') then
        Nsk := Nsk+1;
      end if;
      if (IN_SK_R'event and IN_SK_R='1') then
        Nsk := Nsk+1;
      end if;
      if (IN_SK_T'event and IN_SK_T='1') then
        Nsk := Nsk+1;
      end if;

      if (CURRENT_T = 0.0 and CURRENT_L = 0.0 and CURRENT_R = 0.0) then
        OUT_SK_B <= '0';
        OUT_SK_L <= '0';
        OUT_SK_R <= '0';
      end if;

```

```
      if(out_en = '1') then
        if (CURRENT_L'event and CURRENT_L /= 0.0) then
          if (Nsk=1) then
            OUT_SK_B <= '0';
            OUT_SK_L <= '1', '0' after 10 ps;
            OUT_SK_R <= '0';
            Nsk := Nsk-1;
            out_en := '0';
          else
            OUT_SK_B <= '0';
            OUT_SK_L <= '0';
            OUT_SK_R <= '0';
            out_en := '0';
          end if;
        elsif (CURRENT_R'event and CURRENT_R /= 0.0 and outr_en = '1') then
          if (Nsk=1) then
            OUT_SK_B <= '0';
            OUT_SK_L <= '0';
            OUT_SK_R <= '1', '0' after 10 ps;
            Nsk := Nsk-1;
            out_en := '0';
            outr_en := '0';
          else
            OUT_SK_B <= '0';
            OUT_SK_L <= '0';
            OUT_SK_R <= '0';
            out_en := '0';
            outr_en := '0';
          end if;
        elsif (CURRENT_T'event and CURRENT_T /= 0.0) then
          if (Nsk=1) then
            OUT_SK_B <= '1', '0' after 10 ps;
            OUT_SK_L <= '0';
            OUT_SK_R <= '0';
            Nsk := Nsk-1;
            out_en := '0';
          else
            OUT_SK_B <= '0';
            OUT_SK_L <= '0';
            OUT_SK_R <= '0';
            out_en := '0';
          end if;
        end if;
      end if;

      if (CURRENT_T'event and CURRENT_T /= 0.0) then
        if (out_en = '0' and Nsk /= 0) then
          out_en:='1';
        end if;
```

```
      end if;

      if (CURRENT_R'event and CURRENT_R = 0.0) then
        if (outr_en='0') then
          outr_en := '1';
        end if;
      end if;
    else
      OUT_SK_B <= '0';
      OUT_SK_L <= '0';
      OUT_SK_R <= '0';
    end if;

    Nsks <= Nsk;
    outens <= out_en;

  end process SK_DEV;
end architecture Behavioural;
```

D.1.2. Cells of row 0

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Cell0 is
port(  IN_R:     in std_logic;
    IN_L:     in std_logic;
    CTRL_MTJ:  in std_logic;
    CURRENT_OP:  in real;
    CURRENT_TR:  in real;
    OUTFEED_R:  out std_logic;
    OUTFEED_L:  out std_logic;
    OUTTRAN:  out std_logic;
    OUTOR:    out std_logic;
    OUTOP:    out std_logic);
end entity Cell0;

architecture Behaviour of Cell0 is
  component skyrmionDUPLICATE is
  port(  IN_SK:       in std_logic;
      CURRENT:    in real;
```

```
        OUT_SK_TOP:    out std_logic;
        OUT_SK_BOTTOM:  out std_logic);
    end component skyrmionDUPLICATE;

    component SKYRMIONNOTCH is
    port( INPUT : in std_logic;
        CURRENT : in real;
        OUTPUT : out std_logic);
    end component SKYRMIONNOTCH;

    component MTJ_W is
    port(  CTRL: in std_logic;
        OUT_SK:  out std_logic);
    end component MTJ_W;

    component SKYRMIONH is
    port (  INPUTA : in std_logic;
        INPUTB : in std_logic;
        CURRENT : in real;
        OUTPUTAND : out std_logic;
        OUTPUTOR : out std_logic);
    end component SKYRMIONH;

    signal xRout_bottom, MTJout, notchtop_out, notchbot_out: std_logic;

begin
    xR:     skyrmionDUPLICATE port map (IN_SK => IN_R, CURRENT => CURRENT_OP,
    ↪  OUT_SK_TOP => OUTFEED_R, OUT_SK_BOTTOM => xRout_bottom);
    MTJ:     MTJ_W port map (CTRL => CTRL_MTJ, OUT_SK => MTJout);
    notchtop:   SKYRMIONNOTCH port map (INPUT => xRout_bottom, CURRENT =>
    ↪  CURRENT_OP, OUTPUT => notchtop_out);
    notchbot:   SKYRMIONNOTCH port map (INPUT => MTJout, CURRENT => CURRENT_OP,
    ↪  OUTPUT => notchbot_out);
    ANDc:    SKYRMIONH port map (INPUTA => notchtop_out, INPUTB => notchbot_out,
    ↪  CURRENT => CURRENT_OP, OUTPUTAND => OUTOP, OUTPUTOR => OUTOR);

    xL:     skyrmionDUPLICATE port map (IN_SK => IN_L, CURRENT => CURRENT_TR,
    ↪  OUT_SK_TOP => OUTFEED_L, OUT_SK_BOTTOM => OUTTRAN);

end architecture Behaviour;
```

D.1.3. Cells of any other row

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Cellx is
port(  IN_R:     in std_logic;
    IN_L:     in std_logic;
    IN_TR:    in std_logic;
    CTRL_MTJ:  in std_logic;
    CURRENT_OP:  in real;
    CURRENT_TR:  in real;
    OUTFEED_R:  out std_logic;
    OUTFEED_L:  out std_logic;
    OUTTRAN:  out std_logic;
    OUTOR:    out std_logic;
    OUTOP:    out std_logic
);
end entity Cellx;

architecture Behaviour of Cellx is
  component skyrmionDUPLICATE is
  port(  IN_SK:       in std_logic;
      CURRENT:    in real;
      OUT_SK_TOP:    out std_logic;
      OUT_SK_BOTTOM:  out std_logic);
  end component skyrmionDUPLICATE;

  component SKYRMIONNOTCH is
  port( INPUT : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONNOTCH;

  component MTJ_W is
  port(  CTRL: in std_logic;
      OUT_SK:  out std_logic);
  end component MTJ_W;

  component SKYRMIONH is
  port (  INPUTA : in std_logic;
      INPUTB : in std_logic;
      CURRENT : in real;
      OUTPUTAND : out std_logic;
      OUTPUTOR : out std_logic);
```

```
  end component SKYRMIONH;

  component SKYRMIONJOIN is
  port( A : in std_logic;
      B : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
  end component SKYRMIONJOIN;

  signal xRout_bottom, MTJout, notchtop_out, notchbot_out, JOIN_out: std_logic;

begin
  xR:      skyrmionDUPLICATE port map (IN_SK => IN_R, CURRENT => CURRENT_OP, OUT_SK_TOP => OUTFEED_R, OUT_SK_BOTTOM => xRout_bottom);
  MTJ:      MTJ_W port map (CTRL => CTRL_MTJ, OUT_SK => MTJout);
  join:     SKYRMIONJOIN port map (A => MTJout, B => IN_TR, CURRENT => CURRENT_OP, OUTPUT => JOIN_out);
  notchtop:   SKYRMIONNOTCH port map (INPUT => xRout_bottom, CURRENT => CURRENT_OP, OUTPUT => notchtop_out);
  notchbot:   SKYRMIONNOTCH port map (INPUT => JOIN_out, CURRENT => CURRENT_OP, OUTPUT => notchbot_out);
  ANDc:     SKYRMIONH port map (INPUTA => notchtop_out, INPUTB => notchbot_out, CURRENT => CURRENT_OP, OUTPUTAND => OUTOP, OUTPUTOR => OUTOR);

  xL:      skyrmionDUPLICATE port map (IN_SK => IN_L, CURRENT => CURRENT_TR, OUT_SK_TOP => OUTFEED_L, OUT_SK_BOTTOM => OUTTRAN);

end architecture Behaviour;
```

D.1.4. Word 0

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Word0 is
port(  CTRL_Vbl:       in std_logic;
    CTRL_Vop:      in std_logic;
    CTRL_Vtr0:      in std_logic;
    CTRL_Vtr1:      in std_logic;
    CTRL_Vtr2:      in std_logic;
    CTRL_MTJW:       in std_logic;
```

```
    CTRL_in0_EN:     in std_logic;
    CTRL_in1_EN:     in std_logic;
    CTRL_in2_EN:     in std_logic;
    CTRL_MTJ_cell0:    in std_logic;
    CTRL_MTJ_cell1:    in std_logic;
    CTRL_MTJ_cell2:    in std_logic;
    BL_out:        out std_logic;
    MTJ_RR0_out:    out std_logic;
    MTJ_R0_out:      out std_logic;
    MTJ_R1_out:      out std_logic;
    MTJ_R2_out:      out std_logic;
    CELL0_tran:      out std_logic;
    CELL1_tran:      out std_logic;
    CELL2_tran:      out std_logic;
    CELL0_tran_CURR:  out real;
    CELL1_tran_CURR:  out real;
    CELL2_tran_CURR:  out real);
end entity Word0;

architecture Behaviour of Word0 is
  component Cell0 is
  port(  IN_R:     in std_logic;
      IN_L:     in std_logic;
      CTRL_MTJ:  in std_logic;
      CURRENT_OP:  in real;
      CURRENT_TR:  in real;
      OUTFEED_R:  out std_logic;
      OUTFEED_L:  out std_logic;
      OUTTRAN:  out std_logic;
      OUTOR:    out std_logic;
      OUTOP:    out std_logic);
  end component Cell0;

  component MTJ_W is
  port(  CTRL: in std_logic;
      OUT_SK:  out std_logic);
  end component MTJ_W;

  component MTJ_R is
  port(  IN_SK:     in std_logic;
      CURRENT:  in real;
      OUT_SIGN:  out std_logic);
  end component MTJ_R;

  component SKYRMIONJOIN is
  port( A : in std_logic;
      B : in std_logic;
      CURRENT : in real;
      OUTPUT : out std_logic);
```

```
end component SKYRMIONJOIN;

component voltage_genL is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genL;

component voltage_genPEAK is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genPEAK;

component cell_input is
port(  IN_SK_L:   in std_logic;
    IN_SK_R:   in std_logic;
    IN_SK_T:   in std_logic;
    ENABLE:    in std_logic;
    CURRENT_T:  in real;
    CURRENT_L:  in real;
    CURRENT_R:  in real;
    OUT_SK_L:  out std_logic;
    OUT_SK_R:  out std_logic;
    OUT_SK_B:  out std_logic);
end component cell_input;

component SKYRMIONCROSS is
port(   A:       in std_logic;
    B:       in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end component SKYRMIONCROSS;

signal CURRENT_Vbl, CURRENT_Vop, CURRENT_Vtr0, CURRENT_Vtr1, CURRENT_Vtr2: real;
signal MTJW_out: std_logic;
---
signal cell0_IN_R, cell0_IN_L, cell0_OUTFEED_R, cell0_OUTFEED_L, cell0_OUTTRAN, cell0_OUTOR, cell0_OUTOP: std_logic;
signal D0_Bout, C0L_Aout, C0L_Bout, C0R_Aout, C0R_Bout: std_logic;
---
signal cell1_IN_R, cell1_IN_L, cell1_OUTFEED_R, cell1_OUTFEED_L, cell1_OUTTRAN, cell1_OUTOR, cell1_OUTOP: std_logic;
signal D1_Bout, C1L_Aout, C1L_Bout, C1R_Aout, C1R_Bout, JOIN1_out: std_logic;
---
signal cell2_IN_R, cell2_IN_L, cell2_OUTFEED_R, cell2_OUTFEED_L, cell2_OUTTRAN, cell2_OUTOR, cell2_OUTOP: std_logic;
signal D2_Bout, C2L_Aout, C2L_Bout, C2R_Aout, C2R_Bout, JOIN2_out: std_logic;
```

```

begin
  Vbl:     voltage_genL port map (CTRL => CTRL_Vbl, CURRENT => CURRENT_Vbl);
  Vop:    voltage_genPEAK port map (CTRL => CTRL_Vop, CURRENT => CURRENT_Vop);
  MTJW:     MTJ_W port map (CTRL => CTRL_MTJW, OUT_SK => MTJW_out);

  Vtr0:    voltage_genL port map (CTRL => CTRL_Vtr0, CURRENT => CURRENT_Vtr0);
  input0:    cell_input port map (IN_SK_L => cell0_OUTFEED_L, IN_SK_R => cell0_OUTFEED_R, IN_SK_T => MTJW_out, ENABLE => CTRL_in0_EN, CURRENT_T => CURRENT_Vbl, CURRENT_L => CURRENT_Vtr0, CURRENT_R => CURRENT_Vop, OUT_SK_L => cell0_IN_L, OUT_SK_R => cell0_IN_R, OUT_SK_B => D0_Bout);
  cell0c:    Cell0 port map (IN_R => cell0_IN_R, IN_L => cell0_IN_L, CTRL_MTJ => CTRL_MTJ_cell0, CURRENT_OP => CURRENT_Vop, CURRENT_TR => CURRENT_Vtr0, OUTFEED_R => cell0_OUTFEED_R, OUTFEED_L => cell0_OUTFEED_L, OUTTRAN => cell0_OUTTRAN, OUTOR => cell0_OUTOR, OUTOP => cell0_OUTOP);
  C0L:    SKYRMIONCROSS port map (A => cell0_OUTTRAN, B => D0_Bout, CURRENTA => CURRENT_Vtr0, CURRENTB => CURRENT_Vbl, Aout => C0L_Aout, Bout => C0L_Bout);
  C0R:    SKYRMIONCROSS port map (A => C0L_Aout, B => cell0_OUTOP, CURRENTA => CURRENT_Vtr0, CURRENTB => CURRENT_Vop, Aout => C0R_Aout, Bout => C0R_Bout);
  MTJR0:    MTJ_R port map (IN_SK => cell0_OUTOR, CURRENT => CURRENT_Vop, OUT_SIGN => MTJ_R0_out);

  Vtr1:    voltage_genL port map (CTRL => CTRL_Vtr1, CURRENT => CURRENT_Vtr1);
  input1:    cell_input port map (IN_SK_L => cell1_OUTFEED_L, IN_SK_R => cell1_OUTFEED_R, IN_SK_T => C0L_Bout, ENABLE => CTRL_in1_EN, CURRENT_T => CURRENT_Vbl, CURRENT_L => CURRENT_Vtr1, CURRENT_R => CURRENT_Vop, OUT_SK_L => cell1_IN_L, OUT_SK_R => cell1_IN_R, OUT_SK_B => D1_Bout);
  cell1:    Cell0 port map (IN_R => cell1_IN_R, IN_L => cell1_IN_L, CTRL_MTJ => CTRL_MTJ_cell1, CURRENT_OP => CURRENT_Vop, CURRENT_TR => CURRENT_Vtr1, OUTFEED_R => cell1_OUTFEED_R, OUTFEED_L => cell1_OUTFEED_L, OUTTRAN => cell1_OUTTRAN, OUTOR => cell1_OUTOR, OUTOP => cell1_OUTOP);
  C1L:    SKYRMIONCROSS port map (A => cell1_OUTTRAN, B => D1_Bout, CURRENTA => CURRENT_Vtr1, CURRENTB => CURRENT_Vbl, Aout => C1L_Aout, Bout => C1L_Bout);
  join1:    SKYRMIONJOIN port map (A => cell1_OUTOP, B => C0R_Bout, CURRENT => CURRENT_Vop, OUTPUT => JOIN1_out);
  C1R:    SKYRMIONCROSS port map (A => C1L_Aout, B => JOIN1_out, CURRENTA => CURRENT_Vtr1, CURRENTB => CURRENT_Vop, Aout => C1R_Aout, Bout => C1R_Bout);
  MTJR1:    MTJ_R port map (IN_SK => cell1_OUTOR, CURRENT => CURRENT_Vop, OUT_SIGN => MTJ_R1_out);

  Vtr2:    voltage_genL port map (CTRL => CTRL_Vtr2, CURRENT => CURRENT_Vtr2);
  input2:    cell_input port map (IN_SK_L => cell2_OUTFEED_L, IN_SK_R => cell2_OUTFEED_R, IN_SK_T => C1L_Bout, ENABLE => CTRL_in2_EN, CURRENT_T => CURRENT_Vbl, CURRENT_L => CURRENT_Vtr2, CURRENT_R => CURRENT_Vop, OUT_SK_L => cell2_IN_L, OUT_SK_R => cell2_IN_R, OUT_SK_B => D2_Bout);
```

```
  cell2:    Cell0 port map (IN_R => cell2_IN_R, IN_L => cell2_IN_L, CTRL_MTJ => CTRL_MTJ_cell2, CURRENT_OP => CURRENT_Vop, CURRENT_TR => CURRENT_Vtr2, OUTFEED_R => cell2_OUTFEED_R, OUTFEED_L => cell2_OUTFEED_L, OUTTRAN => cell2_OUTTRAN, OUTOR => cell2_OUTOR, OUTOP => cell2_OUTOP);
  C2L:    SKYRMIONCROSS port map (A => cell2_OUTTRAN, B => D2_Bout, CURRENTA => CURRENT_Vtr2, CURRENTB => CURRENT_Vbl, Aout => C2L_Aout, Bout => C2L_Bout);
  join2:    SKYRMIONJOIN port map (A => cell2_OUTOP, B => C1R_Bout, CURRENT => CURRENT_Vop, OUTPUT => JOIN2_out);
  C2R:    SKYRMIONCROSS port map (A => C2L_Aout, B => JOIN2_out, CURRENTA => CURRENT_Vtr2, CURRENTB => CURRENT_Vop, Aout => C2R_Aout, Bout => C2R_Bout);
  MTJR2:    MTJ_R port map (IN_SK => cell2_OUTOR, CURRENT => CURRENT_Vop, OUT_SIGN => MTJ_R2_out);

  MTJRR:    MTJ_R port map (IN_SK => C2R_Bout, CURRENT => CURRENT_Vop, OUT_SIGN => MTJ_RR0_out);

  BL_out <= C2L_Bout;
  CELL0_tran <= C0R_Aout;
  CELL1_tran <= C1R_Aout;
  CELL2_tran <= C2R_Aout;
  CELL0_tran_CURR <= CURRENT_Vtr0;
  CELL1_tran_CURR <= CURRENT_Vtr1;
  CELL2_tran_CURR <= CURRENT_Vtr2;

end architecture Behaviour;
```

D.1.5. Any other word

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Wordx is
port(  CTRL_Vbl:       in std_logic;
    CTRL_Vop:      in std_logic;
    CTRL_Vtr0:      in std_logic;
    CTRL_Vtr1:      in std_logic;
    CTRL_Vtr2:      in std_logic;
    CTRL_MTJW:       in std_logic;
    CTRL_in0_EN:     in std_logic;
```

```
    CTRL_in1_EN:     in std_logic;
    CTRL_in2_EN:     in std_logic;
    CTRL_MTJ_cell0:    in std_logic;
    CTRL_MTJ_cell1:    in std_logic;
    CTRL_MTJ_cell2:    in std_logic;
    CELL0_intr:      in std_logic;
    CELL1_intr:      in std_logic;
    CELL2_intr:      in std_logic;
    CELL0_intr_CURR:  in real;
    CELL1_intr_CURR:  in real;
    CELL2_intr_CURR:  in real;
    BL_out:        out std_logic;
    MTJ_RRx_out:    out std_logic;
    MTJ_R0_out:    out std_logic;
    MTJ_R1_out:    out std_logic;
    MTJ_R2_out:    out std_logic;
    CELL0_tran:      out std_logic;
    CELL1_tran:      out std_logic;
    CELL2_tran:      out std_logic;
    CELL0_tran_CURR:  out real;
    CELL1_tran_CURR:  out real;
    CELL2_tran_CURR:  out real);
end entity Wordx;

architecture Behaviour of Wordx is
  component Cellx is
  port(  IN_R:     in std_logic;
      IN_L:     in std_logic;
      IN_TR:    in std_logic;
      CTRL_MTJ:  in std_logic;
      CURRENT_OP:  in real;
      CURRENT_TR:  in real;
      OUTFEED_R:  out std_logic;
      OUTFEED_L:  out std_logic;
      OUTTRAN:  out std_logic;
      OUTOR:    out std_logic;
      OUTOP:    out std_logic
  );
  end component Cellx;

  component MTJ_W is
  port(  CTRL: in std_logic;
      OUT_SK:  out std_logic);
  end component MTJ_W;

  component MTJ_R is
  port(  IN_SK:     in std_logic;
      CURRENT:  in real;
      OUT_SIGN:  out std_logic);
```

```
end component MTJ_R;

component SKYRMIONJOIN is
port( A : in std_logic;
    B : in std_logic;
    CURRENT : in real;
    OUTPUT : out std_logic);
end component SKYRMIONJOIN;

component voltage_genL is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genL;

component voltage_genPEAK is
port(  CTRL: in std_logic;
    CURRENT: out real);
end component voltage_genPEAK;

component cell_input is
port(  IN_SK_L:   in std_logic;
    IN_SK_R:   in std_logic;
    IN_SK_T:   in std_logic;
    ENABLE:    in std_logic;
    CURRENT_T:  in real;
    CURRENT_L:  in real;
    CURRENT_R:  in real;
    OUT_SK_L:  out std_logic;
    OUT_SK_R:  out std_logic;
    OUT_SK_B:  out std_logic);
end component cell_input;

component SKYRMIONCROSS is
port(   A:       in std_logic;
    B:       in std_logic;
    CURRENTA:   in real;
    CURRENTB:  in real;
    Aout:     out std_logic;
    Bout:     out std_logic);
end component SKYRMIONCROSS;

signal CURRENT_Vbl, CURRENT_Vop, CURRENT_Vtr0, CURRENT_Vtr1, CURRENT_Vtr2: real;
signal MTJW_out: std_logic;
---
signal cell0_IN_R, cell0_IN_L, cell0_OUTFEED_R, cell0_OUTFEED_L, cell0_OUTTRAN, cell0_OUTOR, cell0_OUTOP: std_logic;
signal D0_Bout, C0L_Aout, C0L_Bout, C0R_Aout, C0R_Bout, I0L_Aout, I0L_Bout, I0R_Aout, I0R_Bout: std_logic;
```

```
  ---
  signal cell1_IN_R, cell1_IN_L, cell1_OUTFEED_R, cell1_OUTFEED_L,
  ↪ cell1_OUTTRAN, cell1_OUTOR, cell1_OUTOP: std_logic;
  signal D1_Bout, C1L_Aout, C1L_Bout, C1R_Aout, C1R_Bout, I1L_Aout, I1L_Bout,
  ↪ I1R_Aout, I1R_Bout, JOIN1_out: std_logic;
  ---
  signal cell2_IN_R, cell2_IN_L, cell2_OUTFEED_R, cell2_OUTFEED_L,
  ↪ cell2_OUTTRAN, cell2_OUTOR, cell2_OUTOP: std_logic;
  signal D2_Bout, C2L_Aout, C2L_Bout, C2R_Aout, C2R_Bout, I2L_Aout, I2L_Bout,
  ↪ I2R_Aout, I2R_Bout, JOIN2_out: std_logic;

begin
  Vbl:     voltage_genL port map (CTRL => CTRL_Vbl, CURRENT => CURRENT_Vbl);
  Vop:    voltage_genPEAK port map (CTRL => CTRL_Vop, CURRENT => CURRENT_Vop);
  MTJW:     MTJ_W port map (CTRL => CTRL_MTJW, OUT_SK => MTJW_out);

  Vtr0:    voltage_genL port map (CTRL => CTRL_Vtr0, CURRENT => CURRENT_Vtr0);
  input0:    cell_input port map (IN_SK_L => cell0_OUTFEED_L, IN_SK_R =>
  ↪ cell0_OUTFEED_R, IN_SK_T => MTJW_out, ENABLE => CTRL_in0_EN, CURRENT_T =>
  ↪ CURRENT_Vbl, CURRENT_L => CURRENT_Vtr0, CURRENT_R => CURRENT_Vop, OUT_SK_L
  ↪ => cell0_IN_L, OUT_SK_R => cell0_IN_R, OUT_SK_B => D0_Bout);
  I0L:    SKYRMIONCROSS port map (A => CELL0_intr, B => cell0_OUTTRAN, CURRENTA
  ↪ => CELL0_intr_CURR, CURRENTB => CURRENT_Vtr0, Aout => I0L_Aout, Bout =>
  ↪ I0L_Bout);
  I0R:    SKYRMIONCROSS port map (A => I0L_Aout, B => D0_Bout, CURRENTA =>
  ↪ CELL0_intr_CURR, CURRENTB => CURRENT_Vbl, Aout => I0R_Aout, Bout =>
  ↪ I0R_Bout);
  cell0:    Cellx port map (IN_R => cell0_IN_R, IN_L => cell0_IN_L, IN_TR =>
  ↪ I0R_Aout, CTRL_MTJ => CTRL_MTJ_cell0, CURRENT_OP => CURRENT_Vop,
  ↪ CURRENT_TR => CURRENT_Vtr0, OUTFEED_R => cell0_OUTFEED_R, OUTFEED_L =>
  ↪ cell0_OUTFEED_L, OUTTRAN => cell0_OUTTRAN, OUTOR => cell0_OUTOR, OUTOP =>
  ↪ cell0_OUTOP);
  C0L:    SKYRMIONCROSS port map (A => I0L_Bout, B => I0R_Bout, CURRENTA =>
  ↪ CURRENT_Vtr0, CURRENTB => CURRENT_Vbl, Aout => C0L_Aout, Bout =>
  ↪ C0L_Bout);
  C0R:    SKYRMIONCROSS port map (A => C0L_Aout, B => cell0_OUTOP, CURRENTA =>
  ↪ CURRENT_Vtr0, CURRENTB => CURRENT_Vop, Aout => C0R_Aout, Bout =>
  ↪ C0R_Bout);
  MTJR0:    MTJ_R port map (IN_SK => cell0_OUTOR, CURRENT => CURRENT_Vop,
  ↪ OUT_SIGN => MTJ_R0_out);

  Vtr1:    voltage_genL port map (CTRL => CTRL_Vtr1, CURRENT => CURRENT_Vtr1);
  input1:    cell_input port map (IN_SK_L => cell1_OUTFEED_L, IN_SK_R =>
  ↪ cell1_OUTFEED_R, IN_SK_T => C0L_Bout, ENABLE => CTRL_in1_EN, CURRENT_T =>
  ↪ CURRENT_Vbl, CURRENT_L => CURRENT_Vtr1, CURRENT_R => CURRENT_Vop, OUT_SK_L
  ↪ => cell1_IN_L, OUT_SK_R => cell1_IN_R, OUT_SK_B => D1_Bout);
  I1L:    SKYRMIONCROSS port map (A => CELL1_intr, B => cell1_OUTTRAN, CURRENTA
  ↪ => CELL1_intr_CURR, CURRENTB => CURRENT_Vtr1, Aout => I1L_Aout, Bout =>
  ↪ I1L_Bout);
```

```
I1R:    SKYRMIONCROSS port map (A => I1L_Aout, B => D1_Bout, CURRENTA => CELL1_intr_CURR, CURRENTB => CURRENT_Vbl, Aout => I1R_Aout, Bout => I1R_Bout);
cell1:    Cellx port map (IN_R => cell1_IN_R, IN_L => cell1_IN_L, IN_TR => I1R_Aout, CTRL_MTJ => CTRL_MTJ_cell1, CURRENT_OP => CURRENT_Vop, CURRENT_TR => CURRENT_Vtr1, OUTFEED_R => cell1_OUTFEED_R, OUTFEED_L => cell1_OUTFEED_L, OUTTRAN => cell1_OUTTRAN, OUTOR => cell1_OUTOR, OUTOP => cell1_OUTOP);
C1L:    SKYRMIONCROSS port map (A => I1L_Bout, B => I1R_Bout, CURRENTA => CURRENT_Vtr1, CURRENTB => CURRENT_Vbl, Aout => C1L_Aout, Bout => C1L_Bout);
join1:    SKYRMIONJOIN port map (A => cell1_OUTOP, B => C0R_Bout, CURRENT => CURRENT_Vop, OUTPUT => JOIN1_out);
C1R:    SKYRMIONCROSS port map (A => C1L_Aout, B => JOIN1_out, CURRENTA => CURRENT_Vtr1, CURRENTB => CURRENT_Vop, Aout => C1R_Aout, Bout => C1R_Bout);
MTJR1:    MTJ_R port map (IN_SK => cell1_OUTOR, CURRENT => CURRENT_Vop, OUT_SIGN => MTJ_R1_out);

Vtr2:    voltage_genL port map (CTRL => CTRL_Vtr2, CURRENT => CURRENT_Vtr2);
input2:    cell_input port map (IN_SK_L => cell2_OUTFEED_L, IN_SK_R => cell2_OUTFEED_R, IN_SK_T => C1L_Bout, ENABLE => CTRL_in2_EN, CURRENT_T => CURRENT_Vbl, CURRENT_L => CURRENT_Vtr2, CURRENT_R => CURRENT_Vop, OUT_SK_L => cell2_IN_L, OUT_SK_R => cell2_IN_R, OUT_SK_B => D2_Bout);
I2L:    SKYRMIONCROSS port map (A => CELL2_intr, B => cell2_OUTTRAN, CURRENTA => CELL2_intr_CURR, CURRENTB => CURRENT_Vtr2, Aout => I2L_Aout, Bout => I2L_Bout);
I2R:    SKYRMIONCROSS port map (A => I2L_Aout, B => D2_Bout, CURRENTA => CELL2_intr_CURR, CURRENTB => CURRENT_Vbl, Aout => I2R_Aout, Bout => I2R_Bout);
cell2:    Cellx port map (IN_R => cell2_IN_R, IN_L => cell2_IN_L, IN_TR => I2R_Aout, CTRL_MTJ => CTRL_MTJ_cell2, CURRENT_OP => CURRENT_Vop, CURRENT_TR => CURRENT_Vtr2, OUTFEED_R => cell2_OUTFEED_R, OUTFEED_L => cell2_OUTFEED_L, OUTTRAN => cell2_OUTTRAN, OUTOR => cell2_OUTOR, OUTOP => cell2_OUTOP);
C2L:    SKYRMIONCROSS port map (A => I2L_Bout, B => I2R_Bout, CURRENTA => CURRENT_Vtr2, CURRENTB => CURRENT_Vbl, Aout => C2L_Aout, Bout => C2L_Bout);
join2:    SKYRMIONJOIN port map (A => cell2_OUTOP, B => C1R_Bout, CURRENT => CURRENT_Vop, OUTPUT => JOIN2_out);
C2R:    SKYRMIONCROSS port map (A => C2L_Aout, B => JOIN2_out, CURRENTA => CURRENT_Vtr2, CURRENTB => CURRENT_Vop, Aout => C2R_Aout, Bout => C2R_Bout);
MTJR2:    MTJ_R port map (IN_SK => cell2_OUTOR, CURRENT => CURRENT_Vop, OUT_SIGN => MTJ_R2_out);

MTJRR:    MTJ_R port map (IN_SK => C2R_Bout, CURRENT => CURRENT_Vop, OUT_SIGN => MTJ_RRx_out);

```

```
  BL_out <= C2L_Bout;
  CELL0_tran <= C0R_Aout;
  CELL1_tran <= C1R_Aout;
  CELL2_tran <= C2R_Aout;
  CELL0_tran_CURR <= CURRENT_Vtr0;
  CELL1_tran_CURR <= CURRENT_Vtr1;
  CELL2_tran_CURR <= CURRENT_Vtr2;

end architecture Behaviour;
```

D.1.6. Memory array

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity MemArray is
port(  CTRL_Vbl_0: in std_logic;
    CTRL_Vop_0: in std_logic;
    CTRL_Vtr_00: in std_logic;
    CTRL_Vtr_01: in std_logic;
    CTRL_Vtr_02: in std_logic;
    CTRL_MTJW_0: in std_logic;
    CTRL_in_EN_00: in std_logic;
    CTRL_in_EN_01: in std_logic;
    CTRL_in_EN_02: in std_logic;
    CTRL_MTJ_cell_00: in std_logic;
    CTRL_MTJ_cell_01: in std_logic;
    CTRL_MTJ_cell_02: in std_logic;
    MTJ_RR0_out: out std_logic;
    MTJ_R00_out: out std_logic;
    MTJ_R01_out: out std_logic;
    MTJ_R02_out: out std_logic;
    ---
    CTRL_Vbl_1: in std_logic;
    CTRL_Vop_1: in std_logic;
    CTRL_Vtr_10: in std_logic;
    CTRL_Vtr_11: in std_logic;
    CTRL_Vtr_12: in std_logic;
    CTRL_MTJW_1: in std_logic;
    CTRL_in_EN_10: in std_logic;
    CTRL_in_EN_11: in std_logic;
```

```
    CTRL_in_EN_12: in std_logic;
    CTRL_MTJ_cell_10: in std_logic;
    CTRL_MTJ_cell_11: in std_logic;
    CTRL_MTJ_cell_12: in std_logic;
    MTJ_RR1_out: out std_logic;
    MTJ_R10_out: out std_logic;
    MTJ_R11_out: out std_logic;
    MTJ_R12_out: out std_logic;
    ---
    CTRL_Vbl_2: in std_logic;
    CTRL_Vop_2: in std_logic;
    CTRL_Vtr_20: in std_logic;
    CTRL_Vtr_21: in std_logic;
    CTRL_Vtr_22: in std_logic;
    CTRL_MTJW_2: in std_logic;
    CTRL_in_EN_20: in std_logic;
    CTRL_in_EN_21: in std_logic;
    CTRL_in_EN_22: in std_logic;
    CTRL_MTJ_cell_20: in std_logic;
    CTRL_MTJ_cell_21: in std_logic;
    CTRL_MTJ_cell_22: in std_logic;
    MTJ_RR2_out: out std_logic;
    MTJ_R20_out: out std_logic;
    MTJ_R21_out: out std_logic;
    MTJ_R22_out: out std_logic);
end entity MemArray;

architecture Behaviour of MemArray is
  component Word0 is
  port(  CTRL_Vbl:       in std_logic;
      CTRL_Vop:      in std_logic;
      CTRL_Vtr0:      in std_logic;
      CTRL_Vtr1:      in std_logic;
      CTRL_Vtr2:      in std_logic;
      CTRL_MTJW:       in std_logic;
      CTRL_in0_EN:     in std_logic;
      CTRL_in1_EN:     in std_logic;
      CTRL_in2_EN:     in std_logic;
      CTRL_MTJ_cell0:    in std_logic;
      CTRL_MTJ_cell1:    in std_logic;
      CTRL_MTJ_cell2:    in std_logic;
      BL_out:        out std_logic;
      MTJ_RR0_out:    out std_logic;
      MTJ_R0_out:      out std_logic;
      MTJ_R1_out:      out std_logic;
      MTJ_R2_out:      out std_logic;
      CELL0_tran:      out std_logic;
      CELL1_tran:      out std_logic;
      CELL2_tran:      out std_logic;
```

```
      CELL0_tran_CURR:  out real;
      CELL1_tran_CURR:  out real;
      CELL2_tran_CURR:  out real);
  end component Word0;

  component Wordx is
  port(  CTRL_Vbl:       in std_logic;
      CTRL_Vop:      in std_logic;
      CTRL_Vtr0:      in std_logic;
      CTRL_Vtr1:      in std_logic;
      CTRL_Vtr2:      in std_logic;
      CTRL_MTJW:       in std_logic;
      CTRL_in0_EN:     in std_logic;
      CTRL_in1_EN:     in std_logic;
      CTRL_in2_EN:     in std_logic;
      CTRL_MTJ_cell0:    in std_logic;
      CTRL_MTJ_cell1:    in std_logic;
      CTRL_MTJ_cell2:    in std_logic;
      CELL0_intr:      in std_logic;
      CELL1_intr:      in std_logic;
      CELL2_intr:      in std_logic;
      CELL0_intr_CURR:  in real;
      CELL1_intr_CURR:  in real;
      CELL2_intr_CURR:  in real;
      BL_out:        out std_logic;
      MTJ_RRx_out:    out std_logic;
      MTJ_R0_out:    out std_logic;
      MTJ_R1_out:    out std_logic;
      MTJ_R2_out:    out std_logic;
      CELL0_tran:      out std_logic;
      CELL1_tran:      out std_logic;
      CELL2_tran:      out std_logic;
      CELL0_tran_CURR:  out real;
      CELL1_tran_CURR:  out real;
      CELL2_tran_CURR:  out real);
  end component Wordx;

  signal BL_out_0, CELL_tran_00, CELL_tran_01, CELL_tran_02: std_logic;
  signal CELL_tran_CURR_00, CELL_tran_CURR_01, CELL_tran_CURR_02: real;
  ---
  signal BL_out_1, CELL_tran_10, CELL_tran_11, CELL_tran_12: std_logic;
  signal CELL_tran_CURR_10, CELL_tran_CURR_11, CELL_tran_CURR_12: real;
  --
  signal BL_out_2, CELL_tran_20, CELL_tran_21, CELL_tran_22: std_logic;
  signal CELL_tran_CURR_20, CELL_tran_CURR_21, CELL_tran_CURR_22: real;

begin

  Word0_c: Word0 port map (
```

```
    CTRL_Vbl => CTRL_Vbl_0,
    CTRL_Vop => CTRL_Vop_0,
    CTRL_Vtr0 => CTRL_Vtr_00,
    CTRL_Vtr1 => CTRL_Vtr_01,
    CTRL_Vtr2 => CTRL_Vtr_02,
    CTRL_MTJW => CTRL_MTJW_0,
    CTRL_in0_EN => CTRL_in_EN_00,
    CTRL_in1_EN => CTRL_in_EN_01,
    CTRL_in2_EN => CTRL_in_EN_02,
    CTRL_MTJ_cell0 => CTRL_MTJ_cell_00,
    CTRL_MTJ_cell1 => CTRL_MTJ_cell_01,
    CTRL_MTJ_cell2 => CTRL_MTJ_cell_02,
    BL_out => BL_out_0,
    MTJ_RR0_out => MTJ_RR0_out,
    MTJ_R0_out => MTJ_R00_out,
    MTJ_R1_out => MTJ_R01_out,
    MTJ_R2_out => MTJ_R02_out,
    CELL0_tran => CELL_tran_00,
    CELL1_tran => CELL_tran_01,
    CELL2_tran => CELL_tran_02,
    CELL0_tran_CURR => CELL_tran_CURR_00,
    CELL1_tran_CURR => CELL_tran_CURR_01,
    CELL2_tran_CURR => CELL_tran_CURR_02
  );

  Word1: Wordx port map (
    CTRL_Vbl => CTRL_Vbl_1,
    CTRL_Vop => CTRL_Vop_1,
    CTRL_Vtr0 => CTRL_Vtr_10,
    CTRL_Vtr1 => CTRL_Vtr_11,
    CTRL_Vtr2 => CTRL_Vtr_12,
    CTRL_MTJW => CTRL_MTJW_1,
    CTRL_in0_EN => CTRL_in_EN_10,
    CTRL_in1_EN => CTRL_in_EN_11,
    CTRL_in2_EN => CTRL_in_EN_12,
    CTRL_MTJ_cell0 => CTRL_MTJ_cell_10,
    CTRL_MTJ_cell1 => CTRL_MTJ_cell_11,
    CTRL_MTJ_cell2 => CTRL_MTJ_cell_12,
    ---
    CELL0_intr => CELL_tran_00,
    CELL1_intr => CELL_tran_01,
    CELL2_intr => CELL_tran_02,
    CELL0_intr_CURR => CELL_tran_CURR_00,
    CELL1_intr_CURR => CELL_tran_CURR_01,
    CELL2_intr_CURR => CELL_tran_CURR_02,
    ---
    BL_out => BL_out_1,
    MTJ_RRx_out => MTJ_RR1_out,
    MTJ_R0_out => MTJ_R10_out,
```

```
    MTJ_R1_out => MTJ_R11_out,
    MTJ_R2_out => MTJ_R12_out,
    CELL0_tran => CELL_tran_10,
    CELL1_tran => CELL_tran_11,
    CELL2_tran => CELL_tran_12,
    CELL0_tran_CURR => CELL_tran_CURR_10,
    CELL1_tran_CURR => CELL_tran_CURR_11,
    CELL2_tran_CURR => CELL_tran_CURR_12
  );

  Word2: Wordx port map (
    CTRL_Vbl => CTRL_Vbl_2,
    CTRL_Vop => CTRL_Vop_2,
    CTRL_Vtr0 => CTRL_Vtr_20,
    CTRL_Vtr1 => CTRL_Vtr_21,
    CTRL_Vtr2 => CTRL_Vtr_22,
    CTRL_MTJW => CTRL_MTJW_2,
    CTRL_in0_EN => CTRL_in_EN_20,
    CTRL_in1_EN => CTRL_in_EN_21,
    CTRL_in2_EN => CTRL_in_EN_22,
    CTRL_MTJ_cell0 => CTRL_MTJ_cell_20,
    CTRL_MTJ_cell1 => CTRL_MTJ_cell_21,
    CTRL_MTJ_cell2 => CTRL_MTJ_cell_22,
    ---
    CELL0_intr => CELL_tran_10,
    CELL1_intr => CELL_tran_11,
    CELL2_intr => CELL_tran_12,
    CELL0_intr_CURR => CELL_tran_CURR_10,
    CELL1_intr_CURR => CELL_tran_CURR_11,
    CELL2_intr_CURR => CELL_tran_CURR_12,
    ---
    BL_out => BL_out_2,
    MTJ_RRx_out => MTJ_RR2_out,
    MTJ_R0_out => MTJ_R20_out,
    MTJ_R1_out => MTJ_R21_out,
    MTJ_R2_out => MTJ_R22_out,
    CELL0_tran => CELL_tran_20,
    CELL1_tran => CELL_tran_21,
    CELL2_tran => CELL_tran_22,
    CELL0_tran_CURR => CELL_tran_CURR_20,
    CELL1_tran_CURR => CELL_tran_CURR_21,
    CELL2_tran_CURR => CELL_tran_CURR_22
  );

end architecture Behaviour;
```

D.2. Control blocks

D.2.1. Detector

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Detector is
port(  L_MTJ_RR_out_0:   in std_logic;
    L_MTJ_RR_out_1:   in std_logic;
    L_MTJ_RR_out_2:   in std_logic;
    LATCH0_OUT:      in std_logic;
    LATCH1_OUT:      in std_logic;
    LATCH2_OUT:      in std_logic;
    ENABLE_Rowdis:    out std_logic);
end entity Detector;

architecture Behaviour of Detector is
  component mux2_1b is
  port(  IN0:   in std_logic;
      IN1:   in std_logic;
      SEL:  in std_logic;
      OUTM:  out std_logic
  );
  end component mux2_1b;

  signal mux0_L_out, mux1_L_out, mux2_L_out, mux0_R_out, mux1_R_out, mux2_R_out, exnor_inL, exnor_inR: std_logic;
begin
  mux0_L: mux2_1b port map (IN0 => L_MTJ_RR_out_0, IN1 => '1', SEL => LATCH0_OUT, OUTM => mux0_L_out);
  mux1_L: mux2_1b port map (IN0 => L_MTJ_RR_out_1, IN1 => '1', SEL => LATCH1_OUT, OUTM => mux1_L_out);
  mux2_L: mux2_1b port map (IN0 => L_MTJ_RR_out_2, IN1 => '1', SEL => LATCH2_OUT, OUTM => mux2_L_out);

  mux0_R: mux2_1b port map (IN0 => L_MTJ_RR_out_0, IN1 => '0', SEL => LATCH0_OUT, OUTM => mux0_R_out);
  mux1_R: mux2_1b port map (IN0 => L_MTJ_RR_out_1, IN1 => '0', SEL => LATCH1_OUT, OUTM => mux1_R_out);
  mux2_R: mux2_1b port map (IN0 => L_MTJ_RR_out_2, IN1 => '0', SEL => LATCH2_OUT, OUTM => mux2_R_out);

  exnor_inL <= (mux0_L_out and mux1_L_out) and mux2_L_out;
```

```
  exnor_inR <= ((not mux0_R_out) and (not mux1_R_out)) and (not mux2_R_out);
  ENABLE_Rowdis <= exnor_inL xnor exnor_inR;

end architecture Behaviour;
```

D.2.2. Row disabler

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Row_disabler is
port(  L_MTJ_RR_out_0:   in std_logic;
    L_MTJ_RR_out_1:   in std_logic;
    L_MTJ_RR_out_2:   in std_logic;
    MAX_min_n:      in std_logic;
    ENABLE_Rowdis:    in std_logic;
    FIND:        in std_logic;
    CTRL_LATCH_RST:    in std_logic;
    LATCH0_OUT:      out std_logic;
    LATCH1_OUT:      out std_logic;
    LATCH2_OUT:      out std_logic);
end entity Row_disabler;

architecture Behaviour of Row_disabler is
  component mux2_1b is
  port(  IN0:   in std_logic;
      IN1:   in std_logic;
      SEL:  in std_logic;
      OUTM:  out std_logic
  );
  end component mux2_1b;

  component SRlatch is
  port(  SET:   in std_logic;
      RST:   in std_logic;
      Q:    out std_logic
  );
  end component SRlatch;

  signal notMTJ_0, notMTJ_1, notMTJ_2, mux0_Mm_out, mux1_Mm_out, mux2_Mm_out,
   ↪  sel_mux0_01, sel_mux1_01, sel_mux2_01, mux0_01_out, mux1_01_out,
   ↪  mux2_01_out: std_logic;
```

```
begin
  notMTJ_0 <= not L_MTJ_RR_out_0;
  notMTJ_1 <= not L_MTJ_RR_out_1;
  notMTJ_2 <= not L_MTJ_RR_out_2;

  mux0_Mm: mux2_1b port map (IN0 => L_MTJ_RR_out_0, IN1 => notMTJ_0, SEL => MAX_min_n, OUTM => mux0_Mm_out);
  mux1_Mm: mux2_1b port map (IN0 => L_MTJ_RR_out_1, IN1 => notMTJ_1, SEL => MAX_min_n, OUTM => mux1_Mm_out);
  mux2_Mm: mux2_1b port map (IN0 => L_MTJ_RR_out_2, IN1 => notMTJ_2, SEL => MAX_min_n, OUTM => mux2_Mm_out);

  sel_mux0_01 <= mux0_Mm_out and ENABLE_Rowdis and FIND;
  sel_mux1_01 <= mux1_Mm_out and ENABLE_Rowdis and FIND;
  sel_mux2_01 <= mux2_Mm_out and ENABLE_Rowdis and FIND;

  mux0_01: mux2_1b port map (IN0 => '0', IN1 => '1', SEL => sel_mux0_01, OUTM => mux0_01_out);
  mux1_01: mux2_1b port map (IN0 => '0', IN1 => '1', SEL => sel_mux1_01, OUTM => mux1_01_out);
  mux2_01: mux2_1b port map (IN0 => '0', IN1 => '1', SEL => sel_mux2_01, OUTM => mux2_01_out);

  latch0: SRlatch port map (SET => mux0_01_out, RST => CTRL_LATCH_RST, Q => LATCH0_OUT);
  latch1: SRlatch port map (SET => mux1_01_out, RST => CTRL_LATCH_RST, Q => LATCH1_OUT);
  latch2: SRlatch port map (SET => mux2_01_out, RST => CTRL_LATCH_RST, Q => LATCH2_OUT);
end architecture Behaviour;
```

D.2.3. Encoder

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
use work.globals.all;

entity Encoder is
port(  CTRL_ENCODER_EN:    in std_logic;
    LATCH0_OUT:       in std_logic;
    LATCH1_OUT:       in std_logic;
    LATCH2_OUT:       in std_logic;
    CLK:          in std_logic;
```

```
    CTRL_ADDREG_RST:    in std_logic;
    CTRL_ADDREG_STORE:    in std_logic;
    ADDRESS_FOUND_REGout:   out std_logic_vector(1 downto 0) );
end entity Encoder;

architecture Behaviour of Encoder is
  component Reg is
  generic (N: integer:= 3);
  port(  CLK:   in std_logic;
      RST:   in std_logic;
      STORE:  in std_logic;
      DATA_IN: in std_logic_vector (N-1 downto 0);
      DATA_OUT: buffer std_logic_vector (N-1 downto 0) );
  end component Reg;

  signal ADDRESS_FOUND: std_logic_vector(1 downto 0);
  signal NOT_LATCH0_OUT, NOT_LATCH1_OUT, NOT_LATCH2_OUT: std_logic;
begin
  NOT_LATCH0_OUT <= (not LATCH0_OUT) and CTRL_ENCODER_EN;
  NOT_LATCH1_OUT <= (not LATCH1_OUT) and CTRL_ENCODER_EN;
  NOT_LATCH2_OUT <= (not LATCH2_OUT) and CTRL_ENCODER_EN;

  ADDRESS_FOUND <=   "00" when NOT_LATCH0_OUT='1' else
           "01" when NOT_LATCH1_OUT='1' else
           "10" when NOT_LATCH2_OUT='1' else
           "ZZ";

  ADD_REG: Reg generic map (N=>2) port map (CLK => CLK, RST => CTRL_ADDREG_RST,
  ↪  STORE => CTRL_ADDREG_STORE, DATA_IN => ADDRESS_FOUND, DATA_OUT =>
  ↪  ADDRESS_FOUND_REGout);

end architecture Behaviour;
```

D.2.4. And array

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity And_array is
port(  RESET_n:    in std_logic;
    IDLE:      in std_logic;
```

```
    FIND:      in std_logic;
    DEC0:       in std_logic;
    DEC1:       in std_logic;
    DEC2:       in std_logic;
    LATCH0_OUT:    in std_logic;
    LATCH1_OUT:    in std_logic;
    LATCH2_OUT:    in std_logic;
    AAout0:      buffer std_logic;
    AAout1:      buffer std_logic;
    AAout2:      buffer std_logic;
    AAoutOR:    out std_logic);
end entity And_array;

architecture Behaviour of And_array is
  signal OR0, OR1, OR2: std_logic;
begin
  OR0 <= (not RESET_n) or IDLE or DEC0 or FIND;
  AAout0 <= OR0 and (not LATCH0_OUT);

  OR1 <= (not RESET_n) or IDLE or DEC1 or FIND;
  AAout1 <= OR1 and (not LATCH1_OUT);

  OR2 <= (not RESET_n) or IDLE or DEC2 or FIND;
  AAout2 <= OR2 and (not LATCH2_OUT);

  AAoutOR <= AAout0 or AAout1 or AAout2;
end architecture Behaviour;
```

D.2.5. FSM-Array adapter

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity FSM_Array_adapter is
port(  AAout0:      in std_logic;
    AAout1:      in std_logic;
    AAout2:      in std_logic;
    ---
    CTRL_Vbl:    in std_logic;
    CTRL_Vbl_0:    out std_logic;
    CTRL_Vbl_1:    out std_logic;
```

```
    CTRL_Vbl_2:    out std_logic;
    ---
    CTRL_MTJW:      in std_logic;
    SHIFT_WORD_OUT_0:   in std_logic;
    SHIFT_WORD_OUT_1:   in std_logic;
    SHIFT_WORD_OUT_2:   in std_logic;
    CTRL_MTJW_0:    out std_logic;
    CTRL_MTJW_1:    out std_logic;
    CTRL_MTJW_2:    out std_logic;
    ---
    CTRL_in_EN_0:    in std_logic;
    CTRL_in_EN_1:    in std_logic;
    CTRL_in_EN_2:    in std_logic;
    CTRL_in_EN_00:    out std_logic;
    CTRL_in_EN_01:    out std_logic;
    CTRL_in_EN_02:    out std_logic;
    CTRL_in_EN_10:    out std_logic;
    CTRL_in_EN_11:    out std_logic;
    CTRL_in_EN_12:    out std_logic;
    CTRL_in_EN_20:    out std_logic;
    CTRL_in_EN_21:    out std_logic;
    CTRL_in_EN_22:    out std_logic;
    ---
    CTRL_Vop:      in std_logic;
    CTRL_Vop_0:      out std_logic;
    CTRL_Vop_1:      out std_logic;
    CTRL_Vop_2:      out std_logic;
    ---
    CTRL_MTJ_CELL:    in std_logic;
    SHIFT_MASK:      in std_logic_vector(2 downto 0);
    CTRL_MTJ_CELL_00:  out std_logic;
    CTRL_MTJ_CELL_01:  out std_logic;
    CTRL_MTJ_CELL_02:  out std_logic;
    CTRL_MTJ_CELL_10:  out std_logic;
    CTRL_MTJ_CELL_11:  out std_logic;
    CTRL_MTJ_CELL_12:  out std_logic;
    CTRL_MTJ_CELL_20:  out std_logic;
    CTRL_MTJ_CELL_21:  out std_logic;
    CTRL_MTJ_CELL_22:  out std_logic;
    ---
    CTRL_Vtr_0:      in std_logic;
    CTRL_Vtr_1:      in std_logic;
    CTRL_Vtr_2:      in std_logic;
    CTRL_Vtr_00:    out std_logic;
    CTRL_Vtr_01:    out std_logic;
    CTRL_Vtr_02:    out std_logic;
    CTRL_Vtr_10:    out std_logic;
    CTRL_Vtr_11:    out std_logic;
    CTRL_Vtr_12:    out std_logic;
```

```
    CTRL_Vtr_20:    out std_logic;
    CTRL_Vtr_21:    out std_logic;
    CTRL_Vtr_22:    out std_logic;
    ---
    CTRL_RST_L_MTJRR:  in std_logic;
    CTRL_RST_L_MTJRR_0:  out std_logic;
    CTRL_RST_L_MTJRR_1:  out std_logic;
    CTRL_RST_L_MTJRR_2: out std_logic;
    ---
    CTRL_RST_L_MTJR:  in std_logic;
    CTRL_RST_L_MTJR_0: out std_logic;
    CTRL_RST_L_MTJR_1: out std_logic;
    CTRL_RST_L_MTJR_2: out std_logic);
end entity FSM_Array_adapter;

architecture Behaviour of FSM_Array_adapter is
begin
  CTRL_Vbl_0       <= AAout0 and CTRL_Vbl;
  CTRL_MTJW_0     <= AAout0 and CTRL_MTJW and SHIFT_WORD_OUT_0;
  CTRL_in_EN_00    <= AAout0 and CTRL_in_EN_0;
  CTRL_in_EN_01    <= AAout0 and CTRL_in_EN_1;
  CTRL_in_EN_02    <= AAout0 and CTRL_in_EN_2;
  CTRL_Vop_0      <= AAout0 and CTRL_Vop;
  CTRL_MTJ_CELL_00  <= AAout0 and CTRL_MTJ_CELL and SHIFT_MASK(0);
  CTRL_MTJ_CELL_01  <= AAout0 and CTRL_MTJ_CELL and SHIFT_MASK(1);
  CTRL_MTJ_CELL_02  <= AAout0 and CTRL_MTJ_CELL and SHIFT_MASK(2);
  CTRL_Vtr_00       <= AAout0 and CTRL_Vtr_0;
  CTRL_Vtr_01       <= AAout0 and CTRL_Vtr_1;
  CTRL_Vtr_02       <= AAout0 and CTRL_Vtr_2;
  CTRL_RST_L_MTJRR_0  <= AAout0 and CTRL_RST_L_MTJRR;
  CTRL_RST_L_MTJR_0  <= AAout0 and CTRL_RST_L_MTJR;

  CTRL_Vbl_1       <= AAout1 and CTRL_Vbl;
  CTRL_MTJW_1     <= AAout1 and CTRL_MTJW and SHIFT_WORD_OUT_1;
  CTRL_in_EN_10    <= AAout1 and CTRL_in_EN_0;
  CTRL_in_EN_11    <= AAout1 and CTRL_in_EN_1;
  CTRL_in_EN_12    <= AAout1 and CTRL_in_EN_2;
  CTRL_Vop_1      <= AAout1 and CTRL_Vop;
  CTRL_MTJ_CELL_10  <= AAout1 and CTRL_MTJ_CELL and SHIFT_MASK(0);
  CTRL_MTJ_CELL_11  <= AAout1 and CTRL_MTJ_CELL and SHIFT_MASK(1);
  CTRL_MTJ_CELL_12  <= AAout1 and CTRL_MTJ_CELL and SHIFT_MASK(2);
  CTRL_Vtr_10       <= AAout1 and CTRL_Vtr_0;
  CTRL_Vtr_11       <= AAout1 and CTRL_Vtr_1;
  CTRL_Vtr_12       <= AAout1 and CTRL_Vtr_2;
  CTRL_RST_L_MTJRR_1  <= AAout1 and CTRL_RST_L_MTJRR;
  CTRL_RST_L_MTJR_1  <= AAout1 and CTRL_RST_L_MTJR;

  CTRL_Vbl_2       <= AAout2 and CTRL_Vbl;
  CTRL_MTJW_2     <= AAout2 and CTRL_MTJW and SHIFT_WORD_OUT_2;
```

```
  CTRL_in_EN_20    <= AAout2 and CTRL_in_EN_0;
  CTRL_in_EN_21    <= AAout2 and CTRL_in_EN_1;
  CTRL_in_EN_22    <= AAout2 and CTRL_in_EN_2;
  CTRL_Vop_2      <= AAout2 and CTRL_Vop;
  CTRL_MTJ_CELL_20  <= AAout2 and CTRL_MTJ_CELL and SHIFT_MASK(0);
  CTRL_MTJ_CELL_21  <= AAout2 and CTRL_MTJ_CELL and SHIFT_MASK(1);
  CTRL_MTJ_CELL_22  <= AAout2 and CTRL_MTJ_CELL and SHIFT_MASK(2);
  CTRL_Vtr_20      <= AAout2 and CTRL_Vtr_0;
  CTRL_Vtr_21      <= AAout2 and CTRL_Vtr_1;
  CTRL_Vtr_22      <= AAout2 and CTRL_Vtr_2;
  CTRL_RST_L_MTJRR_2  <= AAout2 and CTRL_RST_L_MTJRR;
  CTRL_RST_L_MTJR_2  <= AAout2 and CTRL_RST_L_MTJR;

end architecture Behaviour;
```

D.2.6. Decoder

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
use work.globals.all;

entity Decoder is
generic (N: integer := 3);
port(  ENABLE:   in std_logic;
    ADDRESS:   in std_logic_vector(N-1 downto 0);
    WLINES:    out std_logic_vector(2**N-1 downto 0) );
end entity Decoder;

architecture Behaviour of Decoder is
begin
  process(ENABLE, ADDRESS)
  begin
    if (ENABLE = '1') then
      WLINES <= (others => '0');
      WLINES(to_integer(unsigned(ADDRESS))) <= '1';
    else
      WLINES <= (others => 'Z');
    end if;
  end process;
end architecture Behaviour;
```

D.2.7. Latches inside the memory array

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity memarray_latches is
port(  MTJ_R00_out, MTJ_R01_out, MTJ_R02_out: in std_logic;
    MTJ_R10_out, MTJ_R11_out, MTJ_R12_out: in std_logic;
    MTJ_R20_out, MTJ_R21_out, MTJ_R22_out: in std_logic;
    EN_READ: in std_logic;
    CTRL_RST_L_MTJR_0: in std_logic;
    CTRL_RST_L_MTJR_1: in std_logic;
    CTRL_RST_L_MTJR_2: in std_logic;
    L_MTJ_R_out_00, L_MTJ_R_out_01, L_MTJ_R_out_02: out std_logic;
    L_MTJ_R_out_10, L_MTJ_R_out_11, L_MTJ_R_out_12: out std_logic;
    L_MTJ_R_out_20, L_MTJ_R_out_21, L_MTJ_R_out_22: out std_logic;
    ---
    MTJ_RR0_out: in std_logic;
    MTJ_RR1_out: in std_logic;
    MTJ_RR2_out: in std_logic;
    CTRL_RST_L_MTJRR_0: in std_logic;
    CTRL_RST_L_MTJRR_1: in std_logic;
    CTRL_RST_L_MTJRR_2: in std_logic;
    L_MTJ_RR_out_0: out std_logic;
    L_MTJ_RR_out_1: out std_logic;
    L_MTJ_RR_out_2: out std_logic);
end entity memarray_latches;

architecture Behaviour of memarray_latches is
  component SRlatch is
  port(  SET:   in std_logic;
      RST:   in std_logic;
      Q:    out std_logic);
  end component SRlatch;

  signal SET00, SET01, SET02, SET10, SET11, SET12, SET20, SET21, SET22: std_logic;
begin
  SET00 <= MTJ_R00_out and EN_READ;
  SET01 <= MTJ_R01_out and EN_READ;
  SET02 <= MTJ_R02_out and EN_READ;
  SET10 <= MTJ_R10_out and EN_READ;
  SET11 <= MTJ_R11_out and EN_READ;
  SET12 <= MTJ_R12_out and EN_READ;
  SET20 <= MTJ_R20_out and EN_READ;
```

```
    SET21 <= MTJ_R21_out and EN_READ;
    SET22 <= MTJ_R22_out and EN_READ;

    SR_R00: SRlatch port map (SET => SET00, RST => CTRL_RST_L_MTJR_0, Q => L_MTJ_R_out_00);
    SR_R01: SRlatch port map (SET => SET01, RST => CTRL_RST_L_MTJR_0, Q => L_MTJ_R_out_01);
    SR_R02: SRlatch port map (SET => SET02, RST => CTRL_RST_L_MTJR_0, Q => L_MTJ_R_out_02);
    SR_R10: SRlatch port map (SET => SET10, RST => CTRL_RST_L_MTJR_1, Q => L_MTJ_R_out_10);
    SR_R11: SRlatch port map (SET => SET11, RST => CTRL_RST_L_MTJR_1, Q => L_MTJ_R_out_11);
    SR_R12: SRlatch port map (SET => SET12, RST => CTRL_RST_L_MTJR_1, Q => L_MTJ_R_out_12);
    SR_R20: SRlatch port map (SET => SET20, RST => CTRL_RST_L_MTJR_2, Q => L_MTJ_R_out_20);
    SR_R21: SRlatch port map (SET => SET21, RST => CTRL_RST_L_MTJR_2, Q => L_MTJ_R_out_21);
    SR_R22: SRlatch port map (SET => SET22, RST => CTRL_RST_L_MTJR_2, Q => L_MTJ_R_out_22);

    SR_RR0: SRlatch port map (SET => MTJ_RR0_out, RST => CTRL_RST_L_MTJRR_0, Q => L_MTJ_RR_out_0);
    SR_RR1: SRlatch port map (SET => MTJ_RR1_out, RST => CTRL_RST_L_MTJRR_1, Q => L_MTJ_RR_out_1);
    SR_RR2: SRlatch port map (SET => MTJ_RR2_out, RST => CTRL_RST_L_MTJRR_2, Q => L_MTJ_RR_out_2);
end architecture Behaviour;
```

D.2.8. Output multiplexer and register

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity readout is
port(  L_MTJ_R_out_00, L_MTJ_R_out_01, L_MTJ_R_out_02: in std_logic;
    L_MTJ_R_out_10, L_MTJ_R_out_11, L_MTJ_R_out_12: in std_logic;
    L_MTJ_R_out_20, L_MTJ_R_out_21, L_MTJ_R_out_22: in std_logic;
    AAout0:        in std_logic;
    AAout1:        in std_logic;
```

```
    AAout2:        in std_logic;
    CLK:         in std_logic;
    CTRL_READWORD_RST:  in std_logic;
    CTRL_READWORD_EN:  in std_logic;
    READWORD_OUT:     out std_logic_vector (2 downto 0) );
end entity readout;

architecture Behaviour of readout is
  component Reg is
  generic (N: integer:= 3);
  port(  CLK:   in std_logic;
      RST:   in std_logic;
      STORE:  in std_logic;
      DATA_IN: in std_logic_vector (N-1 downto 0);
      DATA_OUT: buffer std_logic_vector (N-1 downto 0) );
  end component Reg;

  signal SEL: std_logic_vector(2 downto 0);
  signal IN0, IN1, IN2, MUX_OUT: std_logic_vector(2 downto 0);
begin
  SEL <= AAout2 & AAout1 & AAout0;
  IN0 <= L_MTJ_R_out_02 & L_MTJ_R_out_01 & L_MTJ_R_out_00;
  IN1 <= L_MTJ_R_out_12 & L_MTJ_R_out_11 & L_MTJ_R_out_10;
  IN2 <= L_MTJ_R_out_22 & L_MTJ_R_out_21 & L_MTJ_R_out_20;

  with SEL select
    MUX_OUT <=   IN0 when "001",
          IN1 when "010",
          IN2 when "100",
          (others => 'Z') when others;

  READ_WORD_REG: Reg generic map (N=>3) port map (CLK => CLK, RST =>
  ↪   CTRL_READWORD_RST, STORE => CTRL_READWORD_EN, DATA_IN => MUX_OUT, DATA_OUT
  ↪   => READWORD_OUT);
end architecture Behaviour;
```

D.3. Basic components

D.3.1. Globals

```
package GLOBALS is
  type coordinates_xy is array (0 to 1) of real;
  type parameters_array is array(0 to 9) of coordinates_xy;
  type bool_array is array(integer range <>) of boolean;
  type real_array is array(integer range <>) of real;

  constant HORIZONTAL_SPEED : real := 150.0;  --m/s
  constant VERTICAL_SPEED : real :=  40.0;  --m/s
  constant DEPINNING_CURRENT : real := 260.0;  --nA
  constant NOTCH_DEPINNING_CURRENT : real := 3200.0;  --nA
  constant HORIZONTAL_SPEED_HIGH : real := 484.0;   --m/2
  constant SKYRMION_DIAMETER : real := 18.0;      --nm
  constant SKYRMION_MIN_DISTANCE : real := 22.0;    --nm
  constant CURRENT_LOW : real := 800.0;  --nA
  constant CURRENT_HIGH : real := 3200.0;  --nA
  constant CLOCK_LOW : time := 5.35 ns;
  constant CLOCK_HIGH : time := 150 ps;
  constant CLOCK_PERIOD : time := 5.5 ns;
  constant INPUTS_HIGH : time := 500 ps;

end package GLOBALS;
```

D.3.2. Two-way 1 bit multiplexer

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity mux2_1b is
port(  IN0:   in std_logic;
    IN1:   in std_logic;
    SEL:  in std_logic;
    OUTM:  out std_logic
);
end entity mux2_1b;

architecture Behaviour of mux2_1b is
begin
  with SEL select
    OUTM <= IN0 when '0',
        IN1 when '1',
        '0' when others;
```

```
end architecture Behaviour;
```

D.3.3. Register

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Reg is
generic (N: integer:= 3);
port(  CLK:   in std_logic;
    RST:   in std_logic;
    STORE:  in std_logic;
    DATA_IN: in std_logic_vector (N-1 downto 0);
    DATA_OUT: buffer std_logic_vector (N-1 downto 0) );
end entity Reg;

architecture Behaviour of Reg is
begin
  process (CLK, RST)
  begin
    if (RST='1') then
      DATA_OUT <= (others => '0');
    elsif(CLK'event and CLK='1') then
      if (STORE='1') then
        DATA_OUT <= DATA_IN;
      end if;
    end if;
  end process;

end architecture Behaviour;
```

D.3.4. Shift register

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```vhdl
use ieee.numeric_std.all;
use work.globals.all;

entity Shift_reg is
generic (N: integer:= 3);
port(  CLK:   in std_logic;
    RST:   in std_logic;
    SHIFT:  in std_logic;
    STORE:  in std_logic;
    RIGHT_LEFT_n: in std_logic;
    DATA_IN: in std_logic_vector (N-1 downto 0);
    DATA_OUT: buffer std_logic_vector (N-1 downto 0) );
end entity Shift_reg;

architecture Behaviour of Shift_reg is
begin
  latch: process (CLK, RST)
    variable number: std_logic_vector(N-1 downto 0);
  begin
    if (RST='1') then
      DATA_OUT <= (others => '0');
    elsif(CLK'event and CLK='1') then
      if (STORE='1') then
        DATA_OUT <= DATA_IN;
      elsif (SHIFT = '1') then
        if (RIGHT_LEFT_n='1') then
          number := '0'& DATA_OUT(N-1 downto 1);
        else
          number := DATA_OUT(N-2 downto 0) & '0';
        end if;
        DATA_OUT <= number;
      end if;
    end if;
  end process latch;

end architecture Behaviour;
```

D.3.5. SR latch

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;
```

```

entity SRlatch is
port(  SET:   in std_logic;
    RST:   in std_logic;
    Q:    out std_logic);
end entity SRlatch;

architecture Behaviour of SRlatch is

begin
  latch: process (SET, RST)
  begin
    if (RST='1') then
      Q <= '0';
    elsif(SET'event and SET='1') then
      Q <= '1';
    end if;
  end process latch;

end architecture Behaviour;
```

D.4. FSM

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.all;
use work.globals.all;


entity FSM is
port(  CLK:          in std_logic;
    RESET_n:       in std_logic;
    START_FIND:     in std_logic;
    READ_WRITE_n:     in std_logic;
    AAoutOR:       in std_logic;
    ---
    CTRL_Vbl:      out std_logic;
    CTRL_MTJW:      out std_logic;
    CTRL_in_EN_0:    out std_logic;
    CTRL_in_EN_1:    out std_logic;
    CTRL_in_EN_2:    out std_logic;
```

```
    CTRL_Vop:      out std_logic;
    CTRL_MTJ_CELL:    out std_logic;
    CTRL_Vtr_0:      out std_logic;
    CTRL_Vtr_1:      out std_logic;
    CTRL_Vtr_2:      out std_logic;
    CTRL_WORD_SHIFT:  out std_logic;
    CTRL_WORD_RST:    out std_logic;
    CTRL_RST_L_MTJRR:  out std_logic;
    CTRL_RST_L_MTJR:  out std_logic;
    CTRL_MASK_STORE:  out std_logic;
    CTRL_MASK_SHIFT:  out std_logic;
    CTRL_MASK_RST:    out std_logic;
    FIND:         out std_logic;
    IDLE:         out std_logic;
    EN_READ:       out std_logic;
    CTRL_READWORD_EN:  out std_logic;
    CTRL_READWORD_RST:  out std_logic;
    CTRL_LATCH_RST:    out std_logic;
    CTRL_ENCODER_EN:  out std_logic;
    CTRL_ADDREG_RST:  out std_logic;
    CTRL_ADDREG_STORE:  out std_logic);
end entity FSM;

architecture Behaviour of FSM is

  type state_type is (
    reset, S0, S1, S2, S3, S4, S6, S7, S8, S9
  );

  signal pstate, nstate: state_type;

begin

  state_register: process (CLK)
  begin
    if (CLK'event and CLK='1') then
      if (RESET_n = '0') then
        pstate <= reset;
      else
        pstate <= nstate;
      end if;
    end if;
  end process state_register;

  state_transition: process (pstate, CLK)
  begin
    case pstate is
      when reset  => nstate <= S0;
      when S0    => if (START_FIND='1') then nstate <= S1; elsif
       ↪  (READ_WRITE_n='1') then nstate <= S6; elsif(AAoutOR='1') then nstate
       ↪  <= S7; else nstate <= S0; end if;
```

```
      when S1    => nstate <= S2;
      when S2   => nstate <= S3;
      when S3   => nstate <= S4;
      when S4   => nstate <= S0;
      when S6   => nstate <= S0;
      when S7   => nstate <= S8;
      when S8   => nstate <= S9;
      when S9   => nstate <= S0;
      when others => nstate <= S0;
    end case;
  end process state_transition;

  output: process (pstate)
  begin
    CTRL_Vbl      <= '0';
    CTRL_MTJW      <= '0';
    CTRL_in_EN_0    <= '1';
    CTRL_in_EN_1    <= '1';
    CTRL_in_EN_2    <= '1';
    CTRL_Vop      <= '0';
    CTRL_MTJ_CELL    <= '0';
    CTRL_Vtr_0      <= '0';
    CTRL_Vtr_1      <= '0';
    CTRL_Vtr_2      <= '0';
    CTRL_WORD_SHIFT    <= '0';
    CTRL_WORD_RST    <= '0';
    CTRL_RST_L_MTJRR  <= '0';
    CTRL_RST_L_MTJR    <= '0';
    CTRL_MASK_STORE    <= '0';
    CTRL_MASK_SHIFT    <= '0';
    CTRL_MASK_RST    <= '0';
    FIND        <= '0';
    IDLE        <= '0';
    EN_READ        <= '0';
    CTRL_READWORD_EN  <= '0';
    CTRL_READWORD_RST  <= '0';
    CTRL_LATCH_RST    <= '0';
    CTRL_ENCODER_EN    <= '0';
    CTRL_ADDREG_RST    <= '0';
    CTRL_ADDREG_STORE  <= '0';

    case pstate is
      when S0    =>  CTRL_RST_L_MTJRR  <= '1';
              CTRL_RST_L_MTJR    <= '1';
              CTRL_MASK_STORE    <= '0';
              IDLE        <= '0';
              CTRL_LATCH_RST    <= '1';

      when S1    =>  FIND <= '1';
```

```
                CTRL_MTJ_CELL <= '1';
                CTRL_Vop <= '1';
                CTRL_MASK_SHIFT <= '1';
                CTRL_Vbl <= '0', '1' after CLOCK_HIGH, '0' after 2*CLOCK_HIGH;

        when S2    =>  CTRL_RST_L_MTJRR <= '1', '0' after CLOCK_HIGH;
                FIND <= '1';
                CTRL_MTJ_CELL <= '1';
                CTRL_Vop <= '1';
                CTRL_MASK_SHIFT <= '1';
                CTRL_Vbl <= '0', '1' after CLOCK_HIGH, '0' after 2*CLOCK_HIGH;

        when S3    =>  CTRL_RST_L_MTJRR <= '1', '0' after CLOCK_HIGH;
                FIND <= '1';
                CTRL_MTJ_CELL <= '1';
                CTRL_Vop <= '1';
                CTRL_MASK_RST <= '1';
                CTRL_Vbl <= '0', '1' after CLOCK_HIGH, '0' after 2*CLOCK_HIGH;

        when S4    =>  CTRL_RST_L_MTJRR <= '1', '0' after CLOCK_HIGH;
                CTRL_ADDREG_STORE <= '1';
                CTRL_ENCODER_EN <= '1';
                FIND <= '1';

        when S6    =>  CTRL_READWORD_EN <= '1';
                CTRL_Vop <= '1';
                EN_READ <= '1';

        when S7    =>  CTRL_MTJW <= '1', '0' after CLOCK_HIGH;
                CTRL_Vbl <= '0', '1' after CLOCK_PERIOD/11, '0' after
                ↪  2*CLOCK_PERIOD/11, '1' after 3*CLOCK_PERIOD/11, '0' after
                ↪  4*CLOCK_PERIOD/11, '1' after 5*CLOCK_PERIOD/11, '0' after
                ↪  6*CLOCK_PERIOD/11, '1' after 7*CLOCK_PERIOD/11, '0' after
                ↪  8*CLOCK_PERIOD/11, '1' after 9*CLOCK_PERIOD/11, '0' after
                ↪  10*CLOCK_PERIOD/11;
                CTRL_WORD_SHIFT <= '1';

        when S8    =>  CTRL_in_EN_2 <= '0';
                CTRL_MTJW <= '1', '0' after CLOCK_HIGH;
                CTRL_Vbl <= '0', '1' after CLOCK_PERIOD/7, '0' after
                ↪  2*CLOCK_PERIOD/7, '1' after 3*CLOCK_PERIOD/7, '0' after
                ↪  4*CLOCK_PERIOD/7, '1' after 5*CLOCK_PERIOD/7, '0' after
                ↪  6*CLOCK_PERIOD/7;
                CTRL_WORD_SHIFT <= '1';

        when S9    =>  CTRL_in_EN_2 <= '0';
                CTRL_in_EN_1 <= '0';
                CTRL_MTJW <= '1', '0' after CLOCK_HIGH;
                CTRL_Vbl <= '0', '1' after CLOCK_PERIOD/3, '0' after
                ↪  2*CLOCK_PERIOD/3;
```

```
              CTRL_WORD_SHIFT <= '1';

      when others =>  CTRL_Vbl      <= '0';
              CTRL_MTJW      <= '0';
              CTRL_in_EN_0    <= '1';
              CTRL_in_EN_1    <= '1';
              CTRL_in_EN_2    <= '1';
              CTRL_Vop      <= '0';
              CTRL_MTJ_CELL    <= '0';
              CTRL_Vtr_0      <= '0';
              CTRL_Vtr_1      <= '0';
              CTRL_Vtr_2      <= '0';
              CTRL_WORD_SHIFT    <= '0';
              CTRL_WORD_RST    <= '1';
              CTRL_RST_L_MTJRR  <= '1';
              CTRL_RST_L_MTJR    <= '1';
              CTRL_MASK_STORE    <= '0';
              CTRL_MASK_SHIFT    <= '0';
              CTRL_MASK_RST    <= '1';
              FIND        <= '0';
              IDLE        <= '0';
              EN_READ        <= '0';
              CTRL_READWORD_EN  <= '0';
              CTRL_READWORD_RST  <= '1';
              CTRL_LATCH_RST    <= '1';
              CTRL_ENCODER_EN    <= '0';
              CTRL_ADDREG_RST    <= '1';
              CTRL_ADDREG_STORE  <= '0';
    end case;
  end process output;
end Behaviour;
```

D.5. Memory architecture

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;

entity Memory is
port(  CLK:        in std_logic;
    RESET_n:      in std_logic;
```

```
    CTRL_DEC_en:      in std_logic;
    ADDRESS:        in std_logic_vector (1 downto 0);
    START_FIND:       in std_logic;
    READ_WRITE_n:     in std_logic;
    MAX_min_n:       in std_logic;
    CTRL_WORD0_STORE:  in std_logic;
    CTRL_WORD1_STORE:  in std_logic;
    CTRL_WORD2_STORE:  in std_logic;
    WORD0:         in std_logic_vector(2 downto 0);
    WORD1:         in std_logic_vector(2 downto 0);
    WORD2:         in std_logic_vector(2 downto 0);
    ADDRESS_FOUND_REGout: out std_logic_vector(1 downto 0));
end entity Memory;

architecture Behaviour of Memory is

  component FSM is
  port(  CLK:          in std_logic;
      RESET_n:        in std_logic;
      START_FIND:      in std_logic;
      READ_WRITE_n:      in std_logic;
      AAoutOR:        in std_logic;
      ---
      CTRL_Vbl:       out std_logic;
      CTRL_MTJW:       out std_logic;
      CTRL_in_EN_0:     out std_logic;
      CTRL_in_EN_1:     out std_logic;
      CTRL_in_EN_2:     out std_logic;
      CTRL_Vop:       out std_logic;
      CTRL_MTJ_CELL:     out std_logic;
      CTRL_Vtr_0:       out std_logic;
      CTRL_Vtr_1:       out std_logic;
      CTRL_Vtr_2:       out std_logic;
      CTRL_WORD_SHIFT:   out std_logic;
      CTRL_WORD_RST:     out std_logic;
      CTRL_RST_L_MTJRR:   out std_logic;
      CTRL_RST_L_MTJR:   out std_logic;
      CTRL_MASK_STORE:   out std_logic;
      CTRL_MASK_SHIFT:   out std_logic;
      CTRL_MASK_RST:     out std_logic;
      FIND:         out std_logic;
      IDLE:         out std_logic;
      EN_READ:        out std_logic;
      CTRL_READWORD_EN:   out std_logic;
      CTRL_READWORD_RST:   out std_logic;
      CTRL_LATCH_RST:     out std_logic;
      CTRL_ENCODER_EN:   out std_logic;
      CTRL_ADDREG_RST:   out std_logic;
      CTRL_ADDREG_STORE:   out std_logic);
```

```
end component FSM;

component MemArray is
port(  CTRL_Vbl_0: in std_logic;
    CTRL_Vop_0: in std_logic;
    CTRL_Vtr_00: in std_logic;
    CTRL_Vtr_01: in std_logic;
    CTRL_Vtr_02: in std_logic;
    CTRL_MTJW_0: in std_logic;
    CTRL_in_EN_00: in std_logic;
    CTRL_in_EN_01: in std_logic;
    CTRL_in_EN_02: in std_logic;
    CTRL_MTJ_cell_00: in std_logic;
    CTRL_MTJ_cell_01: in std_logic;
    CTRL_MTJ_cell_02: in std_logic;
    MTJ_RR0_out: out std_logic;
    MTJ_R00_out: out std_logic;
    MTJ_R01_out: out std_logic;
    MTJ_R02_out: out std_logic;
    ---
    CTRL_Vbl_1: in std_logic;
    CTRL_Vop_1: in std_logic;
    CTRL_Vtr_10: in std_logic;
    CTRL_Vtr_11: in std_logic;
    CTRL_Vtr_12: in std_logic;
    CTRL_MTJW_1: in std_logic;
    CTRL_in_EN_10: in std_logic;
    CTRL_in_EN_11: in std_logic;
    CTRL_in_EN_12: in std_logic;
    CTRL_MTJ_cell_10: in std_logic;
    CTRL_MTJ_cell_11: in std_logic;
    CTRL_MTJ_cell_12: in std_logic;
    MTJ_RR1_out: out std_logic;
    MTJ_R10_out: out std_logic;
    MTJ_R11_out: out std_logic;
    MTJ_R12_out: out std_logic;
    ---
    CTRL_Vbl_2: in std_logic;
    CTRL_Vop_2: in std_logic;
    CTRL_Vtr_20: in std_logic;
    CTRL_Vtr_21: in std_logic;
    CTRL_Vtr_22: in std_logic;
    CTRL_MTJW_2: in std_logic;
    CTRL_in_EN_20: in std_logic;
    CTRL_in_EN_21: in std_logic;
    CTRL_in_EN_22: in std_logic;
    CTRL_MTJ_cell_20: in std_logic;
    CTRL_MTJ_cell_21: in std_logic;
    CTRL_MTJ_cell_22: in std_logic;
```

```
        MTJ_RR2_out: out std_logic;
        MTJ_R20_out: out std_logic;
        MTJ_R21_out: out std_logic;
        MTJ_R22_out: out std_logic);
    end component MemArray;

    component Detector is
    port(  L_MTJ_RR_out_0:   in std_logic;
        L_MTJ_RR_out_1:   in std_logic;
        L_MTJ_RR_out_2:   in std_logic;
        LATCH0_OUT:      in std_logic;
        LATCH1_OUT:      in std_logic;
        LATCH2_OUT:      in std_logic;
        ENABLE_Rowdis:    out std_logic);
    end component Detector;

    component Row_disabler is
    port(  L_MTJ_RR_out_0:   in std_logic;
        L_MTJ_RR_out_1:   in std_logic;
        L_MTJ_RR_out_2:   in std_logic;
        MAX_min_n:      in std_logic;
        ENABLE_Rowdis:    in std_logic;
        FIND:        in std_logic;
        CTRL_LATCH_RST:    in std_logic;
        LATCH0_OUT:      out std_logic;
        LATCH1_OUT:      out std_logic;
        LATCH2_OUT:      out std_logic);
    end component Row_disabler;

    component And_array is
    port(  RESET_n:    in std_logic;
        IDLE:      in std_logic;
        FIND:      in std_logic;
        DEC0:       in std_logic;
        DEC1:       in std_logic;
        DEC2:       in std_logic;
        LATCH0_OUT:    in std_logic;
        LATCH1_OUT:    in std_logic;
        LATCH2_OUT:    in std_logic;
        AAout0:      buffer std_logic;
        AAout1:      buffer std_logic;
        AAout2:      buffer std_logic;
        AAoutOR:    out std_logic);
    end component And_array;

    component Decoder is
    generic (N: integer := 3);
    port(  ENABLE:   in std_logic;
        ADDRESS:   in std_logic_vector(N-1 downto 0);
```

```
        WLINES:    out std_logic_vector(2**N-1 downto 0) );
    end component Decoder;

    component Encoder is
    port(  CTRL_ENCODER_EN:    in std_logic;
        LATCH0_OUT:       in std_logic;
        LATCH1_OUT:       in std_logic;
        LATCH2_OUT:       in std_logic;
        CLK:          in std_logic;
        CTRL_ADDREG_RST:    in std_logic;
        CTRL_ADDREG_STORE:    in std_logic;
        ADDRESS_FOUND_REGout:   out std_logic_vector(1 downto 0) );
    end component Encoder;

    component Shift_reg is
    generic (N: integer:= 3);
    port(  CLK:   in std_logic;
        RST:   in std_logic;
        SHIFT:  in std_logic;
        STORE:  in std_logic;
        RIGHT_LEFT_n: in std_logic;
        DATA_IN: in std_logic_vector (N-1 downto 0);
        DATA_OUT: buffer std_logic_vector (N-1 downto 0) );
    end component Shift_reg;

    component memarray_latches is
    port(  MTJ_R00_out, MTJ_R01_out, MTJ_R02_out: in std_logic;
        MTJ_R10_out, MTJ_R11_out, MTJ_R12_out: in std_logic;
        MTJ_R20_out, MTJ_R21_out, MTJ_R22_out: in std_logic;
        EN_READ: in std_logic;
        CTRL_RST_L_MTJR_0: in std_logic;
        CTRL_RST_L_MTJR_1: in std_logic;
        CTRL_RST_L_MTJR_2: in std_logic;
        L_MTJ_R_out_00, L_MTJ_R_out_01, L_MTJ_R_out_02: out std_logic;
        L_MTJ_R_out_10, L_MTJ_R_out_11, L_MTJ_R_out_12: out std_logic;
        L_MTJ_R_out_20, L_MTJ_R_out_21, L_MTJ_R_out_22: out std_logic;
        ---
        MTJ_RR0_out: in std_logic;
        MTJ_RR1_out: in std_logic;
        MTJ_RR2_out: in std_logic;
        CTRL_RST_L_MTJRR_0: in std_logic;
        CTRL_RST_L_MTJRR_1: in std_logic;
        CTRL_RST_L_MTJRR_2: in std_logic;
        L_MTJ_RR_out_0: out std_logic;
        L_MTJ_RR_out_1: out std_logic;
        L_MTJ_RR_out_2: out std_logic);
    end component memarray_latches;

    component readout is
```

```
port(  L_MTJ_R_out_00, L_MTJ_R_out_01, L_MTJ_R_out_02: in std_logic;
    L_MTJ_R_out_10, L_MTJ_R_out_11, L_MTJ_R_out_12: in std_logic;
    L_MTJ_R_out_20, L_MTJ_R_out_21, L_MTJ_R_out_22: in std_logic;
    AAout0:        in std_logic;
    AAout1:        in std_logic;
    AAout2:        in std_logic;
    CLK:         in std_logic;
    CTRL_READWORD_RST:  in std_logic;
    CTRL_READWORD_EN:  in std_logic;
    READWORD_OUT:     out std_logic_vector (2 downto 0) );
end component readout;

component FSM_Array_adapter is
port(  AAout0:      in std_logic;
    AAout1:      in std_logic;
    AAout2:      in std_logic;
    ---
    CTRL_Vbl:    in std_logic;
    CTRL_Vbl_0:    out std_logic;
    CTRL_Vbl_1:    out std_logic;
    CTRL_Vbl_2:    out std_logic;
    ---
    CTRL_MTJW:      in std_logic;
    SHIFT_WORD_OUT_0:   in std_logic;
    SHIFT_WORD_OUT_1:   in std_logic;
    SHIFT_WORD_OUT_2:   in std_logic;
    CTRL_MTJW_0:    out std_logic;
    CTRL_MTJW_1:    out std_logic;
    CTRL_MTJW_2:    out std_logic;
    ---
    CTRL_in_EN_0:    in std_logic;
    CTRL_in_EN_1:    in std_logic;
    CTRL_in_EN_2:    in std_logic;
    CTRL_in_EN_00:    out std_logic;
    CTRL_in_EN_01:    out std_logic;
    CTRL_in_EN_02:    out std_logic;
    CTRL_in_EN_10:    out std_logic;
    CTRL_in_EN_11:    out std_logic;
    CTRL_in_EN_12:    out std_logic;
    CTRL_in_EN_20:    out std_logic;
    CTRL_in_EN_21:    out std_logic;
    CTRL_in_EN_22:    out std_logic;
    ---
    CTRL_Vop:      in std_logic;
    CTRL_Vop_0:      out std_logic;
    CTRL_Vop_1:      out std_logic;
    CTRL_Vop_2:      out std_logic;
    ---
    CTRL_MTJ_CELL:    in std_logic;
```

```
        SHIFT_MASK:      in std_logic_vector(2 downto 0);
        CTRL_MTJ_CELL_00:  out std_logic;
        CTRL_MTJ_CELL_01:  out std_logic;
        CTRL_MTJ_CELL_02:  out std_logic;
        CTRL_MTJ_CELL_10:  out std_logic;
        CTRL_MTJ_CELL_11:  out std_logic;
        CTRL_MTJ_CELL_12:  out std_logic;
        CTRL_MTJ_CELL_20:  out std_logic;
        CTRL_MTJ_CELL_21:  out std_logic;
        CTRL_MTJ_CELL_22:  out std_logic;
        ---
        CTRL_Vtr_0:      in std_logic;
        CTRL_Vtr_1:      in std_logic;
        CTRL_Vtr_2:      in std_logic;
        CTRL_Vtr_00:    out std_logic;
        CTRL_Vtr_01:    out std_logic;
        CTRL_Vtr_02:    out std_logic;
        CTRL_Vtr_10:    out std_logic;
        CTRL_Vtr_11:    out std_logic;
        CTRL_Vtr_12:    out std_logic;
        CTRL_Vtr_20:    out std_logic;
        CTRL_Vtr_21:    out std_logic;
        CTRL_Vtr_22:    out std_logic;
        ---
        CTRL_RST_L_MTJRR:  in std_logic;
        CTRL_RST_L_MTJRR_0:  out std_logic;
        CTRL_RST_L_MTJRR_1:  out std_logic;
        CTRL_RST_L_MTJRR_2: out std_logic;
        ---
        CTRL_RST_L_MTJR:  in std_logic;
        CTRL_RST_L_MTJR_0: out std_logic;
        CTRL_RST_L_MTJR_1: out std_logic;
        CTRL_RST_L_MTJR_2: out std_logic);
    end component FSM_Array_adapter;

    signal AAout0, AAout1, AAout2, AAoutOR, L_AAoutOR, L_AAoutOR0, L_AAoutOR1,
    ↪  L_AAoutOR2: std_logic;
    signal CTRL_Vbl, CTRL_Vbl_0, CTRL_Vbl_1, CTRL_Vbl_2: std_logic;
    signal CTRL_MTJW, CTRL_MTJW_0, CTRL_MTJW_1, CTRL_MTJW_2: std_logic;
    signal CTRL_in_EN_0, CTRL_in_EN_1, CTRL_in_EN_2, CTRL_in_EN_00, CTRL_in_EN_01,
    ↪  CTRL_in_EN_02, CTRL_in_EN_10, CTRL_in_EN_11, CTRL_in_EN_12, CTRL_in_EN_20,
    ↪  CTRL_in_EN_21, CTRL_in_EN_22: std_logic;
    signal CTRL_Vop, CTRL_Vop_0, CTRL_Vop_1, CTRL_Vop_2: std_logic;
    signal CTRL_MTJ_CELL, CTRL_MTJ_CELL_00, CTRL_MTJ_CELL_01, CTRL_MTJ_CELL_02,
    ↪  CTRL_MTJ_CELL_10, CTRL_MTJ_CELL_11, CTRL_MTJ_CELL_12, CTRL_MTJ_CELL_20,
    ↪  CTRL_MTJ_CELL_21, CTRL_MTJ_CELL_22: std_logic;
    signal CTRL_Vtr_0, CTRL_Vtr_1, CTRL_Vtr_2, CTRL_Vtr_00, CTRL_Vtr_01,
    ↪  CTRL_Vtr_02, CTRL_Vtr_10, CTRL_Vtr_11, CTRL_Vtr_12, CTRL_Vtr_20,
    ↪  CTRL_Vtr_21, CTRL_Vtr_22: std_logic;
```

```vhdl
  signal CTRL_WORD_SHIFT, CTRL_WORD0_SHIFT, CTRL_WORD1_SHIFT, CTRL_WORD2_SHIFT: std_logic;
  signal CTRL_RST_L_MTJRR, CTRL_RST_L_MTJRR_0, CTRL_RST_L_MTJRR_1, CTRL_RST_L_MTJRR_2: std_logic;
  signal CTRL_RST_L_MTJR, CTRL_RST_L_MTJR_0, CTRL_RST_L_MTJR_1, CTRL_RST_L_MTJR_2: std_logic;

  signal L_MTJ_RR_out_0, L_MTJ_RR_out_1, L_MTJ_RR_out_2, L_MTJ_R_out_00, L_MTJ_R_out_01, L_MTJ_R_out_02, L_MTJ_R_out_10, L_MTJ_R_out_11, L_MTJ_R_out_12, L_MTJ_R_out_20, L_MTJ_R_out_21, L_MTJ_R_out_22, LATCH0_OUT, LATCH1_OUT, LATCH2_OUT, ENABLE_Rowdis, DEC0, DEC1, DEC2: std_logic;
  signal MTJ_RR0_out, MTJ_R00_out, MTJ_R01_out, MTJ_R02_out, MTJ_RR1_out, MTJ_R10_out, MTJ_R11_out, MTJ_R12_out, MTJ_RR2_out, MTJ_R20_out, MTJ_R21_out, MTJ_R22_out: std_logic;
  signal FIND, IDLE, EN_READ, CTRL_READWORD_EN, CTRL_READWORD_RST, CTRL_LATCH_RST, CTRL_ENCODER_EN: std_logic;
  signal CTRL_ADDREG_RST, CTRL_ADDREG_STORE: std_logic;

  signal CTRL_MASK_RST, CTRL_MASK_SHIFT, CTRL_MASK_STORE: std_logic;
  signal CTRL_WORD_RST: std_logic;
  signal WLINES, ENC_INPUTS: std_logic_vector(3 downto 0);
  signal MASK_OUT, WORD0_OUT, WORD1_OUT, WORD2_OUT: std_logic_vector(2 downto 0);
  signal ADDRESS_FOUND: std_logic_vector(1 downto 0);
  signal READWORD_OUT: std_logic_vector(2 downto 0);
begin

  FSM_mem: FSM port map (
    CLK => CLK,
    RESET_n => RESET_n,
    START_FIND => START_FIND,
    READ_WRITE_n => READ_WRITE_n,
    AAoutOR => AAoutOR,
    ---
    CTRL_Vbl => CTRL_Vbl,
    CTRL_MTJW => CTRL_MTJW,
    CTRL_in_EN_0 => CTRL_in_EN_0,
    CTRL_in_EN_1 => CTRL_in_EN_1,
    CTRL_in_EN_2 => CTRL_in_EN_2,
    CTRL_Vop => CTRL_Vop,
    CTRL_MTJ_CELL => CTRL_MTJ_CELL,
    CTRL_Vtr_0 => CTRL_Vtr_0,
    CTRL_Vtr_1 => CTRL_Vtr_1,
    CTRL_Vtr_2 => CTRL_Vtr_2,
    CTRL_WORD_SHIFT => CTRL_WORD_SHIFT,
    CTRL_WORD_RST => CTRL_WORD_RST,
    CTRL_RST_L_MTJRR => CTRL_RST_L_MTJRR,
    CTRL_RST_L_MTJR => CTRL_RST_L_MTJR,
```

```
    CTRL_MASK_STORE => CTRL_MASK_STORE,
    CTRL_MASK_SHIFT => CTRL_MASK_SHIFT,
    CTRL_MASK_RST => CTRL_MASK_RST,
    FIND => FIND,
    IDLE => IDLE,
    EN_READ => EN_READ,
    CTRL_READWORD_EN => CTRL_READWORD_EN,
    CTRL_READWORD_RST => CTRL_READWORD_RST,
    CTRL_LATCH_RST => CTRL_LATCH_RST,
    CTRL_ENCODER_EN => CTRL_ENCODER_EN,
    CTRL_ADDREG_RST => CTRL_ADDREG_RST,
    CTRL_ADDREG_STORE => CTRL_ADDREG_STORE);

  MEM: MemArray port map (
    CTRL_Vbl_0 => CTRL_Vbl_0,
    CTRL_Vop_0 => CTRL_Vop_0,
    CTRL_Vtr_00 => CTRL_Vtr_00,
    CTRL_Vtr_01 => CTRL_Vtr_01,
    CTRL_Vtr_02 => CTRL_Vtr_02,
    CTRL_MTJW_0 => CTRL_MTJW_0,
    CTRL_in_EN_00 => CTRL_in_EN_00,
    CTRL_in_EN_01 => CTRL_in_EN_01,
    CTRL_in_EN_02 => CTRL_in_EN_02,
    CTRL_MTJ_cell_00 => CTRL_MTJ_cell_00,
    CTRL_MTJ_cell_01 => CTRL_MTJ_cell_01,
    CTRL_MTJ_cell_02 => CTRL_MTJ_cell_02,
    MTJ_RR0_out => MTJ_RR0_out,
    MTJ_R00_out => MTJ_R00_out,
    MTJ_R01_out => MTJ_R01_out,
    MTJ_R02_out => MTJ_R02_out,
    ---
    CTRL_Vbl_1 => CTRL_Vbl_1,
    CTRL_Vop_1 => CTRL_Vop_1,
    CTRL_Vtr_10 => CTRL_Vtr_10,
    CTRL_Vtr_11 => CTRL_Vtr_11,
    CTRL_Vtr_12 => CTRL_Vtr_12,
    CTRL_MTJW_1 => CTRL_MTJW_1,
    CTRL_in_EN_10 => CTRL_in_EN_10,
    CTRL_in_EN_11 => CTRL_in_EN_11,
    CTRL_in_EN_12 => CTRL_in_EN_12,
    CTRL_MTJ_cell_10 => CTRL_MTJ_cell_10,
    CTRL_MTJ_cell_11 => CTRL_MTJ_cell_11,
    CTRL_MTJ_cell_12 => CTRL_MTJ_cell_12,
    MTJ_RR1_out => MTJ_RR1_out,
    MTJ_R10_out => MTJ_R10_out,
    MTJ_R11_out => MTJ_R11_out,
    MTJ_R12_out => MTJ_R12_out,
    ---
    CTRL_Vbl_2 => CTRL_Vbl_2,
```

```
    CTRL_Vop_2 => CTRL_Vop_2,
    CTRL_Vtr_20 => CTRL_Vtr_20,
    CTRL_Vtr_21 => CTRL_Vtr_21,
    CTRL_Vtr_22 => CTRL_Vtr_22,
    CTRL_MTJW_2 => CTRL_MTJW_2,
    CTRL_in_EN_20 => CTRL_in_EN_20,
    CTRL_in_EN_21 => CTRL_in_EN_21,
    CTRL_in_EN_22 => CTRL_in_EN_22,
    CTRL_MTJ_cell_20 => CTRL_MTJ_cell_20,
    CTRL_MTJ_cell_21 => CTRL_MTJ_cell_21,
    CTRL_MTJ_cell_22 => CTRL_MTJ_cell_22,
    MTJ_RR2_out => MTJ_RR2_out,
    MTJ_R20_out => MTJ_R20_out,
    MTJ_R21_out => MTJ_R21_out,
    MTJ_R22_out => MTJ_R22_out);

  latches: memarray_latches port map (
    MTJ_R00_out => MTJ_R00_out,
    MTJ_R01_out => MTJ_R01_out,
    MTJ_R02_out => MTJ_R02_out,
    MTJ_R10_out => MTJ_R10_out,
    MTJ_R11_out => MTJ_R11_out,
    MTJ_R12_out => MTJ_R12_out,
    MTJ_R20_out => MTJ_R20_out,
    MTJ_R21_out => MTJ_R21_out,
    MTJ_R22_out => MTJ_R22_out,
    EN_READ => EN_READ,
    CTRL_RST_L_MTJR_0 => CTRL_RST_L_MTJR_0,
    CTRL_RST_L_MTJR_1 => CTRL_RST_L_MTJR_1,
    CTRL_RST_L_MTJR_2 => CTRL_RST_L_MTJR_2,
    L_MTJ_R_out_00 => L_MTJ_R_out_00,
    L_MTJ_R_out_01 => L_MTJ_R_out_01,
    L_MTJ_R_out_02 => L_MTJ_R_out_02,
    L_MTJ_R_out_10 => L_MTJ_R_out_10,
    L_MTJ_R_out_11 => L_MTJ_R_out_11,
    L_MTJ_R_out_12 => L_MTJ_R_out_12,
    L_MTJ_R_out_20 => L_MTJ_R_out_20,
    L_MTJ_R_out_21 => L_MTJ_R_out_21,
    L_MTJ_R_out_22 => L_MTJ_R_out_22,
    ---
    MTJ_RR0_out => MTJ_RR0_out,
    MTJ_RR1_out => MTJ_RR1_out,
    MTJ_RR2_out => MTJ_RR2_out,
    CTRL_RST_L_MTJRR_0 => CTRL_RST_L_MTJRR_0,
    CTRL_RST_L_MTJRR_1 => CTRL_RST_L_MTJRR_1,
    CTRL_RST_L_MTJRR_2 => CTRL_RST_L_MTJRR_2,
    L_MTJ_RR_out_0 => L_MTJ_RR_out_0,
    L_MTJ_RR_out_1 => L_MTJ_RR_out_1,
    L_MTJ_RR_out_2 => L_MTJ_RR_out_2);
```

```
  read_out: readout port map (
    L_MTJ_R_out_00 => L_MTJ_R_out_00,
    L_MTJ_R_out_01 => L_MTJ_R_out_01,
    L_MTJ_R_out_02 => L_MTJ_R_out_02,
    L_MTJ_R_out_10 => L_MTJ_R_out_10,
    L_MTJ_R_out_11 => L_MTJ_R_out_11,
    L_MTJ_R_out_12 => L_MTJ_R_out_12,
    L_MTJ_R_out_20 => L_MTJ_R_out_20,
    L_MTJ_R_out_21 => L_MTJ_R_out_21,
    L_MTJ_R_out_22 => L_MTJ_R_out_22,
    AAout0 => AAout0,
    AAout1 => AAout1,
    AAout2 => AAout2,
    CLK => CLK,
    CTRL_READWORD_RST => CTRL_READWORD_RST,
    CTRL_READWORD_EN => CTRL_READWORD_EN,
    READWORD_OUT => READWORD_OUT);

  DET: Detector port map (
    L_MTJ_RR_out_0 => L_MTJ_RR_out_0,
    L_MTJ_RR_out_1 => L_MTJ_RR_out_1,
    L_MTJ_RR_out_2 => L_MTJ_RR_out_2,
    LATCH0_OUT => LATCH0_OUT,
    LATCH1_OUT => LATCH1_OUT,
    LATCH2_OUT => LATCH2_OUT,
    ENABLE_Rowdis => ENABLE_Rowdis);

  DIS: Row_disabler port map (
    L_MTJ_RR_out_0 => L_MTJ_RR_out_0,
    L_MTJ_RR_out_1 => L_MTJ_RR_out_1,
    L_MTJ_RR_out_2 => L_MTJ_RR_out_2,
    MAX_min_n => MAX_min_n,
    ENABLE_Rowdis => ENABLE_Rowdis,
    FIND => FIND,
    CTRL_LATCH_RST => CTRL_LATCH_RST,
    LATCH0_OUT => LATCH0_OUT,
    LATCH1_OUT => LATCH1_OUT,
    LATCH2_OUT => LATCH2_OUT);

  ENC: Encoder port map (
    CTRL_ENCODER_EN => CTRL_ENCODER_EN,
    LATCH0_OUT => LATCH0_OUT,
    LATCH1_OUT => LATCH1_OUT,
    LATCH2_OUT => LATCH2_OUT,
    CLK => CLK,
    CTRL_ADDREG_RST => CTRL_ADDREG_RST,
    CTRL_ADDREG_STORE => CTRL_ADDREG_STORE,
    ADDRESS_FOUND_REGout => ADDRESS_FOUND_REGout);
```

```

DEC: Decoder generic map (N => 2) port map (
  ENABLE => CTRL_DEC_en,
  ADDRESS => ADDRESS,
  WLINES => WLINES);

DEC0 <= WLINES(0);
DEC1 <= WLINES(1);
DEC2 <= WLINES(2);

ANDARR: And_array port map (
  RESET_n => RESET_n,
  IDLE => IDLE,
  FIND => FIND,
  DEC0 => DEC0,
  DEC1 => DEC1,
  DEC2 => DEC2,
  LATCH0_OUT => LATCH0_OUT,
  LATCH1_OUT => LATCH1_OUT,
  LATCH2_OUT => LATCH2_OUT,
  AAout0 => AAout0,
  AAout1 => AAout1,
  AAout2 => AAout2,
  AAoutOR => AAoutOR);

MASK_REG: Shift_reg generic map (N => 3) port map (
  CLK => CLK,
  RST => CTRL_MASK_RST,
  SHIFT => CTRL_MASK_SHIFT,
  STORE => CTRL_MASK_STORE,
  RIGHT_LEFT_n => '1',
  DATA_IN => "100",
  DATA_OUT => MASK_OUT);

WORD0_REG: Shift_reg generic map (N => 3) port map (
  CLK => CLK,
  RST => CTRL_WORD_RST,
  SHIFT => CTRL_WORD0_SHIFT,
  STORE => CTRL_WORD0_STORE,
  RIGHT_LEFT_n => '0',
  DATA_IN => WORD0,
  DATA_OUT => WORD0_OUT);

WORD1_REG: Shift_reg generic map (N => 3) port map (
  CLK => CLK,
  RST => CTRL_WORD_RST,
  SHIFT => CTRL_WORD1_SHIFT,
  STORE => CTRL_WORD1_STORE,
  RIGHT_LEFT_n => '0',
```

```
    DATA_IN => WORD1,
    DATA_OUT => WORD1_OUT);

  WORD2_REG: Shift_reg generic map (N => 3) port map (
    CLK => CLK,
    RST => CTRL_WORD_RST,
    SHIFT => CTRL_WORD2_SHIFT,
    STORE => CTRL_WORD2_STORE,
    RIGHT_LEFT_n => '0',
    DATA_IN => WORD2,
    DATA_OUT => WORD2_OUT);

  FSM_Array_adapt: FSM_Array_adapter port map (
    AAout0 => AAout0,
    AAout1 => AAout1,
    AAout2 => AAout2,
    ---
    CTRL_Vbl => CTRL_Vbl,
    CTRL_Vbl_0 => CTRL_Vbl_0,
    CTRL_Vbl_1 => CTRL_Vbl_1,
    CTRL_Vbl_2 => CTRL_Vbl_2,
    ---
    CTRL_MTJW => CTRL_MTJW,
    SHIFT_WORD_OUT_0 => WORD0_OUT(2),
    SHIFT_WORD_OUT_1 => WORD1_OUT(2),
    SHIFT_WORD_OUT_2 => WORD2_OUT(2),
    CTRL_MTJW_0 => CTRL_MTJW_0,
    CTRL_MTJW_1 => CTRL_MTJW_1,
    CTRL_MTJW_2 => CTRL_MTJW_2,
    ---
    CTRL_in_EN_0 => CTRL_in_EN_0,
    CTRL_in_EN_1 => CTRL_in_EN_1,
    CTRL_in_EN_2 => CTRL_in_EN_2,
    CTRL_in_EN_00 => CTRL_in_EN_00,
    CTRL_in_EN_01 => CTRL_in_EN_01,
    CTRL_in_EN_02 => CTRL_in_EN_02,
    CTRL_in_EN_10 => CTRL_in_EN_10,
    CTRL_in_EN_11 => CTRL_in_EN_11,
    CTRL_in_EN_12 => CTRL_in_EN_12,
    CTRL_in_EN_20 => CTRL_in_EN_20,
    CTRL_in_EN_21 => CTRL_in_EN_21,
    CTRL_in_EN_22 => CTRL_in_EN_22,
    ---
    CTRL_Vop => CTRL_Vop,
    CTRL_Vop_0 => CTRL_Vop_0,
    CTRL_Vop_1 => CTRL_Vop_1,
    CTRL_Vop_2 => CTRL_Vop_2,
    ---
    CTRL_MTJ_CELL => CTRL_MTJ_CELL,
```

```
    SHIFT_MASK => MASK_OUT,
    CTRL_MTJ_CELL_00 => CTRL_MTJ_CELL_00,
    CTRL_MTJ_CELL_01 => CTRL_MTJ_CELL_01,
    CTRL_MTJ_CELL_02 => CTRL_MTJ_CELL_02,
    CTRL_MTJ_CELL_10 => CTRL_MTJ_CELL_10,
    CTRL_MTJ_CELL_11 => CTRL_MTJ_CELL_11,
    CTRL_MTJ_CELL_12 => CTRL_MTJ_CELL_12,
    CTRL_MTJ_CELL_20 => CTRL_MTJ_CELL_20,
    CTRL_MTJ_CELL_21 => CTRL_MTJ_CELL_21,
    CTRL_MTJ_CELL_22 => CTRL_MTJ_CELL_22,
    ---
    CTRL_Vtr_0 => CTRL_Vtr_0,
    CTRL_Vtr_1 => CTRL_Vtr_1,
    CTRL_Vtr_2 => CTRL_Vtr_2,
    CTRL_Vtr_00 => CTRL_Vtr_00,
    CTRL_Vtr_01 => CTRL_Vtr_01,
    CTRL_Vtr_02 => CTRL_Vtr_02,
    CTRL_Vtr_10 => CTRL_Vtr_10,
    CTRL_Vtr_11 => CTRL_Vtr_11,
    CTRL_Vtr_12 => CTRL_Vtr_12,
    CTRL_Vtr_20 => CTRL_Vtr_20,
    CTRL_Vtr_21 => CTRL_Vtr_21,
    CTRL_Vtr_22 => CTRL_Vtr_22,
    ---
    CTRL_RST_L_MTJRR => CTRL_RST_L_MTJRR,
    CTRL_RST_L_MTJRR_0 => CTRL_RST_L_MTJRR_0,
    CTRL_RST_L_MTJRR_1 => CTRL_RST_L_MTJRR_1,
    CTRL_RST_L_MTJRR_2 => CTRL_RST_L_MTJRR_2,
    ---
    CTRL_RST_L_MTJR => CTRL_RST_L_MTJR,
    CTRL_RST_L_MTJR_0 => CTRL_RST_L_MTJR_0,
    CTRL_RST_L_MTJR_1 => CTRL_RST_L_MTJR_1,
    CTRL_RST_L_MTJR_2 => CTRL_RST_L_MTJR_2);

end architecture Behaviour;
```

D.6. Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.globals.all;
```

```vhdl

entity tb_mem is
end entity tb_mem;

architecture Behaviour of tb_mem is
  component Memory is
  port(  CLK:        in std_logic;
      RESET_n:      in std_logic;
      CTRL_DEC_en:     in std_logic;
      ADDRESS:       in std_logic_vector (1 downto 0);
      START_FIND:      in std_logic;
      READ_WRITE_n:    in std_logic;
      MAX_min_n:      in std_logic;
      CTRL_WORD0_STORE:  in std_logic;
      CTRL_WORD1_STORE:  in std_logic;
      CTRL_WORD2_STORE:  in std_logic;
      WORD0:        in std_logic_vector(2 downto 0);
      WORD1:        in std_logic_vector(2 downto 0);
      WORD2:        in std_logic_vector(2 downto 0);
      ADDRESS_FOUND_REGout: out std_logic_vector(1 downto 0));
  end component Memory;

  signal CLK, RESET_n, CTRL_DEC_en, CTRL_WORD0_STORE, CTRL_WORD1_STORE, CTRL_WORD2_STORE, START_FIND, READ_WRITE_n, MAX_min_n: std_logic;
  signal ADDRESS, ADDRESS_FOUND_REGout: std_logic_vector(1 downto 0);
  signal WORD0, WORD1, WORD2: std_logic_vector(2 downto 0);

begin
  RESET_n <= '0', '1' after 2*CLOCK_HIGH;
  CTRL_DEC_en <= '0', '1' after 5*CLOCK_PERIOD;
  ADDRESS <= "00", "01" after 15*CLOCK_PERIOD, "10" after 30*CLOCK_PERIOD;
  WORD0 <= "111";
  CTRL_WORD0_STORE <= '0', '1' after 4*CLOCK_PERIOD, '0' after 5*CLOCK_PERIOD;
  WORD1 <= "011";
  CTRL_WORD1_STORE <= '0', '1' after 4*CLOCK_PERIOD, '0' after 5*CLOCK_PERIOD;
  WORD2 <= "101";
  CTRL_WORD2_STORE <= '0', '1' after 4*CLOCK_PERIOD, '0' after 5*CLOCK_PERIOD;
  START_FIND <= '0', '1' after 50*CLOCK_PERIOD, '0' after 51*CLOCK_PERIOD;
  READ_WRITE_n <= '0', '1' after 60*CLOCK_PERIOD;
  MAX_min_n <= '1';

  DUT: Memory port map (
    CLK => CLK,
    RESET_n => RESET_n,
    CTRL_DEC_en => CTRL_DEC_en,
    ADDRESS => ADDRESS,
    START_FIND => START_FIND,
    READ_WRITE_n => READ_WRITE_n,
    MAX_min_n => MAX_min_n,
```

```
    CTRL_WORD0_STORE => CTRL_WORD0_STORE,
    CTRL_WORD1_STORE => CTRL_WORD1_STORE,
    CTRL_WORD2_STORE => CTRL_WORD2_STORE,
    WORD0 => WORD0,
    WORD1 => WORD1,
    WORD2 => WORD2,
    ADDRESS_FOUND_REGout => ADDRESS_FOUND_REGout);

  clk_gen: process
  begin
    CLK <= '1';
    wait for CLOCK_HIGH;
    CLK <= '0';
    wait for CLOCK_LOW;
  end process clk_gen;

end architecture Behaviour;
```