POLITECNICO DI TORINO

SCUOLA DI DOTTORATO
Dottorato in Ingegneria Elettronica e delle Comunicazioni – XXVIII ciclo

Tesi di Dottorato

Architectural Solutions for NanoMagnet Logic

Giovanni Causapruno

Tutore
prof. Maurizio Zamboni

Coordinatore del corso di dottorato
prof. Ivo Montrosset

Maggio 2016

Summary

The successful era of CMOS technology is coming to an end. The limit on minimum fabrication dimensions of transistors and the increasing leakage power hinder the technological scaling that has characterized the last decades. In several different ways, this problem has been addressed changing the architectures implemented in CMOS, adopting parallel processors and thus increasing the throughput at the same operating frequency. However, architectural alternatives cannot be the definitive answer to a continuous increase in performance dictated by Moore's law. This problem must be addressed from a technological point of view.

Several alternative technologies that could substitute CMOS in next years are currently under study. Among them, magnetic technologies such as NanoMagnet Logic (NML) are interesting because they do not dissipate any leakage power. Moreover, magnets have memory capability, so it is possible to merge logic and memory in the same device.

However, magnetic circuits, and NML in this specific research, have also some important drawbacks that need to be addressed: first, the circuit clock frequency is limited to 100 MHz, to avoid errors in data propagation; second, there is a connection between circuit layout and timing, and in particular, longer wires will have longer latency. These drawbacks are intrinsic to the technology and for this reason they cannot be avoided. The only chance is to limit their impact from an architectural point of view.

The first step followed in the research path of this thesis is indeed the choice and optimization of architectures able to deal with the problems of NML. Systolic Arrays are identified as an ideal solution for this technology, because they are regular structures with local interconnections that limit the long latency of wires; moreover they are composed of several Processing Elements that work in parallel, thus exploit parallelization to increase throughput (limiting the impact of the low clock frequency). Through the analysis of Systolic Arrays for NML, several possible improvements have been identified and addressed: 1) it has been defined a rigorous way to increase throughput with interleaving, providing equations that allow to estimate the number of operations to be interleaved and the rules to provide inputs; 2) a latency insensitive circuit has been designed, that exploits a data communication

protocol between processing elements to avoid data synchronization problems. This feature has been exploited to design a latency insensitive Systolic Array that is able to execute the Floyd-Steinberg dithering algorithm. All the improvements presented in this framework apply to Systolic Arrays implemented in any technology. So, they can also be exploited to increase performance of today's CMOS parallel circuits. This research path is presented in Chapter 3.

While Systolic Arrays are an interesting solution for NML, their usage could be quite limited because they are normally application-specific. The second research path addresses this problem. A Reconfigurable Systolic Array is presented, that can be programmed to execute several algorithms. This architecture has been tested implementing many algorithms, including FIR and IIR filters, Discrete Cosine Transform and Matrix Multiplication. This research path is presented in Chapter 4.

In common Von Neumann architectures, the logic part of the circuit and the memory one are separated. Today bus communication between logic and memory represents the bottleneck of the system. This problem is addressed presenting Logic-In-Memory (LIM), an architecture where memory elements are merged in logic ones. This research path aims at defining a real LIM architectures. This has been done in two steps. The first step is represented by an architecture composed of three layers: memory, routing and logic. In the second step instead the routing plane is no more present, and its features are inherited by the memory plane. In this solution, a pyramidal memory model is used, where memories near logic elements contain the most probably used data, and other memory layers contain the remaining data and instruction set. This circuit has been tested with odd-even sort algorithms and it has been benchmarked against GPUs and ASIC. This research path is presented in Chapter 5.

MagnetoElastic NML (ME-NML) is a technological improvement of the NML principle, proposed by researchers of Politecnico di Torino, where the clock system is based on the induced stretch of a piezoelectric substrate when a voltage is applied to its boundaries. The main advantage of this solution is that it consumes much less power than the classic clock implementation. This technology has not yet been investigated from an architectural point of view and considering complex circuits. In this research field, a standard methodology for the design of ME-NML circuits has been proposed. It is based on a Standard Cell Library and an enhanced VHDL model. The effectiveness of this methodology has been proved designing a Galois Field Multiplier. Moreover the serial-parallel trade-off in ME-NML has been investigated, designing three different solutions for the Multiply and Accumulate structure. This research path is presented in Chapter 6.

While ME-NML is an extremely interesting technology, it needs to be combined with other faster technologies to have a real competitive system. Signal interfaces between NML and other technologies (mainly CMOS) have been rarely presented in literature. A mixed-technology multiplexer is designed and presented as the basis for a CMOS to NML interface. The reverse interface (from ME-NML to CMOS) is instead based on a sensing circuit for the Faraday effect: a change in the polarization of a magnet induces an electric field that can be used to generate an input signal for a CMOS circuit. This research path is presented in Chapter 7.

The research work presented in this thesis represents a fundamental milestone in the path towards nanotechnologies. The most important achievement is the design and simulation of complex circuits with NML, benchmarking this technology with real application examples. The characterization of a technology considering complex functions is a major step to be performed and that has not yet been addressed in literature for NML. Indeed, only in this way it is possible to intercept in advance any weakness of NanoMagnet Logic that cannot be discovered considering only small circuits. Moreover, the architectural improvements introduced in this thesis, although technology-driven, can be actually applied to any technology. We have demonstrated the advantages that can derive applying them to CMOS circuits. This thesis represents therefore a major step in two directions: the first is the enhancement of NML technology; the second is a general improvement of parallel architectures and the development of the new Logic-In-Memory paradigm.

Contents

Chapter 1

Motivation

This Chapter explains the motivations that guided me through research path described in this Thesis.

Over the past three decades, the constant evolution of electronics has been founded on the ever-smaller device dimensions of silicon-based CMOS technology. CMOS has exponentially improved in both performance and density of integration, producing the transistor trend described by Moore's law. Today, however, the conventional physical scaling is slowing down. As forecasted in the International Technology Roadmap for Semiconductors [1], CMOS scaling is expected to reach its boundaries soon, probably marking the end of CMOS era. Scaling in CMOS cannot continue forever for two main reasons: 1) the minimum achievable dimensions is in any case limited; 2) the off-state leakage power is increasing with device shrinking. We will describe in next Chapter with more details the physical limitations that CMOS is encountering. With these limitations, it seems to be clear that CMOS will not be the leading technology in the next 20 years, because it cannot keep-up with improvements expected by Moore's law.

The negative effects of CMOS scaling have been already addressed from an architectural point of view: parallel architectures have been introduced in CMOS to avoid an excessive increase of clock speed leading to too high power consumption. Therefore, in this framework, where architectural solutions have already been explored, the only chance to keep-up with Moore's law is to address these limitations from a technological point of view. This happened in past decades when technology shifted from BJT to CMOS transistors, and more recently with the introduction of 3D transistors. Today, one of the most important challenges of VLSI and technology researchers is to find a valid alternative that could substitute CMOS in next years.

Several technologies are currently under study. The International Technology Roadmap for Semiconductors (ITRS) [1] summarizes some of them, among which Quantum-dot Cellular Automata (QCA) and its magnetic implementation Nano Magnetic Logic (NML). NML is particularly interesting for its physical features: a magnetic circuit does not have leakage power consumption, and magnets have an intrinsic memory capability. Therefore, NanoMagnet Logic could be an important solution to reduce power consumption (and area occupation), that are the main drivers of today logic circuit design. The most important drawback of magnetic circuits is the low clock frequency achievable (hundreds of MegaHertz). Another limitation of this kind of circuits is that the layout influences the timing characteristics of the circuit. Indirectly, long wires have long latency. A complete description of NML technology is given in Section 2.3.

The approach here followed is to design logic circuits in NML, addressing the limitations that this technology has nowadays from an architectural point of view. We will explain in detail in Chapter 3 that Systolic Arrays can be an ideal architecture for NML technology. Systolic Arrays are parallel architectures made of identical and locally interconnected Processing Elements. In this way they can address the low clock frequency of NML, because the throughput is guaranteed with parallelization of tasks in several Processing Elements. At the same time the short local interconnections allow to reduce the long latency of wires problem.

We start introducing Systolic Arrays for NML and analyzing several alternatives to improve them. Then we consider another kind of architecture called Logic-In-Memory. In this way we push forward the research for NanoMagnet Logic architectures as it has never been done before. Indeed, literature presents only small architectures with few gates designed for NanoMagnet Logic. However, we believe that a correct and complete study of a technology shall be done taking into account real circuits that implement complex functions. This is the only possible way to intercept any limitation of the technology that can appear only when big circuits are considered. In fact, it is not at the device level that the features of a technology can be asserted. It is therefore necessary to analyze how NML behaves in real operating conditions (many cells with complex interconnections and complex algorithms).

NanoMagnet Logic and in general all post-CMOS proposed technologies are still under development. So, the technological solution is not definitive, yet it is already interesting for its preliminary results. For this reason it is worth working on NML in its actual state. It is probable that the technology will evolve with new features and different behaviors: for example considering Domain Wall interconnections, it could be possible to eliminate the long latency of wire problem. Therefore, it is important that the enhancements introduced with this research can be used also with other technologies. Without this assumption the work here presented may be useless in few years if the technology is modified.

In general, an architectural approach allows to refrain from the technology itself. In this way the improvement introduced thinking to the circuit in NML can also be applied to a CMOS circuit (or a circuit implemented in any other nanotechnology). All the improvements that we propose at architectural level are indeed applicable to CMOS and in several cases the effect on the current technology has also been asserted to show the achievable improvement.

The whole research activity is divided into two parts: the first deals with Parallel Architectures for NanoMagnet Logic; the second is focused on MagnetoElastic NML Circuit Design.

The first step in this architectural approach towards NML logic circuits has been the adaption of current existing solutions to overcome main limitations of this technology. Indeed, starting from technological constraints (long delay of wires and low clock frequency) architectures like Systolic Arrays have been identified as an ideal solution. The Systolic Array concept has been enriched with a "Pipeline Interleaving" concept based on input data feeding, that has been analyzed in detail and described with mathematical equations. This approach has been adapted also to CMOS circuits that can benefit from input interleaving. Moreover Systolic Arrays have been made latency-insensitive through a communication protocol among Processing Elements. In this way even if Processing Elements have different delays, the asynchronous communication protocol guarantees the correct circuit behavior. Finally to test this approach a real-case scenario has been identified, based on the Floyd-Steinberg dithering algorithm for image processing. This research path is described in Chapter 3.

The second step has been the definition of a new Systolic Array that overcomes the main limitation of the original one, i.e. the algorithm-dependency. The Array has been made reconfigurable through new interfaces and functions inside each Processing Element. The reconfigurable Systolic Array has been mapped both in CMOS and NML technologies. Moreover, it has been tested with several algorithms to properly show the reconfigurability potentiality. Algorithms belong to real processing applications such as FIR and IIR filters. This research path is described in Chapter 4.

Finally other parallel architectures that exploit the Logic-In-Memory concept have been designed. In these kind of circuits, logic and memory can be mixed in the same device, overcoming the common bottleneck of today's systems in communication between ALU and memory. The Logic-In-Memory array was designed first as a 3-layered circuit (logic - routing - memory) and then as 2-layered circuit with a pipelined memory. Also in this case, a real application has been used to test these circuits: the Odd-Even Sort algorithm. Results show that this circuit can outperform both GPUs and ASIC when big datasets are considered. This research path is described in Chapter 5.

An important phase of the research has been tailored to the new technological solution proposed for NML clock, called MagnetoElastic NML (ME-NML) from our group. ME-NML has a quite different approach at circuital level with respect to classic NML. For this reason in this thesis we have defined: 1) a set of standard cells for these circuits and 2) a VHDL model that can be used to design and simulate ME-NML circuits. In addition, this VHDL model can compute area occupation and power dissipation of the circuit. To evaluate how well ME-NML performs when complex circuits are considered, we have designed a Galois Field Multiplier. Synthesis results are encouraging since ME-NML outperforms CMOS and classic NML in area and power dissipation. Finally we have analyzed the trade-off between Parallel and Serial circuits in ME-NML. While in CMOS the parallel solution is always the best one, in ME-NML this is not always true. For example the Galois Field Multiplier performs extremely well in ME-NML thanks to its partially parallel and partially serial input protocol. For this reason we have investigated in general parallel and serial solutions in ME-NML. We have used Multiply and Accumulate (MAC) as test circuit, thus exploiting this opportunity to introduce new ME-NML circuits never designed before. This research path is described in Chapter 6.

The Reconfigurable Systolic Array mentioned before has great features but also one important drawback: the high number of multiplexers used to reconfigure the array have a big impact on circuit area in NML. It was hence proposed a mixed architecture NML/CMOS for multiplexers, where a simple logic gate made by 3 NML cells is used to work as multiplexer with correct CMOS clock signals. Actually the multiplexer is moved to the CMOS plane instead of being in the NML one. While this approach has been successfully verified for MUX structures, it could be also extended to other gates to have a fully hybrid CMOS/NML circuit. The basis for this kind of circuits were laid identifying the readout circuit for NML and the command circuit for CMOS. This last research path is described in Chapter 7.

Before starting the description of the five research paths and their results, it is worth giving an advice. Throughout the thesis and in general during the research activities, several technological comparisons are presented (among CMOS and NML implementations). Usually this comparison is given in terms of area occupation and power dissipation. While in CMOS it is possible to obtain true and reliable estimations using Synopsys and Encounter, this is not true for NML. The only tools that we have to make a simple estimation of area and power are ToPoliNano for classic NML and a VHDL model developed ad-hoc for ME-NML. These two ways are neither official, nor tested against thousands of circuits as it is done for CMOS. It is then possible that the real numbers vary from those obtained for NML and ME-NML in our synthesis. It is also reasonable that changes in the technology itself may lead to circuits requiring less or more area, less or more power. At this moment it is not possible to state that one technology that outperforms CMOS in our tests will truly behave better in a real environment. Nevertheless, the comparison is still meaningful and important if seen in a different way: With our architectural approach, we want to intercept as soon as possible potential weaknesses of NML technology that cannot be discovered looking only at small circuits made of few cells. With this in mind, it is clear that if a technology has already worse performance than CMOS, it will probably not be the technology of the future. We will see that this is exactly the case of classic NML with magnetic clock. Differently, it is possible to identify if a technology is encouraging on one aspect and instead is performing worse on another aspect: in this way technologists can focus on the weak aspect and try to identify technological solutions in a precise scenario. The final consideration of this advice is that even if numbers are not reliable at 100%, it is still very important to analyze and compare them, to have a clearer idea of the characteristics of the technology when complex circuits are considered. An analysis of this kind is done in NML in this thesis, and similarly it is done on other technologies by other researchers.

Chapter 2

Technological Background

2.1 CMOS scaling

Over the past three decades, the constant evolution of electronics has been founded on the ever-smaller device dimensions of silicon-based CMOS technology. CMOS has exponentially improved in both performance and density of integration, producing the transistor trend explicited in Moore's law. Today, however, the conventional physical scaling is slowing down. As forecasted in the International Technology Roadmap for Semiconductors [1], it is expected to reach its boundaries soon, probably marking the end of CMOS era.

CMOS decay is due to several factors [2], mainly due to physical and material limits. Basically, both electrostatics and tunneling mechanisms cause leakage current increase in ultra-small MOSFETs, till it has become comparable to the drain current. The increased leakage current negates the threshold and supply voltages reduction, denying a speed increase. These are some of the well known effects of down scaling: Drain Induced Barrier Lowering (DIBL), Short Channel Effect (SCE), Punch-Through and subthreshold inversion, mobility degradation, band-to-band tunneling [3][4]. Another challenge involves power consumption and thermal dissipation: Power density has been growing, as the supply voltage did not scale as much as the channel length.

Due to all these factors, keeping up with the Moore's Law will most probably be a challenge that will not be answered by Silicon CMOS nanoelectronics. For this reason many alternative technologies are under study to preserve the same rate of performance improvements. The efforts have been focused toward two main directions [1]:

- Innovation of CMOS materials and structures. Some examples are: SOI (Silicon On Insulator) transistors, with an insulator layer between substrate silicon body, and FinFET, where a multigate structure heavily reduces short channel effects.

- Creation of completely new nanoelectronic devices, called “Beyond CMOS Devices”, able to replace CMOS technology. One of the most promising principle is Quantum-dot Cellular Automata (QCA). Nanotechnologies like QCA offer very high integration density, but they are still in a premature stage: A reliable and functional realization still requires extended study from the device up to the architectural level.

Current transistors exploit electronic charge to store information, therefore switching between logic levels involves charge movement, thus requiring a current flow and a consequent Joule dissipation. Energy losses are then an intrinsic characteristic of charge based electronics and, as explained before, highly scaled transistors will not be able to preserve the charge due to significant leakage. It is clear that charge based devices do not seem to be able to maintain the cost per function improvements of the last decades. The spreading concept is to replace the charge with a new kind of information token such as for instance: Polarization of nanomagnets, change in molecular configuration, electron spin or position of a micro-mechanical object.

2.2 Quantum-Dot Cellular Automata (QCA)

Since the introduction of the Cellular Automata idea in 1993 [5], Quantum-Dot Cellular Automata (QCA) has been attracting an increasing interest. QCA is a valuable candidate for the post-CMOS era, because it effectively addresses the problems of device density and power dissipation.

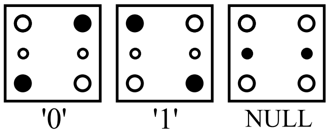


Figure 2.1. Possible states of a QCA cell: Stable states ’0’ and ’1’ and unstable NULL state.

QCA technology is based on a bistable cell; properly organized arrays of these cells can realize logic functions. The first proposed implementation used a square cell with 4 quantum dots in the corners and 2 electronis. Due to electric repulsion, electrons will place at opposites corner in steady state. The two possible configurations represent hence logic values ’0’ and ’1’ [6]. In reality, to allow a correct signal propagation a third unstable state (NULL state) is necessary, therefore two more dots need to be added (Figure 2.1). The explanation of the NULL state is given in paragraphs 2.3.2 and 2.3.3. In Figure 2.1 it is shown the generic base cell, but the theoretical principle of QCA can be realized in different ways, depending on the technology used.

Several technological implementations have been proposed in the literature. Here we present three physical solutions that seem to be more promising.

- Molecular QCA [7]. The fundamental states of the Molecular QCA cell correspond to different charge distributions in a complex molecule. Charge movements can be triggered by electrons reacting with the oxide-reduction center of the molecule. Using molecules, every QCA cell would be identical to the others and would have dimensions in the order of few nanometers, making this structure extremely attracting. Moreover, molecules reactions work perfectly at room temperature and are extremely fast. Indeed, the expected switching speed of this implementation is in the order of THz [8][9][10]. This is the most promising approach, even though a real implementation is unreachable at the moment: Current technology cannot manipulate single molecules as required yet. Another delicate issue is the transduction of electrical signals from and to information understandable by the molecule, as up to now there are no valid solutions to this problem.

- Silicon Atomic QCA [11]. The QCA principle is implemented using atoms as quantum-dots. It has been proved that the dangling bond (DB) state of silicon atoms can be exploited as a quantum dot. Up to now the experimental results are promising and the electrostatic control over the charge within DB assemblies has been verified [12].

- Magnetic QCA or NanoMagnet Logic (NML) [13]. The base cell is a single-domain nanomagnet with dimensions lower than $100\,nm$. Magnets of this size can have only two possible magnetization states, corresponding to '0' and '1' logic values [14]. The key interesting factor of Magnetic QCA is that, thanks to its magnetic nature, it has exceptionally low power consumption and a strong logic-in-memory predisposition [15][16][17].Concerning operating frequency (hundreds of MHz) and dimensions, this implementation is instead less interesting than the Molecular QCA; it is also slower than CMOS systems. Another key advantage for this technology is the physical realization feasibility with current technology, that allows to study and experiment on QCA based architectures on a higher abstraction than the single cell, facing directly design problems common to any QCA implementation.

Magnetic QCA is the addressed technological implementation chosen in this research. This technology is presented with more details in Section 2.3.

To design logic circuits, QCA cells are placed close each other, so that the electrostatic interaction between cells allow signal to propagate. This is further analyzed for the case of NML in paragraph 2.3.1. However the electrostatic interaction cannot generate an infinite signal propagation. For this reason it is necessary an external mean as explained in the following paragraph.

2.2.1 Signal propagation and Clock

Generally the electrostatic interaction between QCA cells is not strong enough for a signal to propagate through a number of cells (a wire). The switching of a cell requires as much energy as the barrier between its two stable states, that is the energy keeping electrons trapped in the dots.This value is generally high enough to not allow autonomous data propagation. For this reason there is the need for an external mean able to control the signal propagation by acting on the energy barrier between the two stable states. Such barrier can be lowered by applying an electric field, that will force electrons into the central dots leaving the cell in an unstable state, which is referred to as NULL state. Once removed the external field the cell will stabilize either at '0' or '1', depending on the state of neighbor cells.

So the main idea is that if we want that a cell to assumes the same value as its neighbor, we force such cell in an unstable state through an external electric field, and then we simply release the field. This control field is called *clock*. In principle this technique could work with an infinite number of cascaded cells, but practically this number is limited, otherwise there will be propagation errors mainly due to thermal noise [18]. Therefore a spatial flow control system is mandatory.

From the remarks above it is clear that a signal cannot pass through a whole circuit at once because the cells pattern would be too long. The solution is to divide the circuit in small sections and let signals go over one section at a time, in a pipelined manner. So circuits are partitioned in small areas, where each area counts a limited number of cascaded cells; these areas will be called *clock zones*.

This concept is further extended, presenting the actual implementation of a clock mechanism for NML, in paragraphs 2.3.2 and 2.3.3.

2.3 NanoMagnet Logic

In this Section we describe NanoMagnet Logic technology. Some key concepts are given in the following. Then the fundamental elements of an NML circuits are shown in paragraph 2.3.1. Two possible clock solutions are presented: Magnetic Clock in paragraph 2.3.2, and MagnetoElastic Clock in paragraph 2.3.3. Finally we focus on the Intrinsic Pipelined nature of NML in paragraph 2.3.4, as this is one of the most relevant aspects of this technology that we want to address in this research.

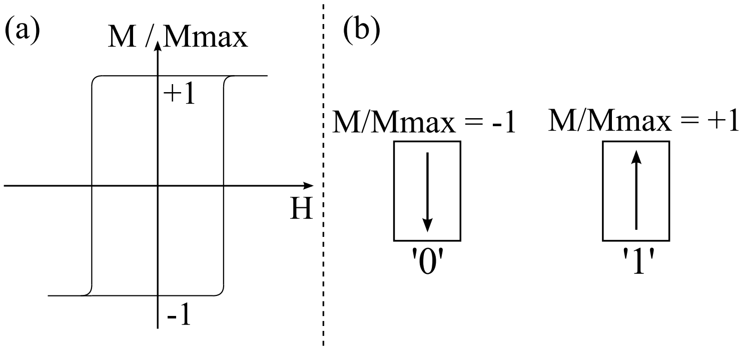


Figure 2.2. (a) Hysteresis cycle of a single domain magnetic signal. (b) The two stable states of the NML base cell.

Magnetic Quantum dot Cellular Automata (MQCA), also called NanoMagnetic Logic (NML), is an innovative technology based on the idea of using magnets to implement logic functions. The result are digital circuits with intrinsic memory capability [19].

The base element of NML is a small bistable magnetic cell. Since it is not a permanent magnet, its magnetization can be influenced by external means. Therefore nanomagnets placed side by side will arrange themselves in an antiferromagnetic manner, because of the attraction between opposite poles.

The nanomagnets dimensions must be between $50\,nm$ and $100\,nm$. The upper limit assures that the magnets only have one magnetic domain, which means that the magnetization does not vary across the magnet and the hysteresis cycle is coherent with the one shown in Figure 2.2(a). The two saturation values $M = +1$ and $M = -1$ are the only stable states, therefore they are associated to logic values *'0'* and *'1'*. The lower bound of $50\,nm$ is, instead, crucial to avoid the superparamagnetic effect, which would cause the magnetization to vary together with thermal fluctuations. To assure thermal stability the energy barrier between the two stable states must be at least $30k_BT$. As from Figure 2.2(b) the two states have magnetizations in opposite directions, so they both lie on the same axis. In steady state conditions, if one side of the magnet is longer than the other, thanks to shape anisotropy, the magnetization will be forced along the longer axis (easy axis). Therefore it is important that in NML the ratio between the magnets dimensions (aspect ratio) is at least 1.2.

In similar way to general QCA, when we put several magnets in chain to make a logic circuit, if the inputs switch, only few cascaded magnets will flip accordingly. This is due to the fact that the energy produced by one magnet switching is not enough to align all other magnets. Indeed only few magnets can switch autonomously, while after them the polarization of successive magnets may not change or become unpredictable [15]. As explained for general QCA, the solution to this problem is a clocking mechanism explained in paragraphs 2.3.2 and 2.3.3.

There are several reasons that make the NML study worthy, even if the working frequency is limited:

- NML is the only QCA implementation that works at room temperature and it can be fabricated with current technology [20].
- Magnets do not dissipate static power and a single magnet switching absorbs around $30k_BT$. Therefore NML potentially has an extremely low power consumption.
- Since the difference between QCA and CMOS technologies is bottomless, to fully comprehend the potential of QCA, it is mandatory to investigate complex architectures, also considering all the working and fabrication constrains. Fortunately most of the architectural study on NML could probably be applied to other implementations like the molecular QCA, which seems far more promising than Magnetic QCA but it is still not supported by current technology.

The main advantage of Magnetic QCA is to be realizable with current technology (electron beam lithography or high end optical lithography) together with its ability to operate at room temperature. The fabrication feasibility was first proven by researchers of the University of Notre Dame in Indiana (US). They built horizontal wires, vertical wires and majority gates [21]. A Magnetic QCA horizontal wire was also created by researchers at Politecnico di Torino.

2.3.1 Logic Gates

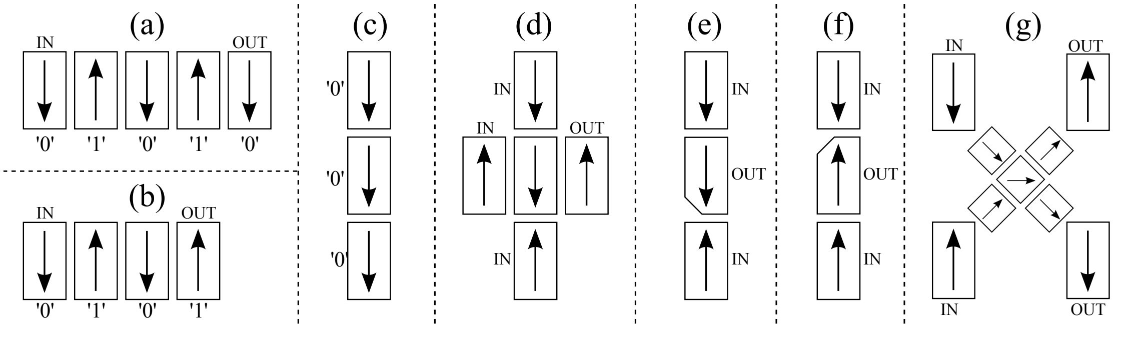


Figure 2.3. (a) Horizontal wire. (b) Inverter. (c) Vertical wire. (d) Majority Voter. (e) AND port. (f) OR port. (g) Crosswire.

Figure 2.3 shows a complete set of logic blocks for NML circuits. Gates rely on coupling between magnets. In particular it is important to notice horizontal coupling: Horizontally magnets align themselves antiferromagnetically, where each magnet has inverted polarization with respect to the neighbors. So the inverter can be simplified to a simple horizontal wire with an even number of magnets as in Figure 2.3(a). On the other hand an odd number of adjacent magnets would result in a buffer function, that is a simple wire (Figure 2.3(b)). Vertically the coupling is ferromagnetic, so no inversion is possible (Figure 2.3(c)). The majority voter, depicted in Figure 2.3(d), is pretty much the same as for general QCA.

Moreover, it is possible to obtain specific logic gates modifying the shape of a magnet: By making magnets with slanted edges it is possible to create *AND* and *OR* logic functions [22]. QCA would generally need a three inputs majority gate to obtain AND and OR logic ports, while only two inputs are needed for non-majority based gates, considerably optimizing area occupation and layout entanglements. The different-shaped magnets acquire a preferential state, which they will leave only when both inputs, from above and below, are up or down, implementing as a consequence an AND or OR logic function (Figure 2.3(e), Figure 2.3(f)). Another important advantage of this solution is that existing synthesizers with a dedicated library cell including AND and OR can already be used for the first stages in the design of an NML circuit.

Since NML is a planar technology at the time of writing, a crosswire gate is necessary. A possible implementation is the one represented in Figure 2.3(g), the crossing is made of five square cells ($50\,nm - 100\,nm$ of edge) that have four stable states instead of two. In this way they can let pass through two signals simultaneously.

2.3.2 Magnetic Clock NML

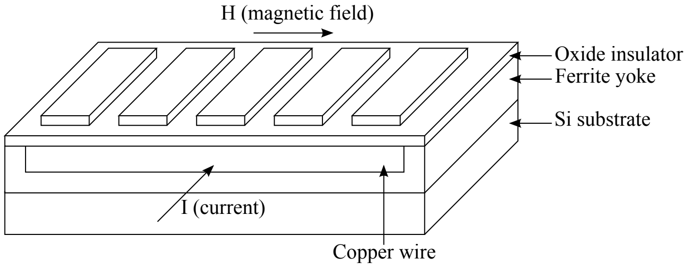


Figure 2.4. NML with Magnetic Clock mechanism. The magnetic field H is generated by the current I flowing through the copper wire, which is placed under the magnets plane.

One solution for controlling the nanomagnets magnetization in NML circuits is the Magnetic clock, as proposed in [13] and verified experimentally in [20]. The magnetic field is generated by a current flowing through a wire positioned under the magnets plane (Figure 2.4). The material for the wire is copper, buried in a ferrite yoke envelope for field confinement. The wire's thickness must be enough to generate a magnetic field able to force cells to the intermediate state (NULL state) [23].

As explained in Section 2.2.1 a clock system is required. This is normally done with a multiphase signal, with each phase assigned to a clock zone. The classic scheme has 4 phases, but also a 3-phase clock is feasible [24][25][26]. The Magnetic NML normally exploits a 3 phase clock system.. Figure 2.5 shows the functioning of the 3-phase clock of a horizontal wire over time (vertical axis).

Each clock zone undergoes three phases in the following temporal sequence: RESET, SWITCH and HOLD. The RESET ($clock = 1$) erases the information, leading cells to an intermediate state. In the SWITCH phase the clock goes to zero, so cells can assume a magnetic orientation. The orientation is influenced by the nearby cells being in HOLD state, as cells in the RESET state cannot affect the neighbors. When a group of cells, in the same clock zone, is in the HOLD phase, they have a stable magnetization.

To assure a correct signal propagation the RESET phase applied to different zones must overlap in time as in Figure 2.5(b), where the RESET state lasts slightly more than $2\pi/3$. The reason lies in the fact that when a zone is in the SWITCH phase, the two neighbor zones must be respectively in HOLD and RESET phase. However if the field of the SWITCH zone is removed and the field is applied to the RESET zone at the same time, a back propagation phenomenon could take place. Initially, when the field is removed from the SWITCH zone, the RESET zone would still be in the HOLD state, as magnets need a finite time to switch from a stable polarization to the intermediate state. In Figure 2.5(a) we can see how the value in *Time step 1* on the left is propagated step by step to magnets in the clock zone on the right.

2.3.2.1 Snake Clock Layout

The generic QCA is based on a 4-phase clock system, however it is also possible to use a 3-phase clock [13], given that the signals are overlapped. The clock network for Magnetic NML is a 3-phase overlapped system, called Snake-clock; its 3D structure and top layout are depicted respectively in Figure 2.6(a) and Figure 2.6(b).

The Snake-clock is a phase arrangement able to manage both left to right and right to left signal propagation. It is based on clock wires where current flows generating an induced magnetic field that will force magnets in the reset state.

The clock wires are basically simple metal wires parallel to the magnets plane, two positioned above and one below [24]. Two thin oxide layers provide separation between clock wires and nano-magnets. One clock wire is straight (number 1), while the other two have a complementary zig-zag shape. They are like twisted wires, but

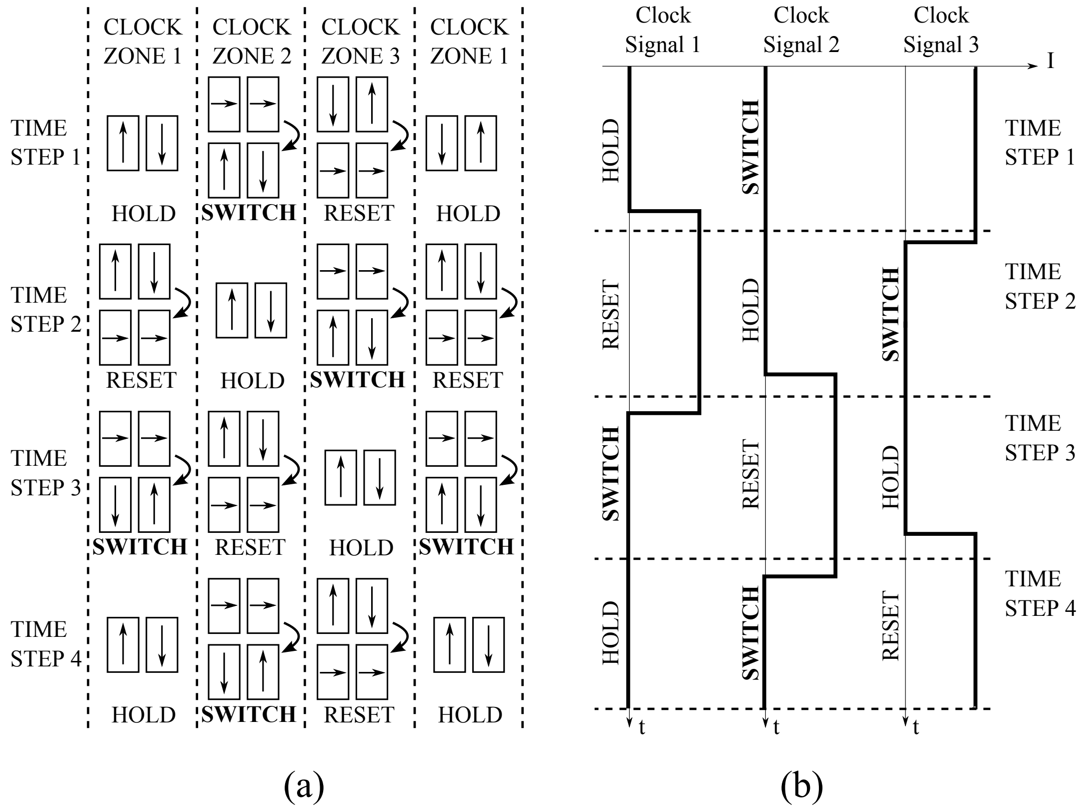


Figure 2.5. The clock phase sequence is RESET, SWITCH, HOLD. (a) Functioning in space (horizontally) and time (vertically) of a horizontal NML wire. (b) The 3 clock signals. They are applied to different zones in space and they are repeated over time. They are the same in magnitude but with a 120° phase shift. [27]

they do not display any interference, as they are on different planes. In the case in Figure 2.6(a) the wires 1 and 2 are routed on the same plane, while the clock 3 is on the other one.

Considering the top view in Figure 2.6(b), it is straightforward to understand that magnets cannot be placed on areas corresponding to the wires twisting, as they would be affected by both clock wires 2 and 3.

Figure 2.7 shows a very simple circuit based on the Snake-Clock system. The direction of the information flow is highlighted by arrows. Signals propagate through clock zones in the order 1, 2, 3 and so on. The clock wires twisting divides the circuit area in horizontal stripes with alternate propagation directions. Furthermore, as required by this clock mechanism, there are no magnets placed over the twisting areas. The magnets with a slanted edge required for the AND logic function are highlighted in black.

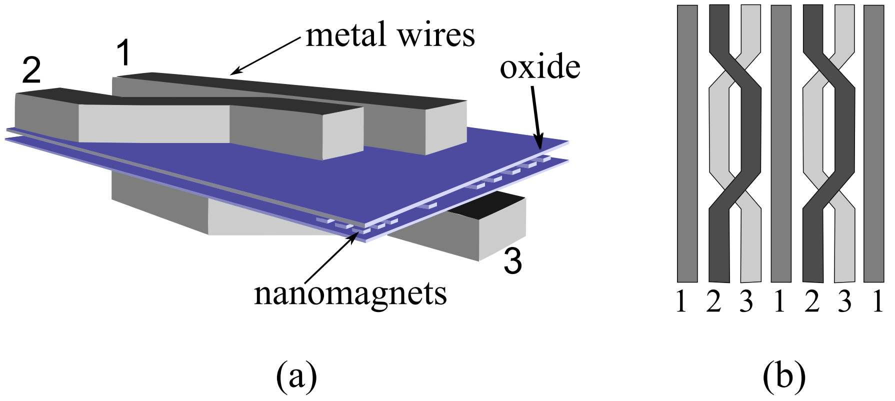


Figure 2.6. Snake-clock. (a) 3-D layout. (b) Top 2-D layout. The nanomagnets are placed between the two planes. Magnets cannot be placed in the zone where two phases twist each other.

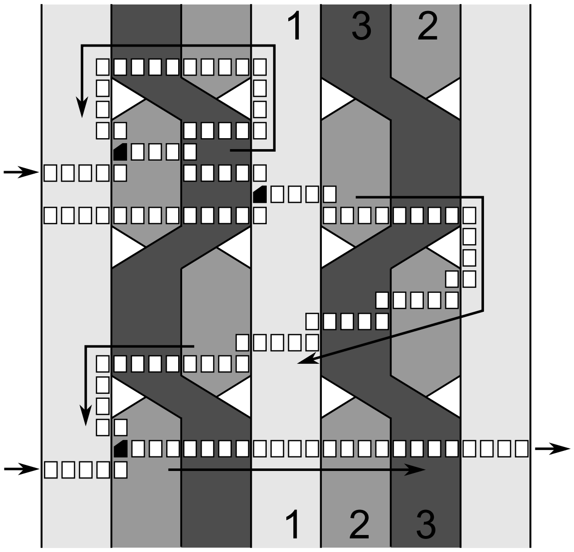


Figure 2.7. Example of a simple circuit based on the *Snake-Clock* system. Different background colours refer to different clock zones. The arrows show the signal flow direction. [27]

2.3.2.2 Working frequency

The main limitation of NML technology is the maximum working frequency, which is intrinsically bounded. To obtain the highest possible clock frequency the clock zone width should be equal to that of a single magnet. However the usual width is sufficient to contain 3-5 magntes [18] because of several factors: fabrication limitations, thermal noise, latency, throughput. The more are the consecutive magnets in a clock zone, the lower will be the clock frequency. The constraints on the clock frequency are mainly related to the clock mechanism chosen and the fall and rise time of the adiabatic switching of clock signals, mandatory to reduce power consumption. Less critical is instead the bound derived from the switching time of nanomagnets from the intermediate (NULL) state to a stable one and viceversa. The NML circuit speed is expected to be of the order of $100MHz$ [28][29][30].

In the beyond-CMOS scenario, NML technology is a good solution but it cannot aim to completely substitute CMOS. Despite the clear benefits for what concern occupied area, power consumption and memory ability, NML's clock frequency cannot compete with existing solutions like CMOS. We will address this clock limitation at an architectural scale introducing architectures able to increase the throughput. Nevertheless, it is clear that a NML system cannot live on its own. Rather, it should be thought as part of a multi-technology environment in which NML is used to perform power consuming operations where a long operational time is acceptable.

2.3.3 Magnetoelastic Clock NML (ME-NML)

Recently a valuable alternative to the Magnetic Clock NML has been proposed and studied by Politecnico di Torino researchers: the MagnetoElastic Clock NML (ME-NML) [15][31].

In previous paragraph 2.3.2 the proposed external mean, responsible for the magnets switching, was the Magnetic Clock with a Snake-clock layout. The idea was to position clock wires below or above the magnets plane. A current flowing through the wires would generate a magnetic field able to control the cells magnetization. The generated field is then along the magnets' short side of the magnets, forcing cells in an intermediate unstable state.

The interest in Magnetic QCA is mainly due to the very low power consumption, several times lower than the latest CMOS transistors. While this is true for the magnets switching, unfortunately the clock generation system based on magnetic clock is not able to guarantee this low power performance. Indeed, it is based on $1\mu m$ copper wires with a required current of $545mA$ [15]. Therefore, due to Joule losses, the power dissipation of the clocking system is very high, nullifying the advantage of a low-power magnets switching.

To solve this problem an alternative solution has been recently proposed [15][31], based on the Magnetoelastic effect. Using a piezoelectric material, applying a voltage to its boundaries, a mechanical stress will derive in the piezoelectric. With this stress, magnetic cells above can be forced into the RESET state. The magnetic cells ($10nm$ thick) are coupled with a $40nm$ thick PZT layer (Figure 2.8(a)). To maximize the mechanical coupling, magnets are deposited directly onto the piezoelectric

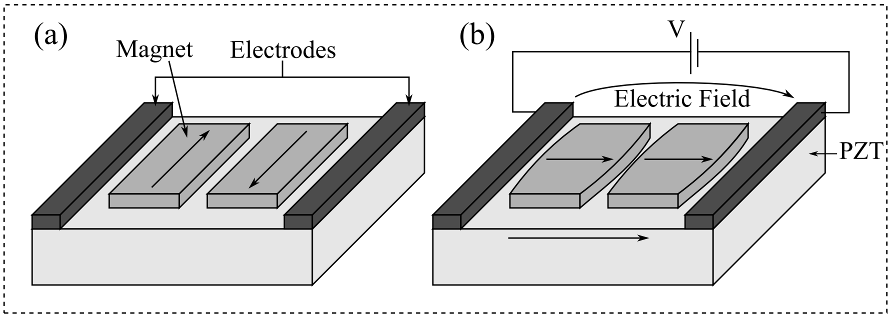


Figure 2.8. Magnetoelastic NML clocking mechanism. (a) No voltage applied. (b) Voltage applied to the electrodes. The PZT substrate induces a strain on the nanomagnets forcing their magnetization to their intermediate state.

material. For a proper strain transfer, the PZT substrate has to be much thicker than the magnets. The magnetic material is then controlled by applying a voltage (few mW) to the piezoelectric. When the voltage is applied, the strain induced by the piezoelectric material, forces the magnetization of the magnets layer to the intermediate position, parallel to the short edges (see Figure 2.8(b)).

The electrodes are deposited on top of the PZT, while the wires that drive the electrodes can be placed in additional layers, just as for CMOS. This makes this NML implementation compatible with CMOS fabrication.

This approach comes from a previous idea based on multiferroic structures instead of simple magnets [30][32]. The performances of the pure multiferroic structure are better, but there are two major fabrication problems. The aspect ratio is critical, there are only $2nm$ of difference between the length of the two cell's sides. Such a low resolution is hardly achieved with the Electron Beam Lithography. Moreover the electrodes should be only a few nanometers thick, a request that does not comply with the current technology. A pair of them is necessary for every element, to apply the required voltage. The advantage of the solution with the simple magnets is the feasibility with current fabrication techniques. Even if its performances are slightly worse than the multiferroic solution, they are anyway remarkably better than the previous NML solutions.

Since the clock system exploits a voltage instead of a current, the power consumption is extremely low, meeting the unmatched expectations for the initial Magnetic QCA concept. In [15], after a detailed analysis, the selected magnetic material is Terfenol, an alloy of Terbium, Disprosium and Iron. The choice is mainly based on three parameters:

- maximum stress that can be applied to avoid permanent damage on the magnets;

- maximum value of electric field that can be tolerated by the piezoelectric material, since it is an insulator;
- minimum stress to force magnets in the RESET state;
- assure shape anisotropy equal of at least $30K_bT \approx 1.24 \cdot 10^{-19}J$, to have negligible effects of the thermal noise on the magnets stability;
- minimum aspect ratio for fabrication feasibility;
- tolerance to process variation of ±20%, remaining within the working range.

2.3.3.1 Circuit Layout

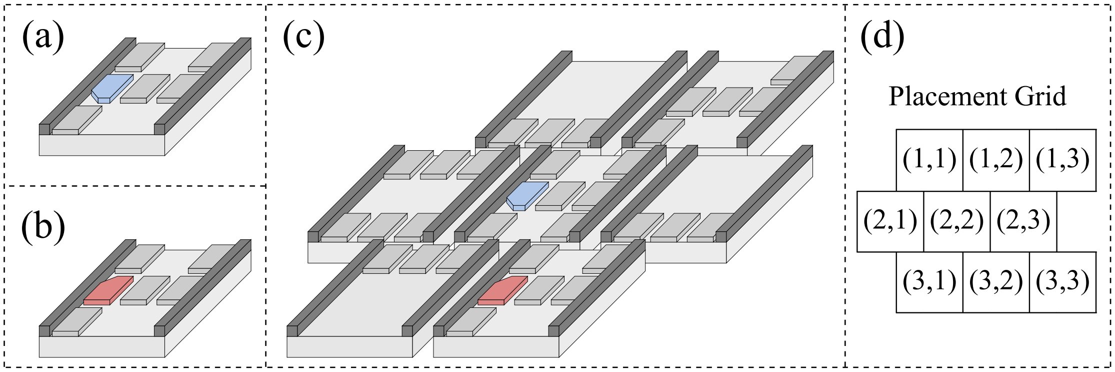


Figure 2.9. Clock zones of the ME-NML. (a) Clock zone with AND logic function. (b) Clock zone with OR logic function. (c) Circuit layout example. (d) Placement grid for ME-NML Cells.

Starting from the structure described in paragraph 2.3.3, MagnetoElastic clock NML (ME-NML) circuits are composed by mechanically isolated islands, like the one in Figure 2.9. Each island corresponds to a clock zone and it is driven by one of the clock signals, applied as a voltage on the Platinum electrodes. Notice that the electrodes position on top of the PZT is compatible with CMOS fabrication and leads to a uniform electric field distribution on the magnets plane.

The presence of the electrodes makes the clock zones communication on those sides impossible. The signal propagation among cells is allowed only through the top and bottom sides. For this reason the Majority Voter port cannot be constructed. Therefore the basic logic gates exploited are inverter, AND (Figure 2.9(a)) and OR (Figure 2.9(b)) [22], so that any logic circuit can be implemented.

Figure 2.9(c) shows how to put together the clock zones to create a circuit. As already said, the communication among cells can take place only through the top and bottom corners, because of the electrodes. For this reason the cells in a row are shifted with respect to the adjacent ones, to assure a correct signal propagation. In fact the cells are placed on a grid as in Figure 2.9(d), where the coefficients identify row and column of the cell's positioning within the circuit.

In the example of Figure 2.9(c) the clock zones have both height and width equal to three nanomagnets. Thermal noise [18] and fabrication constraints allow cells dimensions to vary only between 3 and 5 nanomagnets. Small dimensions lead to smaller electrodes and cells, requiring then a very high resolution fabrication process. The minimum size feasible with current technology is 3. Bigger dimensions will relax the technology constraints, but will increase the error probability due to thermal noise and decrease the maximum circuit speed. If two many cascaded magnets are present in a clock zone, the signal propagation will be error prone. In this thesis the 3 magnets cells are used.

The size of the electrodes varies according to the clock zones dimensions. They are $30 - 40nm$ for the three magnets cells, while $70 - 100nm$ for the five magnets case. This kind of electrodes are already available for CMOS technology.

2.3.4 Intrinsic Pipeline

In a N-phase clock system, signals need a clock period to propagate through N clock zones. As a consequence the delay of a signal depends on how many clock zones it has to cross. This is quite different from CMOS where wires with different lengths have very similar delays. Each clock zone crossed by a signal can be modeled as a register, therefore it is easy to understand that NML circuits (just like QCA) are intrinsically pipelined. Every group of N adjacent clock zones has an overall delay of a clock cycle.

For this reason signal synchronization is a very delicate issue in NML circuits. In Figure 2.10 we present an example useful to clarify the difficulties in routing signals in NML. The input wires routing is correct in Figure 2.10(b), while incorrect in Figure 2.10(a). For a proper circuit functioning the three input signals must reach the two AND ports simultaneously,. To do so, the routing must assure that the input wires cross the same amount of clock zones. The example is presented for the Magnetic NML case, but the same concept applies to ME-NML as well as any QCA implementation.

This is normally called the "Timing = Layout" problem, because the layout of the circuit directly influences its timing property.

The problem gets more complex when dealing with feedback signals, see for example the feedback in Figure 2.7 at the top left corner. While the external input of the AND port arrives at every clock cycle, the second one (the feedback) arrives

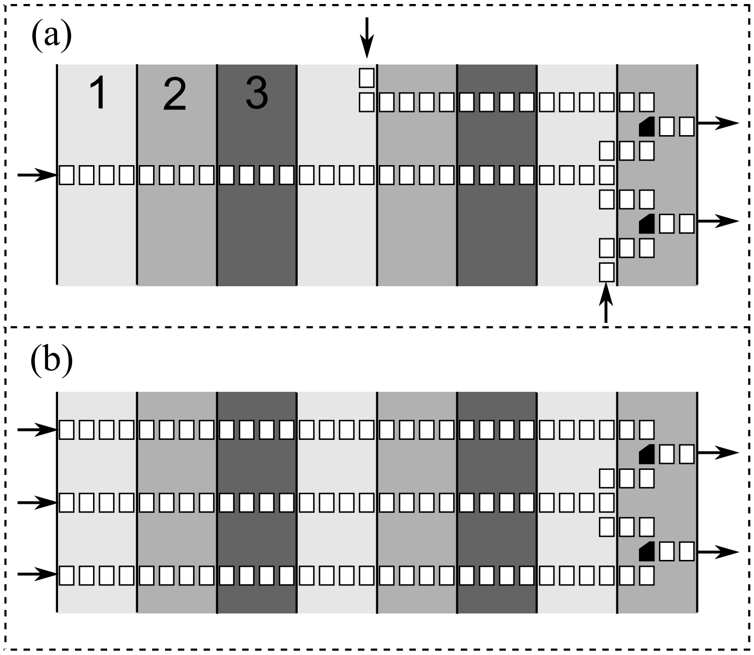


Figure 2.10. NML signal synchronization. The three inputs must arrive to the two AND ports simultaneously. To do so the input wires must pass through the same number of clock zones. (a) Not working routing. (b) Correct routing.

later. The output of the AND port needs two clock cycles to be fed back. Therefore at every clock cycle the AND operation is performed between the new input and the output result obtained 2 cycles before. The proper result will arrive at the next time step. Notice that the longer the feedback wire, the longer the delay. The management of long feedback loops is common in NML, and for this reason we have decided to approach it with a systematic analysis presented in Chapter 3.

2.3.5 Summary on NanoMagnet Logic

In this paragraph the main advantages and drawbacks of NanoMagnet Logic are mentioned. The objective is twofold: on the one hand we highlight the reasons that made this technology so attractive for real implementation; on the other hand we mention all the drawbacks, that led us to the study reported in this thesis to address them from an architectural point of view.

Main advantages of NanoMagnet Logic are:

- No power dissipation in steady state conditions: as mentioned before, magnets that are not affected by other near magnets or clock polarization change will maintain their polarization and will not consume any energy. This has two important derivative advantages, that are the two following.

- Intrinsic memory capability: when a magnet is not driven by a reset clock and not affected by other external higher magnetic fields, it will keep its polarization and thus maintain the logic value stored.
- Low power consumption: this is mainly due to MagnetoElastic NML technology, that has the advantage of an efficient clock generation system. Moreover energy required by magnets to switch in normal conditions is very low.
- Small area occupation: this technology allows to compact logic circuits in few magnets, guaranteeing a small area occupation.

Main drawbacks are instead:

- Low clock frequency: with respect to CMOS, NML circuits can run at much lower operating frequencies (about 100 MHz). This makes the comparison between these two technologies, based only on performance, unsatisfying.
- Long wires have long latency: this is the main effect of Intrinsic Pipeline effect and “Timing = Layout” problem. A long wire may require hundreds of clock cycles to be traveled by a logic signal. This would mean a reduction in throughput and hence generally in performance.

In the rest of this thesis, NanoMagnet Logic will be the principal objective of the study. We will show the advantages of this technology through comparisons with CMOS mainly. The drawbacks will be instead addressed from an architectural point of view and we will demonstrate that it is possible to overcome them and reduce at the minimum their impact.

Nevertheless, it is normal to think of future systems made by components produced with different technologies, where each of them can exploit its main features. In this scenario it is clear that NML will not be used for timing-constrained operations, because it has a low clock frequency. Instead for recursive operations where the main constraints are on power dissipation, NML circuits can be an ideal choice.

Part I

Parallel Architectures for NanoMagnet Logic

Chapter 3

Systolic Arrays Optimization

In this Chapter the first step of the architectural analysis for NML logic circuits is presented. This deals with “Systolic Arrays”, a particular class of architectures that is interesting for the regularity of the layout and the short interconnections. Moreover, Systolic Arrays are parallel architectures that can execute several tasks at the same time to increase the throughput. While this architecture is particularly interesting in its classical aspect, as presented in Section 3.1, it is possible to introduce several improvements to Systolic Arrays, in particular for the NanoMagnet Logic implementation.

Data interleaving can be applied to Systolic Arrays, with proficient results as described in Section 3.2; Latency Insensitive circuits can be embedded in Systolic Arrays to allow asynchronous communication between blocks with different delays, as presented in Section 3.3. The results of a Latency Insensitive circuit can be analyzed using as case study the Floyd-Steinberg algorithm; for this reason we have designed a Systolic Array that is able to execute this algorithm. The final results of this research activity on the enhancement of Systolic Arrays are presented in Section 3.5.

3.1 Introduction to Systolic Arrays

In this Section we will introduce Systolic Array (SA) architecture, dealing with the concept of this architecture, the most important field of applications and its advantages and disadvantages.

Systolic Arras (SA) concept was introduced for the first time by Kung and Leiserson in 1978. In [33] they stated: “a systolic system is a network of processors which rhythmically compute and pass data through the system”. In their common representation, Systolic Arrays are composed of Processing Elements (PEs) locally interconnected. Each PE receives data from neighbor cells and it provides results to other close PEs. PEs at the boundaries of the array are used for input/output data exchange.

An example of Systolic Array, used for matrix multiplication, is shown in Figure 3.1. In this case top and left boundary cells are used to retrieve input data, while results are stored inside each PE. Through a final downloading phase, results can then be obtained from bottom cells. This example has been also proposed by Kung in [33].

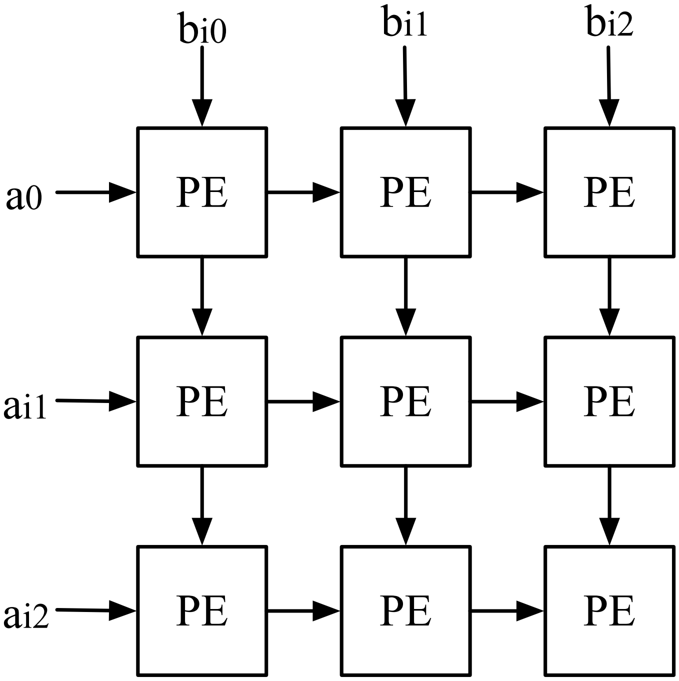


Figure 3.1. An example of square Systolic Array. This can be used for matrix multiplication.

There are two main concepts at the basis of SAs:

1. Parallel Computation: all PEs work in the same way and at the same time on different data;

2. Local Transmission: data are only transmitted between near PEs, so there are not global signals.

For their parallel nature, SAs can be used in signal processing [34][35][36]. They can also be used to execute algorithms for video processing (such as those for MPEG compression). For example, in [37] and [38] a SA for logarithmic search motion estimation is presented; it exploits a bi-dimensional systolic architecture and *pipeline* *interleaving* to achieve a 256x improvement with respect to the classical linear array implementation. The usage of pipeline interleaving to increase the performance of a Systolic Array, especially in the NanoMagnet Logic implementation, is one of the key research points of this path, described in 3.2. While till now interleaving in Systolic Arrays has been proposed randomly depending on the algorithm, we propose a rigorous approach to this technique that allows to adopt Data Interleaving quite straightforwardly.

SAs have been exploited also for image processing [39][40], exploiting the parallelism between Processing Elements and image pixels. Each PE will be in charge one or more pixels depending on the size of the image and output data granularity. We present in Section 3.4 the Floyd-Steinberg image processing algorithm, that is used as case study to design a Latency Insensitive Systolic Array with octagonal cells.

Finally Systolic Arrays have been used for biological sequence comparison [41][42] [43]; the reader can refer to [44] to have an overview of the different hardware solutions for biosequence analysis; this comparison shows that the best performance can be achieved adopting SAs, with a focus on nanotechnologies.

Although Systolic Arrays are now used in several fields, their history has not been very successful. After their introduction in 1978, they did not emerge as a relevant architecture solution till recent years. The reason is quite simple: Throughout the years, technological scaling has allowed to increase operating frequencies and this has been enough to produce the required performance improvement; therefore, pure transliterations of algorithms to hardware modules have been sufficient to achieve desired results thanks to the technology improvement. In recent years the technological scaling has slowed down [45][46] and so parallel architectures have been adopted to guarantee a continuous increase in computing performance. Among them, Systolic Arrays have gained interest because they can combine the high computational capacity, given by parallelism of processing elements, with the short interconnections propriety that makes them one of the ideal architectures for nanotechnologies based on the Quantum-dot Cellular Automata principle.

3.1.1 Systolic Arrays for NanoMagnet Logic

In Section 2.3 we have identified the strengths and limitations of NanoMagnet Logic. We anticipated that if we want to exploit the advantages of low power consumption and small area occupation, we need to address the limitations from an architectural point of view. The main limitations identified were:

1. Low Clock Frequency (100 MHz), due to technological constraints;

2. “Timing = Layout” problem: long wires have long latency.

To address these limitations it is possible to: 1) increase throughput of the system using parallelization, thus reducing the impact of the low clock frequency; 2) use regular structures that have only short and regular interconnections, thus reducing the impact of the "Timing = Layout" problem.

If we consider these two constraints, it is evident that Systolic Arrays represent one ideal architecture for NanoMagnet Logic. Nevertheless, there are still improvements that can be applied to Systolic Arrays for NanoMagnet Logic. We have investigated these improvements and next Sections present the results of our study. Moreover, while these improvements (for example Data interleaving) have been thought to further enhance Systolic Arrays for NML, the same approach can be used in CMOS to have similar benefits. For this reason we will present results that comprise both NML and CMOS.

3.2 Data Interleaving in Systolic Arrays

The first step in the improvement of Systolic Arrays has been a rigorous definition of the interleaving technique applied to this kind of architectures, that is thoroughly described in this Section.

In this Section we first provide an introduction to the Interleaving technique (paragraph 3.2.1). Then a taxonomy of SAs based on the structure of the cell is described in paragraph 3.2.2. Starting from this taxonomy, then for each case a full analysis of the number of interleaved operations that can be executed and architectural solutions to enhance this feature are provided (paragraph 3.2.3 and 3.2.4). Results are shown to demonstrate that *pipeline interleaving* applied to SAs can guarantee better performance in terms of Giga Cell Updates Per Second (paragraph 3.2.5. Results are provided both in CMOS and other nanotechnologies; for the latter, as we have detailed before, the adoption of interleaving is quite obliged, unless under-using drastically the SA. We detail the difference in the approach to introduce data interleaving in Systolic Arrays in paragraph 3.2.6.

3.2.1 Interleaving Technique

Interleaving is in general a way to arrange data in a non-contiguous way to increase performance of a computing system.

Consider the circuit shown in Figure 3.2: this is an accumulator based on an adder and a feedback loop with three registers (represented with squares). Figure 3.2(a) shows the execution of $A + B + C$. Three clock cycles of delay are necessary between one input and the successive one, which means that one addition can be executed every three clock cycles. This is due to the data dependency present between each addition and the successive one, because each addition can be

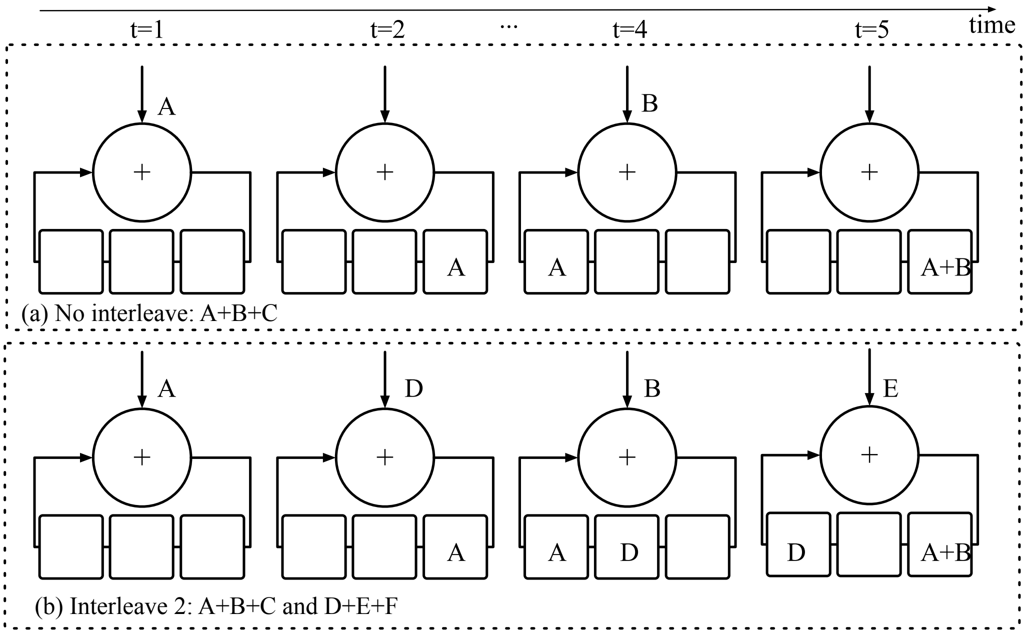


Figure 3.2. Interleave example: an accumulator that contains in its internal feedback three registers. (a) the circuit without exploiting interleave requires to provide one input every 3 cycles, thus having an addition every 3 cycles. (b) the circuit with interleave 2: input values are interleaved and in this way the throughput can be simply doubled because it is possible to execute second addition $D + E$ immediately after $A + B$.

executed only when the previous one has been completed and the result is ready at the input of the adder. Figure 3.2(b) shows the same circuit applying interleaving: in this case it is required to have at least another operation to execute, for example $D + E + F$; *interleaving* consists in providing inputs in a non-continuous manner, i.e. A, D, B, E, C, F in the example, and in exploiting the delay of the loop to store results of different operations. This can be applied in general to any loop-based circuit.

Data interleaving can be applied only when there is no data-dependency between successive steps; $D + E$ for example can be executed immediately after $A + B$. It could be possible to interleave also another operation to maximize the throughput, since the pipeline cue is three registers long. In the example, there is no architectural reason to design an adder with 3 registers in the loop; however, it is possible to think that through a retiming procedure, these registers are inserted in the middle of the adder to reduce its critical path and increase the operating frequency, in CMOS. This would still mean that three clock cycles are required to execute addition and feedback of the signal and for this reason it would be possible to apply pipeline interleaving as explained. This can clearly apply to any CMOS circuit to reduce critical path. If we think to NML instead, the three clock cycles may be requested by a longer wire, due to "Timing = Layout" problem.

The benefits of interleaving have been analyzed in literature. In the case of digital filters [47][48][49], for example, internal feedbacks negate the most obvious ways of improving performance, that is pipelining. Indeed, while in nonrecursive systems it is possible to place latches across any *feed-forward cutset* without changing the transfer function (increasing latency of course, but reducing the critical path) and achieve the desired level of pipelining, recursive systems cannot be pipelined at an arbitrary level by simply inserting latches. This problem can be solved by changing the internal structure of the algorithm to create additional logic delay operators inside the recursive loop, which can then be used for pipelining.

3.2.2 Proposed SA Taxonomy

It is possible to divide Systolic Arrays into two main classes: those With cells that have an Internal Loop (herein WIL), and those WithOut Internal Loop (herein WOIL): the former can be further split in systolic arrays that Store results in the cells (WIL-S) and systolic arrays where the partial result is Passed Through the cells to obtain the final value (WIL-PT). Each of these classes is analyzed hereinafter.

3.2.2.1 WOIL Systolic Arrays

WOIL SAs are those in which each cell does not have an internal loop. It is important to highlight that, since there is no loop inside cells, the PE can be pipelined and interleaving is not required (meaning that it is possible to input data in the same order of the original case, but latency is increased according to the pipeline depth).

It is possible to define in a general way a WOIL Processing Element, as shown in Figure 3.3.

The PE is made of j different blocks, each requiring d_n clock cycles to be completed, $n = 1, 2, \ldots, j$. It is assumed that these blocks cannot be internally pipelined. Some of them (for example from block $i + 1$ to j) are along the path that connects the cells each other (called P), and their total delay is D:

$$D = \sum_{n=i+1}^{j} d_n \tag{3.1}$$

while the others (from 1 to i) work on input data and on stored ones, and are not interested by partial results.

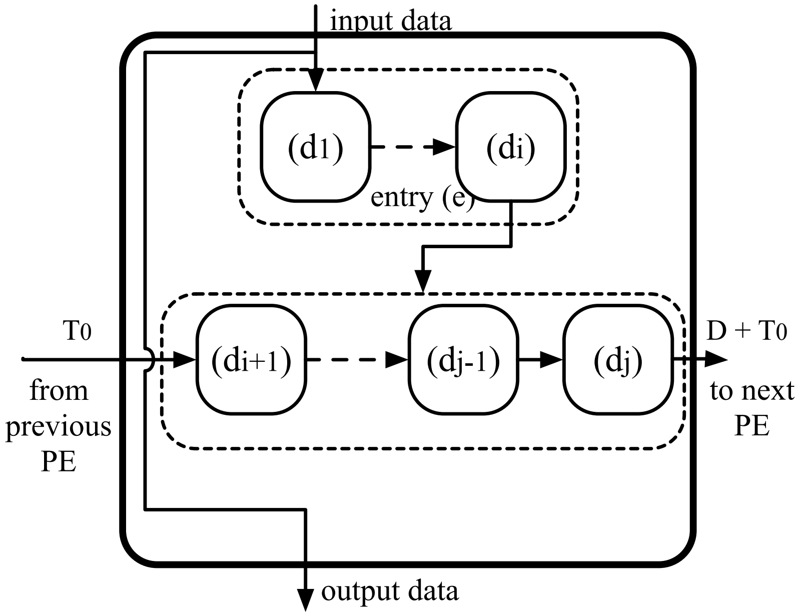


Figure 3.3. WOIL cell: d_n, $n = 0, 1, \ldots j$ is the delay of each block in clock cycles; blocks from 1 to i work on input data and on stored ones, while blocks from $i + 1$ to j are along the path P that connects the cells each other and their total delay is D.

3.2.2.2 WIL Systolic Arrays

WIL SAs are those in which each PE has one (or more) loops. In this case each cell exhibits data dependencies between signals traveling through the loop and data coming from outside; it is hence not possible to increase unconditionally the speed of the PE, without changing the rule for providing inputs. In this case it is necessary to exploit data interleaving if we want to increase the throughput.

WIL SAs are divided in WIL-S in which the cell stores the partial result and WIL-PT where results are evaluated through cells of one line. However, from a PE point of view, they are the same.

A cell with internal loop is shown in Figure 3.4. It is made of 4 parts: an entry section, made of blocks numbered from 1 to i; the forward part of the loop, made of blocks from $i + 1$ to j, the feedback part of the loop, made of blocks from $j + 1$ to $k - 1$; the output block, called k. Each of these blocks is associated to a delay d_n, $n = 1, 2, \ldots, k$.

Let us call T_e the total delay of the entry block, T_{ff} the delay of the forward side of the loop, T_{fb} the delay of the feedback in the loop and T_o the output delay. They can be computed as expressed in equation 3.2.

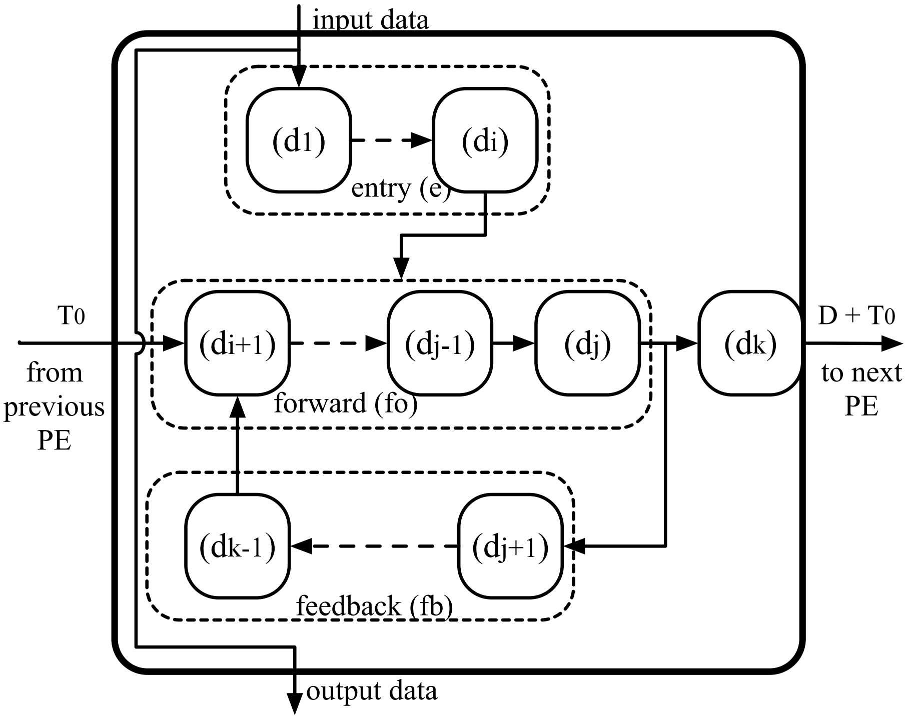


Figure 3.4. WIL cell: this PE is made of 4 parts: an entry section, the forward and feedback parts of the loop, and the output section. Data coming from previous PE can enter at any stage of the cell.

$$T_e = \sum_{n=1}^{j} d_n; \qquad T_{ff} = \sum_{n=i+1}^{j} d_n; \qquad T_{fb} = \sum_{n=j+1}^{k-1} d_n; \qquad T_o = d_k \tag{3.2}$$

Input data coming from outside enter in the first block, while data coming from the neighboring processing element can enter at any stage of the cell. This cell can be part of a systolic array where each result is finally stored in the cells of the array (WIL-S); or it can be part of an array where local results are re-used in the cell and also passed to neighboring cells (WIL-PT).

These cells usually have also a shift register used to pass inputs to neighboring cells, whose delay is L, and that is not shown in Figure 3.4 for sake of simplicity.

3.2.3 WOIL SA Optimization

WOIL SAs can be optimized using pipelining rather than interleaving because they do not present loop structures.

Let us assume that the j blocks of the structure depicted in Figure 3.3 cannot be internally pipelined (if any of them can be internally pipelined, then it could be divided into two different blocks). Each of theese blocks requires d_n cycles to complete the operation and during these cycles no new inputs can be given. Therefore, each of these blocks can receive valid inputs every d_n cycles (notice that if a block can be pipelined in s stages, it can be thought as s different blocks, each with its own delay).

The output rate of each block is the same of the input one, so, in order to match the delay condition for all the blocks of the processing element, input data should be given at least every K cycles: $K = \max\{d_n\}$ with $n = 1, 2, \ldots, j$.

The throughput that can be achieved is $1/K$; however, without exploiting the intrinsic pipelined nature of the cell (where each block is a new stage of pipe), data would be given at the end of the whole computation, hence the throughput would be $1/\sum_n d_n$; it is then clear that a speed-up of $\sum_n d_n/K$ can be achieved.

As shown in Figure 3.3, one cell receives data from previous cell at time T_0 and will produce the output at time $T_0 + D$. Inputs from the external are given every K_{min} cycles; the delay between one cell and the successive is D, hence the external inputs to the following cell shall also have a delay of D with respect to the corresponding inputs in the previous cell. In other words, first processing element can be fed with input data at time 0, K, $2K$, $\ldots$; second processing element instead is fed with data at time D, $D + K$, $D + 2K$, $\ldots$ and so on.

While this analysis is quite simple and straightforward, it is possible to define some general rules for the optimization of WOIL SAs: given one Systolic Array implementing a specific function, first it is necessary to understand if cells have no internal loops. If this is the case, the cell must be decomposed in the minimum atomic operations. For each of these atomic operations it is necessary to evaluate the delay. Then, the maximum of the delays will set the timing constraints for the PE.

3.2.4 WIL SA Optimization

WIL-S and WIL-PT systolic arrays can be optimized at a cell level exploiting *interleaving*.

Consider the PE shown in Figure 3.4: in order to match timing of inputs with delay of the feedback, inputs must be given every $T_{loop} = T_{ff} + T_{fb}$ cycles, that is the total time of the feedback loop. However, given the intrinsic pipelined nature of the structure, we can improve performance and usage of the cell giving inputs every $K = \max\{d_n\}$, as done for the cells without loop.

Every K cycles a new operation can start, and in this way N different operations can be interleaved, being $N = T_{loop}/K$ (integer division). After T_{loop} cycles, the second set of inputs is fed; evaluations on these inputs will be done and stored in the registers of the loop.

When T_{loop} is not a perfect multiple of K, the remainder of the division, called R, must be taken into account: after N operations have been started, the following one must start with a delay of $K + R$ with respect to the previous, so to have synchronization with the result coming from the loop. R represents a number of "stalls" that must be inserted between one set of N inputs and the following set.

Consider the following example: $T_e = 3$, $T_{ff} = 3$, $T_{fb} = 10$; it is possible to interleave $T_{loop}/K = 13/3 = 4$ operations, inserting a stall ($R = 1$) after each set of 4 inputs.

With respect to the normal usage of this kind of PEs, it is possible to evaluate in the same time N different operations, having an increase in performance of N. The number of stalls represents a factor of performance decrease, that must be carefully analyzed at design time; consider this other example: $T_{ff} = 20, T_{fb} = 6, K = 9$; it will result in $N = 2$ and $R = 8$. In this case it would be favorable to increase of 1 cycle the delay of the feedback, so to have 3 possible interleaved operations, and no stalls.

As far as the global array is concerned, in the case of WIL-S SAs, processing elements work independently each from the others, and the only connections are the shift registers to pass inputs from one cell to another. Being L the length of the shift register in each cell, inputs must be given with the same rule for all cells (number of interleaved operations and stall cycles), but starting at cycle $S_i = m_i \times L$; m_i is the Manhattan distance between cell i and the top-left one (if we consider data moving from top to bottom and left to right).

In the case of WIL-PT it is instead common to have a synchronization relation between cells that must be guaranteed. It is therefore not possible to derive a general equation to give some standard rule to provide inputs to the whole array.

3.2.5 Results

We have defined how it is possible to increase the performance of a Systolic Array acting on the elements of a Processing Element, inserting more pipeline stages to have higher operating frequencies, or acting on inputs, to provide them at the maximum rate supported by the Array. Now it is necessary to have some metrics that can tell us how good is the enhancement that we obtain increasing the pipeline stages or providing more inputs.

In order to evaluate this improvement, the Cell Updates Per Second (CUPS) parameter can be computed. This is a measurement parameter widely used when comparing architectures for protein sequence alignment; here we inherit it to compare performance among SAs in the most general case. However, the peak CUPS are often evaluated as the number of PEs times the frequency; here we introduce different methods to evaluate CUPS that directly takes into account the interleave level, the size of the array and even the number of inputs. In a systolic array, PEs rhythmically compute and pass data through the system [50]. Timing of operations follow a "wavefront" order; if we assume that the top-left PE is the one that starts operating first, the bottom-right one will be the last to complete operations. Given a finite number of inputs, the bottom-right PE is also the one that finishes later computation; hence, total time will be given by the time at which this PE will finish to compute the last result. Total time, T_{end}, is then given by the time for last inputs to reach last PE, plus the time to execute the operation inside the PE itself, called T_{cell}. In the following we will analyze the case of WOIL SA and WIL SA, evaluating the total time of computation and then the CUPS.

3.2.5.1 WOIL Systolic Arrays results

In a WOIL Systolic Array we have to differentiate between vertical and horizontal propagation. We consider here that results are evaluated through rows. Then, vertical propagation of inputs is achieved through shift registers of length L; horizontal propagation of partial results depends on the computational time of the cell, hence it is given by the delay of the horizontal path called D according to Figure 3.3. In the following we also consider that the SA is a square one with N PEs per side.

Call $T_{end}^{(i)}$ the time at which computation of i-th result is available. Then: $T_{end}^{(1)} = (N-1)L + (N-1)D + T_{cell}$, and considering last input p, we obtain equation 3.3.

$$T_{end} = T_{end}^{(p)} = (N-1)L + (N-1)D + (p-1)K + T_{cell} \tag{3.3}$$

Notice that this equation does not change if results are evaluated through vertical lines rather than through rows.

Equation 3.3 expresses the total time needed for executing operations on a $N \times N$ SA that receives p successive data from each input path. During this period of time each cell will execute p operation (one every time a new input is received). Then, the total cell updates are pN^2, and, given the clock frequency in ns, GCUPS (Giga Cell Updates Per Second) can be evaluated as:

$$GCUPS = f_{clk} \frac{pN^2}{(N-1)L + (N-1)D + (p-1)K + T_{cell}} \tag{3.4}$$

This equation must be adapted in two cases: when the array is used without interleaving operations (i.e., without exploiting internal pipeline of the cell), and when interleaving is exploited to achieve an improvement in performance.

In case of no-interleaving equation 3.4 can be adapted considering $p = N$. In case of n-interleaving instead, each cell will update n times than the previous case, hence $p = nN$; this increase reflects also at the denominator of equation 3.4 as an increase in total time.

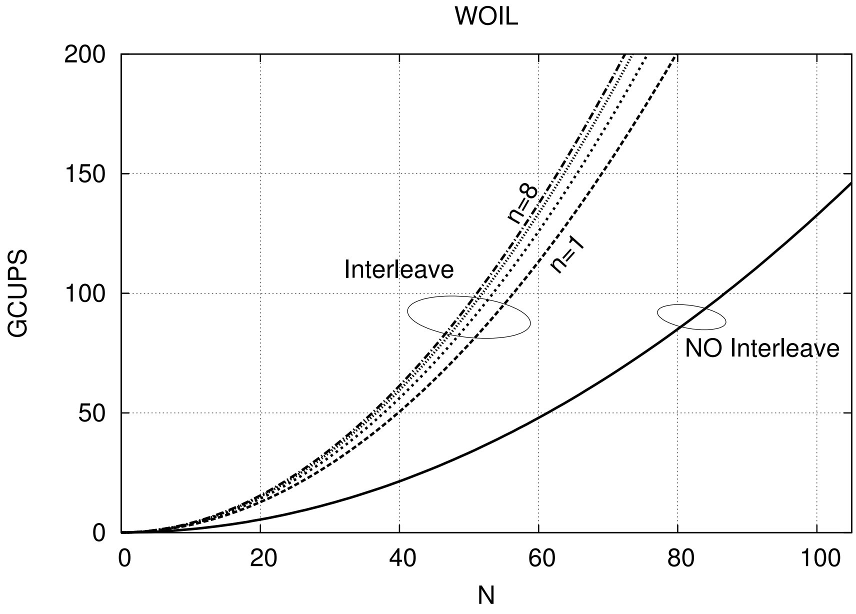


Figure 3.5. The effect of interleaving in terms of GCUPS: for the same $N \times N$ WOIL-S SA, interleaving and increasing frequency allow achieving better results. [51]

In Figure 3.5 we have plotted the GCUPS depending on number of cells N. It is clear that better results can be achieved by increasing frequency and using deep interleaves; if we just increase frequency and exploit pipelining, still with $n = 1$ we have an improvement with respect to the original case. The improvement saturates with the increase of n.

Since pipeline requires more hardware resources (registers) and in general it increases latency, designers should be aware of the fact that after a certain point performance improvement do not justify higher costs in terms of area (and consequently power dissipation).

3.2.5.2 WIL Systolic Arrays results

Consider a WIL-S Systolic Array; in this case the inputs for the last PE are inputs to the whole array that have been shifted through registers. We have previously defined L as the number of cycles needed for an input to pass through a cell, that is the number of registers of the shift chain. In this case we can assume that this value is equal to transmit data left to right or top to bottom. The first set of inputs will be available at the last cell after $2(N-1)L$, where $2(N-1)$ is the Manhattan distance between first cell and last one in an array of $N \times N$ PEs. Following equation results:

$T_{end}^{(1)} = 2(N-1)L + T_{cell}$.

Let us call K the delay between one input and the following one; if we have p inputs, then we have the total time will be as expressed in equation 3.5:

$$T_{end} = T_{end}^{(p)} = 2(N-1)L + (p-1)K + T_{cell} \tag{3.5}$$

Equation 3.5 and equation 3.3 correspond when $D = L$.

GCUPS in the case of WIL-S SA can be computed as reported in equation 3.6:

$$GCUPS = f_{clk}\,\frac{pN^2}{2(N-1)L + (p-1)K + T_{cell}} \tag{3.6}$$

This equation must be adapted in two cases: when the array is used without interleaving operations, and when interleaving is exploited to achieve an improvement in performance.

As already discussed for WOIL SA, in case of no-interleaving equation 3.6 can be adapted considering $p = N$. In case of n-interleaving instead, each cell will update n times more than the previous case, hence $p = nN$; this increase reflects also at the denominator of equation 3.6 as an increase in total time.

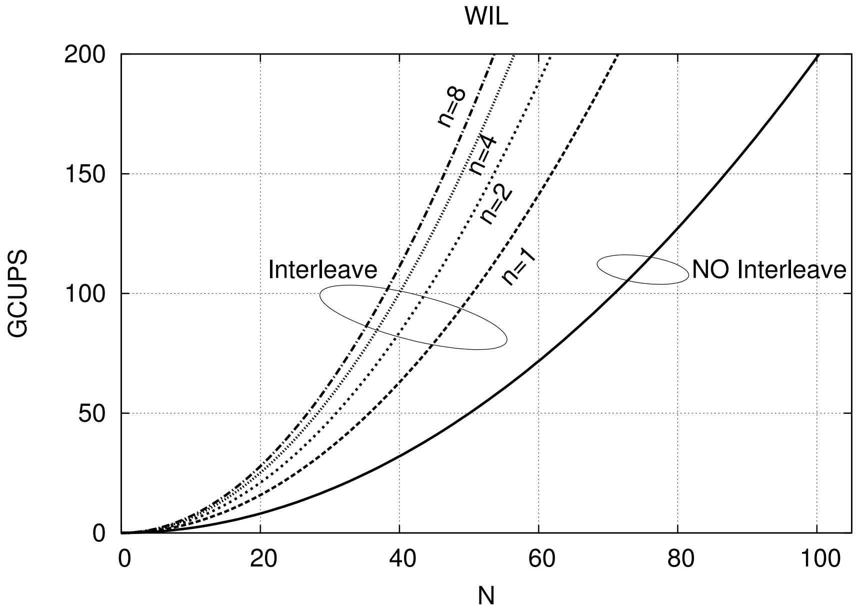


Figure 3.6. The effect of interleaving in terms of GCUPS: for the same $N \times N$ WIL-S SA, interleaving and increasing frequency allow achieving better results. [51]

The advantage of interleaving is shown in Figure 3.6. The higher the number of PEs in the array, the higher is also the increase, in terms of GCUPS, that is achieved using interleaving. However, the increase in performance saturates, and after a certain point, higher values of p do not translate in further significant increase in performance.

One particular case is interleave with $n = 1$; in this case there is not an interleave inside the loop (i.e. there are not 2 values traveling along the loop together) but still the pipelined nature of the PE is exploited to improve performances.

3.2.6 Data Interleaving in CMOS and NML

In previous paragraphs we have detailed the performance improvements that can be achieved applying pipeline interleaving to Systolic Arrays. The rigorous analysis allows to adopt this mechanism for nearly all kind of SAs. In this paragraph we detail the actions that must be taken by a circuit designer to achieve best-performing circuits with the right adoption of pipeline interleaving. The approach is different for CMOS and NML, so we will also point out what are the analogies and the differences.

Circuit design in CMOS is about finding the right balance between operating frequency and area occupation. Power dissipation will be closely dependent on area. Given an initial design for a CMOS Systolic Array, used to implement a certain algorithm, the designer must first identify the atomic elements of each Processing Element. This allows to understand what is the current critical path delay and the basic level of pipelining/interleaving that can be applied. Sometimes this could also be enough for an efficient usage: if the array has been originally designed trying to balance as much as possible the critical paths and achieving a high operating frequency, it could be sufficient to act on inputs providing them in the right manner (i.e. with the right delay and if the circuit already supports it, interleaving unrelated operations).

However, in most of the cases the design will need a refinement and should be speeded-up. Here is when pipelining and interleaving techniques can give their contribute. Given that pipelining technique is quite common and similar to interleaving except for the relation between successive inputs, let us concentrate on interleaving.

If the critical path is outside the loop, it is sufficient to insert additional registers in between and in this way reduce the value of $K = \max\{d_n\}$. If the critical path is in the feedback loop instead, it is possible to apply proficiently interleaving. In this case the number of operations that can be interleaved will grow, but at the same time area and power dissipation will be impacted by additional registers. Usually the first stages of interleaving are quite advantageous, while increasing too much the interleave level will not produce additional value.

So the design is an iterative process, in which additional stages of interleaving are added one by one and every time some common metrics like the Power Delay Product are evaluated to find the best level of interleave. It is also possible to perform an initial analysis with the equations that we have provided. These allow to extrapolate a trend in performance increase given by additional level of interleave. In the same way it is possible to design some curves that reflect the circuit area increase with interleaving. Combining these curves it is possible to identify the best-minimum.

The description below is valid for CMOS, where it is possible to change the level of interleaving adding new registers in the loop. When we consider NML instead, the delay of the loop is fixed and it is given by its layout, due to the "Timing = Layout" problem. Changing (increasing) the delay would mean to re-design all the circuit to have the correct synchronization among signals. For this reason in NML the approach to data interleaving is different: given a circuit layout already defined, a static analysis allows to understand how many operations can be interleaved. Most of the work is then devoted to the generation of the correct sequence of inputs to use interleaving and exploit the intrinsic pipelined nature of this technology. Without interleaving, NML circuits would have dramatic performance due to these long delays.

3.3 Latency Insensitive Systolic Arrays

Another enhancement that can be applied to Systolic Arrays in the eye of NML implementation is the creation of Latency Insensitive Processing Elements. We will go more in the details that drove us to take this research path in paragraph 3.3.1. Then we will describe the communication protocol adopted in paragraph 3.3.2 and the structure of the Processing Element in paragraph 3.3.3. Finally we present an application example in paragraph 3.3.4.

3.3.1 Motivation

NanoMagnet Logic (NML) is affected by the "Timing=Layout" problem, as we have described in the introduction. Generally there can be two ways to approach this problem.

The first is to have a careful design that is able to balance all the paths delays so to ensure that data arrive at a computational block with the correct timing. This is a common approach, that we have used for example for the design of NML Systolic Arrays where all processing elements can be identical. In this phase we do not consider eventual tolerances necessary for the production process, but we consider that the final circuit can respect the delays imposed during the design phase and verified through simulations.

The second way to deal with the "Timing=Layout" problem is to make circuits able to support different delays of inputs. This is the path that we want to follow in this research topic. Basically, the circuit is enriched with some synchronization blocks, so that when an input arrives at the boundary of a computational cell, it is stored and saved until all other inputs are available. Only at that moment the processing is executed.

Normally, a Latency Insensitive circuit requires additional area, power and generally introduces some delay. However, without this kind of circuit the design of NML circuits becomes nearly unpredictable. While for classic systolic arrays it is possible to approach carefully one Processing Element and then replicate it N^2 times, thus reducing the design of the Array to the design of the singe PE, there are other architectures where there is not such regularity of structure.

In Chapter 4 we will introduce a Reconfigurable Systolic Array (RSA), where each processing element can be configured to execute a different operation. Of course operations have not the same delay: a multiplication may require much more time than an addition. If we need to synchronize all the operations with the same delay, we should set on the lowest operation, wasting an incredible amount of time. On the contrary, adopting a Latency Insensitive approach, each Processing Element can execute operations in the minimum necessary time. Then synchronization blocks are used to manage all the inputs. This is far more advantageous and gives even more flexibility to this architecture.

We still consider Systolic Arrays as the principal architecture for NML because, as we have seen in Section 3.1, they can deal with the main disadvantages of this technology. Nevertheless this approach can be used also with any other circuit. Moreover, this approach is valid also for CMOS circuits in case of logic blocks with different delays.

Given a Systolic Array, we design in the next paragraphs the elements necessary to provide this Latency Insensitive interface among processing elements. The whole circuit is of course synchronized using clock signals, but the PEs communicate using an asynchronous protocol.

We have exploited the opportunity of this re-design of Processing Elements (with interface blocks for Latency Insensitive circuits) to introduce also a new kind of PEs: these are octagonal elements that can communicate with eight boundary cells. This gives more flexibility to the array and will be particularly interesting for image processing algorithms as we will detail in next section. In our flexible design PE inputs that do not receive any signal are simply ignored and an eight-sided cell can be used as a cell with less I/O sides. Each I/O side has a data bus and a control bus. The first one carries both information that has to be processed and results of execution, while the latter contains configuration bits for the PEs.

3.3.2 Proposed Communication Protocol

The first step in the design of Latency Insensitive circuits is the communication protocol that must be used between PE. The final goal is to have array cells that handle the local synchronization on their own and perform their computation correctly even if they receive different inputs at different times. This can achieved employing a self-timed approach using an handshaking protocol. Several possible solutions have been analyzed before settling for a two-way handshake, the simplest protocol that prevents errors in the propagation of information.

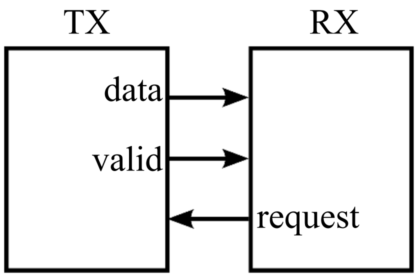


Figure 3.7. Signals involved in the communication between a transmitting cell (TX) and a receiving cell (RX).

Even if all the cells are identical, neighboring PEs are not necessarily performing the same task at the same time; if a PE is ready to send out its output, the reader might not be ready, or counter wise the reader can be ready but the sender is not. The chosen two-way handshaking technique has the advantage of providing blocking read and write operations, which means that a PE is stuck in either operation until it is over. This means that no data can ever be lost, but there is a drawback in the timing overhead. Synchronization between PEs requires an exchange of signals and this takes time: introducing additional delay slows down the whole structure, but that is a small price to pay if correct data transmission is guaranteed. The asynchronous communication protocol involves two synchronization signals: **request** and **valid**, as shown in Figure 3.7.

When the transmitting PE (TX) wants to send out an output, it sets the **valid** signal and waits. The receiving PE (RX) now knows that its input is valid and it reads it as soon as it can, storing it inside its input registers. The RX PE then sets the **request** signal for the duration of one clock cycle to communicate that the data has been received. To prevent loss of information a new read will not start until the **valid** signal goes to ‘0’ again.

Upon seeing the request the TX PE sets the valid signal back to ‘0’ and the communication is over.

The example of Figure 3.8 presents the ideal situation where the output data is read immediately. In this case the TX PE is stuck in write mode for the least amount of time: 3 clock cycles. If the RX PE is not ready to receive a new data as soon as **Valid** becomes True, the sender PE has to wait for a longer time. The RX

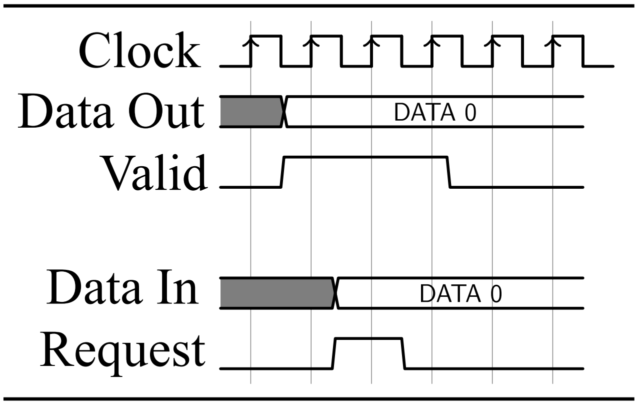


Figure 3.8. Timing diagram for the transmission of DATA 0.

PE takes only one clock cycle to read and store its input data, but it may be stuck waiting for it, introducing a delay.

In conclusion, the delays are unpredictable, but they are always correctly managed. They are related to the delays on the array inputs, to the way the data propagates through the structure and to the difference in execution time of the different PEs. If a PE has more than one input side, it waits for all inputs to be available before starting its computation. In the same way, before starting a new computation, a PE waits until all its outputs are read by its neighbors. Because of this, the reader block outside plays a key role: reading the outputs of the array is necessary to make it work. The blocking nature of the read and write operations can slow down the circuit, but it guarantees correct execution. At this point the true data-driven nature of the structure becomes evident, and it is exactly this behavior that keeps the computational wavefronts separated and prevents loss of information or wrong computation. No global control is required, each PE independently follows a few synchronization rules and that is enough to make the system work.

3.3.3 Latency Insensitive PE

We will now describe in more detail the architectural elements that must be introduced to give a Processing Element the Latency Insensitive feature.

Normally a Processing Element has inside the computational elements to execute the algorithm. In this case instead, the architecture of the PE shall be modified as shown in Figure 3.9. The cell is composed of three type of blocks:

- **Algorithm Block:** this is the common block that executes the operations to implement the algorithm;

- **I/O Blocks:** these blocks, one for each side, manage communication between PEs.
- **Communication Block:** this element manages synchronization among all I/O blocks, to guarantee that no new operation is started inside the PE if the output blocks have not yet delivered their results.

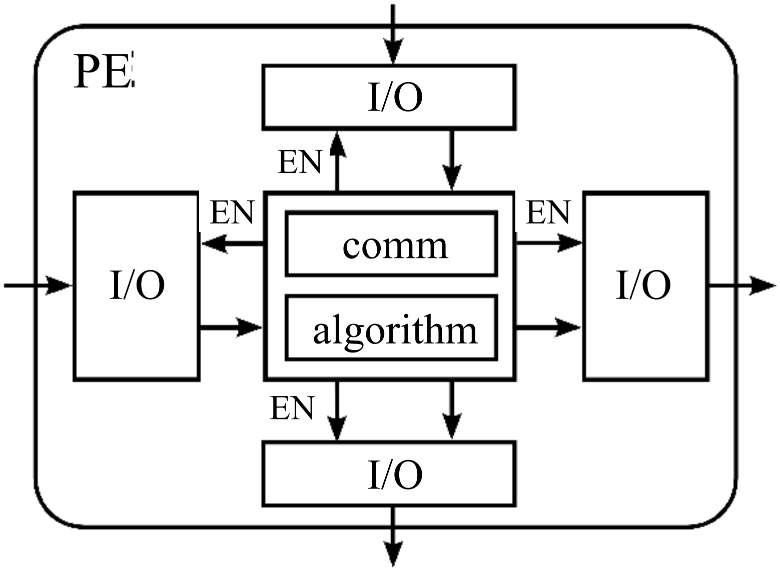


Figure 3.9. Internal blocks of a 4-sided cell. The EN signals are the enables for the registers of the I/O blocks.

In the following each of these elements is described in more details.

3.3.3.1 Algorithm Block

This is simply the element that executes the algorithm. It is composed of logic computational blocks and intermediate registers if necessary. In Chapter 4 we will describe how to design an Algorithm block that can execute different operations and can be run-time configured to change the algorithm that it implements. An example of simple algorithm block is instead presented in Section 3.4, implementing the Floyd-Steinberg algorithm. This is the only block that does not depend on the Latency Insensitive nature of the circuit.

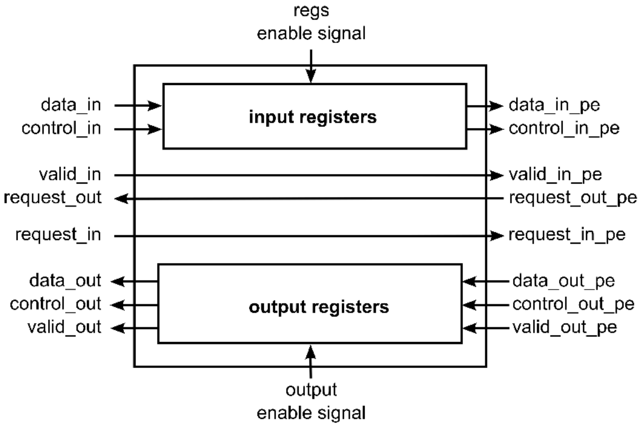


Figure 3.10. Internal I/O block structure. The signals on the left are the ones coming from the outside, while those on the right go inside the cell. [52]

3.3.3.2 I/O Blocks

Input/Output blocks are placed at the boundaries of the PE. There will be one I/O block for each side. While commonly one side is used only as input or output, this I/O block can manage both input and output at the same time. In this way it is not necessary to differentiate at design time and this makes the entire PE more flexible also in the eye of the Reconfigurable version.

A simple schematic of the I/O Block is shown in Figure 3.10. The input valid and request signals do not pass through registers, they are just used for synchronizing data exchange. Data and control bits, instead, need to be stored in a set of registers enabled by the PE only when new data is requested. The outputs are also stored in a set of registers, so that they do not change until the next outputs are ready.

The management signals that enter in the I/O Block are controlled by the Communication Block described in next paragraph.

3.3.3.3 Communication Block

The Communication block handles synchronization between I/O blocks and indirectly with other PEs. Its role is to execute the protocol defined in paragraph 3.3.2. Let us define a “communication cycle” the time from when inputs are read to when outputs are delivered inside a PE. During a communication cycle, as long as the cell is waiting for new inputs, it is said to be in *waiting* state. Once the inputs are received, it exits this state and does not return in it until it has delivered all its outputs.

Inputs are of course received using the **valid** and **request** signals to synchronize

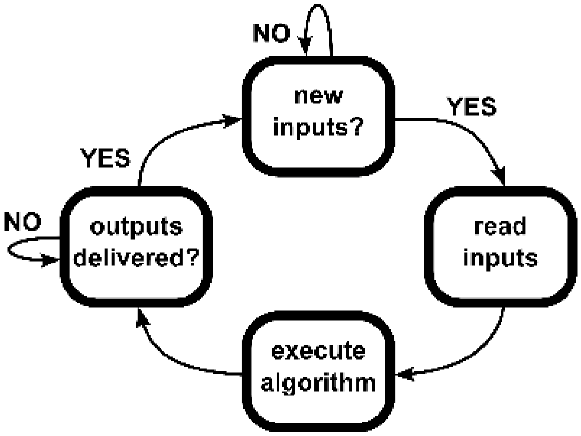


Figure 3.11. The algorithm implemented by the Communication block. [52]

the communication. The communication block now waits for the algorithm block to produce the output values; once they are all ready, the corresponding **valid** signals are set. When all outputs have been delivered, the PE goes back to the waiting state and a new cycle can begin.

To generate this logic, several sub-blocks have been defined in the Communication Block. These sub-blocks generate flags that are used to synchronize the communication algorithm and verify its execution. Mainly they are:

- Indata_valid: a flag that indicates that inputs are valid and can be used for computation;
- Waiting: a flag that indicates whether the PE is in *waiting* phase where not all inputs are ready, or it is in computation/delivering phases;
- Valid: this is the flag that indicates that result is ready to be delivered;
- Delivered: this is the flag that indicates that results have been delivered and a new cycle can start.

We will not enter into the details of the logic circuit design, for which it is possible to refer to [52], but is useful to highlight some of the main interactions among flags and blocks. A simple scheme of the algorithm processed by the communication block is shown in Figure 3.11.

First, the **Valid** signal is set by the algorithm block and directly used by the output blocks. The **Delivered** signal is instead dependent on the successive PEs that refer when they have received and started using the result of the PE. When

Delivered is set, **Waiting** flag is asserted. It will be reset when the other flag **Indata_valid** will become true.

This is basically the behavior of the communication block, that has been designed in VHDL and simulated along with the others. One example is reported in next paragraph, while a more complex one based on Floyd-Steinberg algorithm is described in next Section.

3.3.4 Application Example: Matrix Multiplication

The first chosen algorithm to show the impact of Latency Insensitive circuitry on Systolic Arrays is Matrix Multiplication. This algorithm can be mapped to a square Systolic Array in two ways: a WIL Systolic Array with results that will be stored inside the PE, or a WOIL Systolic Array with results traveling in a column (or row) and one of the two multiplicand matrices stored in the PE. For our test we consider the WOIL implementation.

The array we have proposed for Latency Insensitive approach is an Octagonal array with eight-sided PEs. If we want to map the Matrix Multiplication in this array, we propose two options as shown in Figure 3.12. It is clear that the array is not used completely, since some cells will not be necessary. Nevertheless this is a good option if this array shall implement several algorithms. As we will see in next Section, the octagonal Systolic Array can be used for image processing algorithms such as the Floyd-Steinberg algorithm. In the eye of a reconfigurable array, the flexibility of this architecture is fundamental to allow as many algorithms as possible to be mapped in this array.

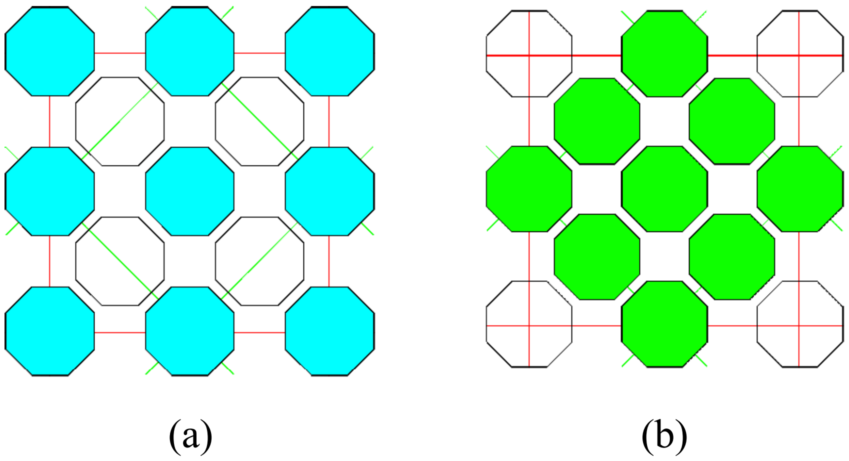


Figure 3.12. Octagonal Systolic Array used for Matrix Multiplication. Two possible mappings.

In the WOIL implementation of the Matrix Multiplication algorithm each PE is composed by a multiplier and an adder. The parallel multiplier could be extremely expensive in terms of area occupation and power dissipation. For this reason we designed a serial Booth multiplier that works on N bits at every cycle. While this is more time consuming in general, it is possible to identify a speed-up mechanism, described in next paragraph.

3.3.4.1 Serial Booth Multiplier

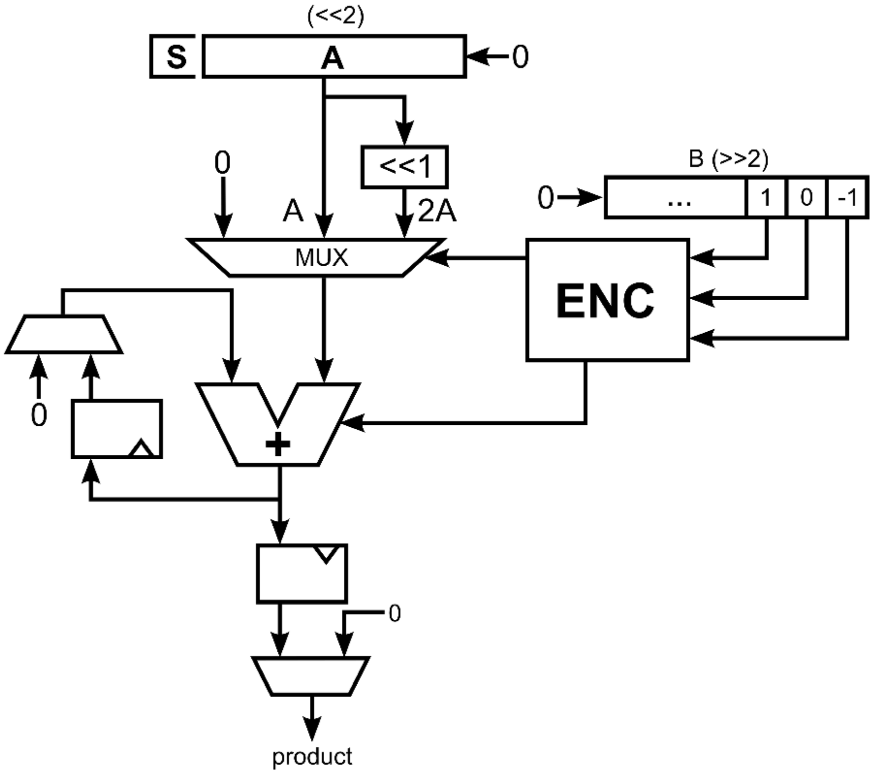


Figure 3.13. Structure of the serial version of the Booth multiplier.

The Serial Booth multiplier shown in Figure 3.13 basically implements the common Booth algorithm, with an accumulator to sum up the partial result. Usually when a multiplication is executed bitwise, there will be some elements that remain at 0 and do not add any information. For example, imagine to execute 00110101b × 00000101b. The second operand has five '0' in most significant bits, therefore the result of the multiplication of the first operand for these five bits is '0' and can be neglected.

Through a control mechanism we enhanced the Booth multiplier, in such a way to be able to detect when the successive operations will be useless. So, the **Valid** signal is asserted even before the real computation has finished, but when the result is already the final one. Of course this will give an uncertain delay to each Processing Element. Depending on the inputs, each PE could need more or less clock cycles to produce the result. This result availability uncertainty is managed with the Latency Insensitive Array.

We have implemented this array in VHDL and simulated with a random set of inputs. Results were correctly achieved and the variable delay was managed by the Latency Insensitive circuit as expected [52].

3.4 Systolic Array for the Floyd-Steinberg algorithm

In previous Sections we have introduced Systolic Arrays and their optimization techniques. Then we have focused our attention on Latency Insensitive Systolic Arrays, showing one additional mechanism to give this architecture more flexibility and to make it more attractive for NML.

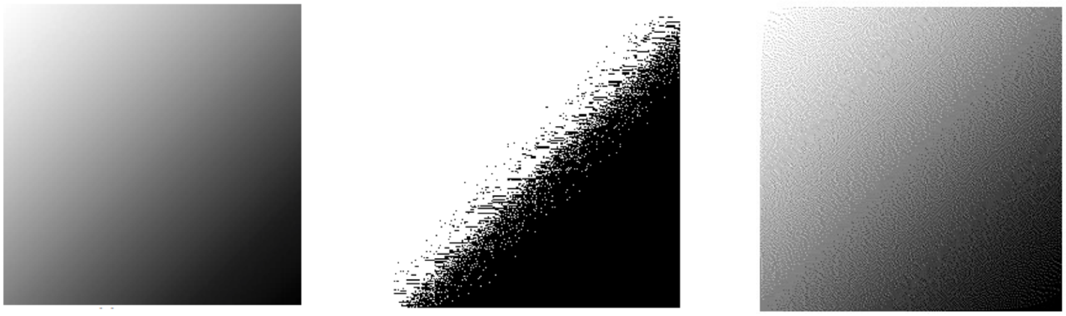
In this Section we present an example of application based on image processing. This on the one hand shows that the Systolic Array is a great architecture when multiple data can be processed in parallel, as in image processing algorithms. On the other hand it is used to demonstrate that the latency insensitive circuit can be applied to real operating environments.

We first provide an introduction to the Floyd-Steinberg algorithm in paragraph 3.4.1, and then we show how to map it to the Latency Insensitive Array in paragraph 3.4.2. Finally

3.4.1 Floyd-Steinberg Algorithm

In digital audio and video processing, the quantization of data yields error patterns that produce undesirable artifacts, such color-banding in images. Dithering techniques are then used to redistribute the quantization error to prevent these patterns. One of the most typical application is the conversion of a grey-scale image to a black and white image.

When the conversion is performed without dithering (Figure 3.14(b)), each pixel is simply compared to a fixed threshold value. With this reduction of quantization levels large monochrome patches are generated and the re-quantization error on the whole image would be very large. Employing an error diffusion dithering technique, the redistribution of the quantization error lowers the overall error on the whole



(a) Original image

(b) Threshold

(c) Floyd-Steinberg dithering

Figure 3.14. Image processing - bit depth reduction.

image, eliminates the monochrome patches and preserves more details of the original image. In Figure 3.14(c) Floyd-Steinberg dithering has been applied.

The Floyd-Steinberg algorithm is quite old as concept but, since it produces a very fine-grained dithering, it is still used and there exist a lot of variations. The algorithm scans the image pixel by pixel, from left to right and from top to bottom, starting in the top left corner. It reduces the quantization level on the current pixel and pushes the residual error onto the neighboring pixels. The distribution of the error follows is done weighting the error as in matrix 3.7.

$$\begin{bmatrix} & * & 7/16 \\ 3/16 & 5/16 & 1/16 \end{bmatrix} \tag{3.7}$$

The asterisk represents the current pixel while the different values are the weights associated to the four neighboring pixels. The blanks are the positions of the pixels that have been already scanned. Mapping this algorithm onto an array structure is rather intuitive: each pixel is assigned to an octagonal PE, which reads its four inputs, performs quantization on its own pixel and diffuses the quantization error onto its four neighboring PEs.

The original pixel values have to be preloaded inside the PEs and then, during execution, they are reevaluated considering the redistributed quantization error. The local communication involves only the quantization error, so the new values do not propagate through the structure. Instead, they can be retrieved in parallel with the successive preload phase.

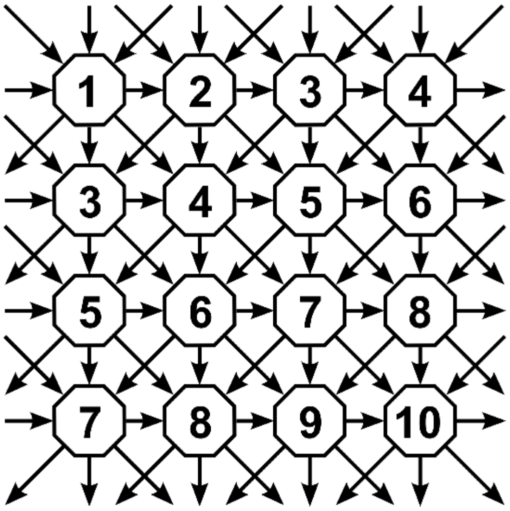


Figure 3.15. Signal propagation in the octagonal Systolic Array for Floyd-Steinberg dithering.

3.4.2 Latency Insensitive Implementation

The PEs have eight sides and are placed in an orthogonal grid, replicating the grid of pixels in an image. The array structure of Figure 3.15 presents the order of processing with numbers inside the cells.

During execution phase the PEs add the four inputs to their own pixel value, reduce the quantization level of the result and multiply the quantization error by the output coefficients. So, the computation of the elements of matrix 3.7 are done on the pixel PE, and transmitted already with their weight to the destination PEs.

In our PE implementation, the four sums are performed one after the other by a serial adder, controlled by a 2-bit counter, as shown in Figure 3.16(a). This structure has been chosen over a parallel approach to save area. Once the input quantization errors have been added, the quantization level is reduced finding the closest color in the available palette (black or white). Finally, the new pixel value is compared with the original one to obtain the quantization error and, in the Coefficients block, the distribution coefficients are applied to obtain the outputs of the PE.

The schematic of the entire PE circuit is shown in Figure 3.16(a).

The coefficients are applied through a series of shifts and sums, so no multiplier is required, as shown in Figure 3.16(b). This is possible because the values are fractions of a power of 2. Shifting right performs an integer division, which truncates any fractional value. In general such approximation could cause issues during computation and lead to wrong results, but in the case of image dithering this does not matter. The purpose of these operations is to obtain a new quantized image

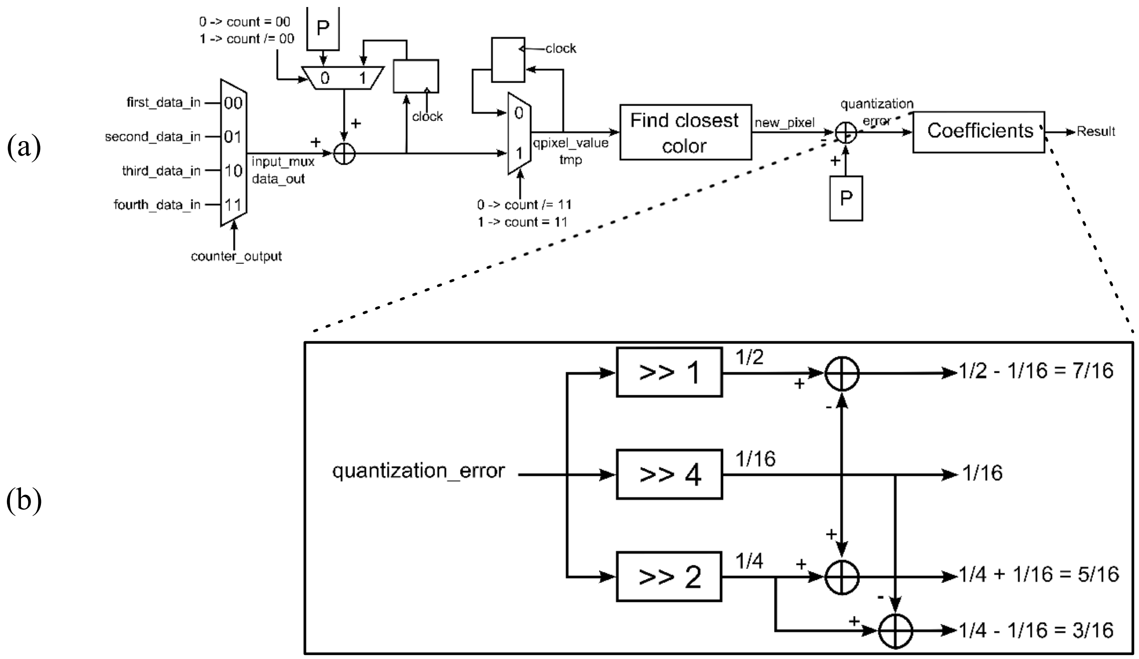


Figure 3.16. (a) PE Logic for Floyd-Steinberg dithering algorithm. (b) Implementation details for the Coefficients block.

with redistributed error and no color banding, so that, while being represented on a smaller number of bits, it maintains a certain level of detail and clarity when compared to the original.

The quality of the final result is to be judged by the human eye, so small approximations on the decimal numbers of a pixel does not matter, especially if the original representation cover enough bits. What matters is that the density of black pixels after dithering is close to the average level of grey in the original image.

3.4.3 Simulation

The Floyd-Steinberg algorithm implemented in the Latency Insensitive Systolic Array has been a great opportunity to push forward our research in several ways. First, we have mapped this algorithm to a new, octagonal, systolic array. Second, it has been the occasion to see the performance of such systolic array for image processing algorithm. Last, but not least, we had the opportunity to verify with more details and with a real scenario the capability of the Latency Insensitive Array.

We have created a Matlab script that performs the Floyd-Steinberg algorithm (and we have double checked with some library functions in Matlab). Then we have provided the same inputs to the Matlab script and to the VHDL of the Array in simulation. The delay of each cell was chosen randomly by a configuration file.

In this way we have verified that: the result of our hardware module are correct, compared to the Matlab results; the random delay given to each cell does not block the computation and is correctly managed by the Latency Insensitive SA. Each PE is able to autonomously wait for all the inputs and produce the output only when it is correct. Further details about this simulation are given in [52]. Notice that this algorithm has no back-dependencies, that is one important constraint for this type of data dependent architectures. If there was a data dependency between the output of one PE and the input of a previous PE, then deadlock prevention mechanism are necessary.

3.5 Final Remarks

In the approach to NanoMagnet Logic we have identified some technological limitations: low clock frequency and long latency of wires are the most important ones. While at the moment there is not a technological solution to address these limitations (except from a drastic change of technology), it is possible to provide answers to these problems through careful architectural choices. Long interconnections should be avoided and parallel architectures that can increase the throughput (limited by the low clock frequency) should be preferred.

In this framework the ideal architecture solution is represented by Systolic Arrays. These are parallel architectures made of regular Processing Elements locally interconnected. Systolic Arrays have been thought till now only for CMOS technology, and only recently they have gained a certain amount of interest.

The Systolic Array simple concept can be enhanced with several considerations. In this research path we have thought to Systolic Arrays for NML and we have identified some improvements that can be applied. The following achievements can be mentioned:

- Classification of Systolic Array based on PE structure: till now some classification of Systolic Arrays have been presented, based on the size and shape of the architecture and on the way data travel through the array. Here we provide a new taxonomy, that is based on the internal structure of the PE and on the way results are exchanged with other PEs. This taxonomy is fundamental to distinguish the improvement that can be applied (pipelining for PEs without internal loops, interleaving in the other cases).

- Data Interleaving in Systolic Arrays: if implemented in NML, a Systolic Array will still be inefficient due to a certain delay of connections (even the shorter ones). We have defined a rigorous way to introduce data interleaving and avoid the throughput reduction due to this delay of wires. The approach can be used with any Systolic Array and any technology.

- Latency Insensitive Array: synchronizing an NML circuit is quite complex due to significant delays of logic elements. To avoid that, it is possible to use Asynchronous logic circuits. We have introduced this improvement in Systolic Arrays with a precise communication protocol. Nevertheless, the same approach can be generalized to any logic circuit (in NML and other technologies).

- Systolic Array for the Floyd-Steinberg algorithm: with the aim to test the latency insensitive systolic array, we have designed a processor able to execute correctly and with random delays the Floyd-Steinberg algorithm for image dithering.

With all these achievements it is possible to state that Systolic Arrays are a great architecture for NML and that it can be really exploited using the improvements proposed to outperform CMOS and trace the path towards the post-CMOS era. Nevertheless, other improvements to the Systolic Array concept can be introduced and they are described in next chapters.

One important aspect to underline is that these improvements, tailored for NML in our discussion, are valid also for other technologies. Also CMOS circuits can benefit from data interleaving as we have stated in paragraph 3.2.6. Therefore, even if NML will not be the technology of the future, these achievements can be still considered and could improve new nanotechnologies with similar characteristics as well.

Chapter 4

Reconfigurable Systolic Array

In previous Chapter we have analyzed Systolic Arrays as ideal architectures to design NanoMagnet Logic (NML) circuits. Systolic Arrays can guarantee a throughput increase with parallelization of tasks in several Processing Elements (PEs) and at the same time reduce at the minimum the delay of wires, having regular and short interconnections.

Nevertheless, we have seen that some improvements shall be introduced in order to meet the peculiar characteristics of this new technology: asynchronous logic circuits and data interleaving have been presented and discussed. These optimization are applicable also to CMOS technology, thus making them even more interesting.

With the same principle we have studied an enhanced architecture that we have called "Reconfigurable Systolic Array" (RSA). This is a Systolic Array able to implement several algorithms. We present in this Chapter this research path, starting from the motivations that drove us to this choice, in Section 4.1, to the description of the architecture in Section 4.2. Then we present some of the algorithms that can be mapped onto the RSA in Section 4.3, and finally we summarize the results in Section 4.4.

4.1 Motivation

In this Section the reasons that led us to the introduction of the Reconfigurable Systolic Array are discussed. First, we describe the limits of common Systolic Arrays in paragraph 4.1.1. Then the approach used for reconfigurability that we want to introduce is presented in paragraph 4.1.2. Finally we will cite some already existing reconfigurable architectures in paragraph 4.1.3.

4.1.1 Limits of Systolic Arrays

We have analyzed thoroughly the adoption of Systolic Arrays for NML technology in previous Chapter. The adoption of Systolic Arrays for NML is quite obliged, in order to deal with the main limitations of this technology.

Let us consider for a moment the actual usage of Systolic Arrays (SAs) with current technology. In CMOS, SAs are usually designed as dedicated co-processors to accelerate a given task (usually, a computational-bounded algorithm). This is done implementing ad-hoc designs that cannot be reused for other operations, i.e. they are algorithm-dependent. For this reason SAs have been adopted only for a small subset of problems requiring a high number of calculations: signal processing [34], video processing [37], biological sequence comparison [43][44].

If, instead, it is required to accelerate a given task without the effort of designing and producing a costly dedicated Systolic Arrays, the same algorithms can be mapped to FPGA [53][54]. In this way the hardware could be reused if a different algorithm must be implemented.

The first case (ad-hoc ASIC) can be used only for a small set of algorithms while the second (implementation on FPGA) limits the operating frequency and the number of Processing Elements that can be mapped. We propose, as a solution to this disadvantageous duality, the introduction of a Reconfigurable Systolic Array. It must inherit the following attributes:

- From classic Systolic Arrays, it must have a regular layout and high clock frequency in CMOS. It must be faster than FPGAs, so to guarantee higher throughput.

- From FPGA, it must have a reconfigurability capability. It must be able to execute several algorithms and must have a configuration phase to choose the operation to execute inside each PE.

The concept of the Reconfigurable Systolic Array is graphically summarized in Figure 4.1: the RSA shall have an adaptability similar to the one of FPGAs but with speed performance of normal Systolic Arrays.

With these constraints we have approached the design of the Reconfigurable Systolic Array as described in next paragraph.

4.1.2 The Reconfigurable approach

The Reconfigurable Array Architecture shall overcome one of the main limitations of classical Systolic Arrays, i.e. the algorithm-dependent relation. Nevertheless it shall maintain some of the key features of Systolic Arrays, mainly for NML implementation, that are: avoid long wires and maintain identical PEs.

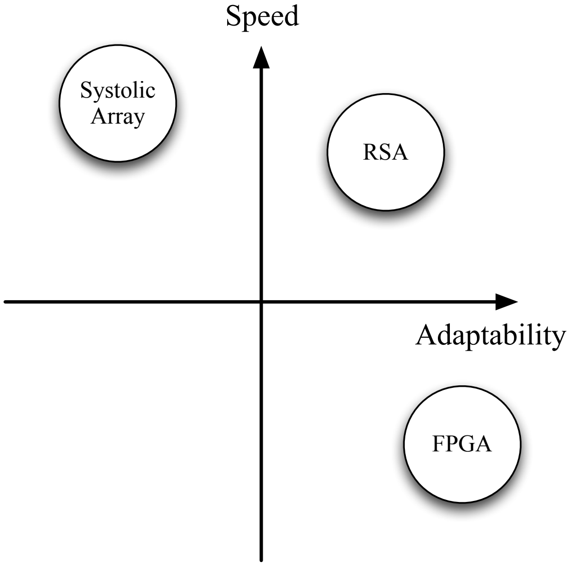


Figure 4.1. Reconfigurable Systolic Array concept: in a speed-adaptability graph, it must guarantee an adaptability similar to FPGAs but with speed of normal Systolic Arrays.

In particular it is important that all PEs are almost identical when we consider the implementation in NML. Since an automatic tool for NML is not yet available, the design of PEs in this technology shall be done ad-hoc and by hand. Subsequently, designing several different PEs would be too costly and time-consuming. Therefore, it is necessary to have one single PE type that can be configured in different ways in order to implement several operations. Moreover it is necessary to balance the PE so that the delay to execute one operation or another is always the same. This is done for sake of simplicity in the first version of this circuit, while in future version we could introduce the Latency Insensitive circuitry and thus use the only minimum time necessary to execute each operation. Without the asynchronous approach however, to guarantee synchronization, it is necessary that all the operations require the same amount of time. Feedback signals will be present in the architecture to have WIL Processing Elements. The delay is not known before the actual implementation, but it is possible to use Data Interleaving to maximize performance acting on inputs.

The Reconfigurable Systolic Array shall comprise three phases: the first phase is called “Reset” and is necessary to clear the registers and reset the array to an initial configuration; the second phase is called “Configure” and is used to choose the operations to execute in each Processing Element; the third phase is called “Execute” and is when the actual operations are performed.

Concerning the “Configure” phase, there are two different options that we have approached for the design of the RSA. Taking into account that the inputs are always provided from boundaries, we have decided to use the left side to introduce the configuration signals. It is possible in this case to manage in two ways the configuration: 1) each configuration word is provided to one row of the array and is used to configure the entire line; 2) in each configuration line N words are provided, each corresponding to one PE, used to configure with a fine-grained approach each PE independently. This difference in further described in paragraph 4.2.2.

4.1.3 Existing Reconfigurable architectures

In this paragraph we present some Reconfigurable architectures that can be considered as competitors of our RSA. This analysis has been performed to extrapolate some of the key features of these solutions that can be implemented in our RSA, and in order to evidence the situations in which our architecture allows more flexibility.

Among the Reconfigurable SAs that have been proposed in recent years, the work in [55] presents a reconfigurable architecture for VLSI implementation of BP neural networks with on-chip learning, while in [56] a general purpose architecture for the parallelization of nested loops in reconfigurable architectures is described. Both SAs adopt reconfigurability to address a specific problem's scale.

There are several other examples of reconfigurable architectures presented in literature. We present here some of the most interesting: You *et al.* [57] have presented a Reconfigurable Systolic Array for solving either single-source shortest path problem or 0-1 knapsack problem, where the array can be reconfigured into the other and vice versa according to the problem; Ishimura *et al.* [58] have proposed a dynamically reconfigurable array, tested only for matrix multiplication with different size of operands; finally Mishra *et al.* [59], have presented an array dynamically reconfigurable to execute Viterbi decoder with different length of K parameter (from 3 to 6). All these example show that the field of application is extremely limited.

A fully reconfigurable architecture has been proposed in [60]. In this case, a systematic design approach to map two or more algorithms into a single SA is exploited. While this research is quite similar to the one we propose, the work in [60] however lacks real data and physical implementation, and it has been conceived for CMOS implementation only.

Finally it is clear that while some interest has been given to these reconfigurable architectures, a final solution has not been yet found. It is also evident that a clear and exhaustive comparison is beyond the scope of our activities. In this framework we propose an array that is as much as possible reconfigurable, shifting to the mapping and configuration phases the effort of making it effective for a given algorithm. In this way the hardware costs for the design of the architecture do not impact as it would have done an application specific array. On the other hand it is of course necessary to work on the correct choice of configurations to avoid any area or time wasting in the execution of the algorithm.

4.2 Proposed Reconfigurable Systolic Array

In this Section we describe in details the Reconfigurable Systolic Array that we propose. The architecture is shown in paragraph 4.2.1; then we discuss the important preloading and configuration phase, in paragraph 4.2.2. Finally we show the results that can be achieved implementing this architecture with NML and CMOS in paragraph 4.2.3.

4.2.1 Architecture

In this paragraph we describe the architecture of the Reconfigurable Systolic Array.

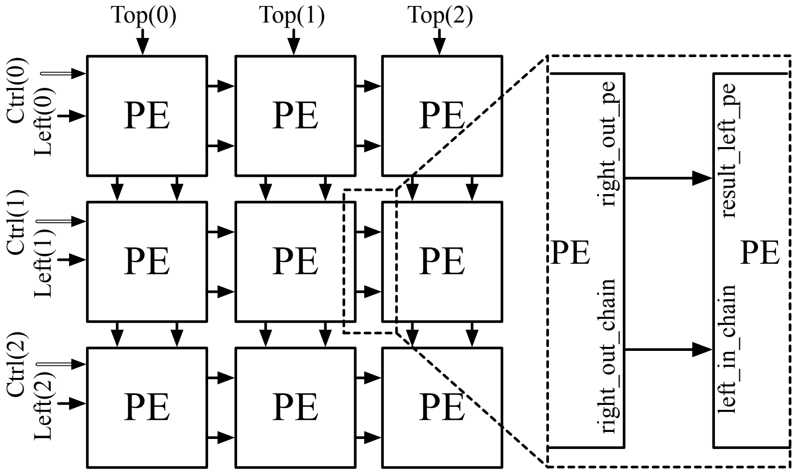


Figure 4.2. Reconfigurable Systolic Array: this is composed by a square array of Reconfigurable Processing Elements (RPE). Controlling signals are given from the left boundary (**Ctrl**). It is also shown the interface between two PEs that is composed of two signals: one is the result of one PE that is given to the successive PE, while the other is the signals that carry input values from the external (**chain** signal).

The Reconfigurable Systolic Array is composed by a square array of Reconfigurable Processing Elements (RPEs), as shown in Figure 4.2. The interface between PEs is composed by two signals: **right_out_chain** that carries the value coming from the external inputs through all the PEs and **right_out_pe** that is the result of one PE transmitted to the successive PE.

The structure of an RPE is shown in Figure 4.3. The RPE is composed of a Reconfigurable ALU, registers, multiplexers and a CTRL block. The latter is

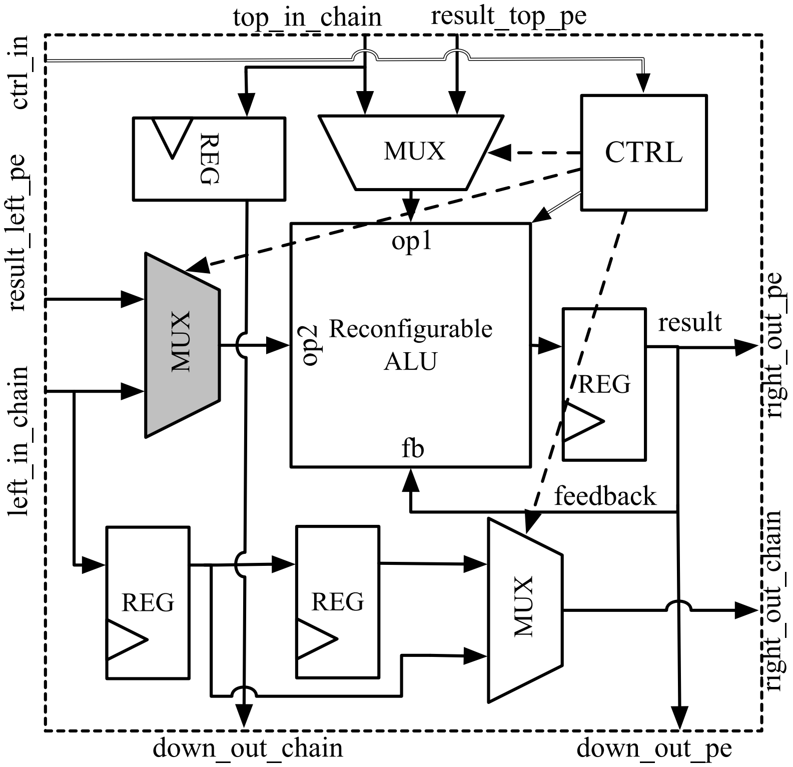


Figure 4.3. Reconfigurable Processing Element (RPE): CTRL block redirects the **ctrl_in** signal to all the configurable elements of the RPE, i.e. the 2 input multiplexers, the horizontal chain multiplexer, and the Reconfigurable ALU. The transmission of **ctrl_in** signal to RPE below is not represented for sake of simplicity. [61]

used to redirect the **ctrl_in** signal to multiplexers and to the Reconfigurable ALU. **ctrl_in** signal defines the behavior of each PE, i.e. its configuration, and it is transmitted locally from left to right in each PE (this transmission is not represented in Figure 4.3). Each RPE has 4 input signals: **top_in_chain** is a signal provided from the boundaries of the SA and transmitted with a direct propagation, through each PE, via a register to **down_out_chain**. The same solution is implemented for **left_in_chain** signal, but in this case the propagation can occur through 1 or 2 registers depending on the control signal of the multiplexer. Finally the other two inputs, **result_top_pe** and **result_left_pe** transmit the values computed in the above and left PE respectively (Figure 4.2).

Input multiplexers are used to choose between the two inputs for each side: the value evaluated in previous PE (for example **result_top_pe** for inputs from above) and the one that arrives from the outside (**top_in_chain** for inputs from above).

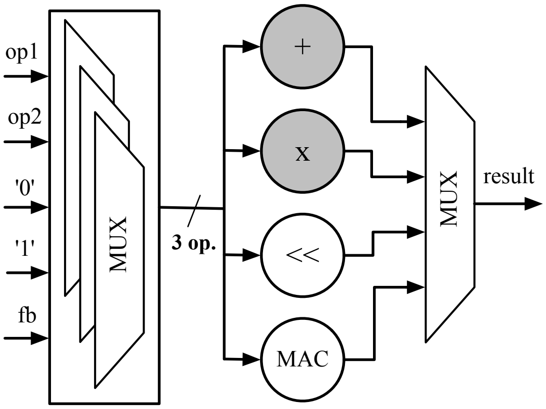


Figure 4.4. Reconfigurable ALU inside Reconfigurable Processing Element. The set of operations to implement can be chosen during design phase to allow more or less flexibility depending on the area and power available.

The Reconfigurable ALU can implement a given set of operations, always working on 3 input data: they can be chosen among input data coming from multiplexers, stored values ‘0’ and ‘1’, and feedback signal shown in Figure 4.3. In our minimum proposal, the Reconfigurable ALU can execute addition, multiplication, Multiply and ACcumulate (MAC), and logic left shifting (Figure 4.4). The adder is simply implemented as a classical Ripple Carry Adder, while the multiplier is an Array Multiplier composed of AND gates to perform multiplications and Ripple Carry Adders to sum partial products. The MAC is actually a multiply and add structure that can use the **fb** (feedback) signal as input to implement a MAC. Depending on the available area, the designer can decide to enhance the ALU providing hardware for other arithmetic or logic functions. Normally this can also depend on the typology of algorithms that the designer is willing to address with this reconfigurable architecture.

Each RPE can be programmed independently from the others. In this way a given PE in a custom SA that implements several operations can be mapped to a set of RPEs in the RSA. RPEs are programmed sending a set of **Ctrl** signals to the first column of the array (in Figure 4.3 local transmission from one PE to the successive is not represented). Each PE contains a configuration register that stores the **Ctrl** signal. One bit of **Ctrl** signal is used to select between programming and normal operation mode (it represents the Write Enable of the configuration register). The programming/preloading phase is further described in next paragraph.

4.2.2 Preloading Phase

Preloading Phase is basically composed of two activities: 1)preload necessary data inside the registers of the PE and 2) configure each PE to execute the requested operation on selected inputs. The two activities can run in parallel as they use different buffers.

The preloading is done using the chain signals. These carry the value to preload and a dedicated bit of the **Ctrl** signal is used to select the correct Processing Element. N clock cycles are necessary, being N the number of Processing Elements in a row of the array. Even though this is a time-consuming operation, it can be done in parallel with the downloading of data or the final turn of computation. In this way the overhead for each new preloading after the first one is of 1 clock cycle only.

Let us focus now on the configuration of the PEs. We have analyzed two possible solutions for the configuration. In the first solutions, all PEs of one row are programmed in the same way. In this case it is sufficient to send the programming word one time from the left boundary. This will be transmitted trough the several PEs and will configure each of them. In this case the “config” bit of **Ctrl** signal is enough to manage the configuration. We will use this configuration for the successive example of Matrix Multiplication. In the second solution, instead, it is possible to configure each PE independently. This option gives more granularity but at the same time requires more hardware to be implemented. The solution we have thought has been called “Four-in-a-row” method, from the name of the famous game. Basically The configuration word shall be sent considering that they are configured right to left in one row. When the first configuration word reaches the last PE (PE N), one bit inside the PE is set (called “configured” bit). This bit is transmitted to the previous PE. At successive clock cycle the PE $N-1$ receives the configuration word and at the same time the “configured” bit from PE N. In this way PE $N-1$ knows that it is its turn to be configured. This mechanism continues till the first PE and requires N clock cycles. The implementation of this mechanism basically requires one additional register in each PE and some simple logic to manage these registers. The “configured” bit of the successive PE is used as enable of register in one PE, that takes the value 1 as default. The reset phase is used to clear these registers.

4.2.3 Results in CMOS and NML

In this paragraph we describe the results that can be obtained with the Reconfigurable Systolic Array, synthesized for CMOS, classic NML with magnetic clock and MagnetoElastic NML (ME-NML). In this thesis, this is the first technological comparison presented, although many others will be proposed in Chapter 6.

In CMOS, the RSA was synthesized using Synopsys Design Compiler and a 28nm low power commercial library. This is near to the state of the art for CMOS technology so we can assume that these results are quite similar to a real implementation.

The RSA was then mapped on NML technology using ToPoliNano, a tool developed by Politecnico di Torino and able to design NML logic circuits starting from HDL description. At the time of writing the tool is not yet able to handle sequential circuits and tackle hierarchical floorplanning. Both functions are under development. So we have automatically generated the layout of all main blocks with ToPoliNano, while the floorplan is designed assembling these main blocks in a custom way.

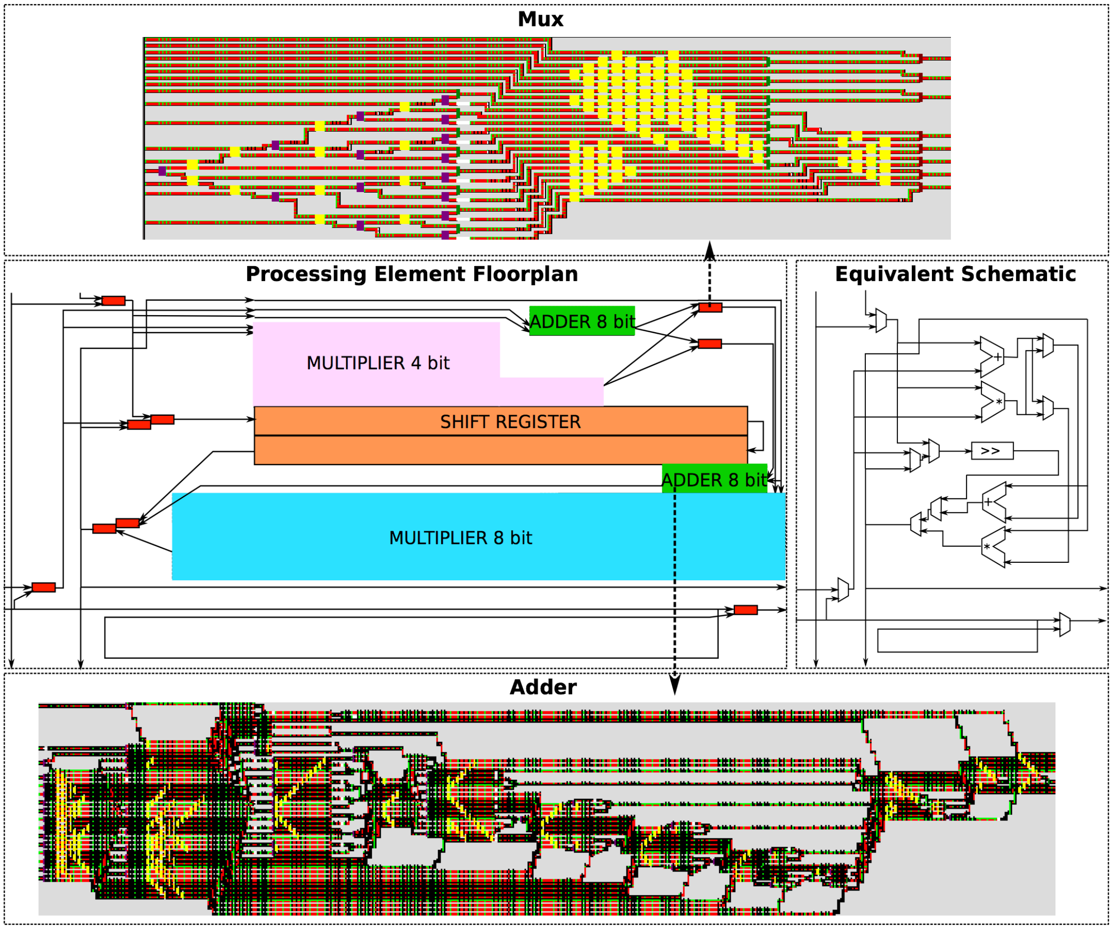


Figure 4.5. Processing element floorplan and layout. The general floorplan is shown in the central figure, while on the right the equivalent circuit can be seen. On top and on the bottom two circuits obtained by ToPoliNano are reported: A 2to1 8 bits multiplexer (top) and an 8 bits adder (bottom). [61]

Figure 4.5 shows the general floorplan of the processing element, on the left the equivalent schematic. We have reviewed the design in order to optimize it for NML. The processing element can be divided formally into two stages. The first stage is composed of an 8 bits adder and an 8 bits multiplier. The two outputs are connected through a multiplexer , whose selection is transmitted to the second stage. The second stage comprises an 8 bits adder and a 16 bits multiplier. This particular organization was chosen to better exploit the technology characteristics. Depending on how the multiplexers are configured we can obtain the three operations required by the programmable processing element (3-operands sum, 3-operands multiplication and multiply-and-accumulate) and an additional operation (sum and multiplication) that expands the capabilities of the SA. A shift register is also required to complete the logic functionality of the processing element. Figure 4.5 (top and bottom) shows an example of two blocks obtained using ToPoliNano, a 2to1 8 bits multiplexer in Figure 4.5 (top) and an 8 bits adder in Figure 4.5 (bottom).

The floorplan organization has a particular U-shape, where the second stage is bent under the first one. In this way input signals are from the top-left side and output signals are connected to the bottom-right side. This solution was chosen considering the matrix-like structure of the SA, so that input and output signals among neighbor processing elements are perfectly matched.

ToPoliNano at the time of writing is not able to synthesize ME-NML circuits. Till now the design of these circuits has been done manually. We have indeed designed all the logic blocks of a PE and evaluated the interconnections overhead to have an estimation of the total power and occupied area. Perhaps with an automatic synthesizer it would be possible to have a more compact and efficient circuit, but this rough estimation is sufficient to appreciate the performance of this technology.

Some of the main blocks present in the Reconfigurable Systolic Array are presented in Chapter 6.

Table 4.1. Synthesis results of the Reconfigurable Systolic Array

Technology	Frequency (MHz)	Power (μW)	Area (μm^2)	Figure Of Merit (nJ)
CMOS 28nm	1200	430.00	854.52	358.33
NML	100	506.90	5167.92	506.90
ME-NML	100	32.67	361.94	326.66

Table 4.1 resumes the synthesis results. Frequency is fixed and due to technological constraints for NML and ME-NML. It is limited by the time necessary to reset magnets and their successive switching. According to the analysis in [16], the clock frequency can be set to 100 MHz to guarantee a proper functioning of the circuit.

From the CMOS synthesis we obtain a maximum clock frequency of 1.2 GHz.

ME-NML has the best performances in terms of power consumption. The RSA implemented with this technology consumes indeed 13 times less than the equivalent CMOS circuit. NML instead cannot compete with the other two technologies, resulting in an higher value of power consumption.

ME-NML is also the best solution in terms of area occupation, since it needs less than half the area occupied by CMOS circuit. Also in this case NML results are the worst. This is due to the limited maturity of this technology: indeed classical NML has a planar layout and a rigorous propagation direction due to clock mechanism that leads to this high area occupation. In the case of ME-NML, instead, since signals have not a fixed propagation direction, it is possible to achieve extremely compact layouts and values of area occupation sensibly lower. Several technological improvements are under study to reduce NML area occupation and power dissipation: Out-of-Plane NML [62], 3D circuits with Logic-In-Memory approach [63], NML circuits with Domain-Walls [64].

Finally to better compare ME-NML and CMOS we have also computed the *Figure Of Merit* of a digital circuit, that is given by the Speed-Power product, defined as the product of propagation delay (in ns) and power dissipation (in mW) and is measured in pico joules. ME-NML is finally the best solution (lowest value) because its low power consumption greatly compensates the lower clock frequency.

4.3 Algorithms

In previous Section we have presented the architecture of the Reconfigurable Systolic Array and we have shown that it can be implemented in ME-NML achieving also advantages in terms of area occupation and power dissipation with respect to a CMOS implementation. In this Section we focus instead on the reconfigurability property, showing some of the algorithms that can be mapped in this Reconfigurable Array. From this discussion it should be evident how this architecture gives extremely high flexibility with respect to an algorithm dependent Systolic Array.

We will describe the mapping of four different algorithms: Matrix Multiplication in paragraph 4.3.1, Discrete Cosine Transform (DCT) in paragraph 4.3.2, FIR Filters in paragraph 4.3.3 and IIR Filters in paragraph 4.3.4. Finally, we will also describe our preliminary implementation of a tool, called “RSA Configurator” that is able to generate the necessary inputs to configure the Reconfigurable Systolic Array in the desired way (paragraph 4.3.5).

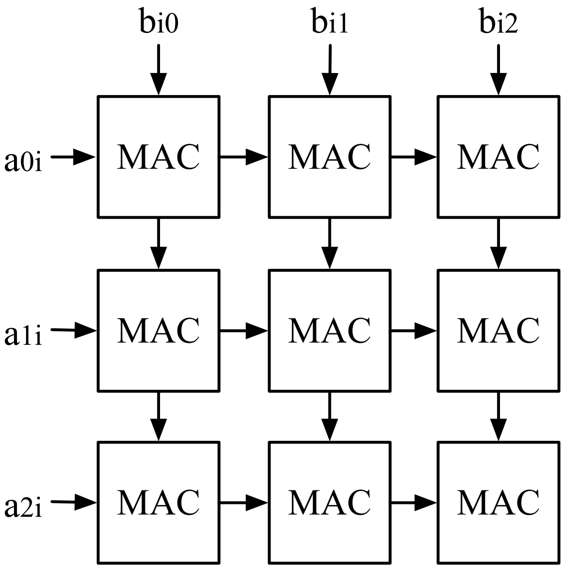


Figure 4.6. Matrix Multiplication mapped onto the Reconfigurable Systolic Array. Each PE is configured as a Multiply and Accumulate (MAC) block.

4.3.1 Matrix Multiplication

Given two rectangular matrices $A = (a_{ik})$ and $B = (b_{kj})$ of order $N_1 \times N_3$ and $N_3 \times N_2$ respectively, their product, matrix $C = A \times B$, $C = (c_{ij})$, of order $N_1 \times N_2$, can be obtained according to the equation (4.1):

$$c_{ij} = \sum_{k=1}^{N_3} a_{ik} \cdot b_{kj}, \quad i = 1, 2, \ldots, N_1 \quad j = 1, 2, \ldots, N_2 \tag{4.1}$$

This equation can be mapped to a SA of $N_1 \times N_2$ cells, each performing Multiply and Accumulate operations to store partial results $c_{ij}(k)$ at each iteration k (Figure 4.6) [33]. This is the WIL implementation of matrix multiplication, where an internal loop is necessary. To map this solution into the RSA, it is necessary to configure the array in this way: the Reconfigurable ALU shall be configured to use the MAC resource, with **op1** and **op2** as inputs of the multipliers and **fb** signal as second input for the adder. The other multiplexers must select **top_in_chain** and **left_in_chain** signals as input data, and one single register in the left-to-right transmission of **left_in_chain** signal.

4.3.2 Discrete Cosine Transform (DCT)

Discrete Cosine Transform (DCT) algorithm works in the following way: a sequence of input data x_n $(n = 1, \ldots, N)$ is translated in the sum of cosine at different frequency. The output is a sequence X_k $(k = 1, \ldots, N)$. DCT is similar to discrete Fourier transform, but while in DFT coefficients can have an imaginary part, in DCT they have only the real part. There are different types of DCTs; in this case we refer to the one used in JPEG compression that can be expressed by equation 4.2:

$$X_k = w(k) \sum_{n=1}^{N} x_n \cos\left[\frac{\pi}{2N}(2n-1)(k-1)\right], \qquad k = 1, \ldots, N \tag{4.2}$$

where $w(k)$ is computed as in equation 4.3.

$$w(k) = \begin{cases} 1/\sqrt{N} & \mathrm{k = 1} \\ \\ \sqrt{2/N} & 2 \leq \mathrm{k} \leq \mathrm{N} \end{cases} \tag{4.3}$$

Our objective is to map this algorithm in a similar way of matrix multiplication. Our following analysis is therefore aimed at mapping this algorithm to a square SA to perform matrix multiplication. In this case the mapping will be to a WOIL Systolic Array for matrix multiplication, differently from the previous mapping. The indexes of the array are n and k. Given a couple (n, k) that will identify one PE of the SA, the value of $\cos[\pi/2N(2n-1)(k-1)]$ is known and constant. Therefore the cosine values can be preloaded in the PEs, stored in **internal register** and maintained for the entire computation. It is then possible to map the algorithm as shown in Figure 4.7: each X_k is computed by one column of the array, while inputs x_n are provided from the left boundary. Each cell performs a MAC operation: signal **left_in_chain** is multiplied by signal **In_reg** (value stored in the internal register) and then added to the signal **result_top_pe**. The result is then sent to the **down_out_pe**, while **left_in_chain** signal is propagated to **right_out_chain**. Finally, it is necessary to consider the multiplicative factor $w(k)$. One additional row of the matrix (at the bottom) is used to multiply the signal **result_top_pe** value by the $w(k)$ factor in order to have the correct result. This SA uses therefore two different configurations: Multiplier for the last row, MAC for all the other cells.

4.3.3 FIR Filters

The following explanation refers to Figure 4.8(a) to describe the logic organization of FIR filters, to Figure 4.8(b) to present the implementation in the reconfigurable

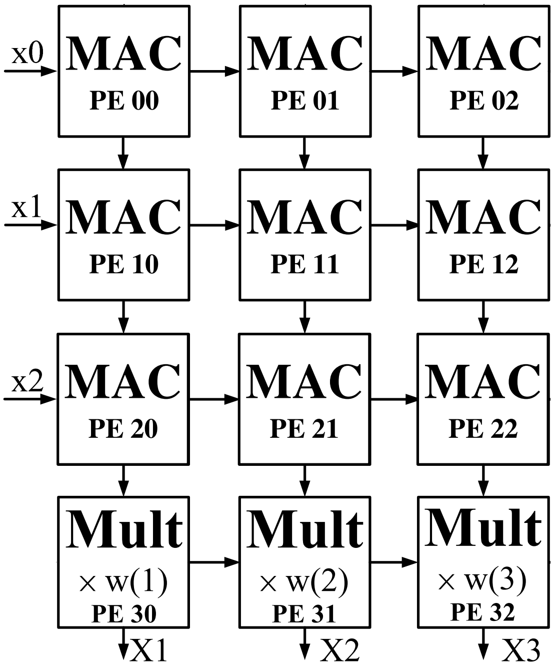


Figure 4.7. Mapping of the Discrete Cosine Transform (DCT) onto the Reconfigurable Systolic Array. PEs are configured in two different ways: as Multiplier for the last row and as MAC for the others.

architecture and to Fig. 4.9 to discus the simulations.

Given a discrete-time FIR filter (Figure 4.8(a)), the output sequence $y[n]$ can be expressed in terms of input sequence $x[n]$ and weights b_j with equation (4.4):

$$y[n] = \sum_{j=0}^{N} b_j \cdot x[n-j] \tag{4.4}$$

where N is the filter order.

Looking at Figure 4.8(a), it is evident that, to map FIR filters in the RSA, cells must be programmed in three different ways (with a granularity that exceeds one configuration per row): this is possible since we can manage one configuration signal for each PE. In Figure 4.8(b), two rows of the Reconfigurable Array are used to implement a FIR filter. Other FIR filters can be mapped in other rows of the RSA. They must all share the same weights b_j since these must be provided from the external and are locally transmitted to PEs below (through each column). The three following configurations must be used: MUL, ADD and top-to-right signal transmission (the bottom-left PE in Figure 4.8(b)), hereinafter called TRANSMIT. In the following each of these configuration is described.

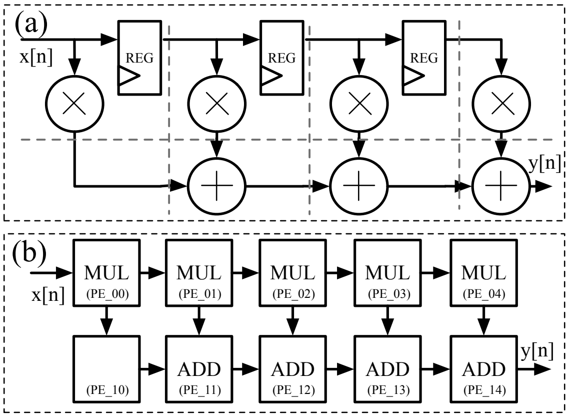


Figure 4.8. Mapping of one FIR Filter onto the Reconfigurable Systolic Array. PEs are configured in three different ways: as Multiplier for the first row, as Adders for the second row except for the leftmost cell that is configured as "Transmit" cell.

1. MUL cells are configured to execute multiplications on operands provided from top and left, therefore the Reconfigurable ALU is programmed to use **op1**, **op2** and '1' as operands to be multiplied. The other multiplexers select **top_in_chain** and **left_in_chain**. Notice that in this case the left-to-right transmission must follow the path with two registers, to have one clock delay difference with respect to the horizontal path between Adders, where one register only (the one that stores the result) is present. This can be evidenced from Figure 4.8(a), where each dashed line represents a cut-set for retiming. According to this analysis, additional registers must be inserted in each left-to-right signal transmission and in each path from multipliers to adders. Configuring word for MUL cells is shown Figure 4.9(a).

2. ADD cells are configured to execute addition on input operands from top and left, therefore the Reconfigurable ALU is programmed to use **op1**, **op2** and '0' as operands to be added. The other multiplexers select **result_top_pe** and **result_left_pe**. In this case the delay for the partial result to be transmitted to neighbor PE is always 1 clock cycle as expected. Configuring word for ADD cells is shown Figure 4.9(c).

3. TRANSMIT: **result_top_pe** must be transmitted as result to the PE at its right. This can be done configuring the PE to execute an addition with operands **result_top_pe** (**op1** inside the Reconfigurable ALU), '0' and '0'. So the chosen solution is a fake addition which does not add any information. Of course it could have also been configured as multiplier with '1' and '1' operands. Configuring word for TRANSMIT cell is shown Figure 4.9(b).

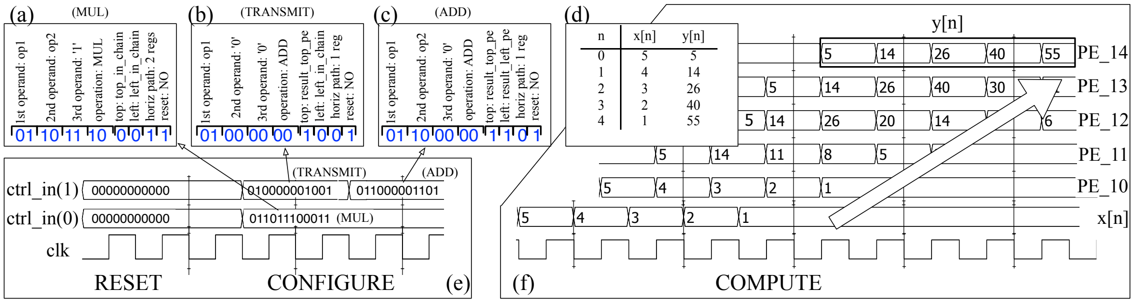


Figure 4.9. Simulation of the RSA configured to implement a FIR Filter. (a) Configuration word for MUL cells. (b) Configuration word for TRANSMIT cells. (c) Configuration word for ADD cells. (d) FIR filters input values and correspondent results. (e) RESET and CONFIGURE waveforms. (f) COMPUTE waveforms that represent inputs ($x[n]$) and results of PEs in the bottom row (PE_1x). [61]

Figure 4.9 summarizes the example of FIR filter implementation in the RSA. Three phases are necessary: reset, to clear registers; configure, to program each PE to execute a given function (Figure 4.9(e)); compute, when the array is actually used for its scope, in this case FIR filtering (Figure 4.9(f)).

4.3.4 IIR Filters

The last example of algorithm mapping onto the RSA is Infinite Impulse Response (IIR) filters. Generally an IIR filter can be expressed by equation 4.5:

$$y(t) = \underbrace{\sum_{i=0}^{N} c_i\, x(t-i)}_{A(t)} - \underbrace{\sum_{j=0}^{M} b_j\, y(t-j)}_{B(t)} \tag{4.5}$$

In equation 4.5 N is the feedforward filter order, c_i are the feedforward filter coefficients, M is the feedback filter order and b_j are the feedback filter coefficients. To ease the description we will refer to $A(t)$ for the feedforward part of the equation, and to $B(t)$ for the feedback one, as shown by brackets below equation 4.5 . It is important to notice that this kind of filter has memory, meaning that outputs $y(t)$ depend not only on input values, but also on previous outputs $y(t-j)$. To implement it, external feedback loops are necessary to provide outputs back in the array to be used for computation of next steps.

To compute the result $y(t)$ at a given step t it is necessary to perform two sum of products as expressed in equation 4.5. This can be done in the same way of the

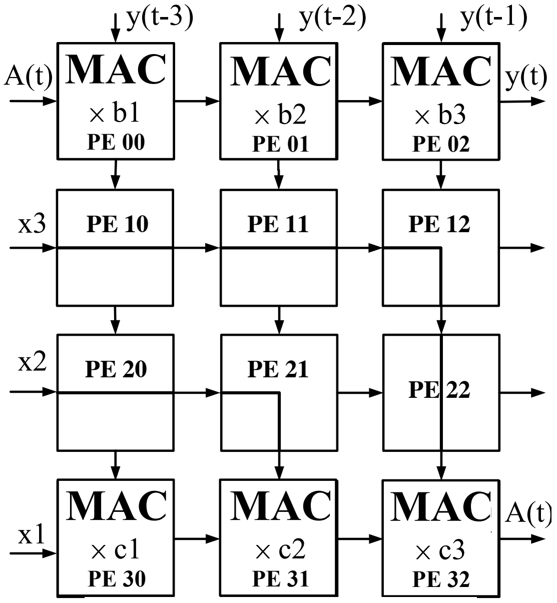


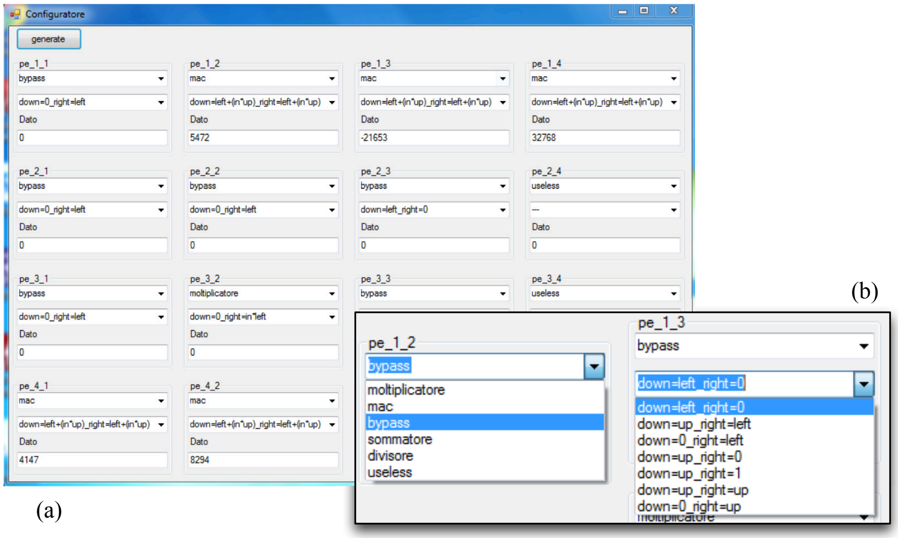
Figure 4.10. Infinite Impulse Response (IIR) filter configuration. PEs are configured in 4 different ways: MAC, Bypass left to right, Bypass left to down (PE 12 and PE 21) and Bypass up to down (PE 22).

DCT mapping, using a row or column configured as MAC. In this case the mapping is reported in Figure 4.10: the first row is used to sum the weighted outputs, i.e. compute $B(t)$. In each of these PE, signal **top_in_chain** is multiplied by signal **In_reg** and then summed to signal **result_left_pe**. The rightmost PE of the first row produces the result $y(t)$. Externally this is fed back to the other PEs of the first row (external loops not shown in Figure 4.10). These external feedbacks give memory capability to the circuit. The same configuration is used in the last row to execute the other part of computation, i.e. $A(t)$. The other rows in the middle are used to synchronize the inputs: a new data is given every clock cycle as input to all the leftmost PEs and then the structure handles the timing requirements, i.e. provide x signals to the bottom row in the right manner. All the elements of this type are BYPASS cell. Three different cases are present in the structure: a bypass from left to right, a *corner bypass* that route the data from the left to the bottom of the PE and a vertical one. Finally, to execute the subtraction between the two sum-of-products of equation 4.5, the output of the last row (bottom-right PE), indicated as $A(t)$ is fed back to the top-left PE. The PEs are hence configured in four different ways: MAC, Bypass left to right, Bypass left to down and Bypass top to down.

We have seen four algorithms mapped into the RSA with different complexity: Matrix Multiplication had cells all configured in the same way; to design DCT, it was necessary to configure cells in two different way; for FIR filters, three different kind of cells were necessary; finally for IIR filters we counted four different configurations.

Several other algorithms have been mapped to the RSA: Kalman Filters, Discrete Fourier Transform (DFT), Finite Difference Derivative are some of the examples. Others will be mapped in the future depending on the applications to implement for our purposes. With this description it this evident the rate of flexibility that can be achieved by this architecture, that represents an extremely important step ahead with respect to original, algorithm-dependent, Systolic Arrays.

4.3.5 RSA Configurator



pe_1_1: bypass; down=0_right=left; Dato 0
pe_1_2: mac; down=left+(in*up)_right=left+(in*up); Dato 5472
pe_1_3: mac; down=left+(in*up)_right=left+(in*up); Dato -21653
pe_1_4: mac; down=left+(in*up)_right=left+(in*up); Dato 32768

pe_2_1: bypass; down=0_right=left; Dato 0
pe_2_2: bypass; down=0_right=left; Dato 0
pe_2_3: bypass; down=left_right=0; Dato 0
pe_2_4: useless; ---; Dato 0

pe_3_1: bypass; down=0_right=left; Dato 0
pe_3_2: moltiplicatore; down=0_right=in*left; Dato 0
pe_3_3: bypass
pe_3_4: useless

pe_4_1: mac; down=left+(in*up)_right=left+(in*up); Dato 4147
pe_4_2: mac; down=left+(in*up)_right=left+(in*up); Dato 8294

pe_1_2: bypass
pe_1_3: bypass; down=left_right=0

Figure 4.11. Reconfigurable Systolic Array Configurator: this tool, runnable in Windows, is able to automatically generate configuration words depending on the chosen configuration of each PE in the Array.

Finally in this paragraph we present an useful tool that we have designed to automatically generate configuration words for the Reconfigurable Systolic Array. This is called “RSA Configurator” and its graphical interface is shown in Figure 4.11.

The tool generates a **.sac** file that is automatically read by the test bench and used in simulation of the RSA. The RSA Configurator starts with an input block in which the number of PEs present in a row/column shall be inserted by the user. It is assumed that the array is a square one. Then the interface shown in Figure 4.11(a) is opened. For each PE it is possible to select the operation that it must execute.

Depending on the chosen operation, several configurations are possible and can be chosen in the successive drop down menu. For example in Figure 4.11(b) are shown the several configurations that can be selected for a “bypass” (or transmit) cell. Moreover to each PE it is possible to assign a value to be stored inside the internal register for computations.

While this is only a simple tool, useful to speed-up the testing phase of the Reconfigurable Array, it could be part of a more complex and useful environment. In the future it could be possible to insert the configurator inside a tool that, starting from the selected algorithm to implement, automatically generates the whole configuration of the RSA. In this way all the mapping could be automatic and the effort that is now allocated to the designer could be assigned to the machine.

4.4 Final Remarks

Once that we have identified Systolic Arrays as ideal structure for NanoMagnet Logic (NML), our attention has focused on how to make this architecture interesting for a real implementation. The proposed path aims at making Systolic Arrays more flexible, by providing them with a reconfigurability property that allows to use the same architecture to execute several algorithms. While this idea has been centered on the usage of Systolic Arrays for NML, it can be exploited also with other technologies, like CMOS.

The Reconfigurable Systolic Array (RSA) that we propose has the adaptability of an FPGA (or similar) but it guarantees the high operating frequency of Systolic Arrays. It is an architecture based on a square array of Reconfigurable Processing Elements, where each of them can be programmed in a different way to execute different tasks and use different input data.

Several achievements can be mentioned in this design path:

- Definition of the Reconfigurable Systolic Array architecture: it has been defined this flexible, yet extremely powerful, architecture, that is based on multiplexers to select data and operations to execute. The architecture can be enriched by designers simply adding further resources if space and power available allow it.

- Technology comparison based on RSA: this architecture is the first example of benchmark that we have used to analyze the different performance of the three technologies, i.e. CMOS, classic NML with magnetic clock and ME-NML. Results show that ME-NML, even if it can work at lower clock frequencies, has overall better performance than CMOS.

- Algorithm mapping examples: the RSA has been tested on several algorithms.

We have shown as examples the Matrix Multiplication, Discrete Cosine Transform, FIR Filters and IIR Filters. This is fundamental to distinguish our approach from other architectures proposed in literature that address only a specific problem' scale with reduced reconfigurability option.

- RSA Configurator: finally we have designed a tool to make the configuration of this Reconfigurable Array as simple as possible. This could be in the future one of the elements of an integrated RSA-designer that could autonomously configure an RSA starting from an algorithm description.

The Reconfigurable Systolic Array is another step forward in the design of architectures for NML. With this solution, we have separated the algorithm/application domain from the design one. In this way the design is more effective and reusable.

Chapter 5

Logic-In-Memory

We have deeply analyzed the adoption of Systolic Arrays for NanoMagnet Logic: in Chapter 3 we have introduced this architecture and we have shown several improvements that can be put in place to overcome some of NML drawbacks. In Chapter 4 we have instead addressed one of the main limitations of classical Systolic Arrays, i.e. the algorithm dependency, introducing our Reconfigurable Systolic Array that can be programmed to execute several algorithms.

This research path is instead aimed at addressing another limitation of Systolic Arrays: memory availability inside the processor. Systolic Arrays that we have analyzed till now have little to no memory available inside the PE. The Reconfigurable Systolic Array contains one memory cell to store data and one memory cell to store results. However big memory banks will be placed outside the SA.

With Systolic Array architecture (as well as with newer Von Neumann paradigm architectures), the communication between processor and memory becomes the bottleneck of the system. Indeed, the bus connecting the two elements cannot feed data in the processor at the necessary bandwidth to exploit at the maximum the computational power of the array.

For this reason in this research path we have focused on a new processor paradigm, called Logic-In-Memory (LIM). In this architecture, Logic elements and Memory ones are merged in one single device. In this way bus communication does not exist anymore, and the complexity is shifted to an algorithm level: processing elements should have data to use stored in their memory element or in memories of near PEs. So it is necessary to define a communication protocol between PEs to exchange data and memory structure should be provided with a sort of "intelligence" to have the right data in the right place and at the right moment.

We introduce the main concept behind Logic-In-Memory in Section 5.1. Then we present two different kind of architectures: the first, described in Section 5.2, is a first version of LIM architecture with a one-to-one mapping between memory blocks and logic; the second, described in Section 5.3, is instead a more complex version with a pyramidal pipelined memory. The results achievable with these solutions are shown in Section 5.4, while a final discussion is presented in Section 5.5.

5.1 Concept

In this Section we introduce the concept of Logic-In-Memory (LIM). We start analyzing the main limit of common Von Neumann Architecture, that rely on separated logic and memory elements, in paragraph 5.1.1. Then we analyze other parallel architectures existing in literature, with the aim to give a framework in which it is possible to introduce LIM structure, showing advantages and differences with respect to all the other existing architectures. This analysis is provided in paragraph 5.1.2. Finally we introduce improvements that can be achieved with the LIM in paragraph 5.1.3.

5.1.1 Limit of Von-Neumann Architecture

Von Neumann architecture represents the backbone of modern computational systems. Indeed, the majority of the existing electronics devices are based on the Von Neumann model [65]. The basic idea behind this architecture is to have a computational unit connected with an interconnection bus to a memory, where program instructions and data are memorized. The success of this idea lies probably in its simplicity. This is a concept that has served well the purpose of building electronic computational systems in the last 70 years. However, the huge advancements in MOSFET technology, mainly due to transistors scaling [46][66], have highlighted the main limitation of this approach: Data communication is the bottleneck of this system and it limits the performance of the processing unit.

Using ultra-scaled MOSFET transistors it is now possible to create microchips with billions of transistors working at a frequency of several GigaHertz. The full computational power of a processing unit cannot however be exploited due to the limited bandwidth of interface between the processing unit and the memory. This is due on the one hand to the limited number of physical connections on a microchip and on the other hand to the frequency limits on signals traveling on communication buses. This problem is attenuated in modern computational system by exploiting memory hierarchy and by embedding part of the memory on-chip [67]. But, since this is an issue intrinsic to the model, the memory bottleneck problem cannot be completely solved adopting these techniques. Moreover, given that many computational systems exploit parallel computing to improve performances, this problem is exacerbated.

To address this issue, a new computing paradigm must be developed. Logic-In-Memory is aimed at solving this bottleneck creating a system where logic and memory are embedded together. From a theoretical point of view, in this way the memory bottleneck problem is completely solved. We will analyze in next Section in further details what means from a practical point of view "to embed memory and logic in the same device" and what kind of applications benefit most from Logic-In-Memory concept.

5.1.2 Other Parallel Architectures

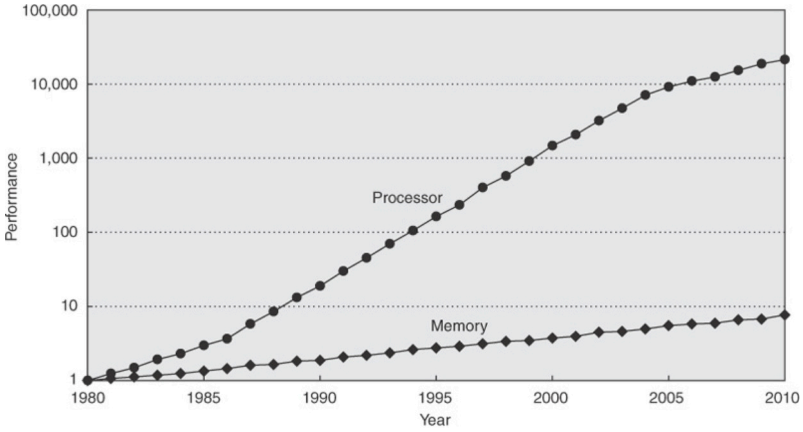


Figure 5.1. Graph showing how the gap in terms of performance between logic and memory increases through the years.

Before analyzing in detail the Logic-In-Memory principle, it is useful to analyze other parallel architectures. The objective is to give evidence of the strengths of parallel architectures and at the same time highlight the limitations due to bus communication between memory and logic.

Different types of processing units are available nowadays. Among them, the processing units that mostly benefit of the philosophy behind the design of modern computational systems are probably superscalar microprocessors [68]. In this type of processors an execution unit, coordinated by a control unit, executes software instructions on data stored on a dedicated memory. The concept behind microprocessors is indeed very simple, but their architecture has evolved greatly throughout the years to cope with the advancement of technology. Transistors scaling allows to pack an always increasing number of transistor on a chip and higher clock frequencies can be reached. One of the evolution that mostly affected microprocessors architecture is the memory structure. Microprocessors can elaborate huge amount of instructions, but extremely fast and big memories are required. Since this result cannot be physically achieved, a caching mechanism coupled with a hierarchical memory structure was introduced. But also applying these solutions memory cannot keep up with the speed of computational units. This problem is known as “the memory wall” problem [69][70], and is schematically represented in Figure 5.1. The performance gap between processing units and memories with their communication buses is steadily increasing with the CMOS technology advancements [71]. As a consequence processing systems cannot exploit their full potential.

The bandwidth limitations of communication buses is nowadays one of the major problems of processing units. The reason lies in the diffusion of multicore structures, circuits where several units working in parallel are used to enhance performance reducing at the same time power consumption. More processing units means however more data that can be elaborated and must be fetched from memory. An example of this problem are graphical processing units (GPUs) [72] performing texturing operation. Texturing is an operation performed by GPUs where bitmap images are attached to surfaces of object to generate realistic 3D images. The main problem of texturing process is that a huge amount of high resolution bitmap images must be loaded from memory. As a consequence the texturing operation considerably slows down GPUs computation [73].

Traditional memories are getting faster and smaller to cope with parallel architectures. The recent introduction of 3D structures is an answer to the demand of faster and more capable memories. The study and development of new memory structures, such like 3D memories [74][75] and magnetic RAMs [76][77][78], is also an attempt to reduce the performance between logic and memory. Parallel circuits are an ideal ground to develop new kind of architectures based on a completely different structure.

Two architectures have been the major source of inspiration that we followed in the development of our Logic-In-Memory circuit: 1) Systolic Arrays, that we have analyzed in previous Chapters and that we resume in the following; 2) GPUs whose particular memory structure is interesting for our purposes.

As already thoroughly described, a Systolic Array [33][50] is composed by several processing elements working in parallel. Processing elements are generally small and execute a single operation (multiplication, subtraction, etc..). Each processing element receives data from neighbor elements or from the outside. Output signal are sent to neighbor processing elements or to outside. Each processing element can contain few registers to temporary store the result of the operation depending on the type of systolic array [51][43]. Few key elements distinguish therefore Systolic Arrays:

1. Processing elements work in parallel and generally perform the same operation [79].

2. Processing elements are small and therefore there is a huge number of them [36].

3. Communication is local among processing elements easing the memory wall problem [34][35].

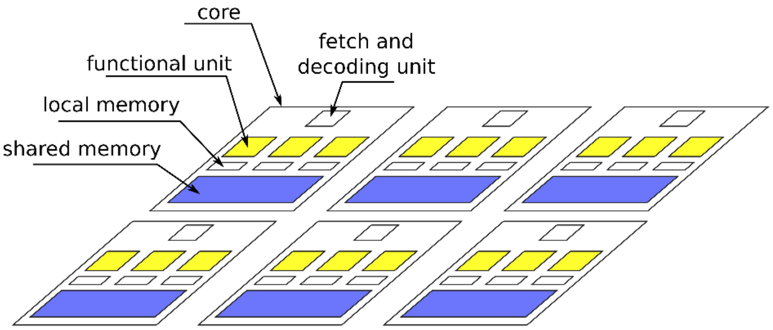


Figure 5.2. Representation of a GPU architecture, with local and shared memory.

Differently from Systolic Arrays, GPUs exploit the parallelism at two different levels. Referring to Figure 5.2, a GPU is made of a number of cores composed of different functional units working in parallel and having access to a local and a shared memory [80]. When a program is executed on a GPU, firstly each thread is associated with a single core, then, inside each core, an instruction is fetched and simultaneously executed in parallel by the functional units inside the core like in a SIMD (single instruction multiple data) structure [81][82]. In GPUs, functional units are complex and execute a huge number of mathematical and logical operation. In addition every unit has access to its own local memory and generally the only way to communicate with other units it to access to the shared memory. Figure 5.2 is only a simplified schematics. In current GPUs, functional units are similar to programmable processors, where several algorithms can be executed. The communication with memory is also much more complex than in a Systolic Array, due to the different types of memories available, local, shared and global. A correct handling of memory communication can have a huge impact on GPUs performance.

Current GPUs are very complex devices that are increasingly used as algorithm hardware accelerators. An algorithm can be defined as suitable to be computed on a GPU if it is massively parallel and if the number of mathematical operations to be executed is higher than the number of memory accesses. This concept is expressed by the arithmetic intensity that is calculated as the number of mathematical operations executed by a single functional unit divided by the number of memory accesses to load data.

$$\text{Arithmetic Intensity} = \frac{\#\text{arith. operations}}{\#\text{MEM access}} \tag{5.1}$$

The higher is the arithmetic intensity, the more suitable is an algorithm for a GPU.

This analysis gives us precious information regarding how to overcome the memory wall problem. GPUs and Systolic Arrays have been used as inspiration for the design phase of our Logic-In-Memory architectures.

5.1.3 Logic-In-Memory Improvements

We have analyzed the limitations of Von Neumann architecture and other parallel architectures inspiring the design of Logic-In-Memory. Now we enter into the details of this new architecture defining the main concepts and improvements that can be achieved.

The idea behind the Logic-In-Memory principle is relatively simple: Remove the separation between logic and memory and embed them in a single entity. However this simple principle rises an important question: How it is possible to translate it in a new and effective electronic circuit architecture? We believe that there are two key concepts that distinguish a Logic-In-Memory system.

1. **Memory Locality**. The first key concept implies the distribution of memory elements among the whole circuit, instead of having a big memory block interfaced with the logic core. Furthermore data communication should happen through local data exchange among local memories, avoiding access to external memories that compromise performance.

2. **Intelligent Memory**. The second key concept implies instead the necessity of providing memories with "intelligence", that means logic capabilities. Memories should be able to automatically provide data to logic processors, fetching data from neighbor memories and/or from outside.

Given these two key concepts it is clear why we have focused our attention on parallel architectures, like systolic arrays and GPUs. They intrinsically have local memories and local connections. They represent therefore the perfect basis upon building the Logic-In-Memory architecture. In Sections 5.2 and 5.3 we will describe how we have translated these two principles in our two implementations of the LIM principle.

Given the nature of the LIM architecture that we propose, not all algorithms are fitted for it. Particularly we have identified two requirements that an algorithm must have to best exploit the LIM architecture.

1. **High parallelism**. An algorithm should feature an high number of operations that can be executed in parallel.

2. **High percentage of local interactions among neighbor processing elements**. Similarly to what happens in systolic arrays, where input data are obtained from neighbor processing elements, and output data are provided to neighbor processing elements, the same concept shall be applied for Logic-In-Memory. While not strictly necessary (as we will see our architectures work also if data are fetched from outside), a high percentage of local data exchanges can maximize performance.

Using these requirements as a base, we have identified four classes of algorithm that can exploit the true potential of our LIM architecture: sorting algorithms, cryptography, mathematical problems and image processing [83][84][85]. One of them, presented in paragraph 5.4.1, is used for our results analysis and comparison.

5.2 LIM 1.0 Architecture

In this Section we present the first proposal for the implementation of our Logic-In-Memory architecture.

Logic-In-Memory is a structure divided into multiple cells operating concurrently, each one combining processing and storage capabilities.

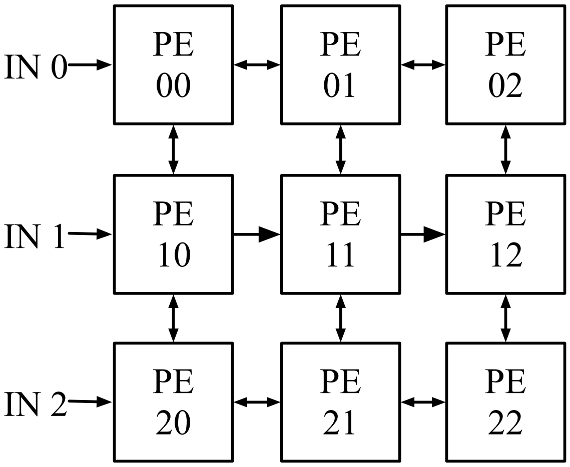


Figure 5.3. Logic-in-memory grid, with the input interface at the left border.

The Logic-In-Memory structure is a rectangular array (grid) of cells, as shown in Figure 5.3. Even though they operate independently each from the others, the interactions among them is a key feature of the system. Inter-cell communication is defined by fixed protocols, with operations (also referred as instructions or commands) coded in packets of a fixed length.

The grid access points are situated in the left border of the grid, with a number of input/output points equal to the number of rows of the grid (one access point for each row). The interfacing must follow the inter-cell communication protocols. The I/O requests must be expressed in read/write instructions coded in packets.

Till now the description is extremely similar to Systolic Arrays. What we describe hereinafter represents instead the important innovation of this architecture.

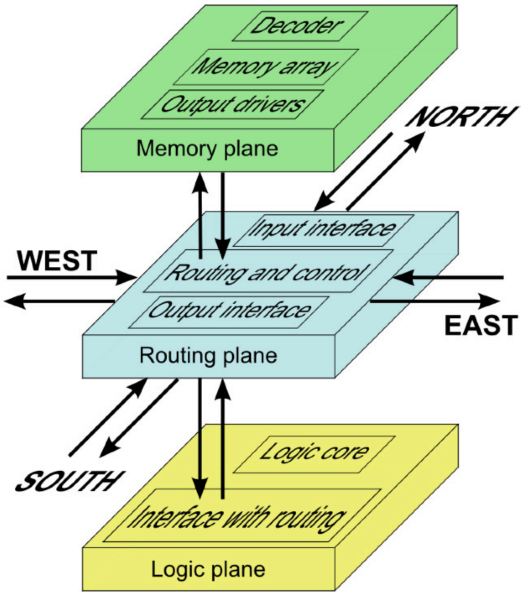


Figure 5.4. Logic-in-memory cell, composed of three layers (or planes): Memory, Routing and Logic.

The structure of a Logic-In-Memory cell is displayed in Figure 5.4. Each cell is internally divided into three planes: logic, routing and memory. The distinction is only at the architectural point of view, as they do not necessarily belong to three different physical layers. In general, physical mapping is technology- dependent. The center of the cell is the routing plane. It acts like a router, receiving and forwarding instructions. It is connected to the logic plane, to the memory plane and to the adjacent cells, as in Figure 5.4. It cannot start an operation on its own initiative, but it simply routes messages as instructed by the commands it receives. The memory plane is dedicated to storage. It is a rectangular array of words accessible by the routing plane only. Finally, the logic plane is dedicated to computation. It is the only "active" plane in the system. It performs arithmetic operations as well as memory accesses. The interaction with memory is not direct. A memory read/write request has to be encoded in a read or write operation, which is transmitted to the routing plane, which in turn accesses memory.

Logic-in-Memory is not simply a way to distribute computation in several parallel units. A key element is the possibility of interactions among adjacent cells. Cells can interact through memory read/write operations. The architecture privileges communication between near cells. Each cell is connected directly just to the adjacent cells (NORTH, SOUTH, EAST, WEST in Figure 5.4). Two distant cells can communicate sending packets through all the cells in between, with a delay proportional to the distance among them. Memory accesses are classified differently if logic (which requests the memory access) and memory (which receives the request) plane belong to the same cell or not. The fastest access is when the memory plane is in the same cell of the logic plane, not needing any inter-cell interaction. A slower, but still fast, access is to the adjacent cells, denominated toc-toc access. Finally, there is the possibility to access the memory of a random cell in the grid, but at the cost of a high access time, operation called remote access.

The logic plane is strongly dependent on the algorithm. It is a custom circuit executing a certain sequence of operations. The set of all the cells active at the same time is able to produce the result of the algorithm on the input data store in the cells memory plane. Therefore, any change of algorithm requires a modification of the logic plane. Conversely, routing and memory plane are algorithm-independent. The first is standard because the communication system is a function of the division in cells and the set of available operations, which are properties of the architecture, independently of the algorithm. Likewise, the second is a simple array of words, which receives read/write commands from the routing plane, completely ignoring their purpose.

The remaining part of this Section describes in detail the architecture of the Logic-In-Memory cell: Routing plane (in paragraph 5.2.1), Logic plane (in paragraph 5.2.2 and Memory plane (in paragraph 5.2.3. Finally the instruction set is described in paragraph 5.2.4.

5.2.1 Routing Plane

The most complex layer is the routing one. This is the layer that manages the communication inside one PE and with near PEs. Its datapath is shown in Figure 5.5. It can be formally divided into three parts to ease the description: the input interface, the selection unit and the output interface. They are separated by thick dashed lines in Figure 5.5. In the following each of these parts is described.

5.2.1.1 The input interface

In Figure 5.5 it is possible to see that the input interface includes mainly the "Priority Management" component and some additional logic to select inputs. Indeed, each cell is able to communicate with its adjacent cells and it can receive signals from north, west, east and south cells. Only one of these inputs is selected using a multiplexer which is controlled by the priority manager. This component selects only one word according to its priority. Some of the rules to manage priority are described hereinafter: 1) If the logic plane performs a request, it has to be satisfied immediately, independently from the cell that is performing its own request. In addition, the priority manager detects if a request made by the cell itself has been satisfied or not: in the second case, an *acknowledge bit* is sent to the logic plane to inform it that the operation has to be performed again. 2) If more requests arrive at the same clock cycle, the priority manager selects the one with highest priority and sends to the other cell the acknowledge bit. Further, the two bits used by the priority manager to control the multiplexer are used as *destination bits*, useful to decide the cell that will receive a request.

5.2.1.2 The selection unit

Each instruction is made of a word composed by several fields. The selection unit task is to divide the word into the several fields, in order to save or manage them. Two main blocks compose the selection unit: the Target Cell Address (TCA) manager and the TAG manager (the TAG is the binary encoding of an operation).

The target address (TCA) is used to indicate each cell inside the grid: for this reason it is composed of two parts, one that specifies the position in the row and one for the column. The TCA is sent to a comparator to understand if the target cell is exactly the one that received the request; if not, the received word has to be passed to another cell until it reaches its destination. The comparator is therefore useful to determine the destination. Notice that since TCA represents the position in the grid, its size depends on the grid size (number of rows and number of columns). Hence, the bigger the grid, the longer the instruction.

The TAG is also sent either to the control unit (FSM in Figure 5.5) and to the tag generator. This component is fundamental to define which TAG has to be sent inside the successive word to send. Indeed, if a request for a read arrives, the sending TAG will be an “answer to read”; or, if a remote operation arrives to the cell that is non the target one, that operation has to be passed to another cell and therefore the TAG will be different. More details on the available operations and the relative TAGS are provided in paragraph 5.2.4.

5.2.1.3 The output interface

In this region the main components are the control unit (FSM) and the decoder. The control unit is the brain on this layer and sends the control bits to the other components. Its behavior is described by a state machine that manages the several situations that could occur. It has 4 inputs coming from previous blocks in the

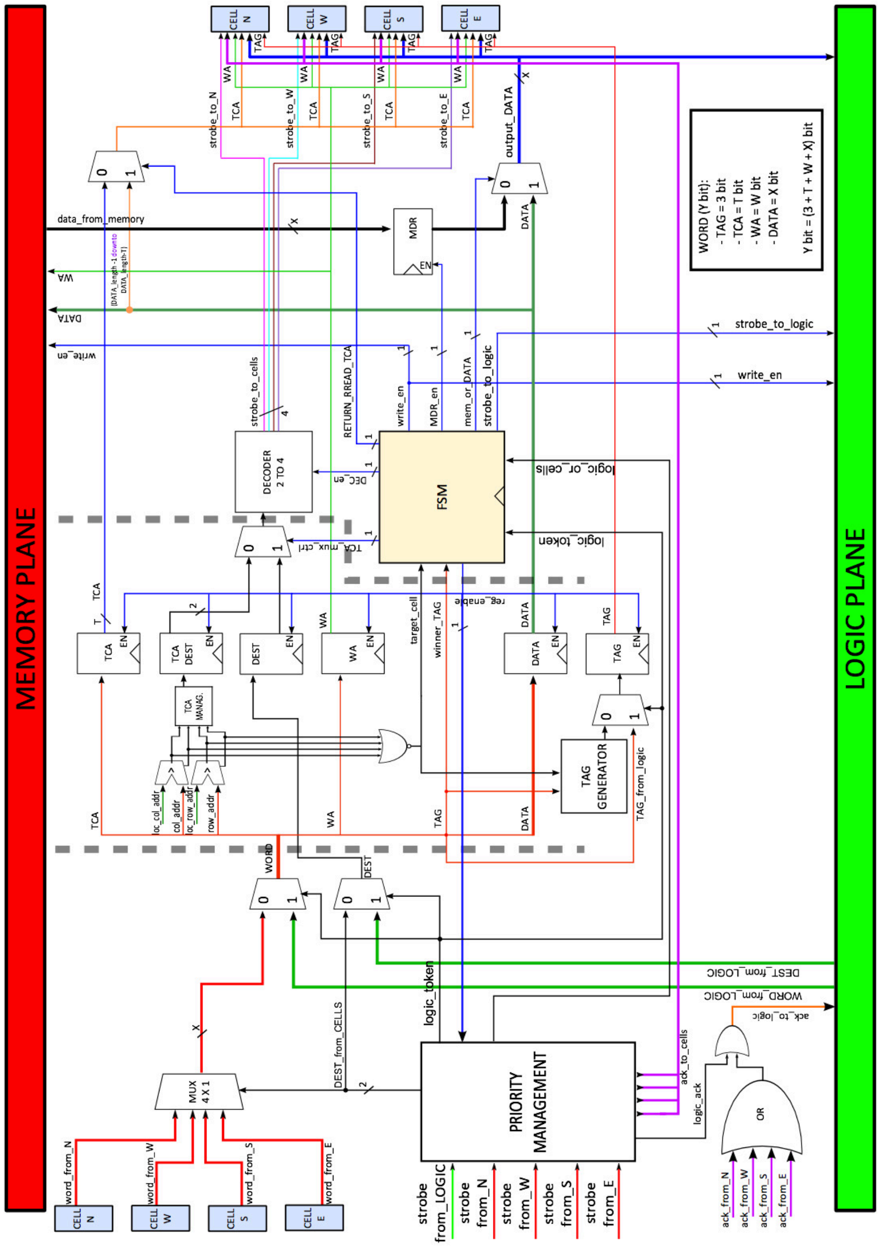


Figure 5.5. Logic-In-Memory Processing Element with focus on the datapath of the routing plane. [86]

routing plane, and has 8 outputs that manage the logic plane, the memory plane and some multiplexer in the outputs interface itself. For a more detailed description of this component it is possible to refer to [86][87]. The decoder is instead used to compute the acknowledge signal and to transmit it to the right near cell (selecting which one of the four adjacent cells is the destination one).

5.2.2 Logic Plane

The logic plane is the algorithm-dependent part of the logic-in-memory structure. It interacts with the routing plane, sending commands to it. In the case of a read operation it can receive values from the routing plane. Its role is to implement the algorithm, using the routing plane to interact with the memory. Even though the logic plane is algorithm-dependent, some parts can be standardized. For instance, the routing plane receives inputs and sends outputs in the form of packets. It is useful to implement a standard unit that composes/decomposes packets, distinct from the core of the logic plane (logic core), in which only the algorithm is implemented, without considering the subdivision in packets. This choice further simplifies the design of new algorithms, that practically require only the editing of the VHDL files describing the logic core. The two principal standard modules are Converters and Cell Logic Plane.

5.2.2.1 Converters

A series of converters receive in input the various fields of the instruction and composes the packets that are sent to the routing plane. In the same way, some de-converters decompose the input from the routing plane. This is a combinational part of the design, standardized and in this way it is separated from the logic core.

5.2.2.2 Cell Logic Plane

Cell logic plane is the standardized part of the logic plane. It forwards the commands received from the logic core to the routing plane, formatting them into packets, one at each clock cycle. At the same time it reads the input coming from the routing plane and transmit it to the logic core, decomposing the packets. The datapath is composed of a converter for each operation plus a de-converter for the read operations. When the logic plane starts a new instruction, a multiplexer selects the active converter according to the kind of operation. The choice is maintained until the operation is completed, i.e. all the packets have been transmitted. A counter counts the number of packets so that the converters can provide the right output at the right clock cycle.

5.2.3 Memory Plane

The memory plane is the storage area of the cell. It is designed as a rectangular memory, that receives a write enable signal from the routing plane, used to discern between write and read. The address in input is divided into two parts: the row and column address. The memory has a single input port with synchronous write and asynchronous reads.

5.2.4 Operation Set

To complete the description of the LIM 1.0 architecture, it is necessary to indicate what are the operations that can be performed between PEs. There are three main kind of operations: operations that occur inside a PE, called "Local"; operations that occur between one PE and an adjacent one, called "toc-toc"; operations that occur between distant cells, called "Remote". The following is the entire set of operations. Each of them can be assigned to a binary value for the TAG field.

- Local Write: the logic layer asks to write a data inside the memory of the same cell;
- Local Read: the logic layer asks to read a data from the memory of the same cell;
- *toc-toc* Write: a cell asks to an adjacent cell to write a data inside its memory;
- *toc-toc* Read: a cell asks to an adjacent cell to read a data from its memory;
- Return Read: after a *toc-toc* read, the data to be read is sent to the cell which performed the request. Once arrived, the data is sent to the logic layer;
- Remote Write: specifying the target cell address (TCA), a cell asks to write a data inside the memory of the target cell;
- Remote Read: specifying the target cell address (TCA), a cell asks to read a data from the memory of the target cell;
- Remote Return Read: it is the equivalent of the *return read.* Obviously in this case it follows the remote read operation;
- Logic-Logic: this operation has been introduced to increase the overall performance of the LIM. This operation allows a cell to compute a data and to send it immediately to the logic layer of another cell without saving it inside the memory. In this way not only clock cycles are saved but also the number of memory accesses is reduced.

5.3 LIM 2.0 Architecture

In previous Section we have analyzed the LIM 1.0 concept and architecture. If we consider the LIM Improvements define in paragraph 5.1.3, we can state that *Memory Locality* is perfectly fulfilled since each PE has its own memory layer and there is not an external memory. On the other hand, *Intelligent Memory* concept is not exploited, since memory is just distributed among the PEs but it has not any intelligence.

Therefore the aim of LIM 2.0 concept is to exploit also the Intelligent Memory, re-designing our structure in a different way. The principle of the improvement is presented in paragraph 5.3.1, while the new memory organization (pyramidal) is described in paragraph 5.3.2.

5.3.1 Improvement Concept

The architecture described in Section 5.2 is a first example of a Logic-In-Memory circuit, but it has several limitations and weak points. First of all, the way how the circuit works is similar to a 3D systolic array, where the brain of the architecture is the logic and the memory is just an instrument of the logic itself.

However, with Logic-In-Memory we mean a structure where the memory is the real brain of the circuit. In particular, the concept behind is that there must be no necessity to access a huge memory to read data since each cell is already provided with necessary information (so, data are available in the right cell and at the right moment). Obviously, sometimes it will be necessary to access the memory in order to store the computed data and provide each functional unit with new ones.

To accomplish these constraints, we need an architecture that loads a big amount of data from the memory and distributes data to each functional unit. Afterwards, while each unit is performing some mathematical operations, other data are collected, ready to be distributed again to each unit. This can be considered as a 3D pipeline of memories where those memories are smart and can decide and predict what data is going to be needed and which functional unit is going to request that data.

Differently from the first version of the LIM, in the LIM 2.0 the logic plane does not care about the data that it needs but its the memory plane duty to provide the logic with the correct data. This new structure may help to solve definitely the problem of memory bottleneck. Indeed, the amount of data needed by functional units and the quantity that can be loaded from the memory in a time unit is extremely different; with common architectural concepts, the logic is under-utilized. Exploiting the idea of 3D pipeline of smart memories, instead (whose detailed description is provided in paragrap 5.3.2) the vertex of the pyramidal memory accesses the outside memory and loads data. Since the frequency reached by memories is not high, the vertex may load a small quantity of data and then passes them to the lower layers that compute them working at their own frequency. While the logic layers are working, the vertex continuously collect data and provide them to the logic at the right moment. It can be said that the main role of the smart memories is to continuously feed the logic with proper data. Moreover, since the memory layer store themselves a certain amount of data, the slowest accesses to the outside memory do not compromise the performance of the system. Indeed, exploiting time and spatial locality of algorithms, as done in caching mechanism, data necessary to the logic layer will be probably already available somewhere in the intelligent memory plane: it is up to this intelligence to provide them to the right logic PE.

Moreover, the logic layer should be designed like an ALU, able to perform a large variety of mathematical and logical operations. Each operation should be identified by an opcode generated by the memories; in particular, each memory should communicate with its own ALU. The operation to be performed has to be specified by the memory according to the described algorithm: the new architecture should be therefore programmable in order to avoid to design the datapath of the logic layer every time that the algorithm changes.

5.3.2 Pyramidal Memory Design

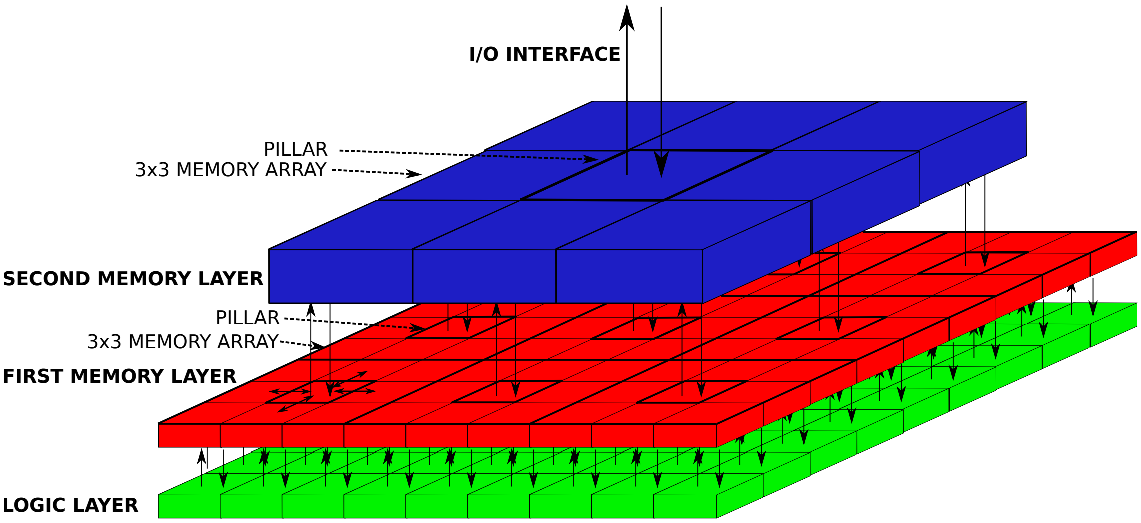


Figure 5.6. Pyramidal structure of LIM 2.0. The logic plane (green) is composed of as many units as the cells in the bottom plane (red). The upper plane (blue) has 9 cells, 1 for every 9 cells of the lowest memory plane. [86]

The new version of the LIM inherits a number of features from the previous architecture. First of all, the idea of having many independent functional unit organized in a matrix structure, that has been demonstrated to be successful. For this reason, in this new architecture, the PE is the key element but it is simpler than before. In principle each cell is composed of only two layers: the Memory and the Logic one. The Routing plane features available in LIM 1.0 are inherited by the memory plane. This concept is shown in Figure 5.6, where the green part is the logic layer and the red part is the memory layer, divided into the several PEs.

The fundamental novelty of this structure is that each memory layer is surrounded by other memories in order to create a 3D pipeline. The idea is to create a lower layer of memories where each of them communicates with a logic plane creating a cell. This cell is an element of a grid as it happened in the LIM 1.0. In addition, above the grid, there are other layers of memories only. In this way each memory is able to communicate with its neighboring memories and with the memories above and below. Only the memories at the bottom of the structure exchange data with the logic plane. To avoid an excess of interconnects, each memory can communicate with the layer below but only the "pillar" can communicate with the layer above. The pillar is a memory not different from the others but it is the only one that has connections to the above plane. In our design it is the central cell of a group of 9 memory cells. The entire design of the new architecture is shown in Figure 5.6.

The new LIM follows a pyramid model: it is composed of a vertex that communicates with the layer below made of 9 memories. Each of them exchanges data with another layer below made of 9 memories for each pillar, therefore 81 memories. So each plane is composed of a number of memories in a number of power of 9: the vertex is only one (9^0), the layer below is of $9^1 = 9$ memories, the other one is of $9^2 = 81$ memories and so on.

To summarize, the green box in Figure 5.6 is the logic layer: it is composed of a number of functional units equal to the number of memories of the first layer of the pyramid. Those functional units are ALUs and therefore are able to perform many different mathematical and logical operations. Each functional unit of the logic plane is controlled by one smart memory at the bottom layer of the pyramid, in red in Figure 5.6. Then it is shown in blue a second memory layer. Only the top of the pyramid is able to communicate with the user. The I/O pins should be used to provide each memory element with data and instructions (i.e., this becomes the interconnection with external slow memory in a Von Neumann style architecture).

As said before, the memory is the brain of the architecture, so each smart memory should be provided with a proper control unit. The idea is to model an algorithm into a sequence of instructions to be given to each memory element. To do so, the instructions ought to be passed to the vertex of the pyramid that will then distribute them to the neighboring memory cells and to the layer below. Each memory element at the bottom layer should have a sequence of instructions to provide to the logic plane. The logic plane is only constituted of an ALU to which the memory layer will transmit codes that identify the mathematical and logical operations to be performed. At the end of the algorithm, the logic ought to write the final result into the memory of the corresponding PE that will give new data to the functional unit and that will send the final result to the top of the pyramid. For this reason, the cells of the above layers should have two different memories to separate the data coming from the top and the neighboring cells and the data coming from the layer below. In this way it is possible to have an horizontal communication and a vertical communication: the first one is referred to the exchange of data between two different layers; the second one is referred to the exchange of data among cells of the same layer.

A brief description of the elements present inside each cell is given hereinafter. Memory elements incorporate features of the routing and memory planes of the LIM 1.0. Each memory block contains the *memory* used to store data, the *priority management* block used to handle conflicts in case of multiple requests and the *request management* block used to identify the operation that must be executed by a logic brick. The control unit is now fully programmable.

Logic elements are composed by an ALU that communicates only with its correspondent memory cell in the lowest memory layer. Data are fetched and provided to the logic cell by the memory, which also control the type of logic operation that must be executed. Each logic cell contains an adder/subtract unit, a multiplier, a unit capable of logic operations (LU), a comparator, a shifter and a counter. Multiplexers are used to control the data flow inside the logic cell. A more detailed description of the implementation of this architecture is provided in [86].

The instruction set of the LIM 2.0 is similar to the instruction set of the LIM 1.0, containing therefore the instructions used to exchange information among cells. Moreover the instruction set features a long set of arithmetic and logic instructions and commands used to program the control unit in each memory cell. Each memory cell can be programmed independently, giving to the architecture the maximum possible flexibility.

One consideration is necessary now about the mechanism to provide data to logic elements at the right time. Predictive techniques can be used, as done in caching mechanism, and the spatial and temporary locality of data will help as well. We have still not developed a mechanism to produce this predictive technique, so in the example reported in paragraph 5.4.1 we have designed the data exchange ad-hoc for the algorithm. Nevertheless, this is an important improvement that will be analyzed in the future.

5.4 Results

We have shown two possible implementations of the LIM concept and we have pointed out the main differences with respect to Von Neumann architectures and

GPUs. To have a clear indication of the benefits and disadvantages of our proposed architectures, it has been decided to run one particular algorithm on the two LIM architectures, as well as on a GPU and using a dedicated ASIC. The algorithm is presented in paragraph 5.4.1, while the comparison of results is shown in paragraph 5.4.2.

5.4.1 Test Algorithm

As target algorithm for the demonstration of the performance of our LIM architectures we use the odd-even sort algorithm. This is a sorting algorithm developed to be used on parallel processor. The purpose of the algorithm is to compare all odd/even indexed pairs of adjacent elements in a list and, if a pair is in the wrong order, the elements are switched. The next step repeats this procedure for even/odd indexed pairs (of adjacent elements). Then it alternates between odd/even and even/odd steps until the list is sorted. The code of the algorithm is reported in listing 5.1.

Listing 5.1. Odd Even Sort algorithm

```
void OddEvenSort (T a[], int n) {
  for (int i = 0 ; i < n ; i++) {
     if (i & 1) /* 'i' is odd */ {
         for (int j = 2 ; j < n ; j += 2) {
              if (a[j] < a[j-1])
                  swap (a[j-1], a[j]);
              }
         }
      else {
          for (int j = 1 ; j < n ; j += 2) {
               if (a[j] < a[j-1])
                   swap (a[j-1], a[j]) ;
          }
      }
  }
}
```

5.4.2 Results Comparison

In this paragraph we compare the obtained results of the Odd-Even sort algorithm with those achievable with other kind of digital architectures. The goal is to understand the quality and the relevance of our architecture. In particular we focused on two targets in our tests, Application Specific Integrated Chips (ASIC) and GPUs. ASIC circuit is chosen because generally it allows to reach the maximum speed among integrated circuits. GPUs are nowadays largely used as hardware accelerators. They are fast parallel architectures where memory is already embedded locally inside the chip. They offer therefore a fair comparison with our circuits, especially with the LIM 2.0. Since we do not have the possibility to implement the odd-even sort algorithm on a GPU to test its performance, we have used data presented in the paper of Khan et al. *"Analysis of Fast Parallel Sorting Algorithms for GPU Architectures"* [88].

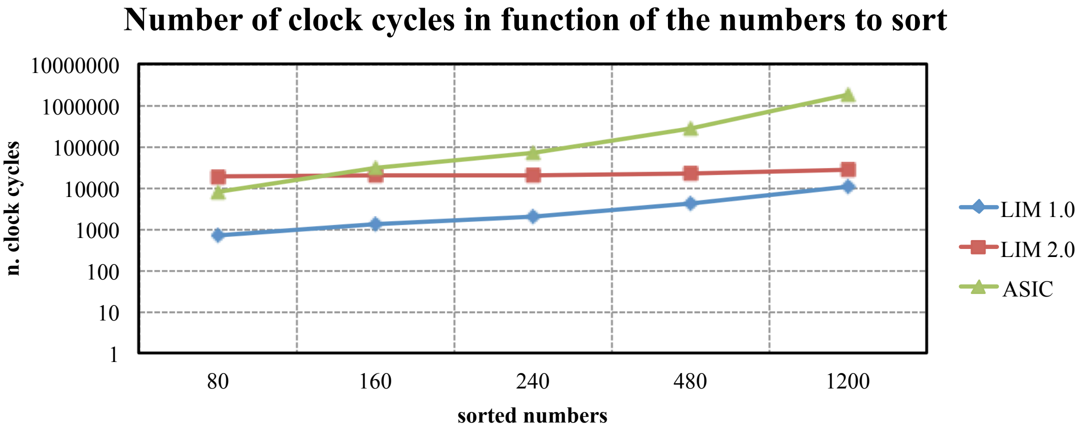


Figure 5.7. Comparison between LIM 1.0 and LIM 2.0 and ASIC implementation of the odd-even sort algorithm. Logarithmic graph.

To compare the results obtained we designed a custom ASIC circuit and synthesized it on a 28 nm low power technology. We followed two rules in the designing the ASIC test circuit. First, it has a traditional architecture, a logic circuit executing the sorting algorithm and fetching data from a separated memory. Second, to do a fair comparison the test architecture is partially parallel. Four units work in parallel inside the chip fetching data from four independent memories. Since the architecture is very simple, the working frequency obtained after the synthesis is extremely high, 5 GHz, much higher then the working frequency of our LIM circuits. Instead of synthesizing the memory directly on chip, we choose to use the GDDR5 memories recently developed by Samsung [89]. The Samsung GDDR5 memory are capable of reaching a theoretical frequency of 8 GHz for each signal, which is higher than the frequency that we obtain from the synthesis on 28 nm technology. In doing the comparison, we suppose that our custom chip is surrounded by four GDDR5 modules that work in parallel at the maximum frequency. Figure 5.7 shows the time required by the LIM architecture and the test ASIC to sort up to 1200 numbers. The time is expressed in number of clock cycles, so it is independent from the technology chosen and the type of memory selected. Notice that the graph is in logarithmic scale. The difference between the LIM architecture and the ASIC is huge. With 1200 numbers nearly 1800000 clock cycles are required by the ASIC against 10000 for the LIM 1.0 and 28000 for the LIM 2.0. Results in term of timing are reported in Table 5.1. Results presented in [88] are used to compare the LIM performance with a GPU. The GPU used in [88] is a NVDIA QUADRO 6000. It is based on a 40 nm technology, the processor clock is fixed at 1148 MHz and it has 6 Gb of GDDR5 memory. Results comparison, when sorting 2^{15} numbers, is reported in Table 5.1.

The GPU needs 230 ms to sort all the numbers. Our test ASIC circuit has similar performance, requiring slightly more time, 268 ms. The difference with both LIM versions is very high, since the LIM 1.0 needs only 0.283 ms and the LIM 2.0 just 0.269 ms. The LIM architecture is nearly 800 times faster. These results also demonstrate that the LIM 2.0 is faster then the LIM 1.0 with millions of numbers to sort. The comparison with the GPU is particularly important, because they are both two fully programmable parallel architectures. The LIM is faster and it has also a lower power consumption, 40 W against the 204 W of the NVIDIA QUADRO 6000 at full load.

The results for our test ASIC circuit were obtained considering a state of the art memory working in ideal conditions. Moreover we are not considering any memory access time, supposing that data are continuously fed to the circuit. The working frequency and the test conditions that we have chosen allow the logic part to work always at maximum speed. In many situations this condition is not verified and the logic circuit is slowed down by the memory. To test these assumption we simply consider two other types of memory, the same GDDR5 used by the NVIDIA QUADRO GPU working at 3 GHz and a common DDR3 memory working at 1.6 GHz. The complete comparison in terms of timing required to sort 2^{15} numbers is reported in Table 5.1.

Odd-Even Sort	n. of clock cycles	Freq. (GHz)	Time (ms)	Speed-up wrt GPU
LIM 1.0	$2.83{*}10^5$	1.0	0.283	813.8
LIM 2.0	$2.69{*}10^5$	1.0	0.269	853.6
ASIC (GDDR5 8GHz)	$1.34{*}10^9$	5.0	268	0.857
ASIC (GDDR5 3GHz)	$1.34{*}10^9$	3.0	447	0.5
ASIC (DDR3 1.6GHz)	$1.34{*}10^9$	1.6	838	0.26
Quadro 6000		1.1	230	

Table 5.1. Performance comparison among the two version of LIM architecture, the test ASIC circuit with three types of memory and the NVIDIA Quadro 6000 GPU.

Considering a GDDR5 memory working at 3 GHz the execution time is increased to 447 ms, while with a DDR3 memory the time required is 838 ms. These results are useful to highlight a common situation in many computational systems, where the overall speed of the circuit is limited by the memory. In these more realistic cases, the advantages provided by our Logic-In-Memory architecture are much evident.

5.5 Final Remarks

The concept of Systolic Arrays used in previous Chapters as an interesting solution for NML technology, shows its limitations when inserted in a more general picture. This architecture, given its parallel nature, requires a huge amount of data for computations. These data can be retrieved from an external memory, connected to the processor with a bus as in a common Von-Neumann architecture. The problem of this architecture is that bus communication becomes the bottleneck of the system; the Systolic Array cannot be exploited at its best because memory cannot feed data at the same operating rate. Finally this results in a reduction of the overall performance.

To overcome this limitation, we have introduced a new architecture called Logic-In-Memory, where logic elements and memory ones are merged in one single device. This architecture presents several new features, while inherits some others from Systolic Arrays and GPUs. Several achievements can be mentioned in this research path:

- Analysis of the scenario and alternatives: first an analysis on other parallel processing units has been performed to analyze their strengths and weaknesses. Based on this analysis the principal characteristics of Logic-In-Memory have been defined.

- LIM 1.0 Architecture: a new architecture composed of Processing Elements, each made of a memory, logic and routing plane. In particular the routing plane is the core of this architecture, that manages communication inside and between cells.

- LIM 2.0 Architecture: an evolution of the first architecture, composed by a logic plane and a 3D pipelined memory. This architecture is definitively different from the previous one and respects the given constraint of intelligent memory. Each PE is made by a logic plane (reconfigurable) and a memory plane that is in turn connected to upper memory planes through pillars. This architecture guarantees high flexibility and reduced access to higher memory levels.

- Overall comparison: using as benchmark the Odd-Even Sort algorithm, we have compared the results of our LIM architectures with ASIC chips and GPU. Our LIM finally turn out to be the best solutions in terms of processing time when huge amount of data are sorted.

With this scenario we close the first part of this research having analyzed Systolic Arrays and with incremental improvements having reached Reconfigurable Systolic Arrays. Then, with a complete change of paradigm, we have introduced a new kind of architecture that can still benefit of the improvements presented for Systolic Arrays, but is also able to eliminate the memory communication bottleneck of common systems, that we have called Logic-In-Memory.

Part II

MagnetoElastic NML Circuit Design

Chapter 6

Design Rules for ME-NML Circuits

The Second Part of the research has focused on MagnetoElastic NML (ME-NML). This technology is quite different from the classical magnetic clock NML, but it allows to achieve important reduction in power dissipation and area occupation. While this technology is extremely interesting for its characteristics, it is still in development and few small circuits have been proposed. The objective of this research path is to enrich the tools and methodologies necessary to approach the design of this kind of circuits. At the same time, it is important to approach circuits with higher complexity to analyze the features or drawbacks that can arise when many ME-NML cells are considered. Indeed, only considering small circuits with few cells, it is not possible to understand the projection to real circuits. It is only with big and significant circuits that the characteristics of this technology can be analyzed proficiently.

In this Chapter the MagnetoElastic NML (ME-NML) technology is studied to make the design of circuits with this technology more effective. In particular, Section 6.1 deals with the definition of a Standard Cell Approach for this technology, that can be used for a future automatic tool for the design of ME-NML circuits. Section 6.2 presents the design of a Galois Field Multiplier (GFM), the first conceived complex logic circuit in ME-NML. From the analysis of the GFM, it came out the opportunity to study in more detail the Parallel versus Serial approach in ME-NML, that is treated in Section 6.3. Finally the most important results achieved are resumed in Section 6.4.

6.1 Standard Cell Approach for ME-NML Circuits

In this research topic, a completely new and rigorous approach for the design of ME-NML circuits, is defined. This is based on: 1) a set of Standard Cells, analyzed in Section 6.1.1; 2) a VHDL model to simulate any logic circuit in ME-NML, described in Section 6.1.2; 3) a standard circuit layout presented in Section 6.1.3. With these achievements the automatic synthesis and routing of ME-NML logic circuits can become much simpler.

6.1.1 Standard Cells Library

In Section 2.3.3 we have anticipated the layout of ME-NML circuits. This is based on mechanically isolated islands, also called ME-NML cells. Each of these cells receives its own clock signal. For fabrication and physical limitations, the height and width of a ME-NML cell can be of either 3 or 5 magnets. In particular using these cell sizes, it is possible to guarantee an operating frequency of 100 MHz, being sure that all the magnets inside the cell will flip correctly.

In this work we consider the 3×3 cell dimension, that is the smallest size feasible with current lithographic resolution. Compared to bigger cells, it has a shorter critical path (number of cascaded magnets) leading to both an higher working speed and a better signal propagation reliability. The drawings and circuits described hereinafter consider therefore 3×3 cells, but the VHDL model presented in Section 6.1.2 is generalized for any cell size.

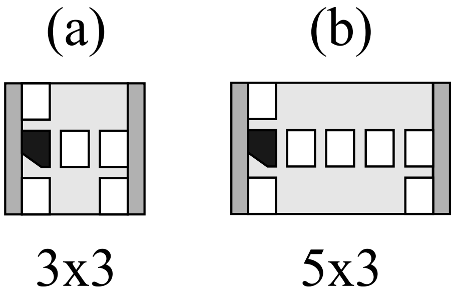


Figure 6.1. ME-NML cells. (a) 3×3 size. (b) 5×3 size.

Given these ME-NML cells sizes, we have identified a limited number of possible magnets configurations. Hence the totality of the conceivable cells configuration is reasonably small. In this way it is possible to define a Standard Cell Library, where each element is described in VHDL language. The result is that, using the cells of this library, any digital circuit can be designed. This feature is particularly interesting in the perspective of automatic synthesis tools for ME-NML, that do not exist at the time of writing but for which some work has been done at Politecnico di Torino in ToPoliNano [90][91].

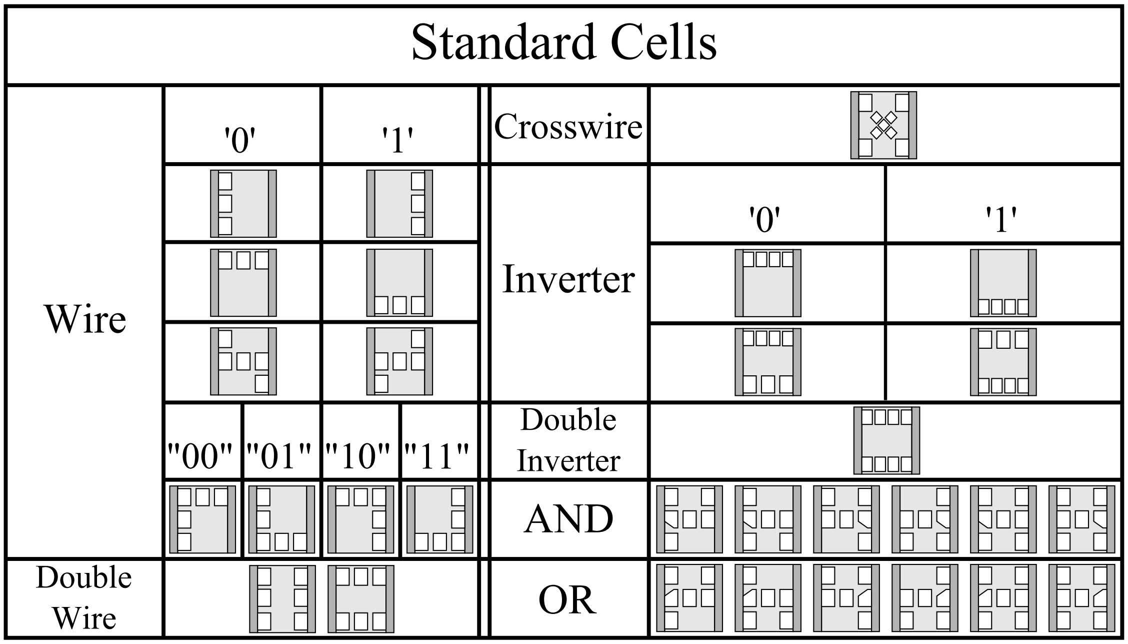


Figure 6.2. Standard 3 × 3 Cells Library for ME-NML. [27]

The full Standard 3 × 3-Cells Library is presented in Figure 6.2 [27]. The logic gates used are: Wire, Crosswire, Inverter, AND, OR. So, except for Crossiwire that is used to cross two signals in this planar technology, the logic gates used are the same of CMOS circuits. In a complete Standard Cells Library, these cells must be distinguished also by layout and orientation, not only by their logic function. In this way it will be possible, from a general circuit layout composed of Nanomagnets, isolating a 3 × 3 magnets area, to assign a corresponding ME-NML Standard Cell.

Cells aligned in the same row in Figure 6.2 can be derived from each other by horizontal and/or vertical flipping. Since they represent different orientations of the same cell, they are described using the same VHDL `entity`. The numbers in Figure 6.2 for a certain cell, will be given as `generic` parameters to identify the cell orientation. This does not apply to *Double Wire*, *AND*, *OR*: These cells are shown in the same row to get a more compact image, but they have to be defined with different VHDL `entity`.

In the following the several Standard Cells are described. In general each cell is modeled as a CMOS register plus, if needed, an ideal logic port. The register models the delay of the cell, that is one clock phase, while the logic port is necessary to model *AND*, *OR* and *Inverter*.

AND: In Section 2.3.1 we described how *AND* and *OR* gates can be obtained by modifying the shape of a magnet [22]. A cut on the bottom left corner gives a preferred orientation to the magnet, thus creating the AND function. None of the six AND cells in the library can be derived from another one by vertical or horizontal flipping. Notice that the slanted edge is always on the bottom left corner of the magnet. Therefore each AND cell is described with a different VHDL `entity`. The first four cells have one output, while the others have two outputs. The third, fourth, and sixth configuration have inputs on the right side of the cell, while the others have inputs on the left side of the cell.

OR: The six OR cells of the library are exactly the same of the AND cells, except for the slanted magnet that produces the logic gate. In this case the slanted edge is on the top left corner of the magnet.

Wire: The cells belonging to the *wire* group are composed by a number of adjacent magnets. In NML technology these magnets can propagate signals with a domino-like behavior. The horizontal alignment of magnets is antiferromagnetic, as explained in Section 2.3, while vertically each magnet has the same polarization of its neighbors. Therefore, to work as a wire, an ME-NML cell must have an odd number of horizontal magnets. Since wires have no logic function, each wire cell is simply described as a register.

As clear from Figure 6.2, there are four different wires in the library: Vertical Wire with two possible orientations (*left* and *right*); Horizontal Wire with two possible orientations (*up* and *down*); Long Wire, that connect one signal from one corner to the opposite one, with two possible orientations; 2 Outputs Wire with four possible orientations.

Double Wire: It contains two independent wires with length of three magnets. In the model this cell is described with two different registers. There are two Double Wire cells in the library, described by two different VHDL `entities`, one for horizontal propagation and the other for vertical one.

Crosswire: It is modeled similarly to the Double Wire, but physically the wires cross each other. This interference-immune crossing is fundamental, since for now NML is still a planar technology.

Inverter: The horizontal antiferromagnetic alignment of magnets is exploited to obtain the inverter function: Any even number of adjacent horizontal magnets generates an inversion in the signal. The VHDL model is modeled as an ideal CMOS inverter plus register. There are two possible configurations depending if the inverter is placed on the *top* or *bottom* zone of the cell. Just like for the wires, two inverters can be present in the same cell, but only horizontally (Double Inverter). The vertical coupling is ferromagnetic, so the inversion does not take place. The library also contains a cell with both an inverter and a horizontal wire (Inverter plus Wire).

6.1.2 VHDL Model for ME-NML Circuits Design

Once that the Standard Cell Library has been defined, each of the cells can be modeled with a VHDL `entity`. In this Section the VHDL model is described in details. This model is used to represent the logic behavior of the circuit, but it also computes the Area and Power and can hierarchically sum up these values. In this way the top entity can evaluate the total area and power of the circuit. The Listing 6.1 is used as example; it contains the code that models the *Inverter plus Wire*. The inverter (4 adjacent magnets) and the wire (3 adjacent magnets) are horizontal, so the cell can be flipped around its horizontal axis.

6.1.2.1 Generic parameters

Each VHDL `entity` has some `generic` parameters that are used to assign a clock zone to each cell and its relative positioning within the circuit (see lines 2-4 of listing 6.1). In Figure 6.4(b) they are represented as inputs of the Standard Cell.

PHASE: ME-NML has a 4-phase clocking system. This `generic` defines which one of the four clock signals must be connected to the cell. This information is redundant, as the required clock signal is directly connected to the *clk* port, but it is included to assure a better suitability of this model to a design tool.

ROW and COLUMN: ME-NML circuits are composed by cells arranged in a grid-like fashion, as it will further described in Section 6.1.3. ROW and COLUMN refer to the relative position of a cell within the circuit described by the upper level entity.

ORIENTATION: As represented in Figure 6.2, when cells can be obtained from each other by a simple flipping, they are described using the same VHDL file. The ORIENTATION parameter defines which one to use. It does not affect the logic or the circuit performance, but it can be used for automatic layout with a design tool.

H and L: The choice for this work has been to exploit 3 × 3 clock zones. So the height and width (in terms of nanomagnets) of a cell are always equal to 3. Anyway the model is as generic as possible, so the height and width are defined as parameters: H and L.

Listing 6.1. "Inverter plus Wire" VHDL Model

```
 1 entity inv_with_wire is
 2     generic(PHASE: std_logic_vector(1 downto 0);-- Clk phase.
 3                    ROW: natural;                          -- Relative cell position (row)
 4                    COLUMN: natural;                    -- Relative cell position (col)
 5                    ORIENTATION:std_logic;
 6                    H: natural;                            -- Height (# of magnets)
 7                    L: natural);                           -- Width (# of magnets)
 8     port(       d1,d2: in std_logic;             -- Inputs
 9                    clk: in std_logic;                 -- Depends on the phase
10                    q1_n,q2: out std_logic;      -- Outputs
11                    n_mag: buffer natural;       -- # of magnets
12                    n_zones: out natural := 1;-- # number of cells
13                    area_eff: out natural;       -- Total magnets area
14                    area_tot: out natural;       -- Cell area
15                    Er: out natural;                     -- Switching energy
16                    Ec: out natural);                    -- Clock network losses
17 end inv_with_wire;
18
19 architecture behavior of inv_with_wire is
20     component reg is -- D FlipFlop (1bit)
21         ...
22     end component;
23     component area_and_energy is
24         generic(H: natural;                         -- Height (# of magnets)
25                        L: natural);                        -- Width (# of magnets)
26         port(       n_mag: in natural;            -- # of magnets
27                        area_eff: out natural;  -- Total magnets area
28                        area_tot: out natural;  -- Cell area
29                        Er: out natural;                 -- Switching energy
30                        Ec: out natural);                -- Clock network losses
31     end component;
32     signal q1: std_logic;
33 begin
34     n_mag <= L*2+1; -- Evaluate the number of magnets using H and L.
35     q1_n <= not q1; -- Inversion
36
37     Wire1: reg port map(d => d1,clk => clk,q => q1);
38     Wire2: reg port map(d => d2,clk => clk,q => q2);
39
40     Evaluate_area_energy: area_and_energy generic map(H,L)
41         port map(n_mag,area_eff,area_tot,Er,Ec);
42 end behavior;
```

6.1.2.2 Logic Behavior of the Cell

Referring to the listing 6.1, it is necessary to describe the logic behavior of the cell. The "Inverter plus Wire" cell is composed by two parallel series of magnets: 4 for the inverter and 3 for the wire. Therefore it is modeled by two D Flip Flops, plus an ideal inverter applied to one of the outputs. Lines 37-38 of the listing contain the registers instances, while the inversion function is described at line 35.

6.1.2.3 Area and Energy

As previously described, the model is able to compute Area and Energy hierarchically. Each cell described with VHDL evaluates and gives as output its own number of magnets (*n_mag*), its area occupation (*area_eff*, *area_tot*) and power consumption

(*Er, Ec*) (6.4.B). The number of magnets is evaluated at line 34, while the other values are calculated by a component named `area_and_energy` (lines 23-31 and 40-41 in listing 6.1). This component, starting from the number of magnets, height and width of a cell, provides as output the required information on area and power.

The two following paragraphs clarify the equations implemented in the VHDL model to evaluate area occupation and power dissipation.

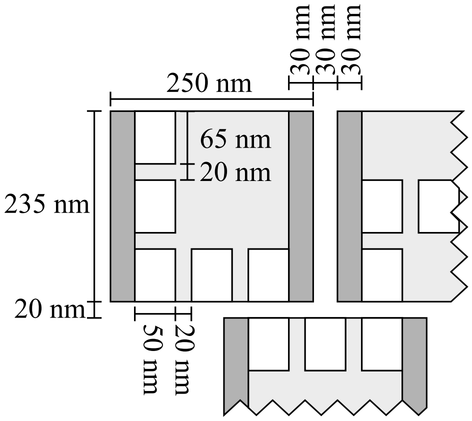


Figure 6.3. Detailed measures of the ME-NML 3 × 3 cell. [27]

Area information Figure 6.3 shows the complete clock zone measures and the distance from nearby cells. The relevant parameters are:

- Magnets height: $H_{mag} = 65\,nm$, width: $W_{mag} = 50\,nm$.
- Magnets separation. Both horizontal and vertical separation: $Sep_{mag} = 20\,nm$.
- Electrodes width: $W_{electrode} = 30\,nm$.
- Cells separation: horizontal: $Sep_{h\_cell} = 30\,nm$, vertical: $Sep_{v\_cell} = 20\,nm$.

The values above are assigned to the constants in the model in a dedicated constants file, so that the component `area_and_power` can evaluate the correct area information for each cell. Each cell gives as output two data related to area occupation:

Magnets Area: this is the total area occupied by magnets. It is normally given by the area of a magnet multiplied for the number of magnets.

$$A_{magnets} = (H_{mag} \cdot W_{mag}) \cdot n_{mag} \tag{6.1}$$

Cell Area: It is the area of the cell, including the electrodes and the separation space among cells. It is used to evaluate the total area of the circuit. Since in this work the cell dimension is fixed to 3 × 3, the Cell Area will be the same for every cell.

$$H_{cell} = 3 \cdot (H_{mag} + Sep_{mag}) = 255\,nm \tag{6.2}$$

$$W_{cell} = 3 \cdot W_{mag} + 2 \cdot (Sep_{mag} + W_{electrode}) + Sep_{h\_cell} = 280\,nm \tag{6.3}$$

$$A_{cell} = H_{cell} \cdot W_{cell} = 71400\,nm^2 \tag{6.4}$$

Energy information The `area_and_power` component actually estimates the energy dissipation E and not the power. Knowing the working frequency f_{clk}, that for this technology and with this cell layout can be assumed to be $100\,MHz$, the power P can be easily derived with equation 6.5.

$$P = E \cdot f_{clk} \tag{6.5}$$

Energy constants used for energy computation are defined in the VHDL model and reported in Listing 6.2. The main sources of energy dissipation in NML circuits are basically two:

Magnets Switching: this is the intrinsic energy loss required to force magnets in the NULL state (E_r in Listing 6.1). The switching can be either adiabatic or abrupt: For the Magnetic clock NML the difference in term of losses is extremely wide, so the switching has to be adiabatic. However, in ME-NML, the behavior is different: The energy consumption is still equal to $30K_bT$ if adiabatic, but only $180K_bT$ (the whole energy barrier for $50 \times 65 \times 10\,nm^3$ nanomagnets) if abrupt. Since in both cases the consumption will be negligible compared to the second component, the choice is the abrupt switching, that allows to reach better performance.

After defining how much energy is dissipated by the switching of a single magnet (E_{mag}), to obtain the energy consumption of the entire cell it is possible to multiply for the number of magnets:

$$E_{cell} = n_{mag} \cdot E_{mag} \tag{6.6}$$

Clock Network: this is the energy dissipated by the clock network mainly due to Joule losses (E_c in Listing 6.1). Since PZT (like all piezoelastic materials) is an insulator, a ME-NML cell behaves as a capacitor. Therefore the main contribution to clock losses is the charge of such capacitor.

The capacitance is estimated in equation 6.7 [15].

$$C = \frac{\epsilon_0 \cdot \epsilon_r \cdot t_{PZT} \cdot H_{cell\_eff}}{W_{cell\_eff}} \tag{6.7}$$

The first three constants are the absolute dielectric constant (ϵ_0), the relative dielectric constant of PZT (ϵ_r), the thickness of the PZT substrate ($t_{PZT} = 40\,nm$[15]). The other two values are the effective dimensions of a clock zone, without the inclusion of the separation between cells. Hence $H_{cell\_eff} = 235\,nm$ and $W_{cell\_eff} = 250\,nm$ (Figure 6.3).

Equation 6.8 evaluates instead the voltage that must be applied to a clock zone to force it into the RESET state.

$$V = \frac{W_{cell\_eff} \cdot \sigma}{Y \cdot d_{33}} \tag{6.8}$$

Listing 6.2. Constants for Energy estimation

```
-- CONSTANTS FOR ENERGY EVALUATION ------------------------------------------

-- For switching energy evaluation
constant Kb:                         real := 13.8065e-23;--Boltzmann const
constant T:                          real := 300.0;         --Room temperature (K)
constant E_MAG:                   real := 180*Kb*T;  --
-- For clock energy evaluation
constant VACUUM_PERM:   real := 8.854e-12;   --Vacuum permittivity (F/m).
constant REL_PERM:          real := 1300.0;         --Substrate relative perm. (-)
constant T_PZT:                   real := 40e-9;            --Electrodes thickness (m)
constant STRESS:                  real := 28e+6;            --Applied stress (Pa)
constant YOUNG_MODULUS: real := 80e+9;            --Young modulus of Terfenol (Pa)
constant PZT_CONST:         real := 150e-12;        --Piezo coeff. (m/V)
```

In this equation $\sigma = 28\,MPa$ is the applied stress, $Y = 80\,GPa$is the Young modulus for Terfenol and $d_{33} = 150\,pm/V$ it the coefficient for strain and applied voltage coupling in the PZT substrate. Normally the applied voltage should be in the range of $0.7 - 1.3V$ [15]. Finally the energy required to charge the capacitance of one cell is expressed in well known equation 6.9.

$$E_{clk} = \frac{1}{2} \cdot C \cdot V^2 \tag{6.9}$$

The power contribution of the circuit for clock generation is negligible, as the circuit counts a limited number of transistors [13]. Therefore this component is not be taken into account.

6.1.2.4 Hierarchical model

We have exploited the hierarchical property of VHDL language to obtain a hierarchical model. The standard cells compose the bottom layer, while components in the

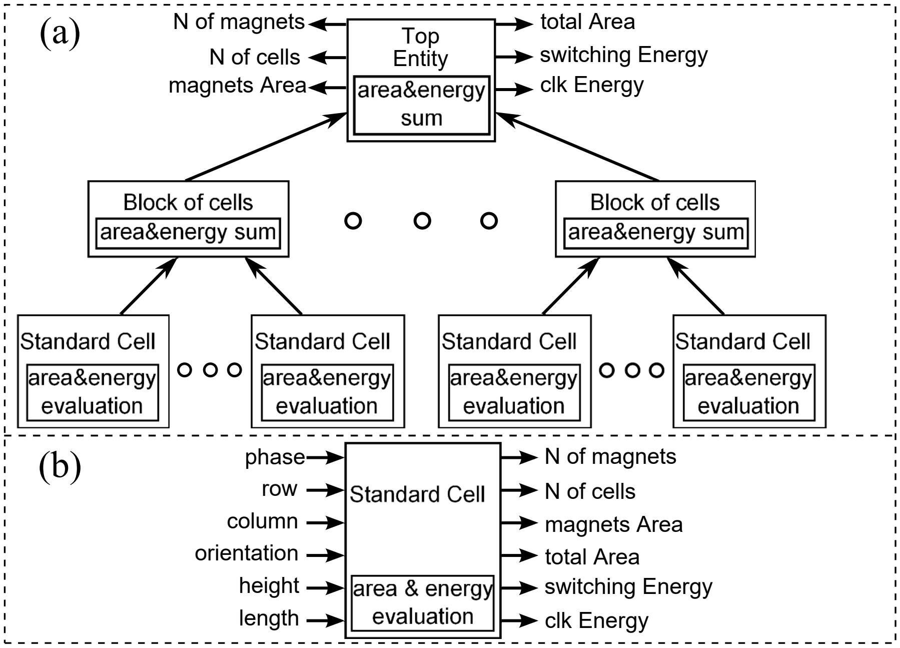


Figure 6.4. (a) VHDL hierarchical model. The information on energy dissipation and area occupation are propagated hierarchically toward the top entity. (b) `generic` inputs and outputs of a Standard Cell.

upper layer are assembled together to create logic circuits. These blocks of cells can be then instantiated by bigger circuits and so on up to the top entity. Figure 6.4(a) depicts a generic 3-layers hierarchy. The Top Entity (layer 3) is composed by many Block of cells (layer 2), while each block of cells encloses the required standard cells (layer 1).

This hierarchy is exploited for a bottom-up evaluation of the number of magnets, number of cells and performance in terms of area and power. As explained in paragraph 6.1.2.3, each Standard Cell provides in output all these information about itself thanks to the `area_and_power` component. The elements in the upper layer sum up the data received from every element in the lower layer (using the component *arrays sum* shown in Figure 6.4), providing the results of that Block. This mechanism goes on recursively up to the Top Entity, that produces as output the total results for the whole circuit. Notice that the model provides exact results, as there is no approximation in the hierarchical evaluation and the circuit design for ME-NML provides a layout correspondent with the actual physical mapping.

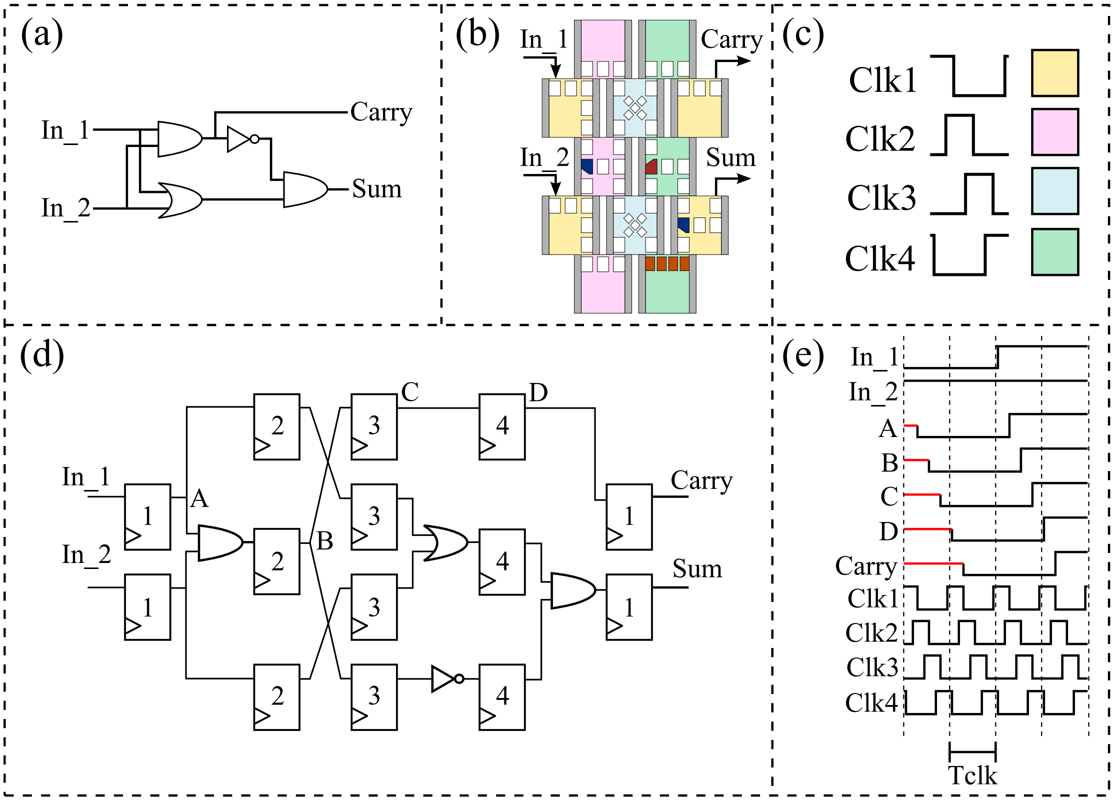


Figure 6.5. (a) CMOS Half Adder. (b) ME-NML Half Adder. (c) Waveforms for the 4-phase overlapped clock system. A color is associated to each clock signal. (d) VHDL counterpart of the ME-NML circuit, it is the circuit described by the VHDL model. (e) Timing diagram of an example of signal propagation through the adder. [27]

6.1.3 Circuit layout

In the Standard Cell Approach proposed for ME-NML logic circuits, we have till now presented the Cells Library that can be used to design any circuit and the VHDL model that can be used to simulate it and to estimate power and area parameters. In this section we provide the first example of a ME-NML circuit, focusing on many general aspects of the design: the circuit layout, the CMOS circuit described by the model, the multiphase clocking system, the timing of signal propagation. The simple circuit presented as case study is a Half Adder (HA). A configuration comprising only Inverter, AND, OR logic gates (the gates available in ME-NML technology) is presented in Figure 6.5(a), while the corresponding ME-NML circuit is in Figure 6.5(b).

Cells are placed on a grid-like scheme as we have pointed out in the introduction(Figure 2.9(d)). The path from inputs to outputs is 5 clock zones long. To ease the comprehension of the ME-NML circuit, the AND, OR and inverter magnets are highlighted respectively in blue, red and orange, while the substrate color identifies the clock phase of a cell. The clock system for ME-NML is composed of 4-phases overlapped clocks, whose 4 waveforms, with their assigned colors, are shown in Figure 6.5(d).

The model presented in this chapter maps each clock zone to one register, plus a logic gate if needed. The VHDL code for the ME-NML HA describes an equivalent RTL circuit as shown in 6.5(d). Notice that the path from input to outputs counts 5 registers (5 pipeline stages), just like the 5 clock zones needed to pass through the ME-NML version. The numbers inside registers define their clock phase.

The timing graph in Figure 6.5(e) shows a simple propagation example. The signals follow the path from the inputs to the Carry output, passing through the nodes indicated with A-B-C-D in Figure 6.5(d). All clock signals have the same period T_{clk}, but they are shifted by 90° depending on the clock phase. It is quite clear from the timing that a signal needs one clock cycle T_{clk} to cross 4 clock zones (registers in the VHDL counterpart of the ME-NML circuit). Hence a signal has a latency of $T_{clk}/4$ to cross a clock zone.

6.2 Circuit Design Example: Galois Field Multiplier

In previous section the tools and methodologies for an effective design of ME-NML circuits have been provided. To show the benefits of this approach, and to qualify the effectiveness of this technology, in this section it is provided a complex circuit design example.

In particular, this research topic aims at giving a preliminary answer to the following fundamental question: does ME-NML offer significant improvements over state-of-the-art CMOS transistors? Moreover, it will be clear if this technology really overcomes the classic Magnetic Clock NML in terms of power dissipation (that is the main drawback of classic NML).

To prove the benefits of ME-NML, it is presented an accurate comparison of performances between three different implementations of the same circuit: 28 nm CMOS, Magnetic Clock NML and ME-NML.

The circuit chosen as example is a Galois Field Multiplier (GFM). This circuit is interesting because it is used in several applications in the field of cryptography, digital signal processing, coding theory and computer algebra. We will discuss in detail in paragraph 6.2.1 the circuit adopted for Galois Field Multiplication. It is however worth noticing in advance that the common hardware implementation of the Montgomery algorithm, to execute Galois Field Multiplication, is a Systolic Array structure.

The advantages of this kind of architecture for NML have been described in Chapter 3. Here we recall that this kind of circuits show strong modularity [33], avoiding long interconnections that are inefficient in NML technology.

6.2.1 Galois Field Multiplier circuit

To describe the Galois Field Multiplier (GFM) circuit, we start recalling the main principles of Galois Field computing. A Galois Field GF(q) encloses a finite number q of elements, together with the definition of addition and multiplication operations on pair of elements [92]. When $q = p^m$, with m positive integer and p prime number, the field exists and is unique. For this work we are exclusively interested in Binary Galois Fields (GF(2^m), $p = 2$), as they perfectly suit digital systems. So GF(2^1), the smallest possible Binary Galois Field, only has the two elements $\{0, 1\}$ and modulo 2 operations. Table 6.1 shows the addition and multiplication results for GF(2^1), that are basically XOR and AND functions.

Table 6.1. Addition and multiplication for GF(2)

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Table 6.2. Polynomial mapping and multiplication table for GF(8). Primitive: $x^3 + x + 1$.

Element	Polynomial	Binary Repr.
0	0	000
1	1	001
A	x	010
B	$x + 1$	011
C	x^2	100
D	$x^2 + 1$	101
E	$x^2 + x$	110
F	$x^2 + x + 1$	111

×	0	1	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0
1	0	1	A	B	C	D	E	F
A	0	A	C	E	B	1	F	D
B	0	B	E	D	F	C	1	A
C	0	C	B	F	E	A	D	1
D	0	D	1	C	A	F	B	E
E	0	E	F	1	D	B	A	C
F	0	F	D	A	1	E	C	B

However, when $m > 1$, modulo operations between polynomials are required. A polynomial with degree up to $m - 1$ can be associated to each element of a field GF(2^m). Its coefficients are elements of the field GF(2), that is 0 or 1, so each polynomial can be represented by a binary number composed by its own coefficients. In Table 6.2 we can see the polynomial mapping and the corresponding binary representation for the field GF(2^3). Its elements are eight: $\{0, 1, A, B, C, D, E, F\}$. This representation has as primitive polynomial $x^3 + x + 1$, which guarantees an efficient hardware implementation.

The results of multiplication in GF(2^3) are presented in Table 6.2. To obtain these results, it is necessary to execute the algorithm for multiplication of two polynomials $a(x)$ and $b(x)$ modulo an irreducible polynomial $p(x)$ (called primitive) reported in listing 6.3. It is called the Montgomery Multiplication Algorithm [93]. For GF(2^m) the primitive polynomial has degree equal to m. The algorithm can perform modular multiplication without requiring division, which would be very costly. The multiplication is performed by *sum-and-shift* of partial products, while the modulo operation is obtained by subtracting the irreducible polynomial whenever the degree of the intermediate result gets equal to m. The $a_i \cdot b(x)$ term is either equal to 0 or to $b(x)$, when $a_i = 0$ and $a_i = 1$ respectively. So one coefficient of $a(x)$ at a time is multiplied with all the coefficients of b(x) (without taking into account any carry). Then the current result is shifted left (multiplying by x) before adding the new partial result.

Listing 6.3. Montgomery multiplication algorithm.

```
r(x) := 0
for i = m−1 downto 0 do
    r(x) := x*r(x) + a_i*b(x)
    if degree(r(x)) = m then r(x) := r(x)−p(x)
return r(x)
```

6.2.1.1 Galois Field Multiplier scheme

The RTL circuit that implements the Montgomery algorithm is shown in Figure 6.6 for GF(2^4). 1-bit registers are exploited to hold inputs and partial results (so they are used to memorize the state of previous loop of the algorithm), while the × and + symbols represent multiplication and addition.

The various steps of the algorithm can be associated to elements of the circuit. The Shift operation ($\mathbf{x} \cdot \mathbf{r}(\mathbf{x})$) is implemented with a 1-bit shift register toward the MSB. The partial product ($\mathbf{a_i} \cdot \mathbf{b}(\mathbf{x})$) is implemented with m bit-wise multiplications with AND gates in the upper part of the circuit. Notice that data $a(x)$ has to be fed serially, while data $b(x)$ is a parallel input. The Intermediate result ($\mathbf{r}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{r}(\mathbf{x}) + \mathbf{a_i} \cdot \mathbf{b}(\mathbf{x})$) is obtained with 4 bit-wise additions implemented as three-input XOR gates. Finally the modulo operation executed with subtraction ($\mathbf{if\ degree}(\mathbf{r}(\mathbf{x})) = \mathbf{m} \rightarrow \mathbf{r}(\mathbf{x}) = \mathbf{r}(\mathbf{x}) + \mathbf{p}(\mathbf{x})$) is implemented with a bottom row of AND gates. These are used to input $p(x)$ or 0 to the XOR gate depending if the degree of result is m or not, respectively.

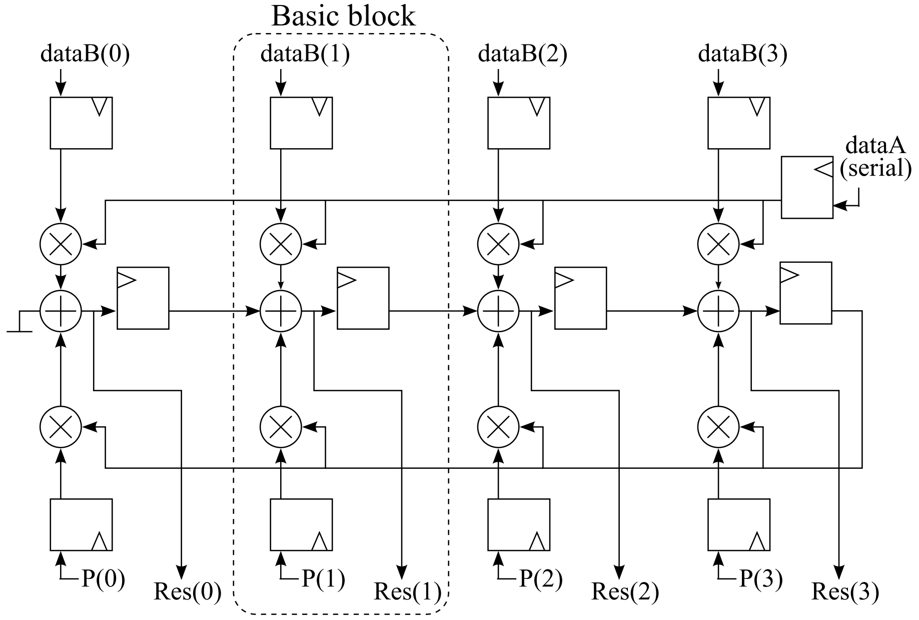


Figure 6.6. Scheme of a 4-bit bit-serial Galois Field Multiplier (GF(2^4)).

In Figure 6.6 the systolic array organization is evident: the Processing Element (PE) is evidenced with a dashed line, and multiple PEs are used in a chain to create the GFM. A N-bit GFM requires N almost identical basic blocks: The only exception are the first and last which are slightly different from the others. Simply connecting a different number of thee blocks it is possible to obtain any parallelism. Therefore a generalized GFM can be designed defining only three blocks, which will be named hereinafter *first*, *central* and *last*. This characteristic will be valid for any GFM implementation explored throughout the whole work.

Now that the circuit to be used for this example has been thoroughly described, in next paragraph the three implementations (CMOS, NML and ME-NML) are presented, and an overview of the various results obtained is reported in paragraph 6.2.5.

6.2.2 CMOS Implementation

For CMOS implementation, the scheme in Figure 6.6 has been modified into the one in Figure 6.7 to make it fully pipelined. This is necessary because without the pipeline, *dataA* and feedback propagation would have too long critical paths, as these critical paths grow proportionally to the circuit parallelism. The full pipeline guarantees a constant critical path for any parallelism leading to a greater throughput, but requiring additional registers that will have an impact on circuit area. Moreover, the pipelined version would be more similar to the ME-NML circuit, thus making the comparison more appropriate. Indeed, the scheme in Figure 6.7 will be the starting point to design the ME-NML version of the Galois Field Multiplier (GFM).

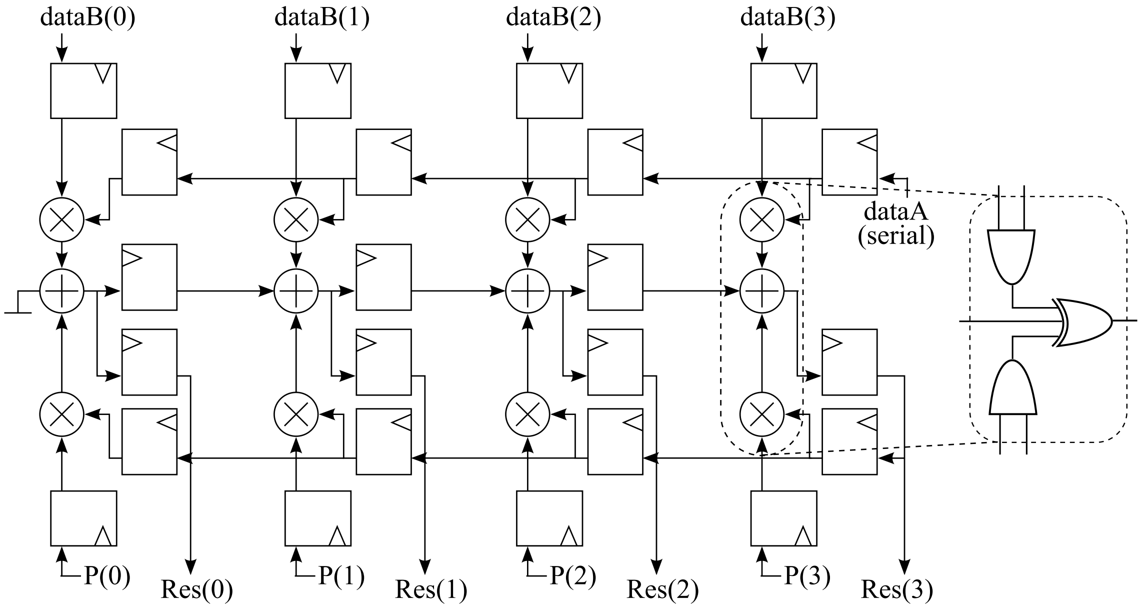


Figure 6.7. Scheme of the 4-bit fully pipelined Galois Field Multiplier. [27]

Because of this strong modularity, once defined the three basic blocks (first, central, last), it is straightforward to create a GFM with any parallelism just by tuning the number of central blocks ($N_{bit} - 2$ central blocks). For example a 4-bit multiplier, like in Figure 6.6, is composed of 2 central blocks. Increasing the parallelism the circuit layout will simply grow horizontally with more central blocks.

The generalized N-bit CMOS GFM has been described with VHDL language. The top entity, called `Galois_Multiplier`, instantiates N `basic_block` components. The `basic_block` has slightly different configurations, depending on its position within the circuit: first, last or center. This exact organization has been used also for the two NML implementations.

Now it is necessary to provide some additional information concerning the timing characteristic of this circuit. GFM is a sequential circuit, in which input *dataA* is provided serially. When *dataA(n)* must be provided, the result of the computation done with *dataA(n-1)* must be available at the rightmost XOR gate. The delay between the input of one bit of *dataA* and the corresponding usage of the computed result is 2 clock cycles. Hence a new input can be given every 2 clock cycles. Therefore the overall time for *dataA* to be fed in input is $2N \cdot T_{clk}$, leading to a throughput of $1/(2N \cdot T_{clk})$.

From a deepen timing analysis, there are three principal issues that derive from the required protocol:

- There is an unused clock cycle between a *dataA* bit and the next. This means that meaningful inputs are fed only for half of the time, so at any time half of the registers in the circuit would contain useless data.
- It is not possible to supply all bits of *dataB* simultaneously. The same is true for P and to acquire *Res*.
- To guarantee a continuous data flow, the inputs of an operation are fed right after the ones from the previous one. Therefore the new operation starts while the previous one is still processing. The first partial product has to be summed to 0, so the central shift register would be required to contain zero when the new data arrives. However it would still be carrying the final result from the previous operation.

.

These issues can be addressed with the following adopted solutions, that are valid also in the case of the two nanomagnetic implementations:

- **Interleaving:** To have a continuous flow of input data multiple non correlated sets of operations can be executed in parallel, so that the delay time between an input and the next is filled with other operations (this has been thoroughly described in Section 3.2).
- **Preskew and deskew networks:** A full set of additional registers must be added to the multiplier's body, in order to form preskew (for *dataB* and P) and deskew (for *Res*) networks.
- **Shift Register Reset:** Each register of the central row has to be reset (set to $'0'$) when data from a new operation arrives. Since in that moment it will contain the final result of the previous operation, such result will be erased. Therefore a line of additional registers, with the same input as the shift registers, is added right below. In this way the final result can be preserved, allowing to execute a continuous flow of operations. The reset of the shift register is applied in the same way as *dataB* is fed to the circuit. A single clock cycle reset is applied to each register when a new data is fed to its correspondent *dataB* register.

6.2.3 NML Implementation

Once implemented in CMOS, the GFM can now be designed in classic NML and ME-NML in order to provide a timing, area and power comparison. In this section we analyze the Magnetic clock implementation (we will refer to this technology as “NML”, while the MagnetoElstic one will be “ME-NML”).

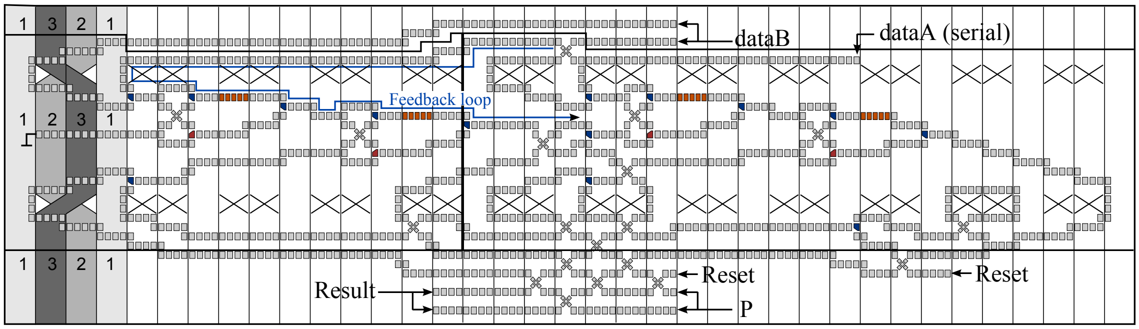


Figure 6.8. The 2-bit Magnetic NML Galois Multiplier, comprehensive of preskew and deskew networks. [27]

The snake-clock configuration for NML leads to a peculiar circuit organizations. Nevertheless, also with this technology it is possible to define a Systolic Array like structure.

We enclose the drawing of the 2-bit Magnetic NML Galois Multiplier including the synchronization networks in Figure 6.8. The circuit body (central stripe) i separated by the preskew/deskew networks (top and bottom).

A small area on the left in Figure 6.8 shows the exact layout of snake-clock wires and the signal propagation directions, while in the rest of the drawing the forbidden areas are simply marked by black crosses.

Notice also the feedback critical path for this implementation, represented in blue. Its length is 30 clock zones, which correspond to 10 clock cycles, since the snake-clock is a 3-phase clocking system. So, a new bit of *dataA* can be sent to the circuit every 10 clock cycles. The basic block depth is instead equal to 15 clock zones (5 clock periods), so that will be the delay between bits of *dataB*, *P* and *Res*. So the throughput can be computed considering that 10 clock cycles are necessary to provide a new *dataA* signal and 15 clock cycles are necessary for each block computation (so for each bit). For the 2-bit implementation, throughput will be $1/(40 \cdot T_{clk})$. However it is possible to interleave two operations to increase the throughput, that will be in general $1/(2N \cdot T_{clk})$.

6.2.4 ME-NML Implementation

We have described till now the CMOS and NML implementation of the Galois Field Multiplier, showing how much different are the two circuits in this technology. Now we consider the MagnetoElastic NML implementation. This is actually the first approach to the design of ME-NML circuits ever done, and it is particularly interesting because it takes into account the technological and physical constraints of this newly proposed technology. The first part of this Section is devoted to the design of the circuit in ME-NML technology, while in the second part we will focus on the VHDL description and circuit simulation of the circuit.

The starting point for the circuit design in ME-NML is the RTL circuit shown in Figure 6.7. Considering ME-NML technology, the register function is embedded in clock zones and magnets memory property. So, we need to consider only the combinational logic. Since the only available ports are AND, OR and Inverter the 3-inputs XOR has been realized as in Figure 6.9.

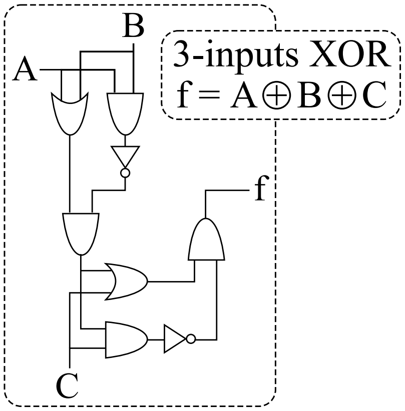


Figure 6.9. 3-inputs XOR function constructed with AND, OR and Inverter gates. [27]

We start our design considering the three different Processing Elements identified in CMOS circuit. These represent our basic blocks. All of them contain two AND and one XOR gates, plus a certain number of registers. At the time of writing there is not an automatic tool for ME-NML circuits, so the design has been done manually trying to identify the optimum solution and guaranteeing modularity. Through several steps of manual design and optimization, the final basic blocks for the GFM were designed as reported in Figure 6.10. The in/out signals for each block are indicated for an easier comparison with the CMOS circuit.

The reset network is not shown for CMOS, but its functioning was explained in paragraph 6.2.2. In the ME-NML implementation the reset (*rst*) is treated just like any other signals. It is applied to the signal (*PEin*) that propagates the temporary result from a block to the next one. The reset is obtained through an AND gate with as inputs *PEin* and the reset signal itself. The same is true for the reset applied to the feedback wire in the *Last* block (bottom-right corner).

For the sake of clarity each ME-NML cell in the picture has no electrodes, and there is no vertical separation between cells. The cells color is used to identify the

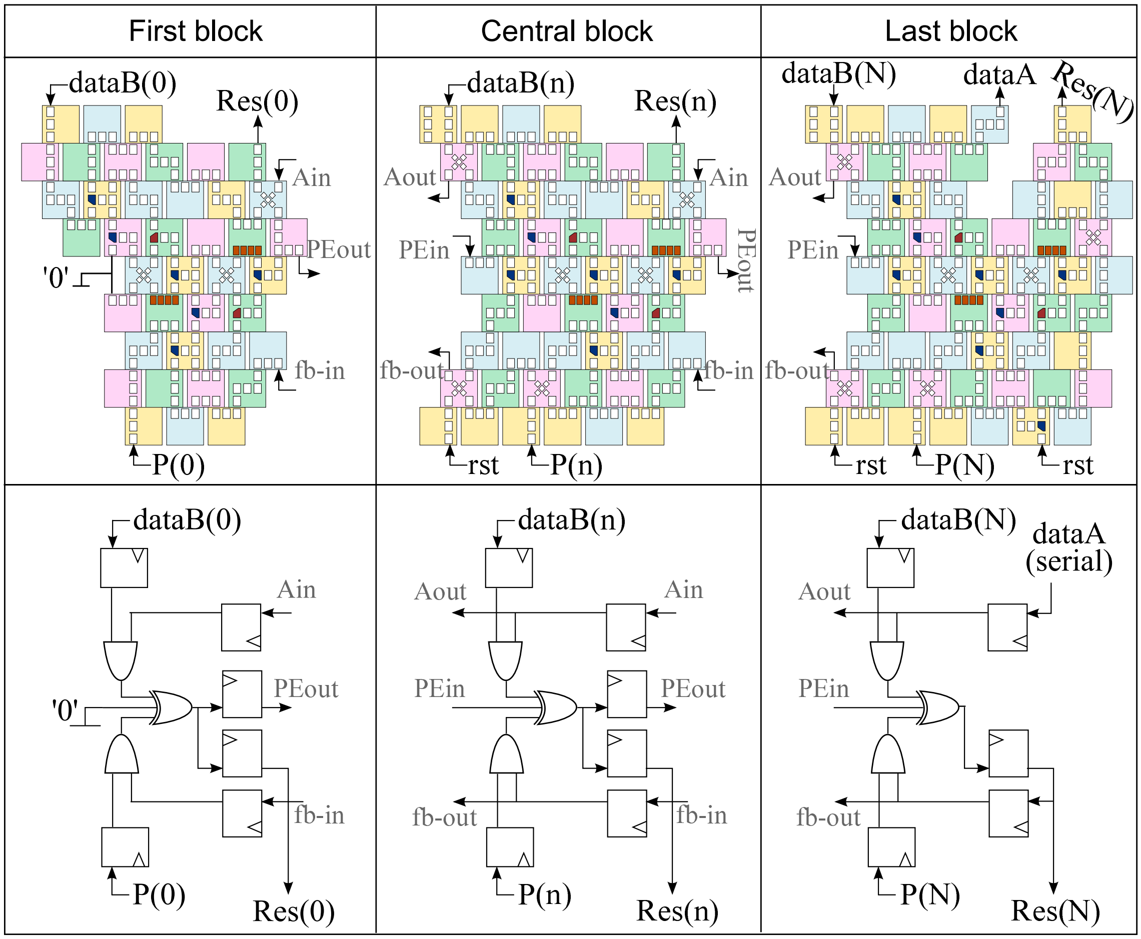


Figure 6.10. Basic blocks of the GFM. ME-NML blocks on top are matched with the correspondent CMOS blocks.

clock phase: yellow for phase 1, pink for phase 2, light blue for phase 3, green for phase 4. A N-bit multiplier requires N adjacent blocks: 1 *First* block, $N-2$ *Central* blocks, 1 *Last* block. Notice that the right border of a general n block has the same shape as the left border of the *n+1* block; this was carefully achieved by design to have the maximum modularity. We can now describe an complete ME-NML circuit.

In Figure 6.11 the basic blocks have been assembled to compose the 4-bit GFM. The timing propriety of the circuit is highly affected by the delay of single ME-NML cells. For example, a feedback path is highlighted in the drawing: It is 6 clock cycles long, that is the time for crossing 24 ME-NML cells (simply counting the number of cells crossed by that signal). The delay between *dataA* bits has to correspond to this critical path length. This delay is much longer compared to the CMOS circuit, because of the intrinsic pipeline nature of NML.

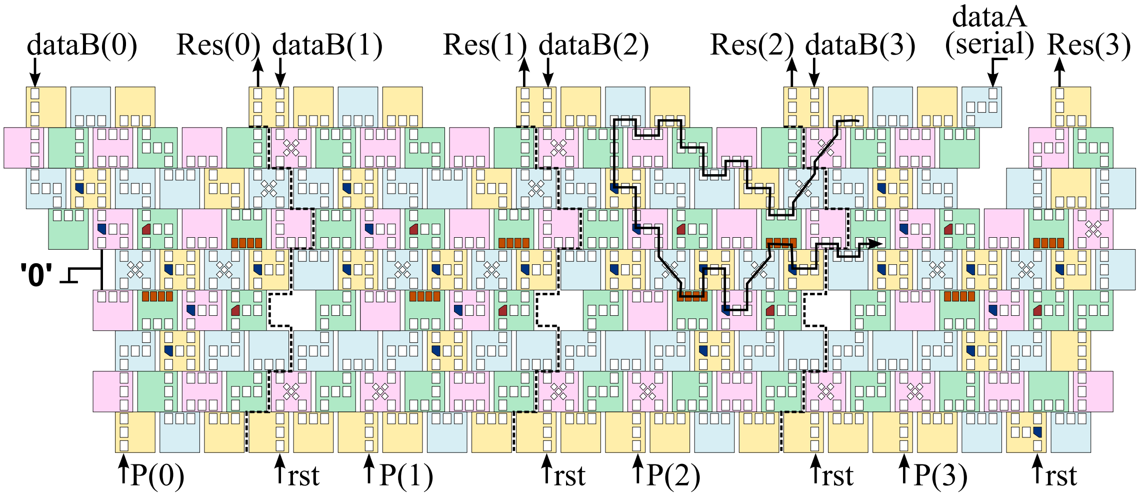


Figure 6.11. Magnetoelastic NML implementation of a 4-bit Galois Multiplier.

One aspect described in Section 6.2.2 is the necessity to introduce a preskewdeskew network, so that all bits of *dataB*, *P* and *Res* can be served/acquired simultaneously. The additional circuitry has been designed and added to the GFM body, as showin Figure 6.12. This image is divided into three horizontal stripes. The central one is the GFM's body (same of Figure 6.11), while the top and bottom ones are the required synchronization networks. The preskew/deskew circuitry can also be decomposed in basic blocks and described with VHDL generically for any number of bits, even though they are not as regular as the central section. They do not contain any logic, only interconnections.

To verify the circuit functioning and to evaluate performances, the ME-NML Galois Multiplier has been described with the RTL model presented Paragraph 6.1.2. The top entity `Galois_Multiplier` instantiates and connects the required number of basic blocks (Figure 6.10), which are defined by another entity called `Base_Blocks`.

The timing protocol is very similar to the CMOS case but with 3 times longer delays, because the critical path is 6 clock cycles instead of 2 of the CMOS version. The result is a 6 clock periods delay between *dataA* bits, and 3 clock cycles of delay for the others: *dataB*, *P*, *Res*. To reach the maximum throughput 6 uncorrelated operations should be interleaved.

The total time to execute one operation is given by $4 \times 6T_{clk}$ to provide inputs and a same amount of time to complete the operations. So without interleaving the throughput would be $1/(48 \cdot T_{clk})$. Applying interleaving it is instead possible to achieve a more interesting $1/(8 \cdot T_{clk})$. Generalizing for any number of bit, adopting interleaving it is possible to have a throughput of $1/(2N \cdot T_{clk}$.

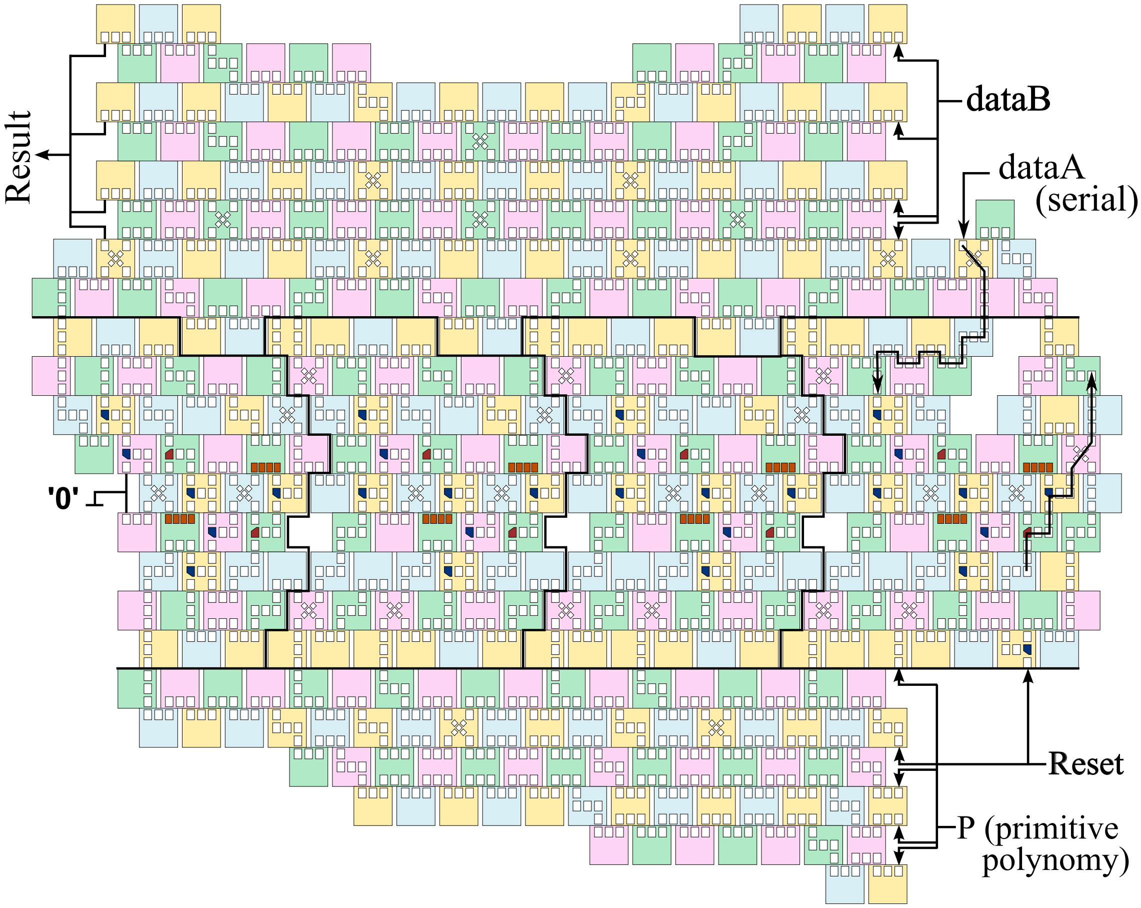


Figure 6.12. ME-NML Galois Multiplier with additional preskew and deskew networks.

6.2.5 Results

This chapter is devoted to performance evaluation of the three GFM implementations in terms of occupied area and power consumption. First of all the results produced for each technology are discussed separately, providing details on their evaluation. Then the three versions are put together, presenting an accurate comparison. NML circuits are handled keeping into account technological constraints and the exact details on the clock network chosen.

The outcomes demonstrate the effectiveness of ME-NML for power and area performances. For each implementation the results are evaluated for 4 to 64 bits, both with and without the preskew/deskew circuitry for input and outputs signals. The additional synchronization networks are a factor generally neglected in literature, even though they bring a significant increase in circuit area.

6.2.5.1 CMOS Results

The CMOS version of the GFM has been presented in previous Section 6.2.2. All the results are extracted after finalizing the physical layout through Cadence Encounter 13.1. For the place&route we exploited a low power CMOS 28 nm FDSOI standard cell library, with the following working conditions: $V = 0.9V$, $T = 25°C$.

Table 6.3 contains all the results of area occupation for the CMOS GFM. The same results are plot in Figure 6.13 to make them more understandable at a glance.

Table 6.3. Area occupation of CMOS GFM both with and without synchronization circuitry.

Circuit Area (μm^2)	Number of bits				
	4	8	16	32	64
Without Synch Network	154,6	320,6	646,9	1299,7	2605,3
With Synch Network	262,3	810,1	2745,2	9972,5	37856,0
Interconnection Overhead	1,7	2,5	4,2	7,7	14,5

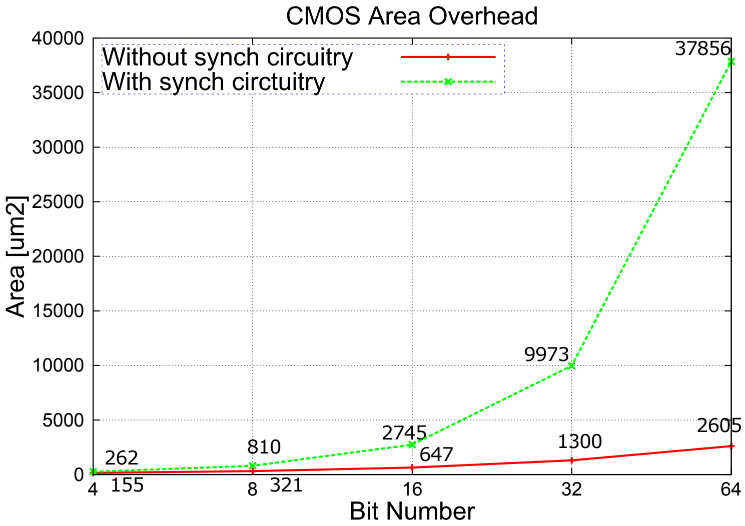


Figure 6.13. Comparison of area occupation for the CMOS GFM both with and without synchronization circuitry.

The interconnection overhead is simply evaluated as the ratio between values with and without preskew/deskew networks. The impact of the additional circuitry goes from 1.7 (4 bit) to 14.5 (64 bit). In other words it means that it is necessary to increase the area of the 70% for the 4 bit circuit, while to increase it more than 13 times for the 64 bit circuit.

The post-route power estimation gave the results in Table 6.4. The increase in losses, due to interconnection overhead, is also evident in Figure 6.14. The additional circuitry affects the power consumption less than the area occupation, reaching a maximum increase of 12.5 times with respect to the power required by the GFM body itself.

Table 6.4. Power consumption of the CMOS GFM both with and without synchronization circuitry.

Power Consumption (μW)	Number of bits				
	4	8	16	32	64
Without Synch Network	14,30	33,63	68,62	140,24	294,03
With Synch Network	23,72	81,37	278,82	977,21	3687,80
Interconnection Overhead	1,7	2,4	4,1	7,0	12,5

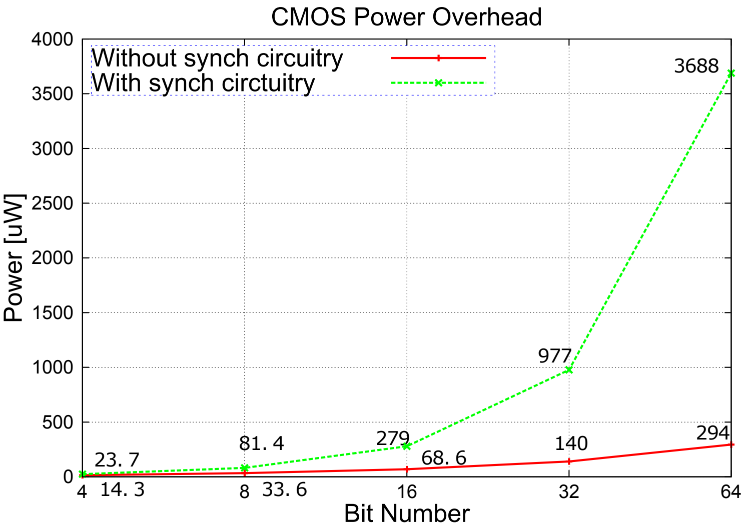


Figure 6.14. Comparison of power consumption for the CMOS GFM both with and without synchronization circuitry.

6.2.5.2 NML Results

In this paragraph we discuss how to evaluate the performance of Magnetic Clock NML circuits.

Table 6.5. Dimensions and number of magnets of the Magnetic NML GFM both with and without synchronization circuitry.

		Number of bits				
		4	8	16	32	64
W/O Synch	Number of magnets	1818	3678	7398	14838	29718
	Width (clock zones)	67	127	247	487	967
	Height (magnets)	24	24	24	24	24
With Synch	Number of magnets	3154	7388	18880	53960	172504
	Width (clock zones)	67	127	247	487	967
	Height (magnets)	40	56	88	152	280

The evaluation of area and power performances requires: the number of clock zones, the length of the clock zones (circuit height) and the total number of magnets. These values are first computed for each basic block and then put together to obtain results for each parallelism and with or without the upper and lower interconnections parts. The final results are presented in Table 6.5. The number of clock zones is given by the circuit horizontal width, while the circuit height is measured in terms of magnets.

Table 6.6. Area of the Magnetic NML GFM both with and without synchronization circuitry.

Circuit Area (μm^2)	Number of bits				
	4	8	16	32	64
Without Sync Network	57	107	209	411	817
With Synch Network	94	250	765	2610	9530
Interconn. overhead	1.7	2.3	3.7	6.3	11.7

The Magnetic Clock NML exploits $90 \times 60\, nm^2$ magnets with separation $Sep_{mag} = 20\, nm$. Horizontally each clock zone contains four magnets, therefore its width is $W_{zone} = 4 \cdot (W_{mag} + Sep_{mag}) = 320\, nm$. These data, together with those in Table 6.5, allow to evaluate the total area of magnets and the rectangle circumscribed to the circuit, shown in Table 6.6. Information on the preskew/deskew networks overhead are reported as well. This is the lowest among the three technologies considered. We will see that the interconnection overhead is the same for both area and power estimation. Figure 6.15 gives an idea of the GFM behavior increasing the number of bits, with and without the additional synchronization circuits.

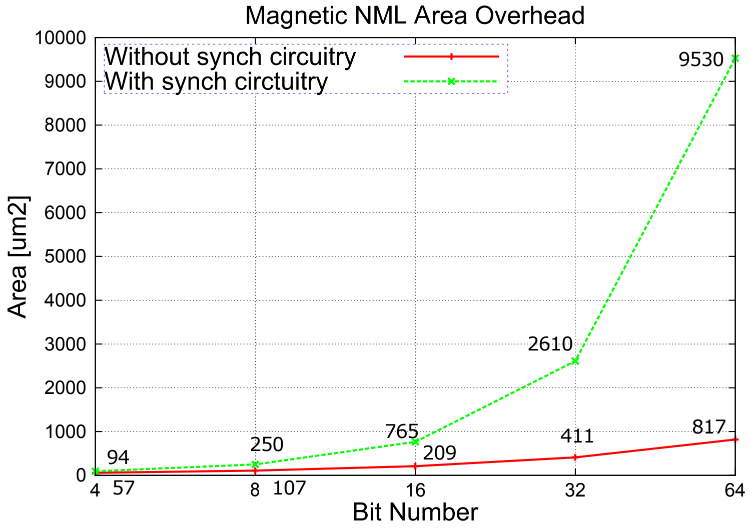


Figure 6.15. Comparison of area occupation for the Magnetic NML GFM both with and without synchronization circuitry.

The power dissipation, as for the ME-NML, has two sources: magnets switching and clock wires. The average energy required by the switching of a single nanomagnet is equal to $\delta E = 30K_bT = 1.24 \cdot 10^{-19}J$, since an adiabatic switch has to be exploited. The switching energy is obtained multiplying this value for the total number of magnets. However the main contribution is due to the clock network losses, because the current needed to generate the magnetic field is very high: $I = 3\,mA$. The power consumption is therefore the dissipation of the current I flowing through a copper wire, which has resistivity $\rho = 16.8n\Omega \cdot m$. For each clock zone we consider a copper wire with width $W_{clk} = W_{zone} = 320\,nm$ and thickness of $T_{clk} = 400\,nm$, so its section is $S_{clk} = W_{clk} \cdot T_{clk}$. At any instant, only one third of the clock zones is active, since only one of the clock wires at a time is active. Summing the length H_{zone} of one third of the clock zones N_{zones_eff} we obtain the length $L_{clk} = N_{zones\_eff} \cdot H_{zone}$ to assign to the copper wire, that will model the clock dissipation of the whole circuit. The power consumption is derived from the following formula:

$$P = I^2 \cdot \rho \frac{L_{clk}}{S_{clk}}$$

The power consumption results are reported in Table 6.7. For further information on the Magnetic NML model refer to [94] and [27].

Table 6.7. Power of the Magnetic NML GFM, both with and without synchronization circuitry.

Power Consumption μW		Number of bits				
		4	8	16	32	64
W/O Synch	Magnets Switching	0.023	0.046	0.092	0.18	0.37
	Clock Wires	70	132	257	506	1010
	TOTAL	70	132	257	506	1010
With Synch	Magnets Switching	0.040	0.092	0.24	0.67	2.14
	Clock Wires	116	308	941	3210	11700
	TOTAL	116	308	942	3210	11700
Interconn. overhead		1.7	2.3	3.7	6.3	11.7

6.2.5.3 ME-NML Results

Finally we consider results in terms of Area and Power for the MagnetoElastic NML case. The methodology and equations to compute area occupation and power dissipation have been detailed in paragraph 6.1.2.3. The results for the GFM body are directly evaluated by the VHDL model. Total area and energy components are given as output of the top entity `Galois_Multiplier` during simulation.

Table 6.8. Number of magnets and cells of ME-NML GFM both with and without synchronization circuitry.

Magnets and Cells		Number of bits				
		4	8	16	32	64
W/O Synch	N of magnets	974	1990	4022	8086	16214
	N of cells	199	403	811	1627	3259
With synch	N of magnets	2007	6431	22287	82509	297710
	N of cells	427	1273	4117	14547	50235

First, the number of nanomagnets and cells are determined; they are listed in Table 6.8. The results concerning area occupation, both with and without the preskew/deskew circuits, are organized in Table 6.9 and plotted in Figure 6.16, where the interconnection overhead can be observed clearly. The overhead due to the upper and lower interconnections behaves similarly to the CMOS implementation: It grows quadratically with the number of bits, going from 2.1 (4 bit) to 15.4 (64 bit).

Table 6.9. Occupied area of ME-NML GFM both with and without synchronization circuitry.

Circuit Area		Number of bits				
(μm^2)		4	8	16	32	64
No	Magnets	3.2	6.5	13	26	53
Synch	Cells	14	29	58	116	233
With	Magnets	6.5	21	72	268	968
synch	Cells	31	91	294	1040	3590
Interc. overhead		2.1	3.2	5.1	8.9	15.4

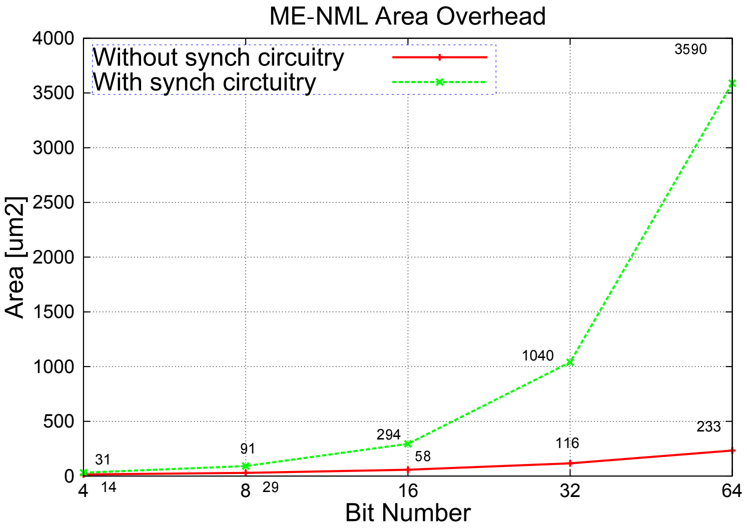


Figure 6.16. Comparison of area occupation for the ME-NML GFM both with and without synchronization circuitry.

The power consumption is proportional to the area occupation, because both measures have the number of cells as factor. Therefore the interconnections overhead is the same as for the occupied area. The results are reported in Table 6.10. It is worth underlining that the magnets switching energy is negligible (20 times smaller) compared the clock network dissipation.

6.2.5.4 Results Comparison

Now that all the results have been presented, we compare the performances of the three implementations in terms of area and power. This is where we can finally

Table 6.10. Power consumption of ME-NML GFM both with and without synchronization circuitry.

Power Consumption (μW)	Number of bits				
	4	8	16	32	64
Without Synch Network	1,28	2,60	5,22	10,5	21,0
With Synch Network	2,74	8,21	26,7	94,5	327,0
Interconnection Overhead	2,1	3,2	5,1	9,0	15,6

tag ME-NML as an interesting technological alternative to CMOS and where we can demonstrate that the MagnetoElastic clock allows to dramatically reduce power consumption with respect to the classic magnetic field clock.

First we consider the circuit without synchronization networks.

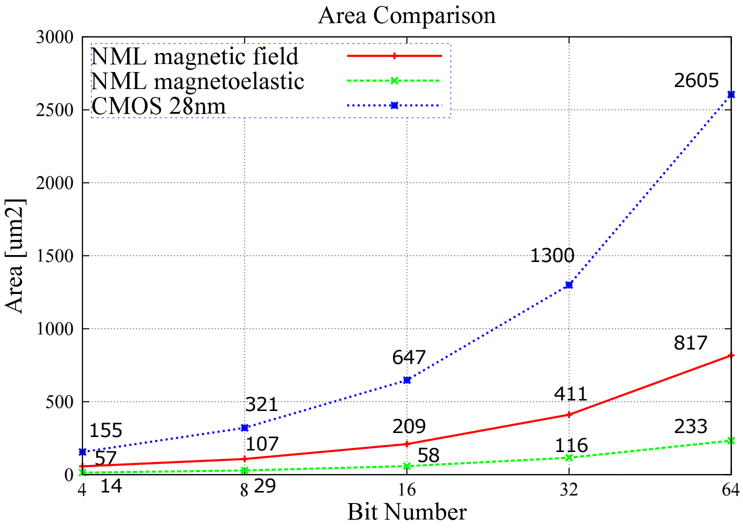


Figure 6.17. Area comparison between the three GFM implementations without synchronization networks.

Data on occupied area, without considering the additional networks, are plotted in Figure 6.17. Of course the area increases with the number of bits: It is interesting to notice that the CMOS implementation has the worst performance (blue line), while the smallest area is achieved with ME-NML circuit (green line). On average CMOS circuit is 11 times larger than ME-NML, Magnetic NML instead is 3.5-4.1 times bigger.

Figure 6.18 depicts instead the power comparison, still neglecting the upper and lower interconnections. The curves are similar to the graphs of the circuit area.

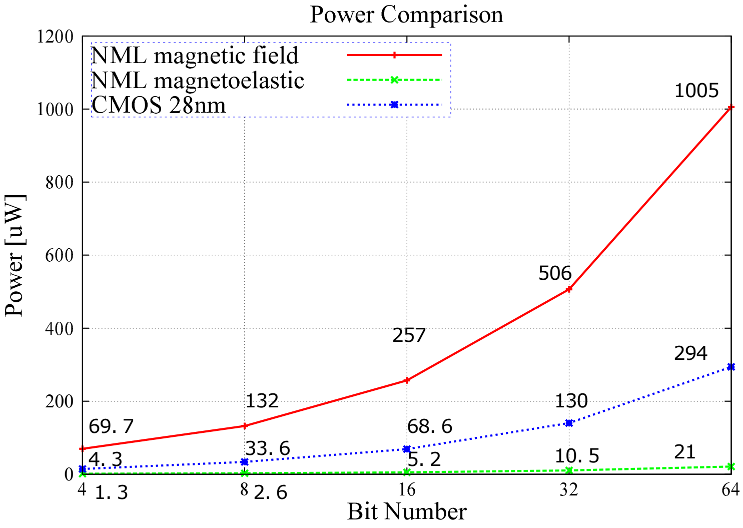


Figure 6.18. Power comparison between the three GFM implementations without synchronization networks.

However, while ME-NML confirms itself as the best technology, the Magnetic NML is now definitely the worst one. It is though what expected, as the Magnetic Clock network requires a very high current to generate the magnetic field. On average CMOS consumes 11-14 times more energy than ME-NML, Magnetic NML instead requires around 50 times more than ME-NML.

In the next graphs we take into account the upper and lower synchronization networks.

For what concerns the synchronization networks, it is important to state immediately that the circuit body only grows horizontally, while the upper and lower networks grow also vertically, hence they grow quadratically. This additional cost is often neglected in literature, even though such circuitry is essential to properly interface our module with others. This is a recurring problem of QCA circuits [19], because of their intrinsic pipeline nature.

Figure 6.19 shows the occupied area for the three GFM versions after adding the preskew/deskew modules. All the curves have similar trends. ME-NML and CMOS have respectively the best and worst performance, like when considering the area of the GFM's body only. However the additional interconnections have a slightly stronger impact on ME-NML than on the others. The ratio between technologies lowers to 8.5-10.5 for CMOS and 2.5-3.0 for Magnetic NML.

Figure 6.20 shows instead the power consumption for the three GFM versions after adding the preskew/deskew modules. Just like for the area, when considering the full circuit, ME-NML performance suffers more for the additional interconnections. However this does not weaken its leadership significantly. The ratios decrease

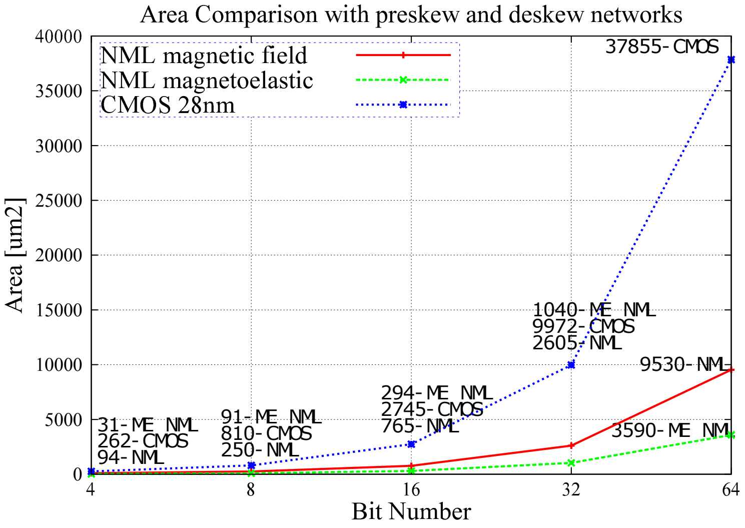


Figure 6.19. Area comparison between the three GFM implementations with synchronization networks.

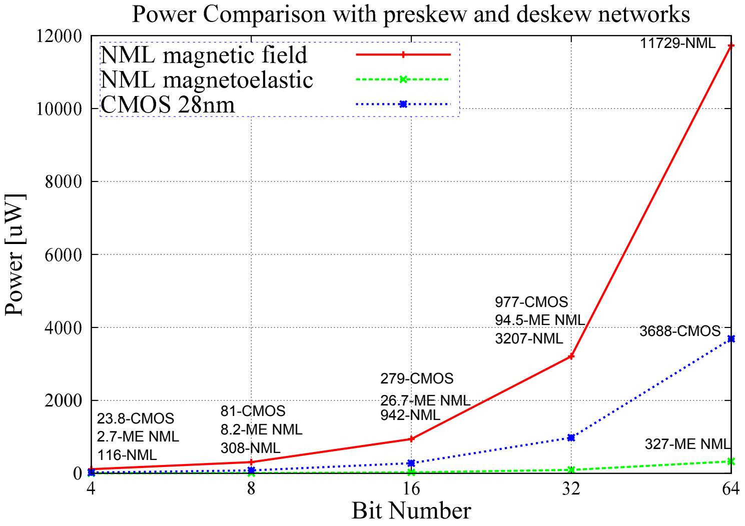


Figure 6.20. Power comparison between the three GFM implementations with synchronization networks.

to 8.7-11.3 for CMOS and 42-36 for Magnetic NML.

The final considerations are mostly three. First, the MagnetoElastic NML has confirmed its potentialities in this circuit example. With a proper architectural choice it leads to a great reduction of circuit area and power losses of the clock network, which was the unavoidable drawback of previous NML implementation.

Second, the synchronization networks have a huge impact on performances, thus it is imperative to take them into consideration when they are required. Third, even with these excellent results, NML technology is not meant as a replacement for CMOS technology, since its speed is intrinsically limited. The benefits of NML technology are bounded to circuit area and power consumption, together with its intrinsic memory ability. As already thoroughly described in Chapter 3, to address the low clock frequency of this technology it is necessary to adopt parallelization of tasks using massive parallel architectures such as square or octagonal Systolic Arrays.

6.3 Parallel and Serial Computation in ME-NML

Results of previous Section analysis, highlight that Magnetoelastic NML can overcome both Magnetic Clock NML and CMOS technologies in terms of circuit area and power consumption. The ME-NML implementation of the bit-serial Galois Multiplier, organized as a Systolic Array, is extremely compact and easily scalable. The Galois Field Multiplier (GFM) exploiting the Montgomery algorithm is interesting for its input rules: one input is provided serially to the circuit, while the other is fed in parallel. The architecture is parallel but it adopts this hybrid input approach with recurrence on the logic block (N loops on the logic core to achieve the final result, where N is the number of bits of inputs).

Generally CMOS circuits are designed as much parallel as possible to avoid recurrences (loops on same computational blocks) and increase performance. This is also advantageous because in CMOS additional signal lines to increase the parallelism of signals have an extremely low impact on the circuit, so the only circuit area increase is due to additional logic gates. This is definitively not true for NML technology. We have seen in Section 6.2 how interconnection networks produce an important area increase in NML and ME-NML.

The idea of this research path is to investigate what can be the best solutions for ME-NML: on the one hand parallel circuits can guarantee faster operations and it is not necessary to use the same logic block several times for one computation, but these may suffer from area devoted to the routing of signals; on the other hand serial circuits can guarantee smaller sizes and an interesting logic/routing area ratio, but they require several reiterations to achieve the final result.

So, our inquiry focuses on the dualism between serial and parallel structures, trying to determine which one of the approaches gets the best out of ME-NML. The case study chosen is a generalized Multiply Accumulate unit (MAC), which will be realized in three different versions: fully parallel, serial-parallel, fully serial. The serial-parallel solutions exploits the principle seen for the GFM, where one input is provided serially to the circuit while the other is fed in parallel.

The three generalized MAC are designed, modeled, simulated and compared in terms of area, power, throughput and latency. The MAC unit is composed by a multiplier, an adder and an accumulator: The main scheme is depicted in Figure 6.21.

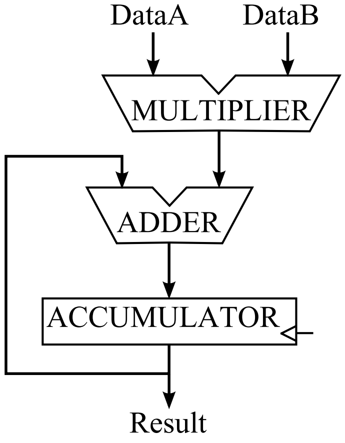


Figure 6.21. Multiply Accumulate unit scheme.

6.3.1 Parallel MAC Unit

The first implementation presented is a parallel version of the MAC unit. It is basically composed by a parallel multiplier and an adder with feedback. The accumulator function is instead embedded, as ME-NML is intrinsically pipelined. The array multiplier and the ripple carry adder (RCA) have been chosen as components of the parallel MAC, because they both have a Systolic Array architecture. They are composed of blocks that communicate only with their neighbors, avoiding as much as possible long interconnections and feedback.

6.3.1.1 Circuit Scheme

The scheme of the 4-bit Array Multiplier (left) and the 8-bit Ripple Carry Adder (right) are drawn in Figure 6.22, where *FA* and *HA* represent Full Adder and Half Adder. The two inputs *A* and *B* an the output *Res* are parallel. Consider a MAC unit with N_{bit} inputs *A* and *B*. The result of the N-bit multiplication is a $2N_{bit}$ number, therefore the adder will have $2N_{bit}$ inputs. Indeed in Figure 6.22 we have a 4-bit multiplier and a 8-bit adder.

Notice that the multiplier is basically a matrix of Full Adders, so it is two-dimensional and its area grows quadratically with the circuit parallelism. The Array Multiplier algorithm is the simplest one that follows step by step the handmade

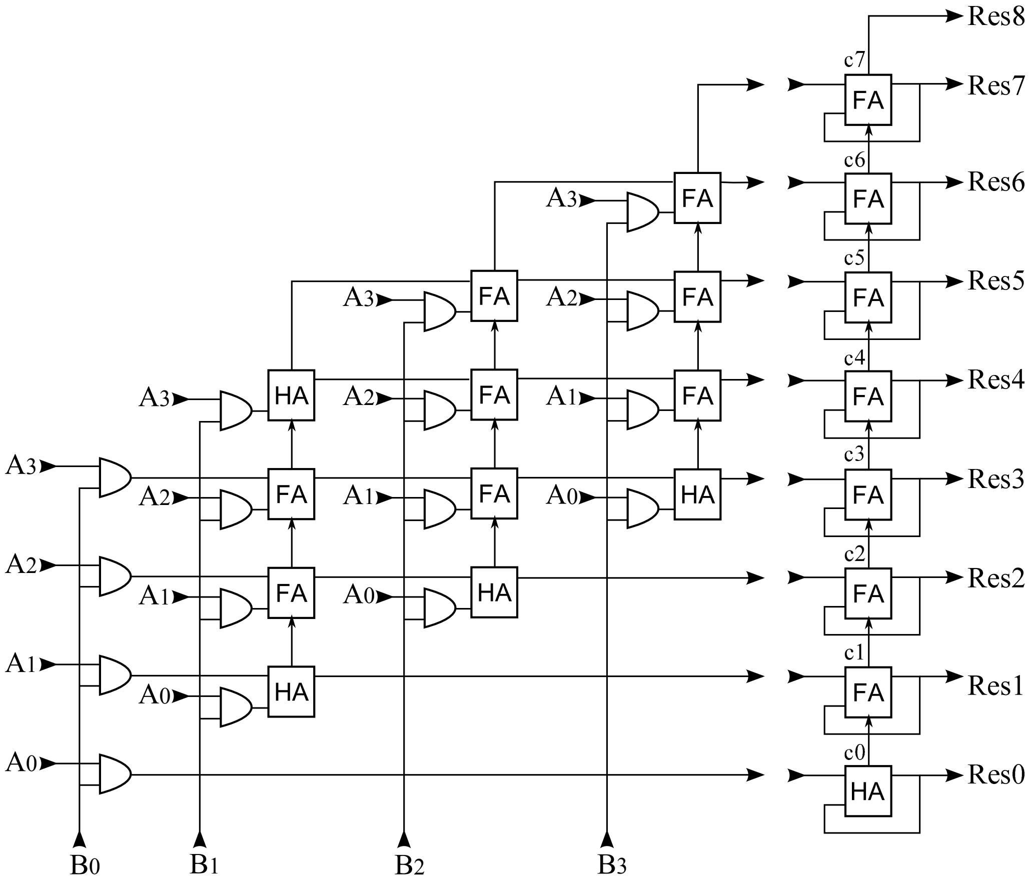


Figure 6.22. 4-bit MAC scheme. Array Multiplier on the left and Ripple Carry Adder on the right. [27]

multiplication. Partial products are shifted and added to an intermediate result. Each AND ports column in the drawing evaluates a partial product, which is then added to the intermediate result by the Full Adders. Moreover every AND column has a 1-bit shift with respect to the previous column to assure the proper alignment of the partial products sum. The final product goes to the RCA, that sums it with the accumulator value (stored in the RCA feedback). Within the adder the carry propagates vertically from one FA to the next.

The circuit arrangement and orientation imitates the ME-NML implementation that will be presented shortly, to guarantee an easy visual comparison between the two circuits. However there are some differences. The scheme in Figure 6.22 does not have any pipeline stage, while the ME-NML MAC will be fully pipelined. To have the two circuits more similar, each row and column in Figure 6.22 should represent a pipeline stage.

6.3.1.2 ME-NML Implementation

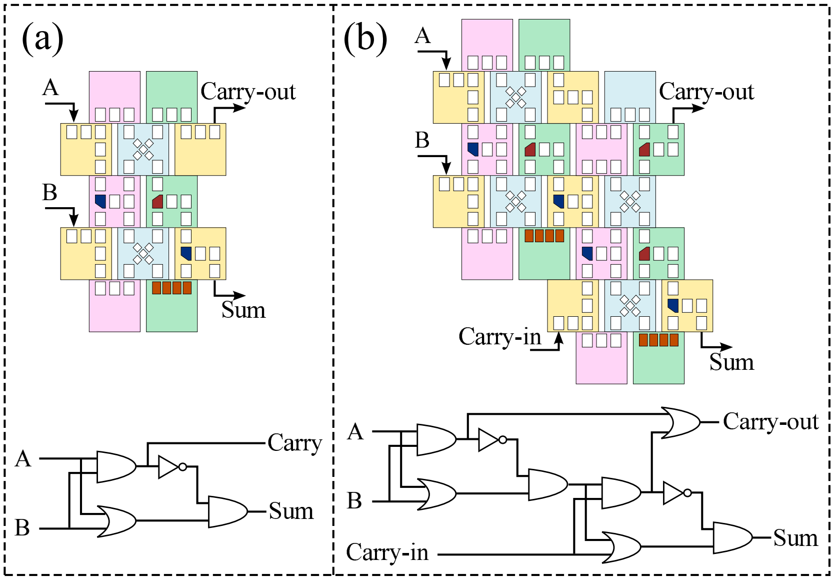


Figure 6.23. Half Adder and Full Adder realized with both ME-NML and CMOS technologies. (A) Half Adder. (B) Full Adder. [27]

The basic modules of the MAC unit are Full Adder (FA) and Half Adder (HA). These modules can be arranged in many different ways. One version of the Half Adder has already been depicted in Figure 6.5. Here we present the FA and HA that have been exploited to create the parallel MAC. Figure 6.23(a) shows the ME-NML HA together with its CMOS scheme, Figure 6.23(b) represents instead the FA.

Basically the whole parallel MAC has been designed exploiting these blocks only, providing them with a properly routed network of interconnections. The final Parallel MAC circuit in ME-NML is shown in Figure 6.24.

The VHDL model and simulation procedure is the same of the Galois Multiplier. The generic parallel MAC has been modeled with the usual components' hierarchy and tested up to 64 bits. Using a Matlab script, for each parallelism to be tested, we created a set of 1000 random inputs and related results. The VHDL testbench acquires those random inputs and writes the simulation results into another file, which can be compared to the expected results.

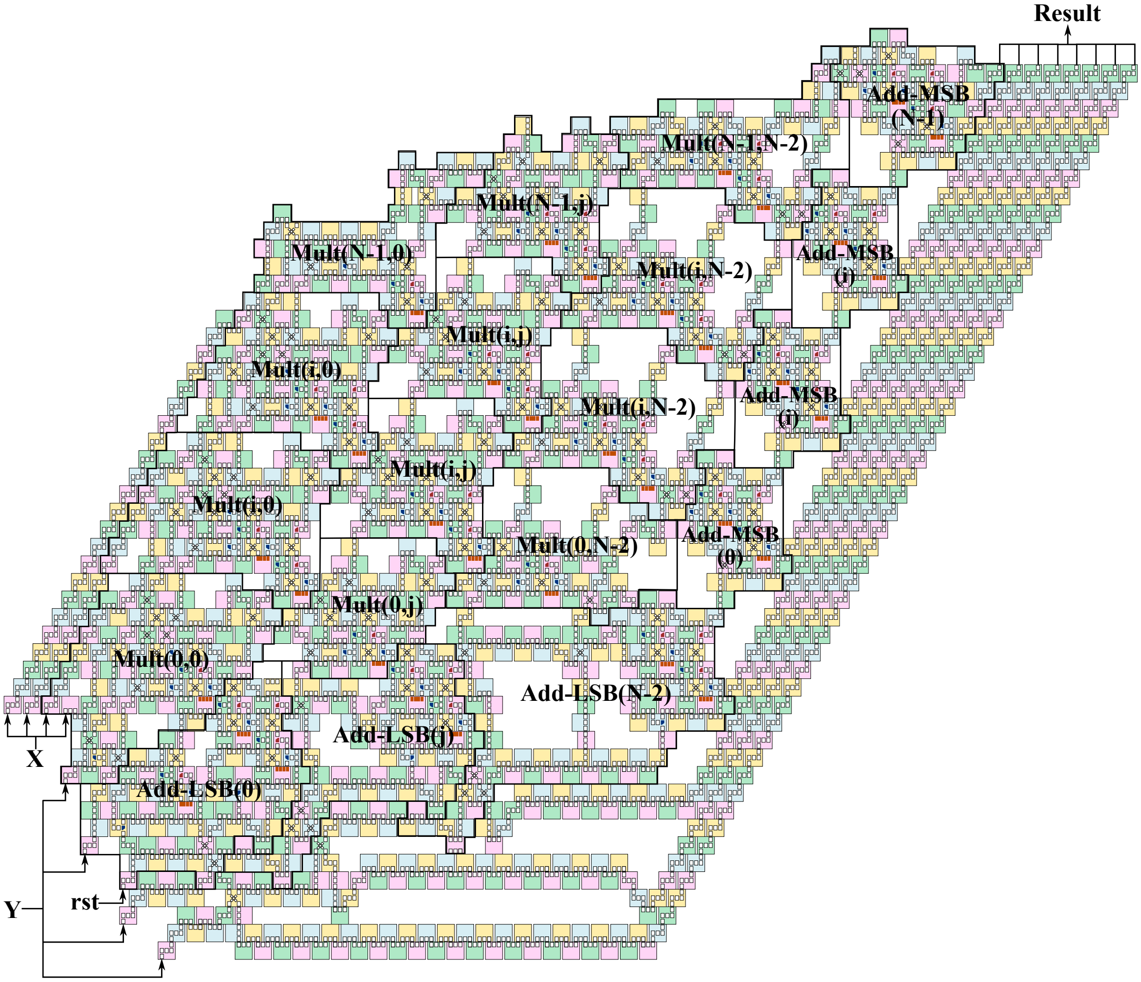


Figure 6.24. 4-bit parallel ME-NML MAC unit. Labels identify the base blocks of Multiplier and Adder. [27]

6.3.1.3 Timing Analysis

The Array Multiplier is composed by a matrix of $N \times (N-1)$ base blocks. Increasing the circuit parallelism the matrix will get bigger, affecting the overall circuit latency. On the other hand for any number of bits the RCA will always be only one column thick, having a constant impact on the latency. Every block of the Multiplier requires 5 clock cycles to be crossed horizontally (signal x) and 2 vertically (signal y in Figure 6.25). Therefore the inputs (bottom-left) need $(5(N-1)+2N+5) \cdot T_{clk}$ to reach the result. The additional 5 clock cycles are fixed and mainly refer to the time needed to pass through the RCA. The critical paths are highlighted with blue in Figure 6.25 Mult(i,i).

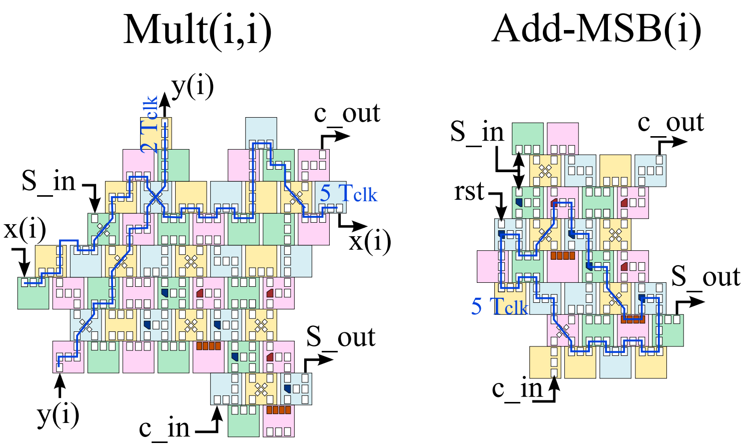


Figure 6.25. Critical paths of the parallel MAC: 1) Critical paths of multiplier's base blocks. 2) Feedback loop of adder's base blocks. [27]

In a MAC each multiplication result is added to the value in the accumulator. Since each block of the adder has a 5 clock long feedback loop (Figure 6.25 Add-MSB(i)), operations cannot be fed to the MAC in a continuous flow. Two operations must be fed with 5 cycles of delay in order to be added to each other. Therefore to reach the maximum throughput 5 uncorrelated operations should be interleaved. With the interleaving, throughput is of one operation per clock cycle. All information regarding timing performance of the parallel MAC are listed in Table 6.11. This table indicates the maximum achievable throughput, when interleaving technique is exploited.

Table 6.11. Timing performance of the Parallel MAC

N bit	Interleaving	Throughput	Latency: 1st Result out
4	5 op.	$1/(T_{clk})$	$28T_{clk}$
8	5 op.	$1/(T_{clk})$	$56T_{clk}$
N	5 op.	$1/(T_{clk})$	$5(N-1)+2N+5\cdot T_{clk}$

6.3.2 Serial-Parallel MAC Unit

The parallel MAC described in the previous section has a 2D layout. The idea for the second version of the MAC is to create a circuit organized as a 1D array of elements. It is called "serial-parallel MAC", because it has serial inputs and parallel output. While the design of the parallel MAC was trivial, in this case it was not possible to design a simple circuit able to keep up with the parallel implementation. The circuit's body itself has excellent characteristics, but its input/output protocol is unique, it would be very difficult to interface it directly with other devices. Moreover additional interconnections are required, as in the case of the Galois Multiplier, terribly spoiling the performances.

6.3.2.1 Circuit scheme

The scheme in Figure 6.26 is the body of the 4-bit serial-parallel MAC, but to have serial inputs and parallel output it requires additional registers. The circuit is composed of $2N_{bit}$ 1-bit adders. Each adder has its own feedback, so that the array of FAs can work as an accumulator. A reset signal allows to reset the accumulator whenever necessary. As usual the scheme is fully pipelined to imitate ME-NML behavior. The timing protocol follows the handmade multiplication procedure, where the N partial products are evaluated one by one and summed together.

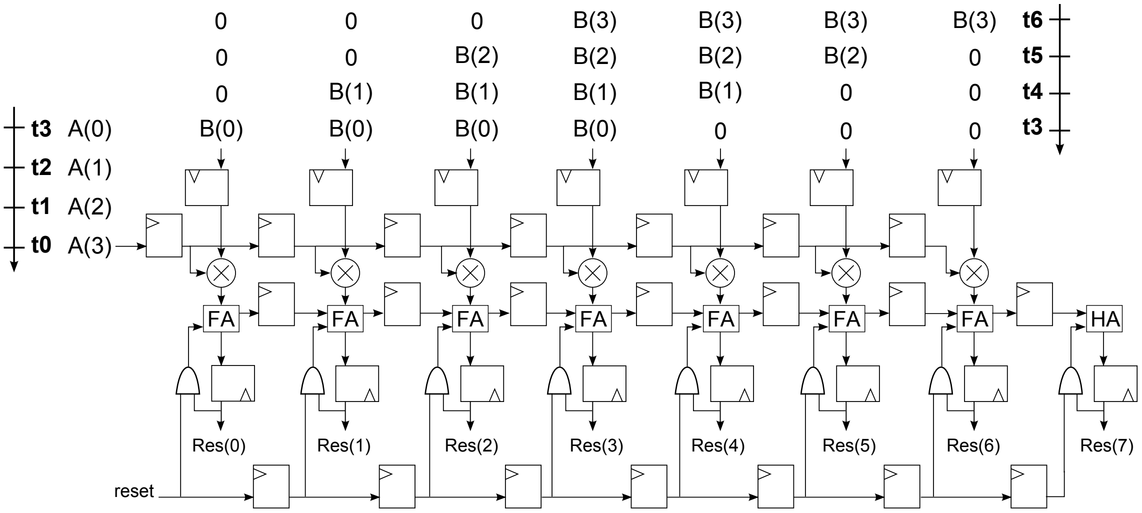


Figure 6.26. Body of the 4-bit serial-parallel MAC. [27]

Figure 6.26 also shows a timeline that explains the input protocol to execute a 4-bit operation. At t_0 A is fed serially starting from the MSB. After all 4 bits of A are fed in the shift register, they are multiplied bitwise with *B(0)*, which has been applied in the meantime. This gives the first 4-bit partial product which goes in the first four Full Adders, while the remaining three Adders receive $'0'$. Data B always has $N - 1 = 3$ bits equal to $'0'$, because partial products have a N-bit width. After the first partial product is evaluated data A bits shift to the right and are multiplied with data *B(1)* which arrives right after *B(0)* but shifted of one step toward the MSB (right). In this way the second partial product is correctly aligned to the first one, so that they are added properly.

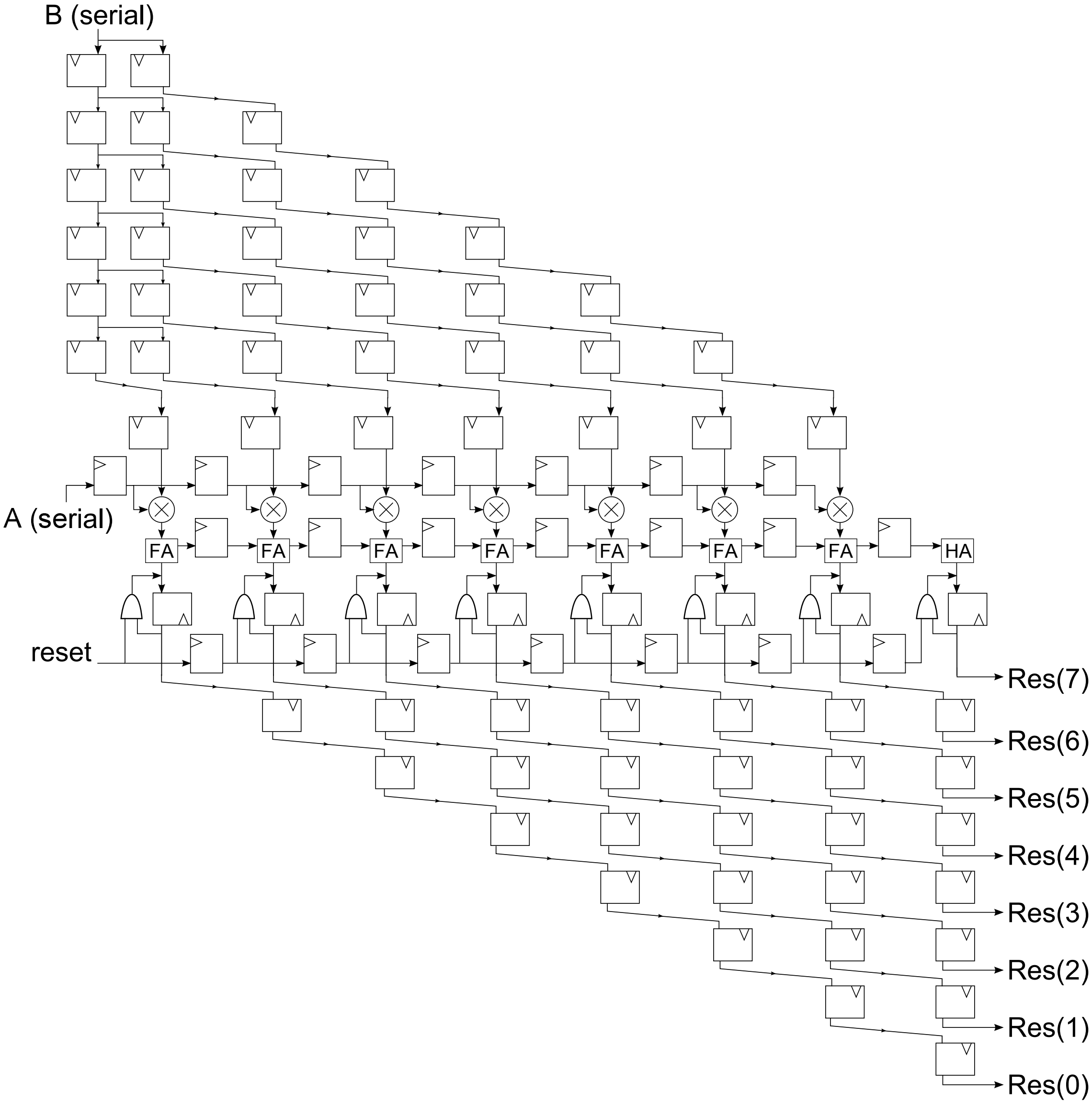


Figure 6.27. Full scheme of the 4-bit serial-parallel MAC. [27]

Evaluating all the partial products only requires N clock cycles. But another N cycles have to be spent feeding $'0s'$ to prepare the circuit for the next operation. The Full Adders' carry-out signals are propagated to the carry-in of the next FA on the right. It is now evident that input B enters the circuit in a way that would make it difficult to interface this circuit with others. The same applies to the result, whose bits need to be synchronized, just like for the Galois Multiplier. In Figure 6.27 the preskew (for B) and deskew (for Res) networks are added on top and bottom of the circuit body. It is immediately clear their great impact.

6.3.2.2 ME-NML implementation

The main element of the serial-parallel MAC is a Full Adder with a feedback loop for the result. The ME-NML FA used for our serial-parallel MAC is drawn in Figure 6.28. The feedback loop, highlighted in blue, is 3 clock periods long. Like the previous cases, the feedback is the critical path that determines the delay required between inputs. In this case an input bit has to be served every 3 clock cycles, hence the maximum throughput can be reached with a 3-operations interleaving. The two other patterns point out that the base block takes 2 clock cycles to be crossed horizontally, and 3 cycles vertically.

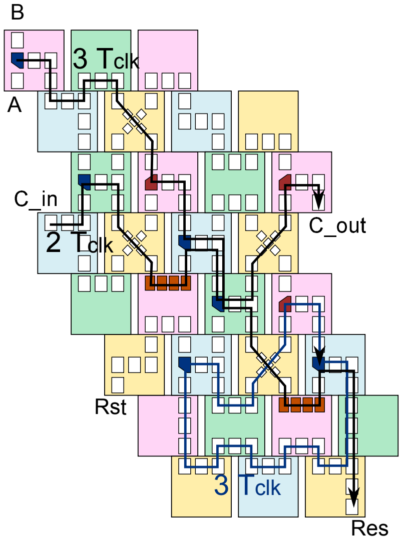


Figure 6.28. Full Adder block for the serial-parallel MAC. Three patterns underline horizontal crossing, vertical crossing and feedback loop. [27]

The full adders in the scheme of Figure 6.27 only have $1T_{clk}$ latency, therefore the timing is slightly different than the final ME-NML implementation. The two circuits are exactly the same apart from the internal delays. For example consider the input conditioning structure for B in Figure 6.27, each register of the column at the top-left corner is realized in ME-NML with a 3 cycles delay.

The ME-NML final circuit of the 4-bit MAC is shown in Figure 6.29. The circuit is divided into four main regions and within each region the dashed lines identify the basic blocks. To construct the generic MAC each region has been treated separately.

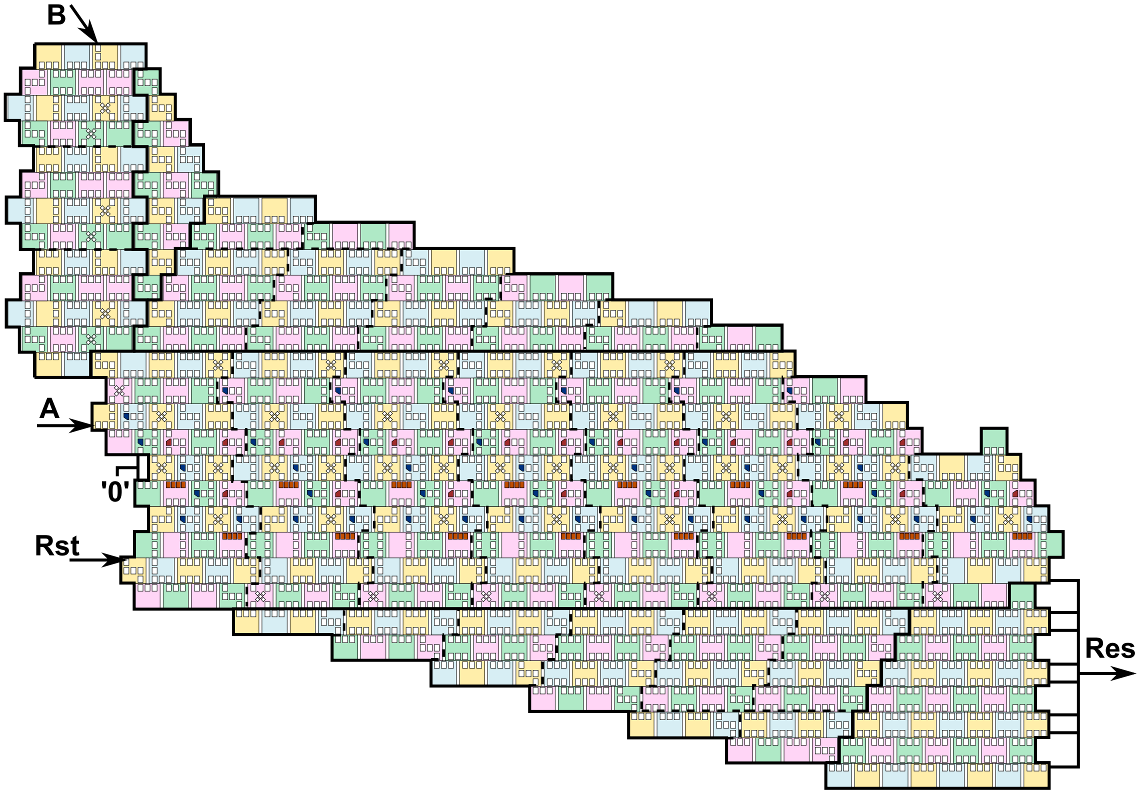


Figure 6.29. ME-NML implementation of the serial-parallel MAC.

First, we selected the set of recurrent blocks, then we investigated how to organize them so that combining them properly it is possible to create a MAC with any number of bits.

The whole circuit has been described with the RTL model we developed for ME-NML technology. A substantial effort was devoted to the generic description of the interconnection networks. The top entity `MAC_1D` instantiates the four entities reported above.

6.3.2.3 Timing analysis

Data A and data B are provided serially with a delay of 3 clock cycles between them. Then the time required to feed all the bits is $3N_{bit} \cdot T_{clk}$. After that, for other $3N_{bit} \cdot T_{clk}$ the inputs are set to $'0'$, until a new operation starts. The throughput would be equal to one operation every $3 \cdot 2N_{bit}$ clock cycles, but exploiting the interleaving technique it can be increased to $1/(2N_{bit} \cdot T_{clk})$. Table 6.12 reports these results and evaluates the overall circuit latency. Data A arrives directly at the

MAC's body, data B instead has to cross the preskew network first.

Table 6.12. Timing performance of the Serial-Parallel MAC

N bit	Interleaving	Throughput	Latency: 1st Result out
4	3 op.	$1/(8T_{clk})$	$36T_{clk}$
8	3 op.	$1/(16T_{clk})$	$76T_{clk}$
N	3 op.	$1/(2N \cdot T_{clk})$	$(6(N-1)+4N+2) \cdot T_{clk}$

6.3.3 Serial MAC Unit

The third and last implementation analyzed in this work is the Serial MAC, which has both serial inputs and output. The starting idea was to create a circuit exploiting only two 1-bit Full Adder, one for the multiplier and one for the adder.

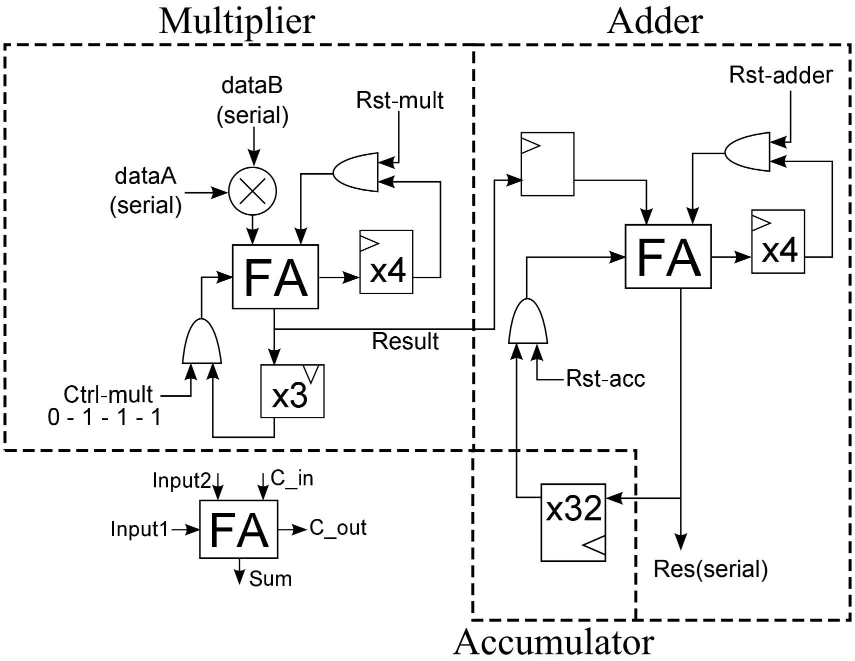


Figure 6.30. Scheme of the 4-bit serial MAC (preliminary implementation). [27]

6.3.3.1 Serial MAC scheme

The architecture that implements our concept is represented in Figure 6.30 in its 4-bit version. It consists of a serial multiplier, a serial adder and an accumulator, implemented as the adder feedback loop. Registers with the $x3, x4, x32$ labels represent multiple cascaded registers (respectively $3, 4, 32$) that have been combined together for a sharper visual understanding. Let us analyze each of the three components in detail.

Multiplier The multiplier accurately imitates the handmade multiplication algorithm (Figure 6.31 shows the 4-bit case). The serial inputs A and B are multiplied and then fed to the first Full Adder. Their products must produce all the 1-bit partial products of the form $A_i \cdot B_j$ (see Figure 6.31). To do so the inputs protocol for a 4-bit multiplication is shown in Figure 6.31.

				A_3	A_2	A_1	A_0	x
				B_3	B_2	B_1	B_0	=
				A_3B_0	A_2B_0	A_1B_0	A_0B_0	
			A_3B_1	A_2B_1	A_1B_1	A_0B_1	–	
		A_3B_2	A_2B_2	A_1B_2	A_0B_2	–	–	
	A_3B_3	A_2B_3	A_1B_3	A_0B_3	–	–	–	
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0	

Figure 6.31. Handmade 4-bit multiplication algorithm.

Data A bits are given in the order $\{A_0, A_1, A_2, A_3\}$ for 4 times (N_{bit} times) and then data A is set to $'0'$ until the end of the operation. To generate the partial product properly, each bit of data B must be multiplied with all the data A bits. Therefore the elapsed time to generate all the $A_i \cdot B_i$ products is $16 \cdot T_{clk}$ (in general $N_{bit}^2 \cdot T_{clk}$). In the 4-bit case B is fed in the following order: $\{B_0, B_0, B_0, B_0, B_1, B_1, B_1, B_1, B_2, B_2, B_2, B_2, B_3, B_3, B_3, B_3\}$. After that data B is set to $'0'$ until the end of the operation.

The Full Adder of the multiplier sums the partial products one bit at a time. It has two feedbacks, one for the result S and one for the carry-out, so that the whole multiplication can be carried out by a single FA module. For a correct alignment of the partial products' sum the carry feedback has to be N_{bit} registers long, while only $N_{bit} - 1$ are required for the result loop. The multiplier produces one bit of the result every N_{bit} clock cycles, therefore the whole operation takes $2N_{bit}^2 \cdot T_{clk}$, as the result is in $2N$ bits. The result is then forwarded to the adder, but only 1 bit out of N is meaningful.

Notice that the multiplier feedbacks require a control signal. The `Rst-mult` simply resets the carry-in before starting a new operation. The `Ctrl-mult` has instead a more complex function. We said that the output of the FA contains a bit of the final result every $N_{bit} \cdot T_{clk}$, all the other data are intermediate results. For a correct circuit functioning (see the algorithm in 6.31), the bits of the final result must not be fed back to the FA. `Ctrl-mult` is supposed to mask those bits, setting the feedback to $'0'$.

Adder The adder sums up the multiplication result to the value in the accumulator starting from the LSB and puts the result back into the accumulator. It also has to keep track of the carry bits. `Rst-adder` resets the carry loop when the LSB of a new result arrives. The other reset signal `Rst-acc` allows to set the accumulator to 0.

Accumulator The accumulator works as a shift registers, its data is always moving. Its length is equal to the duration of a multiplication: $2N_{bit}^2 \cdot T_{clk}$. Because of the circuit functioning, at any instant only $2N$ cells $(1/N)$ of the accumulator registers will contain useful data. A lot of space is then wasted by registers (or cells in ME-NML) that for most of the time do not contain meaningful data. The solution we propose to reduce the great impact of the accumulator on the circuit area is to let multiple MAC units share the same accumulator.

6.3.3.2 Serial MAC with shared Accumulator

The accumulator of the first serial MAC proposed (Figure 6.30) is too long and costly. Even though the data to be stored is $2N_{bit}$ long, the accumulator has a length of $2N_{bit}^2$ registers. At every instant $2N_{bit} \cdot (N_{bit} - 1)$ register contain meaningless data. This means that ideally the same accumulator could be shared by N different MAC units. Moreover, the Adder block can be shared as well, as it processes useful data only once every N_{bit} clock cycles.

We designed hence a circuit comprising N multipliers, one adder and one accumulator. The final scheme is shown in Figure 6.32, where eight 8-bit serial MAC units are represented together. They all share the same Accumulator and Adder. The *Mult.* and *Adder* are simplified as boxes, but they refer to the 8-bit circuit in Figure 6.30.

Under the multiplier blocks there are four rows of shift registers. The top line is necessary to carry the results from all the multipliers to the Adder, but only the meaningful values are allowed to enter that shift register. The output of each multiplier has to pass first through a multiplexer which is controlled by `Ctrl_results`. This signal makes sure that the intermediate results of the multiplication will not enter the shift register, given that all the multipliers output the result bits at the

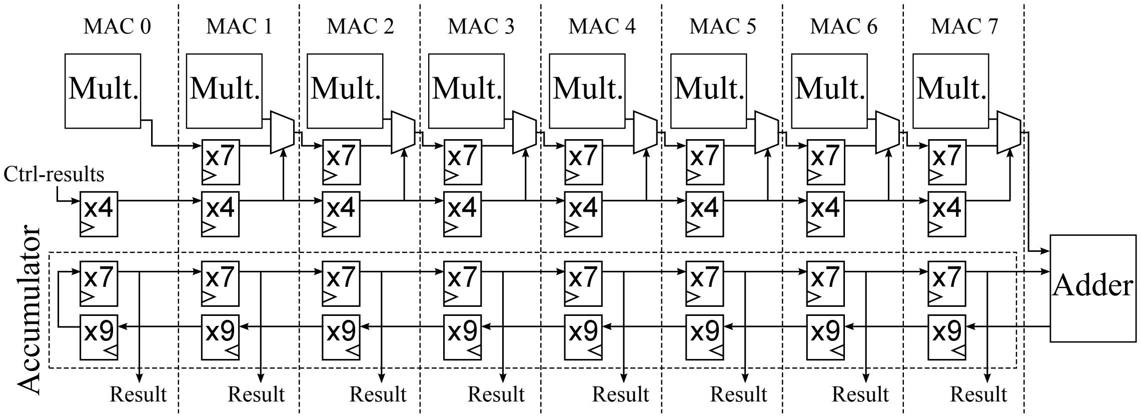


Figure 6.32. Scheme of the 8-bit serial MAC with shared Accumulator and Adder. [27]

same time. The length of 7 registers assures that the results of one MAC do not overwrite those of another one. The signal `Ctrl_results` is set to ′1′ once every N clock cycles, otherwise it is ′0′.

The Adder works as described before, but this time it will be exploited to its best, processing useful data all the time. The MAC result, stored in the accumulator, can be extracted serially from any point of it.

6.3.3.3 ME-NML implementation

The scheme in Figure 6.32 has been designed as suitable as possible to ME-NML technology. Until now all the ME-NML circuits were in some way modular, so that they could be described generically for any number of bits. On the contrary the serial MAC designed in this section is not scalable. All the feedbacks increase in length together with the number of bits, modifying radically the circuit layout.

6.3.3.4 Timing analisys

Changing the parallelism, the MAC requires to be redesigned from scratch, so we only designed and simulated the 8-bit serial MAC. The full 8-bit serial MAC is in Figure 6.33. It contains 8 different MAC units working in parallel and sharing both Accumulator and Adder. The results are provided from the top of each block, while in Figure 6.32 they were outputted at the bottom. Anyway both cases have the same timing. As usual this architecture has been described with our RTL model for ME-NML.

Since the Full Adders of the Multiplier process a continuous flow of data, for this implementation it is not necessary to use the interleaving technique. Table 6.13 contains the main information concerning timing. The throughput is the inverse of the

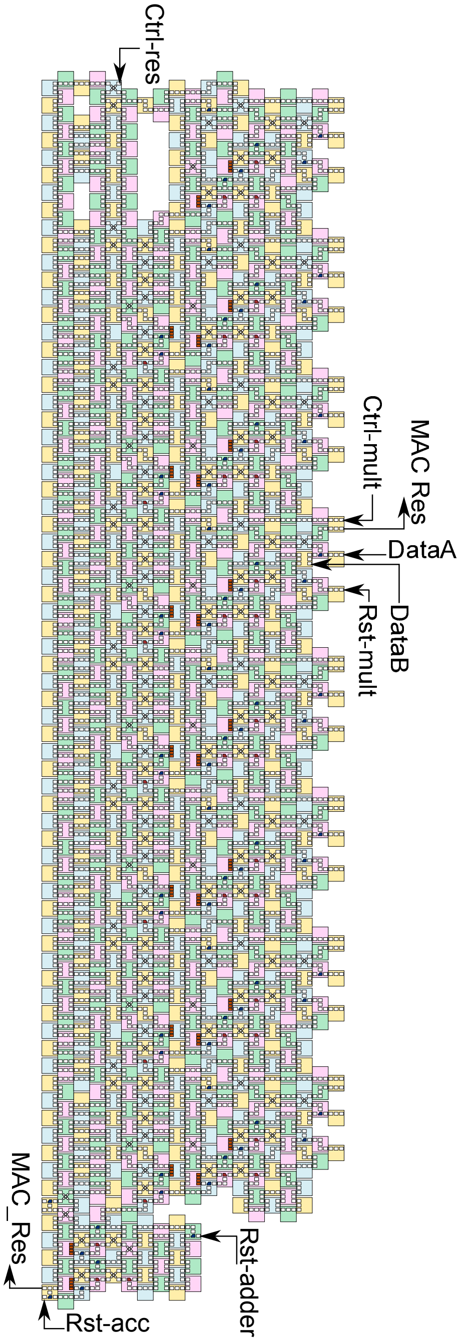


Figure 6.33. ME-NML implementation of the 8-bit serial MAC with shared Accumulator and Adder.

execution time of one operation: $1/(2N^2 \cdot T_{clk})$. Since the proposed circuit requires

N MAC to be linked together, the throughput for the entire shared-accumulator serial MAC (8 MACs) is $1/(2N)$.

Table 6.13. Timing Performance of the Serial MAC

N bit	**Interleaving**	**Throughput**	**Latency: LSB of Result out**
4	1 op.	$1/(32T_{clk})$	$45T_{clk}$
8	1 op.	$1/(128T_{clk})$	$85T_{clk}$
N	1 op.	$1/(2N^2 \cdot T_{clk})$	$(2N^2 + 9) \cdot T_{clk}$

The latency from the beginning of an operation to when the LSB of the result reaches the output is $2N^2 \cdot T_{clk}$.

6.3.4 Results

This section presents the performance outcomes for the MAC unit implementations proposed before. Here the three architectures are examined in terms of occupied area and power consumption, while the throughput and latency information have already been exhibited. At last the different MAC versions are placed side by side, offering a rigorous comparison. The results estimation follows the main guidelines adopted for the case study on the Galois Multiplier.

It will be proved that the parallel MAC outperforms the other two architectures. For a fair comparison of area and power, each implementation should have the same throughput, but this is not the case. Therefore we combined as many MAC modules as needed to reach a throughput equal to 1. For example since the serial MAC has throughput $1/(2N^2)$, the area and power of a single serial MAC have been multiplied by $(2N^2)$ as if $(2N^2)$ MAC units were working together to achieve a 1/1 throughput.

6.3.4.1 Parallel MAC Results

Table 6.14. Parallel MAC performance results.

Parallel MAC	Number of bits			
	4	8	16	32
Area (μm^2)	150	601	2410	9630
Power (μW)	9.7	38	150	600
Increase rate	-	3.92	3.96	3.98

For what concerns area occupation of the Parallel MAC, the layout of this circuit, as clear from Figure 6.24, has many empty internal regions. So the area evaluated by the model (not shown in Table 6.14) is smaller than it should, because it only considers the space occupied by cells. The value actually assigned to the parallel MAC is rounded up to the parallelogram circumscribed to the circuit. To obtain the parallelogram area we derived a generic equation for evaluating height and width (in terms of cells) for any number of bits. Through the VHDL model it is instead possible to evaluate the two power components, which have been added together to get the total consumption. Finally, referring to Table 6.14, it has to be noticed that the increase rate shows the growth of area when the number of bits doubles. So the increase rate in the $N_{bit} = 16$ column is the result for $N_{bit} = 16$ divided by the result for $N_{bit} = 8$. Area is more critical in this implementation, but similar values could be derived for Power.

6.3.4.2 Serial-Parallel MAC Results

Table 6.15. Serial-parallel MAC performance results.

Serial-parallel MAC	Number of bits				
	4	8	16	32	64
Area (μm^2)	41	128	432	1560	5900
Power (μW)	3.7	12	39	140	530
Increase rate	-	3.10	3.37	3.61	3.78

Since the Serial-parallel MAC layout is very compact, the area calculated by the VHDL model corresponds to the actual space occupied by the circuit. So increase rates of area and power are pretty much the same as they are both proportional to the number of cells. Actually the switching energy is only proportional to the number of nanomagnets, but for big circuits it is also in some way proportional to the number of cells.

All the results are displayed in Table 6.15. It is possible to see that the increase rate grows with the number of bits. This means that with higher number of bits this solution will be less convenient. The reason is that while the central block is proportional to the number of bits, the interconnection networks grow quadratically. Hence the impact of preskew/deskew circuits will be higher on the total circuit area occupation and power dissipation.

Table 6.16. Serial MAC performance results.

Serial MAC	Number of bits				
	4	8	16	32	64
Area (μm^2)	9.1	10.2	15.7	27	50
Power (μW)	0.82	0.92	1.4	2.4	4.5
Increase rate	-	1.12	1.54	1.72	1.84

6.3.4.3 Serial MAC Results

Even if only the 8-bit serial MAC has been designed and simulated, a projection of the number of cells for the other parallelisms has been obtained through some consideration on the circuit. What varies with the number of bits are the two feedback loops of the multiplier block (Figure 6.30), the Accumulator and the shift register that brings the products to the Adder (Figure 6.32). Also the loop of the adder increases in length. In each single MAC block, to obtain the 2N-bit circuit from the N-bit one, the multiplier's loops must get N clock periods longer. The same is true for the segment of the products' shift register and for each of the two segments of the accumulator. From this considerations it was possible to predict with good approximation the growth of the serial MAC with the number of bits.

Results are shown in Table 6.16, where area and power are the effective values for a single MAC, not those for the whole structure containing many MAC units.

The throughput of the serial MAC decreases quadratically with the number of bits. Therefore ideally, to keep up with the parallel MAC performance, the increase rate of a single MAC should be equal to 1. Unfortunately this is clearly not the case.

6.3.4.4 Results Comparison

The three architectures have been analyzed in terms of throughput, latency, circuit area and power consumption. Up to now the results of each MAC implementation have been presented singularly, now they are compared to evaluate the best solution.

Comparison conditions

Interleaving To reach their maximum throughput, both Parallel MAC and Serial-Parallel MAC, need to use the interleaving technique. The parallel circuit requires an interleave level equal to 5, otherwise its throughput would be $1/5T_{clk}$ and not $1/T_{clk}$. The serial-parallel version requires instead an interleave level of 3 only. The Serial MAC does not require any interleaving. The comparison is carried out in two different situations, at first considering the interleaving possibility, that would be similar to a real application scenario, then assuming that the interleaving cannot be exploited to analyze the effects of this technique.

Equal throughput To obtain a meaningful comparison, area and power performance should be referred to circuits with the same throughput. The output rate of the Parallel MAC has been used as reference for both cases: With and without interleaving. So the results concerning the parallel MAC are simply those of a single unit. On the other hand the results of the other two implementations have been multiplied by a coefficient, which is the number of units that should work in parallel to reach the same throughput as the Parallel MAC. They have to arrive at a $1/5T_{clk}$ rate without interleaving, and a $1/T_{clk}$ with interleaving.

- **Serial MAC.** Throughput always is $1/(2N^2 \cdot T_{clk})$. $2N^2$ MAC units are required to reach $1/T_{clk}$, $2N^2/5$ MAC units are required to reach $1/5T_{clk}$.
- **Serial-parallel MAC.** Exploiting interleaving its throughput is $1/(2N \cdot T_{clk})$, so $2N$ units are required to reach $1/1T_{clk}$. Without using the interleaving technique output rate is $1/(3 \cdot 2N \cdot T_{clk})$. So $3/5 \cdot 2N$ units are necessary to obtain $1/5T_{clk}$ throughput.

Results exploiting interleaving Results of the comparison for area and power, adopting interleaving, are depicted in Figure 6.34 and Figure 6.35. The 2D implementation is undoubtedly the most efficient, while the 1D (serial-parallel) has the worst outcomes.

The parallel MAC, with respect to the other implementations, is the best both for area and power, but in different ways. The power performance leads on the other architectures definitely more than the area occupation. This fact is to be attributed to the empty regions within the parallel MAC layout (serial and serial-parallel MAC do not have any), which largely increase the area but do not affect the power consumption. Furthermore, the area has been rounded up to the circumscribed parallelogram, including then also some empty space outside of the multiplier.

Results without exploiting interleaving In a situation where the interleaving technique could not be used, the hierarchies among the three MAC versions undergo slight changes. Referring to Figure 6.36 and Figure 6.37, it is clear that the performance of the parallel and serial-parallel MAC units worsen respectively of 5 and 3 times, according to their previous interleaving usage. The serial MAC gains a lot in this situation because it cannot exploit interleaving anyway.

In fact the serial MAC becomes the leading architecture up to a 16 bits parallelism. But since, as explained before, none of the implementations can keep up

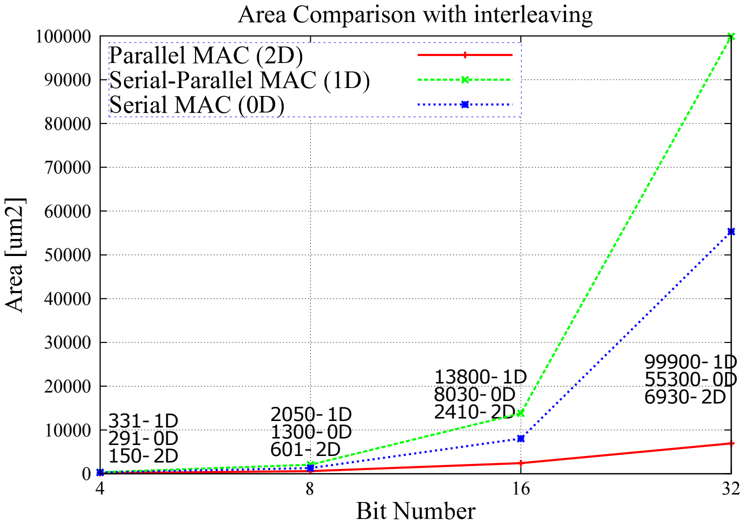


Figure 6.34. Area comparison of the three MAC implementations exploiting interleaving and with the throughput being equal.

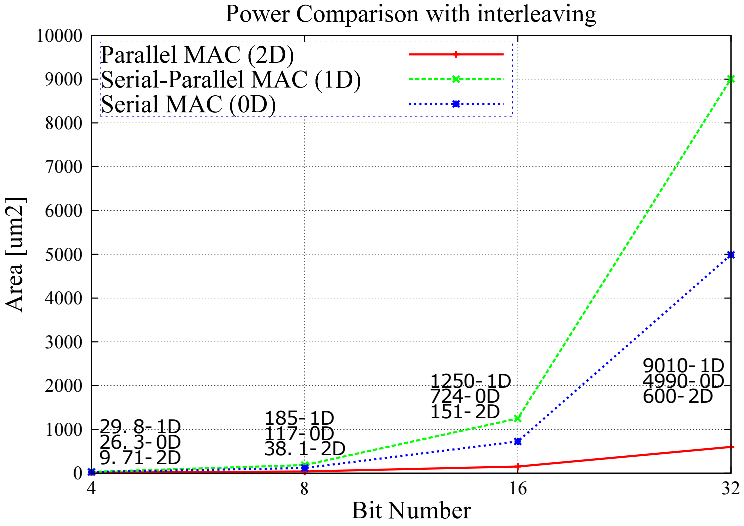


Figure 6.35. Power comparison of the three MAC implementations exploiting interleaving and with the throughput being equal.

with the parallel one when the number of bits increases, finally the parallel MAC takes back its lead for 32 or higher number of bits.

In conclusion the Parallel MAC is the best solution. The Parallel MAC has been designed with a structure similar to a systolic array. In this way the interconnections are kept to the minimum. With this analysis we have also investigated the effect

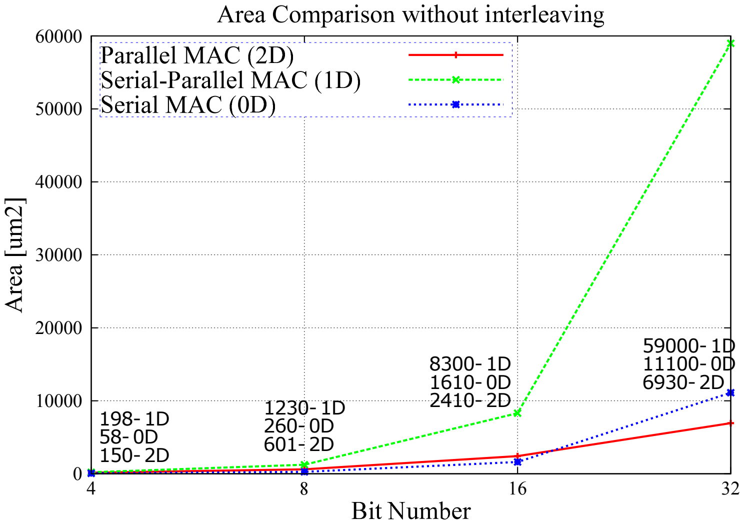


Figure 6.36. Area comparison of the three MAC implementations without interleaving and with the throughput being equal.

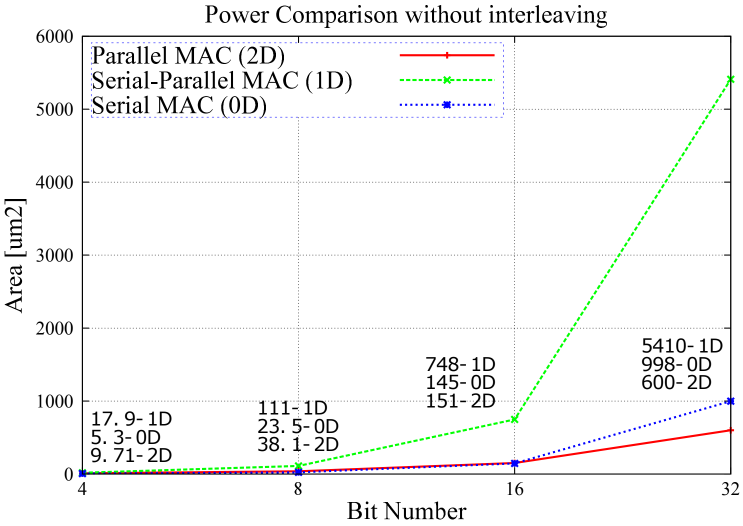


Figure 6.37. Power comparison of the three MAC implementations without interleaving and with the throughput being equal.

of preskew/deskew networks and set a first design of the MAC that can be reused. Indeed, since there is not yet an automatic tool for ME-NML technology, the design is done by hand and it is important to define the principle blocks that can be reused when other designs are approached (as in the Reconfigurable Systolic Array seen in Chapter 4). With this design we have enriched our library with a Ripple

Carry Adder, an Array Multiplier and finally a parallel MAC. These designs can be extended to any number of bits, since they are modular, thus making them more attractive.

6.4 Final Remarks

MagnetoElastic NML (ME-NML) is an extremely interesting enhancement for NML technology. Mainly, its clock generation mechanism is able to dramatically reduce power consumption. At the same time the extremely small definition of clock zones (each cell is a clock zone) allows to compact the design and obtain more flexible circuit paths. Area occupation is smaller than classic NML and at the same time the circuit density is much higher. Moreover, it is quite simple to achieve modular designs that can be extended to higher number of bits and reused in other circuits.

The design of complex ME-NML circuits has never been approached before. In this thesis the foundations for this circuit design are set. Several achievements can be mentioned:

- Standard Cell Library: a set of standard cells have been defined; they can be used to design any logic circuit in ME-NML. They represent the first gate library that can be used by an automatic tool (when it will be available) to translate general VHDL circuit descriptions in logic circuits with this technology.

- Enhanced VHDL Model: the Standard Cell Library has been completed creating the VHDL model of each cell. Moreover, this model can hierarchically compute area and power of the entire circuit. In this way it is also possible to have all the metrics of the circuit during simulation.

- Galois Field Multiplier design: GFM is widely used in cryptography and other relevant fields, so while we have used this as an example for the design of complex circuits, it will be possible in the future to use this generated circuit in a framework of ME-NML design of even higher complexity. The obtained circuit is advantageous over both CMOS and classic NML.

- Parallel approach for ME-NML: the best architectural choices made for CMOS are not always the best when translated in another technology. We have deepen analyzed the Parallel and Serial approach using as case study a Multiply and ACcumulate (MAC) structure. We have identified that the parallel solution is the best one, thus confirming that a CMOS parallel circuit can probably be translated in NML without the necessity to redistribute inputs and adapt the entire circuit (i.e. the exchange protocol between blocks can be maintained as it is).

- Multiplier, Adder and MAC design for ME-NML: since there is not yet an automatic tool for ME-NML, the design of principal blocks that can be then reused is fundamental. We have successfully designed the three blocks in such a way that the number of bits can be simply increased or reduced without further design effort.

With all these improvements, it is clear that ME-NML will make a step forward as potential technology to substitute some aspects of CMOS circuits, principally in such applications where power consumption is the main driver. With the growing usage of mobile technologies that suffer from low battery availability, then the research in this field may become fundamental for next-years advancements and to drive companies to the right choices. While we believe that ME-NML cannot be itself the only substitute of CMOS, with the results here presented it can be definitely one important solution for key logical blocks of low power consumption applications.

Chapter 7

Mixed ME-NML/CMOS Circuits

The work presented in Chapter 6 has evidenced the potentiality of ME-NML technology, in particular its low power consumption with respect to classic NML. The design of other ME-NML logic circuits has however evidenced also some limitations of this technology.

- When it is necessary to use several signals and mainly when they need to be crossed each other, the wasted area is extremely high. This is due to the fact that ME-NML is still a planar technology.
- This technology itself cannot be used to make complete systems, because its operating frequency is too low to respect timing constraints of some algorithms or applications. So, it is necessary to use it jointly with other technologies that can execute faster tasks. At the time of writing the obvious other technology is CMOS.

The idea of this research path is to exploit the jointly contribution of ME-NML and CMOS to have circuits that can benefit from both technologies' strengths. To do so, it is then necessary to design CMOS to NML and NML to CMOS interfaces, to let signals propagate from one technology to the other.

The concept of a mixed ME-NML/CMOS circuit is developed in Section 7.1 of this chapter. In particular the multiplexer with mixed circuitry is described and analyzed in detail. Section 7.2 is instead devoted to the electrical interface that can be used to exchange information between CMOS and ME-NML circuits. Conclusions on this topic and most important achievements are then summarized in Section 7.3.

7.1 Concept

This Section explains the Concept of the Mixed ME-NML/CMOS circuits. First, in paragraph 7.1.1, the advantages that can be gained with a mixed technology approach are explained. So, the reasons that lead us to this research path are evidenced. In paragraph 7.1.2 it is explained how it should look like a mixed circuit; in particular it is presented the mixed technology multiplexer that is the first step in the design of completely mixed circuits.

7.1.1 Advantages of Mixed Circuit

Table 7.1. Advantages and Drawbacks of ME-NML and CMOS technologies.

Technology	Advantages	Drawbacks
ME-NML	Low Power Consumption Small Area Occupation	Inefficient Multiplexer Long Latency of Wires
CMOS	Small Multiplexers Little Latency of Wires	Higher Power Consumption Greater Area Occupation

In Table 7.1.1 some advantages and drawbacks of ME-NML and CMOS are listed. It is evident that those that are the disadvantages of ME-NML can be seen as advantages of CMOS. In particular, two aspects are considered and highlighted:

1. Multiplexers in ME-NML are inefficient, because many wires have to be crossed in a single-layer technology. This is evident from Figure 7.1, where a simple 2 input 1 output 4-bit Multiplexer occupies 49 magnets horizontally and 35 magnets vertically. This area grows quadratically with the number of bits. Notice that the scheme of Figure 7.1 has a different graphical layout with respect to the ME-NML circuits shown in Chapter 6. Figure 7.1 has been obtained with MagCAD, a tool that we have developed at Politecnico di Torino that is able from a ME-NML (or even classic NML) to derive the corresponding VHDL circuit automatically. This tool is not yet stable and releasable, but it has already permitted us to make some preliminary analysis. Most of the circuits presented in this thesis have been imported and where possible optimized with MagCAD.

2. As already evidenced in the technological background (Section 2.3), long wires in NML have long latency. Even if we try to minimize long interconnections adopting a Systolic Array Processor organization, there may still be the necessity to route some external signal; for example some algorithms mapped in the Reconfigurable Systolic Array presented in Chapter 4 require external quick feedback loops. While this is not achievable in ME-NML, a CMOS wire between two distant ME-NML cells would solve this problem. There may be

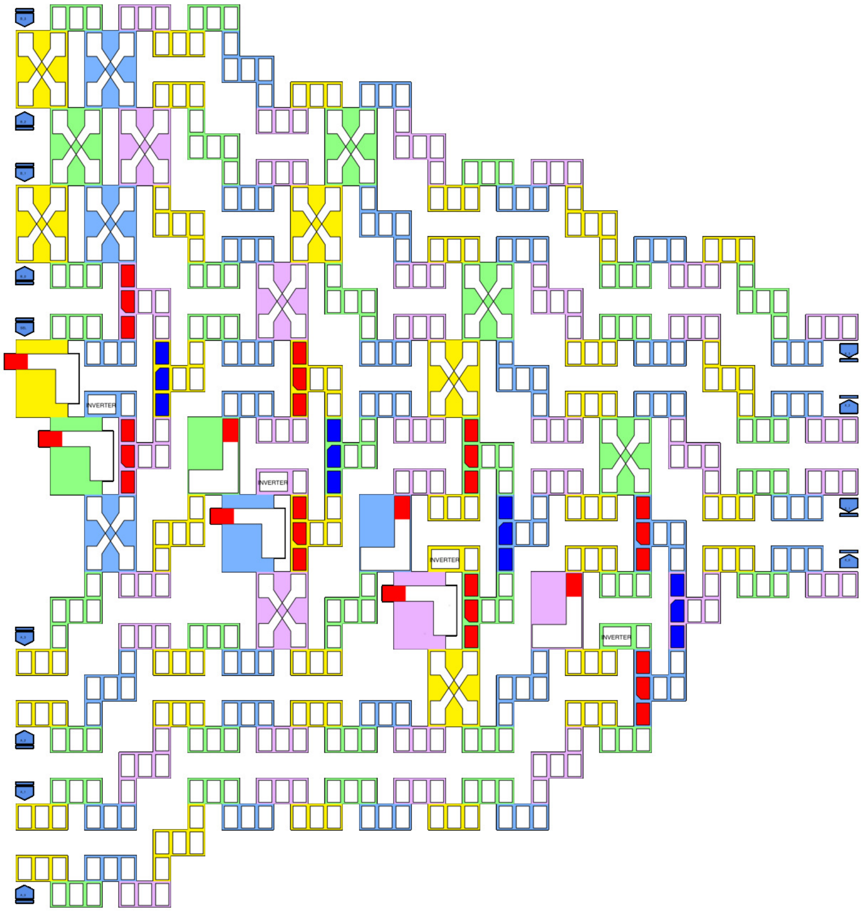


Figure 7.1. A 4-bit 2to1 multiplexer designed in ME-NML technology using MagCAD tool.

other technologies that could be combined with ME-NML to achieve the same result, as for example Domain Walls. This path has not been yet been covered, but it is a possible research path in the prosecution of this work.

7.1.2 Circuit Layout

In this first paragraph we will analyze how it is possible to create a Multiplexer with Mixed ME-NML/CMOS technology. This is the first design stage that we have approached towards multiple-technology circuits. The details of the electrical interface between CMOS and ME-NML are instead given in Section 7.2.

The concept circuit for mixed-technology Multiplexer is shown in Figure 7.2(a).

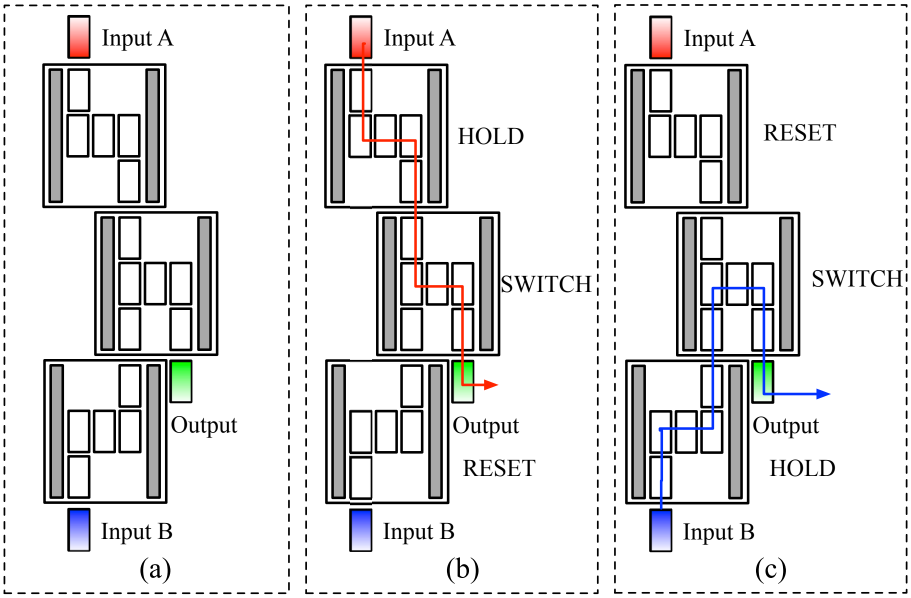


Figure 7.2. The concept of Mixed technology Multiplexer. In (a) it is presented the concept circuit, made by 3 ME-NML cells. Through clock signal selection, it is possible to obtain the Multiplexer function: in (b) it is presented the case of Input A selection, while in (c) it is presented the case of Input B selection.

The circuit is composed by three cells. The peculiar cell is the central one, that has a particular configuration non foreseen in the Standard Cell Library shown in Figure 6.2. This cell has indeed no meaning if applied to a normal circuit, because two inputs would pass through the same magnet that has no preferential polarization (so, it does not implement a logic cell) and the result would be always uncertain. However, the concept for this mixed-technology multiplexer is that only one cell between the top and the bottom one will provide its information to the central cell. The other cell (the one that does not provide the input) will be in conterphase and so it does not influence the central cell.

The signal selection to implement the multiplexer is shown in Figure 7.2(b-c). This is based on an appropriate clock phase selection for the top and bottom cell. In the first case (Figure 7.2(b)), the top cell is set in normal clock phase while the bottom cell will be in counterphase. In this way, top cell will be in HOLD phase and **Input A** will propagate to the center cell. The bottom cell in this moment will be in the RESET state and therefore it will not influence the central cell. The reverse situation will occur in Figure 7.2(c), where in this case **Input B** will be the signal that will propagate through the central cell to the output.

The signal distribution between the two technologies is shown in Figure 7.3. The selection **SEL** signal is placed in the CMOS Plane. This is used to select the correct

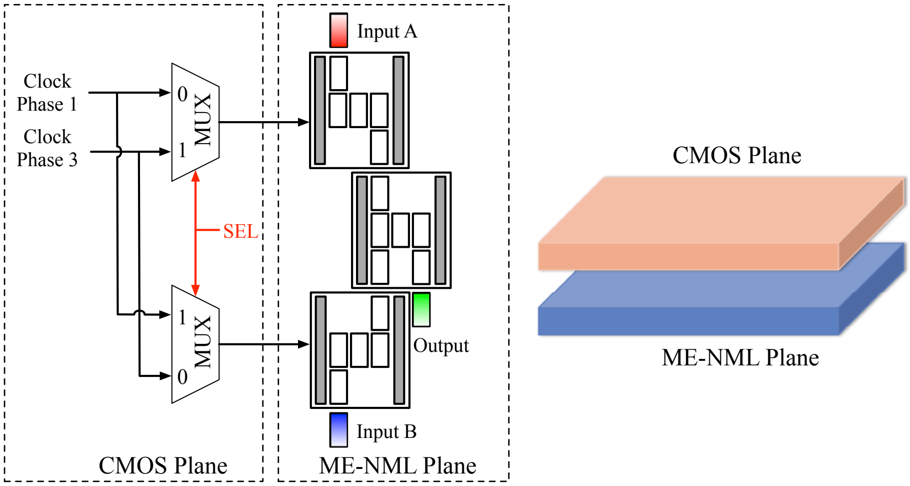


Figure 7.3. The two technology planes in the Mixed Multiplexer: the SEL signal is in the CMOS plane, and it is used to select the right clock phase of the top and bottom cell in ME-NML.

input of the multiplexer. When one of the two cells is assigned to Clock Phase 1, the other cell is assigned to Clock Phase 3, hence it is in counterphase. Notice also that the top and bottom block of the ME-NML multiplexer can be the last cells of two computational blocks. Hence the only additional area required to implement a multiplexer is the central cell. The additional area in CMOS plane for multiplexer is instead negligible with respect to the area necessary for ME-NML circuit.

7.1.2.1 RSA with Mixed Technology Multiplexer

The outstanding assumptions of this Mixed Technology Multiplexer, that can dramatically reduce area occupation (till 1 single ME-NML cell per bit), drove us to the design of the Reconfigurable Systolic Array (RSA), using this technology. The RSA is particularly interesting in this case because it makes broad usage of multiplexers for configuration. Moreover, in this case it is also interesting to notice that the multiplexers are only used for configuration. So all the SEL signals can be placed on the CMOS plane, and it is not necessary to implement an interface from ME-NML to CMOS.

In Figure 7.4 it is shown the effect in terms of number of cells (hence, area occupation), when the Mixed approach is applied to the Reconfigurable Systolic Array (RSA), in the version without multiplier. The RSA was not implemented for any number of bits, but analyzing the projections of circuit area made on single blocks, it is possible to extrapolate the trend shown in Figure 7.4.

In the case without multiplier, all circuit elements grow almost linearly with the

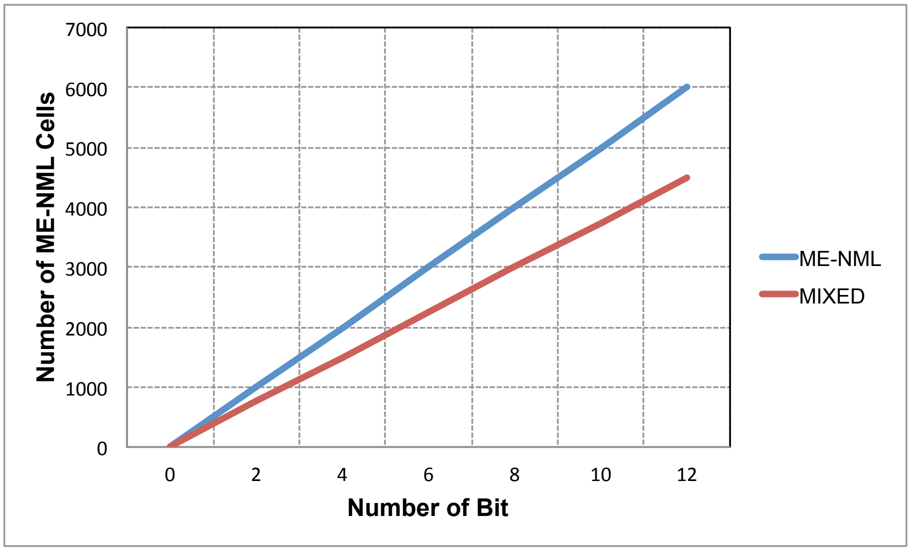


Figure 7.4. The effect of Mixed Multiplexer when applied to the Reconfigurable Systolic Array (RSA) - version without multiplier. Area saving is evident with the increasing number of bits.

number of bits. With this assumption it is clear that the impact of multiplexer is almost constant and in this case equal to 25%. If we consider instead the case with multiplier, RSA increases in area with a quadratic trend with respect to the number of bits. In this case the impact of the multiplexer will be reduced when a higher number of bits is considered.

Even if the effect on total circuit area might be negligible when we consider complex circuits with multipliers, there are other advantages in using this mixed-technology multiplexers: it is possible to completely separate the configuration section from the computation one. In this way it is also possible to change run-time the selected input or operation while the circuit on below ME-NML plane is already computing. This means that the configuration and the execution phase can be partially overlapped, overall resulting in a computational time saving.

7.2 Technological Interfaces

The basics of Mixed ME-NML/CMOS circuits have been described in previous Section. In this Section we will describe in detail the interface between the two technologies. We will first consider the interface from CMOS to ME-NML, that is derived from the multiplexer example, in paragraph 7.2.1; then we will analyze the opposite interface, from ME-NML to CMOS, in paragraph 7.2.2.

7.2.1 From CMOS to ME-NML

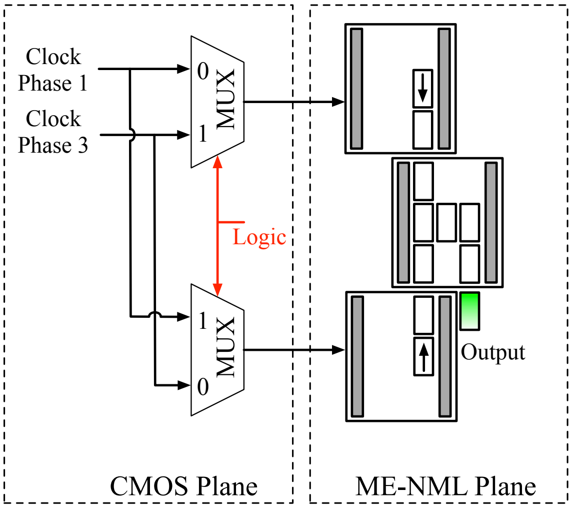


Figure 7.5. The interface from CMOS to ME-NML: the "Logic" signal in CMOS is used to select the right clocks phase that will drive two ME-NML cells. In this way in ME-NML the correct signal is selected.

The basic concept of the interface towards ME-NML is the multiplexer described in previous section and shown in Figure 7.3. In the case of general interface, the top and bottom cells must be initialized with '1' and '0' values. In this way the logic signal in CMOS will be used to drive two CMOS Multiplexers that will select be used in ME-NML to select the correct input. This concept is shown in Figure 7.5. Finally this interface will occupy only 2 multiplexers in CMOS and three cells in ME-NML, that is a reasonable amount of area for a technological interface.

7.2.2 From ME-NML to CMOS

The transduction of the CMOS signal in ME-NML has been quite straightforward, starting from the Multiplexer design. As a matter of fact, the circuit does not require particular sensing element or processing block. The opposite interface is instead more complex and it has been more difficult to conceive. It is described in the remaining of this paragraph.

7.2.2.1 Signal Transduction

For the transduction of the nanomagnet magnetization state the idea is to adopt a simple structure which consists of a piece of metal with the same planar dimensions of a nanomagnet ($50 \times 100 \times 10\,nm$). This metal element is placed above the

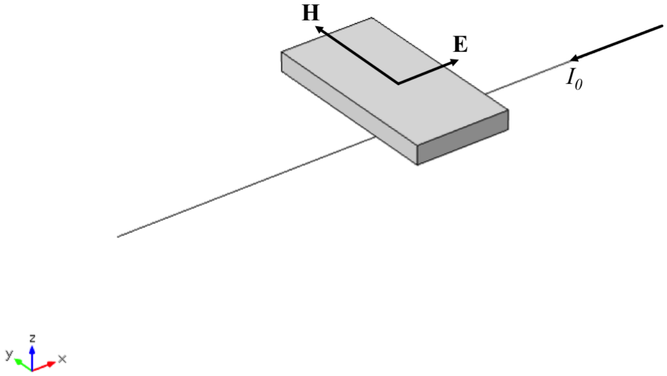


Figure 7.6. Simulated structure

nanomagnets level over a dielectric layer. This element represents the sensor of the technological interface. It exploits the Faraday effect: a variation of the magnetic field due to changes in the nanomagnet logic state induces an electromotive force that can be then processed by a CMOS circuit.

The simulation of this structure has been carried out by exploiting COMSOL [95] and considering the source of magnetic field as an ideal conductor through which a current flows. The current I_0 is described by equation 7.1.

$$I_0 = \sqrt{2}\ 90\Big(2\ rect(t) - 1\Big)10^{-3}[A] \tag{7.1}$$

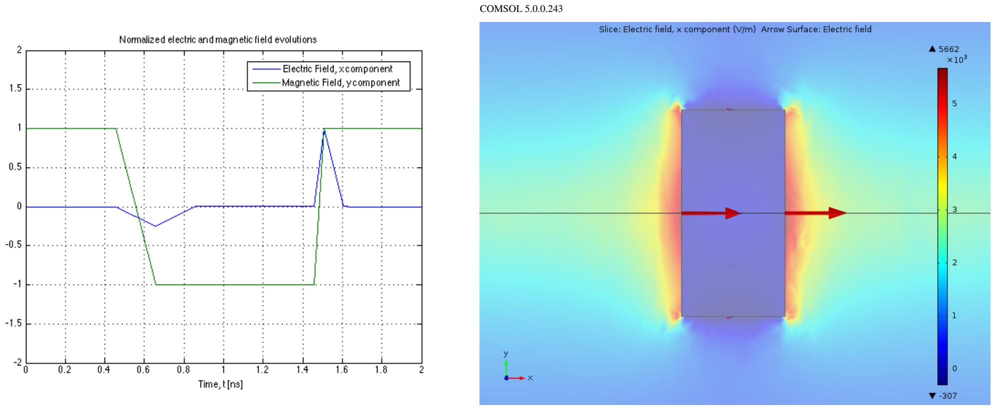
The *rect* function represents a square waveform that has Duty Cycle DC = 50%, period of 2 ns and unitary unipolar amplitude. The peak value of I_0 has been chosen to guarantee a saturation magnetization value of Nickel (Ni) $Ms \approx 0.5 \cdot 10^6$ $[A/m]$ at 5 nm from the ideal conductor. So, the surface of a Ni nanomagnet is assumed to be at 5 nm from the conductor. Simulations have been run with the metal feature, made by Copper (Cu), 25 nm far from the source, thus 20 nm from the nanomagnet. The dielectric layer has not been considered because it is supposed to be made of SiO_2, with $\mu_r \approx 1$, hence transparent to the magnetic field. The described structure is depicted in Figure 7.6.

Transitions between logic states, hence magnetic field variations, do not occur abruptly but a time rise $t_r = t_f$ is set in COMSOL within the function *rect*. By keeping the distance of the metal feature constant, several simulations have been performed by sweeping the rise time during which the electric field induction occurs.

The system has been precisely oriented parallel to the $\hat{x}$, $\hat{y}$ and $\hat{z}$ so to study the simulation by only evaluating $\mathbf{H}\cdot\hat{\mathbf{y}}$ and $\mathbf{E}\cdot\hat{\mathbf{x}}$ and not the absolute values. By the way, the other components resulted to be negligible because of the adopted symmetry.

In Figure 7.7 results of a simulation carried out with $t_r = 10\,ps$ are shown. The time evolution of the electric and magnetic fields are plot, with amplitudes normalized to the respective absolute maximum values just to appreciate the behavior of the Induction Electric Field (IEF) with respect to the magnetic field. In Figure 7.7(b) it has been reported the upper view of the sensor at time $t = 1.5\,ps$ and the color scale refers to average computed value.

What can be observed in Figure 7.7(a) is that t_f does not coincide with t_r, this is because of different time steps adopted by the simulator during the simulation. Indeed, it has been observed that during the simulation the initial time step was not fixed at the same set value (dt = 0.001 ns) but instead the simulator started with an higher value, converging to the set one. This can be noticed in graph plotted during the simulation.



(a) (b)

Figure 7.7. Results of the COMSOL simulation with t_r = 10 ps. (a) Normalized H_y and E_x components with respect to their maximum absolute value. (b) Up view at t = 1.5 ps, in the picture are shown average value and direction of E_x $[V/m]$.

With the same structure it has been also performed a simulation in which the source of magnetic field has been supposed to be spin waves. Because of the harmonic nature of such waves, simulations have been carried out in the frequency domain, keeping the amplitude value of the current I_0 constant at $\sqrt{2}\,90\cdot10^{-3}\,A$. Simulations have been run at different frequencies, as shown in Table 7.2. In Figure 7.8 it is shown the up view of the sensor at $f = 50$ GHz and the average value of $|\mathbf{E}|$.

Table 7.2. Values of IEF in function of the operating frequency achieved

$f[GHz]$	Average IEF $[V/m]$
10	$1.06 \cdot 10^4$
20	$2.11 \cdot 10^4$
30	$3.17 \cdot 10^4$
40	$4.22 \cdot 10^4$
50	$5.28 \cdot 10^4$
60	$6.33 \cdot 10^4$
70	$7.39 \cdot 10^4$
80	$8.45 \cdot 10^4$
90	$9.50 \cdot 10^4$
100	$1.05 \cdot 10^5$

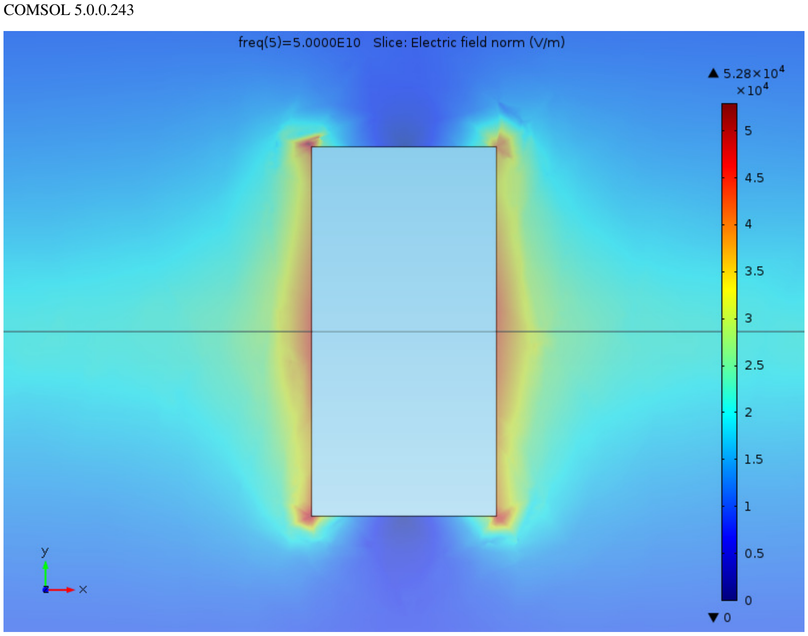


Figure 7.8. Up view at $f = 50$ GHz, in the picture is shown the average value of $|\mathbf{E}|$ $[V/m]$

About numerical solutions related to this sensor, they are encouraging. Values of the IEF obtained by the time dependent simulations present an exponential-like decay with respect to the rise time of the magnetic variation as expected. This is because of the differential relation between the two fields. From frequency based simulations, instead, a linearly increasing trend has been shown for the IEF, indeed as the frequency increases so does the coupling between the wavelength of the spin wave and the sensor dimension.

To summarize what it has been presented in this paragraph, it is possible to state that the COMSOL simulations of the proposed metal sensor for magnetic field in ME-NML guarantees a change in the electric field that can be used to drive a CMOS logic circuit. The amplification on this signal in order to be used correctly with other CMOS logic gates is presented in next paragraph.

7.2.2.2 CMOS bridge

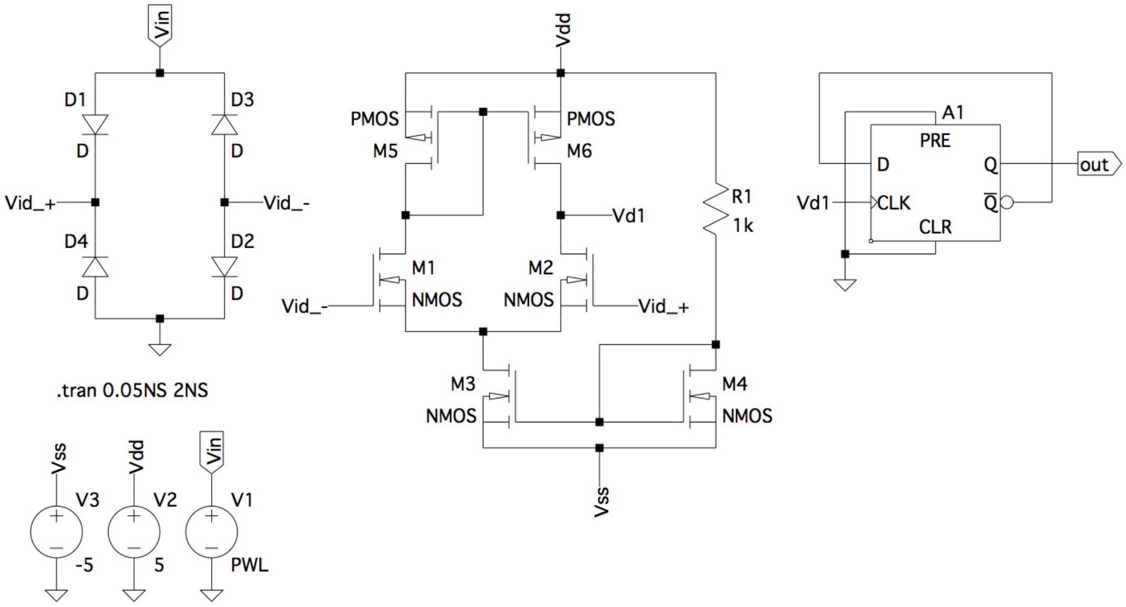


Figure 7.9. CMOS - NML interface, electric scheme designed with LTspice.

The configuration proposed is shown in Figure 7.9. It is able to hold on signals for 500 MHz signals.

The circuit is composed by three blocks, from the left to the right: a Graetz bridge, a differential amplifier and a D-FF used as a T one in toggle mode. The electric signal from the sensor passes through the Graetz configuration and it is supplied as input to the differential amplifier. The amplified signal, V_{d1}, is then sourced as clock to the toggle circuit. Note that the flip flop has a clock input which is sensible to the falling edge of the signal. The simulation of this circuit has been run with ideal components (low level models) and the results are shown in Figure 7.10 for a $t_r = 10$ ps input signal. The output signal has the same frequency of the input one, with a delay due to the decay of the electric field when the magnetic field varies no more. The signal obtained from the flip flop, can be then used to drive a corresponding CMOS circuit. An eventual discrimination for the initial conditions of the system can be accounted for by realizing an ad-hoc logic function which controls PRE and CLR inputs shown in Figure 7.9.

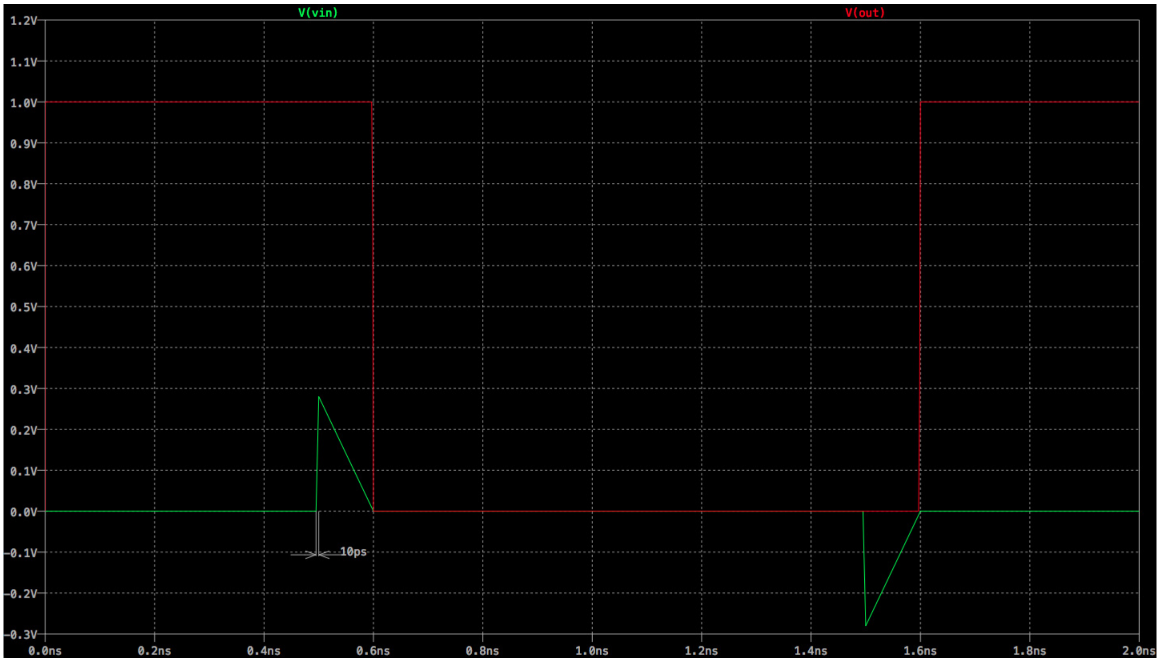


Figure 7.10. Simulation result on a virtual oscilloscope.

7.3 Final Remarks

The design of complex ME-NML logic circuits has evidenced some limitations, in particular the high area occupation of multiplexers. Another important technological limitation is the long latency of wires. To address these items it is possible to design mixed-technology circuits that can benefit from the strength points of both ME-NML and CMOS. Several achievements can be mentioned in this research path:

- Hybrid Multiplexer: since the multiplexer is one of the components that occupy a lot of area in ME-NML, it has been designed an Hybrid Multiplexer where the selection of the output is actually done in the CMOS clocking plane, while the below ME-NML circuit is used only to transmit the correct signal to the other blocks in ME-NML. This approach allows to save a relevant number of cells, as it has been demonstrated with the Reconfigurable Systolic Array example.

- CMOS to ME-NML interface: based on the previous multiplexer, it has been designed a general CMOS to ME-NML interface that can be used to have fully hybrid logic circuits.
- ME-NML to CMOS interface: the transduction of magnetic fields variation generated by a flipping in the state of a nanomagnet can be read with a metal sensor and then translated in a signal that can be used to drive a CMOS circuit. In this way it is possible to achieve fully hybrid circuits.

Finally with these achievements it is possible to state that ME-NML can have a relevant role as future technology. It has outstanding features that have been presented in previous Chapter; moreover, its main limitations can be addressed technologically by using CMOS for special functions such as multiplexers or long wires with no delay.

Chapter 8

Conclusions

In this Thesis I have presented the main topics of my research activity during the PhD.

The entire work has been split into two parts. The first part dealt with Architectural solutions for NanoMagnet Logic, that however can be also implemented with other technologies. The second part was dedicated to MagnetoElastic NanoMagnet Logic technology, introducing some rules and techniques for the design of logic circuits and showing how to implement an interface with other technologies.

Five main topics have been covered: Systolic Arrays optimization, Reconfigurable Systolic Arrays and Logic-In-Memory circuits in the Architectural framework. Design rules and technological interfaces in the ME-NML framework.

All these topics have common points and are in several ways linked together, starting from the technology itself. Indeed the achievements and the research topics were not pre-defined at the beginning of the PhD activity, but they have emerged while the knowledge on the technology augmented. As an example of this process, the Mixed technology multiplexer would have never been thought before the introduction of the Reconfigurable Systolic Array, that made broad use of multiplexer, becoming less efficient in NML.

As a matter of fact, there are no well defined future achievements to pursue. The introduction of all these technological and architectural enhancements has evidenced many strengths of NML but also some limitations. Therefore, the effort in finding the best technological alternative to CMOS will continue in future years. NML can be strengthened with the improvements presented, and it will probably be enhanced with new technological features, mainly to become a faster technology.

Nevertheless, if another new beyond-CMOS technology will be finally chosen, the architectural improvements presented in this thesis could again be used to design new circuits layouts. In a similar way some of the improvements here presented could be immediately applicable to CMOS technology in some peculiar fields.

The final advice is that the improvements presented in this thesis can be mixed together to generate new circuits with incredible features. As an example, it is possible to design a Logic-In-Memory processor with Reconfigurable PE, inheriting the architecture of the Reconfigurable Systolic Array. The multiplexers of this architecture can be done in mixed technology, and the core itself can be made latency-insensitive.

Appendix A

List of Publications

- Giovanni Causapruno, Marco Vacca, Mariagrazia Graziano and Maurizio Zamboni, **"Interleaving in Systolic-Arrays: A Throughput Breakthrough"** in IEEE Transactions on Computers, vol.64, no.7, pp.1940-1953, July 2015.
  DOI: 10.1109/TC.2014.2346208

- Giovanni Causapruno, Gianvito Urgese, Marco Vacca, Mariagrazia Graziano and Maurizio Zamboni, **"Protein Alignment Systolic Array Throughput Optimization"** in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.23, no.1, pp.68-77, January 2015.
  DOI: 10.1109/TVLSI.2014.2302015

- Giovanni Causapruno, Fabrizio Riente, Giovanna Turvani, Marco Vacca, Massimo Ruo Roch, Maurizio Zamboni and Mariagrazia Graziano, **"Reconfigurable Systolic Array: From Architecture to Physical Design for NML"** in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Early Access Article.
  DOI: 10.1109/TVLSI.2016.2547422

- Marco Vacca, Mariagrazia Graziano, Juanchi Wang, Fabrizio Cairo, Giovanni Causapruno, Gianvito Urgese, Andrea Biroli and Maurizio Zamboni, **"Nano-Magnet Logic: An Architectural Level Overview"** in Field-Coupled Nanocomputing, Volume 8280 of the series *Lecture Notes in Computer Science*, Editors Neal G. Anderson and Sanjukta Bhanja, pp. 223-256, June 2014.
  DOI: 10.1007/978-3-662-43722-3_10

- Davide Giri, Marco Vacca, Giovanni Causapruno, Wenjing Rao, Mariagrazia Graziano and Maurizio Zamboni, **"A standard cell approach for MagnetoElastic NML circuits"** in 2014 IEEE/ACM International Symposium on

Nanoscale Architectures (NANOARCH), pp.65-70, 8-10 July 2014.
DOI: 10.1109/NANOARCH.2014.6880491

- Diego Pala, Giovanni Causapruno, Marco Vacca, Fabrizio Riente, Giovanna Turvani, Mariagrazia Graziano and Maurizio Zamboni, **"Logic-in-Memory architecture made real"** in 2015 IEEE International Symposium on Circuits and Systems (ISCAS), pp.1542-1545, 24-27 May 2015.
DOI: 10.1109/ISCAS.2015.7168940

- Mario Cofano, Giulia Santoro, Marco Vacca, Diego Pala, Giovanni Causapruno, Fabrizio Cairo, Fabrizio Riente, Giovanna Turvani, Massimo Ruo Roch, Maurizio Zamboni and Mariagrazia Graziano, **"Logic-in-Memory: A Nano Magnet Logic Implementation"** in 2015 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 286-291, 8-10 July 2015.
DOI: 10.1109/ISVLSI.2015.121

- Giovanni Causapruno, Alessandro Audero, Sergio Tota and Massimo Ruo Roch, **"A Framework for Network-On-Chip comparison based on OpenSPARC T2 processor"** in Applications in Electronics Pervading Industry, Environment and Society, *Lecture Notes in Electrical Engineering*, pp. 99-105, May 2014.
DOI: 10.1007/978-3-319-20227-3_13

- Guoping Xiao, Waqar Ahmad, Syed Azhar Ali Zaidi, Massimo Ruo Roch and Giovanni Causapruno, **"High Speed VLSI architecture for finding the first W maximum/minimum values"** in Applications in Electronics Pervading Industry, Environment and Society, *Lecture Notes in Electrical Engineering*, pp. 35-41, May 2014.
DOI: 10.1007/978-3-319-20227-3_5

- Giovanni Causapruno, Umberto Garlando, Fabrizio Cairo, Mariagrazia Graziano and Maurizio Zamboni, **"A Reconfigurable Array Architecture for NML"**, in 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Accepted for Publication.

Bibliography

[1] "International technology roadmap for semiconductors (ITRS)", 2013, http://www.itrs.com.

[2] N.Z. Haron and S. Hamdioui, "Why is cmos scalig coming to an end?", *Design and Test Workshop, 2008. IDT 2008. 3rd International*, pp. 98–103, 20-22 Dec 2008.

[3] Yong-Bin Kim, "Review paper: Challenges for nanoscale mosfets and emerging nanoelectronics", *Trans. Electr. Electron. Mater. 10(1) 21*, 2009.

[4] S. Deleonibus, B. De Salvo, L. Clavelier, T. Ernst, O. Faynot, T. Poiroux, and M. Vinet, "Cmos devices architectures for the end of the roadmap and beyond", *Solid-State and Integrated Circuit Technology, 8th International Conference on*, pp. 51–54, 23-26 Oct 2006.

[5] C.S. Lent, P.D. Tougaw, W. Porod, and G.H. Bernstein, "Quantum cellular automata", *Nanotechnology*, vol. 4, pp. 49–57, 1993.

[6] P.D. Tougaw and C.S. Lent, "Dynamic behavior of quantum cellular automata", *Journal Of Applied Physics*, , no. 80, pp. 4722–4736, 1996.

[7] C.S. Lent and B. Isaksen, "Clocked Molecular Quantum-Dot Cellular Automata", *IEEE Transactions on Electron Devices*, vol. 50, no. 9, pp. 1890–1896, Sept. 2003.

[8] M. Liu C. Lent Y. Lu, "Molecular electronics - from structure to circuit dynamics", in *Sixth IEEE Conference on Nanotechnology*, Cincinnati-Ohio, USA, 2006, pp. 62–65, IEEE.

[9] A. Pulimeno, M. Graziano, D. Demarchi, and G. Piccinini, "Towards a molecular QCA wire: Simulation of write-in and read-out systems", *Solid-State Electronics, Elsevier*, vol. 1, pp. 7, 2012.

[10] A. Pulimeno, M. Graziano, V.Cauda, A. Sanginario, D. Demarchi, and G. Piccinini, "Bis-ferrocene molecular qca wire: ab-initio simulations of fabrication driven fault tolerance", *IEEE Trans. Nanotechnol.*, vol. 12, no. 3, 2013.

[11] RobertA. Wolkow, Lucian Livadaru, Jason Pitters, Marco Taucer, Paul Piva, Mark Salomons, Martin Cloutier, and BrunoV.C. Martins, "Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics", in *Field-Coupled Nanocomputing*, Neal G. Anderson and Sanjukta Bhanja, Eds., pp. 33–58.

Springer Berlin Heidelberg, 2014.

[12] M. B. Haider and al., "Controlled coupling and occupation of silicon atomic quantum dots at room temperature", *Phys. Rev. Lett.*, vol. 102, Jan. 2009.

[13] M. Niemier and al., "Nanomagnet logic: progress toward system-level integration", *J. Phys.: Condens. Matter*, vol. 23, pp. 34, Nov. 2011.

[14] R.P. Cowburn and M.E. Welland, "Room temperature magnetic quantum cellular automata", *Science*, vol. 287, pp. 1466–1468, 2000.

[15] Marco Vacca, Mariagrazia Graziano, Alessandro Chiolerio, Andrea Lamberti, Marco Laurenti, Davide Balma, Emanuele Enrico, Federica Celegato, Paola Tiberto, Luca Boarino, and Maurizio Zamboni, "Electric Clock for Nano-Magnet Logic Circuits", in *Field-Coupled Nanocomputing*, Neal G. Anderson and Sanjukta Bhanja, Eds., Lecture Notes in Computer Science, pp. 73–110. Springer Berlin Heidelberg, 2014.

[16] M. Vacca, M. Graziano, L. Di Crescenzo, A Chiolerio, A Lamberti, D. Balma, G. Canavese, F. Celegato, E. Enrico, P. Tiberto, L. Boarino, and M. Zamboni, "Magnetoelastic Clock System for Nanomagnet Logic", *IEEE Trans. Nanotechnol.*, vol. 13, no. 5, pp. 963–973, Sep. 2014.

[17] C. Augustine, X. Fong, B. Behin-Aein, and K. Roy, "Ultra-Low Power Nano-Magnet Based Computing: A System-Level Perspective", *IEEE Trans. Nanotechnol.*, vol. 10, no. 4, pp. 778–788, 2011.

[18] G. Csaba and W. Porod, "Behavior of Nanomagnet Logic in the Presence of Thermal Noise", in *International Workshop on Computational Electronics*, Pisa, Italy, 2010, pp. 1–4, IEEE.

[19] Marco Vacca, Mariagrazia Graziano, Juanchi Wang, Fabrizio Cairo, Giovanni Causapruno, Gianvito Urgese, Andrea Biroli, and Maurizio Zamboni, "Nano-Magnet Logic: An Architectural Level Overview", in *Field-Coupled Nanocomputing*, Neal G. Anderson and Sanjukta Bhanja, Eds., Lecture Notes in Computer Science, pp. 223–256. Springer Berlin Heidelberg, 2014.

[20] M.T. Alam, M.J. Siddiq, G.H. Bernstein, M.T. Niemier, W. Porod, and X.S. Hu, "On-chip Clocking for Nanomagnet Logic Devices", *IEEE Transaction on Nanotechnology*, 2009.

[21] Alexandra Imre, *Experimental study of nanomagnets for Quantum-dot cellular automata(MQCA)logic applications*, PhD thesis, University of Notre Dame, Notre Dame, Indiana, Dec. 2005.

[22] M.T. Niemier, E. Varga, G.H. Bernstein, W. Porod, M.T. Alam, A. Dingler, A. Orlov, and X.S. Hu, "Shape Engineering for Controlled Switching With Nanomagnet Logic", *IEEE Trans. Nanotechnol.*, vol. 11, no. 2, pp. 220–230, Mar. 2012.

[23] M. Vacca, M. Graziano, and M. Zamboni, "Majority Voter Full Characterization for NanoMagnet Logic Circuits", *IEEE Trans. Nanotechnol.*, vol. 11, no. 5, pp. 940–947, 2012.

[24] M. Graziano, M. Vacca, A. Chiolerio, and M. Zamboni, "A NCL-HDL Snake-Clock Based Magnetic QCA Architecture", *IEEE Transaction on Nanotechnology*, , no. 10, pp. DOI:10.1109/TNANO.2011.2118229.

[25] Marco Vacca, *Emerging Technologies: NanoMagnets Logic (NML)*, PhD thesis, Politecnico di Torino, Turin, Italy, april 2013.

[26] M. Graziano, A. Chiolerio, and M. Zamboni, "A Technol. Aware Magnetic QCA NCL-HDL Architecture", in *Int. Conf. Nanotechnol.*, Genova, Italy, 2009, pp. 763 – 766, IEEE.

[27] D. Giri, "MagnetoElastic NanoMagnet Logic Circuits", Master's thesis, Politecnico di Torino, December 2014.

[28] J. Das, S.M. Alam, and S. Bhanja, "Low power magnetic quantum cellular automata realization using magnetic multi-layer structures", *J. on Emerging and Selected Topics in Circuits and Systems*, vol. 1(3), pp. 267–276, Sep 2011.

[29] N. Rizos, M. Omar, P. Lugli, G. Csaba, M. Becherer, and D. Schmitt Landsiedel, "Clocking schemes for field coupled devices from magnetic multilayers", in *IEEE International Workshop on Computational Electronics*, Beijing, China, 2009, pp. 1–4.

[30] J. Atulasimha and S. Bandyopadhyay, "Hybrid spintronic/straintronics: A super energy efficient computing scheme based on interacting multiferroic nanomagnets", in *12th IEEE International Conference on Nanotechnolog*, 2012, pp. 1–4.

[31] D. Giri, M. Vacca, G. Causapruno, W. Rao, M. Graziano, and M. Zamboni, "A standard cell approach for MagnetoElastic NML circuits", in *EEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH)*, Jul. 2014, pp. 65–70.

[32] M. S. Fashami, J. Atulasimha, and S. Bandyopadhyay, "Magnetization dynamics, throughput and energy dissipation in a universal multiferroic nanomagnetic logic gate with fan-in and fan-out", *Nanotechnology*, vol. 23(10), February 2012.

[33] H.T. Kung, C.E. Leiserson, and Carnegie-Mellon University. Dept. of Comput. Science, *Systolic Arrays for VLSI*, CMU-CS. Carnegie-Mellon University, Department of Comput. Science, 1978.

[34] Hyesook Lim and Jr. Swartzlander, E.E., "Multidimensional systolic arrays for the implementation of discrete Fourier transforms", *IEEE Trans. Signal Process.*, vol. 47, no. 5, pp. 1359 –1370, may 1999.

[35] H. Herzberg and R. Haimi-Cohen, "A systolic array realization of an LMS adaptive filter and the effects of delayed adaptation", *IEEE Tran.s on Signal Processing*, vol. 40, no. 11, pp. 2799–2803, nov 1992.

[36] L.-W. Chang and M.-C. Wu, "A unified systolic array for discrete cosine and sine transforms", *IEEE Tran.s on Signal Processing*, vol. 39, no. 1, pp. 192–194, jan 1991.

[37] Hangu Yeo and Yum Hen Hu, "A modular high-throughput architecture for logarithmic search block-matching motion estimation", *IEEE Trans. Circuits*

Syst. Video Technol., vol. 8, no. 3, pp. 299 –315, 1998.

[38] Yeu-Shen Jehng, Liang-Gee Chen, and Tzi-Dar Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms", *IEEE Tran.s on Signal Processing*, vol. 41, no. 2, pp. 889–900, feb 1993.

[39] Sung Bum Pan and Rae-Hong Park, "Unified systolic arrays for computation of the dct/dst/dht", *IEEE Tran.s on Circuits and Systems for Video Technology*, vol. 7, no. 2, pp. 413–419, apr 1997.

[40] S. Panchanathan and M. Goldberg, "A systolic array architecture for image coding using adaptive vector quantization", *IEEE Tran. on Cir. and Sys. for Video Technology*, vol. 1, no. 2, jun 1991.

[41] M. Gok and C. Yilmaz, "Efficient cell designs for systolic Smith-Waterman implementations", in *2006. FPL '06. Int. Conf. on Field Programmable Logic and Applications*, aug. 2006, pp. 1–4.

[42] G. Urgese, M. Graziano, M. Vacca, M. Awais, S. Frache, and M. Zamboni, "Protein alignment HW/SW optimizations", in *19th IEEE Int. Conf. Electronics, Circuits and Syst. (ICECS)*, 2012, pp. 145–148.

[43] G. Causapruno, G. Urgese, M. Vacca, M. Graziano, and M. Zamboni, "Protein Alignment Systolic Array Throughput Optimization", *IEEE Trans. Very Large Scale Intgr. (VLSI) Syst.*, 2014.

[44] Mariagrazia Graziano, Stefano Frache, and Maurizio Zamboni, "A Hardware Viewpoint on Biosequence Analysis: What's Next?", *ACM J. Emerging Tech. Computing Syst.*, vol. 9, no. 4, 2013.

[45] A. Pulimeno, M. Graziano, and G. Piccinini, "UDSM trends comparison: From technology roadmap to ultrasparc niagara2", *IEEE Tran.s on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 7, pp. 1341–1346, july 2012.

[46] N.Z. Haron and S. Hamdioui, "Why is CMOS scaling coming to an end?", in *Design and Test Workshop, 2008. IDT 2008. 3rd Int.*, dec. 2008, pp. 98–103.

[47] K.K. Parhi and D.G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters. i. pipelining using scattered look-ahead and decomposition", *IEEE Tran.s on Acoustics, Speech and Signal Processing*, vol. 37, no. 7, pp. 1099–1117, jul 1989.

[48] K.K. Parhi and D.G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters. ii. pipelined incremental block filtering", *IEEE Tran.s on Acoustics, Speech and Signal Processing*, vol. 37, no. 7, pp. 1118–1134, jul 1989.

[49] H. Dawid, S. Bitterlich, and H. Meyr, "Trellis pipeline-interleaving: a novel method for efficient viterbi decoder implementation", in *1992 IEEE Int. Symposium on Circuits and Systems, 1992. ISCAS '92. Proceedings*, may 1992, vol. 4, pp. 1875–1878 vol.4.

[50] Sun-Yuan Kung, K.S. Arun, R.J. Gal-Ezer, and D.V. Bhaskar Rao, "Wavefront array processor: Language, architecture, and applications", *IEEE Tran.s on*

Computers, vol. C-31, no. 11, pp. 1054–1066, nov. 1982.

[51] G. Causapruno, M. Vacca, M. Graziano, and M. Zamboni, “Interleaving in Systolic-Arrays: a Throughput Breakthrough”, *IEEE Trans. Comput.*, vol. 64, no. 7, pp. 1940–1953, 2015.

[52] G. Fenga, “Adaptive Latency Insensitive Systolic Array”, Master’s thesis, Politecnico di Torino, December 2014.

[53] M. Mosleh, S. Setayeshi, M. Mehdi Lotfinejad, and A. Mirshekari, “FPGA implementation of a linear systolic array for speech recognition based on HMM”, in *2nd Int. Conf. Comput. and Automation Engineering (ICCAE)*, Feb 2010, vol. 3, pp. 75–78.

[54] C. K. Wijenayake, A. Madanayake, and L.T. Bruton, “FPGA-prototypes of differential-form 2D-IIR systolic-array DSP architectures for multi-beam plane-wave filters”, in *IEEE Workshop Signal Processing Systems (SIPS)*, Oct 2010, pp. 58–63.

[55] Qin Wang, Ang Li, Zhancai Li, and Yong Wan, “A Design and Implementation of Reconfigurable Architecture for Neural Networks Based on Systolic Arrays”, in *Proc. 3rd Int. Conf. Advances in Neural Networks - Vol. Part III*, Berlin, Heidelberg, ISNN’06, pp. 1328–1333, Springer-Verlag.

[56] Ioannis Panagopoulos, Christos Pavlatos, George Manis, and George Papakonstantinou, “A Flexible General-purpose Parallelizing Architecture for Nested Loops in Reconfigurable Platforms”, in *Proc. 17th Int. Conf. Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation*, Berlin, Heidelberg, 2007, pp. 20–30, Springer-Verlag.

[57] M. K. You, Y. J. Oh, and G. Y. Song, “Case study: Functional verification of a reconfigurable systolic array using truss”, in *IEEE 8th International Conference on ASIC*, Oct 2009, pp. 694–697.

[58] T. Ishimura and A. Kanasugi, “A design and simulation for dynamically reconfigurable systolic array”, in *ICCIT ’08. Third International Conference on Convergence and Hybrid Information Technology*, Nov 2008, vol. 2, pp. 172–175.

[59] A. K. Mishra and P. P. Jiju, “Low power, dynamically reconfigurable, memoryless systolic array based architecture for viterbi decoder”, in *International Conference onEnergy, Automation, and Signal (ICEAS)*, Dec 2011, pp. 1–5.

[60] Wei Jin, C.N. Zhang, and Hua Li, “Mapping multiple algorithms into a reconfigurable systolic array”, in *Canadian Conf. Electrical and Comput. Engineering (CCECE)*, 2008, pp. 001187–001192.

[61] G. Causapruno, F. Riente, G. Turvani, M. Vacca, M. Ruo Roch, M. Zamboni, and M. Graziano, “Reconfigurable Systolic Array: From Architecture to Physical Design for NML”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2016.

[62] F. Cairo, G. Turvani, F. Riente, M. Vacca, S.Breitkreutz-v. Gamm, M. Becherer, M. Graziano, and M. Zamboni, “Out-of-plane NML modeling

and architectural exploration", in *IEEE 15th International Conference on Nanotechnology (IEEE-NANO)*, 2015, pp. 1037–1040.

[63] M. Cofano, G. Santoro, M. Vacca, D. Pala, G. Causapruno, F. Cairo, F. Riente, G. Turvani, M.R. Roch, M. Zamboni, and M. Graziano, "Logic-in-Memory: A Nano Magnet Logic Implementation", in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2015, pp. 286–291.

[64] F. Cairo, M. Vacca, M. Graziano, and M. Zamboni, "Domain magnet logic (dml): A new approach to magnetic circuits", in *IEEE 14th International Conference on Nanotechnology (IEEE-NANO)*, 2014, pp. 956–961.

[65] M.D. Godfrey and D.F. Hendry, "The computer as von neumann planned it", *IEEE Annals of the History of Computing*, vol. 15, no. 1, pp. 11–21, 1993.

[66] J. Kawa, C. Chiang, and R. Camposano, "Eda challenges in nano-scale technology", *Proc. IEEE Custom Integrated Circuit Conference*, 2006.

[67] M.A. Maddah Ali and U. Niesen, "Fundamental limits of caching", *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, March 2014.

[68] N.D. Shah, Y.H. Shah, and H. Modi, "Comprehensive study of the features, execution steps and microarchitecture of the superscalar processors", *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–4, December 2013.

[69] Xian He Sun, "Remove the memory wall: from performance modeling to architecture optimization", *20th International Parallel and Distributed Processing Symposium*, April 2006.

[70] P. Jacob, A. Zia, O. Erdogan, P.M. Belemjian, J.W. Kim, M. Chu, R.P. Kraft, J.F. McDonald, and K. Bernstein, "Mitigating memory wall effects in high-clock-rate and multicore cmos 3-d processor memory stacks", *Proceedings of the IEEE*, vol. 97, no. 1, pp. 108–122, January 2009.

[71] C.C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the processor-memory performance gap with 3d ic technology", *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 556 – 564, November-December 2005.

[72] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman, *Real-Time Rendering 3rd Edition*, A. K. Peters, Ltd., Natick, MA, USA, 2008.

[73] Song Jun Park, "An analysis of gpu parallel computing", *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC)*, pp. 365–369, June 2009.

[74] Sang-Yun Lee and D.K. Schroder, "3d ic architecture for high density memories", *2010 IEEE International Memory Workshop (IMW)*, pp. 1–6, May 2010.

[75] Sang-Yun Lee and Junil Park, "Architecture of 3d memory cell array on 3d ic", *2012 4th IEEE International Memory Workshop (IMW)*, pp. 1–3, May 2012.

[76] Brad N. Engel, Nicholas D. Rizzo, Jason Janesky, Jon M. Slaughter, Renu Dave, Mark DeHerrera, Mark Durlam, and Saied Tehrani, "The science and

technology of magnetoresistive tunneling memory", *IEEE Transaction on Nanotechnology*, vol. 1, no. 1, March 2002.

[77] J. L. Ndai, P. Goel, A. Haixin, and Liu K. Roy, "An alternate design paradigm for robust spin-torque transfer magnetic ram (stt mram) from circuit/architecture perspective", *DAC*, pp. 841–846, January 2009.

[78] M. Bollo, G. Turvani, M. Zamboni, J. Das, S. Bhanja, and M. Graziano, "Design of nml circuits based on m-ram", *IEEE International Conference on Nanotechnology*, July 2015.

[79] Wei Jin, C.N. Zhang, and Hua Li, "Mapping multiple algorithms into a reconfigurable systolic array", in *2008. Canadian Conf. on Electrical and Computer Eng.*, may 2008.

[80] David Luebke and Greg Humphreys, "How gpus work", *IEEE Computers*, pp. 126–130, February 2007.

[81] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips, "Gpu computing", *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[82] Gang Chen, Guobo Li, Songwen Pei, and Baifeng Wu, "High performance computing via a gpu", *2009 1st International Conference on Information Science and Engineering (ICISE)*, pp. 238–241, December 2009.

[83] B. Buyukkurt and W.A. Najj, "Compiler generated systolic arrays for wavefront algorithm acceleration on FPGAs", in *Int. Conf on Field Prog. Logic and App., 2008.*, sept. 2008, pp. 655–658.

[84] K. Mani Chandy and J. Misra, "Systolic algorithms as programs", *Distributed Computing*, vol. 1, no. 3, pp. 177–183, 1986.

[85] *The pocket handbook of image processing algorithms in C*, Englewood Cliffs, N.J. : PTR Prentice Hall, 1993.

[86] M. Cofano, "Design of a Logic-in-Memory architecture for massive parallel algorithms", Master's thesis, Politecnico di Torino, October 2015.

[87] D. Pala, "Design of a Logic-in-Memory architecture for NanoMagnetic Logic", Master's thesis, Politecnico di Torino, March 2015.

[88] Fiaz Gul Khan, Omar Usman Khan, Bartolomeo Montrucchio, and Paolo Giaccone, "Analysis of fast parallel sorting algorithms for gpu architectures"', *Frontiers of Information Technology (FIT)*, pp. 173 – 178, December 2011.

[89] "http://www.samsung.com/semiconductor/products/dram/graphic-dram/".

[90] M. Vacca, S. Frache, M. Graziano, and M. Zamboni, "ToPoliNano: A synthesis and simulation tool for NML circuits ", *IEEE International Conference on Nanotechnology*, Aug. 2012.

[91] M. Vacca, S. Frache, M. Graziano, F. Riente, G. Turvani, M. Ruo Roch, and M. Zamboni, "ToPoliNano: NanoMagnet Logic Circuits Design and Simulation", in *Field-Coupled Nanocomputing*, Neal G. Anderson and Sanjukta

Bhanja, Eds., Lecture Notes in Comput. Science, pp. 274–306. Springer Berlin Heidelberg, 2014.

[92] D. MacKay, *Information Theory, Inference, and Learning Algorithms*, CMU-CS. Hardback, 2003.

[93] J. Grosschadl, "A Low Power Bit-Serial Multiplier For Finite Fields GF(2^m)", in *The 2001 IEEE International Symposium on Circuits and Systems*, Sydney, NSW, May 2001, vol. 4, pp. 37–40.

[94] M. Graziano M. Vacca and M. Zamboni, "Nanomagnetic Logic Microprocessor: Hierarchical Power Model", *IEEE Transactions on VLSI Systems*, Aug. 2012.

[95] "Comsol Multiphysics", http://www.comsol.com/.