POLITECNICO DI TORINO

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

Tesi di Dottorato

Multilevel Modeling and Architectural Solutions for Emerging Technology Circuits

Juanchi Wang

Tutore
prof. Mariagrazia Graziano
prof. Maurizio Zamboni

Coordinatore del corso di dottorato

prof. Ivo Montrosset

March 2016

Summary

In the last decades, the main driving force behind the astonishing development of CMOS technology, was the transistor scaling process. The reduction of transistor sizes has granted a continuous boost in circuits performance. But now that the scaling process is reaching its physical limits, researchers are forcusing on new emerging technologies.
Research on these new technologies is usually carried on using a traditional approach. Some studies concentrate on new devices without analyzing circuits based on them. Other studies analyze circuit architectures without considering devices characteristics and limitations. However, given that the nature of emerging technologies can be very different from CMOS, new research methodologies should be adopted. A clear link between device and architectural analysis is necessary to understand the true potential of the technology under study.
The objective of this PhD thesis is the analysis of emerging technologies using an innovative methodology. Using complex and realistic circuits as benchmark, high level models are built incorporating low level device characteristics. This methodology strongly links device and architectural levels.
The methodology was applied to two emerging technologies: NanoMagnet Logic (NML) and Nanoscale Application Specific Integrated Circuits (NASIC). A brief introduction of fundamental information on the two technologies is given in Chapter 1.
The application of the methodology on NML technology is divided in two parts (Chapter 2): i) architecture-level timing and performance analysis and circuits optimization; (ii) area and power estimations using VHDL modeling. Starting from an exhaustive analysis of the effects and the consequences derived by the presence of loops in a complex NML sequential architecture, solutions have been proposed to address the problem of signal synchronization, and optimization techniques have been explored for performance maximization. Area and power estimations have been performed on multiple NML architectures in order to obtain a complete evaluation on the implementation of NanoMagnet Logic in comparison with the CMOS technology.

Chapter 4 is dedicated to NASIC technology with basic principles described in Chapter 3. Basic computational blocks are implemented using a multilevel modeling approach. A detailed analysis of circuits' area and power estimations is obtained. Techniques to optimize the area of circuits at the cost of reduced throughput were also investigated.
The research activity presented in this thesis highlights the development of an innovative methodology based on high-level models that embed information obtained from physical level simulations. By exploiting this methodology to different emerging technologies, such as NML and NASIC, it allows to efficiently analyze circuits and therefore to bring architectural improvements.

Contents

List of Tables

List of Figures

Chapter 1

Introduction to NanoMagnet Logic

NanoMagnet Logic (NML) is a new technology based on the field coupled principle. Devices are built using identical cells. Information propagates through magnetic field coupling among neighbor nanomagnets. NML main interest lies in its magnetic nature, that leads to circuits with low dynamic power consumption and now leakage, circuits that can act both as logic and memory devices.

1.1 Technology Background Quantum-dot Cellular Automata

NML Technology is one of the implementation of Quantum dot Cellular Automata (QCA). QCA technology has received in recent years a lot of attention thanks to its nanoscale dimensions and a new way of computation and information elaboration.

1.1.1 QCA Logic

Basic QCA cells are quantum wells which confine free electrons in a square shaped nanostructure containing four quantum dots that can hold a single electron each and sit at four corners [1]. Electrons are able to tunnel between the dots but are not able to escape the cell. Only two stable electric charge states, representing logic "0" and "1", are possible by Coulombic repulsion as shown in Figure 1.1 A).

Due to the electrostatic interaction among neighbor cells, QCA cells will reach a stable polarization state depending on the state of neighbor cells. This phenomenon can be seen as a signal propagation (Figure 1.1 B)), which can be employed to construct digital systems. To build logic circuits, a set of logic gates (Figure 1.2), such as Inverters, Majority Voters, and Crosswire blocks must be developed. As

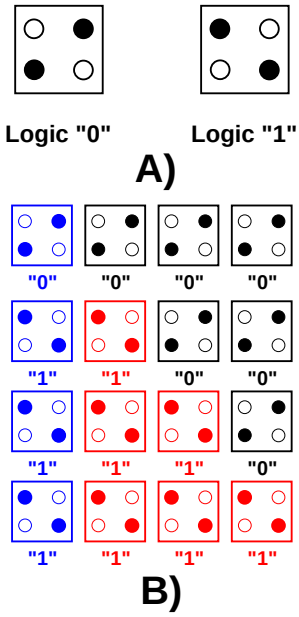


Figure 1.1. Basic 4-dot QCA cells and logic states.

shown in Figure 1.2B), a majority voter computes the output according to the states of three input cells. It can implement AND/OR gates by forcing one input to be "0" or "1" [2]. In a crosswire block, QCA cells rotated by 45 degrees Figure 1.2 C), allows to cross propagate two signals on the same flat plain without influencing each other [3].

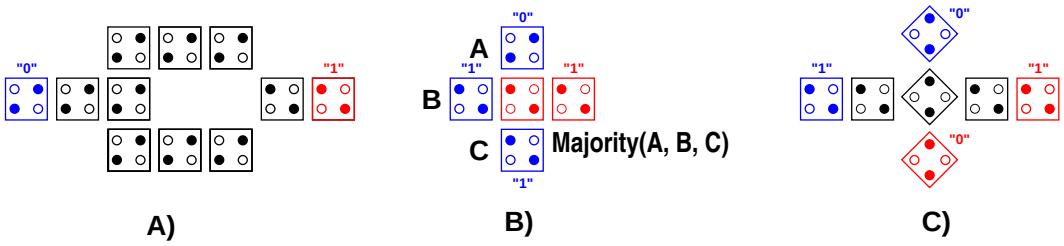


Figure 1.2. Basic QCA logic gates. A)Inverter B)Majority Voter C)Crosswire

1.1.2 QCA Clock Mechanism

By introducing the concept of **"Clock"** mechanism and modifying the 4-dot QCA structure into 6-dot (Figure 1.3), it is possible to set the cells in a metatstable state ("NULL") by applying an external electric field. When the electric field is removed, the cell reach one of the two stable states according to its neighbors, propagating therefore the information through the circuit.

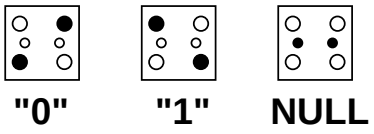


Figure 1.3. Basic 6-dot QCA cells with 3 states.

As depicted in Figure 1.4, a long QCA wire can be divided into **"clock zones"**, each of which is driven by separated clock signals. These clock signals are derived from the same signal with 4 different phases shifting. QCA cells can be therefore in one of four possible states (**HOLD**, **RELEASE**, **RELAX**, **SWITCH**).

When a clock zone is in HOLD phase, the potential barrier of the QCA cells are increased by the external clock field ($V = Vmax$), making them insensible to neighbor cells and impossible to switch to the other state. Then the clock zone enters into RELEASE phase, with the clock signal decreasing to $-Vmax$ gradually. Cells reach "NULL" intermediate state. In the next RELAX phase, by stablizing the field $V = -Vmax$, the previous logic states of all the cells are erased. With the clock signal rising again up to $+Vmax$, there is the transition to the new logic states according to the new input, which is the so-called SWITCH phase.

An example of information propagation is depicted in Figure 1.4. It can be noticed that in time step1, with clock zone 1 in HOLD, clock zone 2 switches according to the cells in clock zone 1. At the same time, clock zone 3 is totally relaxed and can not influence the switching in clock zone 2. Therefore, after a complete clock cycle of 4 phases, the input signal "0" is propagated to the end of the wire.

1.1.3 QCA Implementation

The theoretical principle of the QCA can be implemented in different ways. There are four physical implementation of QCA, which are briefly discussed below.

- **Metal QCA**. As shown in Figure 1.5(A), the metal QCA are made with isles of aluminum, used to represent the dots, on a substrate of silicon dioxide (SiO2). The dots are connected to each other by tunnel-junctions which allow

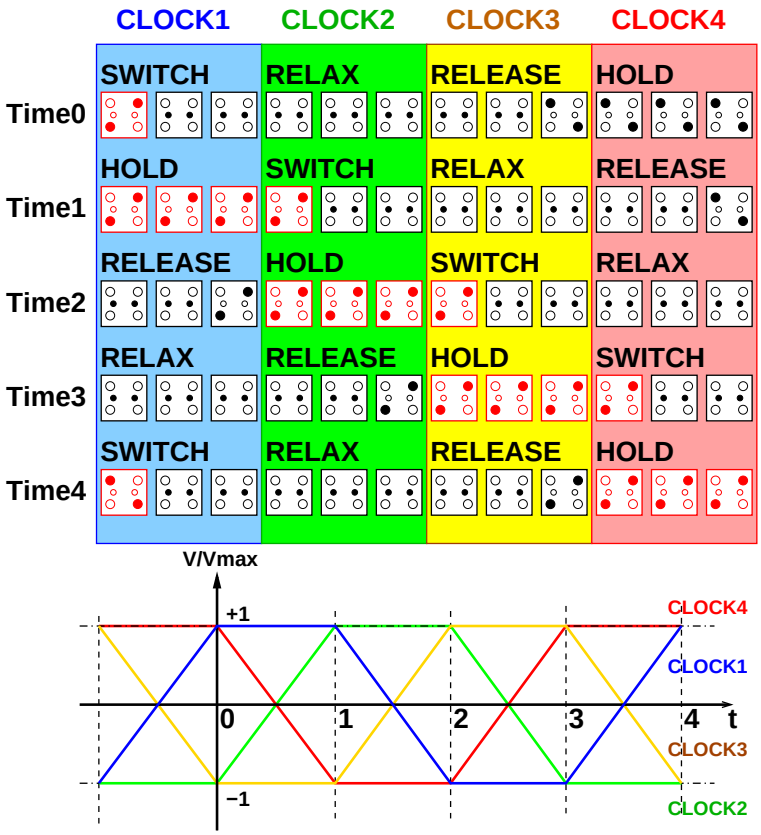


Figure 1.4. QCA clock mechanism with 4-phase clock signal.

the electrons to be exchanged among neighbor dots. This technology has been the first implementation, while its main disadvantage is that it works only at a temperature near the absolute zero [7].

- **Semiconductor QCA**. Complex heterostructures of Si-Ge or GaAs are used to create quantum dots that are able to trap electrons [7] as shown in Figure 1.5(B). The main limitations of this QCA type are the necessity to work at near absolute zero temperature and the huge impact of fabrication defects.

- **Molecular QCA**. It employs single molecules with complex structures as basic cell. Each molecule contains multiple oxide-reduction centers as shown in Figure 1.5(C). Electrons can react with every center inside the molecule, changing the spatial distribution of the electric charge, and thus the logic value associated to it [7]. This is one of the most promising implementation technologies, since the molecules are very small (a few nanometers), allowing

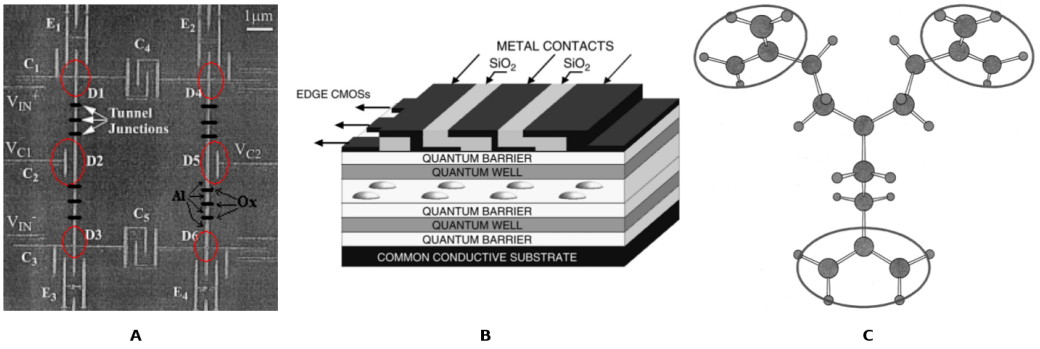


Figure 1.5. A) Metal QCA structure B) Semiconductor QCA structure C) Molecular QCA structure. *A) R. K. Kummamuru et al. "Operation of a Quantum-Dot Cellular Automata (QCA) Shift Register and Analysis of Errors", IEEE Transactions on Eletron Devices, vol. 50, n. 9, Sept. 2003. B) A. Khitun et al. "Multi-functional edge driven nano-scale cellular automata based on semiconductor tunneling nano-structure with a self assembled quantum dot layer", Superlattices and Microstructures, vol. 37, pp. 55-76, 2005. C)C.S. Lent et al. "Clocked Molecular Quantum-Dot Cellular Automata", IEEE Transactions on Electron Device, vol. 50, no. 9, september 2003.*

to reach a very high device density, and it permits an operating frequency of some THz. Moreover, with respect to Metal QCA, it can work under room temperature. The problem is that until now, with the up-to-date technology, Molecular QCA fabricated is challenging because it requires the ability of manipulating single molecule [3].

- **NanoMagnet Logic**. The basic cell of NML is a single domain nanomagnet, and is one of the most promising implementation of QCA. With respect to Molecular QCA, it has a bigger dimension, and can only reach a basic frequency of some hundred of MHz theoretically. However, it is possible to implement it with current technology, allowing to experiment and study the QCA principle so that most of the achievements can be adapted in a near future to molecular QCA, as soon as this solution becomes feasible [7].

1.2 NanoMagnet Logic (NML)

NanoMagnet Logic, also known as Magnetic QCA, is one of the promissing technology based on Quantum Cellular Automata (QCA) principle. The advantage of this technology resides in its magnetic nature which leads to expected low power consumption.

1.2.1 NML Basic Cell

The basic cell of NML is a retangular shaped single domain nanomagnet. Due to its shape, only two stable magnetization states are possible, which can be used to rappresent logic states “0” and “1” (Figure 1.6).

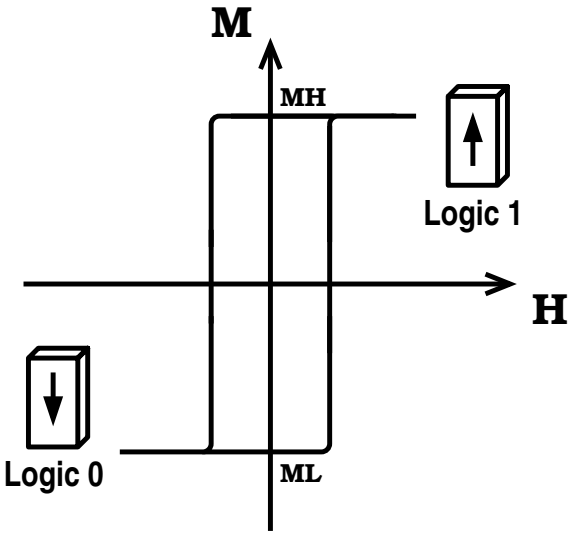


Figure 1.6. NML basic cells with stable magnetizations rappresenting logic states

By placing the nanomagnets in cascade horizontally or vertically, signals can be propagated due to magnetic coupling among neighbor magnets as shown in Figure 1.7.

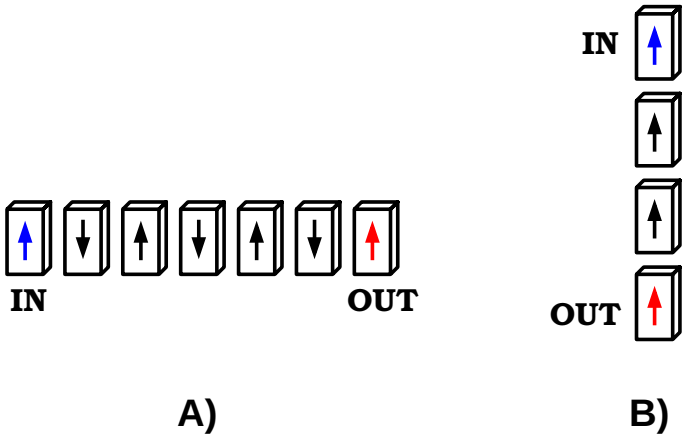


Figure 1.7. NML wire configurations. A) Horizontal wire. B) Vertical wire.

1.2.2 NML Logic Gates

Several basic logic gates can be built with NML cells, such as inverter, crosswire, and majority voter.

- Inverter
  As can be noticed, a horizontal wire is constructed by placing an odd number of nanomagnets between input and output. Therefore, by placing an even number of nanomagnets within the same total length, the signal at the output is inverted as shown in Figure 1.8 A).

- Crosswire
  A crosswire block is a functional block which allows to the cross two signals on the same plane without interferences. This characteristic can be achieved by organizing the square shaped nanomagnets according to Figure 1.8 B).

- Majority Voter
  The output value of a majority voter (Figure 1.8 C)) is equal to the majority of the inputs.

- AND/OR Gate
  By fixing an input signal to "0" or "1", a majority voter can be converted to an AND/OR gate. Another possibility is to exploit the shape dependent switching behavior. Cutting a nanomagnet leads to an asymmetric shape, in order to obtain a preferential magnetization direction for the output signal (Figure 1.8 D)).

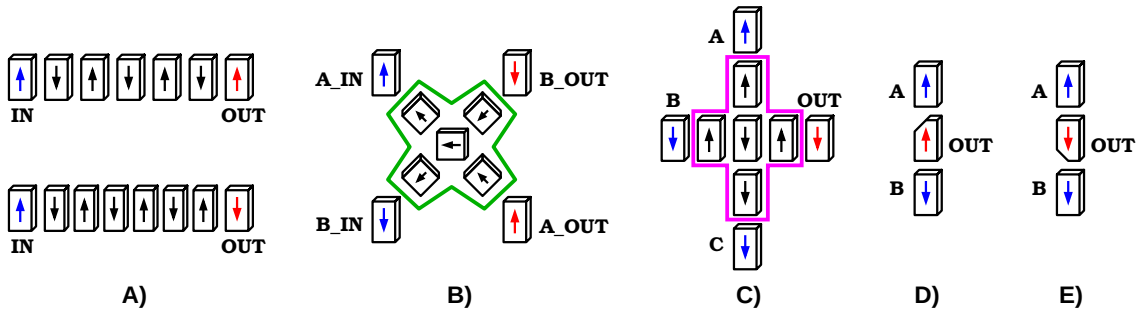


Figure 1.8. NML basic logic gates. A) Inverter. B) Crosswire. C) Majority Voter. D) OR gate. E) AND gate.

1.2.3 Clock Mechanism

In order to lower the energy barrier during the switching between two stable states, a magnetic field [8] can be exploited to force the nanomagnets into a transient metastable state as shown in Figure 1.9. When the field is removed, the nanomagnets re-enter into one of the stable states depending on the value of neighbor magnets. This mechanism is called clock.
The magnetic field can be generated by a current flowing through a wire placed under the magnets plane (Figure 1.10 A)).

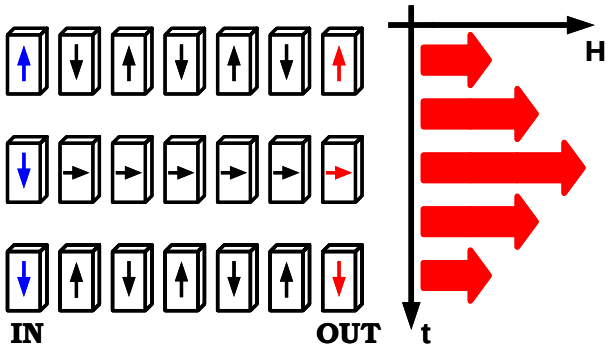


Figure 1.9. NML clock mechanism with magnetic field forcing nanomagnets into an intermediate unstable state.

As shown in [9] a current of 545 mA on a 1 um width wire is required to successfully switch magnets in the RESET state. This is a very high value of current that leads to a very high power consumption, wasting the advantage related to the tiny power dissipation due to magnet switching. To reduce power consumption other mechanisms were proposed. For example in the STT-current approach [21] magneto-tunnel junctions (MTJ) are used as basic element. MTJ can be reset by a current flowing through them leading to a power consumption of just 1.6fJ for each magnet (Figure 1.10 B)). Alternatively, in [10] an innovative clock system is proposed based on the use of an electric field instead of a magnetic field. With this clock solution magnets are deposited on a piezoelectric layer (Figure 1.10 C)). When an electric field is applied, the strain of the piezoelectric layer induces a mechanical stress on the magnets forcing them in the RESET state. With this clock solutions an energy of just 2 pJ is required to switch magnets [10] allowing to build true low power circuits.

1.2.4 Multiphase Clock System

To reduce the influence of thermal noise during magnet switch, a NML circuit must be divided into “clock zones”. A clock zone is a small circuit area composed by a

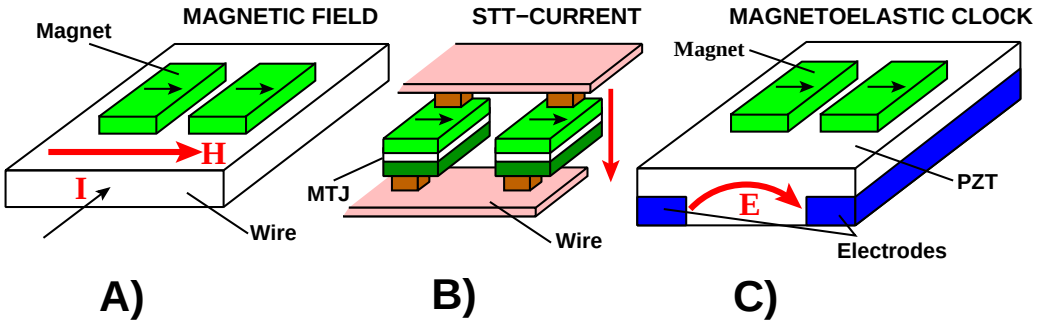


Figure 1.10. NML clock implementation. A) Magnetic field. B) STT-current. C) Magnetoelastic. *J. Wang et al. "Biosequences analysis on NanoMagnet Logic", International Conference on IC Design and Technology (ICICDT), May 2013.*

limited number of magnets. A 3-phase overlapped clock signal is applied in order to propagate information [11].
Taking as an example a NML wire composed by 3 clock zones (Figure 1.11), magnets can be in three different states ("HOLD", "RESET" and "SWITCH").

- Time Step 1: the clock zone 1 is in "HOLD" state, all the magnets in the zone are in a stable state. The nanomagnets in clock zone 2 switch ("SWITCH" state) in anti-ferromagnetic order one by one, as a domino effect, propagating the signal from left to right. In the meantime, clock zone 3 is in the "RESET" state, so it has no influence on the switching magnets.

- Time Step 2: nanomagnets in clock zone 1 are in the "RESET" state, clock zone 2 has finished the switching process and has entered into "HOLD" state. By removing the clock signal in clock zone 3, the magnets start to switch.

- Time Step 3: clock zone 1 starts switching according to the new input, and clock zone 2 is now reset. At this point, with clock zone 3 being in "HOLD" state, the signal propagation reaches the end of the wire.

A NML wire is equivalent to a CMOS shift register, and the multiphase clock system leads to an intrinsic pipelined behavior [12]. Each group of 3 consecutive clock zones has exactly a delay of one clock cycle.

In order to reduce the energy necessary for the magnet to switch, the clock signal should have a rise time of at least 8 to 10 ns for the concept of adiabatic switching. When the number of magnets is higher than five, a long fall time is necessary to assure that magnets switching occurs with a reduced probability of error [13]. The maximum allowed clock frequency is around 1 GHz, while the more realistic value turns to be between 10 and 100 MHz considering all constraints due to thermal noise and clock zone layout [14].

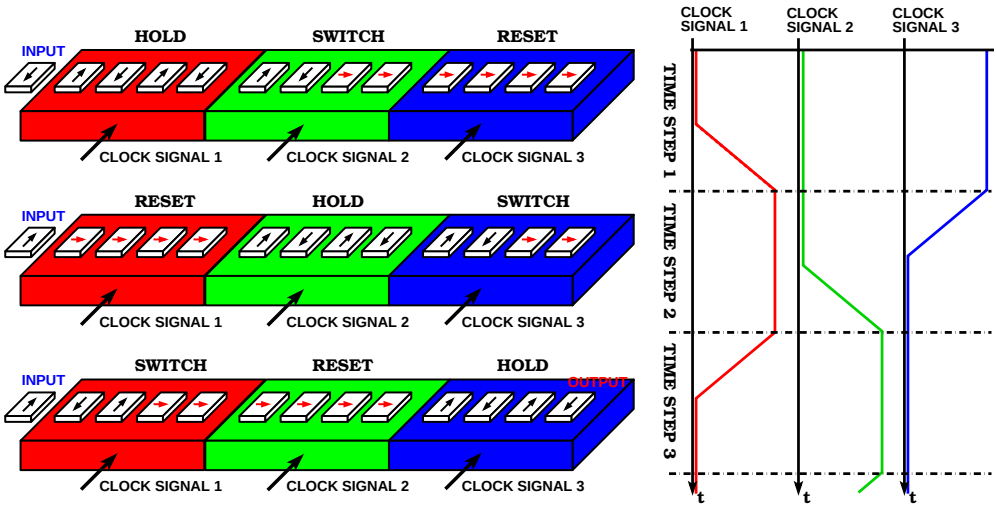


Figure 1.11. NML 3-phase clock system.

1.2.5 Clock Zone Layout

With the 3-phase clock system described previously, NanoMagnet Logic can only propagate in one direction. The delay of a signal which passes through 3 clock zones is equal to one clock cycle. The circuits latency is equivalent to one third of the total number of clock zones. This is the so-called "Layout = Timing" problem (Figure 1.12 A)). Circuits delay depend on the layout of circuits. Careful layout design is required to have a perfect signal synchronization. This traditional layout is not feasible in design of complex NML circuit. As a consequence, a "Snake-like" clock zone layout is proposed to allow feedback signal propagation (Figure 1.12 B)). It helps reducing the "Layout=Timing" problem. Figure 1.13 represents a 3D view of the snake clock wires used to generate the clock signals.

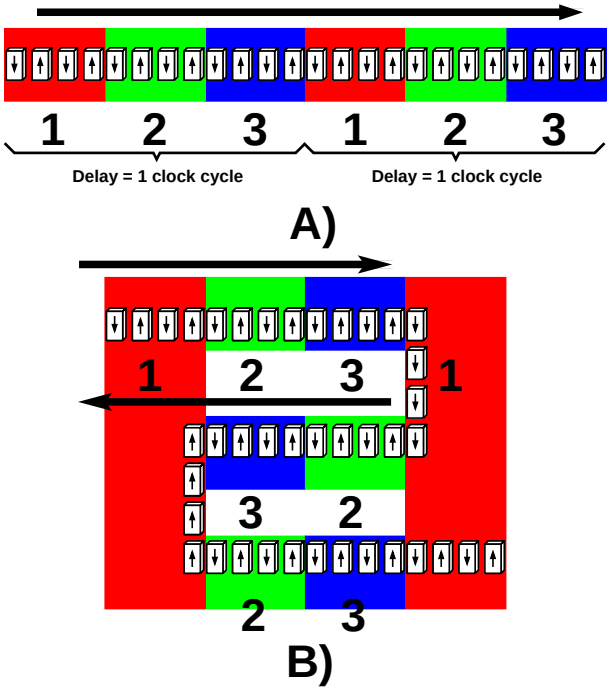


Figure 1.12. 3-phase clock zone layout. A) Traditional layout. B) Snake-like layout.

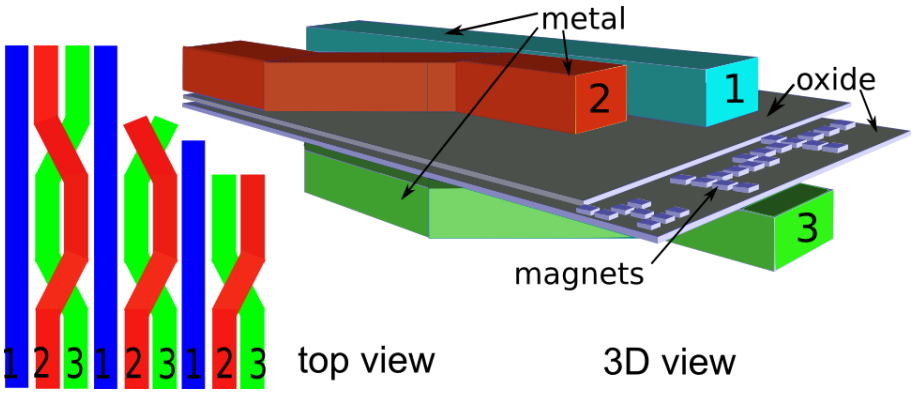


Figure 1.13. Snake-like layout physical view. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, Oct. 2015.

1.2.6 VHDL Modeling

VHDL behavioral model

Independent of the clock mechanism chosen, as described before, the intrinsic pipelined behavior of circuits can be modeled by combinational logic gates with registers controlled by clock signals as shown in Figure 1.14.

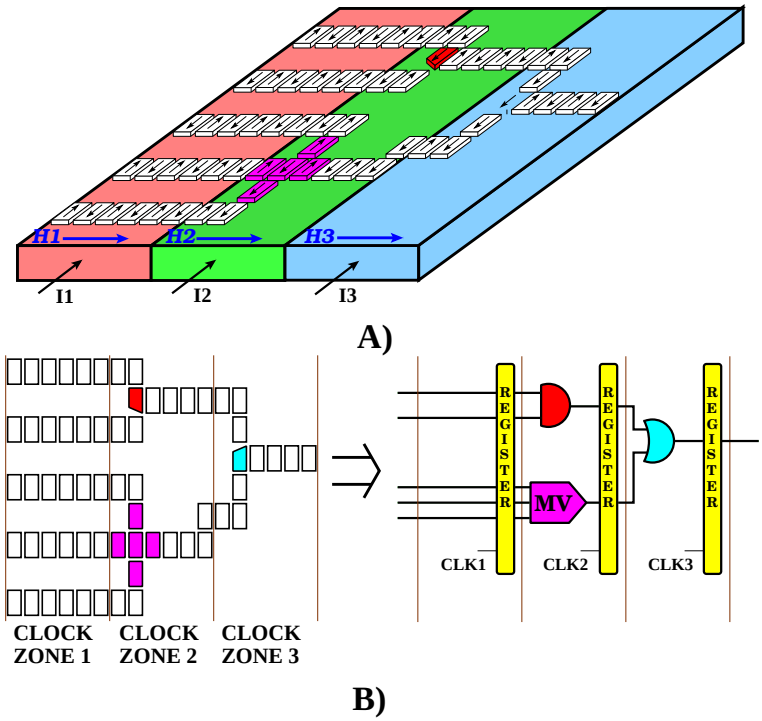


Figure 1.14. NML circuit VHDL modeling. *J. Wang et al. "Biosequences analysis on NanoMagnet Logic", International Conference on IC Design and Technology (ICICDT), May 2013.*

Registers simulate signals propagation delay, while ideal logic gates model the circuits behavior. An RTL model of NML circuits can therefore be easily described using VHDL language. In the following, the VHDL code describing the circuit of Figure 1.14 is reported.

```
-- Example NML circuit with VHDL modeling
-- It occupies 3 clock zones (1 clock cycle)


library ieee;
use ieee.std_logic_1164.all;

entity ExampleNML is
```

```
        port (
                clk     :       in std_logic_vector (3 downto 1);
                ANDIn1  :       in std_logic;
                ANDIn2  :       in std_logic;
                MVIn1   :       in std_logic;
                MVIn2   :       in std_logic;
                MVIn3   :       in std_logic;
                Output  :       out std_logic);

end ExampleNML;

architecture STRUCTURAL of ExampleNML is

        signal S1       : std_logic_vector(4 downto 0);
        signal S1_reg0  : std_logic_vector(4 downto 0);
        signal S2       : std_logic_vector(1 downto 0);
        signal S2_reg0  : std_logic_vector(1 downto 0);
        signal S3       : std_logic_vector(0 downto 0);
        signal S3_reg0  : std_logic_vector(0 downto 0);

component MV is
        port(
                A_in: in std_logic;
                B_in: in std_logic;
                C_in: in std_logic;
                out_MV: out std_logic);
end component;

component gate_AND is
        port(
                A_in: in std_logic;
                B_in: in std_logic;
                out_AND: out std_logic);
end component;

component gate_OR is
        port(
                A_in: in std_logic;
                B_in: in std_logic;
                out_OR: out std_logic);
end component;

component register_generic is
        generic ( NBIT : integer );
        port(
                 CLK:   in std_logic;
                 D:     in std_logic_vector (NBIT-1 downto 0);
                 Q:     out std_logic_vector (NBIT-1 downto 0));
end component;

        begin

        -- enter into Stage1
                S1(0)<= ANDIn1;
                S1(1)<= ANDIn2;
                S1(2)<= MVIn1;
                S1(3)<= MVIn2;
                S1(4)<= MVIn3;

        -- passing the signals through REG1
        REG1:   register_generic
                generic map (5)
```

```
            port map (clk(1), S1, S1_reg0);
        --endREG

        -- enter into Stage2
        and_stage2_0: gate_AND port map (S1_reg0(0), S1_reg0(1), S2(0));

        mv_stage2_0: MV port map (S1_reg0(2), S1_reg0(3), S1_reg0(4), S2(1));

        -- passing the signals through REG2
        REG2:   register_generic
                generic map (2)
                port map (clk(2), S2, S2_reg0);
        --endREG

        -- enter into Stage3
        or_stage3_0: gate_OR port map (S2_reg0(0), S2_reg0(1), S3(0));

        -- passing the signals through REG3
        REG3:   register_generic
                generic map (1)
                port map (clk(3), S3, S3_reg0);
        --endREG

        -- assign the output

        S  <= S3_reg0(0);

end STRUCTURAL;
```

Area and Power model

The VHDL code embeds an area estimator. For each logic block, the total number of magnets is calculated summing the number of magnets of each logic gate. A multiplicative factor takes into account the interconnections overhead. Area is therefore calculated from the total number of magnets. Details are reported in [15].

Regarding Power consumption, as stated in [11], there are two main power losses in NML circuits: the power loss to force magnets in the RESET state and the loss in the clock generation. The first component is calculated by multiplying the switching energy of each magnet for the total number of magnets and for the frequency. The second component is calculated estimating the clock wires length from the total circutis area. Knowing the wire length and by defining some parameters as shown in Figure 1.15, a series of equations are employed to calculate wires section and resistance.

$$\begin{cases} Swire = (Width\_zone - Width\_sep) * Wire\_thick \\ \\ Rwire\_i = Resistivity * \frac{Lwire\_eff\_i}{Swire} \end{cases} \tag{1.1}$$

Knowing the wires resistance, it is possible to estimate clock power consumption by estimating joule losses through the wires.

Parameter name	Default value	Explanation
h_zone	6.0e-7	Height of clock zone (meters)
h_zone_sep	2.0	Vertical separation between zones, where no magnets are allowed.
Width_zone	7.0e-7	Width of a clock zone (meters)
Width_zone_mag	10.0	Width of a clock zone in terms of magnets
Wire_thick	6.0e-7	Clock wire thickness (meters)
Wire_sep	2.0e-8	Space between clock wires (meters)
Wasted_Space	2.0	Relative area used by components (magnets + separation spaces)
h	1.0e-7	Height of a nanomagnet (meters)
W	5.0e-8	Width of a nanomagnet (meters)
vert_space	2.0e-8	Vertical separation space between magnets (meters)
oriz_space	2.0e-8	Horizontal separation space between magnets (meters)
Wire_curves	1.1	Overhead due to path wires for connecting different wires pieces
OV1_logic_gate_level	1.1	Interconnect overhead inside a logic gate
OV2_intermediate_level	2.0	Interc. overhead in small clusters of logic gates (full_adder, mux, registers..)
OV3_logic_block_level	1.5	Interconn. overhead inside functional element (alu, counter,....)
OV4_top_level	1.2	Interconnect overhead due to logic blocks interconnection
I_max	1.0e-3	Maximum current flowing in clock wires (Ampere)
clock_overlap	11.0	Clock overlap percentage
Resistivity	1.78e-8	Clock wire resistivity
T_clock	9.0e-9	Clock period (seconds)
Energy_mag	30.0*KT	Energy associated to a single magnet switch (with T=300 and K=boltzman constant)
N_mag_mv	5.0	Number of magnets in a Majority Voter
N_mag_inv	7.0	Number of magnets in an Inverter

Figure 1.15. Parameters and constants used in the NML power model. *M. Vacca et al. "NanoMagnet Logic Microprocessor: Hierarchical Power Analysis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, pp. 1410-1420, 2012.*

$$Mag\_Power = \sum_{i=1,2,3} N_i * \frac{Energy\_mag}{T\_clock} \tag{1.2}$$

$$P\_joule\_total = \sum_{i=1,2,3} Rwire\_i * I\_max^2 \tag{1.3}$$

Model details can be found in [15]. It is important to underline that the effectiveness of this model relies on the chosen values of overheads such as interconnections overhead and wire curves. Therefore, the constants were extrapolated starting from NML theory and low level simulations, taking into account all the physical and layout constraints actually known. This model is later on employed on the chosen complex NML circuit, the Smith-Waterman Systolic Array architecture in Section 2.4, allowing technological comparison with CMOS technology.

Chapter 2

NanoMagnet Logic Architecture Analysis

In order to link device and architectural analysis, a complex architecture is chosen as benchmark using multilevel models that consider the information gained from physical level finite element simulator. Basic structures and gates are validated through physical level simulator taking into account the limitations in magnets placement. This methodology that includes both fast behavioral simulation and area and power evaluation, allows us to gain significant information on NML performance.

2.1 Smith-Waterman Systolic Array Architecture Implementation

2.1.1 Biosequences Alignment Analysis

The chosen architecture as NML benchmark is the Smith-Waterman (SW) Systolic Array Architecture that is employed in biosequences alignment analysis. Biosequences alignment analysis is an important application iin the bioinformatics field. Being the essential constituents of animal and plant cells, proteins are manufactured by the instructions encoded in Deoxyribonucleic Acid (DNA). It contains the genetic information (genes) to guide the development of organs, to distinguish one species from other living beings and to pass to its descendants from generation to generation. By identifying the similarities between two biosequences, it is possible to reconstruct the evolutionary pathway that led to differentiation of species or to understand the genetic cause of a disease [16].

Each protein is a long chain of Amino Acids (AAs), which are normally represented by alphabetical characters as in Figure 2.1. Therefore, the biosequence

analysis is commonly done by comparing one sequence of AAs (**Query**) to other sequences taken from databases (**Subject**) that have been already developed. There are various sequence alignment methods found in literature. Among them there is **Smith-Waterman** algorithm which is mostly used for finding small regions of local similarity between distantly related biosequences [17].

Letter	Amino acids	Letter	Amino acids	Letter	Amino acids	Letter	Amino acids
A	Alanine	Q	Glutamine	L	Leucine	S	Serine
R	Arginine	E	Glutamic acid	K	Lycine	T	Threonine
N	Asparagine	G	Glycine	M	Methionine	W	Tryptophan
D	Aspartic acid	H	Histidine	F	Phenylalanine	Y	Tyrosine
C	Cysteine	I	Isoleucine	P	Proline	V	Valine

Figure 2.1. Amino Acide alphabetical character representation. *L. R. Murphy et al. "Simplified Amino Acid Alphabets for Protein Fold Recognition and Implication for Folding", Protein Engineering, vol. 13, pp. 149-152, 2000.*

2.1.2 Smith-Waterman Systolic Array Architecture

The Smith-Waterman algorithm evaluates the alignment between two AA sequences using a score mechanism that identify the percentage of similarity between them. As results, it outputs the best alignment identified by the maximum score as shown in Figure 2.2 [3].

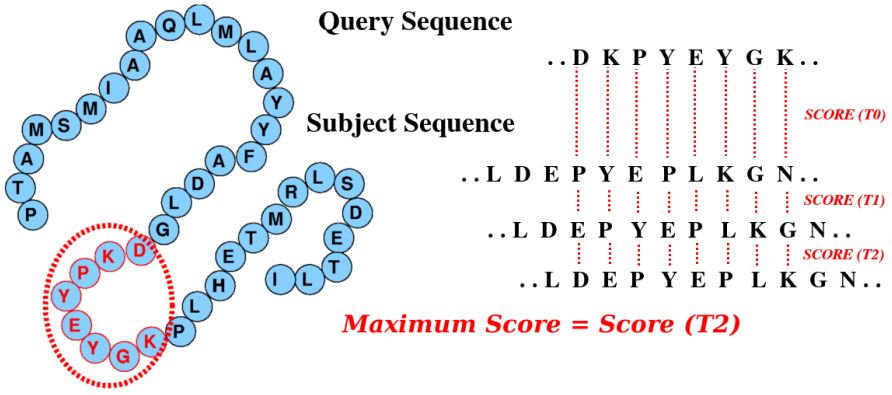


Figure 2.2. Biosequence alignment analysis principle.

The algorithm can be parallelized and implemented using a Systolic Array (SA) to increase the performance and to accelerate the computation. As shown in Figure

2.4, Systolic Arrays are composed of Processing Elements (PEs) which can be organized in a matrix structure, or a linear structure or other parallel structures. The Processing Elements are identical processors that receive data from neighbor PEs and compute results that are passed to the next PEs.

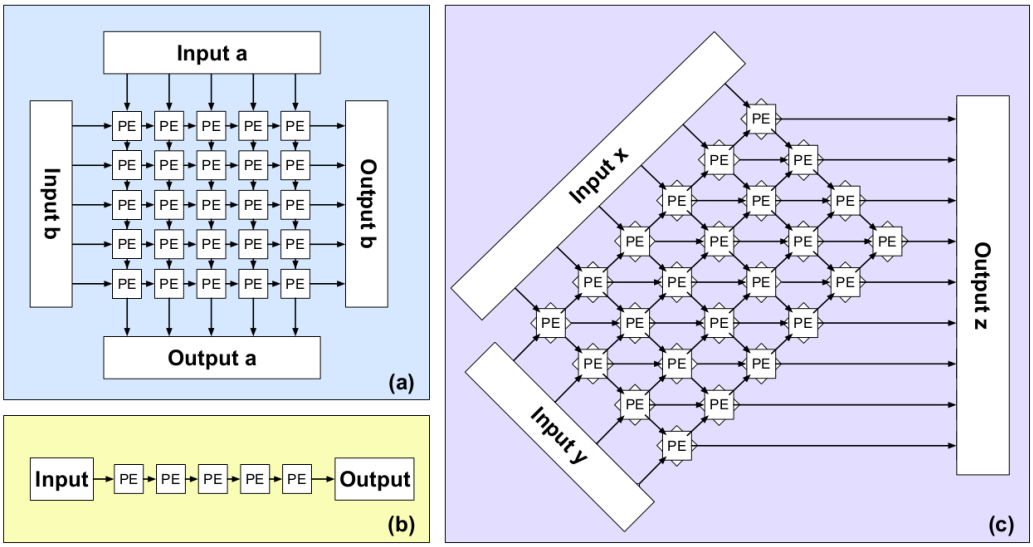
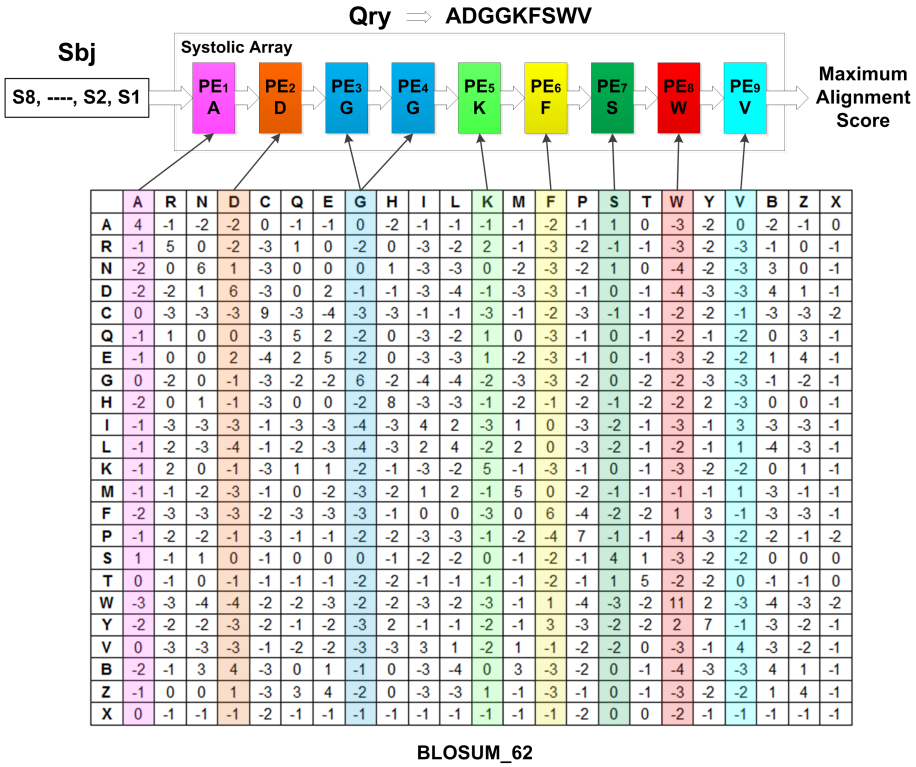


Figure 2.3. Systolic Array Structures. A) Matrix Systolic Array. B) Linear Systolic Array. C) Special Systolic Array. *G. Causapruno et al. "Protein Alignment Systolic Array Throughput Optimization", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 1, pp. 68-77, 2014.*

As depicted in Figure 2.4, a linear systolic array is employed for the Smith-Waterman algorithm is employed. Each amino acid of the *Query sequence* is stored in one PE. Therefore, the more PEs there are in the systolic array, the longer Query sequences can be analyzed. The systolic array receives the *Subject sequence* AAs one by one at the input, passes them through the entire structure and generates the alignment score at the end. By sending multiple sequences in series, the circuit is able to compute the maximum alignment score at the systolic array output.

The detailed CMOS implementation ([18]) of a S-W Processing Element can be divided into two major blocks: Configuration block (*PE_Config*) and Computation block (*PE_Calc*) (Figure 2.5). The configuration block is responsible of generating control signals such as register read/write signals, while the computation block (Figure 2.6) contains the memory to store the alignment scores between the representing Query amino acid with all 23 amino acids. Using these values, the computational block is able to calculate the local alignment score. The datapath of the computational block is composed mostly by adders, used to design comparator blocks,



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	3	-3	-2	0	-1	-4	-3	-3	4	1	-1
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1

BLOSUM_62

Figure 2.4. Smith-Waterman systolic array architecture with processing elements .*G. Urgese, "Analysis and Design of an Optimized HW Accellerator for Protein Alignment", Master thesis, Politecnico di Torino, Dept. Eletr., Torino, Italy, Sept. 2012.*

multiplexers and registers. A detailed description of the Smith-Waterman architecture and its hardware implementation can be found in [18].

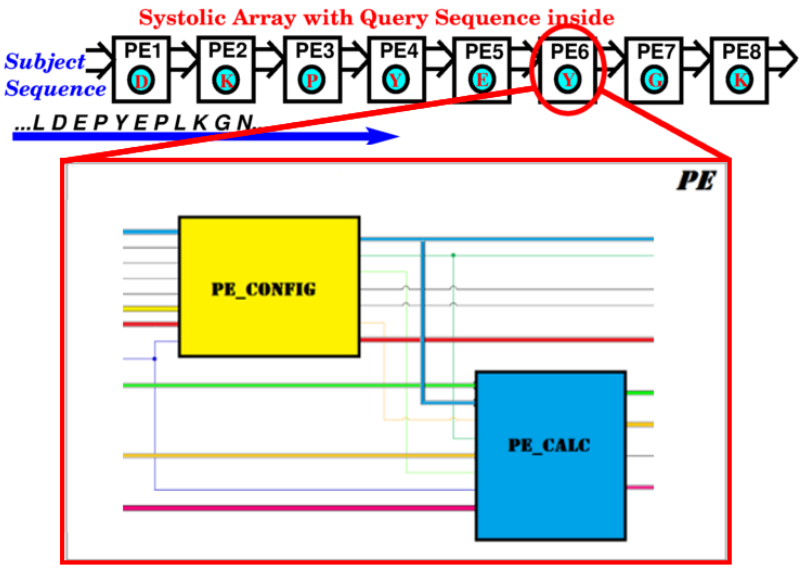


Figure 2.5. Smith-Waterman systolic array architecture with processing elements.

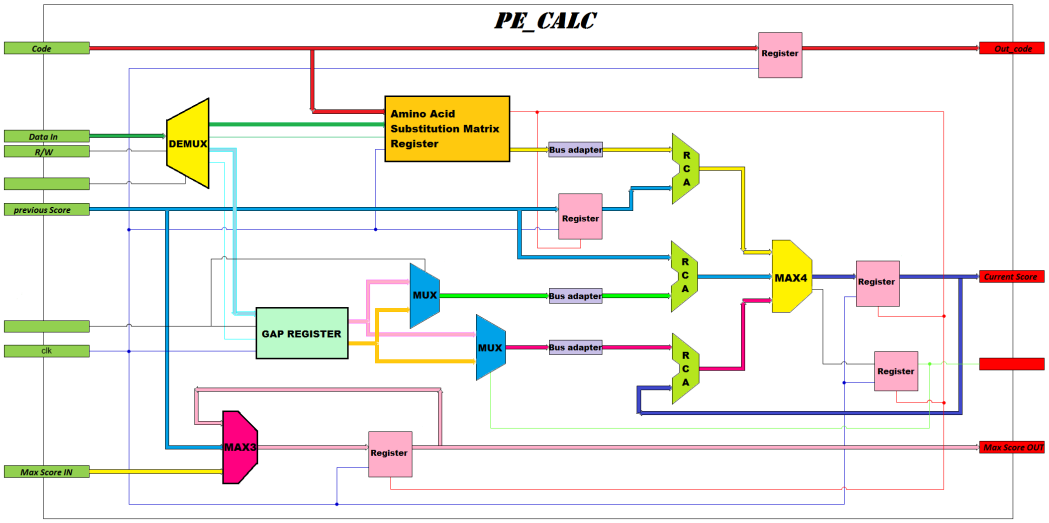


Figure 2.6. Structure of PE_Calc block. *G. Urgese, "Analysis and Design of an Optimized HW Accellerator for Protein Alignment", Politecnico di Torino, Dept. Eletr., Torino, Italy, Sept. 2012.*

2.1.3 SW NML Implementation

To design the Smith-Waterman in NML technology, the layout of each main component was created by hand. Figures 2.7, 2.8, and 2.9 reports for example the NML layout of a multiplexer, a ripple carry adder and a 3-to-8 decoder. Layouts are created considering constraints on magnets placement derived by physical fabrication of clock wires. The circuits are modeled with VHDL and assembled creating the whole processing element. Its schematic is depicted in Figure 2.10.

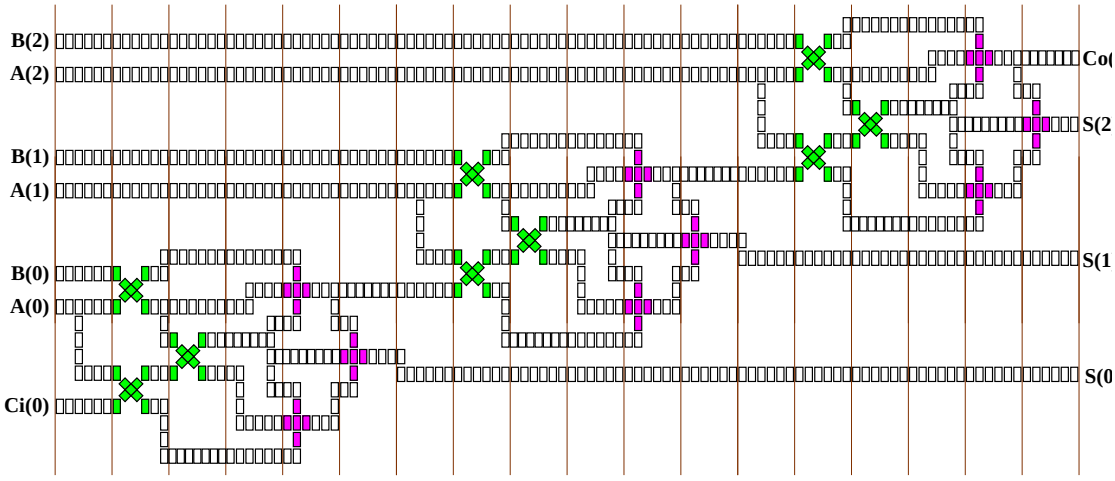


Figure 2.7. NML Multiplexer implementation.

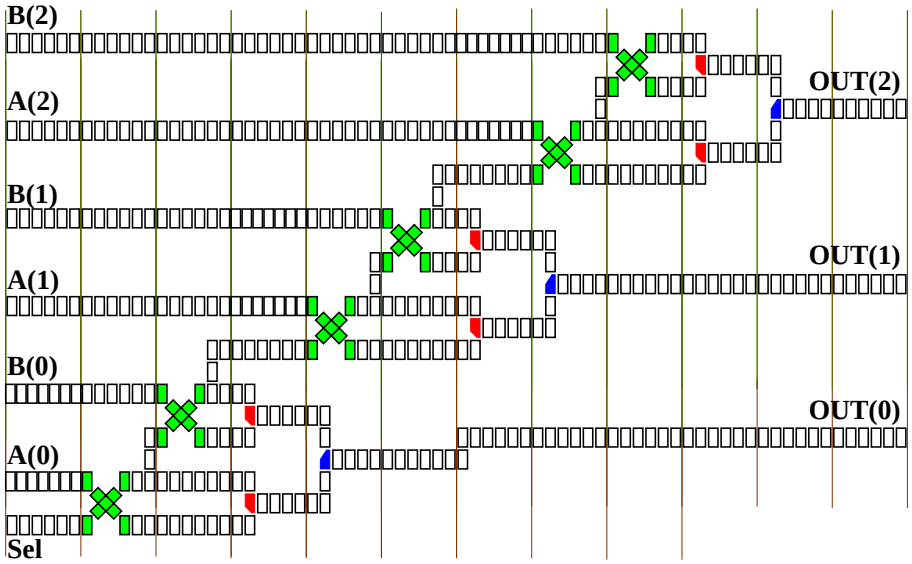


Figure 2.8. NML Ripple Carry Adder implementation.

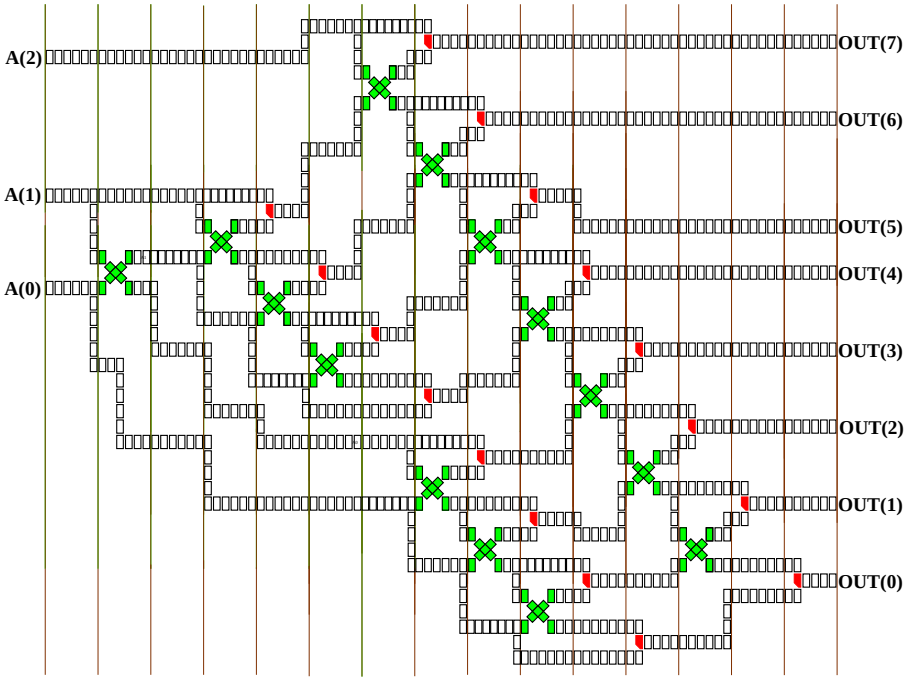


Figure 2.9. NML 3-to-8 Decoder implementation.

Looking at Figure 2.10, it can be noticed that there are two loops (Loop1 and

Loop2) in the structure. Given the intrinsic pipelined nature of NML technology, the presence of loop has a relevant impact on circuits performance. The length of Loop1 is longer than Loop2, and Loop1 includes two groups of signals, one is a 1-bit control signal and the other one is a multiple-bit feedback signal.

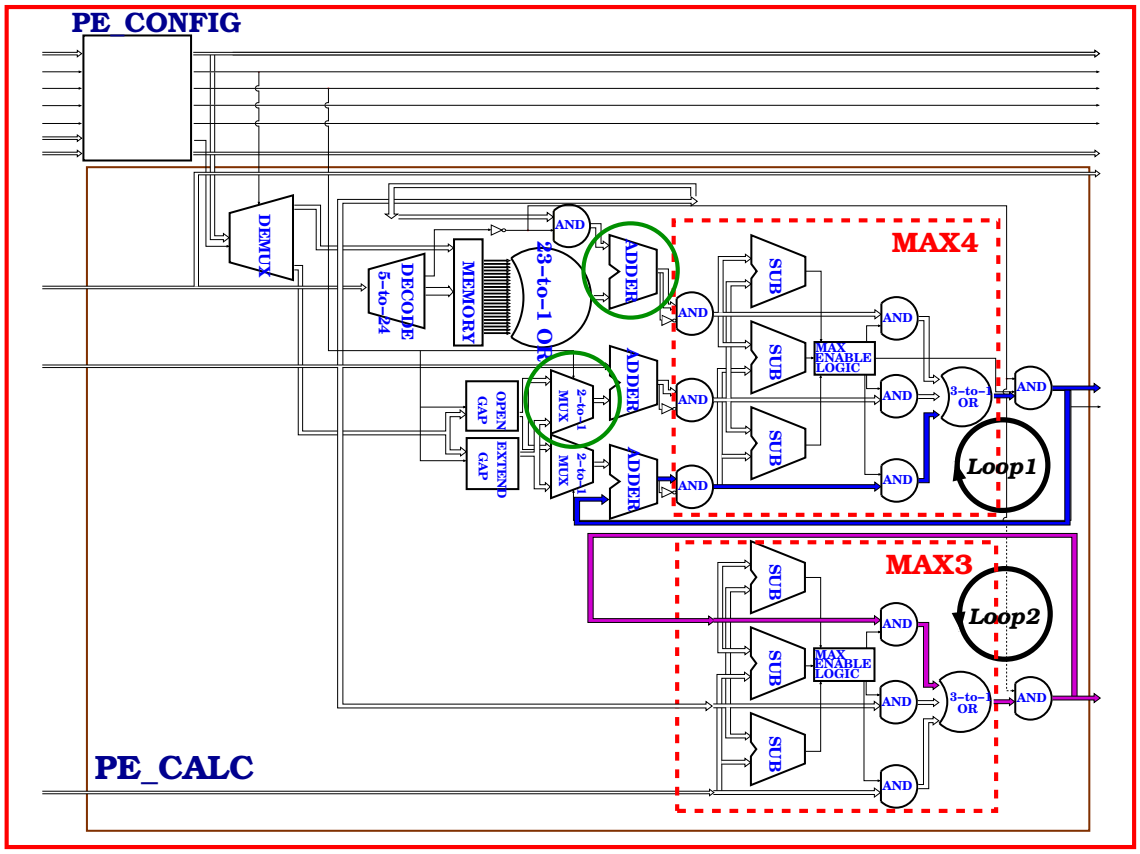


Figure 2.10. NML processing element implementation. There are two loops present in the architecture, Loop1 and Loop2. The longer loop (Loop1) occupies 208x3 clock zones.*M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

To simulate and evaluate the circuit, the Smith-Waterman algorithm was implemented with 8 processing elements, using a parallelism of 9 bit. Given the presence of a loop with a delay of 208 clock cycles, a new AA is fed to the circuit every 208 clock cycles, making the circuit latency equal to 208 clock cycles (Figure 2.11). Since every Subject sequence contains N AAs, to find the maximum alignment score for a particular sequence, $208*N$ clock cycles is the required time. This means about 1.8 ms with an average clock frequency of 100 MHz and a value of N equal to 969

[9]. In this test case, the Subject sequences used for the test were made by the same number of AAs, but in general, every sequence can have a different length. The longer the sequence is, the longer is the time required for the analysis to be completed. Figure 2.11 depicts the simulation waveforms. The *Subject_ID* identifies the biosequence number, while *OUT_MAX* is the calculated alignment score. As can be seen sequence 14 has the highest alignment score among the first 16 sequences.

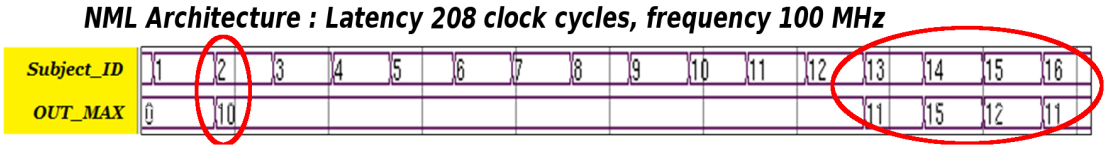


Figure 2.11. Smith-Waterman algorithm architecture simulation results. *Subject_ID* represents the number of the AA sequences analyzed, and *OUT_MAX* is the corresponding maximum alignment score. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

2.2 Performance Optimization

Having implemented a complex and realistic architecture, some architectural-level problems about circuit performance and synchronization were put in evidence. For each problem, a solution is proposed here. Performance optimization techniques are also evaluated.

2.2.1 Data Interleaving

As described previously, the long delay between two subsequent inputs is due to the NML intrinsic pipelined nature which leads to long propagation delay on the feedback path. As demonstrated in Figure 2.12, the new input needs to wait for the previous result to propagate back, if a loop is present. As a consequence, the total latency in terms of clock cycles is determined by the number of registers in the loop.

$$\text{Throughput of SW NML circuit} = \frac{1}{\text{loop length}} = \frac{1}{208 \text{ clock cycles}} \tag{2.1}$$

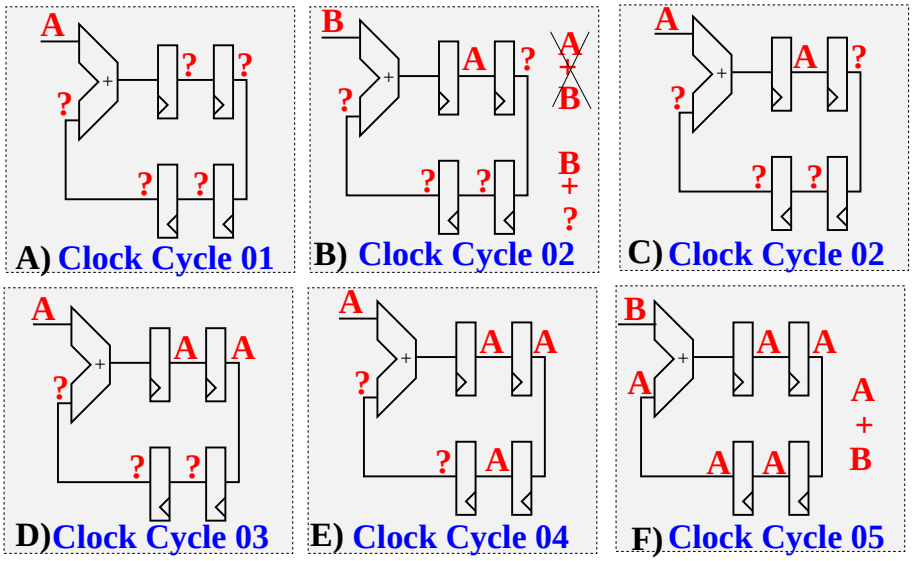


Figure 2.12. Example of NML architecture performance reduction due to the presence of loops inside intrinsically pipelined circuits. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

To maximize performance the technique of **Data Interleaving** can be exploited. This is a technique commonly adopted in standard technology, such as in microprocessors. It can be employed when multiple instructions must be executed in parallel and there is no data dependency among instructions. Interleaving principle is demonstrated in Figure 2.13. Four operations are executed in parallel. At each clock cycle, a new datum of a different operation is sent to the circuit input. Using this technique it is possible to sent a datum theoretically every clock cycle, maximiziing therefore performance.

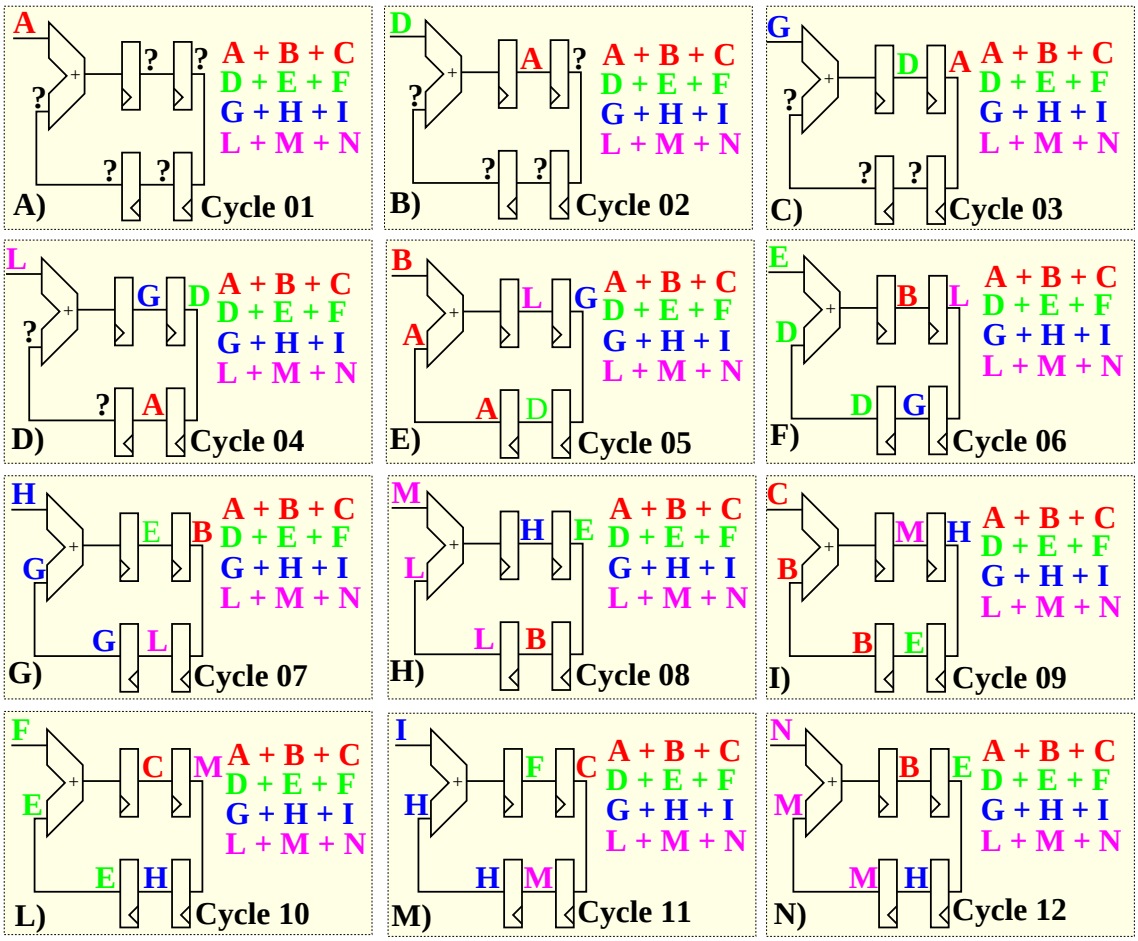


Figure 2.13. Data Interleaving application example. Four operations are executed in parallel to maximize the circuit throughput. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

The improved throughput is given by the equation:

$$\text{Improved Throughput} = \frac{\text{No.operations in parallel}}{\text{circuit latency}} \tag{2.2}$$

Executing a number of operations that equal to the loop length in parallel allows to maximize throughput. Figure 2.14 shows a complete simulation of the Smith-Waterman benchmark using a level of interleaving equal to 3, which analyses 3 subject sequences in parallel. The delay between two AAs of the same subject sequence is 208 clock cyles as before, which equals to about $1.8us$ (clock frequency about $110MHz$). After having sent the $AA(i)$ of Sequence 1 as input, it has to wait for 70 cycles to sent the $AA(i)$ of Sequence 2, and other 70 cycles to sent the $AA(i)$

of Sequence 3 and to switch back to Sequence 1 ($AA(i+1)$) after other 68 cycles. Figure 2.14 depicts the SW simulation with an interleaving equal to 3.

$$\text{Improved Throughput} = \frac{\text{No.operations in parallel}}{\text{circuit latency}} = \frac{3}{208 \text{ clock cycles}} \tag{2.3}$$

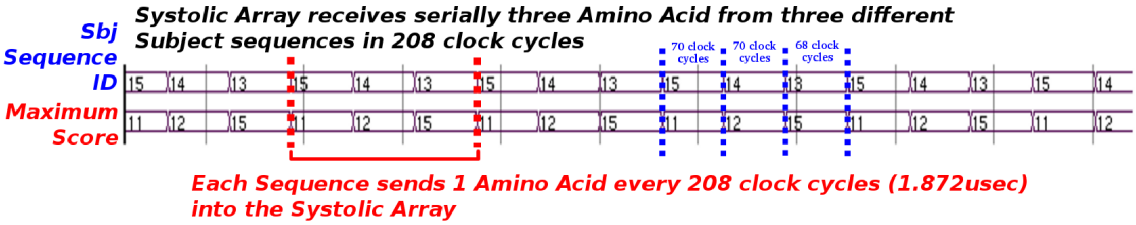


Figure 2.14. Simulation result of SW NML architecture with interleaving 3. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

To maximize the circuit performance, 208 sequences must be analyzed in parallel, reaching a throughput equal to 1. 208 sequences of data should be available for 208 operations to run in parallel, which is not a problem in this application, since it is designed to analyze a massive number of biosequences.
As it can be noticed, interleaving is a necessary optimization technique for NML circuits in presence of loops. However, due to the extremely high level of pipelining, a huge amount of data has to be provided to obtain the maximum throughput, which makes not all applications good candidates for this technology [12].

2.2.2 Architecture Redesign for Loops Lengths Reduction

Given the impact of loop, a simple solution is to reduce the loops lengths, since they are the "bottleneck" of the circuit throughput. The feedback path is as long as the path through the computational blocks.

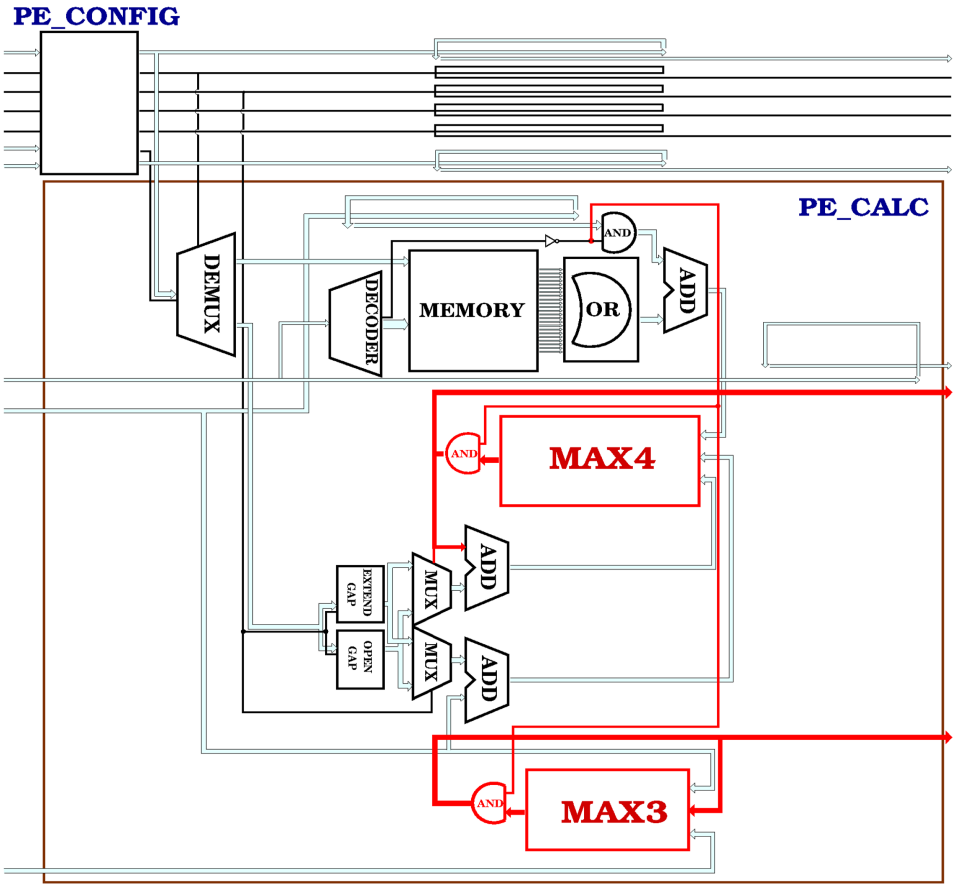


Figure 2.15. PE Architecture redesign to reduce loops lengths. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

The idea is to bend back the loop and changing the chain of blocks into a U-shaped structure (Figure 2.15). The loop length is reduced to 141 clock cycles, so the throughput is increased to 1/141 clock cycles.
The simulation result is shown in Figure 2.16 in comparison with the previous unmodified architecture. As can be noticed, the analysis of 14 sequences takes about $16ms$ while it takes about $24ms$ without optimization.

By reducing the delay to 141 clock cycles, using at the same time the technique of data interleaving, only 141 sequences of inputs are required to maximize the throughput. These combined methods expand the field of applications suitable for NML technology implementation.

It can be noticed that the whole loop length is not reduced to minimum at this point, so additional structure optimization can be done for further performance improvement.

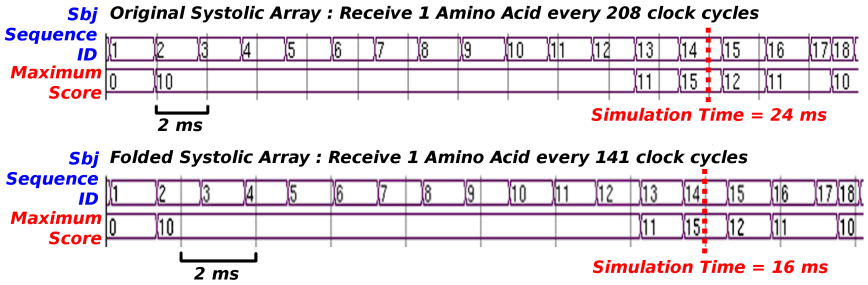


Figure 2.16. Simulation results comparison between redesigned PE architecture with folded loop and the original. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

2.3 Signal Synchronization

In a complex NML circuit, signal synchronization is an important issue, especially when loops are present. It requires careful design of circuits layout to correctly synchronize signals. The NML implementation of Smith-Waterman algorithm here highlight two major problems in signal synchronization related to loops: 1) nested loops and 2) additional loops.

2.3.1 Nested Loops

There are two signal loops that are nested in Loop1. To synchronize signlas, these two nested loops must have the same length. The simplified representation of this situation is shown in Figure 2.17 A), while B) demonstrates how to expand signals path in order to get two loops of equal length. Simulation results are reported in Figure 2.17 C). Without loop length equalization, results are wrong. With loops length equalization, results are instead correct. They are similar to the one reported in Figure 2.16.

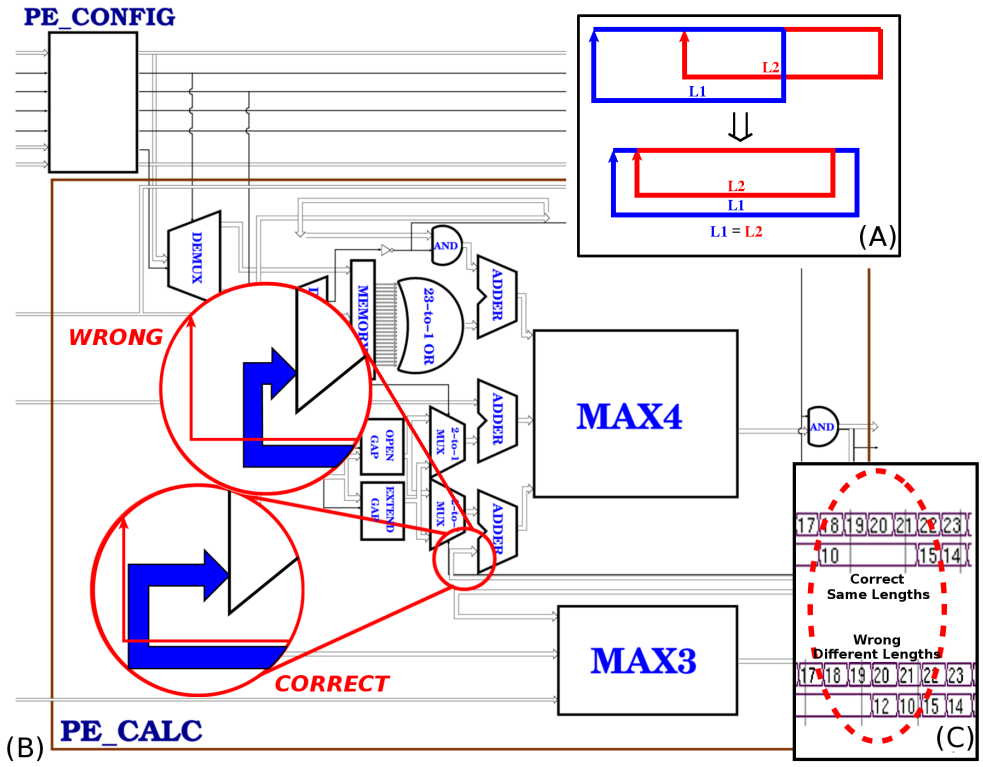


Figure 2.17. PE signal synchronization with nested loops. A) Simplified schematic presentation of nested loops. B) Highlight of nested loops in Smith-Waterman processing element NML architecture. C) Simulation comparison with correct and wrong signal synchronizations. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

2.3.2 Additional Delay Loops

A common technique normally employed in CMOS technology is to add additional registers to delay certain signals for synchronization purpose, for example the skewing and deskewing networks. In the SW processing element, the algorithm requires the block *MAX4* to compute the local alignment score using the results obtained from the previous PE one and two cycles before and its own result one cycle before. The circuit can be simplified as Figure 2.18 A). The "calculation cycle" (also a clock cycle in CMOS) corresponds to the time between sending two amino acids consecutively. In NML technology, this cycle becomes the length of the circuit loop in terms of clock cycles as described above.
The delay registers that are present on both paths can be removed, leaving the additional register on the upper path. The registor is mapped to a loop with a length

of 208 clock cycles, given that a new AA is fed to the inputs every 208 clock cycles. With this technique, it is therefore possible to delay and correctly synchronize signals path.

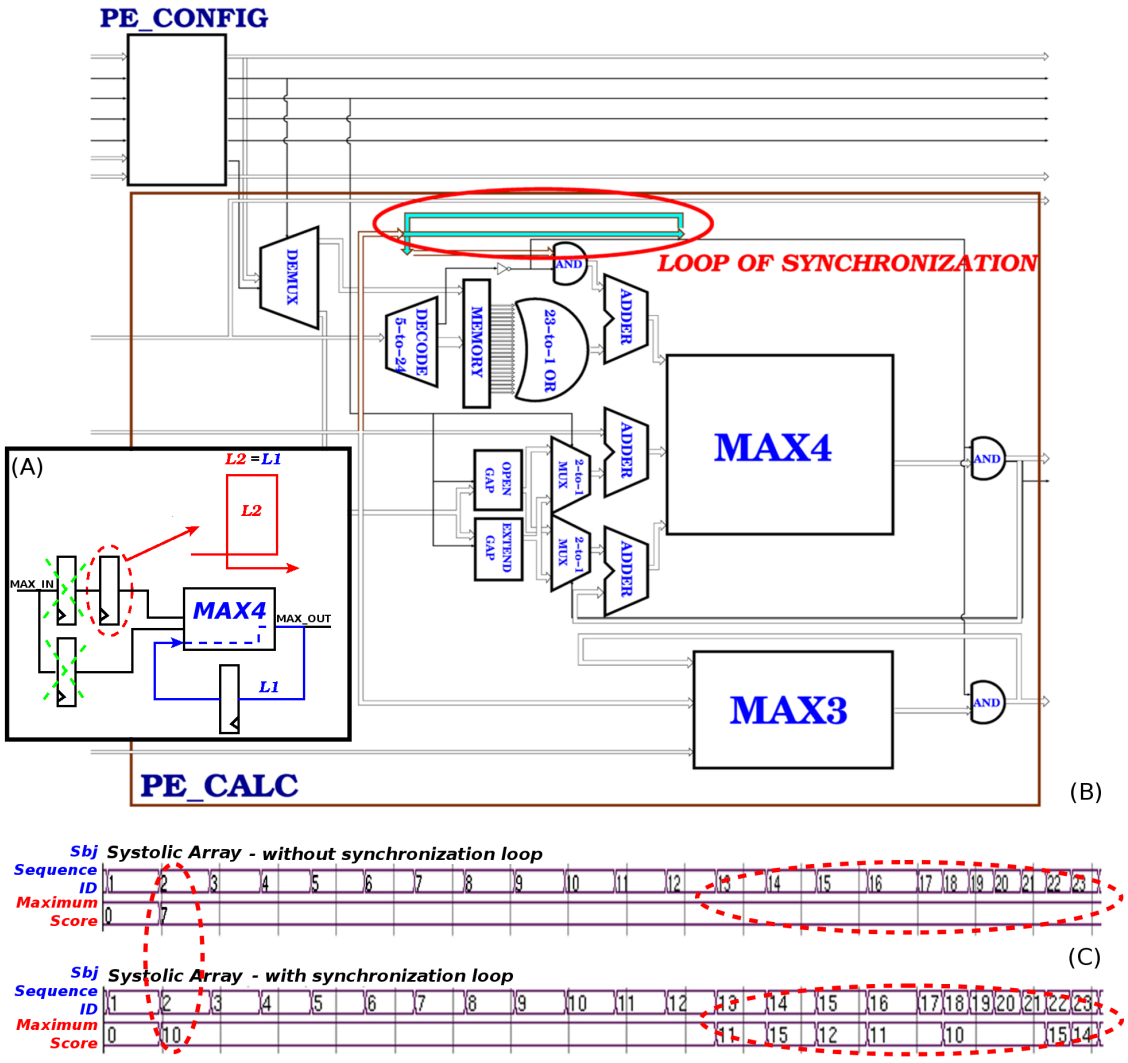


Figure 2.18. PE signal synchronization with addtional delay loops. *M. Vacca et al. "Feedbacks in QCA: A Quantitative Approach", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, Oct. 2015.*

The investigation on the length of this additional loop (**L1**) demonstrates that for signal synchronization purpose, it should be the same as the total length of the feedback loop (**L2**). As shown in Figure 2.18 C), without the additional loop,

computational errors will occur since the signals are not synchronized.

2.4 NML Architecture Area and Power Evaluation

In order to compare the performance of the NML implementation with CMOS technology, a 5 elements systolic array has been synthesized using a standard CMOS technology [20]. It is worth underlining that in CMOS the clock frequency is around 370 MHz and the latency for on PE is 1 clock cycle. The NML implementation uses a 100 MHz clock frequency, so also employing interleaving, it is therefore 4 times slower than CMOS implementation. However, the lack of speed is compensated by a much smaller power consumption.
Using the power models described in Section 1.2.6, the area and power consumption estimations of a single processing element, are compared between CMOS and the NML implementations. For NML data are accurately estimated starting from the circuit layout and technological data. Since most of the Smith-Waterman main blocks are accurately designed by hand, it is possible to know exactly their area and their composition. The total circuit area and the number of magnets of the processing element can be estimated starting from the main blocks and using multiplicative constants to keep into account interconnections overhead. Once the total circuit area and the total number of magnets is known, it is possible to estimate the circuit power consumption. The total estimated number of magnets for a processing element is 470000. Therefore, as shown in Table 2.1 [16], the area of the NML implementations (21000 um^2 in Magnetic Field NML and 20000 um^2 in STT-current NML) is much bigger than CMOS case, which is around 1000 um^2. The explanation is that in NML, according to the current level of technology maturity, circuits are built using only one layer physical, while there are multiple layers for interconnections in CMOS technology.

Table 2.1. Power consumption and area estimation for a single processing element of the systolic array with main NML implementations and CMOS LOP 21nm technology. *J. Wang et al. "Biosequences analysis on NanoMagnet Logic", International Conference on IC Design and Technology (ICICDT), May 2013.*

Implementation	Area $[um^2]$	Power $[mW]$
Magnetic Field NML	21000	2
STT-current NML	20000	131
Magnetoelastic NML	12000	0.01
CMOS LOP 21nm	1000	0.72

Regarding power consumption comparisons, neither magnetic field based NML nor STT-current based NML is suited for low power application [21]. Using instead the magnetoelastic clock solution, it is possible to obtain a considerable reduction in power over CMOS, since the total power consumption is around 0.01 mW. Clearly this is the best solution for NML logic which allows to obtain a remarkable reduction of power consumption with only a relatively limited reduction of speed.

Chapter 3

Introduction to Nanoscale Application Specific Integrated Circuits

The field of semiconductor nanowire (NWs) has become one of the most active research areas within the nanoscience community [22].
With a few decades of research and development activities, semiconductor nanowires can be now fabricated in high-yield with reproducible electronic properties as required for large-scale integrated systems. Moreover, with bottom-up synthesis approach, the body thickness/diameter of nanowires can be well-controlled down to below 10 nm. This represents a significant advantage over CMOS technology, where it has become increasingly difficult to achieve maintaining the electrical integrity with the aggressive scaling of gate length [23].

3.1 Nanowire Field Effect Transistors (NW FETs)

There is a large range of nanoscale devices that employ semiconductor nanowires, such as Field-effect transistors (FETs), p-n diodes, ultraviolet (UV) detectors, single nanowire solar cells, chemical sensors, biosensors and nanogenerators [24]. As shown in Figure 3.1, logic AND, OR and NOR gates have been fabricated by Y. Huang *et al.* in [25], among which 1 (p-Si) by 3 (n-GaN) crossed NW FETs (xnwFETs) are used.

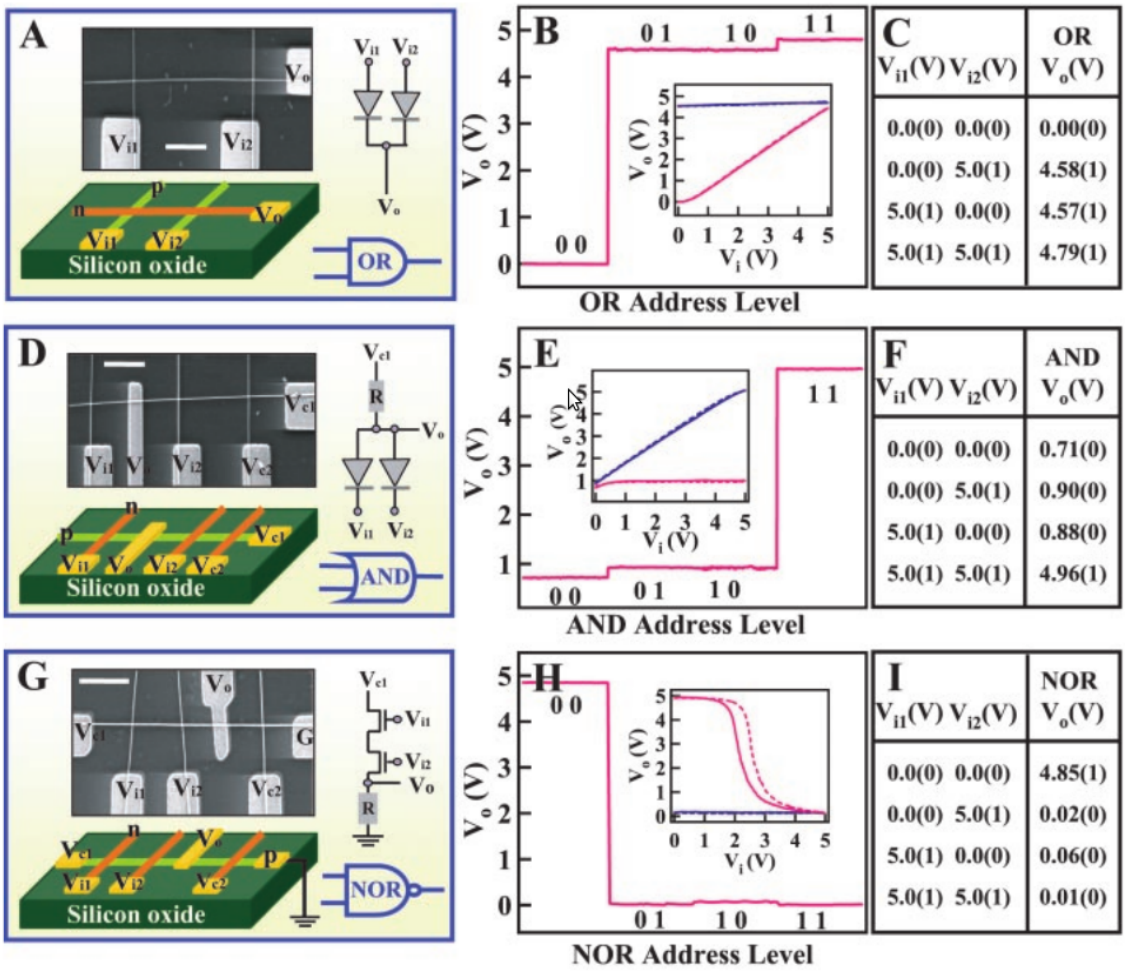


Figure 3.1. Nanowire nano-logic gates. A) Schematics of logic OR gate constructed from a 2 by 1 crossed NW p-n junction. B) OR gate input and output voltage levels. C) The experimental truth table for the OR gate. D) Schematic of logic AND gate constructed from a 1 by 3 crossed NW junction array. E) AND gate input and output voltage levels. F) The experimental truth table for the AND gate. G) Schematic of logic NOR gate constructed from a 1 by 3 crossed NW junction array. H) NOR gate input and output voltage levels. I) The experimental truth table for the NOR gate. *Y. Huang et al. "Logic Gates and Computation from Assembled Nanowire Building Blocks", Science, vol. 294, Nov. 2001.*

Different gate structures are available in literature to implement the nanowire FETs as shown in Figure 3.3 [23], from semicylindrical top gates to gate-all-around configurations. Several gate materials are also in progress of evaluations as presented in Figure 3.2 [26].

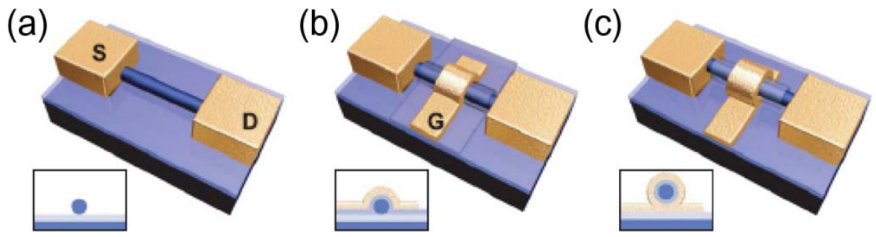


Figure 3.2. Schematic of nwFETs with A) back gate, B) semicylindrical top gate, and C) cylindrical gate-all-around configurations. *W. Lu et al. "Nanowire Transistor Performance Limits and Applications", IEEE Transactions on Electron Devices, vol. 55, Nov. 2008.*

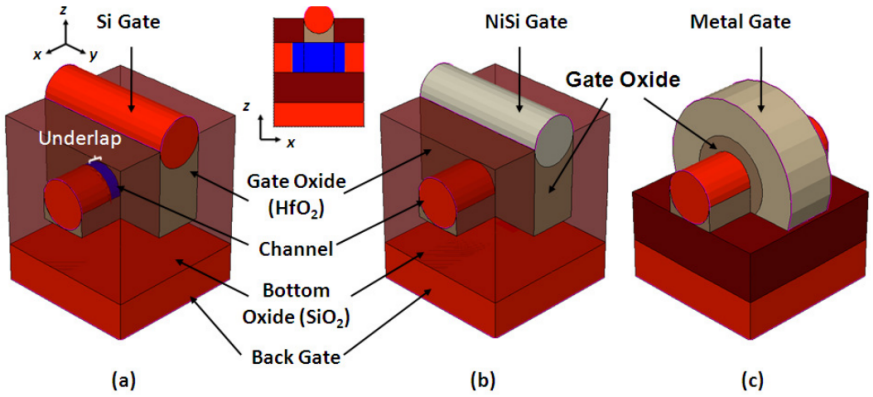


Figure 3.3. Crossed Nanowire Field Effect Transistors. A) Basic device structure with self-aligned n+ drain, gate, source and underlap. B) NiSi gate xnwFET. C) Omega-gated xnwFET. *P. Narayanan et al. "Nanoscale Application Specific Integrated Circuits", IEEE/ACM International Symposium on Nanoscale Architectures, June 2011.*

Using different levels of doping in the nanowires channel and gate [27], multiple input AND and OR gates can be built with xnwFETs as in Figure 3.4. By combining a plane of horizontal nanowires and a plane of vertical nanowires, it comes a 2-D nanoarray PLA-like structure (Figure 3.5 A)). As stated in [28], this PLA-like organization contains a primary horizontal NOR logic plane that generates the

products of inputs, and a secondary NOR logic plane (including the vertical buffering nanoarrays) that computes the sum of the products. An optimized structure has been proposed (Figure 3.5 B)). This type of logic block is the fundamental logic unit (called "Nanotile") of Nanoscale Application Specific Integrated Circuits.

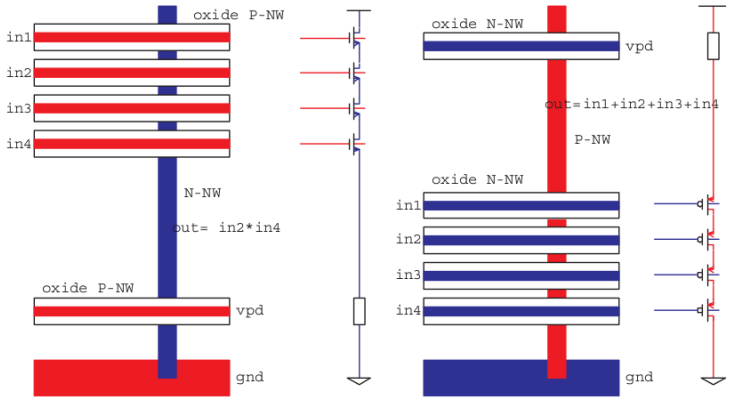


Figure 3.4. 4-input OR and AND logic implemented with xnwFETs. *T. Wang et al. "NASICs: A Nanoscale Fabric for Nanoscale Microprocessors", Electrical and Computer Engineering Department, University of Massachusetts Amherst, USA.*

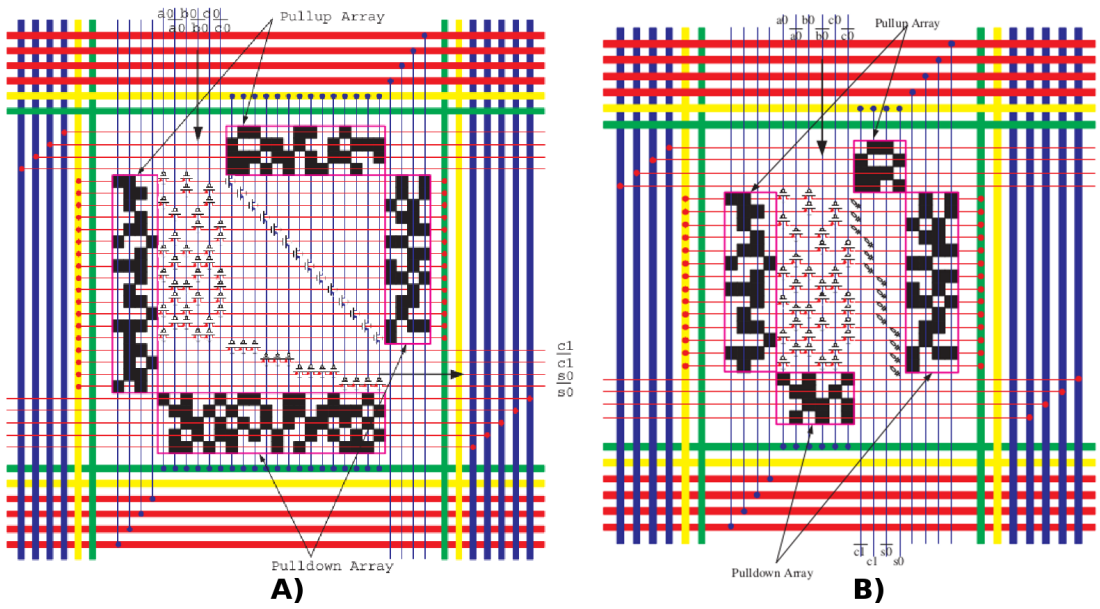


Figure 3.5. 1-bit Full-Adder with 2-D nanoarray structure. *T. Wang et al. "Opportunities and challenges in application-tuned circuits and architectures based on nanodevices", First ACM International Conference On Computing Frontiers, pp. 503-511, april 2004.*

3.2 Nanoscale Application Specific Integrated Circuits (NASICs)

Nanoscale Application Specific Integrated Circuits (NASICs) is an emerging technology which uses silicon nanowire FETs (SiNW FETs) organized in a PLA-like structure. It allows to design high density circuits coupled with high clock frequencies thanks to the small size that silicon nanowires can reach.

In literature, simple circuits like logic gates (Figure 3.1) and complex architectures like microprocessors (i.e. WISP-0) were designed (Figure 3.6) in many of these works. However, technological constraints were not taken into account when evaluating circuits performance. Moreover, medium complexity circuits with detailed layout analysis are not analyzed in literature. A new methodology to design and simulate complex NASIC circuits with careful area and power estimation is necessary to obtain a thorough evaluation of this technology.

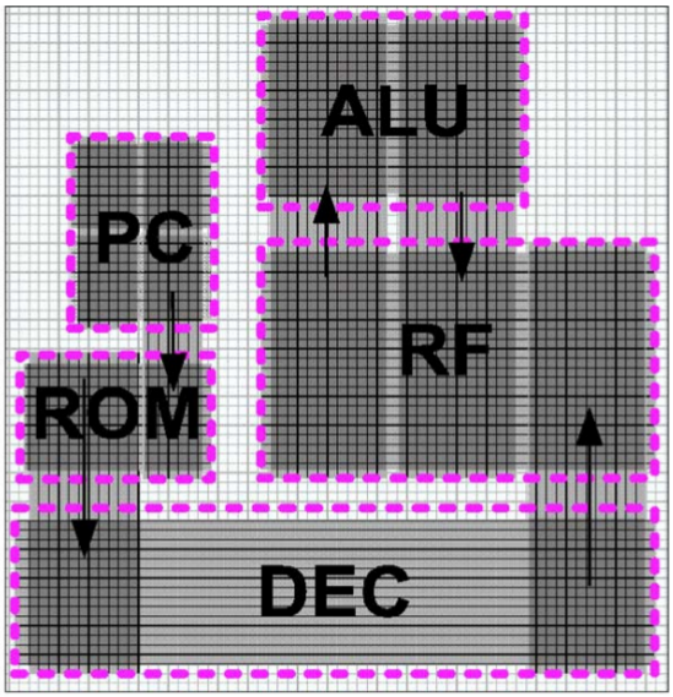


Figure 3.6. Floorplan of the WISP-0 processor. WISP-0 is a five-stage pipelined streaming architecture in five nanotiles: PC, ROM, DEC, RF and ALU. *T. Wang et al. "Heterogeneous Two-Level Logic and Its Density and Fault Tolerance Implications in Nanoscale Fabrics", IEEE Transactions on Nanotechnology, vol. 8, n. 1, Jan. 2009.*

3.2.1 Nanotile

The basic logic block of NASICs is a 2-D nanoarray structure called "Nanotile" (Figure 3.7) fabricated with silicon nanowires, where the intersections forms nanoscale Field Effect Transistors (nanoFETs). Controlling the doping in the gate and in the channel, nanoFETs can be configured as p-type or n-type transistors. In a traditional nanotile combining AND-OR (Figure 3.8 A)) planes, both types of nanoFETs are required. However, because of large differences in transport properties between p-FET and n-FET, and the difficult in building both types of transistors with the same material, it is much more advantageous to employ a single type of FETs in NASIC circuits [29] [30]. This is possible using single n-type FET Nanotile with NAND-NAND planes (Figure 3.8 B)).

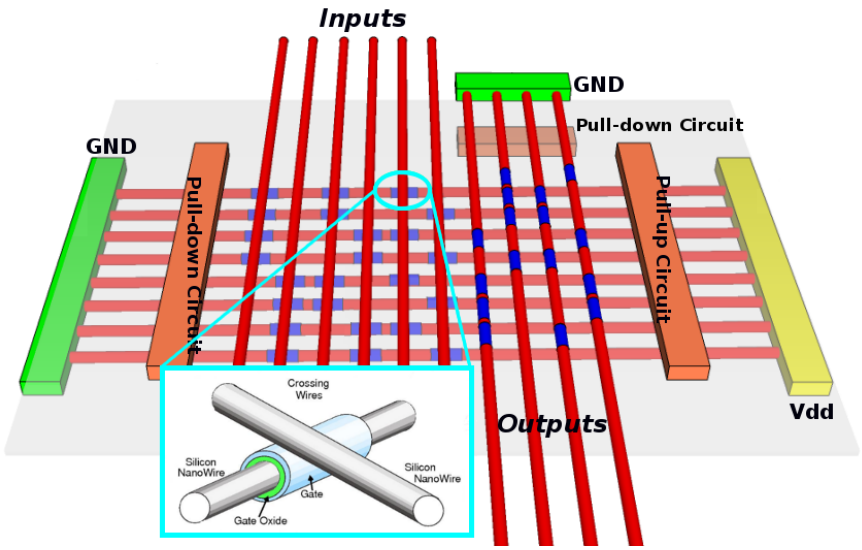


Figure 3.7. Nanotile structure built with silicon nanowires and nanoFETs. *P. Narayanan et al. "Nanoscale Application Specific Integrated Circuits", IEEE/ACM International Symposium on Nanoscale Architectures, June 2011. (Inserted) M. Graziano et al. " A Hardware Viewpoint on Biosequence Analysis: What's Next?", ACM Journal on Emerging Technologies in Computing Systems, Nov. 2013.*

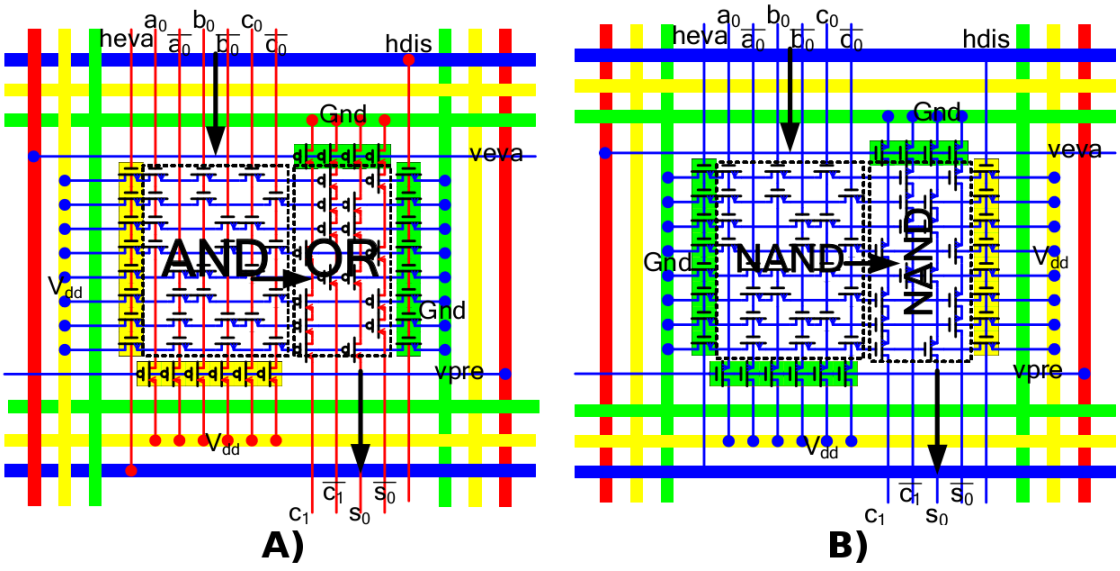


Figure 3.8. Schematic representations of 1-bit Full Adder with A) mixed types of nanoFETs, and B) single n-type nanoFETs. *T. Wang et al. "NASICs: A Nanoscale Fabric for Nanoscale Microprocessors", Electrical and Computer Engineering Department, University of Massachusetts Amherst, USA.*

3.2.2 NASIC Clock Mechanism

Since it is widely used in CMOS technology designs, **Dynamic Logic** can be used also for NASIC circuits (Figure 3.9).

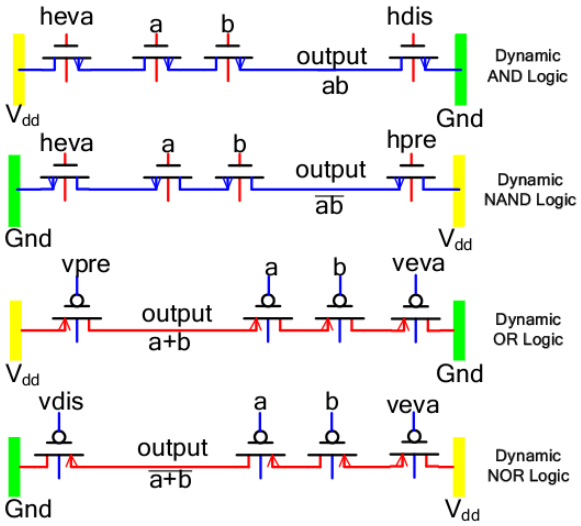


Figure 3.9. Dynamic circuits implementing AND, NAND, OR, and NOR logic functions on NWs. *T. Wang et al. "Heterogeneous Two-Level Logic and Its Density and Fault Tolerance Implications in Nanoscale Fabrics", IEEE Transactions on Nanotechnology, vol. 8, n. 1, Jan. 2009.*

By introducing precharge and evaluation control signals on both horizontal and vertical nanowires in the nanotiles (Figure 3.10), the combinational logic on the two planes (AND-OR or NAND-NAND) can be controlled in a synchronous way, hence a multiphase *Clock* mechanism. The clocked operations in a nanotile is done by driving the horizontal and vertical precharge/evaluation control signals in a clock cycle divided into 4 phases (Figure 3.10).

- Phase I: Precharge of horizontal wires (Hpre active). The horizontal wires are charged to high no matter what input signals are.

- Phase II: Evaluation of horizontal wires (Heva active). The horizontal wires switch states based on input signals. So the output signals of horizontal plane or the input signals of vertical plane are evaluated.

- Phase III: Precharge of vertical wires (Vpre active). The vertical wires are charged to high. This phase can occur contemporarily with Phase II.

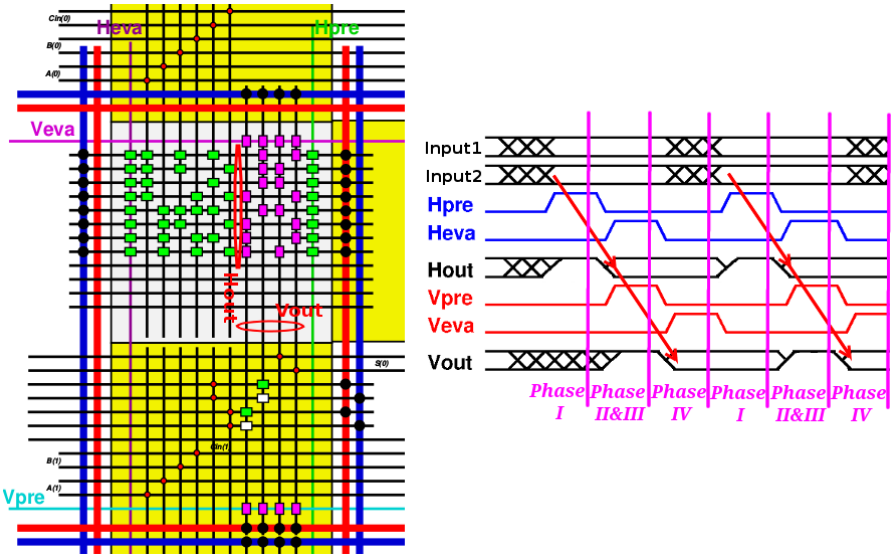


Figure 3.10. Nanotile clock mechanism with dynamic logic. The rectangular green boxes on nanowire cross sections are n-type nanoFETs while the white ones are p-type nanoFETs .

- Phase IV: Evaluation of vertical wires (Veva active). The output signals of the nanotile are ready.

Therefore, every nanotile, no matter what the implementing logic is, has a latency of one clock cycle. This fact is very similar as the "Layout=Timing" characteristic of the previously discussed NanoMagnet Logic, implying also in this case an intrinsic pipelined behavior.

3.2.3 2-bit Full Adder

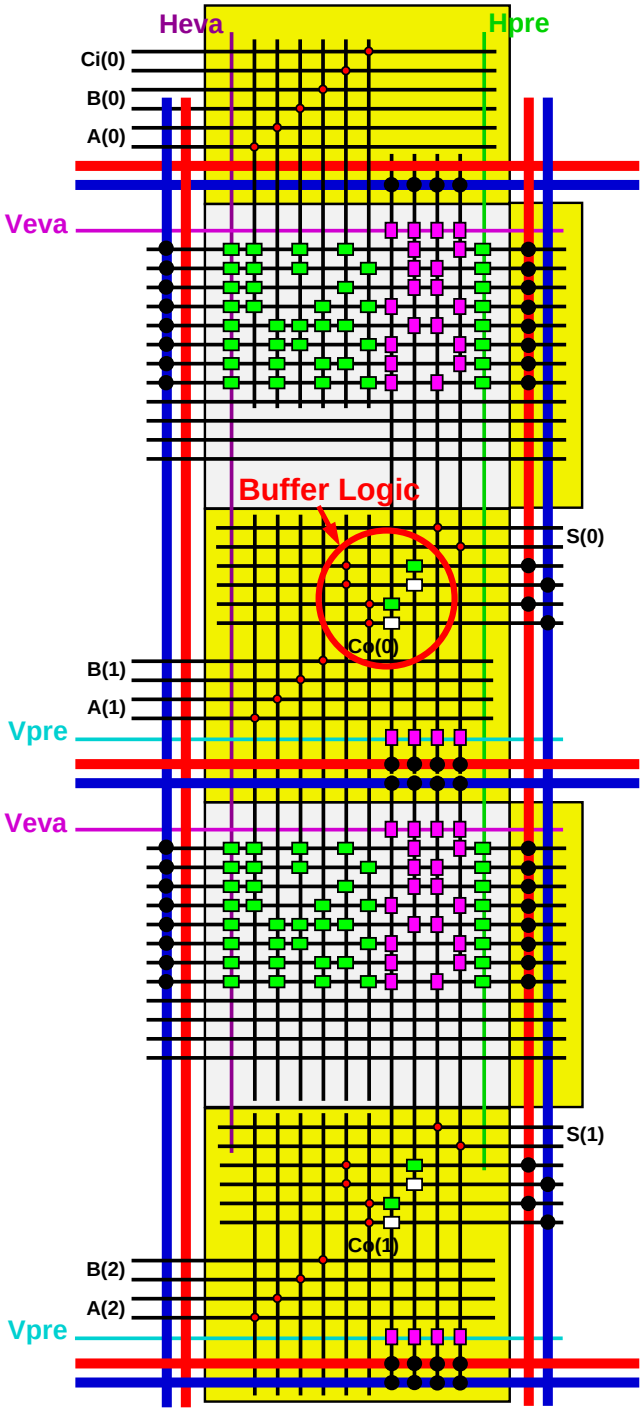


Figure 3.11. Nanotile behavior modeling in VHDL.

As proposed in [31], by putting in cascade the nanotiles, multiple bit logic blocks can be contructed, for example, a 2-bit Full Adder as shown in Figure 3.11.

A buffer tile is required in the nanotile interconnection zone. The buffer is used to rout signals, and to regenerate and enhance logic signals on the nanowires.

3.2.4 N3ASIC

Similarly to NML, NASIC technology leads to single layer circuits, which causes large amounts of area overhead and circuit latency increase. In order to overcome this drawback, a 3-D nanofabric called **N^{3}ASIC** is presented in [32]. With the Omega metal gate NanoFETs (Figure 3.12A)), a N3ASIC nanotile (Figure 3.12B)) can be fabricated with current manufacturing process as shown in Figure 3.13. Using a Silicon-on-Insulator (SOI) wafera as a base, uniform semiconductor nanowire arrays are grown. Then metal gates are deposited at logic (NAND-NAND) planes to define the cross points to define nanowire FETs. At the next step, the lower layer of metal 1 interconnects are deposited, which play the same role as vertical nanowires in original NASIC nanotiles. Therefore, the vias on Logic plane 1 are the outputs of the horizontal NAND plane. With the second layer of metal 2, the two planes are connected.

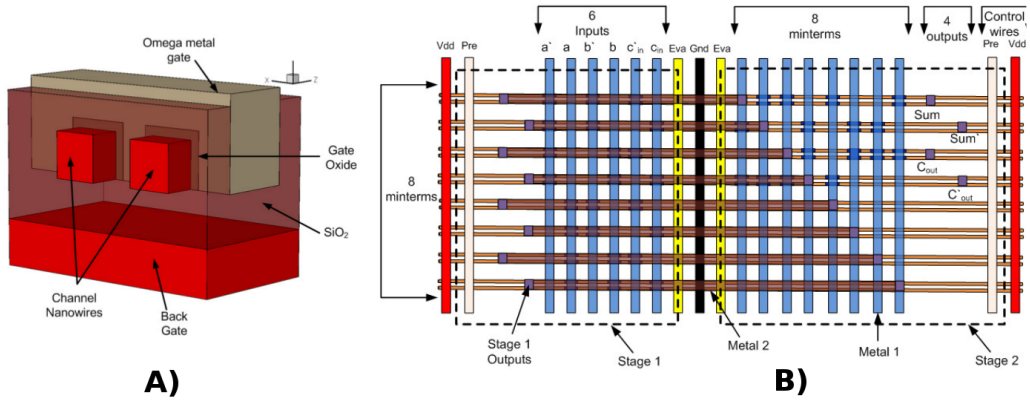


Figure 3.12. N3ASIC Nanotile structure. A) 3D structure of Omega metal gate NanoFET in N3ASIC. B) 1-bit Full Adder N3ASIC. *P. Panchapakeshan et al. "3-D Integration Requirements for Hybrid Nanoscale-CMOS Fabrics", IEEE International Conference on Nanotechnology, Aug. 2011.*

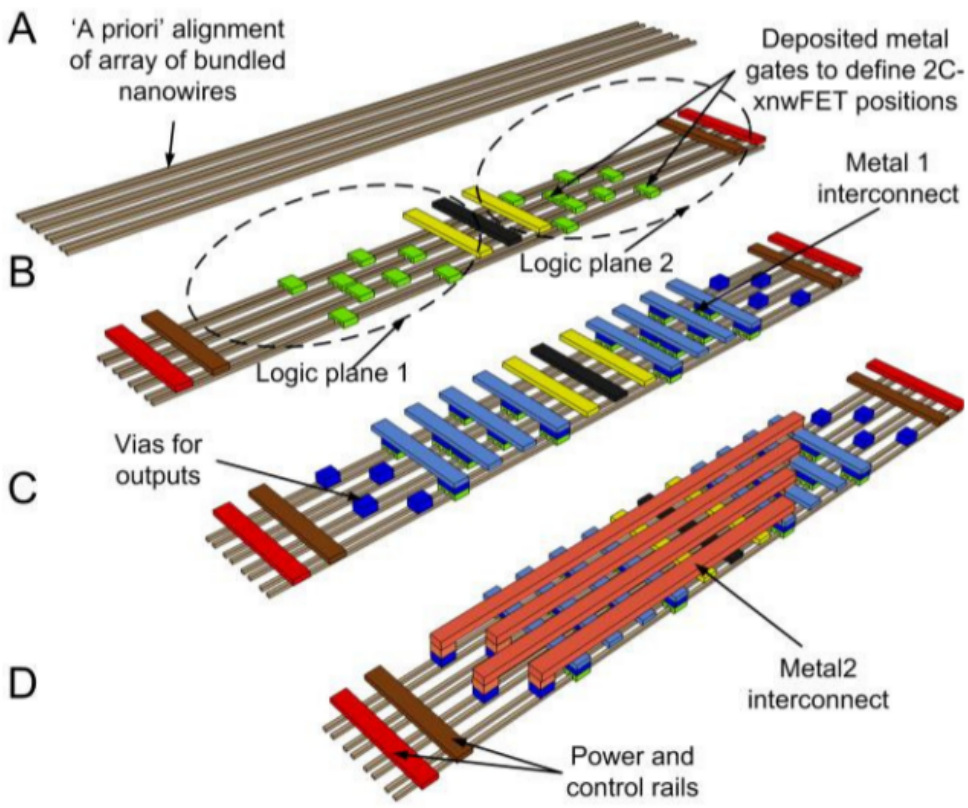


Figure 3.13. N3ASIC Nanotile structure. *P. Panchapakeshan et al. "3-D Integration Requirements for Hybrid Nanoscale-CMOS Fabrics", IEEE International Conference on Nanotechnology, Aug. 2011.*

N3ASIC has the advantage of providing a natural integration with CMOS as shown in Figure 3.14, but the operating mechanism in a N3ASIC nanotile is the same as the orignal NASIC. Therefore, the NASIC clock mechanism also applies here. From the logic implementation point of view, it is only necessary to rotate the vertical NAND plane of NASIC to horizontal direction for N3ASIC implementation. Hereby, the orignal NASIC nanotile structure is kept as the basic structure for analysis.

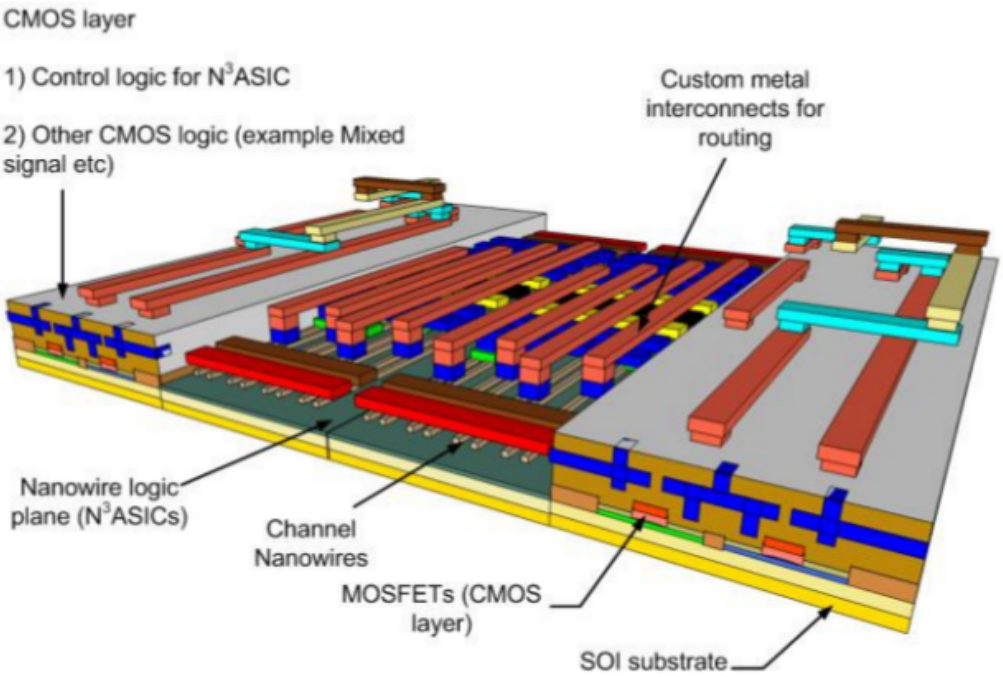


Figure 3.14. Hybric Nano-CMOS 3D integrated fabric structure. *P. Panchapakeshan et al. "N3ASICs: Designing Nanofabrics with Fine-Grained CMOS Integration", IEEE/ACM International Symposium on Nanoscale Architectures, 2011.*

3.2.5 NASIC VHDL Modeling

Similarly to NML technology, a behavioral model written in VHDL was developed. As can be noticed in Figure 3.10, the control signals introduces a "clock" mechanism into nanotile operations. The behavioral model of a nanotile in VHDL is shown as Figure 3.15, where horizontal and vertical wires are modelled as multiple input NAND gates with output registers modeling the delay. Among the VHDL codes, the numbers of horizontal and vertical wires, even the number of transistors on each wire can be extracted which would be useful in Section 4.1 to implement embedded area and power estimation inside the model.

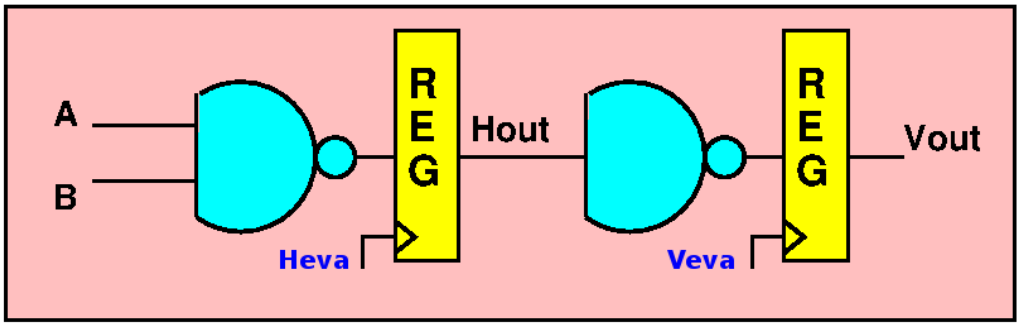


Figure 3.15. Nanotile behavior modeling in VHDL.

1-bit Full Adder Nanotile VHDL behavioral model

In the following, the VHDL code of a single tile is give. The logic function implemented is a 1-bit Full Adder.

```
-- 1 bit Full Adder
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use work.NASIC_package.all;

ENTITY FA_1 IS
        PORT(CLK        :       IN      STD_LOGIC_VECTOR (4 DOWNTO 1);
                A               :       IN      STD_LOGIC;
                nA              :       IN      STD_LOGIC;
                B               :       IN      STD_LOGIC;
                nB              :       IN      STD_LOGIC;
                Ci              :       IN      STD_LOGIC;
                nCi             :       IN      STD_LOGIC;
                Co              :       OUT     STD_LOGIC;
                nCo             :       OUT     STD_LOGIC;
                S               :       OUT     STD_LOGIC;
                nS              :       OUT     STD_LOGIC
                );
END FA_1;

ARCHITECTURE STR OF FA_1 IS

SIGNAL S1                       :       STD_LOGIC_VECTOR (5 DOWNTO 0);
SIGNAL S1_reg0          :       STD_LOGIC_VECTOR (5 DOWNTO 0);
SIGNAL Hout                     :       STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL Hout_reg0        :       STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL Hout_reg1        :       STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL Vout                     :       STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL Vout_reg0        :       STD_LOGIC_VECTOR (3 DOWNTO 0);

COMPONENT register_generic IS
GENERIC (       NBIT    :       INTEGER);
PORT(
        CLK     :       IN      STD_LOGIC;
```

```
        D :     IN      STD_LOGIC_VECTOR (NBIT-1 DOWNTO 0);
        Q :     OUT     STD_LOGIC_VECTOR (NBIT-1 DOWNTO 0));
END COMPONENT;

BEGIN

S1(0)<= A;
S1(1)<= nA;
S1(2)<= B;
S1(3)<= nB;
S1(4)<= Ci;
S1(5)<= nCi;

        --discharge the horizontal tiles
REG0_Hdis : register_generic GENERIC MAP (6) PORT MAP (CLK(1), S1, S1_reg0);

        --horizontal logic computation
Hout(0)<= not(S1_reg0(0) AND S1_reg0(3) AND S1_reg0(4));
Hout(1)<= not(S1_reg0(1) AND S1_reg0(2) AND S1_reg0(4));
Hout(2)<= not(S1_reg0(0) AND S1_reg0(2) AND S1_reg0(4));
Hout(3)<= not(S1_reg0(0) AND S1_reg0(2) AND S1_reg0(5));
Hout(4)<= not(S1_reg0(0) AND S1_reg0(3) AND S1_reg0(5));
Hout(5)<= not(S1_reg0(1) AND S1_reg0(2) AND S1_reg0(5));
Hout(6)<= not(S1_reg0(1) AND S1_reg0(3) AND S1_reg0(4));
Hout(7)<= not(S1_reg0(1) AND S1_reg0(3) AND S1_reg0(5));

        -- clock the horizontal output signal
REG0_Hout : register_generic GENERIC MAP (8) PORT MAP (CLK(2), Hout, Hout_reg0);

        --precharge the vertical tiles
REG0_Vpre : register_generic GENERIC MAP (8) PORT MAP (CLK(3), Hout_reg0, Hout_reg1);

        -- vertical logic computation
Vout(0)<= not(Hout_reg1(0) AND Hout_reg1(1) AND Hout_reg1(2) AND Hout_reg1(3));
Vout(1)<= not(Hout_reg1(2) AND Hout_reg1(4) AND Hout_reg1(5) AND Hout_reg1(6));
Vout(2)<= not(Hout_reg1(4) AND Hout_reg1(5) AND Hout_reg1(6) AND Hout_reg1(7));
Vout(3)<= not(Hout_reg1(0) AND Hout_reg1(1) AND Hout_reg1(3) AND Hout_reg1(7));

        -- clock horizontal output signal
REG0_Vout : register_generic GENERIC MAP (4) PORT MAP (CLK(4), Vout, Vout_reg0);

Co <= Vout_reg0(0);
S <= Vout_reg0(1);
nCo <= Vout_reg0(2);
nS <= Vout_reg0(3);


END STR;
```

Chapter 4

NASIC Circuit Modeling and Implementation

To analyze and understand the effectiveness of NASIC technology, it is necessary to design and to study medium to high complexity circuits. It is the goal of this part of my thesis work. First of all, an area and power estimation model is added to the VHDL code. Secondly, the enhanced model is used to design and analyze complex circuits to understand the strength and the weakness of NASIC technology.

4.1 Area and Power Evaluation

Similarly to what have done for NML technology, physical level information is embedded in the VHDL model. This kind of information, like silicon nanowire sizes, applied voltage, are used to calculate the area and the power consumption of a tile.

4.1.1 Area Evaluation

As depicted in Figure 4.1, the parameters used to evaluate the area are:

- Dpw: Distance between two power wires;
- Dnw: Distance between two nanowires;
- Wpw: Width of power supply wires;
- Wnw: Width of nanowires.

The Nanotile has a PLA-like structure with complementary logic. Two wires are used for each input and output signal.Without considering Karnaugh map simplification, the number of horizontal wires depend on the number of inputs.

$$\text{No.Horizontal Nanowires} = 2^{\text{No.Inputs}} \tag{4.1}$$

$$\text{No.Vertical Nanowires} = 2 * \text{No.Inputs} + 2 * \text{No.Outputs} \tag{4.2}$$

For example, in the nanotile of 1-bit Full Adder,

$$\begin{cases} \text{No.Inputs} = 3 \\ \text{No.Outputs} = 2 \end{cases} \rightarrow \begin{cases} \text{No.Horizontal Nanowires} = 2^3 = 8 \\ \text{No.Vertical Nanowires} = 2{*}3 + 2{*}2 = 10 \end{cases}$$

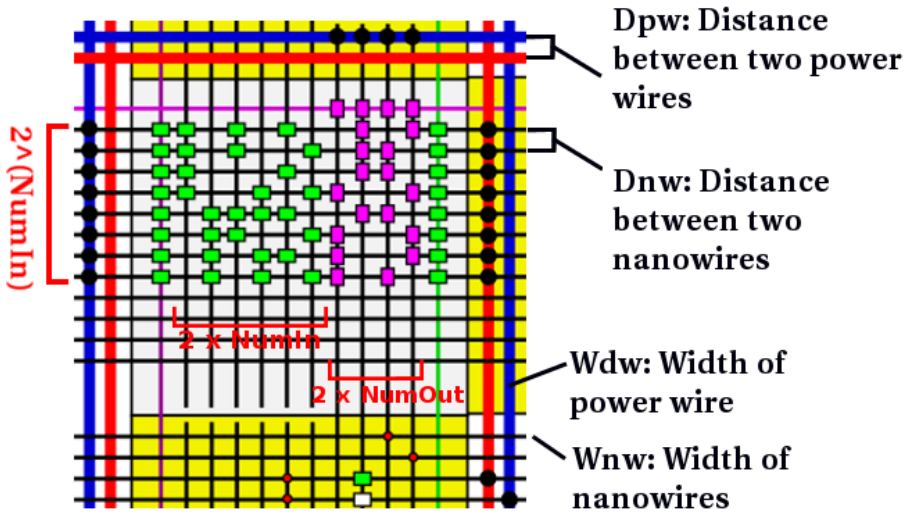


Figure 4.1. Nanotile area evaluation parameters.

Taking into account the precharge and evaluation control wires, the area of nanowire plane can be calculated as:

$$\begin{aligned} Area(Nanowire) = (2^{\text{No.Inputs}} + 2) * (2 * \text{No.Inputs} + 2 * \text{No.Outputs} + \\ 2) * (Wnw + Dnw)^2 \end{aligned} \tag{4.3}$$

The length of horizontal power wires is determined by the nanowire plane length, while the length of vertical power wires is related to the nanowire plane width. The area of power supply wires can therefore be calculated as:

$$\begin{aligned} Area(PowerwiresH) = 2 * Length(nanowire\,plane) * (Wpw + Dpw) * 2 \\ = 2 * (2 * \text{No.Inputs} + 2 * \text{No.Outputs} + 2) * (Wnw + Dnw) * (Wpw + Dpw) * 2 \end{aligned} \tag{4.4}$$

$$Area(PowerwiresV) = 2 * Width(nanowire\,plane) * (Wpw + Dpw) * 2$$
$$= 2 * [(2^{\text{No.Inputs}} + 2) * (Wnw + Dnw) + 2 * 2 * (Wpw + Dpw)] * 2 * (Wpw + Dpw) \quad (4.5)$$

The total tile area is given by the sum of these three contributions:

$$Area_{Tot} = Area(Nanowire) + Area(PowerwiresH) + Area(PowerwiresV) \quad (4.6)$$

4.1.2 Power Estimation

The parameters used for the power estimation of a Nanotile are:

- R_{off}: Off-resistance of a nanowire for static power calculation;
- C_{gate}: Gate capacitance of a nanoFET for dynamic power calculation;
- C_{ds}: Channel capacitance of a nanoFET for dynamic power calculation;
- V_{dd}: Supply voltage on power wires;
- f: Operating frequency of control nanowires;
- SA: Switching activity of a nanowire (control nanowire or computation nanowire).

In [cite NanofaricPower-Graziano], the static power overestimated by the equation:

$$P_{static} = \text{No.NWs} * \frac{V^2}{R_{off}} \quad (4.7)$$

The dynamic power of a nanotile is sum of the dynamic power on each nanowire for control and computation signals [cite NanofaricPower-Graziano].

$$P_{dynamic} = \sum_{NWs} \frac{1}{2} * C_{nanowire} * V^2 * f * SA \quad (4.8)$$

4.1.3 Nanowire Capacitance ($C_{nanowire}$) Estimation

In order to evaluate the capacitance on a nanowire, it is necessary to take into account the number of transistors along the nanowire and the number of transistors' gates that the nanowire drives.

$$C_{nanowire} = \text{No.nanoFET}_{ch} * C_{ds} + \text{No.nanoFET}_{gate} * C_{gate} \quad (4.9)$$

No.nanoFET$_{ch}$: Number of nanoFETs along the nanowire;
No.nanoFET$_{gate}$: Number of nanoFETs driven by the nanowire.

In the case of 1-bit Full Adder, on each horizontal computation nanowire there are ($No.Inputs + 2 = 5$) nanoFETs along the nanowire, and ($No.Outputs = 2$) nanoFETs driven by it. On a vertical input nanowire there are only ($2^{\text{No.Inputs}}/2 = 4$) nanoFETs driven, and on a vertical output nanowire there are at maximum ($2 + 2^{\text{No.Inputs}} * 3/4 = 8$) along the wire and 2 nanoFETs driven. Therefore, the dynamic power of a nanotile can be divided into 3 components:

$$\begin{cases} P_{dyn}|inputs = \sum\limits_{inputNWs} \frac{1}{2} * C_{inputNW} * V^2 * f * SA \\ P_{dyn}|horiz.NWs = \sum\limits_{horiz.NWs} \frac{1}{2} * C_{horiz.NW} * V^2 * f * SA \\ P_{dyn}|vert.NWs(max) = \sum\limits_{vert.NWs} \frac{1}{2} * C_{vert.NW}(max) * V^2 * f * SA \end{cases} \tag{4.10}$$

With

$$\begin{cases} C_{inputNW} = (2^{\text{No.Inputs}}/2) * C_{gate} = constant \\ C_{horiz.NW} = (\text{No.Inputs} + 2) * C_{ds} + \text{No.Outputs} * C_{gate} = constant \\ C_{vert.NW}(max) = (2 + 2^{\text{No.Inputs}} * 3/4) * C_{ds} + 2 * C_{gate} = constant \end{cases} \tag{4.11}$$

the computation of the dynamic power can be simplified as

$$\rightarrow \begin{cases} P_{dyn}|inputs = \frac{1}{2} * V^2 * f * C_{inputNW} * \sum\limits_{inputNWs} SA \\ P_{dyn}|horiz.NWs = \frac{1}{2} * V^2 * f * C_{horiz.NW} * \sum\limits_{horiz.NWs} SA \\ P_{dyn}|vert.NWs(max) = \frac{1}{2} * V^2 * f * C_{vert.NW}(max) * \sum\limits_{vert.NWs} SA \end{cases} \tag{4.12}$$

4.1.4 Nanowire Switching Activity Computation

To obtain a more precise power estimation, it is necessary to estimate the tile switching activity.

Control Nanowires SA

As can be understood by looking at the timing diagram of Figure 4.2, the control signals (*Hpre, Heva, Vpre, Veva*) have a constant switching activity ($SA = 2$), which is also the maximum switching activity of a computation nanowire.

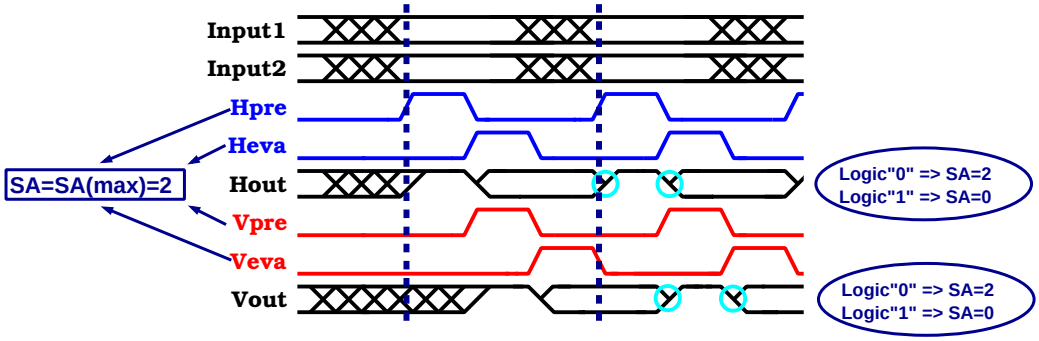


Figure 4.2. Nanotile switching activity.

Horizontal Nanowires SA

Instead of detailed analysis on the switching activity of each nanowire, the maximum SA ("2") can be used to calculate the maximum dynamic power. However, this approach greatly overestimates power consumption given that the number of nanowires is exponential to 2.
In a more realistic case, a computation nanowire swithes twice during a clock cycle only if the resulting logic on the wire is "0", and does not switch if the result is "1" because of the dynamic logic precharge.

Herein probability theory is introduced to improve the evaluation of dynamic power in Equation 4.12. The **average switching activity** of a nanowire is twice the **probability** that it is "0".

$$SA_{AVG} = 2 * Probability(NW = \text{“0”}) \tag{4.13}$$

Considering a 2-input AND gate nanotile as example (Figure 4.3 A)), without Karnaugh map simplification, only one horizontal nanowire switches, while the others stay stable. The switching activity is therefore:

$$\sum_{i=1}^{4} SA_{Ii} = SA_{I1} + SA_{I2} + SA_{I3} + SA_{I4} = 2$$

This situation can be also demonstrated with statistical probability. Since the

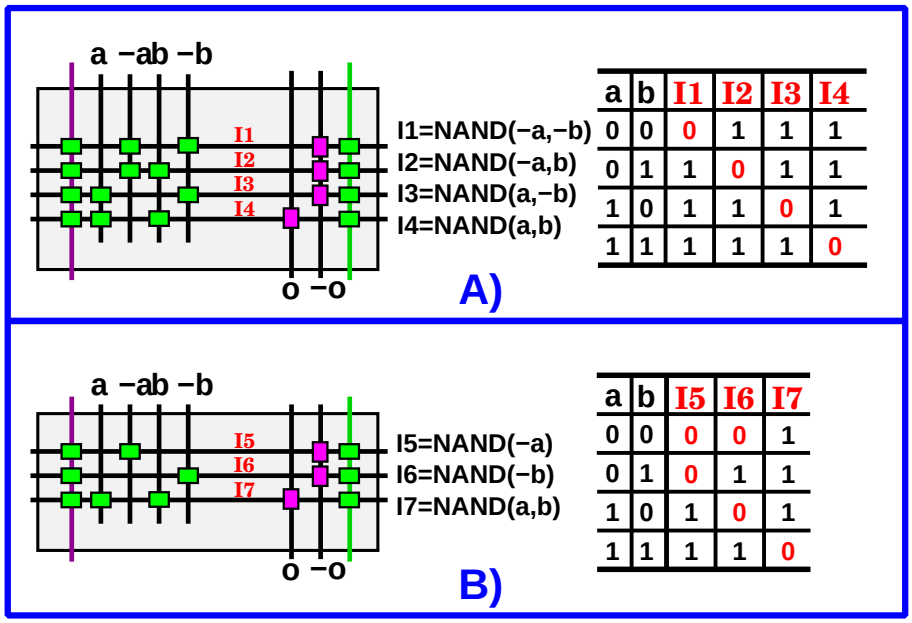


Figure 4.3. 2-input AND gate nanotile switching activity analysis. A) Without Karnaugh map simplification. B) With Karnaugh map simplification.

logic states of input "a" and "b" are independent events, the probabilities are:

$$\begin{cases} Probability(I1=0) = Prob(a=0) * Prob(b=0) \\ Probability(I2=0) = Prob(a=0) * Prob(b=1) \\ Probability(I3=0) = Prob(a=1) * Prob(b=0) \\ Probability(I4=0) = Prob(a=1) * Prob(b=1) \end{cases}$$

The average switching activity is therefore:

$\sum_{i=1}^{4} SA_{AVG}(Ii) = SA_{AVG}(I1) + SA_{AVG}(I2) + SA_{AVG}(I3) + SA_{AVG}(I4)$
$= 2*(Probability(I1=0)+Probability(I2=0)+Probability(I3=0)+Probability(I4=0))$
$= 2*(Prob(a=0)*Prob(b=0)+Prob(a=0)*Prob(b=1)+Prob(a=1)*Prob(b=0)+Prob(a=1)*Prob(b=1))= \mathbf{2}$

Within a nanotile without Karnaugh map simplification, the sum of SA of the horizontal nanowires is a constant value, and it is equal to 2:

$$\sum_{horiz.NWs} SA = 2 \tag{4.14}$$

If circuits are optimized with Karnaugh maps, it greatly increases the complexity of SA calculation. For example, as shown in Figure 4.3 B), the combination of

$(a = 0; b = 0)$ makes both horizontal wires "I5" and "I6" switch twice in a clock cycle. Tthe statistical computation of SA_{AVG} becomes complicated since specific logic combination probability is required.

$\sum_{i=5}^{7} SA_{AVG}(Ii) = SA_{AVG}(I5) + SA_{AVG}(I6) + SA_{AVG}(I7)$
$= 2*(Probability(I5 = 0) + Probability(I6 = 0) + Probability(I7 = 0))$
$= 2*(Prob(a = 0) + Prob(b = 0) + Prob(a = 1)*Prob(b = 1))> \mathbf{2}$

Given that considering Karnaugh map optimization does not introduce a relevant improvement to the model, and it increases the complexity of the model because power consumption depends on the combination probability. Herein the circuits used later are not optimized with Karnaugh map.

Input and Output Nanowires SA

The switching activities on output vertical nanowires are similar to horizontal computation NWs. Since the output signals are pairs of complementary logic, when the **O**output NW switches twice ($Output(\mathbf{O}) = 0$), its complementary output NW remain high because of $Output(-\mathbf{O}) = 1$. Therefore, $SA_{AVG}(O) + SA_{AVG}(-O) = 2$.

$$\sum_{vert.NWs} SA = 2 * \text{No.Outputs} \tag{4.15}$$

Considering in a complex NASIC circuit, each nanotile receives inputs are generated from other nanotiles, obtaining the same average switching activities as outputs. Therefore, each pair of inputs has the total average switching activity equal to 2.

$$\sum_{inputNWs} SA = 2 * \text{No.Inputs} \tag{4.16}$$

4.1.5 Simplified Dynamic Power Estimation

Taking into considerations Equation 4.14, Equation 4.15 and Equation 4.16, Equation 4.12 is simplified as below, which gives us an accurate power estimation for NASIC circuits.

$$\begin{cases} P_{dyn}|inputs = \text{No.Inputs} * V^2 * f * C_{inputNW} \\ P_{dyn}|horiz.NWs = V^2 * f * C_{horiz.NW} \\ P_{dyn}|vert.NWs(max) = \text{No.Outputs} * V^2 * f * C_{vert.NW}(max) \end{cases} \tag{4.17}$$

4.2 NASIC Circuits Implementation

Merging the behavioral VHDL modeling with embedded area and power estimation equations developed previously, a complete and detailed model of NASIC technology is created. It allows to design any kind of NASIC circuits, to simulate and verify the correct circuit behavior, and to obtain also information on area occupation and power consumption. This model is used to design and analyze complex circuits.

4.2.1 Ripple Carry Adder

The first circuit designed is a generic Ripple Carry Adder (Figure **??**). Given that each Full Adder has a delay of one clock cycle, a skew and deskew network is required for signal synchronization. These networks are makd by tiles that do not implement any logic function but delay signals by one clock cycle. They act like buffers. The area and power estimations of a 4-/8-/16-bit RCA circuit is shown in Table 4.1.

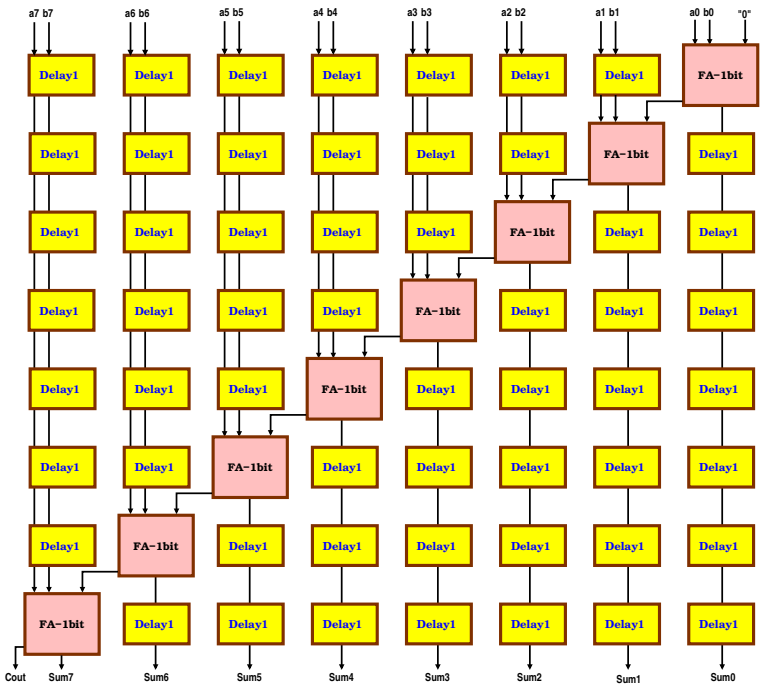


Figure 4.4. 8-bit Ripple Carry Adder NASIC block diagram with pre-skew and de-skew networks.

Table 4.1. Ripple Carry Adder area and power estimation with NASIC circuit modeling.

No.bits	AREA $[um^2]$	POWER $[mW]$
4	4.08	0.1
8	14.76	0.36
16	55.94	1.33

4.2.2 Array Multiplier

A more complex structure is the Array Multiplier, depicted in Figure 4.5 A). A nanotile containing a Full Adder and two AND gates is shown in 4.5 C). The structure of the whole multiplier is similar to the RCA. Skew and deskew networks are used again for signal synchronization (4.5 B)).

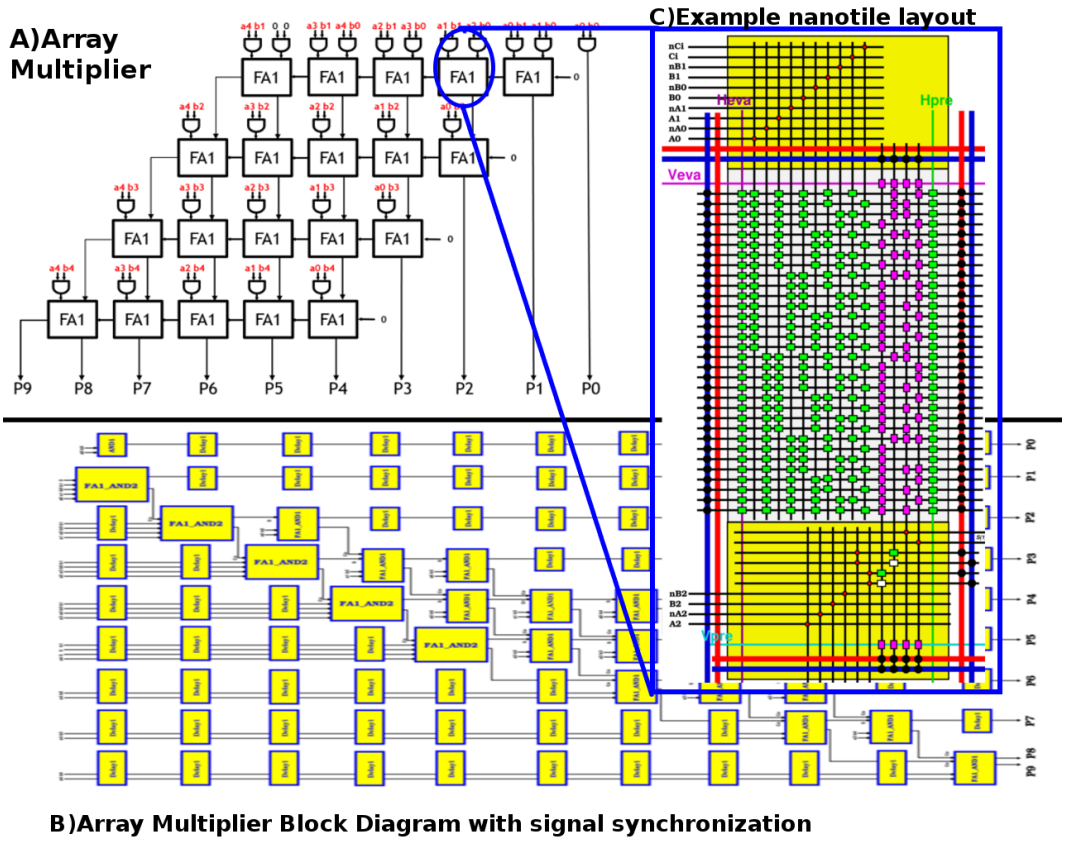


Figure 4.5. NASIC Array Multiplier. A) Circuit schematic of 5-bit Array Multiplier. B) Block diagram of 5-bit Array Multiplier. C)Example nanotile of 1-bit Full Adder with two AND gates at input.

The area and power estimation of the NASIC implementated Array Multiplier in 4 bits, 8 bits, and 16 bits are shown in Table 4.2.

Table 4.2. Array Multiplier area and power estimation with NASIC circuit modeling.

No.bits	AREA $[um^2]$	POWER $[mW]$
4	23	0.59
8	175	4.31
16	1341	32.1

4.2.3 Booth Multiplier

The Booth Multiplier is a very efficient multiplier in CMOS technology. Radix-4 Booth Multiplier algorithm can be expressed with the pseudo codes shown below:

```
i = 0
P = 0
B[-1] =0
while (i<=M-2) loop
    P <= P + vp (B[i+1], B[i], B[i-1])
    A <= A*4
    i <= i+2
end loop
```

Figure 4.6 A) depicts the block diagram of the Booth algorithm with 5-bit multiplicant. The encoder of the selection of **"vp"** (Table 4.3) follows the truth table in Figure 4.6 B). The multiplier uses mainly adders and multiplexers to implement the multiplication.

Table 4.3. "vp" selection truth table.

B[i-1]	B[i]	B[i+1]	vp
0	0	0	0
0	0	1	+A
0	1	0	+A
0	1	1	+2A
1	0	0	-A
1	0	1	-A
1	1	0	-2A
1	1	1	0

The NASIC implementation of the Booth Multiplier, including the first two multiplexers, the 2-level Encoder and the 2-level Adder/Substractor, is depicted in Figure 4.7.

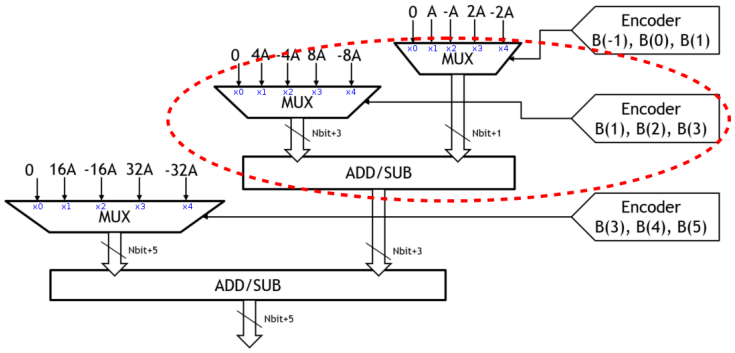


Figure 4.6. Booth Multiplier block diagram.

Also in this case skew and deskew networks are necessary for signal synchronization.

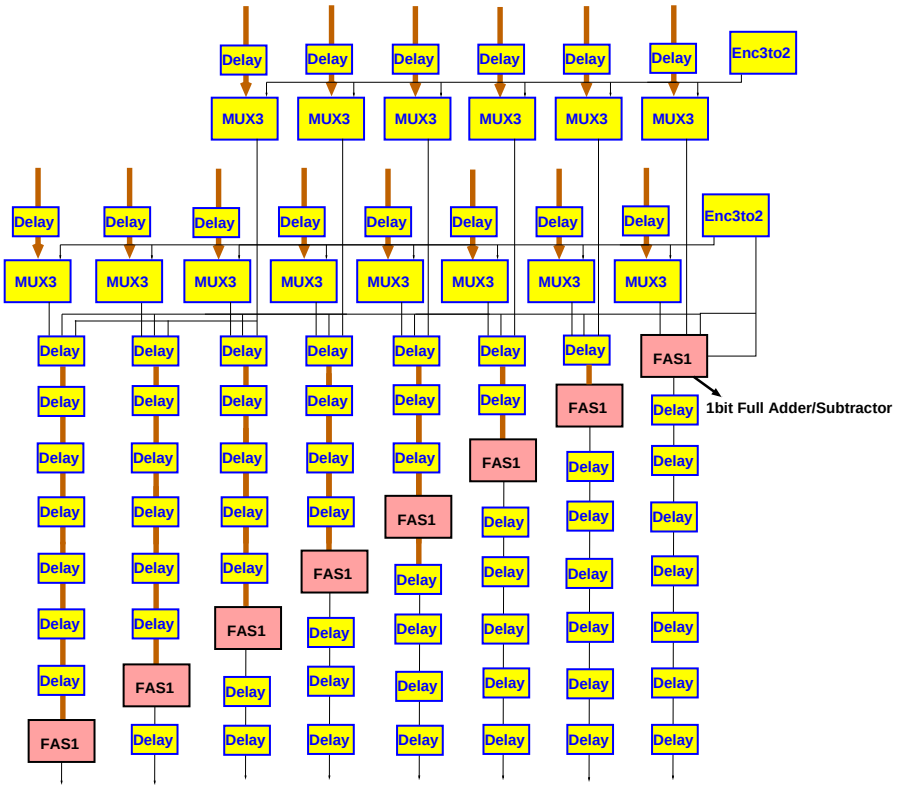


Figure 4.7. Block diagram of NASIC implementation on partial Booth Multiplier.

The area and power estimation of the NASIC implementated Booth Multiplier in 4 bits, 8 bits, and 16 bits is shown below (Table 4.4).

Table 4.4. Booth Multiplier area and power estimation with NASIC circuit modeling.

No.bits	AREA $[um^2]$	POWER $[mW]$
4	67	1.67
8	418	10.14
16	2926	69.57

Results highlight that in NASIC technology, different from CMOS, the array multiplier has much better performance. This is mainly due to its more regular layout. These results gives a clear indications of the best architectural choices for NASIC technology.

4.2.4 FIR

Finit Impulse Response (FIR) filter is a more complex architecture that uses both multipliers and adders. FIR filters follow the equation:

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + ... + b_N * x[n-N] = \sum_{i=0}^{N} b_i * x[n-i] \qquad (4.18)$$

The block diagram of a FIR filter of order 7 with NASIC implementation is shown in Figure 4.8. The multipliers are Array Multipliers and the adders are Ripple Carry Adders implemented previously. Moreover, a delay block is inserted for signal synchronization with the same latency as a RCA. Figure 4.9 presents part of the NASIC implementation of this FIR architecture in 4 bits. Nanotiles dedicated to signal interconnection are taken into account in the design, providing a realistic and accurate area and power analysis .

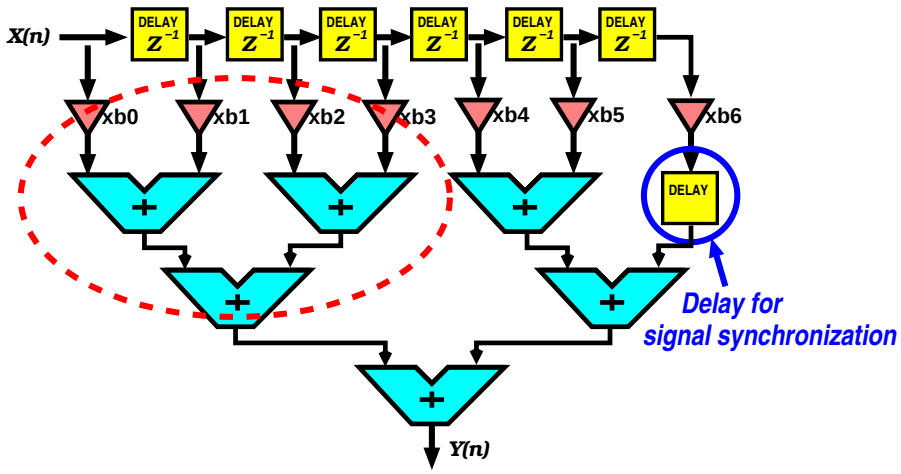


Figure 4.8. FIR Block diagram.

The area and power estimation of the NASIC implementated FIR (of order 8) in 4 bits, 8 bits, and 16 bits is shown in Table 4.5.

Table 4.5. FIR area and power estimation with NASIC circuit modeling.

No.bits	AREA $[um^2]$	POWER $[mW]$
4	441	10.9
8	2250	54.6
16	13810	329

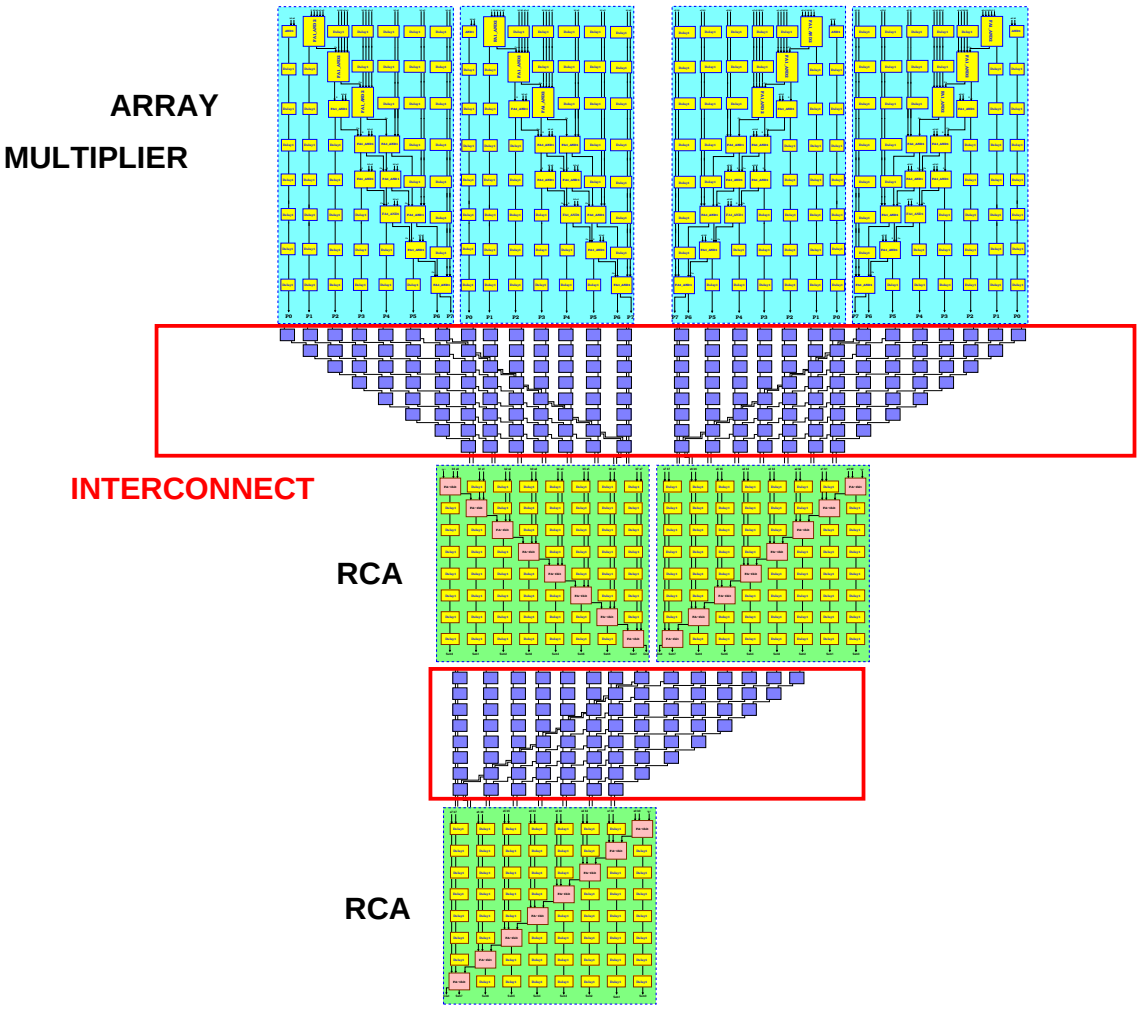


Figure 4.9. Detailed NASIC implementation of partial FIR architecture in 4 bits.

4.3 Structural Optimization

The analysis on NASIC circuits has highlighted the necessity of nanotiles or interconnection blocks with single clock delay. They are used as pre-skew and de-skew networks for signal synchronization. However, they occupy a great percentage of the circuit area. Taking the structure of Ripple Carry Adder (Figure 4.4) as example, a comparison table of area occupation with and without pre-skew and de-skew networks is shown in Table 4.6. In the 16-bit case, preskew and deskew networks represent more than 8/9 of the total area.

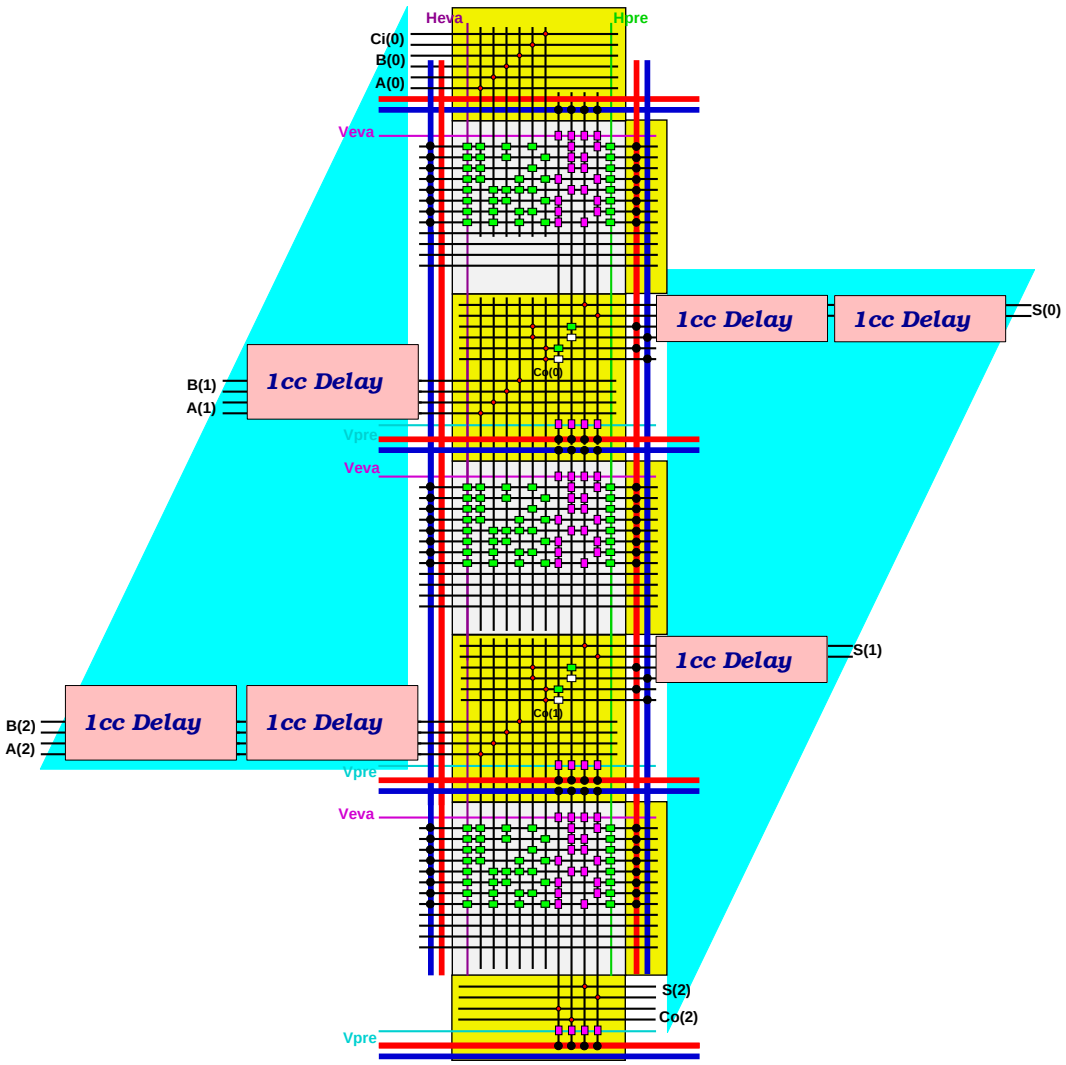


Figure 4.10. 3-bit Ripple Carry Adder structure in NASIC circuit layout.

Table 4.6. RCA area estimation comparison with and without pre-skew and de-skew networks.

No.bits	AREA $[um^2]$ with networks	AREA $[um^2]$ without networks
4	4.08	1.6
8	14.76	3.20
16	55.94	6.41

With the preskew and deskew networks, a 3-bit Ripple Carry Adder becomes a structure like Figure 4.10. Therefore, the idea of structural optimization for area saving on NASIC circuits is to eliminate the preskew and deskew networks. This is possible by accepting a reduction in circuit throughput at the exchange of a huge reduction in circuit area.

Since these delay networks are dedicated to signal synchronization, eliminating them requires a detailed signal synchronization analysis. Other than a purely combinational circuit, the analysis is better done by choosing an architecture with a loop, so an accumulator is employed. Its original representation in NASIC block diagram is shown in Figure 4.11, data parallelism is 4 bits and the feedback path latency is 2 clock cycles. Similarly to the NML case, **Circuit Throughput** = 1/**Loop Length**.

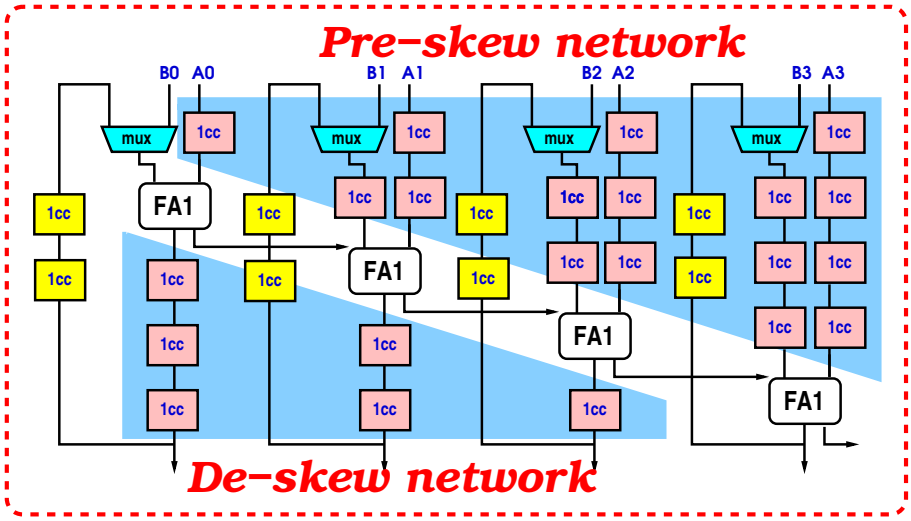


Figure 4.11. NASIC block diagram of 4-bit accumulator structure with preskew and de-skew networks. Feedback latency equals 2 clock cycles and total loop length is 5 clock cycles.

By removing the input and output synchronization blocks, the structure is reduced as in Figure 4.12.

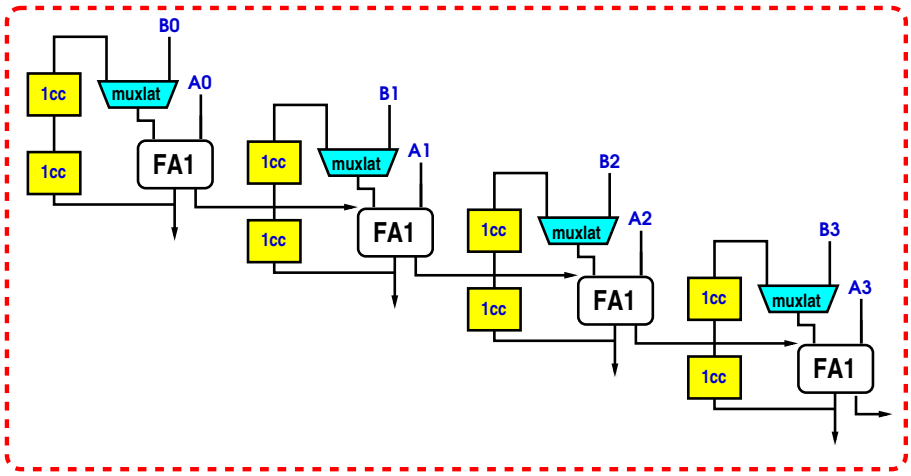


Figure 4.12. Block diagram of the optimized structure of 4-bit accumulator. The pre-skew and de-skew networks are eliminated.

Simulations show that, without preskew and deskew networks, if the circuit is purely combinational, there is a throughput reduction equal to the number of inputs. So a huge area is saved, at the cost of a reduction of performances. In case a loop is present, the situation is different:

1) If **No.bits** >= **Loop Length**, it must wait for ($No.bits$) clock cycles to update new input. => **Circuit Throughput** = 1/**No.bits**

2) If **No.bits** < **Loop Length**, it must wait for ($Loop\,Length$) clock cycles to update new input. => **Circuit Throughput** = 1/**Loop Length**

In the first case there is again a performance reduction. In the second case there is not. Since the original structure with networks has a throughput equal to 1/**Loop Length**, the new structure does not reduce the circuit original throughput at the same time it greatly reduce area and therefore power consumption.

This technique can be exploited in any circuit. Considering for example, a 2-level Accumulator (Figure 4.13), the optimized layout is depicted in Figure 4.14. The area estimation on both structures in case of 6-bit data and 2 clock cycles feedback is shown in Table 4.7.

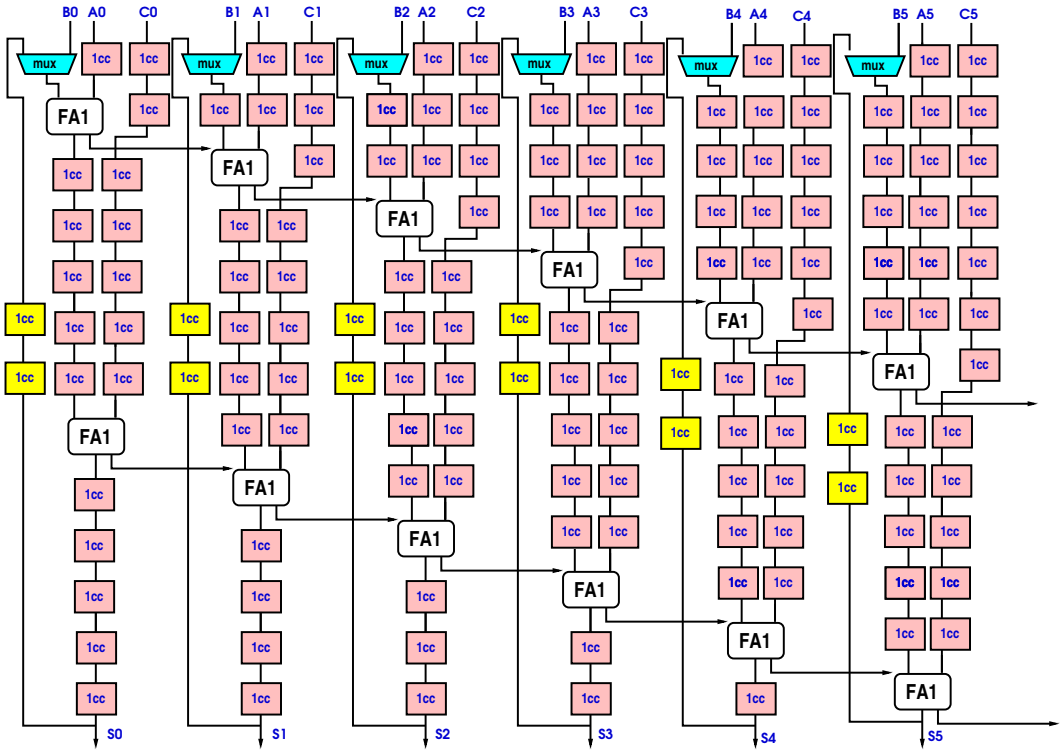


Figure 4.13. Detailed block diagram of traditional 6-bit accumulator with 2 RCAs in cascade.

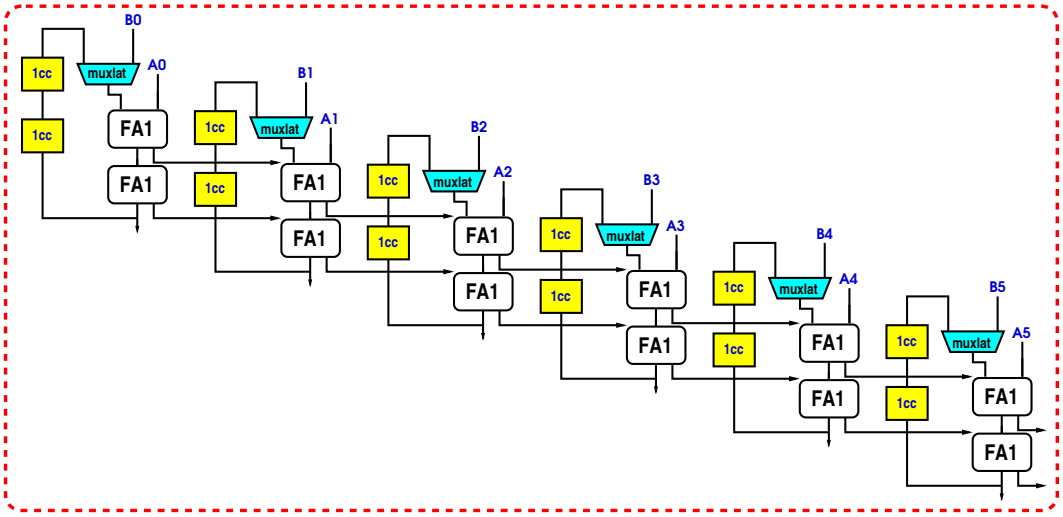


Figure 4.14. Detailed block diagram of optimized 6-bit accumulator with 2 RCAs in cascade.

Table 4.7. Area estimation comparison between two structures of 2-level Accumulator (data = 8/16 bits, feedback latency = 6 clock cycles).

No.bits	AREA $[um^2]$ with networks	AREA $[um^2]$ without networks
8	40.0	26.6
16	129.56	66.42

This technique can therefore provide a huge boost in performance to any circuit based on a technology with only a single layer available. NML, for example, can greatly benefit from it.

The analysis performed in this thesis work, clearly highlights how architecture optimization can improve emerging technologies. This work also highlights how, only using a methodology that links device level research with high level analysis, it is possible to evaluate the potential of a technology. For example, if a new device has a higher power consumption with respect to existing technology, it does not mean that the power consumption of a system based on this new technology will be higher than existing solutions.

Chapter 5

Conclusions

Emerging technologies are a growing and complex reality. Many new technologies are studied to replace or to complement the well developed CMOS circuits. To get a fair assessment of a technology with respect to CMOS in this thesis a new methodology has been developed. The methodology is based on the two main concepts. I) To validate a technology it is mandatory to analyze complex architectures. II) The analysis must be carried on considering technology fabrication constraints.

To reach this goal in this thesis high level models of two emerging technologies were developed. These models embedd informations obtained from experiemntal results and physical simulations to keep the obtained results as close as possible to the real circuits. Using these models complex circuit architectures were analyzed for NML and NASIC technologies. Results obtained allowed us to understand critical problems that arise at architectural level and are related to their intrinsic pipelined nature. As a results we were able to develop solutions to drastically improves performance.

To obtain these results was only possible thanks to the methodology here developed. Furthermore the technological improvements here developed can be applied also to existing CMOS circuits, where pipelining plays a key role to enhance performance. This is a further demonstration of the validity of the developed methodology.

Part I

Appendix

Appendix A

Memristive Devices

A.1 Introduction to Memristive Devices

In 1971 Leon Chua has suggested the existence of a fourth fundamental passive circuit element [34], called **Memristor**, combining the name of "memory" and "resistor". Its property value "Memristance (M)", is defined as a function relation between charge and flux, $d\varphi = Mdq$ as shown in Figure A.1. However, it was necessary to wait until the year 2008 for Strukov *et al.* to announce the successful implementation of a memristor at nanoscale level [35].

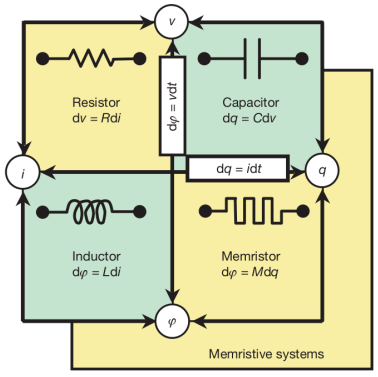


Figure A.1. The four fundamental two-terminal circuit elements: resistor, capacitor, inductor and memristor. *D.B. Strukov et al. "The missing memristor found", Nature, vol. 453, n. 1, 2008.*

According to [35], the memristor can be characterized by an equivalent time-dependent resistor whose value at a time t is linearly proportional to the quantity of charge q that has passed through it. It does not include a factor influenced

explicitly by magnetic field, which might be the reason why memristive devices are not so common in macro world.

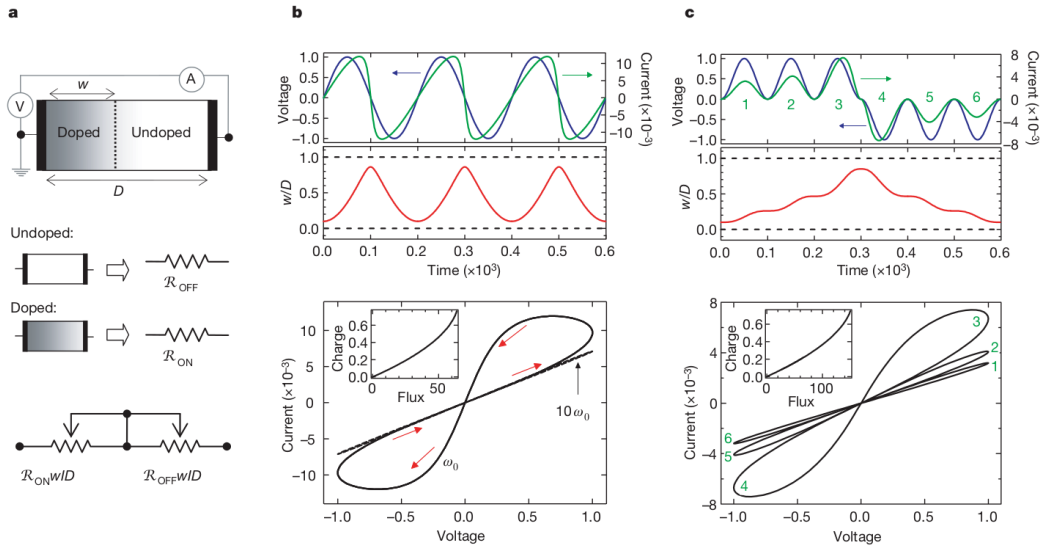


Figure A.2. The coupled variable-resistor model for a memristor. a) Diagram with a simplified equivalent circuit. b) c) The applied voltage (blue) and resulting current (green) as a function of time t for a typical memristor. The resistance ratio are in $ROFF/RON = 380$ b), and $ROFF/RON = 160$ in c). The insets in the i-v plots in b) and c) show that for these examples the charge is a single-valued function of the flux, as it must be in a memristor. *D.B. Strukov et al. "The missing memristor found", Nature, vol. 453, n. 1, 2008.*

In recent years large amount of attention and research efforts are done to study memristive devices, in order to exploit their potential in both logic and memory applications, such as memory storage, multi-state logic, and reconfigurable logic gates.

A.2 Memristive Devices and Switching Mechanisms

Taking inspiration from [35], more and more memristive devices are being studied in nanoscale devices. The main categories of memristive devices are *Thin Film Nanoarrays*, *Atomic Switches* [37], *Molecular Electronics* [38], and *Phase Change Memory* [39]. The most common devices are *Thin Film Nanoarrays*.

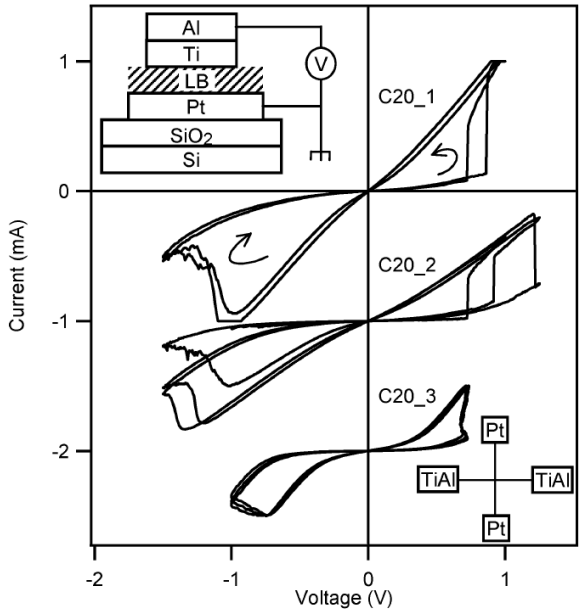


Figure A.3. DC I-V measurements showing the "figure-8" hysteresis loops of three different C_{20} molecular monolayer devices. *D. R. Stewart et al. "Molecule-Independent Electrical Switching in Pt/Organic Monolayer/Ti Devices", Nano Letters, vol. 4, n. 1, 2004.*

Thin Film Nanoarrays

The nanoscale thin film technology has already been employed in the field of resistance switching memory (ReRAM). A structure of metal/insulator/metal (MIM) is adopted, for example, $Pt/TiO_{2-x}/TiO_2/Pt$ is one of the most used materials. A memristor layout using thin film nanoarrays can be seen in Figure A.4 C), showing the possibility of implementing a nanoscale crossbar structure.

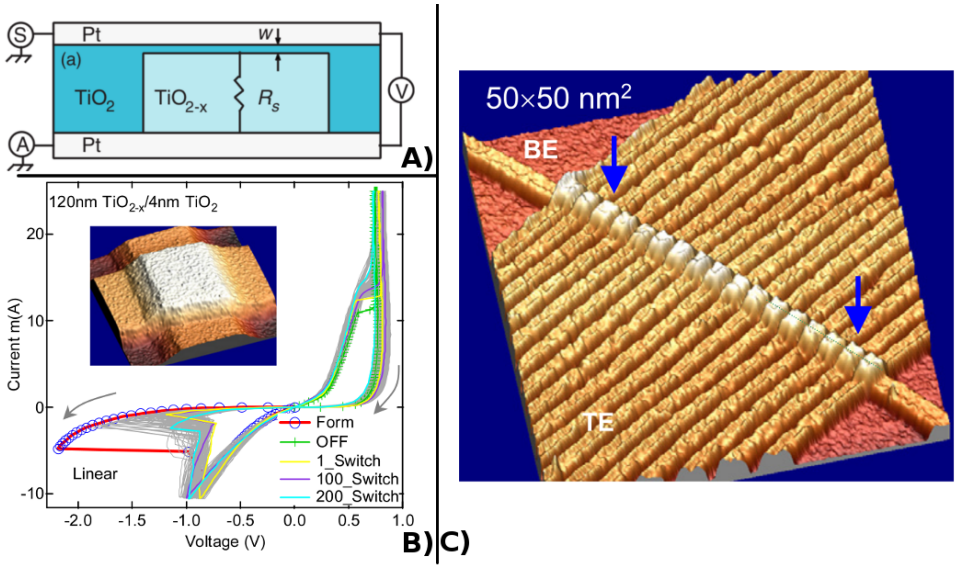


Figure A.4. A)Schematic of the device cross section after electroforming of $Pt/TiO_{2-x}/TiO_2/Pt$ with example switching i-v curve. B)The data from a Pt/120nm TiO_{2-x}/4nm TiO_2/Pt device, showing 200 consecutive switching loops after the forming step. C) AFM image of 1x17 nanojunctions. The cross-section profile shows 50 nm half pitch and 13 nm height nanowires.*A) M. D. Pickett et al. "Switching dynamics in titanium dioxide memristive devices", Journal of Applied Physics, 2009. B) C) J. J. Yang et al. "The mechanism of electroforming of metal oxide memristive switches", Nanotechnology, May 2009.*

Electrical switching behavior in metal oxide meristive devices is caused by the coupled motion of electrons and ions within the oxide material. There are two types of working principles in memristor cells, metallic filaments formed by electrochemical metalization (Figure A.5) [42] and localized high conductance channels of oxygen vacancies through the oxide film (Figure A.6) [41]. Since both of them involve electrochemical reactions and physical deformation, the reproducibility is a major concern for memristive devices, especially when including them in computational logic systems. This problem is mitigated by shrinking to the nanoscale and by carefully control voltage.

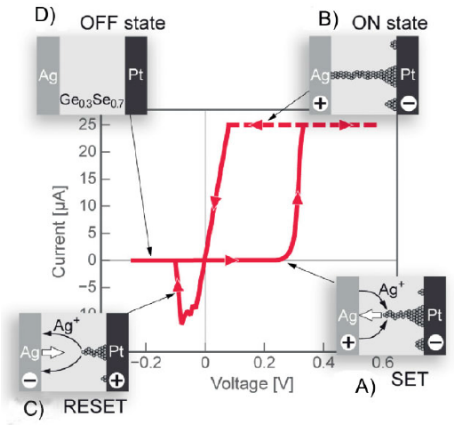


Figure A.5. Typical i-v characteristic of a Ag/Ag-Ge-Se/Pt electrochemical metalization cell. Starting from OFF state D), under the external electrical field rising, metallic filaments are grown gradually as A) to reach ON state B) creating galvanic metallic contacts. With voltage dropping C), the metal filaments dissolute, and resets the cell. *R. Waser et al. "Redox-Based Resistive Switching Memories - Nanoionic Mechanisms, Prospects, and Challenges", Advanced Materials, vol. 21, issue 25-26, July, 2009*

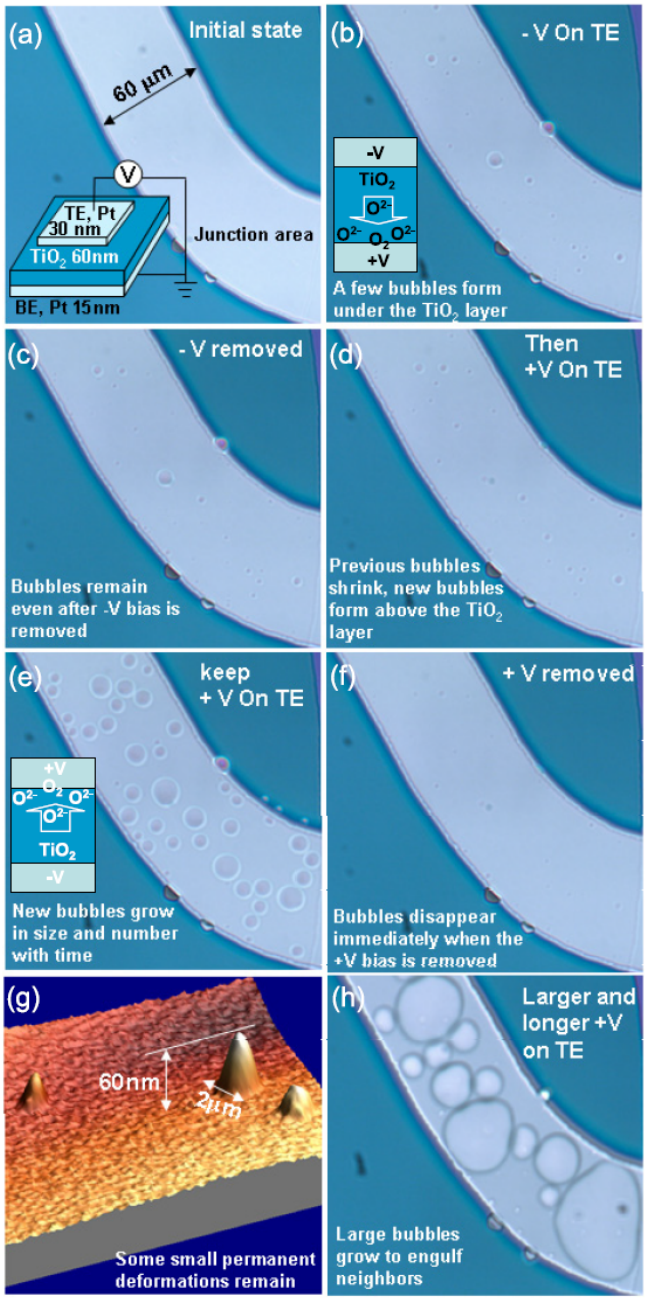


Figure A.6. Gas bubble behavior under electric field in a large 60um device for observation purpose. a) Junction initial state. b) c) Junction negative biased. d)-h) Junction positive biased. g) Atomic force micrograph of eruption features remaining after the bias voltage was removed. *J. J. Yang et al. "The mechanism of electroforming of metal oxide memristive switches", Nanotechnology, May 2009.*

The switching behavior in of memristors is explained in Figure A.7.

In [43], a thin film memristor cell ($W/TiO_2/TiO_{2-x}/W$) was modeled as two head-to-head Schottky diodes. By employing a "electron trapping-detrapping V_O (oxygen vacancy) induced modification of Schottky contact resistance" model, the bipolar switching can be explained as:

- (a) At the beginning of negative voltage bias from 0 to -3V, with negative voltage rising (curve 5), the cell shows a high resistance Schottky junction at bottom electrode (BE) side and a forward bias diode at top electrode (TE) side. At the same time, the majority of electrons are emitted from V_o leaving a high concentration of V_o^{2+} at the BE side.
- (b) When the bias voltage goes from -3V to 0, the resistance of BE Schottky junction reach the minimum value. It represents the high conductance curve (curve 6).
- (c) At the end of (b), the oxygen vacancies are neutralized when the voltage become zero. While it is rising to 3V, the Schottky diode at BE side is forward biased, and the diode at TE side shows high resistance (curve 7).
- (d) With electrons emission at both BE and TE side during the previous phase, low resistance remains while switching from 3V to 0 (curve 9).
- In (e) the same operation of point (a) is repeated. However, with bias voltage higher than -3.2V, local filamentary is formed and ohmic conduction path is created as shown in (f).

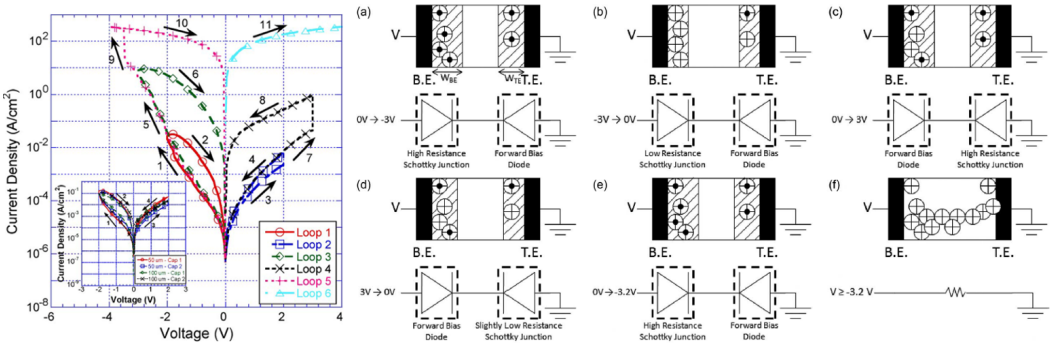


Figure A.7. Electron trapping-detrapping in V_o-induced modification of Schottky contact resistance model. *B. Long et al. "Understanding the Charge Transport Mechanism in VRS and BRS States of Transition Metal Oxide Nanoelectronic Memristor Devices", IEEE Transactions on Electron Devices, vol. 58, n. 11, Nov. 2011.*

A.3 Memristive Devices Architecture

Thin film memristors are normally organized with a crossbar structure, building therefore a hybrid CMOS/Memristor circuit as proposed in Figure A.8 [44]. It combines a CMOS subsystem integrating several layers of nanowire crossbars, where the cross nanojunctions are constructed with thin film memristive materials. This structure is ideal to design high density resistive memories.

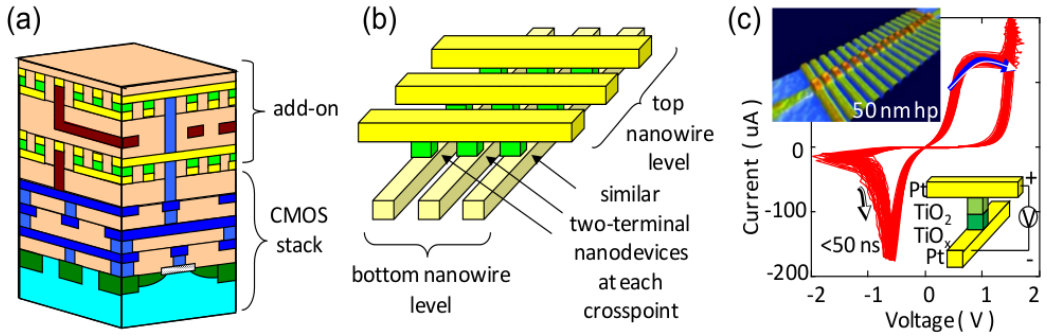


Figure A.8. Basic idea of 3D Hybrid CMOS/Memristor circuits. A) Stackup of CMOS subsystem with layers of memristor crossbar circuits. B) Memristor crossbar topology. C) micrograph of array of metal oxide memristive devices, and typical switching I-V curves. *D. B. Strukov, "3D Hybrid CMOS/Memristor Circuits: Basic Principle and Prospective Applications", COMMAD, Dec. 2012.*

Appendix B

SW NML Implementation VHDL Modeling

B.1 NML Power Estimation VHDL Model

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use work.QCA_package.all;

entity pot_clocco is
port(
        clk :   in std_logic_vector (3 downto 1);
                num_nanomag_f1: in real;
                num_nanomag_f2: in real;
                num_nanomag_f3: in real
    );
end pot_clocco;


architecture behavioural of pot_clocco is

  signal AREA_EFF_NANOMAG_F1, AREA_EFF_NANOMAG_F2,
AREA_EFF_NANOMAG_F3: real := init_real;
--NANOMAGNETS AREA EXPRESSED IN NANOMAGNETS
  signal LUNG_FILO_F1_NANOMAG, LUNG_FILO_F2_NANOMAG,
LUNG_FILO_F3_NANOMAG: real := init_real;
--LENGTH EXPRESSED IN NANOMAGNETS
  signal LUNG_FILO_F1, LUNG_FILO_F2,
LUNG_FILO_F3: real := init_real;
--WIRE LENGTH IN METERS
  signal LUNG_FILO_EFF_F1, LUNG_FILO_EFF_F2,
LUNG_FILO_EFF_F3: real := init_real;
--WIRE LENGHT CONSIDERING WHITE SPACE AMONG CLOCK ZONES
  signal LUNG_FILO_EFF_EFF_F1, LUNG_FILO_EFF_EFF_F2,
LUNG_FILO_EFF_EFF_F3: real := init_real;
--WIRE LENGHT CONSIDERING SERIAL CONNECTION OF CLOCK WIRES

  signal P_clock_to_mag_f1, P_clock_to_mag_f2,
P_clock_to_mag_f3: real := init_real;
```

```
--NUMBERS USED FOR POWER CALCULATION
  signal P_clock_RI_f1, P_clock_RI_f2, P_clock_RI_f3: real := init_real;
  --signal P_clock_LI_f1, P_clock_LI_f2, P_clock_LI_f3: real := init_real;
  signal S_WIRE: real;
--WIRES SECTION
  signal R_WIRE_F1, R_WIRE_F2, R_WIRE_F3: real := init_real;
--WIRES RESISTANCE
  signal L_WIRE_F1, L_WIRE_F2, L_WIRE_F3: real := init_real;
--WIRES INDUCTANCE
  signal Log_aritmo_f1, Log_aritmo_f2,
Log_aritmo_f3: real := init_real;
--LOGARITHM

  signal POWER_LI_tot: real := init_real;
  signal POWER_RI_tot, POWER_clock_to_mag_tot: real := init_real;
  signal CIRCUIT_AREA: real := init_real;
-- CIRCUIT AREA

begin

  AREA_EFF_NANOMAG_F1 <= num_nanomag_f1 * WASTED_SPACE;
  AREA_EFF_NANOMAG_F2 <= num_nanomag_f2 * WASTED_SPACE;
  AREA_EFF_NANOMAG_F3 <= num_nanomag_f3 * WASTED_SPACE;

  LUNG_FILO_F1_NANOMAG <= AREA_EFF_NANOMAG_F1 / WIDTH_ZONE_NANOMAG;
  LUNG_FILO_F2_NANOMAG <= AREA_EFF_NANOMAG_F2 / WIDTH_ZONE_NANOMAG;
  LUNG_FILO_F3_NANOMAG <= AREA_EFF_NANOMAG_F3 / WIDTH_ZONE_NANOMAG;

  LUNG_FILO_F1 <= LUNG_FILO_F1_NANOMAG * (HEIGHT_NANOMAG + NANOMAG_VERT_SEPAR);
  LUNG_FILO_F2 <= LUNG_FILO_F2_NANOMAG * (HEIGHT_NANOMAG + NANOMAG_VERT_SEPAR);
  LUNG_FILO_F3 <= LUNG_FILO_F3_NANOMAG * (HEIGHT_NANOMAG + NANOMAG_VERT_SEPAR);

  LUNG_FILO_EFF_F1 <= LUNG_FILO_F1 * HEIGHT_SPACE_REL;
  LUNG_FILO_EFF_F2 <= LUNG_FILO_F2 * HEIGHT_SPACE_REL;
  LUNG_FILO_EFF_F3 <= LUNG_FILO_F3 * HEIGHT_SPACE_REL;

  LUNG_FILO_EFF_EFF_F1 <= LUNG_FILO_EFF_F1 * WIRE_CURVE_REL;
  LUNG_FILO_EFF_EFF_F2 <= LUNG_FILO_EFF_F2 * WIRE_CURVE_REL;
  LUNG_FILO_EFF_EFF_F3 <= LUNG_FILO_EFF_F3 * WIRE_CURVE_REL;
  S_WIRE <= (WIDTH_ZONE-WIRE_SEPAR) * WIRE_THICKNESS;
  R_WIRE_F1 <= RESISTIVITA * (LUNG_FILO_EFF_EFF_F1/S_WIRE);
  R_WIRE_F2 <= RESISTIVITA * (LUNG_FILO_EFF_EFF_F2/S_WIRE);
  R_WIRE_F3 <= RESISTIVITA * (LUNG_FILO_EFF_EFF_F3/S_WIRE);

  Log_aritmo_f1 <= (4.0*LUNG_FILO_EFF_EFF_F1) / WIDTH_ZONE;
  Log_aritmo_f2 <= (4.0*LUNG_FILO_EFF_EFF_F2) / WIDTH_ZONE;
  Log_aritmo_f3 <= (4.0*LUNG_FILO_EFF_EFF_F3) / WIDTH_ZONE;

  L_WIRE_F1 <= LUNG_FILO_EFF_EFF_F1 * 2.0e-7 * (LOG(Log_aritmo_f1) -1.0);
-- CLOCK WIRES INDUCTANCE
  L_WIRE_F2 <= LUNG_FILO_EFF_EFF_F2 * 2.0e-7 * (LOG(Log_aritmo_f2) -1.0);
  L_WIRE_F3 <= LUNG_FILO_EFF_EFF_F3 * 2.0e-7 * (LOG(Log_aritmo_f3) -1.0);

  CIRCUIT_AREA <= (LUNG_FILO_EFF_F1 * (WIDTH_ZONE + WIRE_SEPAR)) +
(LUNG_FILO_EFF_F2 * (WIDTH_ZONE + WIRE_SEPAR)) + (LUNG_FILO_EFF_F3 *
(WIDTH_ZONE + WIRE_SEPAR));
-- CIRCUIT AREA EXTIMATION



-- POWER CALCULATION
-- PHASE 1
```

```
  Process_pot_clock_f1: process (clk(1))
  begin
   if clk(1) = '1' then
      P_clock_RI_f1 <= R_WIRE_F1 * I_MAX * I_MAX;
      --P_clock_LI_f1 <= ((L_WIRE_F1 * I_MAX * I_MAX)/2.0) / (T_CLOCK / 3.0);
      P_clock_to_mag_f1 <= POT_CLOCK_TO_MAG * num_nanomag_f1;
   elsif clk(1) = '0' then
      P_clock_RI_f1 <= 0.0;
      --P_clock_LI_f1 <= 0.0;
      P_clock_to_mag_f1 <= 0.0;
   end if;
  end process;

-- PHASE 2
  Process_pot_clock_f2: process (clk(2))
  begin
   if clk(2) = '1' then
      P_clock_RI_f2 <= R_WIRE_F2 * I_MAX * I_MAX;
      --P_clock_LI_f2 <= ((L_WIRE_F2 * I_MAX * I_MAX)/2.0) / (T_CLOCK / 3.0);
      P_clock_to_mag_f2 <= POT_CLOCK_TO_MAG * num_nanomag_f2;
   elsif clk(2) = '0' then
      P_clock_RI_f2 <= 0.0;
      --P_clock_LI_f2 <= 0.0;
      P_clock_to_mag_f2 <= 0.0;
   end if;
  end process;

-- PHASE 3
  Process_pot_clock_f3: process (clk(3))
  begin
   if clk(3) = '1' then
      P_clock_RI_f3 <= R_WIRE_F3 * I_MAX * I_MAX;
      --P_clock_LI_f3 <= ((L_WIRE_F3 * I_MAX * I_MAX)/2.0) / (T_CLOCK / 3.0);
      P_clock_to_mag_f3 <= POT_CLOCK_TO_MAG * num_nanomag_f3;
   elsif clk(3) = '0' then
      P_clock_RI_f3 <= 0.0;
      --P_clock_LI_f3 <= 0.0;
      P_clock_to_mag_f3 <= 0.0;
   end if;
  end process;

-- TOTAL POWER CONSUMPTION

  POWER_RI_tot <= (P_clock_RI_f1 + P_clock_RI_f2 + P_clock_RI_f3);
  POWER_LI_tot <= P_clock_LI_f1 + P_clock_LI_f2 + P_clock_LI_f3;
  POWER_clock_to_mag_tot <= (P_clock_to_mag_f1 + P_clock_to_mag_f2 + P_clock_to_mag_f3);


end behavioural;
```

B.2 NML 1-bit Full Adder VHDL code with power estimator

```
-- 1bit FULL ADDER with Xwire0

-- totally there are 6 registers


library ieee;
```

```
use work.QCA_package.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity FA_1 is
        port (
                clk :   in std_logic_vector (3 downto 1);
                A :     in std_logic;
                B :     in std_logic;
                Ci:     in std_logic;
                Co:     out std_logic;
                S :     out std_logic;
                num_nanomag_f1 : out real := init_real;
        num_nanomag_f2 : out real := init_real;
        num_nanomag_f3 : out real := init_rea);

end FA_1;

architecture STRUCTURAL of FA_1 is

        signal S1       : std_logic_vector(3 downto 0);
        signal S1_reg0  : std_logic_vector(3 downto 0);
        signal S2       : std_logic_vector(4 downto 0);
        signal S2_reg0  : std_logic_vector(4 downto 0);
        signal S3       : std_logic_vector(4 downto 0);
        signal S3_reg0  : std_logic_vector(4 downto 0);
        signal S4       : std_logic_vector(6 downto 0);
        signal S4_reg0  : std_logic_vector(6 downto 0);
        signal S5       : std_logic_vector(2 downto 0);
        signal S5_reg0  : std_logic_vector(2 downto 0);
        signal S6       : std_logic_vector(1 downto 0);
        signal S6_reg0  : std_logic_vector(1 downto 0);

        signal G4       : std_logic;

signal num_nanomag_tot_f1, num_nanomag_tot_f2, num_nanomag_tot_f3: real := init_real;

component INV is
        port(
                X: in std_logic;
                Y: out std_logic);
end component;

component MV is
        port(
                A_in: in std_logic;
                B_in: in std_logic;
                C_in: in std_logic;
                out_MV: out std_logic);
end component;

component Xwire is --XWIRE0
        port(
                in_up: in std_logic;
                in_down: in std_logic;
                out_up: out std_logic;
                out_down: out std_logic);
end component;

component register_generic is
        generic ( NBIT : integer );
```

```
        port(
                 CLK:   in std_logic;
                 D:     in std_logic_vector (NBIT-1 downto 0);
                 Q:     out std_logic_vector (NBIT-1 downto 0));
end component;

component pot_clocco
  port(
                clk :   in std_logic_vector (3 downto 1);
                num_nanomag_f1: in real;
                num_nanomag_f2: in real;
                num_nanomag_f3: in real
        );
end component;


        begin

        num_nanomag_tot_f1 <= ((6.0*N_MAG_FF)+(3.0*N_MAG_INV))
*INT_OV_LOGIC_GATE_LEVEL;
        num_nanomag_tot_f2 <= ((5.0*N_MAG_FF)+(2.0*N_MAG_MV)+N_MAG_MV)
*INT_OV_LOGIC_GATE_LEVEL;
        num_nanomag_tot_f3 <= ((5.0 * N_MAG_FF)+N_MAG_INV+N_MAG_MV)
*INT_OV_LOGIC_GATE_LEVEL;

        -- enter into Stage1
                S1(0)<= B;
                S1(1)<= A;
                S1(2)<= A;
        iv_stage1_0:    INV port map (Ci, S1(3));
                --S1(3)<= Ci;

        -- passing the signals through REG1
        REG1:   register_generic
                generic map (4)
                port map (clk(1), S1, S1_reg0);
        --endREG

        -- enter into Stage2, generate two Xwire gates
                cr_stage2_0:    Xwire port map (S1_reg0(0), S1_reg0(1), S2(0), S2(1));
                cr_stage2_1:    Xwire port map (S1_reg0(2), S1_reg0(3), S2(3), S2(4));

                S2(2)<= S2(1);

        -- passing the signals through REG2
        REG2:   register_generic
                generic map (5)
                port map (clk(2), S2, S2_reg0);
        --endREG

        -- enter into Stage3, generate one inverters, one Xwire
                S3(0)<= S2_reg0(0);
                S3(1)<= S2_reg0(1);

                cr_stage3_0:    Xwire port map (S2_reg0(2), S2_reg0(3), S3(2), S3(3));

                --iv_stage3_0:  INV port map (S2_reg0(4), S3(4));
                S3(4)<= S2_reg0(4);


        -- passing the signals through REG3
        REG3:   register_generic
```

```
            generic map (5)
            port map (clk(3), S3, S3_reg0);
        --endREG

        -- enter into Stage4, assign the signals
            --S4(0)<= S3_reg0(0);
            --S4(1)<= S3_reg0(1);
        iv_stage4_0: INV port map (S3_reg0(0), S4(0));
        iv_stage4_1: INV port map (S3_reg0(1), S4(1));

        --iv_stage4_2: INV port map (S3_reg0(2), G4);
            G4 <= S3_reg0(2);
            S4(2)<= G4;
            S4(3)<= G4;
            S4(4)<= G4;

            S4(5)<= S3_reg0(3);
            S4(6)<= S3_reg0(4);

        -- passing the signals through REG4
        REG4:   register_generic
            generic map (7)
            port map (clk(1), S4, S4_reg0);
        --endREG

        -- enter into Stage5, generate two MV and one INV
        mv_stage5_0:    MV port map (S4_reg0(0), S4_reg0(1), S4_reg0(2), S5(0));
        mv_stage5_1:    MV port map (S4_reg0(4), S4_reg0(5), S4_reg0(6), S5(2));
        iv_stage5_0:    INV port map (S4_reg0(3), S5(1));

        -- passing the signals through REG5
        REG5:   register_generic
            generic map (3)
            port map (clk(2), S5, S5_reg0);
        --endREG

        -- enter into Stage6, generic one MV and one INV
        iv_stage6_0:    INV port map (S5_reg0(0), S6(0));
        mv_stage6_0:    MV port map (S5_reg0(0), S5_reg0(1), S5_reg0(2), S6(1));

        -- passing the signals through REG6
        REG6:   register_generic
            generic map (2)
            port map (clk(3), S6, S6_reg0);
        --endREG

        -- assign the output

        Co <= S6_reg0(0);
        S  <= S6_reg0(1);

end STRUCTURAL;
```

B.3 NML Generic Ripple Carray Adder VHDL code with power estimator

```
-- Ripple Carry Adder (Generic)

-- Num_RCA defines the number of bits
```

```
-- the corresponding figure is in RCA.fig


library ieee;
use work.QCA_package.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;



entity RCA_generic is
        generic ( Num_RCA : integer := 9);
        port (
                clk :   in std_logic_vector (3 downto 1);
                A :     in std_logic_vector(Num_RCA-1 downto 0);
                B :     in std_logic_vector(Num_RCA-1 downto 0);
                Ci:     in std_logic;
                Co:     out std_logic;
                S :     out std_logic_vector(Num_RCA-1 downto 0);
                num_nanomag_f1 : out real := init_real;
        num_nanomag_f2 : out real := init_real;
        num_nanomag_f3 : out real := init_real);

end RCA_generic;

architecture STRUCTURAL of RCA_generic is


        signal Cin      : std_logic_vector(Num_RCA downto 1);
        signal X, Y, Z  : std_logic_vector(Num_RCA-1 downto 0);
        signal X_reg0, Y_reg0   : std_logic_vector(Num_RCA-1 downto 1);
        signal Z_reg0 : std_logic_vector(Num_RCA-1 downto 0);

signal num_nanomag_tot_f1, num_nanomag_tot_f2, num_nanomag_tot_f3: real := init_real;

signal num_temp_mag_f1: real_vector (Num_RCA*4-3 downto 0) := (others => init_real);
signal num_temp_mag_f2: real_vector (Num_RCA*4-3 downto 0) := (others => init_real);
signal num_temp_mag_f3: real_vector (Num_RCA*4-3 downto 0) := (others => init_real);

component FA_1 is
        port (
                clk :   in std_logic_vector (3 downto 1);
                A :     in std_logic;
                B :     in std_logic;
                Ci:     in std_logic;
                Co:     out std_logic;
                S :     out std_logic;
                num_nanomag_f1 : out real := init_real;
        num_nanomag_f2 : out real := init_real;
        num_nanomag_f3 : out real := init_real);

end component;


component sig_PG is
        generic (Num_PG : integer);
        port (
                clk:    in std_logic_vector(3 downto 1);
                X:      in std_logic;
```

```
                Z:      out std_logic;
                num_nanomag_f1 : out real := init_real;
        num_nanomag_f2 : out real := init_real;
        num_nanomag_f3 : out real := init_real);
end component;

component pot_clocco
  port(
                clk :   in std_logic_vector (3 downto 1);
                num_nanomag_f1: in real;
                num_nanomag_f2: in real;
                num_nanomag_f3: in real
        );
end component;

component somma_segnali is
  generic(
          INTERCONNECTION_OVERHEAD: real := 1.0
         );
  port(
       clock: in std_logic;
       f1_mag, f2_mag, f3_mag: in  real_vector;
       n1_mag, n2_mag, n3_mag: out real := init_real;
      );
end component;



        begin

        --assign the input signals
        X<= A;
        Y<= B;

        -- generate the registers for the propagation of input signal A except the first bit
        REG0_A:FOR i IN 1 TO Num_RCA-1 GENERATE

                PG_arrayA: sig_PG
                        generic map(i*6)
                        port map(clk, X(i), X_reg0(i),
num_temp_mag_f1(i-1), num_temp_mag_f2(i-1), num_temp_mag_f3(i-1));

        end GENERATE;
        --endREG

        -- generate the registers for the propagation of input signal B except the first bit
        REG0_B:FOR i IN 1 TO Num_RCA-1 GENERATE

                PG_arrayB: sig_PG
                        generic map(i*6)
                        port map(clk, Y(i), Y_reg0(i),
num_temp_mag_f1(Num_RCA-2+i), num_temp_mag_f2(Num_RCA-2+i),
num_temp_mag_f3(Num_RCA-2+i));

        end GENERATE;
        --endREG

        -- generate all the 1bit FA
        FA0: FA_1 port map (clk, X(0), Y(0), Ci, Cin(1), Z(0),
num_temp_mag_f1(Num_RCA*2-2), num_temp_mag_f2(Num_RCA*2-2),
num_temp_mag_f3(Num_RCA*2-2));
```

```
        G1:FOR i IN 1 TO Num_RCA-1 GENERATE

                FA_array: FA_1
                port map (clk, X_reg0(i), Y_reg0(i), Cin(i), Cin(i+1), Z(i),
num_temp_mag_f1(Num_RCA*2+i-2),
num_temp_mag_f2(Num_RCA*2+i-2),
num_temp_mag_f3(Num_RCA*2+i-2));
        END GENERATE;

        --assign the Co output
        Co<= Cin(Num_RCA);

        -- generate the registers for the propagation of output signal S
        REG0_S:FOR i IN 0 TO Num_RCA-1 GENERATE

                PG_arrayB: sig_PG
                        generic map((Num_RCA-1-i)*6)
                        port map(clk, Z(i), Z_reg0(i),
num_temp_mag_f1(Num_RCA*3+i-2),
num_temp_mag_f2(Num_RCA*3+i-2),
num_temp_mag_f3(Num_RCA*3+i-2));

        end GENERATE;
        --endREG


        S<= Z_reg0;

Summa: somma_segnali
  generic map(
              INTERCONNECTION_OVERHEAD => INT_OV_LOGIC_GATE_LEVEL
             )
  port map(
           clock => clk(1),
           f1_mag => num_temp_mag_f1,
           f2_mag => num_temp_mag_f2,
           f3_mag => num_temp_mag_f3,
           n1_mag => num_nanomag_tot_f1,
           n2_mag => num_nanomag_tot_f2,
           n3_mag => num_nanomag_tot_f3
          );

  num_nanomag_f1 <= num_nanomag_tot_f1;
  num_nanomag_f2 <= num_nanomag_tot_f2;
  num_nanomag_f3 <= num_nanomag_tot_f3;

-- CALCOLO DELLA POTENZA
  Pot_clock: pot_clocco
  port map(
                clk => clk,
                num_nanomag_f1 => num_nanomag_tot_f1,
                num_nanomag_f2 => num_nanomag_tot_f2,
                num_nanomag_f3 => num_nanomag_tot_f3
);


end STRUCTURAL;
```

Appendix C

NASIC VHDL Model

C.1 Nanotile Power Estimation VHDL Model

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use ieee.math_real.all;


package NASIC_package is

constant init_real : real := 1.0;

---------------constants for NASIC----------------
---------------TEMPORALI--------------------------
        constant CK_cycle       : time := 9 ns;
        constant CK_cycle_third : time := 3 ns;

        constant D_CK1  : real := 1.0e-1;
        constant D_CK2  : real := 5.0e-1;
        constant D_CK3  : real := 1.0e-1;
        constant D_CK4  : real := 5.0e-1;

        constant CK_space       : time := 1 ns;

        constant T_CK1_L1       : time := CK_space;
        constant T_CK2_L1       : time := CK_space*2;
        constant T_CK3_L1       : time := CK_space*3;
        constant T_CK4_L1       : time := CK_space*4;

        constant T_CK1_H        : time := CK_cycle*D_CK1;
        constant T_CK1_L2       : time := CK_cycle-T_CK1_L1-T_CK1_H;
        constant T_CK2_H        : time := CK_cycle*D_CK2;
        constant T_CK2_L2       : time := CK_cycle-T_CK2_H-T_CK2_L1;
        constant T_CK3_H        : time := CK_cycle*D_CK3;
        constant T_CK3_L2       : time := CK_cycle-T_CK3_H-T_CK3_L1;
        constant T_CK4_H        : time := CK_cycle*D_CK4;
        constant T_CK4_L2       : time := CK_cycle-T_CK4_H-T_CK4_L1;

---------------AREA e POTENZA---------------------------
```

```
        constant WIDTH_NW       : real  := 1.0e-8;
-- WIDTH NANOWIRE
        constant WIDTH_UW       : real  := 1.0e-7;
-- WIDTH MICROWIRE
        constant SPACE_NW       : real  := 1.0e-8;
-- SPACE BETWEEN TWO NANOWIRES
        constant SPACE_UW       : real  := 1.0e-7;
-- SPACE BETWEEN TWO MICROWIRES

        constant SPZWID_NW      : real  := WIDTH_NW + SPACE_NW;
-- WIDTH OCCUPIED IN PRESENCE OF ONE NANOWIRE
        constant SPZWID_UW      : real  := 2.0* WIDTH_UW + 2.0* SPACE_UW;
--WIDTH OCCUPIED BY A SET OF POWER MICROWIRES

        constant NTVdd          : real  := 2.0;
        constant NTFreq : real := 1.0e9;
        constant NTCgate        : real := 5.0e-17;
        constant NTCds          : real  := 5.65e-17;

function CalcArea (NumIn,NumNW_H,NumOUT,INTERCONNECTION_OVERHEAD: real)
return real;
function CalcPow (NumIn,NumNW_H,NumOUT: real)
return real;

end NASIC_package;



Package body NASIC_package is

        function CalcArea (NumIn,NumNW_H,NumOUT,INTERCONNECTION_OVERHEAD: real)
        return real is
        variable ANDwid, ORwid, H, NWArea, UWhor, UWver, UWArea, NTarea: real;
        begin
                ANDwid := (NumIN +2.0) * SPZWID_NW;
-- PIANO AND WIDTH including two Horizontal clock nanowires
                ORwid   := NumOUT * SPZWID_NW;
-- PIANO OR WIDTH
                H := (NumNW_H+2.0) * SPZWID_NW;
-- PIANO HIGHT including two Vertical clock nanowires
-- = Length of Vertical Microwire


                NWArea  := H * (ANDwid + ORwid);
-- PianoAND_WID + PianoOR_WID
--= TOTAL WIDTH of NANOTILE Nanowire space
--= Length of Horizontal Microwire

                UWhor := SPZWID_UW * (ANDwid + ORwid);
                UWver := SPZWID_UW * H;
                UWArea := 2.0* UWhor + 2.0* UWver;

                NTarea := (NWArea + UWArea) * INTERCONNECTION_OVERHEAD;
                return NTarea;
        end CalcArea;

        function CalcPow (NumIn,NumNW_H,NumOUT: real) return real is
        variable HWCds, HWCap, ORCds, ORCap, Pdclk, Pdin, Pdhw, Pdout, Pd_SUM: real;
        begin
-- Dyanmic Power Dissipation on Clock signal nanowires
-- Hpre and Heva dipend on NumHWire
-- Vpre and Veva dipend on NumOutWire
```

```
        Pdclk:=2.0*0.5*2.0*NumNW_H*NTCgate*NTVdd*NTVdd*NTFreq+2.0*0.5
*2.0*NumOut*NTCgate*NTVdd*NTVdd*NTFreq;

-- Dynamic Power Dissipation on Input signal nanowires
-- Switching activity of each input = 1.
-- Each input nanowire has the number of Gate Capacitance of 0.5*NumInWire.
                Pdin:=NumIn*0.5*1.0*0.5*NumIn*NTCgate*NTVdd*NTVdd*NTFreq;

-- Dynamic Power Dissipation on Horizontal signal nanowires
-- Each horizontal wire has the number of Gate Capacitance of
-- 0.5*NumOutWire.
-- Sum of horizontal wire switching activity = 2.
                HWCds := (2.0+0.5*NumIn)*NTCds;
                HWCap := 0.5*NumOut*NTCgate+HWCds;
                Pdhw:=0.5*2.0*HWCap*NTVdd*NTVdd*NTFreq;

-- Dynamic Power Dissipation on Output signal nanowires
-- Sum of a couple of outputs switching activity = 2.
-- Each output wire has the number of Gate Capacitance of 2*Cg.
                ORCds := (2.0+NumNW_H*0.5)*NTCds;
                ORCap := 2.0*NTCgate+ORCds;
                Pdout:=0.5*NumOut*0.5*2.0*ORcap*NTVdd*NTVdd*NTFreq;

                Pd_SUM:=Pdclk+Pdin+Pdhw+Pdout;
                return Pd_SUM;
        end CalcPow;

end NASIC_package;
```

Appendix D

NASIC Structure Optimization

D.1 Testbench for optimized 6-bit Accumulator structure

Testbench for optimized 6-bit Accumulator structure with feedback latency of 2 clock cycles.

```
-- testbench ACCUMULATOR ASYNCRONOUS
-- Looplength <= Nbit
-- Cycling Time limit = Nbit

library ieee;
use work.NASIC_package.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity testbench2_ACC6x2 is
end testbench2_ACC6x2;

architecture behavioral of testbench2_ACC6x2 is

COMPONENT ACCasync IS
GENERIC(        Nbit            :       INTEGER := 4;
                        NFF_loop        :       INTEGER := 1);
PORT (
        CLK                     :       IN      STD_LOGIC_VECTOR(4 DOWNTO 1);
        A, nA           :       IN      STD_LOGIC_VECTOR(Nbit-1 DOWNTO 0);
        B, nB           :       IN      STD_LOGIC_VECTOR(Nbit-1 DOWNTO 0);
        Ci, nCi         :       IN STD_LOGIC;  -- Ci, nCi carry inputs
        Init, nInit     :       IN STD_LOGIC;
        -- SELECT NEW SEED PORT
        W, nW           :       IN STD_LOGIC;
        -- REGISTER NEW VALUE
        Co, nCo         :       OUT STD_LOGIC;
        S, nS           :       OUT STD_LOGIC_VECTOR(Nbit-1 DOWNTO 0);
        SumAREA :       OUT real := init_real;
        SumPOWER:       OUT real := init_real
);
END COMPONENT;
```

```
-- SEGNALI INPUT OUTPUT
signal a_i      : std_logic_vector (5 downto 0) := "000000";
signal na_i     : std_logic_vector (5 downto 0) := "111111";
signal b_i      : std_logic_vector (5 downto 0) := "000000";
signal nb_i     : std_logic_vector (5 downto 0) := "111111";
signal Init_i   : std_logic:= '0';
signal nInit_i  : std_logic:= '1';
signal w_i      : std_logic:= '0';
signal nw_i     : std_logic:= '1';
signal ci_i     : std_logic := '0';
signal nci_i    : std_logic := '1';
signal co_i     : std_logic := '0';
signal nco_i    : std_logic := '1';
signal s_i      : std_logic_vector (5 downto 0);
signal ns_i     : std_logic_vector (5 downto 0);

signal area_i   : real := init_real;
signal power_i  : real := init_real;

signal clock: std_logic_vector(4 downto 1);


begin

dut: ACCasync
generic map(6, 2)
port map (
                CLK => clock,
                A   => a_i,
                nA  => na_i,
                B   => b_i,
                nB  => nb_i,
                Ci      => ci_i,
                nCi     => nci_i,
                Init => Init_i,
                nInit => nInit_i,
                W => w_i,
                nW => nw_i,
                Co  => co_i,
                nCo => nco_i,
                S   => s_i,
                nS  => ns_i,
                SumAREA => area_i,
                SumPOWER        => power_i
        );


-- GENERAZIONE DEL CLOCK1
CLOCCO1: process
begin
for i in 0 to 1000000 loop
   clock(1) <= '0';
   wait for T_CK1_L1;
   clock(1) <= '1';
   wait for T_CK1_H;
   clock(1)<= '0';
   wait for T_CK1_L2;
end loop;
end process;

-- GENERAZIONE DEL CLOCK2
```

```
CLOCCO2: process
begin
for n in 0 to 1000000 loop
   clock(2) <= '0';
   wait for T_CK2_L1;
   clock(2) <= '1';
   wait for T_CK2_H;
   clock(2) <= '0';
   wait for T_CK2_L2;
end loop;
end process;

-- GENERAZIONE DEL CLOCK3
CLOCCO3: process
begin
for t in 0 to 1000000 loop
   clock(3) <= '0';
   wait for T_CK3_L1;
   clock(3) <= '1';
   wait for T_CK3_H;
   clock(3) <= '0';
   wait for T_CK3_L2;
end loop;
end process;

-- GENERAZIONE DEL CLOCK4
CLOCCO4: process
begin
for s in 0 to 1000000 loop
   clock(4) <= '0';
   wait for T_CK4_L1;
   clock(4) <= '1';
   wait for T_CK4_H;
   clock(4) <= '0';
   wait for T_Ck4_L2;
end loop;
end process;

SendData: process
begin
        a_i     <= "000011";
        na_i<= "111100";
        b_i     <= "000010";
        nb_i<= "111101";
    wait for 7* 3 *CK_cycle_third; -- wait for (nbit+1)=7 clock

        a_i     <= "000101";
        na_i<= "111010";
    wait for 6* 3 *CK_cycle_third; -- wait for nbit=6 clock

        a_i     <= "001001";
        na_i<= "110110";
    wait for 6* 3 *CK_cycle_third; -- wait for nbit=6 clock

        a_i     <= "001111";
        na_i<= "110000";
    wait for 6* 3 *CK_cycle_third; -- wait for nbit=6 clock


wait;
end process SendData;
```

```
CONTROL: process
begin
-- 1st clock: Register seed B
    Init_i <= '1';
    nInit_i <= '0';
        w_i <= '1';
        nw_i<= '0';
    wait for 3 *CK_cycle_third;

    Init_i <= '0';
    nInit_i <= '1';
        w_i <= '0';
        nw_i<= '1';
    wait for 3* 3 *CK_cycle_third; -- wait for (fbk+1)=(2+1) clock cycles

        w_i <= '1';
        nw_i<= '0';
        wait for 3 *CK_cycle_third;

        w_i <= '0';
        nw_i<= '1';
    wait for 5* 3 *CK_cycle_third; -- wait for (nbit-1)=(6-1) clock cycles

for i in 0 to 1000000 loop
        w_i <= '1';
        nw_i<= '0';
        wait for 3 *CK_cycle_third;
        w_i <= '0';
        nw_i<= '1';
    wait for 5* 3 *CK_cycle_third; -- wait for (nbit-1)=(6-1) clock cycles

end loop;


wait;

end process;

end behavioral;
```

Bibliography

[1] C.S. Lent, P.D. Tougaw, W. Porod, and G.H. Bernstein. "Quantum cellular automata", *Nanotechnology*, vol.4, pp. 49â57, 1993.

[2] G. L. Snider, A. O. Orlov, R. K. Kummamuru, R. Ramasubramaniam, I. Amlani, G. H. Bernstein, C. S. Lent, J. L. Merz, W. Porod "Shape engineering for controlled switching with nanomagnet logic", *Dept. of Electrical Engineering, University of Notre Dame*, Notre Dame, Indiana, USA.

[3] Juanchi Wang, "Emerging Technologies For Biosequence Analysis" Master Thesis, *Politecnico di Torino, Dept. Eletr.*, Torino, Italy, Nov. 2012.

[4] R. K. Kummamuru, A. O. Orlov, R. Ramasubramaniam, C. S. Lent, G. H. Bernstein, and G. L. Snider, "Operation of a Quantum-Dot Cellular Automata (QCA) Shift Register and Analysis of Errors", *IEEE Transactions on Eletron Devices*, vol. 50, n. 9, Sept. 2003.

[5] A. Khitun, K. L. Wang, "Multi-functional edge driven nano-scale cellular automata based on semiconductor tunneling nano-structure with a self assembled quantum dot layer", *Superlattices and Microstructures*, vol. 37, pp. 55-76, 2005.

[6] C.S. Lent, B. Isaksen, "Clocked Molecular Quantum-Dot Cellular Automata", *IEEE Transactions on Electron Device*, vol. 50, no. 9, september 2003.

[7] M. Graziano, M. Vacca, M. Zamboni, "Magnetic QCA Design: Modeling, Simulation and Circuitsâ, Cellular Automata Innovative Modelling For Science And Engineering, Intechweb.org, 2011

[8] M.T. Alam, J. De Angelis, M. Putney, X.S. Hu, W. Porod, M.T. Niemier, G.H. Bernstein "Clocking Scheme for Nanomagnet QCA", *Center for Nano Science and Technology, Dept. of Electrical Engineering, Dept. of Computer Science and Engineering, University of Notre Dame*, Notre Dame, Indiana, USA.

[9] M. T. Niemier, G. H. Bernstein, G. Csaba, A. Dingler, X. S. Hu, S. Kurtz, S. Liu, J. Nahas, W. Porod, M. Siddiq, E. Varga, "Nanomagnet logic: progress toward system-level integration", *Dept. of Electrical Engineering, Dept. of Computer Science and Engineering, University of Notre Dame*, USA.

[10] M. Vacca, L. Di Crescenzo, M. Graziano, M. Zamboni, A. Chiolerio, A. Lamberti, E. Enrico, F. Celegato, P. Tiberto, and L. Boarino, "Electric clock for NanoMagnet Logic Circuits", *Dept. of Electronics and Telecommunications, Politecnico di Torino, Istituto Italiano di Technologia (IIT), Istituto Nazionale per la Ricerca Metrologica (INRIM), Italy*, 2013

[11] Marco Vacca, "Emerging Technologies - NanoMagnets Logic (NML)" Doctoral Thesis, *Politecnico di Torino, Dept. Eletr.*, Torino, Italy, Mar. 2013.

[12] M. Vacca, J. Wang, M. Graziano, M. RuoRoch, M. Zamboni, "Feedbacks in QCA: A Quantitative Approach", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, Oct. 2015.

[13] G. Csaba, and W. Porod "Behavior of Nanomagent Logic in the Presence of Thermal Noise", *Center for Nano Science and Technology, University of Notre Dame* , Notre Dame, Indiana, USA.

[14] M. Vacca, M. Graziano, and M. Zamboni "Majority Voter Full Characterization for Nanomagnet Logic Circuits" *IEEE Transactions on Nanotechnology*, vol 11, no. 5, Sept. 2012.

[15] M. Vacca, M. Graziano, M. Zamboni, "NanoMagnet Logic Microprocessor: Hierarchical Power Analysis", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, pp. 1410-1420, 2012.

[16] J. Wang, M.Vacca, M. Graziano, M. RuoRoch, and M. Zamboni "Biosequences analysis on NanoMagnet Logic", *International Conference on IC Design and Technology (ICICDT)*, May 2013.

[17] L. R. Murphy, A. Wallqvist and R. M. Levy, "Simplified Amino Acid Alphabets for Protein Fold Recognition and Implication for Foldingâ, *Protein Engineering*,vol. 13, pp. 149â152, 2000.

[18] G. Urgese, "Analysis and Design of an Optimized HW Accellerator for Protein Alignment", Master thesis,*Politecnico di Torino, Dept. Eletr.*, Torino, Italy, Sept. 2012.

[19] G. Causapruno, G. Urgese, M. Vacca, M. Graziano, and M. Zamboni, "Protein Alignment Systolic Array Throughput Optimization", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 1, pp. 68-77, 2014.

[20] G. Urgese, M. Graziano, M. Vacca, M. Awais, S. Frache, and M. Zamboni, "Protein Alignment HW/SW Optimizations", *The IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2012.

[21] J. Das, S. M. Alam, and S.Bhanja, "Low Power Magnetic Quantum Cellular Automata Realization Using Magnetic Multi-Layer Structures", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol 1, no. 3, Sept. 2011.

[22] P. Yang, R. Yan, and M. Fardy, "Semiconductor Nanowire: What's Next?", *Nanoletters*, pp. 1529-1536, 2010.

[23] W. Lu, P. Xie and C. M. Lieber, "Nanowire Transistor Performance Limits and Applications", *IEEE Transactions on Electron Devices*, vol. 55, Nov. 2008.

[24] G. Shen, and D. Chen, "One-dimensional Nanostructure for electronic and optoelectronic devices", *Front. Optoelectron. China*, pp. 125-138, 2010.

[25] Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K. H. Kim and C. M. Lieber, "Logic Gates and Computation from Assembled Nanowire Building Blocks", *Science*, vol. 294, Nov. 2001.

[26] P. Narayanan, J. Kina, P. Panchapakeshan, P. Vijayakumar, and K. S. Shin, M. Rahman, M. Leuchtenburg, I. Koren, C. O. Chui and C. A. Moritz, "Nanoscale Application Specific Integrated Circuits", *IEEE/ACM International Symposium on Nanoscale Architectures*, June 2011.

[27] Y. Cui, X. Duan, J. Hu, and C.M. Lieber. "Doping and electrical transport in silicon nanowires", *J. Phys. Chem. B*, vol. 104, no. 22, pp. 5213-5216, June 2000.

[28] T. Wang, Z. Qi, C. A. Moritz, "Opportunities and challenges in application-tuned circuits and architectures based on nanodevices", *First ACM International Conference On Computing Frontiers*, pp. 503-511, april 2004.

[29] T. Wang, P. Narayanan, M. Leuchtenburg, C. A. Moritz "NASICs: A Nanoscale Fabric for Nanoscale Microprocessors", *Electrical and Computer Engineering Department, University of Massachusetts Amherst*, USA.

[30] P. Narayanan, M. Leuchtenburg, T. Wang, C. A. Moritz, "CMOS Control Enabled Single-Type FET NASIC", *Electrical and Computer Engineering Department, University of Massachusetts Amherst*, USA.

[31] M. Graziano, S. Frache, M. Zamboni, " A Hardware Viewpoint on Biosequence Analysis: What's Next?", *âACM Journal on Emerging Technologies in Computing Systems*, Nov. 2013.

[32] P. Panchapakeshan, P. Vijayakumar, P. Narayanan, C. O. Chui, I. Koren, and C. A. Moritz, "3-D Integration Requirements for Hybrid Nanoscale-CMOS Fabrics", *IEEE International Conference on Nanotechnology*, Aug. 2011.

[33] P. Panchapakeshan, P. Narayanan, and C.A. Moritz, "N3ASICs: Designing Nanofabrics with Fine-Grained CMOS Integration", *IEEE/ACM International Symposium on Nanoscale Architectures*, 2011.

[34] L. Chua, "Memristor - The missing circuit element", *IEEE Transactions on Circuit Theory*, vol. CT-18, no. 5, pp. 507â519, Sep. 1971.

[35] D.B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found", *Nature*, vol. 453, n. 1, 2008.

[36] D. R. Stewart, D. A. A. Ohlberg, P. A. Beck, Y. Chen, R. S. Williams, J.O. Jeppesen, K. A. Nielsen, and J. F. Stoddart, "Molecule-Independent Electrical Switching in Pt/Organic Monolayer/Ti Devices", *Nano Letters*, vol. 4, n. 1, 2004.

[37] M. Aono and T. Hasegawa, "The Atomic Switch", *Proceedings of the IEEE*, vol. 98, issue. 12, 2010.

[38] C. Johns, D. A. A. Ohlberg, S. Wang, R. S. Williams, and M. S. Islam, "Nanoscale Switching Junctions Based on an Organic Monolayer of Molecules and Solid Electrolytes", *IEEE International Conference on Nanotechnology*, Aug. 2007.

[39] W. WeÅnica, M. Wuttigc, "Reversible switching in phase-change materials", *Materials Today*, vol. 11, pp. 20-27, June 2008.

[40] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices", *Journal of Applied Physics*, 2009.

[41] J. J. Yang, F. Miao, M. D. Pickett, D. A. A. Ohlberg, D. R. Stewart, C. N. Lau, and R. S. Williams, "The mechanism of electroforming of metal oxide memristive switches", *Nanotechnology*, May 2009.

[42] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-Based Resistive Switching Memories - Nanoionic Mechanisms, Prospects, and Challenges", *Advanced Materials*, vol. 21, issue 25-26, July, 2009.

[43] B. Long, J. Ordosgoitti, R. Jha, and C. Melkonian, "Understanding the Charge Transport Mechanism in VRS and BRS States of Transition Metal Oxide Nanoelectronic Memristor Devices", *IEEE Transactions on Electron Devices*, vol. 58, n. 11, Nov. 2011.

[44] D. B. Strukov, "3D Hybrid CMOS/Memristor Circuits: Basic Principle and Prospective Applications", *COMMAD*, Dec. 2012.