POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Logical and Memory Architectures in Nanomagnetic Technology

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

Candidato:
Daniel Melis

Aprile 2018

Abstract

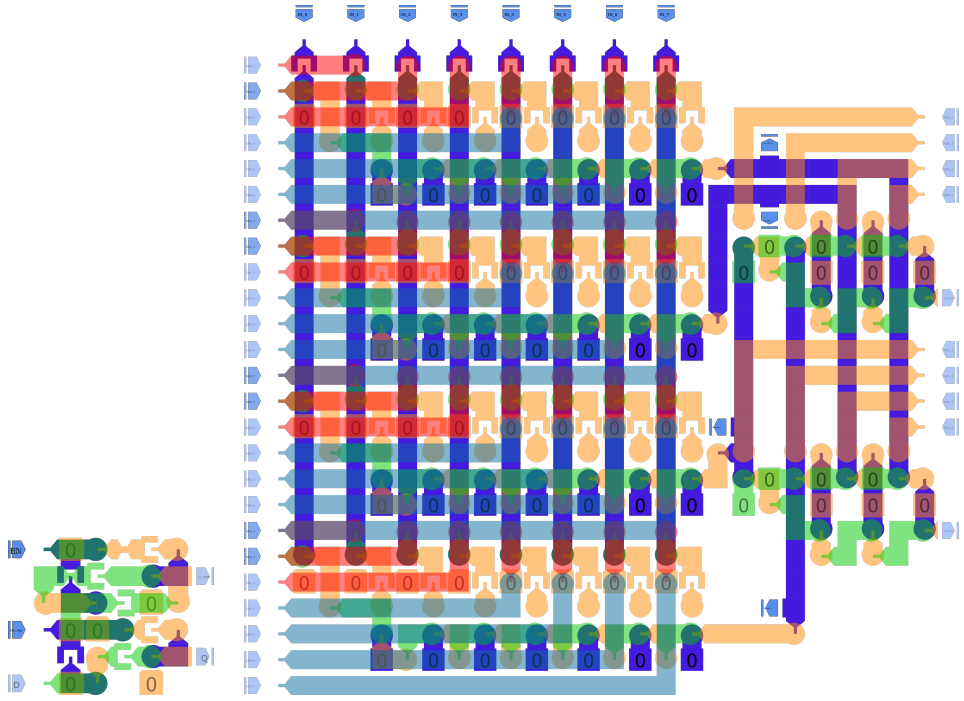
CMOS has been the standard technology for building digital circuits for decades, and one of the key reasons why Moore's law has been holding for such a long time; as a result, the electronic field has experienced astonishing and constant progresses throughout time. In recent years, unfortunately, this progress is in serious jeopardy. Atomic-scaled transistors, bringing along an almost unmanageable power density, will sooner or later become impossible to scale further.

For these reasons, technologist have been looking for alternatives for a while. Some of them are improvements to the CMOS technology, whereas others call for a complete change of paradigm. The QCA (Quantum Cellular Automaton) is a family of technology belonging to this second branch. The possible implementations are many, with some of them being more promising than others: the nanomagnetic implementation (NML, Nano Magnetic Logic) is a fairly good implementation, completely compliant with current CMOS fabrication process, and with interesting features as far as power dissipation and area are concerned. Nanomagnetic technology, in turn, has many different implementations. Among them, the most peculiar one is pNML (perpendicular Nano Magnetic Logic), which is the technology addressed in this thesis work. Being very much 3D oriented, very low power, and being less constrained than other NML are some of its most important and interesting traits. The 3D feature is a great advantage over CMOS, which allows the integration of many metal layers, but just for interconnections, with the whole logic lying on the bottom area of the circuit. pNML is also interconnections-free, since each wire also can be thought as a computational device. Last, as all the other QCA technologies, pNML exhibits a extremely pipelined signal propagation fashion.

With this thesis, different 3D pNML architectures are explored. For this purpose, the tool MagCAD has been adopted. One of the first aims consists in trying to take advantage of the 3D feature. Previous works used more than one layer, but just with the purpose of avoiding wire crossings or as VIAs. In other words, previous works analyzed two-layer architectures with a second layer devoted to the interconnection routing. This work instead endeavors to place roughly the same amount of logic in all the used layers, trying therefore to make the third dimension perfectly equivalent to the two planar ones. That is to say, the idea consists in developing the circuit towards the vertical direction rather than enlarging the footprint. An important requirement is also the characterization of the circuits, before and after the design, in particular with respect to the delay. The design has to be done considering beforehand the delays, which has to be finely tuned in order to achieve the best possible timing.

The research of effective design techniques and the sensible use of the 3D feature can be successfully done only with moderately complex circuits, and this is another important point of the thesis. Despite current technology studies just proving to be able to implement architectures based on two or three layers, the 3D designs presented in this work see the adoption of up to 14 layers.This study has been carried out to explore the feasibility of logic architectures based on pNML structures evaluating how the use of multiple layers can impact the performances. In order to obtain a substantial analysis, version limited to two layers has been developed for some of the proposed architecture.

The most elaborated algorithm implemented in this thesis is the summed area table (SAT), particularly suited because its complexity can be defined by choosing the number of elements, the word width and the control strategy. Moreover, it is an incremental algorithm, allowing many kinds of optimizations and implementations. The algorithm requires several memory, logical and control elements, all done both in 3D and in two layers. Some of them are represented in figure 1. The one in



(a) (b)

Figure 1: Example of pNML circuits designed in this thesis: (a) is a memory cell made of two inverters; (b) is a PLA composed of four ANDs and two ORs

figure 1a shows a memory cell, made with two cascaded multiplexer: one of them actually stores the input data, the other one serves as bit line (if the cell is not to

be written or read, it is bypassed by means of that multiplexer). This is an example of the alternative structures implemented with this technology. The other one, in figure 1b, is a PLA (Programmable Logic Array), with two OR planes and four AND plane. It is made of five layers, that are very convenient to allocate the large amount of wiring needed in such a circuit. The hardware is modular, made with a basic element repeated several times; therefore, it could carry out the algorithm over a matrix with arbitrary size. It is tested with a 2 by 2 matrix and a 4 by 4 one.

The last job involves a technological study of the physical parameters, in order to find out possible delay optimizations. The influence of the most relevant parameters is found, so that in case the optimal value of technological variable is to be found, a method to find it is already available.

The results are encouraging: first of all, no circuit whatsoever has been found to be unfeasible; even though it might seem obvious, it is actually not guaranteed for such a new technology. From the point of view of the area occupation, this new technology could even outperform some of the most recent CMOS technological nodes. In this regard, being able to stack the logical elements one over the other is a great advantage in terms of area reduction. Therefore, the two-layer versions are undoubtedly larger, but that is really the worst case condition. The delay analysis shows that CMOS is much faster, by several orders of magnitude. Anyway, the interest towards pNML does not lie in its delay performances, but rather in its low power and area features. A careful design and a technological parameters' optimization yield a quite good improvement, but still very far from CMOS performances. The plots in figure 2 summarize what are the final results obtained by the study of the SAT algorithm implemented. The 2a plot represents the delay. The times shown

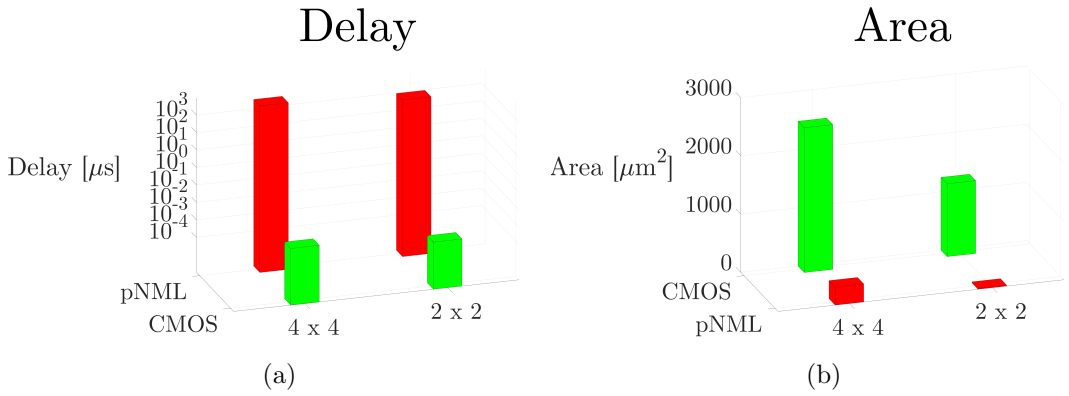


Figure 2: Delay and area results of this work for the pNML circuits performing the SAT algorithm for matrices with size 2 by 2 and 4 by 4, compared with the CMOS implementations

are those needed to complete a SAT algorithm over a matrix with 16 locations (the bars with the *4 x 4* label) and with 4 (the bars with the *2 x 2* label). Likewise, the

2b represent the area that the two versions (4 x 4 and 2 x 2) of the circuit occupy when manufactured in the two technologies analyzed (pNML and CMOS).

The thesis is structured this way: chapter 1 contains an introduction to QCA and nanomagnetic technologies, in particular to pNML. The description is not rigorous, the aim is setting the stage to understand what follows. Chapter 2 is devoted to the delay analysis. Delays are one of the most baffling traits of pNML, and tackling them from the beginning will certainly be beneficial for future understanding. Chapter 3 shows some logical, memory and control elements in their 3D implementation. They are designed and characterized, building then a library of components for the SAT hardware, which is presented in chapter 4. This latter chapter illustrates the algorithm and the hardware for its implementation. Chapter 5 is devoted to the performance analysis of the SAT hardware, compared with various CMOS implementations. In chapter 6, the SAT hardware and some of the most telling circuit are designed in two layers; some performance comparison follows. Chapter 7 contains the technological optimization of the parameters, in order to make the circuits designed faster. Last, in chapter 8 the conclusions and possible future work are presented.

The first appendix contains a short note about the delay model used in this thesis. The following one contains a series of general rules, advices and tricks to design in pNML. Next chapter shows how to make 3D drawings of the circuits designed in MagCAD. The fourth appendix chapter is about a power analysis of the CMOS circuits implementing the SAT is in. The last chapter contains some advices for the use of MagCAD.

Table of contents

List of tables

List of figures

Chapter 1

Introduction to pNML

The expressions "QCA" (Quantum Cellular Automata, [1]) defines a family of new technologies that can be used to represent digital values, and have all in common the fact that the information is represented by means of the two possible states of a quantum variable, rather than a voltage. Here this technological family is briefly described, in particular its nanomagnetic implementation, which is the technology used for this thesis work.

1.1 QCA and logical functions

The basic idea is defining a volume with a high potential barrier, and injecting two electrons in it. This structure is called "cell". The two electrons cannot escape from the cell, and they will settle within the potential well in an arrangement that makes their distance the greatest, because they tend to repel each other. What happens is pictured in a simple drawing in figure 1.1. The picture shows that, when trying to get as far from each other as possible, the two electrons arrange themselves in two alternative patterns, that can be encoded as the two logical values, 1 and 0. A cell alone is not very useful, but luckily, when two or more cells are brought close together, they interact. Let us refer to figure 1.2. If the state of the leftmost cell is "forced" somehow from the outside, the cell on its right will be influenced, and switch its state because the other one is more energetically favorable. The third cell will be influenced as well, and so on until the far right end of this arrangement,

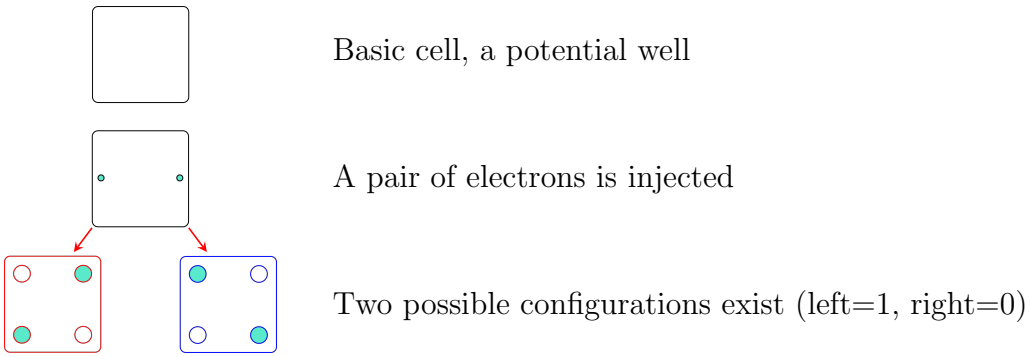


Figure 1.1: The two possible arrangements of a QCA cell, marked in red and blue

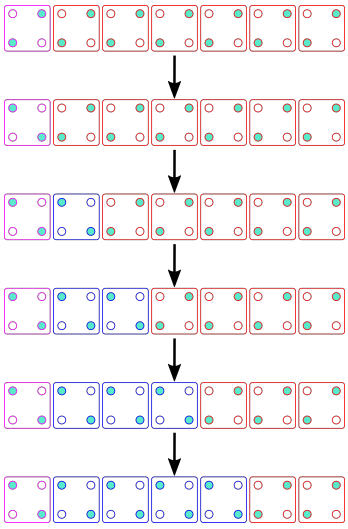


Figure 1.2: Example of information propagation through a QCA wire. Time evolves towards the direction of the arrows

that might be legitimately defined "wire", is reached. A particular arrangement of cells is able to propagate the opposite value of the input. This inverter is shown in figure 1.3.

This technological family has also a more advanced logical elaborating device, the majority voter. It can be seen drawn in figure 1.4. The device has three inputs and one output. It has a cell in the middle (the green one), that will hold the result of the computation, influenced by all the the final ends of each input wire, that could undergo opposing forces. In figure 1.4, for instance, the bottom input would cause the elaborating cell to assume the state "1", and the two remaining ones would cause it to get into the "0" state. These are somehow "stronger" than the bottom input

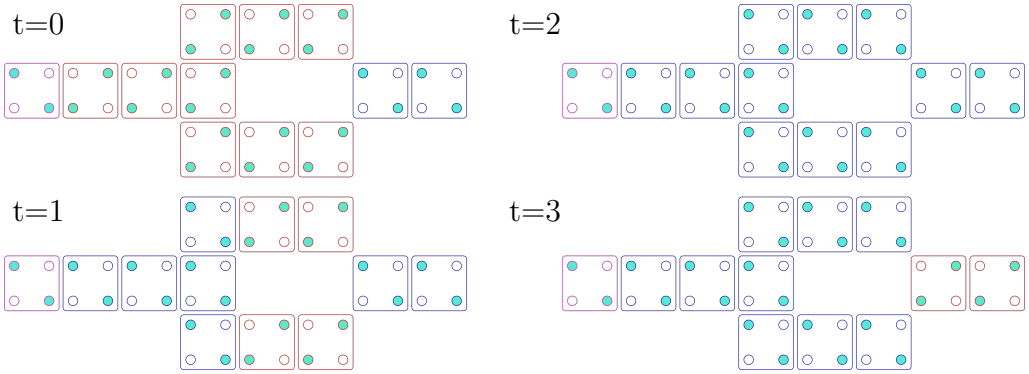


Figure 1.3: QCA inverter and example of signal propagation

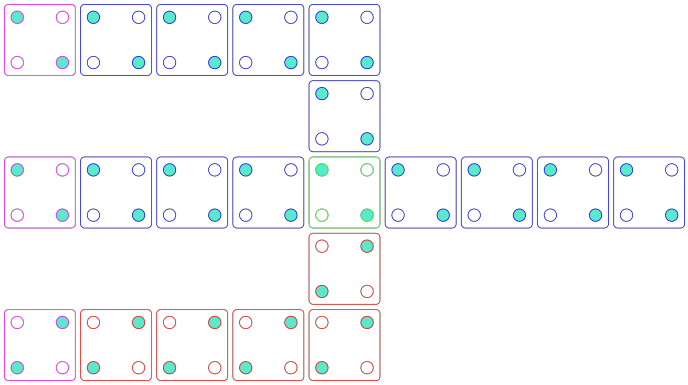


Figure 1.4: QCA Majority Voter

alone, and the final state of the elaborating cell will be 0. Elementary boolean logic proves that if one of the inputs, no matter which one, is always forced in the same state, the device will exhibit either an AND-like or an OR-like logical behavior. For sake of completeness, table 1.1 shows the truth table of such a gate.

In_1	In_2	In_3	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 1.1: Majority Voter's truth table

1.2 QCA Clocking

The description of the propagation phenomenon of the previous section, as it is, leads to non working circuits. The reason is found in the energy configuration of the device: basically, the energy barrier between the two states is too high for the "forcing" cell to cause the cell on its right to switch its state. Thus, the leftmost cell would be switched by some mechanism on the outside, and would be the only cell switching. A way to lower this energy barrier is needed. Once it is found, it must be implemented with some care; issues (namely, backdriving, stuck or unstable circuits) would come out otherwise. Consider figure 1.5. The black cell located exactly in the

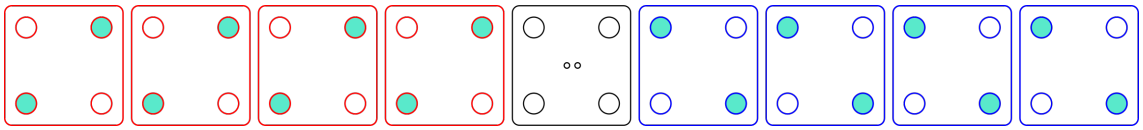


Figure 1.5: Example of non working QCA propagation mechanism

middle has all the cells on its left trying to force a state, and all the cells on its right trying to force the other one. The cell itself is not in anyone of those two states, and experiences a strong influence from both sides. Therefore, some sort of timing or pattern must be found to avoid ambiguous cases like this one. The mechanism that lowers the barriers is in fact the clock, which, in QCA, both times the circuit and supplies it with power. Physically, it is a field (depending on the implementation could be electric, magnetic of both) that provides the energy needed to overcome

the energy barrier. The clock scheme is a four-phases one, shown in figure 1.6. The

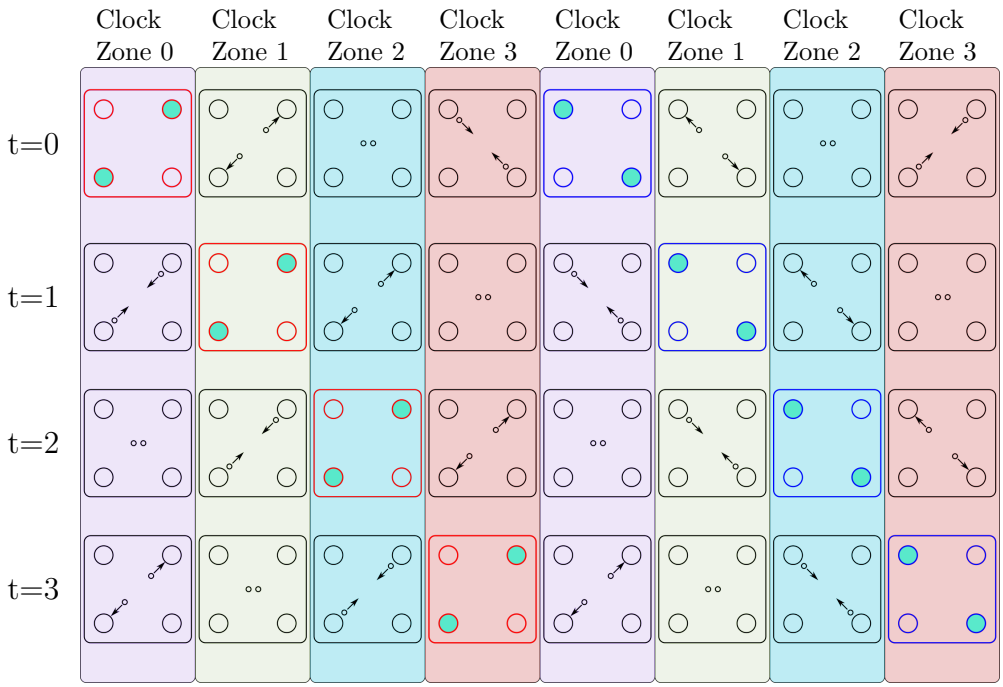


Figure 1.6: QCA clock zones and possible time evolution

meaning of the drawing should be self-explanatory, but in figure 1.7 the "legend" is shown. A preliminary thing to say is that the introduction of the clock, that lowers

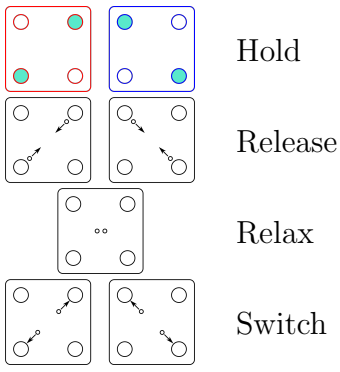


Figure 1.7: Representation of cell states in every clock phase

the cell's potential barrier, adds a possible state to the already existing two. This state is a "Null" state, something like the boolean "X" state; in this state the cell is in neither of the two logical values, and is highly influenced by the neighbors.

As the figure shows, the four phases are:

- **Relax**: the potential barrier of the cell is low, the cell is not in a defined logical state, and is ready to get into the one forced by the neighbor. The clock keeps the potential barrier low;
- **Switch**: the cell is starting to get into a defined state, influenced by the neighboring cell; the clock is being released, so that the energy barrier is raising and the state of the switching cell is becoming less easy to influence;
- **Hold**: the potential barrier is high, the cell stays in a well-defined state and can be used to set the state of the next one. The lowering-barrier feature of the clock is now turned off on this cell;
- **Release**: the clock is lowering the energy barrier, and the cell is loosing its state.

This way, the currently switching cell always stays in between a cell with a strong influencing ability (the one in the *Hold* state) and one with almost no influencing ability (the one in the *Relax* state). This avoids backdriving and sets the direction of propagation of the wire. Hence, the circuit is made of several "strips", and in every one of these strips the clock is in a certain phase: these strips are also called "clock zones". Some variations to this scheme could be found, for example, the clock could be made of just three phases, provided they overlap for a certain portion [2]. The way QCA circuits are clocked and powered makes them a sort of highly pipelined circuit, where each signal undergoes a delay equal to the number of clock zones it travels through. This is also known as the characteristic (or problem) "Layout=Timing".

1.3 Nanomagnetic QCA

One of the possible implementations of QCA is the nanomagnetic one. It leverages nanomagnets, which are magnets whose size is about tens or hundreds of nanometers. When the magnet is so small, it only contains one magnetic domain; this makes its magnetic field well-defined. Moreover, in these nanomagnets the magnetic field is very likely to have just two, opposite directions. This property is called "anisotropy", and will be further elaborated in chapter 7. For the time being, let us just say that it is caused by the elongated shape of the nanomagnets, and that the axis where these two preferred directions lie is called "easy axis", whereas the orthogonal axis is called "hard axis". These two features make the magnet particularly suited to represent the two digital states. If the easy axis lies in the same plane the nanomagnet is lying, the technology is called "in plane Nano Magnetic Logic" (iNML); if the easy axis lies on a plane perpendicular to the one where the nanomagnet lies, the technology is called "perpendicular Nano Magnetic Logic" (pNML).

1.3.1 in plane Nano Magnetic Logic

In figure 1.8 two lines of magnets, one horizontal and one vertical, are shown. Notice

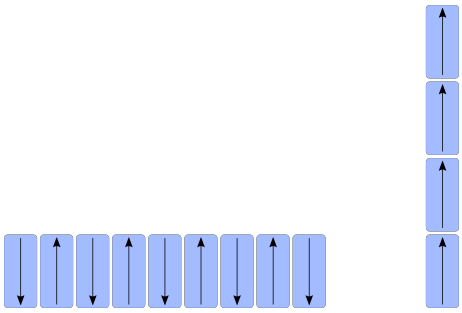


Figure 1.8: iNML wires

that when magnets are lined up in the same direction as the one of their magnetization vector (the vertical line of the figure), each magnet forces into the next one its own state. On the other hand, if the nanomagnets are lined up in a direction orthogonal to the one of their magnetization vector, each nanomagnet forces into the next one the opposite of its own state. The two possible coupling between magnets are called respectively "ferromagnetical" (F) and "antiferromagnetical" (AF).

A line of antiferromagnetically coupled nanomagnet is also the way the inverter is made: a line of an odd number of nanomagnets will always hold opposite values on its ends. The majority voter is still the main logic gate, and is also made in the same way seen before. A nanomagnet is influenced by three other nanomagnets, as in figure 1.9. The only difference is that in this case two inputs are coupled one way

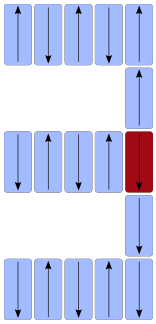


Figure 1.9: iNML majority voter

(the top and bottom ones, F coupled), and the remaining one the other way (the middle one, AF coupled). Of course, inverting the signals as needed is all what is needed to deal with this new characteristic of the device.

Next thing to describe is how the circuit is clocked. There still are clock zones, but the clock phases could be just three (some conditions are to be met), as in here. The nanomagnes are brought to a "Null" state by means of a magnetic field that causes the direction of the magnetization to be parallel to the hard axis. As soon as this clocking magnetic field is removed, the magnetization direction moves back to the hard axis, in the direction forced by the neighboring nanomagnets. Clock zones still exist, and the easiest way to technologically implement them is by means of wires running below the planes of the nanomagnets [2],[3]. Figure 1.10 shows how a nanomagnetic wire evolves across the three clock zones. The three phases are: *Hold*, *Switch* and *Reset*, and appear, from left to right, in this same order in the figure. The nanomagnets in the switch zone are orienting their magnetization vectors according to the influence of the last nanomagnet in the hold zone, without the nanomagnets on the right, in reset zone, able to interfere. The principle is the same as the one seen for generic QCA. The blue arrow shows the direction of the clock field. Another possible way to reset the nanomagnets leverages piezoelectricity and absorbs much less power [4], but a detailed description is beyond the scope of this thesis. There is

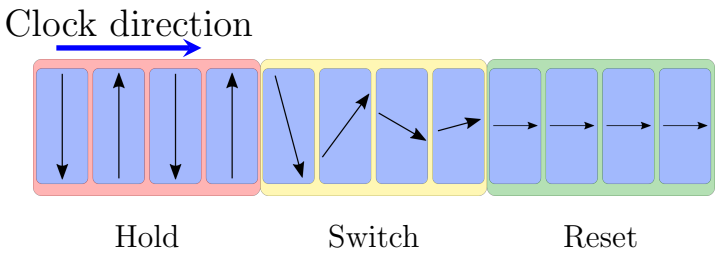
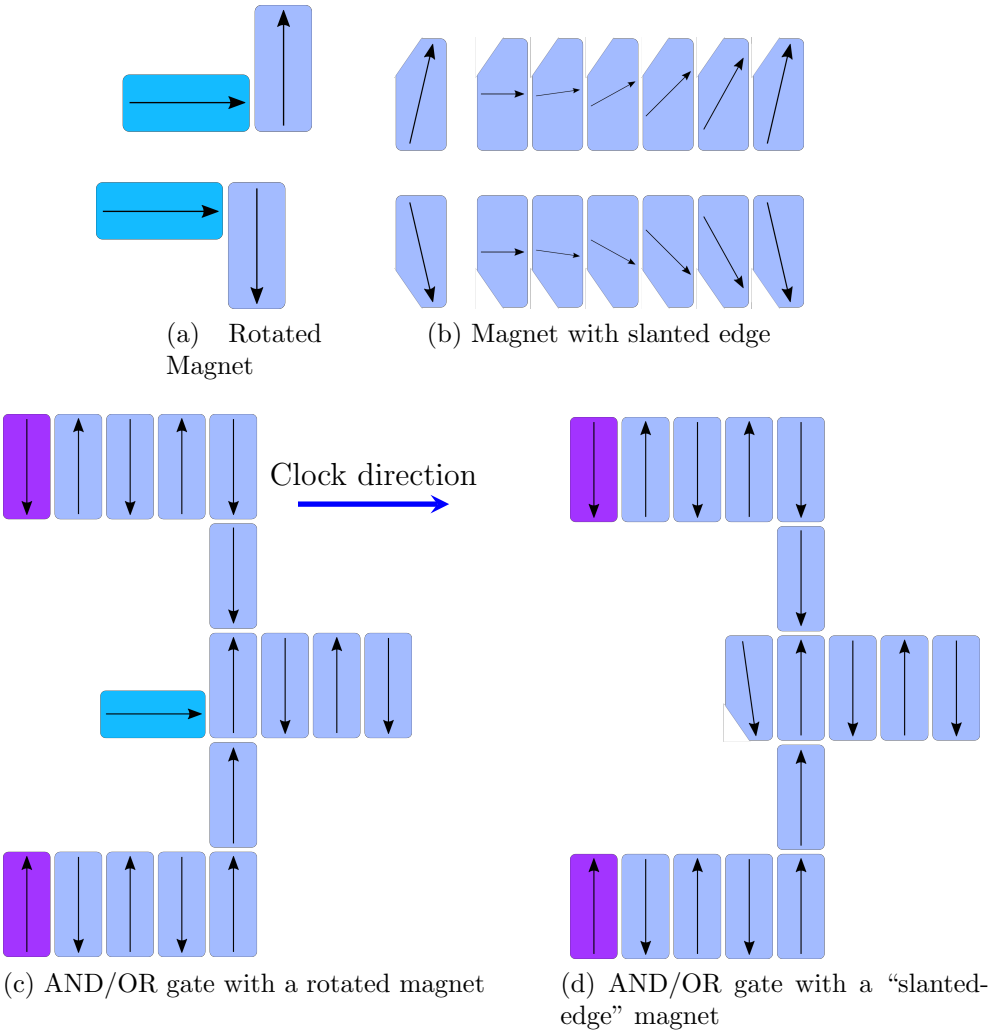


Figure 1.10: Clock phases on a iNML wire

nothing preventing a nanomagnet from being placed above another one, rather than besides: this is a multi layer technology. In order to produce one of the known gates (AND, OR, NAND, NOR) with a majority voter, one of the inputs should hold a fixed logical value. In iNML, this can be easily done: a possible solution consists in placing a nanomagnet rotated by 90 degrees. Given the clocking field direction (figure 1.10), the clock will force a magnetic field on it along the easy axis. When the clocking field is removed, the magnet will stay in this magnetic configuration. By placing it close to either the top or the bottom of the nanomagnets it can be set whether if forces a 1 or a 0 on the nanomagnet. Another way employs nanomagnets with "slanted" edges [5]. The slant produces a slight rotation of the easy axis, so that when the clocking field is removed, the magnetization vector will be much more likely to turn to one side rather than to the other. Figure 1.11 shows what these solutions look like. Figure 1.11a shows how a rotated nanomagnet can serve as "fixed-input"; figure 1.11b shows the rotation of the easy axis, and how consequently the magnetization vector rearranges when the clock field is removed; figures 1.11c and 1.11d show example of the application of the two devices two build a AND/OR gate with a majority voter. The clock direction is the one indicated by the blue arrow.



(a) Rotated Magnet

(b) Magnet with slanted edge

(c) AND/OR gate with a rotated magnet

(d) AND/OR gate with a "slanted-edge" magnet

Figure 1.11: Examples of AND/OR nanomagnetic logic

1.3.2 out of plane Nano Magnetic Logic

The preferred anisotropic direction in a nanomagnet might be also be out of the plane the nanomagnet is placed on. In this case the technology is called "perpendicular Nano Magnetic Logic" (pNML). Usually, this property is achieved by manufacturing the nanomagnets as a stack of two different, alternating, materials, and making them very thin. This difference has a lot of consequences, that make pNML technology quite different from all the rest of NML. First of all, the nanomagnets can have whatever shape. Second, clock zones are not needed anymore. The reason is a little

elaborated, and next subsection is devoted to its illustration.

Figure 1.12 shows some of the pNML nanomagnets. In the picture they are about squared, but they can have any shape.

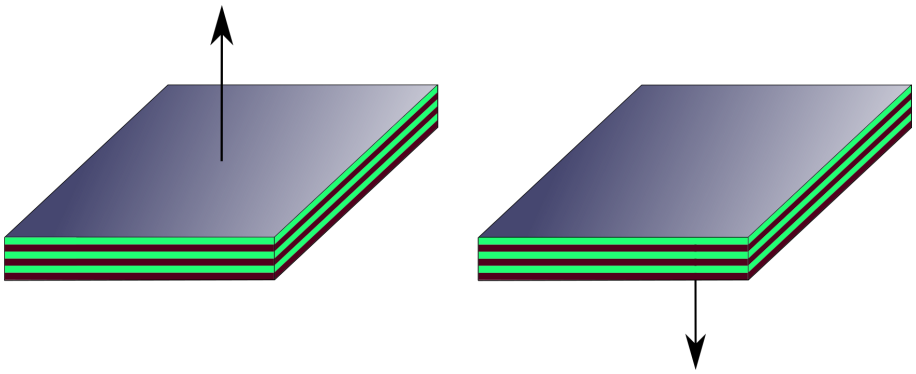
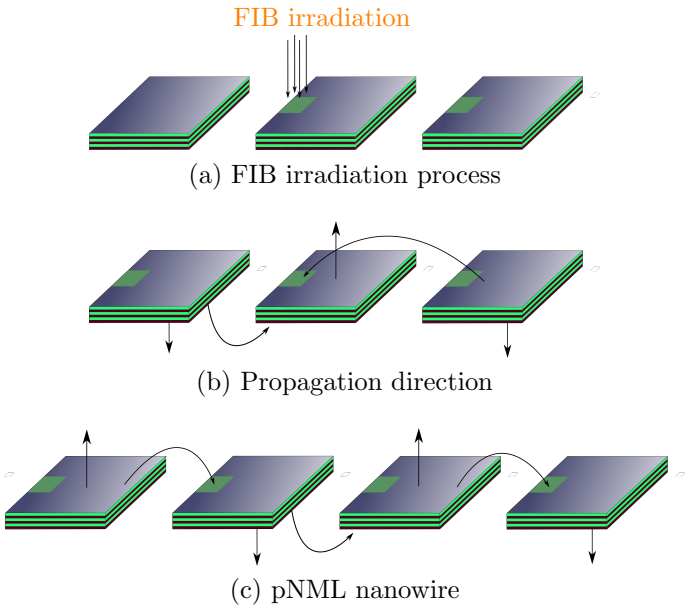


Figure 1.12: pNML nanomagnets in the two possible magnetization states

Local Anisotropy Reduction

In general terms, the way a logical value is propagated through a series of lined up pNML nanomagnets is the same as any other NML technology: the magnetization direction of a nanomagnet influences the magnetization direction of the next one, with the help of a clock. Since in a line of regularly spaced nanomagnets every one of them would be influenced by the one on its right as much as it would be by the one on its left, an appropriate timing of the clocking field is necessary to set the direction of propagation of the information. This last point is exactly what changes in pNML technologies. As mentioned above, the perpendicular anisotropy feature is achieved by manufacturing the nanomagnets as a stack of two different materials. If, in one spot of the nanomagnets, this order is somehow upset or destroyed, that same spot will show a more limited anisotropy. The spots with the lowest anisotropy in a magnet happen to be the ones more sensitive to the influence of neighbors' fringing fields. Therefore, it is much easier forcing a particular direction of the magnetic field within that area. This anisotropy reduction is made by means of a Focused Ion Beam irradiation (FIB), that mixes up the molecules of the two materials the nanomagnets is made of [6]. The spot with modified structure is called Artificial Nucleation Center (ANC). All these concepts are shown in figure 1.13. The figure



(a) FIB irradiation process

(b) Propagation direction

(c) pNML nanowire

Figure 1.13: pNML wire manufacturing

shows also why the propagation direction is set: if the nanomagnets is irradiated on the lefthand side, the ANC will be much closer to the nanomagnet on its left. With a careful study of the technological and geometrical parameters, the ANC can be made very sensitive to the magnetic field of the left hand nanomagnets, and almost insensitive to the on the right side, as shown in row b of figure 1.13. This is what sets the propagation directions. "Nucleating" an ANC means that a magnetic field direction has been forced into the ANC itself. The expression "the ANC nucleates itself" can be found too. Notice that in a line of coplanar pNML nanomagnets, their coupling is antiferromagnetical.

Gates in pNML

The universal gate in pNML is the majority voter. It can be made by a nanomagnet whose ANC is influenced by more than one other nanomagnet (but always an odd number), as it is draw in figure 1.14. The picture also shows an important design feature of pNML: it is a 3D technology, with nanomagnets that can be placed one above the other.

The three input nanomagnets are draw in a different color with respect to the

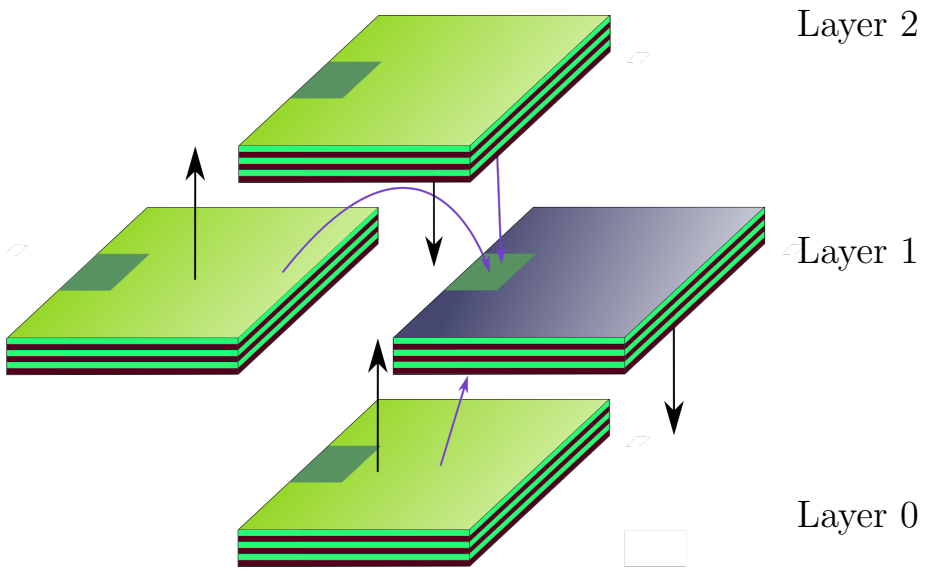


Figure 1.14: 3D pNML majority voter

output nanomagnet. Notice also that the two nanomagnets above and below the output one are ferromagnetically coupled, and the output nanomagnet behaves accordingly. The coupling between two nanomagnet in different layer is called a "via", because of the functional similarity with the CMOS element bearing the same name.

In place of one of the input nanomagnets in figure 1.14 there could be a fixed-values nanomagnet, which of course are available for pNML too. Unlike iNML, though, usually just one logical value (that is, magnetic direction) is available for them, rather than two. This could seem a massive hindrance, but it will be discussed at great length, and it will turn out that it is not a big issue.

pNML Clocking

pNML circuits still need a clock to propagate signals, because the magnetic field generated by the nanomagnets is not strong enough to nucleate the ANC. The clock is also a magnetic field, that is applied over the whole circuit surface with no clock zone or any other spatial distinction. Its direction is the same as the magnetization vector of the nanomagnets, that is, perpendicular to the plane where the magnets are placed, and alternates between two opposite values. In order to understand better, it is very helpful thinking that the clock alternates between logical value 1

and logical value 0. The intensity of the clock field is such that it cannot nucleate an ANC if the nanomagnet on the left of that ANC does not have a field that supports, rather than oppose, the nucleation of that value. In other words, an ANC nucleates with a magnetic field representing the value 1 only if the clock is currently at the "1" value, and the nanomagnet on the left side is holding a "0" value (remember, the coupling is antiferromagnetical, so each nanomagnet forces its opposite value into the next ANC). The propagation of zeros and ones is out-of-phase by 180 degrees, as it is shown in figure 1.15. The first row represents a steady situation, where all the

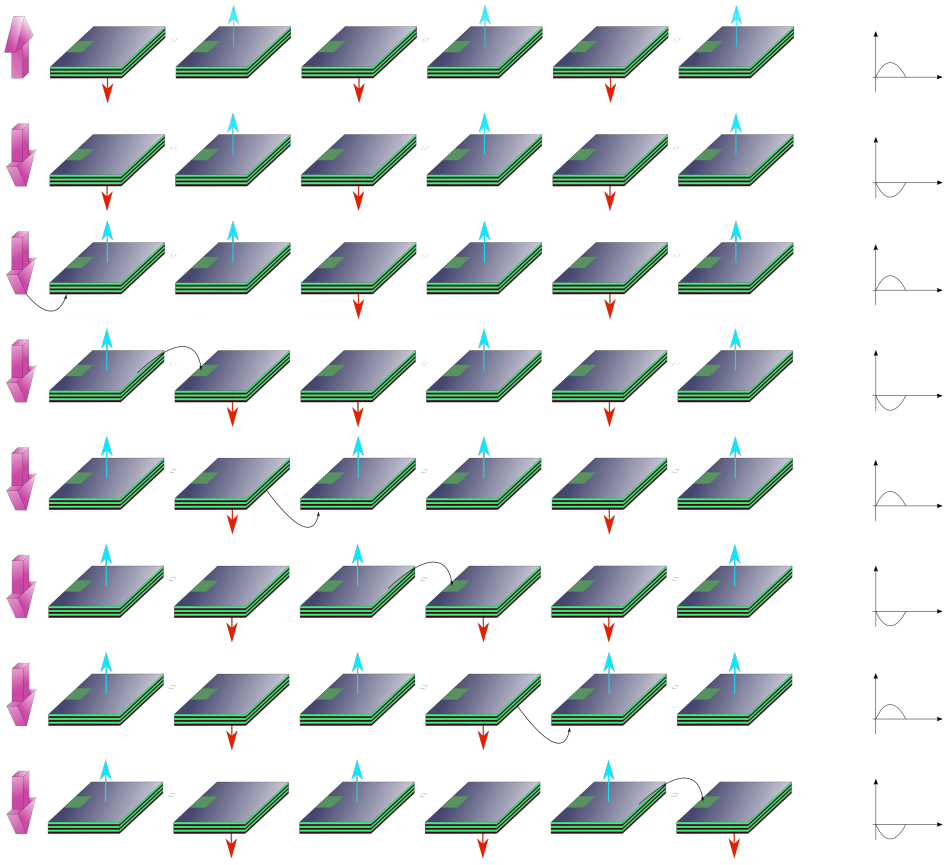


Figure 1.15: Example of signal propagation in pNML

nanomagnets are already updated and coherent. If the input on the left (big, pink arrow) does not change, the line of nanomagnets will never change either. In the second row the input changes indeed, and the remaining rows show the evolution,

pointing out which nanomagnet is being updated.

According to this description, a wire in pNML is made in pretty much the same way it is done in iNML: a line of basic nanomagnets is built. Several clock cycles are needed for a token to travel all the way through the wire, and this could jeopardize the performances and be challenging for the designer to handle. An alternative is available thanks to the ability of pNML nanomagnets to retain the basic characteristic so far listed regardless of the shape they have, that allows to draw long nanomagnets which are in fact wires. To understand how a long wire behaves, it is necessary to introduce the concept of domain wall.

Domain Walls

A domain wall is the border between two adjacent domains within a nanomagnet whose magnetization is different. The subject is very complicated; here only the concepts useful to understand their use along with the pNML technology will be addressed. The domain wall can be "shifted" towards a certain direction by means of an appropriate magnetic field, and the usual way to work with them consists in manufacturing long nanomagnets which contain some domain walls, and then analyzing their movement along the wires. What happens in a pNML nanomagnet is that the magnetization vector is changed in the ANC area, and then is propagated towards the rest of the nanomagnet in the way described above. Figure 1.16 shows how the domain wall moves along a nanomagnet. Domain walls can also be used to build digital circuits: bends, branches and tips within the domain wall path, together with the exposure to a magnetic field with varying direction and intensity, provide logical functions such as AND, OR and NOT ([7]). The most relevant point is that domain walls do not propagate in the high-pipelined fashion described for all the other QCA technologies, but rather as signals in a combinational network. This makes them really attractive to reduce the delay, in terms of clock cycles, needed for signals to run through long wires. Basically, the computation could be carried out in the usual way, whereas the long rows of nanomagnets, with a high latency, are changed with domain walls. The clock period will increase, because what causes the domain wall movement is the clock field, but the wire will deliver its values in just one clock cycle. Thus, three main elements make up this new variation

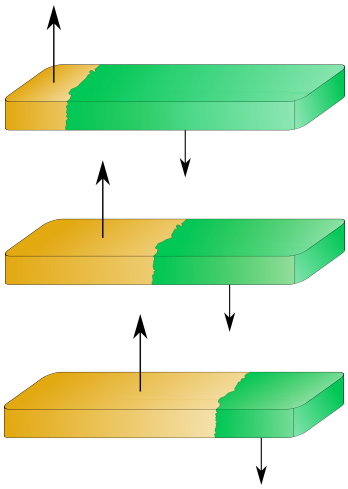
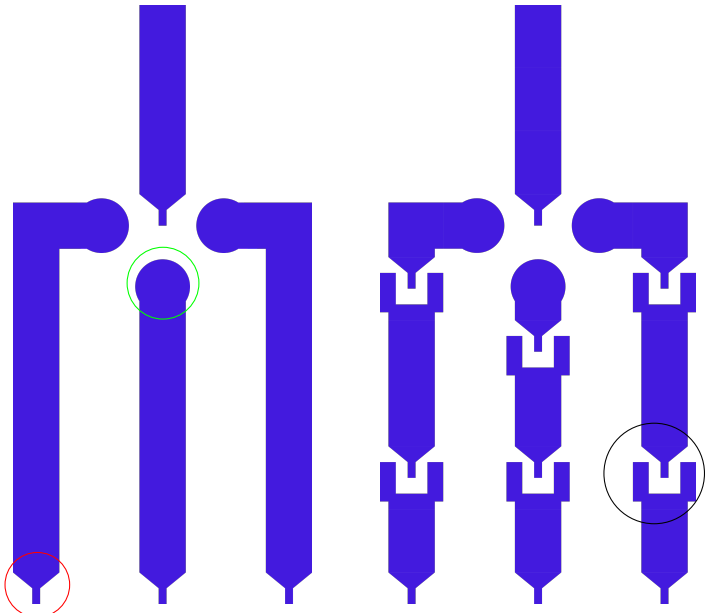


Figure 1.16: Domain wall propagation along a nanomagnet

to the NML technology: nucleation centers, stretched nanomagnets through which a domain wall propagates (really often called just "domain wall": the two terms are interchangeable in practical use, and it is almost impossible to confuse the two meanings. Other words could be "wire" or "nanowire"), and the notch, that will be addressed later on. Of course, these are not separated items, as ANC and notches are both adjustments made on the stretched nanomagnet. This design approach, that combines domain walls and standard pNML coupling, is called Domain Magnet Logic (DML,[8]). Figure 1.17 shows how circuits can be made. The figure presents two alternatives of the same circuit, a majority voter, with the inputs on the bottom and the output on the top. The elements like the one in the red circle are nucleation centers, while in the green circle there is a "Pad", that is just the end of a domain wall, but it is drawn that way to make clear that it is influencing an ANC. The black circle encloses a domain wall antiferromagnetically coupled with an ANC: in practice, an inverter. There is no functional difference between that element and the configuration of two adjacent nanomagnets in figure 1.13, bottom row; however, the actual structure is different, and the configuration inside the circle is specifically designed to increase the coupling between the two nanomagnets. This configuration is usually considered an independent design element, called, of course, "inverter". The key difference between the circuits is that in the 1.17a, long nanowires run from the input ANC to the one of the majority voter, whereas in the 1.17b those



(a) Non pipelined (b) Pipelined

Figure 1.17: Alternative circuit designs with different maximum wire length

nanowires have been considered too long, and the clock period would have been excessive. Hence, it has been divided in sections with a couple of inverters: this way, the signal takes one more clock cycle to reach the majority voter ANC, but the clock period is lower. In other words, the 1.17b is a pipelined version.

Notches

The last feature of pNML technology, very important, is the notch. A domain wall moving along a nanomagnet is sensitive to the geometry of it: this is how domain wall logic is build. Some "deformations" of the nanomagnet cause the domain wall to be trapped in it, and it will not move any farther, unless a stronger magnetic field is applied (different from the usual clocking magnetic field). The expression "pinning a domain wall" is the one used to describe this phenomenon. The magnetic field could be applied to the whole circuit, causing all the notches of the circuit to "open", or "release" ("depin the domain wall")the value the have in input (meaning that the domain wall is again allowed to run through the nanomagnet), or just locally applied by means of other nanomagnets close to the notch [9]. This feature is really useful to implement memory elements with no internal loop, latching devices and

even a pNML equivalent of the CMOS pass-transistor. Figure 1.18 shows how the propagation works. As drawn there, the domain wall moves along the nanomagnet,

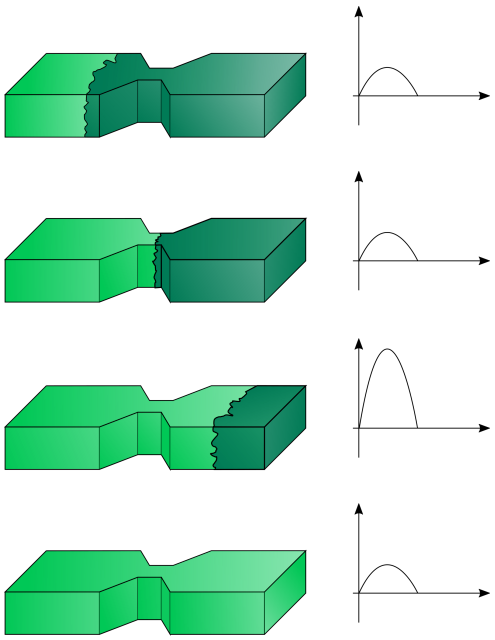


Figure 1.18: Pinning and depinning of a domain wall in a notch

but as soon as it reaches the shrunk section, it gets stuck. A particular value of the magnetic field is able to release it, and once the domain wall gets past the notch, it can propagate with the usual clock values. Notches can be used in many different ways: in all the circuits of this thesis work, they will be opened on a regular basis by the proper clock field, as it is done for registers in CMOS. Thus, a "notch period" can be defined.

1.3.3 Clocking Terminology

The description of the previous sections should have made clear that in pNML there are multiple "levels" of clock. These are:

- The clock that causes the nucleation of ANCs and the propagtion of domain walls;
- The clock that allows domain walls to get past the notches;

Therefore, there are two kinds of clock periods:

- The one that considers the amount of time needed for a domain wall to propagate through a single nanowire, expressed in seconds. This is usually referred to as "Clock Period";
- The one that considers the number of clock cycles (as defined in the previous point) needed for a signal to propagate from the output of a notch in the circuit to the input of another notch. It is expressed in number of clock cycles, or sometimes half clock cycles, and it is usually referred to as "Clock Period in terms of clock cycles", or "Notch period";

If the circuit does not contain notches, the second kind of clock period is not defined. This description shows how a signal is always considered to need an integer number of half clock cycles to propagate from a point to the other: if the absolute time is needed, it could be obtained by multiplying the clock period by the number of clock periods needed to propagate. Hence, a general delay too might be expressed either in terms of clock cycles or in terms of absolute time. The first approach is by far the most frequent. Comparisons with the CMOS circuits, that just have one clock, will be frequent, so it is absolutely worth it solving possible ambiguities. Since a signal delay is usually considered in terms of half clock cycles, in general a circuit without notches is mentally associated with a CMOS combinational circuit. On the other hand, a pNML notch will be associated with a CMOS register. Next chapters will show a pNML circuit descripted in CMOS: the pNML logical paths become combinational paths in CMOS, and the notches becomes registers. The similarity is so deep that this conversion could even be made automatically (with some workarounds). Therefore, really often (at least in this thesis work) the pNML delay in terms of clock cycles of a signal is thought as the counterpart of the CMOS combinational delay.

Last, the concept of "pipelining" will be used at length. If not otherwise stated, this expression refers to the insertion of pairs of inverters along very long domain walls, in order to reduce the clock period. The expression used to insert notches, the same ways as registers are inserted in CMOS, will be "notch insertion". The context will hopefully reduce the ambiguity in the use of these words.

Chapter 2

Delay Analysis

In this chapter, the main characteristics of the timing and signal propagation are addressed, with particular regard to the traits that might turn into troubles and to the ways to deal with them.

2.1 Basic propagation

pNML has the peculiar feature that logical zeros and logical ones do not propagate at the same time, but rather one after the other by half cycle based steps. Figure 2.1 shows how this happens, using an inverter as example.

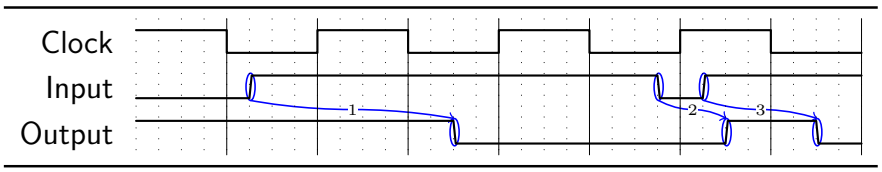


Figure 2.1: Example of propagation through an inverter

Notice, there are three changes of the input signal, numbered within the figure, but the delay is always different. In the first transition, the input signal goes from 0 to 1; the next clock edge is a positive one, and it does not propagate zeros. Thus, the propagation occurs after the next, negative, clock edge. Two clock edges stay in between input and output transition. In the second transition, the input goes back to zero, just before a negative clock edge, that propagates the output: this time,

the output is ready after a single clock edge. The third transition still shows a half clock cycle delay. A few things can be observed:

- The delay depends on the moment in which the transition occurs: it might happen either in the "good half cycle" (sign of the transition and current phase of the clock match), as it does in case 2 and 3, or in the "wrong half cycle" (sign of the signal transition and current phase of the clock do not match). If the input is free to change at any moment, predicting the exact delay entity is not possible (there will always be an half clock cycle of slack);

- Transitions 2 and 3 show the same delay in terms of clock cycles, even though the absolute time is a little different. The reason is that input transition 2 occurs just before the clock edge, whereas transition 3 occurs long before the clock edge. Actually, this happens in CMOS technology too, and is due to the edge-triggered nature of the propagation;

- As already said, in case 2 the input changes very close in time to the clock edge. The change may not get arbitrarily close, since a setup time condition must be met;

- On the other hand, the output transition occurs always with the same delay with respect to the clock edge (apart from tolerances) that causes it (no matter what is its sign). This delay can be considered the propagation delay;

- All these considerations about the sign of the clock edge that propagates the input to the output are so simple because there is only one input, so that the output depends only on a single signal. Gates with more than one input, of course, cannot be left out of the analysis, and must be handled in a slightly more complicated way.

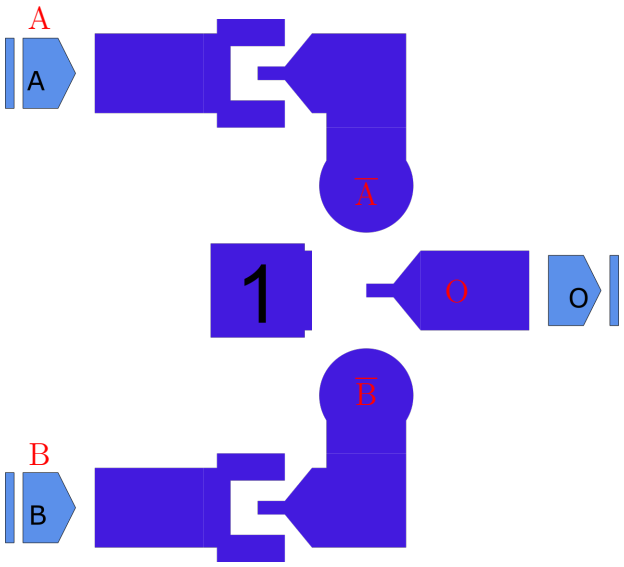


Figure 2.2: AND gate in planar pNML technology

2.2 Propagation through a logic gate and glitches

Now, let us consider a simple two-input circuit, such as an AND. The pNML, one-layer circuit, is shown in figure 2.2; keep in mind that this is just a case study, a circuit like that is not even feasible.

The delay depends on both inputs: for example, if B is zero, and A rises from zero to one, the output will stay unchanged. This case, however, is very trivial. Considering the inputs one at once should not be too difficult. A more elaborated problem might be the following one:

Suppose that A=1 and B=0. At the same time, $A \to 0$ and $B \to 1$. What will happen? The steady output will still be zero, but will there be a glitch?

To answer the question, let us draw the timing diagram in figure 2.3.

No glitch occurs: even though there is a time span where both pads (the rounded spots near to the nucleation center, bearing the $\overline{\text{A}}$ and $\overline{\text{B}}$ labels) are high, and a clock edge occurs within that time span, that edge is a negative one. Thus, no logical one can be propagated. However, this is just a lucky case. Consider now figure 2.4: in this case, a glitch shows up, with the output going to one even if the two input values are never both one at the same time. Of course, this is due to the out-of-phase propagation between zeros and one: in this particular case, the $0 \to 1$ transition

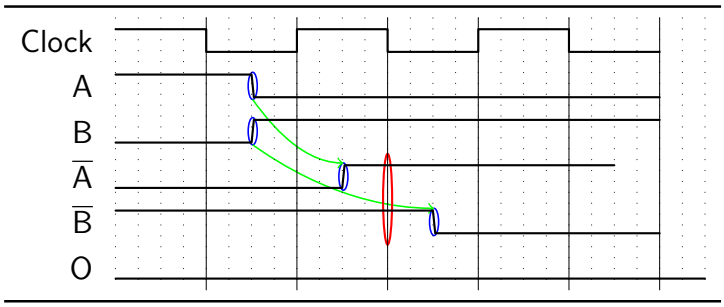


Figure 2.3: Glitch case study for an AND gate, no glitch occurring

is faster than the $1 \rightarrow 0$ transition in propagating towards the output, because the next clock edge is a negative one. It is easy to find a case (or a logical gate) where the opposite phenomenon takes place.

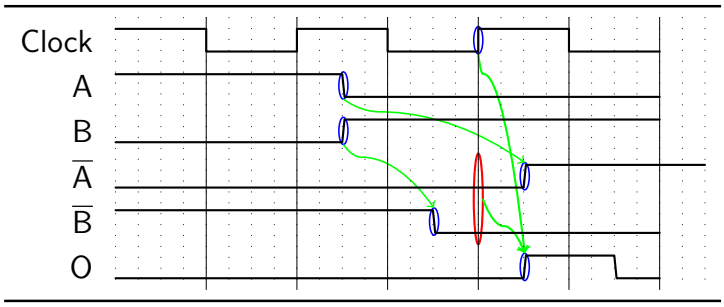


Figure 2.4: Glitch case study for an AND gate, glitch occurring

Glitch occurrences were not uncommon at all in CMOS technology. However, in pNML, because of the clock-based propagation, a glitch lasts for an integer number of half clock cycles, that is at least half clock cycle. Unfortunately, half clock cycle might be the length of a "true" signal, where "true" means here that a value is the correct outcome of the input configuration, not due to glitches. Let us have a look at figure 2.5.

Input signals last for exactly one clock cycle, as it happens in regular CMOS circuits, where a clock cycle is usually enough. It would seem a perfectly sensible choice for pNML as well, inasmuch as a full clock cycle must contain both clock transitions and both logical values possibly present are sampled. They are indeed, producing the outcome on the output, which has exactly the same length as the glitch in figure 2.4.

The problem of telling apart steady values from glitches arises. A different way

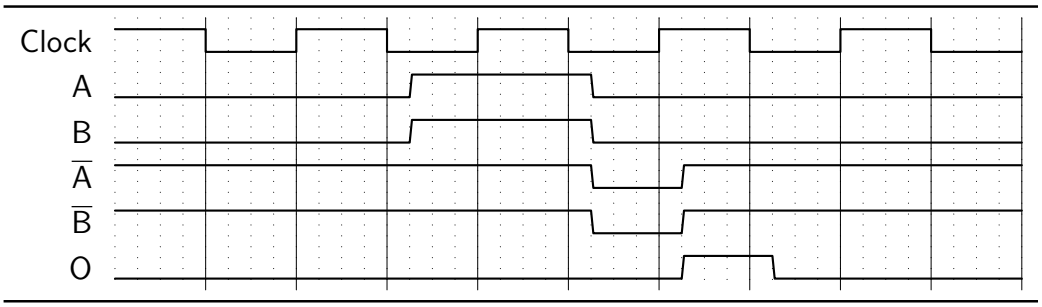


Figure 2.5: Delay case study for and AND gate, output length = 1 half clock cycle

to express the problem might be: *how can be found the time instant where the output value is the correct value, rather than just a glitch manifestation?* Once the answer to the problem is found, output values can simply be sampled in that moment. Moreover, the solution to that problem also provides the delay value for a logic gate, which is referred to the time instant where the output is free from glitches.

Before moving forward, it might be interesting looking the same input configuration seen in 2.5, but with an half clock cycle shift. Figure 2.6 represents this exact scenario.

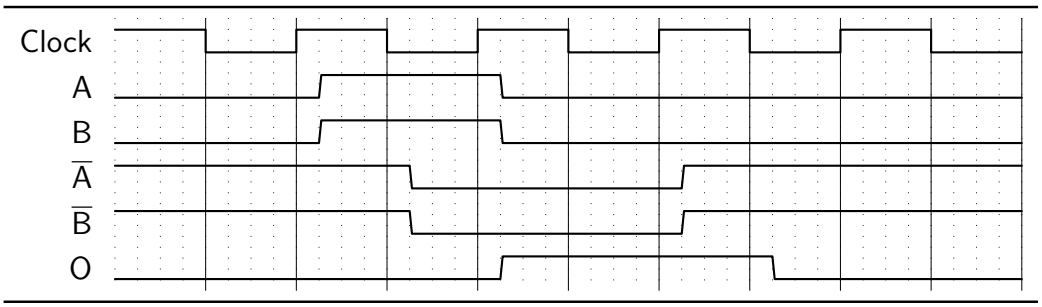


Figure 2.6: Delay case study for and AND gate, output length = 3 half clock cycle

It turns out that the time length of the output signals depends also on the relative order of the two clock transitions within the clock cycle. Bear in mind that there are cases where the inputs are under control, so that, if need be, they might be generated right before the most useful clock edge (though it enormously complicates the control), and cases where the inputs are actually outputs of another network, and there is not a direct control over them. A signal driven by the clock (as any signal within the network is, besides, possibly, the inputs) retains the same value for an odd number of half clock cycles. If, for sake of simplicity, inputs are kept

stable for an integer number of clock cycles, as they are in this example, outputs and inputs will always have a different duration.

2.3 How to deal with glitches in a simple manner

So, all the previous problems, namely, finding the correct time to sample, telling apart glitches and steady values, and finding the delay of a gate, are all different expressions of the very same problem. A really simple solution, used when just the functionality of the circuits is tested, without a fine timing characterization, can be seen in figure 2.7.

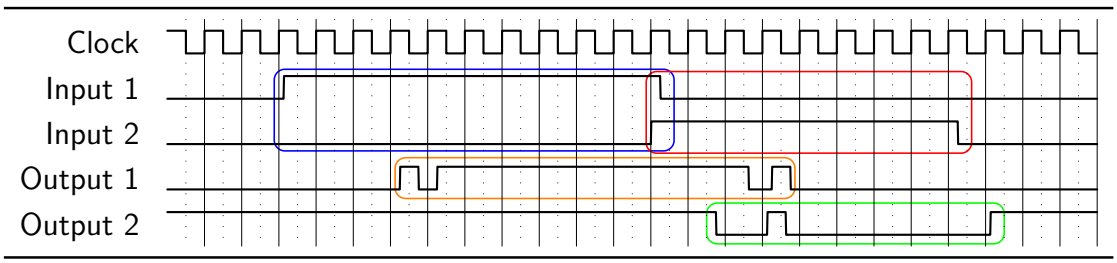


Figure 2.7: Example of the "easy" method to handle pNML delays

In order to understand properly how this works, the context must be somehow outlined. Suppose an arbitrary network with two inputs and two outputs, where each input represents a condition to be checked, and whose outputs are driven this way:

- Output 1 is always low, except when input 1 is a logical one and input 2 is a logical zero;
- Output 2 is always high, except when input 1 is a logical zero and input 2 is a logical one.

Hence, the easiest test to perform consists in checking whether output 1 issues a logical one or output 2 issues a logical zero, because only one out of four possible input configurations produce that outcome. For the time being, let us focus on the former. Output 1 is expected to be always zero, except when the input configuration is 1-0. If the inputs are set for long enough in that combination, the output

should get high, which is exactly what happens in figure 2.7. Of course, this is a simplification, the other input configurations are ignored here: the point here is just proving the concept. In some real cases, one input/output configuration could be more interesting or meaningful than the other (an AND checking whether a set of conditions is met), in others, all the combinations could be equally important (a counter).

The idea is this: when the condition for input 1 or input 2 to get high is met, they are kept high for a long time, several clock cycles. This produces in output a burst long enough, much more than the time span where glitch might occur. In figure 2.7, the input configuration in the blue box causes the output burst in orange box, and the input configuration in red box causes the output burst in green box. This way, the time where the output cannot show glitches is much longer than the time where glitches may occur. All it is left to do is sampling the outputs in these "glitch-free" time periods. To sum up, the method boils down to this: set an input configuration for long enough until the expected output shows up, possibly after some glitches, and keep the inputs steady so that the outputs are steady too for a suitable amount of time. This method mirrors what happened in CMOS technology: there were glitches in CMOS too, but the clock period was long enough so that they only took up a small fraction of the clock cycle. The only difference is this one: since in pNML all the evolution is clock-based, it comes out that glitches are latched as well as any other signal is. This is also why they last as long as the correct outcomes do. Clock cycles cannot be stretched to wait for glitches to die out, because only the clock transitions cause the circuit the evolve. And, given that the glitches are due to the input transitions, it follows that the clock cycles where inputs change must be much fewer than the clock cycles where inputs are steady. This is just another way to express the same thing.

This method has two drawbacks, that make it in fact only feasible for simple cases or study cases rather than an actual solution. The first one is that determining the burst-length of the inputs is not easy for an arbitrary network: an AND gate is made up with a handful of gates; it is easy to guess that roughly after five clock cycles all the glitches are gone. A more complicated network might not be that easy to analyze. The second drawback is that this method is essentially based on oversizing, causing a performance fall.

A way to sum up all these concepts is this one:

- In pNML, it happens that some clock cycles present input transitions, some other do not. This is quite obvious, but a there is something to point out: in CMOS, there is nothing preventing inputs from changing at every clock cycles, whereas a pNML circuit with an input changing rate so high would be hardly useful;

- Signal transitions cause glitches, exactly like CMOS technology; unlike CMOS technology, glitches are sampled and propagated by the same clock that propagates all the signal values;

- Thus, when looking at the output, it turns out that some glitches reached all the way to it (others might be filtered out by the logical function or by delay mismatches). It means that if a pNML gate is fed with an input configuration lasting N clock cycles, the output might last $N - m$ clock cycles, where m is a number of clock cycles where the output is affected by glitches. Hence, glitches must be distinguished from the correct output values. In this regard, an interesting fact is that in a correctly clocked CMOS circuit, glitches are always locally generated: they come from the transient of the local gate. In pNML, glitches are sampled by the clock and might propagated through many gates.

- In order to neutralize the effects of glitches, a simple solution consists in making $N \gg m$. Further delay knowledge is needed to define what exactly is "much greater", otherwise the throughput will be strongly affected.

In conclusion, a designer that just wants to test the logical functionality of his/her circuit could set the inputs to wait an appropriate number of clock cycles (usually an rough assessment is quick to do, or, in alternative, a very high number might be chosen), and, at simulation time, it will be very easy to check whether the circuit issues the expected outputs.

2.4 Delay balancing

The most important reason why understanding correctly the timing is crucial in pNML can be proved by the following example. Suppose to have a set of inputs and two networks. Each one of the two networks evaluates the inputs according to whatever logical function, and then, if some conditions are met, outputs a logical one. If the two networks are both high at the same time, some actions must be taken; therefore, the two outputs are AND-ed. The two networks have, of course, a different logical function, that means different delays, since in pNML the two things are strictly related. Thus, after having designed the two networks, it might happen something like what is shown in figure 2.8.

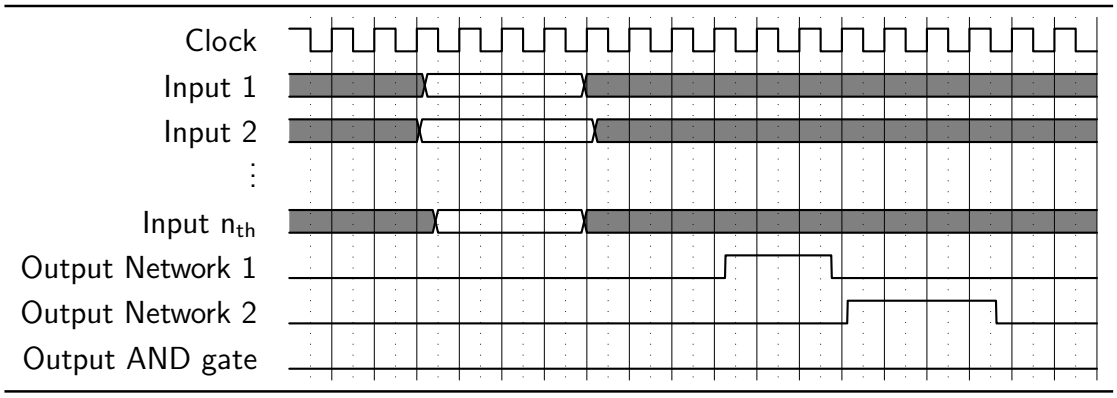


Figure 2.8: Example of time evolution for gates with strongly unbalanced delay. The network does not work

The two networks, having a different logical function, also exhibit a different delay: they outputs are dramatically out of phase, they do not overlap for a single time instant (remember, the inputs are the same for both) and the AND gate never outputs a one, inasmuch as it never has both inputs high. The solution is simple: network 1 will be "slowed down" by means of inverters insertion (actually, pair of inverters). Of course, to put in place the solution, the delays of the two networks must be know in great detail. Why the outputs are out of phase has already been said. Someone might still wonder however why they also have different time lengths (the first one outputs a logic one two clock cycles long, the second one is three cycles long). The reason can be showed resorting to this same example, slightly modified. Have a look at figure 2.9. What happens is more or less the same thing seen in

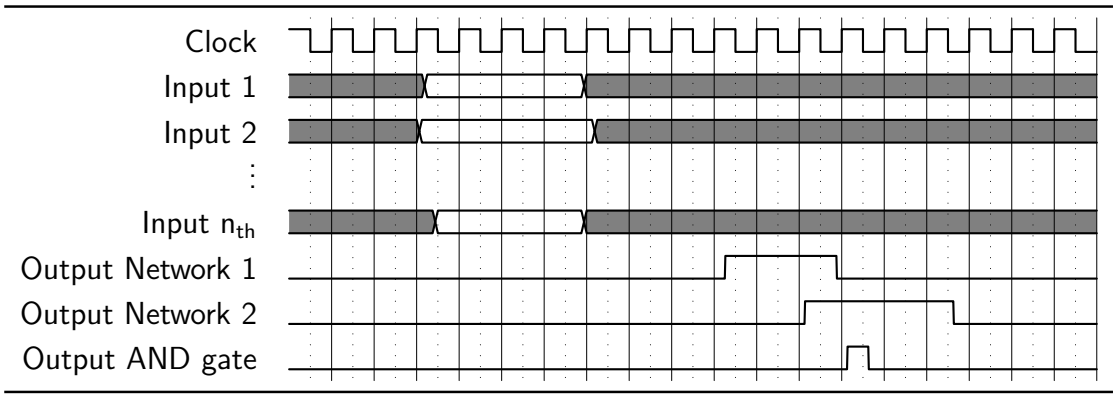


Figure 2.9: Example of time evolution for gates with unbalanced delay. The network works very badly

figure 2.8; nevertheless, the delays here are slightly less out of phase, and the AND gate manages to output a one for a very short time. The AND gate brings out a signal much shorter than supposed (basing on the inputs lengths, almost three clock cycles for both). So the reason why in figure 2.8 the outputs of the two networks stay high for different amounts of time is exactly this: probably inside network 1 a phenomenon alike to the one of figure 2.9 occurred, and, due to internal delays unbalancing, the output got shorter.

2.5 Delay characterization

So far, all the main singularities of pNML timing have been illustrated. The aim of that was proving how necessary a way to compute delays was needed, and in this chapter it will be expressed. Once the delays are know, it will be possible to know in which exact time instant (or, better said, clock edge) an input configuration will show up in output, steady and free from any glitch. Referring to the N and m parameters defined in page 27, it means knowing both the $N - m$ and the clock edge where those $N - m$ clock cycles begin. The rules to compute a delay are very simple:

Follow the path between the two points of interest (even with gates in between), consider half clock cycle delay for each antiferromagnetically coupled pair of domain walls, and a full clock cycle delay for each ferromagnetically coupled pair of domain walls.

Just to test the rule with an example, let us try it on a real circuit, for example, a multiplexer. A multiplexer (designed in [10]) is a good circuit to test the rule with, because is essentially made of an OR gate with inputs coming from two AND gates. The feature that makes it interesting is that the two AND share one of the three inputs, even though for one of the ANDs it is inverted. Figure 2.10 contains the circuit. Since the signal values will be checked all along the circuit, each piece of domain wall is labeled with a letter.

The two figures, 2.11 and 2.10, share the same color code, that works this way: in the time diagram some signal names on the left column are colored, it means they are each one the inversion of the previous (from top to bottom) and can be found in the circuit layout along an arrow of the same color. Then, there are triads of signal in the diagram, all with the same color, one of them in dashed line. The two solid-line signals are the inputs of a gate, whose output is the dashed-line signal. In the multiplexer layout figure this gate is marked with a circle whose color is the same as the three signals. The kind of gate is, however, left unspecified. Hopefully, these colors make easier reading the two figures.

Let us check some paths, for example, the one starting from the signal *Input 1* to the output (signal *Out*) of the multiplexer. Six antiferromagnetically coupled steps are found: B0, B1, B2, B3, D, Out. By looking at figure 2.11, it can be

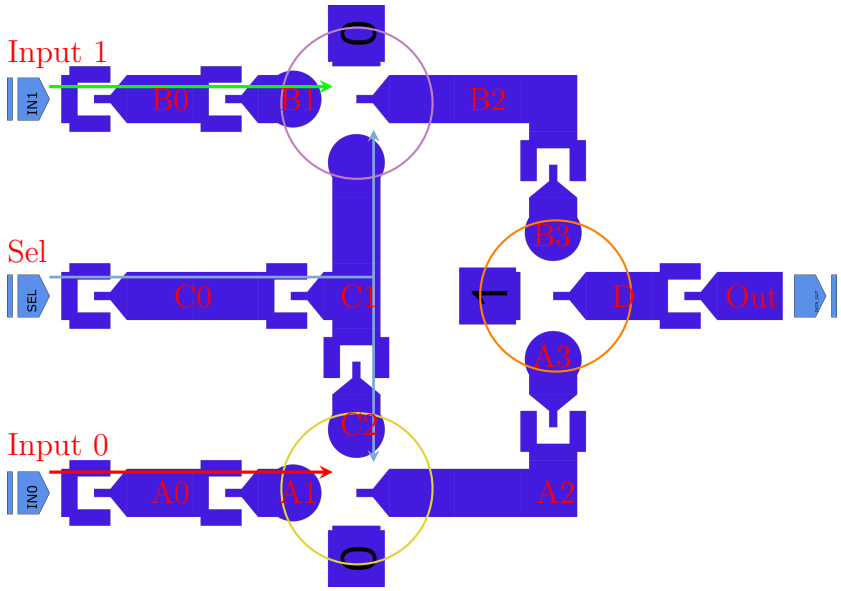


Figure 2.10: A multiplexer in planar pNML technology. Domain walls, inputs and outputs, logical gates and some inverters chains are marked

noted that *Input 1* changes in half-cycle 2 and in half-cycle 20. Since the first transition occurs when the multiplexer is open on the other side, it is much better considering the transition starting in half-cycle 20. By half-cycle 27, the output has changed. It makes 7 half cycles delay. Only 6 half cycles were predicted, and it is correct: the input transition occurs right before a negative clock transition, unable to sample the input value, so that it does not count. Notice that every inverting function has a delay made of an odd number of half cycles; on the contrary, in a non inverting function the delay is made of an even number of half clock cycles. Even though this is a pretty trivial consideration, it might come in handy to check correctness of layouts or delays counts. A short expression for delay values equal to an even number of half clock cycles that will be used, for convenience, in future, is "even delay", while the opposite case will be called "odd delay". As an example, notice that *Input 0* and *Input 1* must both have an even delay, because, when the multiplexer is open on their side, it copies their value in output. If, when assessing the delays of a network, this sort of rules are violated, it certainly means that the circuit carries out the wrong logical function.

Signal *Sel* is more interesting, because it branches in two different gates, whose outputs then converge in the same gate. In this case the delay to be considered as the

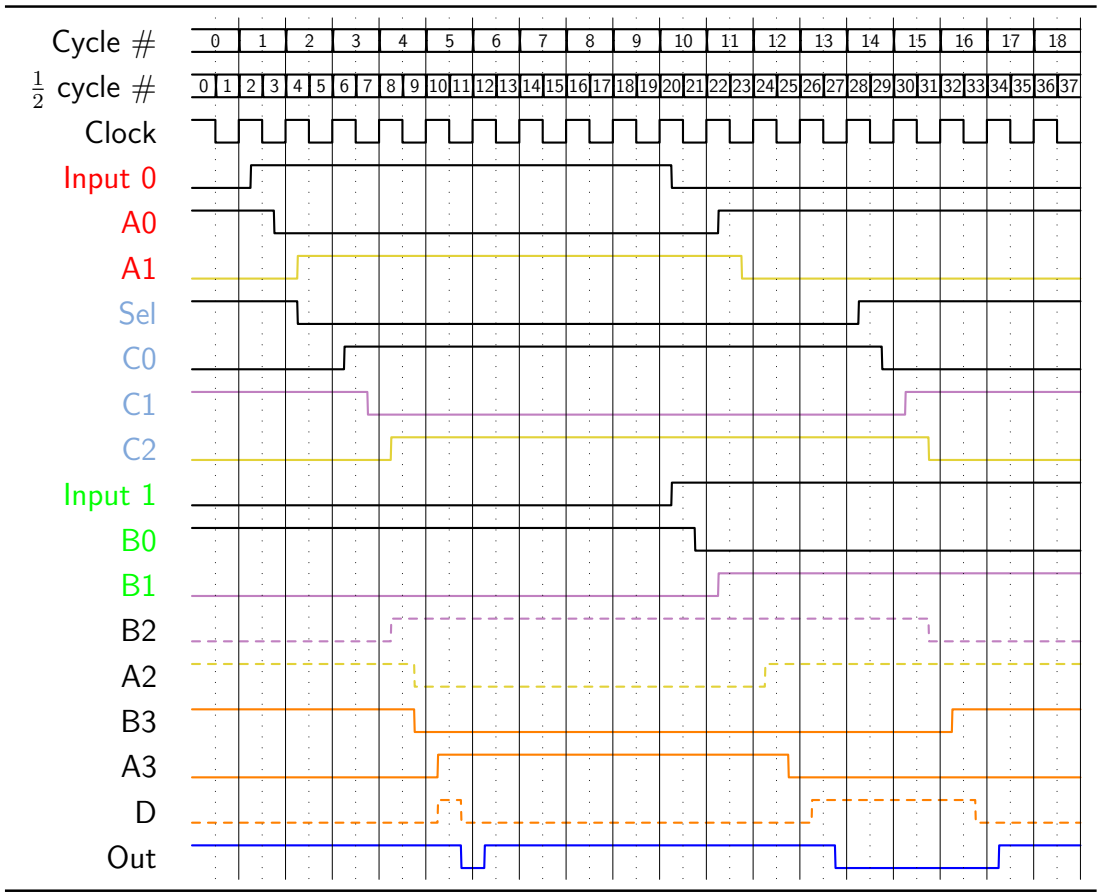


Figure 2.11: Example of possible time evolution of the circuit in figure 2.10

characteristic delay from point to point is the greatest among the two. Sometimes the delay is actually lower, and, being the whole process completely deterministic, it is possible to find out these cases, but it is not always trivial. Just to make things a little more clear, figure 2.12 shows an example with the same level of detail of the previous one.

Roughly speaking, in this case the longer of the two paths from signal *Sel* to signal *Out* is not sensitive to the signal change; therefore, the delay of the transition is the one of the other path. It was not difficult to spot this case, this circuit has only three gates whose function is very intuitive, but much more complicated cases exist. This is why it is much better to stick to the worst case delay rule.

To sum up, once it is known how to compute delays, it can be predicted the time when an input configuration will have reached the output without any glitch, how

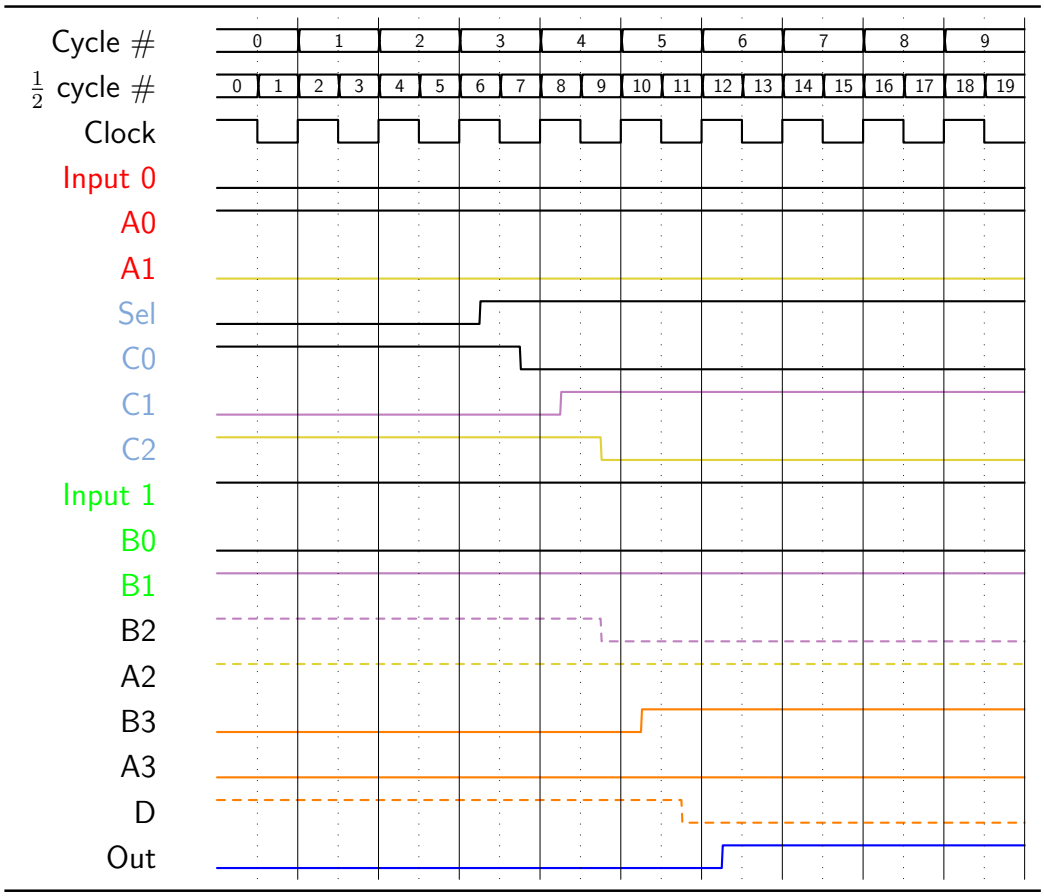


Figure 2.12: Example of possible time evolution of the circuit in figure 2.10

to balance networks with a different delay, and so on. The only missing concept is how to handle feedbacks, and it will be addressed in the following section.

2.6 Feedbacks

When in a circuit an output signal is brought back to the inputs, a feedback is put in place. Some further considerations about delays and timing are necessary. First off, one might wonder what is the delay in this case, and a off-hand answer could be: the delay is given by the input-output path, exactly as the previous cases. In fact this answer is correct, but something more must be said. Let us refer to a simple feedback circuit, which is basically a variation of the multiplexer already seen. Figure 2.13 shows it. It is a memory cell, namely, a circuit able to retain indefinitely

a signal brought in input in a previous moment, ignoring then the current value of the input.

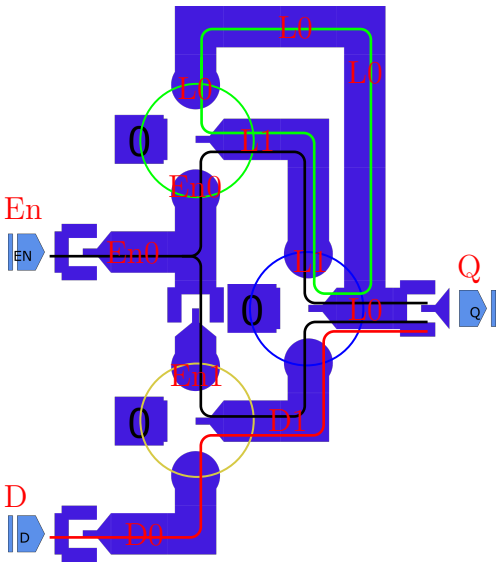


Figure 2.13: A latch made with a multiplexer with the output brought back to one of the inputs. Domain wall, logical gates and loops are marked

Three paths have been drawn in figure 2.13, three "standard ones" and a loop. The characterization of the circuit might be done this way:

- *Enable* signal has a delay equal to 5 half clock cycles. It is a branched path, so it actually has a 4 half cycles delay on one side and 5 on the other, but the same considerations as before apply in this case;
- *D* signal has a 4 half cycles delay;
- The loop has a 2 half cycles delay. All signals that share some elements along their path with the loop will have to stay constant at least for as long as the loop delay, which is a full clock cycle.

It will be now shown what happens when the loop delay rule is ignored. It is easy to guess that the circuit will probably get into a metastable state. Anyway, it might be worth saying that in order for the circuit not to get into metastability, beside the loop delay condition, it has also to be stable from the logical point of view. In other words, the output that is brought back in input must not cause a new output

change, otherwise it will endlessly change. This is a matter of logical function, not of inputs timing. A very good example of this is a NAND gate, whose output is fed back to the input. Suppose the output is 0 and the input is 0. The NAND stays in this state forever. As soon as the input becomes a logical 1, the output switches back and forth between zero and one. In this regard, notice that the loop delays is an even number of half clock cycles: it means that its "logical function" is non inverting. If we consider the loop as a series of inverters (and, to some extent, this is correct), one can say that exactly as a series of an even number of inverters in a loop is not supposed to oscillate, this loop is not supposed to oscillate either. Actually, it is much more sensible saying that a loop whose delay is even (or, whose logical function is non inverting) can, from a theoretical point of view, be stable, whereas a loop whose delay is odd cannot.

Figure 2.14 shows a possible input configuration and timing that brings the circuit into metastability. The color code previously used cannot be used here (because of the loop), and a different one is employed: colored arrows run between some signals, starting from the inputs and reaching the output. In the circuit figure the color of the circles on the gates is the same as the arrows. Some black arrows are there, they simply indicated that the two signals are one the inverted of the other The two domain walls that make the loop always have the same logical value, which continuously changes at each clock edge. The output should take the opposite value with respect to the domain wall marked with *L0*, and it does indeed. It looks like Q follows *L0* because they both change after every clock edge.

Again, it is important to know that the same violation of the loop delay rule (namely, keeping signal *En* to 1 for just half a clock cycle) does not always lead to metastability. Had the input been a logical one, no metastability would have occurred. This idea seems obvious and one could think to take it for granted, since the output and the new input would have had the same logical value; yet, it is uncorrect, because if the initial state is $Q = 0$, $D = 0$, metastability still sets in. Therefore, even if these particular cases in which a violation of the condition does not lead to oscillations, once more it is better never triggering them, and always follow the strictest timing rules. Figure 2.15 represents another case of oscillation, caused by a slightly different input timing. In the example, both D and *En* never change twice within 2 half clock cycles, and yet the circuit oscillates. The reason is

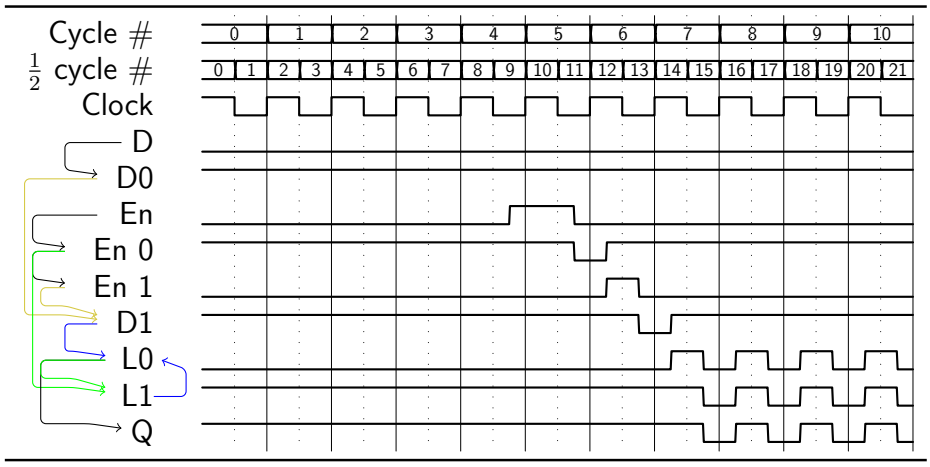


Figure 2.14: Time diagram of a latch led to metastability by signal *En* too rapidly changing

that the loop delay rule should be casted in this form:

When a circuit has an internal loop whose delay is N half clock cycles, a new change of the inputs must wait N half clock cycles from the previous input change.

That having been said, the *D* transition in half clock cycle number 4 and the *En* transition in half clock cycle number 5 (which are both sample at the same time) can be considered two different input changes, whose time distance is smaller than the loop delay (2 half cycles). Given the storing function of the circuit, these constraints might also be interpreted as setup/hold time conditions.

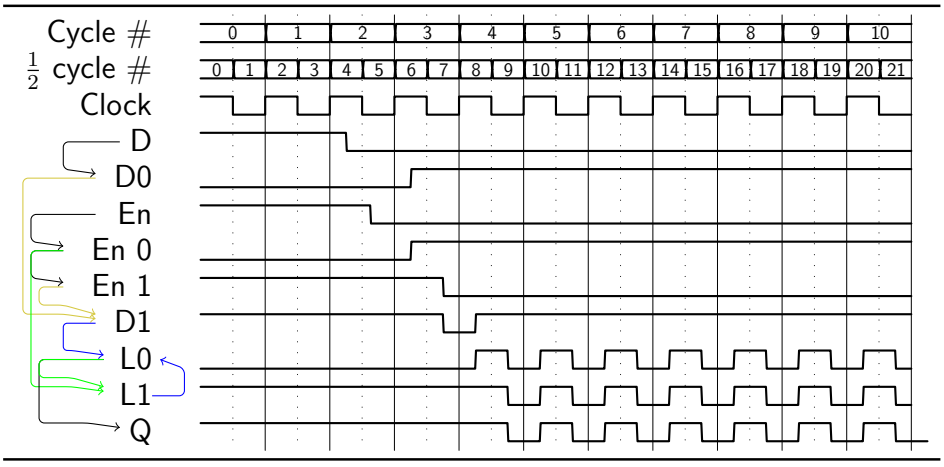


Figure 2.15: Time diagram of a latch led to metastability by signal *En* and *D* changing too close in time to each other

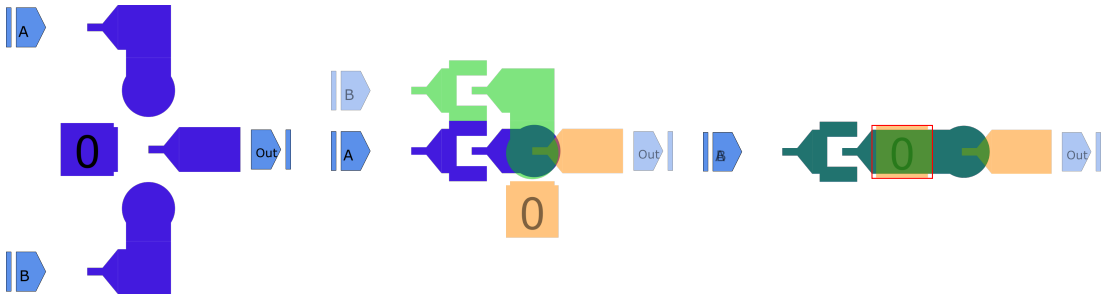
Chapter 3

3D Architectures and Memories

pNML technology, like other NML technologies, allows stacking multiple layers on top of each other. There is no definite difference between wires and logic circuitry in this technology, as the logic gates are basically made with an odd number of wires magnetically coupled; thus, it follows that processing elements can be placed wherever it is most suitable. This cannot be done in CMOS: multiple layers are available, but the processing elements are placed at the bottom, with the upper layers hosting only interconnections. Hence, the mere logic is spread over a plain surface; on the other hand, in pNML technology processing elements can be grown one over the other, with conspicuous area savings.

This chapter shows a few circuits that have been designed trying to exploit as effectively as possible the layers above the bottom one. To this aim, logic gates tend to be built by stacking nucleation centers and pads one over the other. The most obvious way to build a gate is a three layer solution, shown in figure 3.1.

Of course, this way of coupling magnets is ferromagnetic, so that logic connections must be rearranged accordingly (for examples, in figure 3.1 the multilayer implementation has a further inverter). Moreover, placing a certain number of nucleations centers and pads throughout the layers might result in unwanted magnetic coupling: care must be taken in order to avoid this side effect. Inverters too might get coupled, since they contain a nucleation center. Last, passing a signal from a



(a) Planar AND gate (b) 3D AND gate (c) 3D AND gate, alternative version

Figure 3.1: Basic method to build gates in multi layer pNML. The two multi layer gates are equivalent to the on in (a). In 3.1b the fixed magnet and the input B are placed on the sides just to make the circuit easier to be visually analyzed. The actual circuit implemented is something like the one in 3.1c. The inverter must not couple with fixed magnets in layers below and above, that is why the piece of domain wall inside the red rectangle (layer 0 and layer 2) is there, otherwise the gate could have been narrower

layer to the one above (or below) adds a full clock cycle delay to the signal path (it is a ferromagnetic kind of coupling). From the technological point of view, chances are that fixed magnets cannot be biased both ways; this, in terms of boolean logic, means that either a fixed zero or a fixed one is available, not both together. Therefore, circuits must be designed using only one of the two fixed magnets, coupled either ferromagnetically or antiferromagnetically according to the needs. Alternatively, an inverter placed at the "output" of the fixed magnet might be used, provided that it does not get coupled with the layers above and below. Needless to say, these are further constraints to be met when designing the circuit.

Once the circuits have been designed, it is necessary to characterize them with respect to their timing. It has already been shown that pNML timing is quite complicated, and, generally speaking, input signals might have to last several clock cycles. A solid knowledge of the timing behavior allows to reduce the number of clock cycles to the minimum possible value. Memory elements too must be addressed. As already seen in chapter 2, this means having loops within the circuit, which in turn cause a slow-down of the throughput by a certain factor, proportional to the delay of the loop path. Violating this constraint invariably leads to non working circuits, and sometimes to a metastable behavior. A possible alternative consists in using notches, which makes dealing with loops much easier, and reduces metastability risks too. Nevertheless, delay analysis is still necessary, in order to appraise the

length of the notch-opening period. Following sections will give some examples of the concepts above mentioned. Areas are expressed in an arbitrary area unit, called "square", which is the smallest area that an element can occupy in MagCAD, and also the area of a grid cell. The word "square" can be used to define a length as well. Refer to Appendix E for details.

3.1 Decoder

In this section a few decoders are shown. From the logical point of view they are all very much alike, but technologically they are different. Figure 3.2 shows the first attempt. Three layers make up the design, with the top and bottom ones holding the inputs, and the middle one holding the fixed magnets and the nucleation centers: in other words, it is in the middle layer that the computation takes place. As previously mentioned, this is the "standard" way of designing gates with many layers available. Notice that the circuit could not be built as it is, because of two main reasons: fixed magnets are placed above and below inverters (which contain nucleation centers), and also, both fixed-one and fixed-zero magnet are used at the same time. It is shown here just because of its simplicity. Bear in mind that MagCAD does not check any of the above mentioned conditions, and they remain completely up to the designer. The reason is that sometimes introducing inverters where it is not allowed might be convenient when studying how to reduce the critical path, without having to start the whole design from scratch. Examples of this inverter insertion will be provided later on. A second, almost technologically correct, version, is shown in

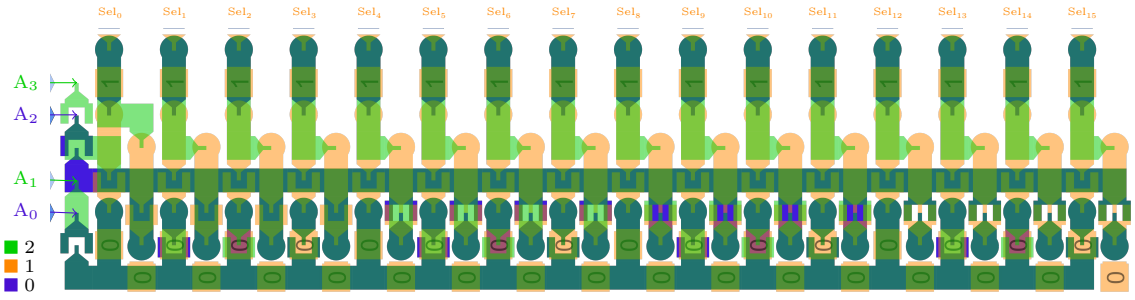


Figure 3.2: Decoder, first version, technology non compliant

figure 3.3. All the pads belonging to the A_2 and A_3 branches have been turned to

the other side. This leaves room enough to allocate the fixed magnets in places not above or below inverters. However, it takes up a little more space. The presence of both fixed-ones and fixed-zeros magnets is the only technological constraint not yet met.

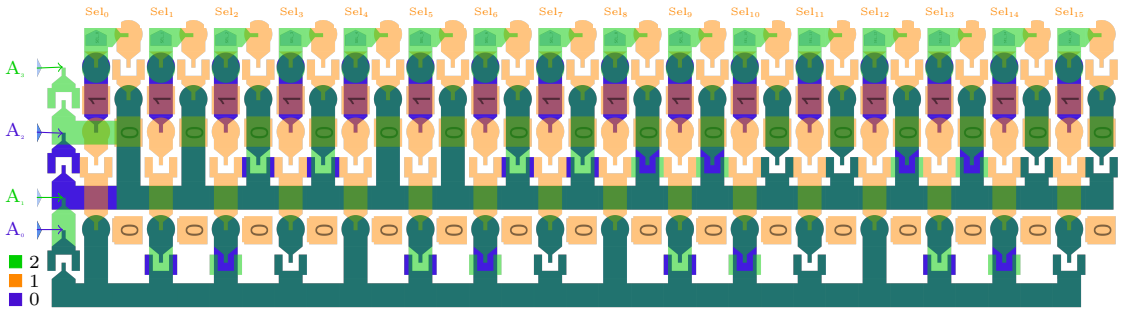


Figure 3.3: Decoder, second version

Finally, it is shown in figure 3.4 how the design is made completely compliant with the technological constraints. The difference between this and the previous version is a good example of how a designer might get by with only a kind of fixed-magnet: the type of coupling must be changed, that means moving signals to other layers. In this case it has been quite easy to do, but in other cases, with denser designs, finding the space might be slightly trickier. Since in this implementation only three-way majority voters were available, one of the other signals too has been coupled the other way, so that the inverter has been removed. Again, in this case it is easy, because the domain wall is long enough. In other cases placing an inverter might not be easy, as the space may be missing, or the inverter might get coupled with surroundings layers. The last one marks a difference with the previous approach. A quick look to the image 3.5 reveals that the delay is not the same for all the outputs: for instance, in the path running from input A_0 to output Sel_0 the delay is 4 half clock cycles shorter with respect to the path A_0-Sel_4. This is quite usual for this technology; sometimes the unbalancing is offset by means of the use of a notch (this concept will be further elaborated in next sections), some other times there is very little to do. In this case the difference between this approach and the previous one consists in only a minor area reduction for the current implementation, but in some cases implementations other than this are very expensive in terms of area, or even unfeasible. In future, when referring to the approach used to build the first example

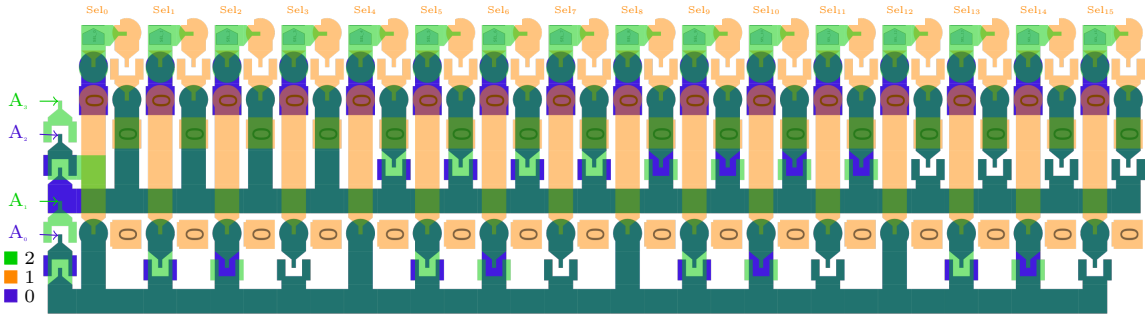


Figure 3.4: Decoder, third version, fully technology compliant

of the decoder, the expression "flat delay" might be used, because the delays are all equal for every output, whereas this last approach might be referred to as "scaled delay", since the delay increases as the outputs get farther away from the inputs.

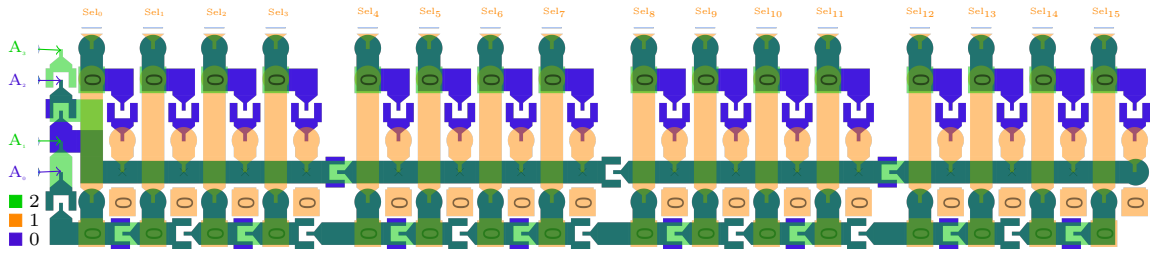


Figure 3.5: Decoder, solution with scaled output delays

Table 3.1 shows the differences in area occupation and delay for the four version of the decoder.
These two possible approaches do not concern only the decoders: almost any circuit

Decoder version	Delay	Area occupation	Feasible
First version	9	$8 \cdot 33 = 264$	No
Second version	9	$9 \cdot 33 = 297$	No
Third version	9	$9 \cdot 33 = 297$	Yes
Fourth version	$\begin{cases} 9 & if\ 0 \leq i \leq 3 \\ 10 & if\ \ i = 4 \\ i+5 & if\ \ i \geq 5 \end{cases}$	$7 \cdot 36 = 252$	Yes

i index is the output number index $0 \leq i \leq 15$

Table 3.1: Comparison of area (expressed in squares) and delays (expressed in half clock cycles) for the decoders described in section 3.1

can be designed in either way. Thus, it is important to underline a further difference between the two models, by means of the timing diagrams of the circuits. Figure 3.6 and 3.7 show a time evolution where all the input combinations are tested. Each input combination is kept stable for a great amount of clock cycles, and this is why the clock signal is not drawn, it would be too dense to be readable.

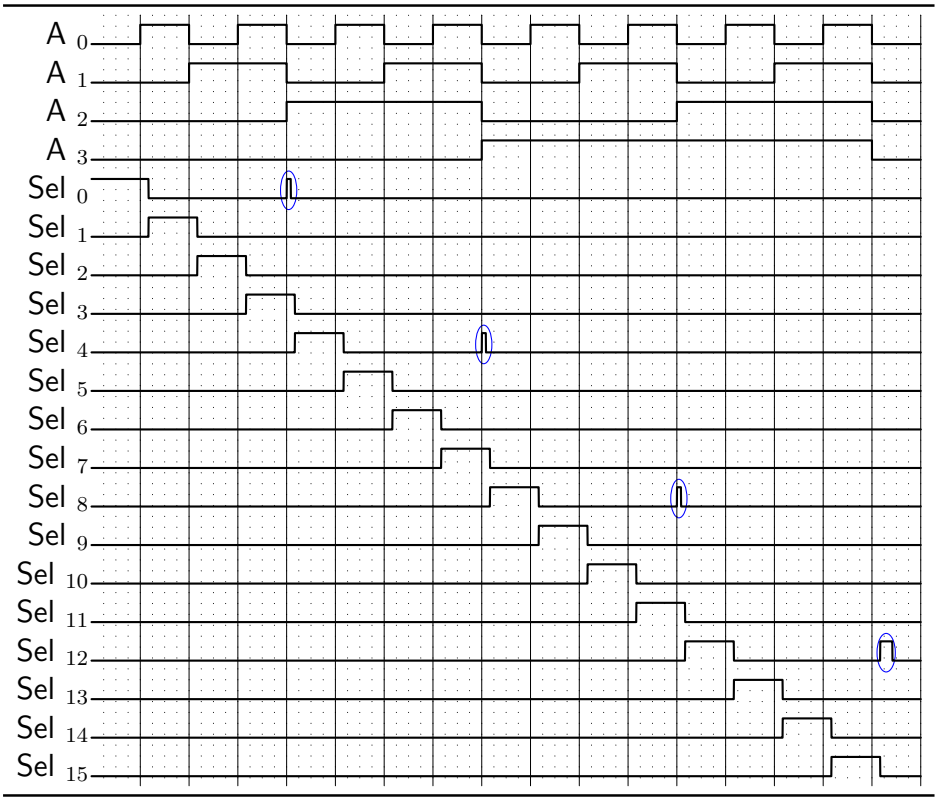


Figure 3.6: Test of the whole set of input combination for the circuit in figure 3.4

Comparing the two time diagrams, one can see that:

- The output are more regular in the flat version, they all last the same amount of time, whereas, for example, in the scaled delay version the Sel_{15} stays at one for less than most of the other output signals. Notice that the input configuration is the same in both tests;

- The scaled version is more prone to glitch. 4 glitches occur in the flat decode test, 10 in the scaled one. Glitches are outlined with a blue ellipse. Moreover,

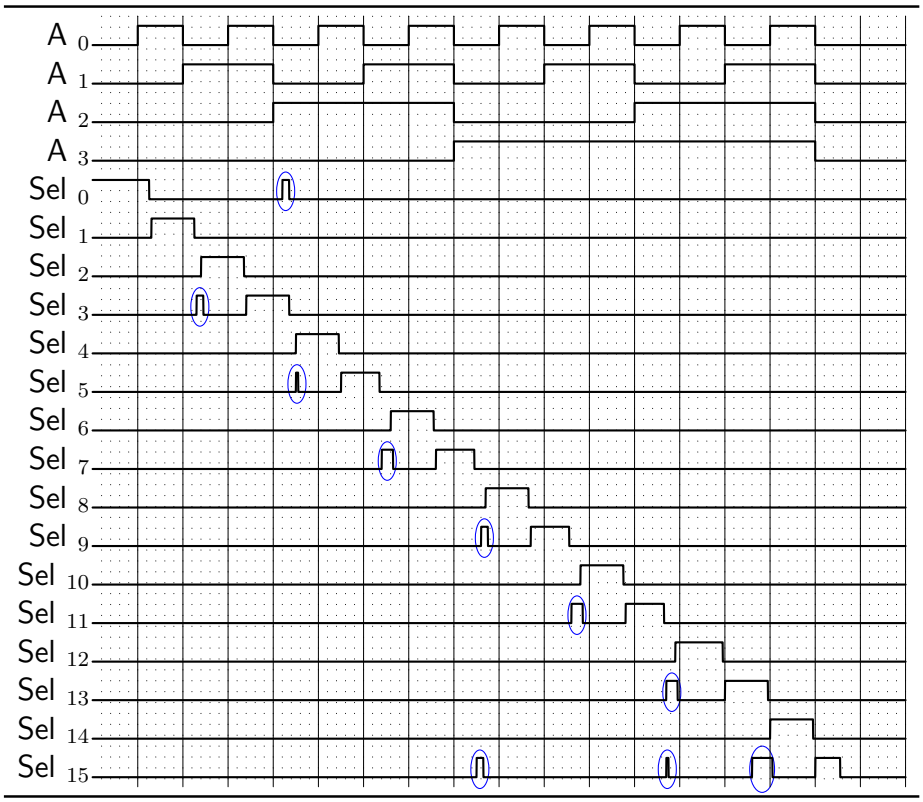


Figure 3.7: Test of the whole set of input combination for the circuit in figure 3.5

the glitches occurring in the flat decoder test cannot be avoided by changing the circuit configuration, they depend on the logical function;

- The glitches too have varying length in the scaled decoder version. They tend to last longer in the outputs placed farther away from the inputs. In output Sel_{15}, the last glitch is similar in duration to the correct output.

The comparison shows that a scaled delay solution is possible and might be more appropriate, but also requires more care in delay analysis.

3.2 Folded Decoder

In order to save space and to try an effective use of the upper layers, the decoder can be "folded". The circuit is in figure 3.8. The outputs from 8 to 15, and relative combinational networks, are brought to three separated layers above. This way the footprint of the decoder is about half of the original one. However, this introduces a delay shift between the outputs 0-7 and the outputs 8-15. This happens because the inputs are placed on the bottom layers; signals run through the 0-7 outputs, climb up to the top layers, and then reach the outputs 8-15. In section 3.1, page 40, it has been already said that a progressively increasing delay among the bits of the output vector is perfectly normal. Hence, this folding just introduces a step-increase of the delay difference between the two groups of outputs (otherwise, the difference would have increased more or less as shown in 3.1) Anyway, another option is available: the input signals should get to the top half without passing through the bottom one. Rather than a "series connection" (inputs → bottom half → top half), there would be a "parallel connection" (inputs → bottom half; inputs → top half; both at the same time). Being this just a study case, it did not seem necessary. Furthermore, this example shows what are the limits when trying to develop the circuitry along the vertical direction: an increasing share of the area is devoted to the routing of signals among the various layers. Given the features of this technology, this also adds to the delay of the paths.

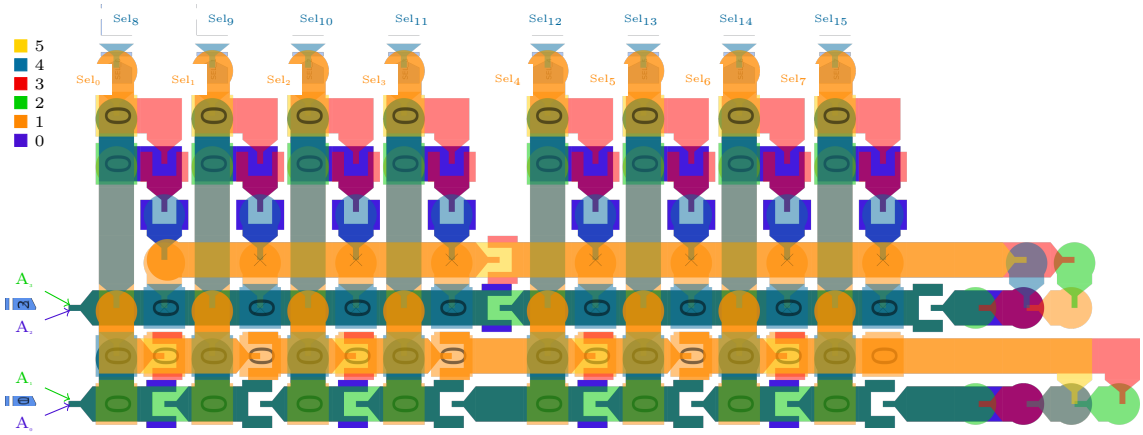


Figure 3.8: Folded Decoder

Table 3.2 lists the delays from each input to each output, and the area occupation. The rightmost column contains the highest value, which can be considered the "true" delay of the output signal, because it is the worst case. The reason why the full table is shown lies in the delay pattern: if each odd delay value is changed with a 1, and each even delay value is changed with a 0, one gets exactly the code for that output. This is not casual at all, but rather a direct consequence of the fact that opposite values propagate with delays whose parity is opposite. Furthermore, between output 7 and 8 can be seen the delay step increase due to the folding of the decoder.

	A_0	A_1	A_2	A_3	Typical delay
Sel_0	4	4	8	8	8
Sel_1	5	4	8	8	8
Sel_2	6	5	8	8	8
Sel_3	7	5	8	8	8
Sel_4	8	6	9	8	9
Sel_5	9	6	9	8	9
Sel_6	10	7	9	8	10
Sel_7	11	7	9	8	11
Sel_8	18	14	16	15	18
Sel_9	19	14	16	15	19
Sel_{10}	20	15	16	15	20
Sel_{11}	21	15	16	15	21
Sel_{12}	22	16	17	15	22
Sel_{13}	23	16	17	15	23
Sel_{14}	24	17	17	15	24
Sel_{15}	25	17	17	15	25
Area			$8 \cdot 23 = 184$		

Table 3.2: Delays of the *Folded Decoder*, expressed in half clock cycles. Area (expressed in squares) is reported at the bottom line

3.3 Multiplexer

Traditional word lines and bit lines are very expensive in pNML technology in terms of area occupation. It has been shown in [10] that a clever and efficient alternative can be a multiplexer, allowing either to feed the current cell with the datum to be written, or to let the latter pass by towards the next cell. Hence, it has been implemented. Figure 3.9 shows a possible solution. Table 3.3 reports the delay for the circuit.

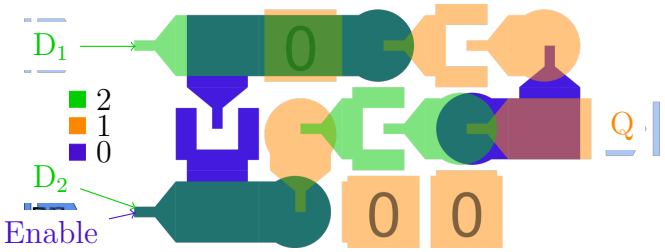


Figure 3.9: Multiplexer in three layers

	Delays
D_1	8
D_2	8
Enable	9 (8) [1]
Area	$6 \cdot 6 = 36$

[1] Signal *Enable* reaches the output through two different paths

Table 3.3: Delays of the *Multiplexer*, expressed in half clock cycles. Area (expressed in squares) is reported at the bottom line

3.4 Storing Cell

If the output of the multiplexer is fed back to one of the inputs, the multiplexer becomes an element with storing capabilities, that from now on will be referred to as "storing element" or "storing cell". The name "memory element" (or "memory cell") will be used for the combination of the storing cell with a multiplexer. Figure 3.10 shows the circuit, whereas timing and area are described in table 3.4.

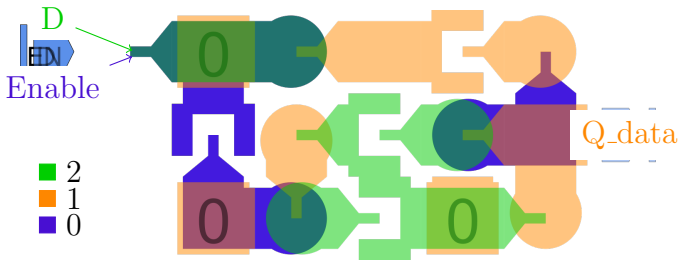


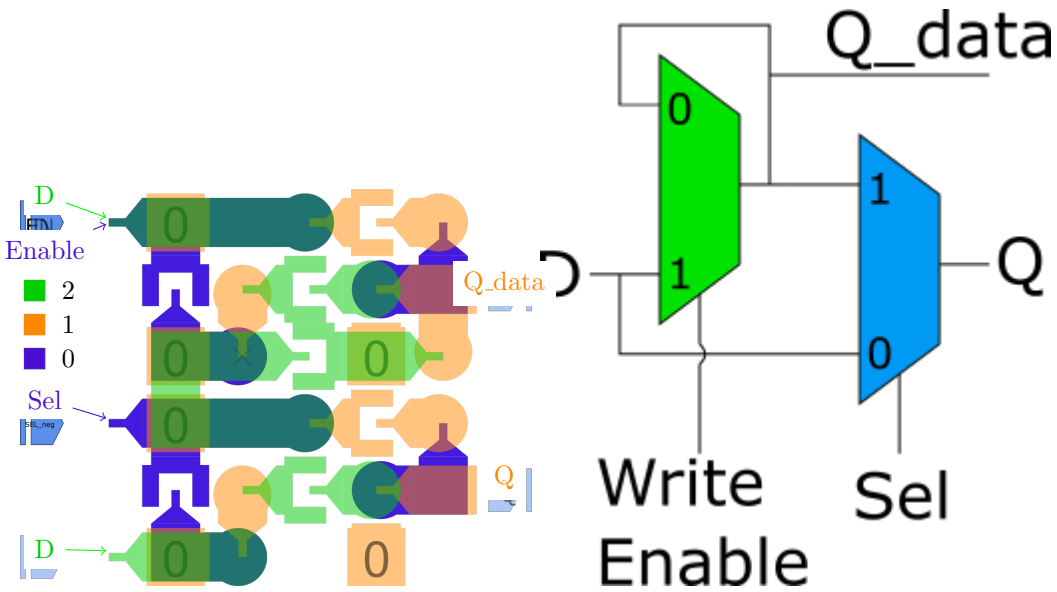
Figure 3.10: Storing cell

Signal	Delay
Data	8
Enable	9
Loop length=10	
Area	$3 \cdot 6 = 18$

Table 3.4: Delays of the *Storing Cell*, expressed in half clock cycles. Area (expressed in squares) is reported at the bottom line

3.5 Memory Cell

A set of two multiplexers can be used to build the generic memory cell. The input data (D) enters both the storing cell and the multiplexer; the output data (Q_data) of the storing cell enters the other input of the multiplexer. The multiplexer will be switched to the storing cell output data in case the current one is the selected memory location, otherwise will bypass the cell and take D as output, making it available for the next location, that will behave likewise. The cell will be always in read mode, unless it is selected and the instruction is a write one.
Figure 3.11 shows the layout of the circuit (a) and a logical representation of it (b). Table 3.5 shows the "truth table" of it, and table 3.6 contains the delay and area figures.



(a) *pNML layout* (b) *Logical representation*

Figure 3.11: Basic memory cell

	Inputs		Outputs	
	Sel	Enable	Q	Q_data
	0	0	D	*Stored Datum*
[1]	0	1	D	D
	1	0	*Stored Datum*	*Stored Datum*
	1	1	*Stored Datum*	D

[1] In practical use, this is a forbidden input configuration

Table 3.5: Logical behavior of the Memory Cell

	Delay to *Q_data* signal	Delay to *Q* signal
Data	8	8
Enable	9	19
Sel	-	8
Q_data	-	10
Loop length=10		
Area	$6 \cdot 6 = 36$	

Table 3.6: Delays of the *Memory Cell*, expressed in half clock cycles. Area (expressed in squares) is reported at the bottom line

3.6 Memory Cell - Notched version

It is well know that pNML technology offers the chance to "pin" a certain logical value in a specific point of the nanowire, depending on the geometrical features of the point. This provides the designer with a latch-like device, the only difference being the different conditions needed to output a logical one or a logical zero. Hence, the memory cell might be latched this way. The storing principles is still the same, namely, the feedback, but the notch reduces the risk of metastability (output oscillating between one and zero because of improper timing). The reason why this happens can be easily explained. First, let us recap that a loop is updated this way: it must be logically "broken" somehow, then a new value is fed into the loop. The updated value must be kept stable long enough for the signal to reach the other end of the broken loop, otherwise, when closed, the loop will retain two incoherent values. This is exactly what is called "metastability". That being said, the way a notch manages to avoid metastability risks is straightforward: a notch keeps stable a certain signal, so that as long as the notch opening period is longer than the loop delay metastability cannot occur. Figure 3.12 shows the layout of this solution.

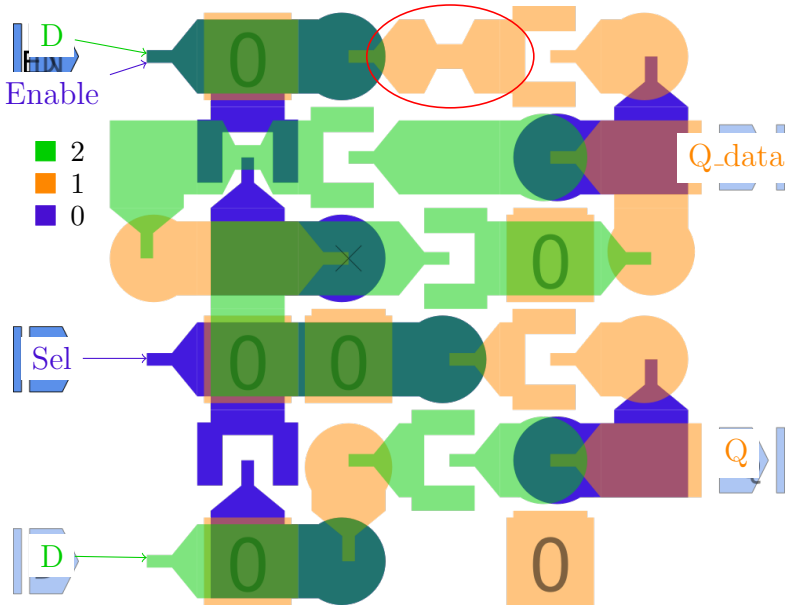


Figure 3.12: Memory Cell with notches

The way a notch is clocked is shown in figure 3.13. The length of the pulse for

the *notch clock* (often known also as "depinning clock"), its phase with respect to *global clock* (the only other clock mentioned thus far), and any other parameter is determined basing on technological and layout-related grounds. In this case, the notch is always followed by an inverter, and this makes the timing simpler to design. In some cases (which will be addressed later on) things might get slightly more twisted.

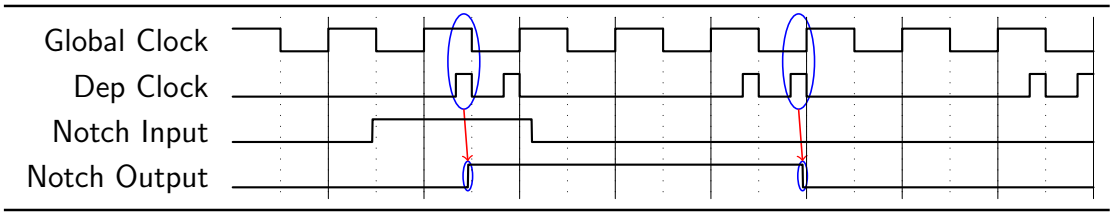


Figure 3.13: Timing of a signal passing through a notch

In order to clarify a little more the matter, let us reconsider the example in figure 2.14, where a metastability condition was triggered by signal *Enable* being too short. Figure 3.14 shows what happens in this case. Consider input *Enable* and the notch marked in red.

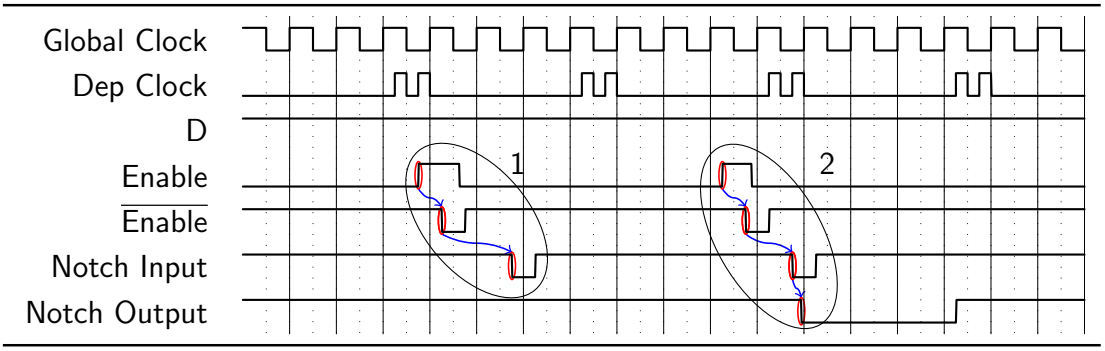


Figure 3.14: Metastability prevention property of a notch

They are all represented in the timing diagram. It can be seen that how quickly the *Enable* signal changes no longer matters: either it finds its way through the notch, as it happends in case 2, or does not as it happens in case 1. In either case, the output of the notch changes at most at the notch opening period, which is set long enough for the circuit not to get into metastable states. This is not different at all from what happens in CMOS when flip-flop are introduced in order to filter out glitches.

The numeric values of the delays are the same as the non-notched version; they can be found in table 3.4. Only three delay parameters, conceptually very similar to the usual setup and propagation delay, listed below:

- Signal *Enable* must be stable 6 half clock cycles before the clock cycle where the notch is opened;
- Signal *Data* must be stable 3 half clock cycles before the clock cycles where the notch is opened;
- Signal *Q_data* is ready 5 half clock cycles after the clock cycle whre the notch is closed

In fact, first two parameters are a sort of setup time, whereas the last one is a propagation time.

3.7 Memory Array

The elements shown so far are all is needed to build a memory array with the usual interface, consisting of an input vector, called D, and address vector, called A, a bit that defines the writing\reading nature of the operation, called R\W. Apart from some wiring, the circuit, shown in 3.15, is only a cut-and-paste and composition of the previous circuits. Each horizontal line made of Memory Cells makes up a different word. A few remarks are needed here. First, this version is not technological correct. It contains two of the most common technological mistakes: domain walls over/below inverters and presence of fixed magnets of both signs. This was a mere test version. Second, the decoder is a shade different from the circuits shown so far, but this is not particularly unsettling. Third, and probably the most important point, very long domain walls can be spotted in the circuit. This makes the clock period very long, which means that the circuit cannot be very fast. Fourth: *D* signal behaves unlikewise, it has to go through all the *Memory Cells* that have been described previously.

This prompts a more general consideration about pNML.

> A very well-known, and sometimes annoying, feature of all the QCA technology is known as “Layout=Timing”. pNML technology suffers

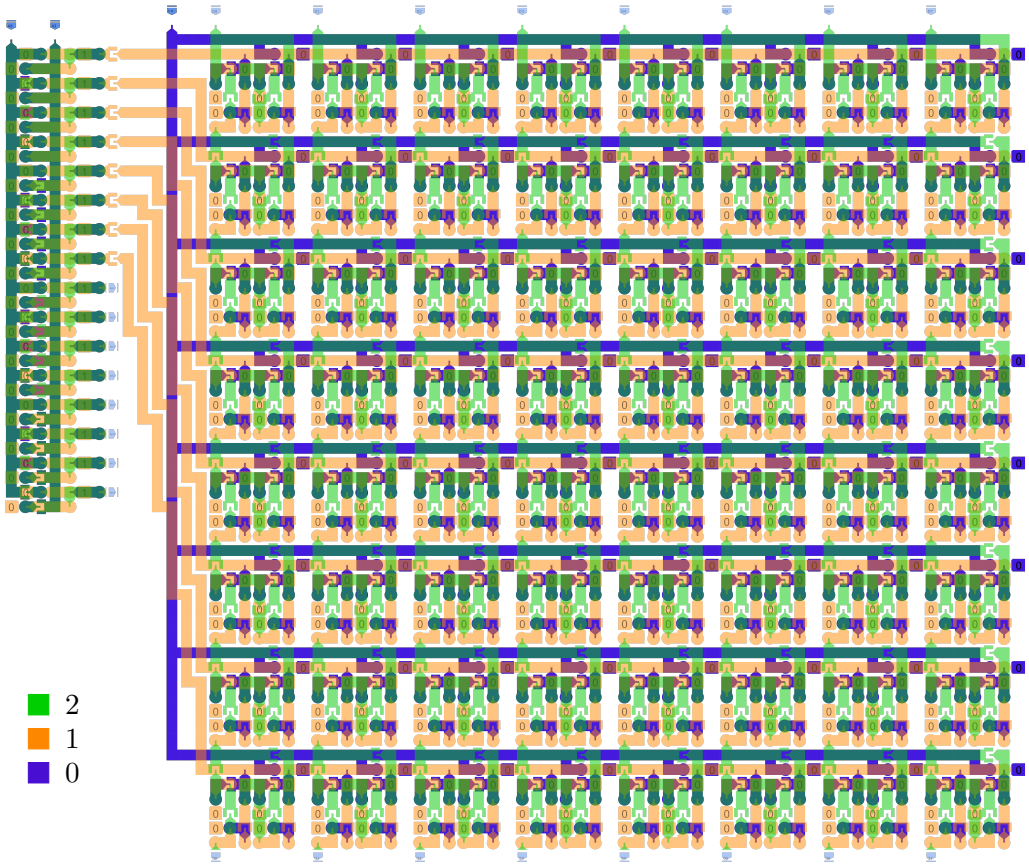


Figure 3.15: Memory Array

from this problem, but offers a further degree of freedom. While in the other QCA implementation a very long wire causes an increase of the delay *in terms of clock cycles*, pNML provides the designer with two alternatives:

- A domain wall is drawn from the starting point to the arriving point. This way, the clock period increases significantly. This is the possibility that other QCA implementation do not offer;
- The domain wall mentioned in the previous point is drawn, but this time and even number of inverters is staggered along the wire. This way the clock period does not increase tremendously, but the signal will take more than one clock cycle to reach the end of the wire. This is somehow equivalent to other QCA cases;

> In other words, pNML is not affected by the "Layout=Timing" issue, as long as only wires are concerned (logic gates will always show such kind of behavior). The problem could be eliminated, provided that one is willing to work with a clock period likely to be very high.

In order to better understand this idea, suppose that the last word must be written. The *D* signal must travel through all the chain of Memory Cells, so that it takes several clock cycles for it to reach the Memory Cells belonging to the last word. Compared to the path signals such as *Write_Enable* must go through, it comes out that the latter signal gets to the needed point much earlier than the *D* signal. If, by choice, all the signals, no matter they are data, address, or control signals, are made to last the same amount of time (a very convenient choice), this means that *Write_Enable* signal will stay active for the whole time needed to *D* signal to reach the point.

The decoder is a 4-to-16, and the memory 8 Words x 8 bits. It is clearly mismatched, but this format is easier to understand than a 16 Words x 8 bits when looking at the circuit.

This problem about signals that have to stay active for very long can be mitigated somehow by means of some circuital modifications. These are shown in figure 3.16. From a timing point of view, the solution simply consists in pipelining the wires that deliver signals such as *Write_Enable* or *Address*, so that they reach each cell together with the *D* signal.

It might be interesting noticing that the amount of space allocated for the inverters that balanced the path of all the signals is quite large, even if this layout is not technologically correct, for the two reason mentioned above. Sometimes finding the room for the path balancing might be troubling.

A schematic representation of the memory connections is depicted in figure 3.17, where the *Write_Enable* signal has been ignored to keep the analysis simpler. Hence, there are the *Address* signals, delivered through long wires, and *Data* signals, that travel all the way through the *Memory Cells*, and therefore their trip is interrupted by several inverters and logic gates. Suppose a signal, lasting 3 clock cycles, is generated and pushed both into *Address* and *Data* lines. The timing might be something like what is shown in figure 3.18 (in all these wave representations, the fact that zeros and ones propagate on opposite clock edges is ignored). *Address*

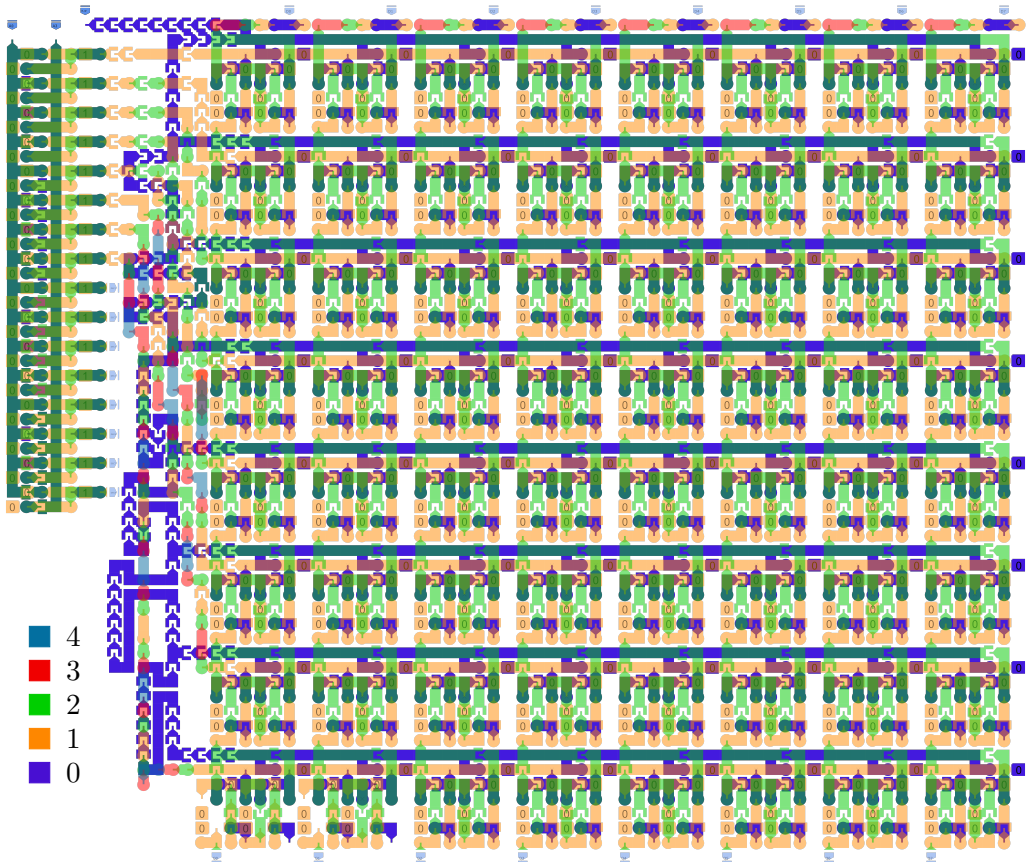


Figure 3.16: Memory Array with balanced paths

signals cross much less logic and get to the farthest cells much earlier than the *Data* signal. A *Memory Cell* must receive all the signals at the same time, otherwise an incoherent operation will be carried out.

Thus, two possible solution can be chosen:

- The *Address* signals are issued later than the *Data* signal, so that they reach the selected cell at the same time. This means that the time difference between the two signals depends on the cell in use. This solution is extremely complicated, even though slightly more effective;

- A worst case solution is put in place: the *Address* signal will stay active long enough so that, if the cell in use is the farthest one, it will still have *Address* and *Data* signals active for a sufficient amount of time.

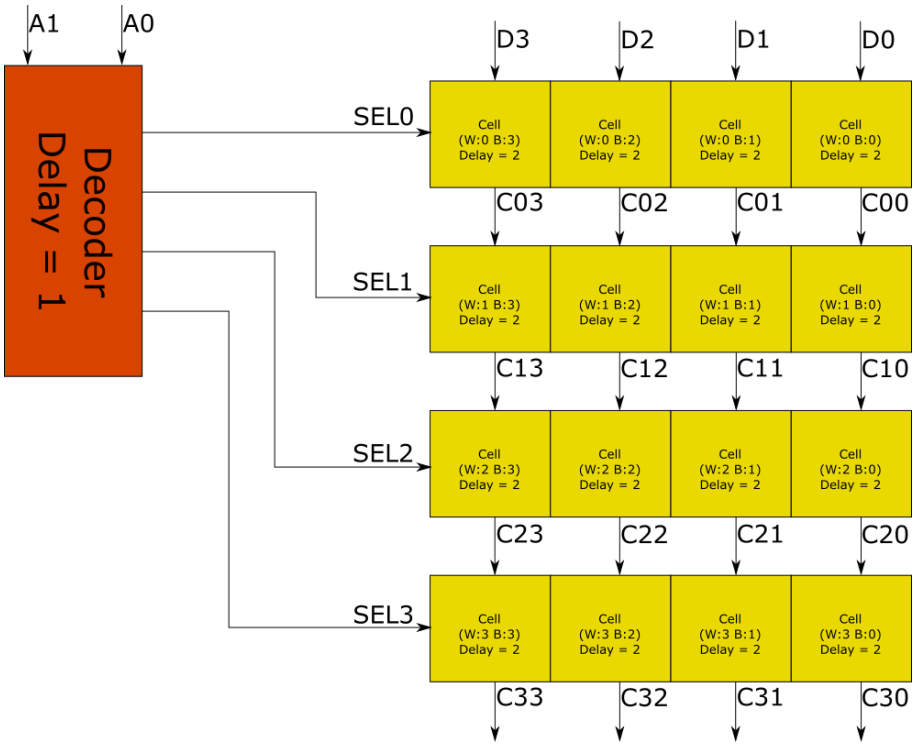


Figure 3.17: Schematic representation of the memory in figure 3.15 (unbalanced paths)

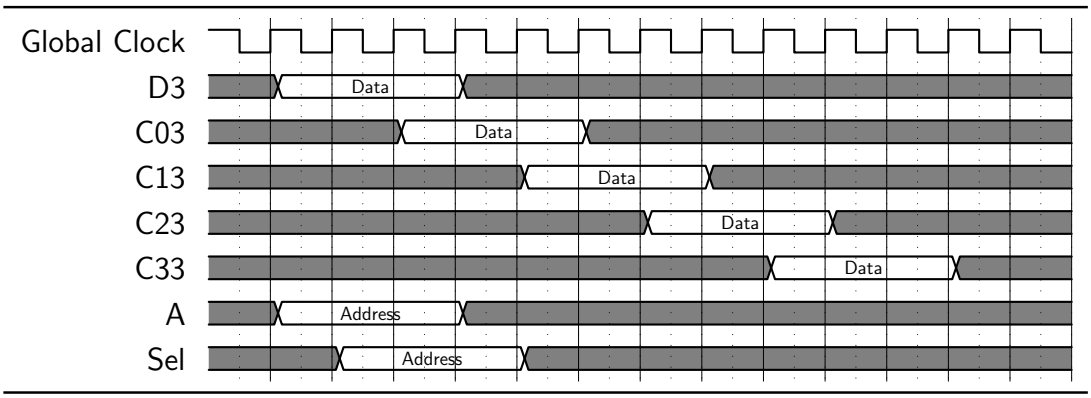


Figure 3.18: Signals' timing when no analysis is carried out. This way, the circuit does not work

The second solution can be seen in figure 3.19. In this case, for every cell the signal *SEL* and the signal *Data* are both valid for some clock cycles. However, having *Address* and *Data* that last for different amounts of time might complicate the control. Therefore, the solution actually implemented is shown in figure 3.20. The memory can be modified and made with all the delay carefully trimmed: it is

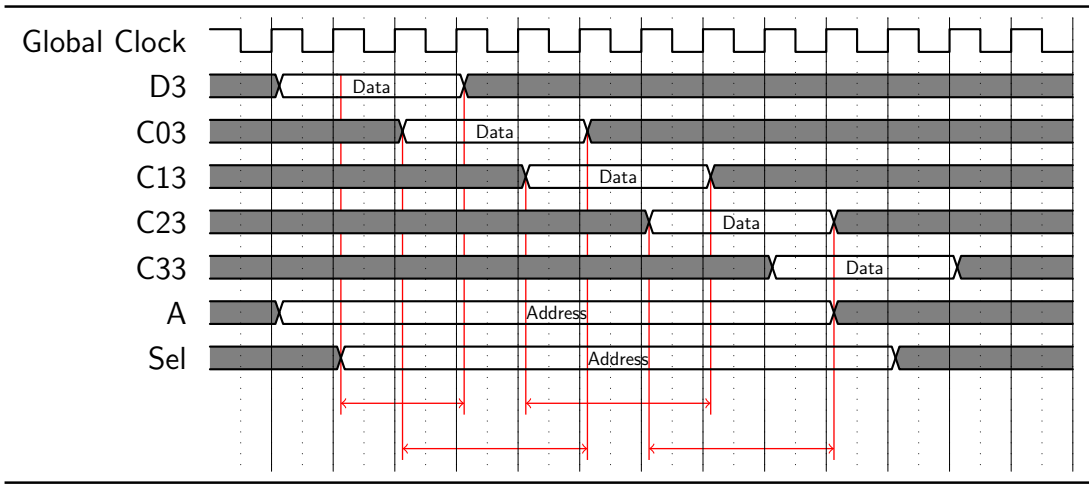


Figure 3.19: Signals' timing when *Address* signal is stretched so that it gets to each cell joined with the *Data* one. This timing works, but its generation is complex

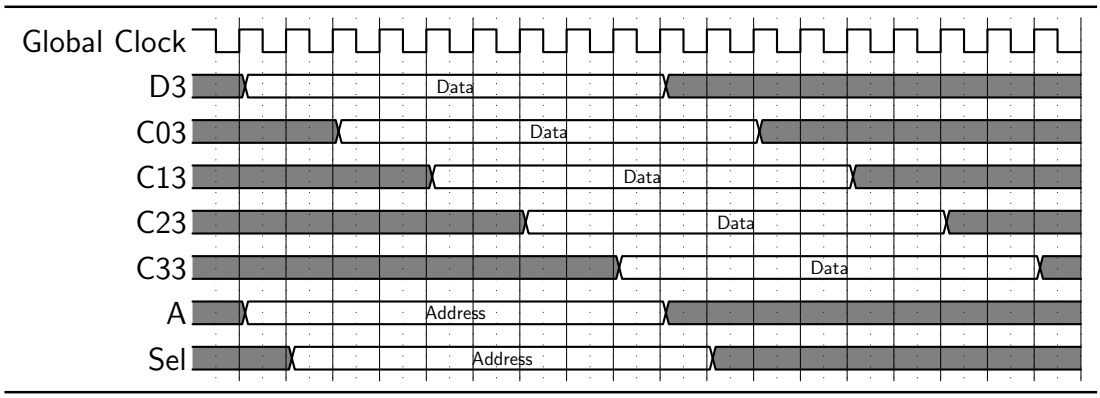


Figure 3.20: Timing actually implemented

schematically represented in figure 3.21. The "buffer" blocks are nothing more than an even number of inverters. Its time wave is shown in figure 3.22. Notice how all the signals reach a cell at the same time: this way, the signal are active for a much shorter time.

Figure 3.22 shows how in each cell delays are perfectly matched. Because of the active time reduction for the signals, the throughput increases too. The delays of this circuit can be calculated from the ones of its components. Just a further delay must be taken into account: from a *Memory cell* to the next one the delays is exactly one clock cycle.

Last, the 16-outputs decoder is fully exploited with the memory in figure 3.23,

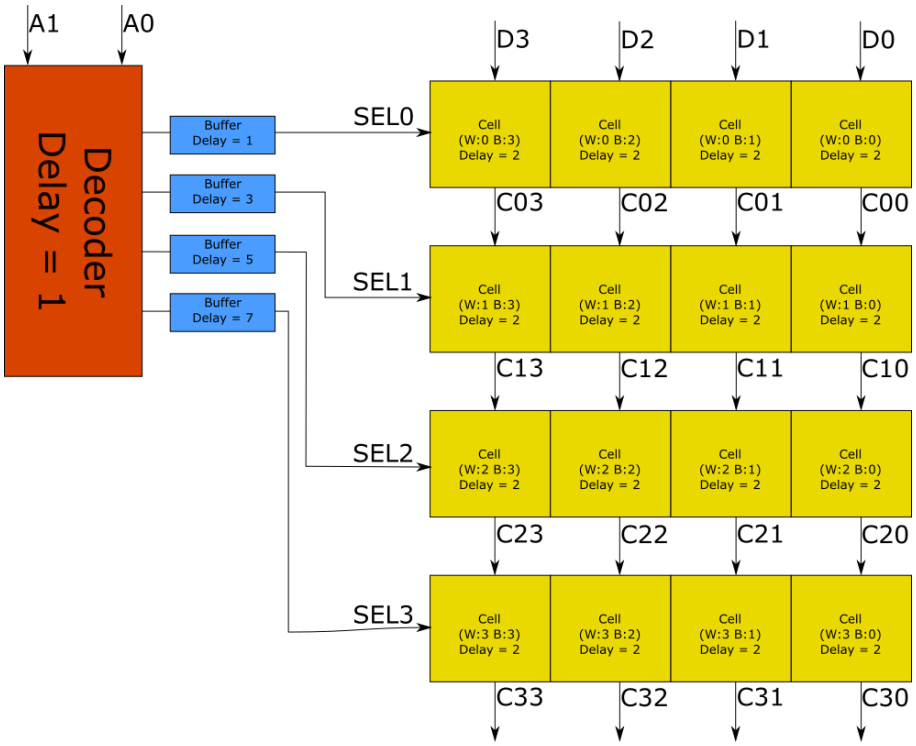


Figure 3.21: Schematic representation of the memory in figure 3.16 (balanced paths)

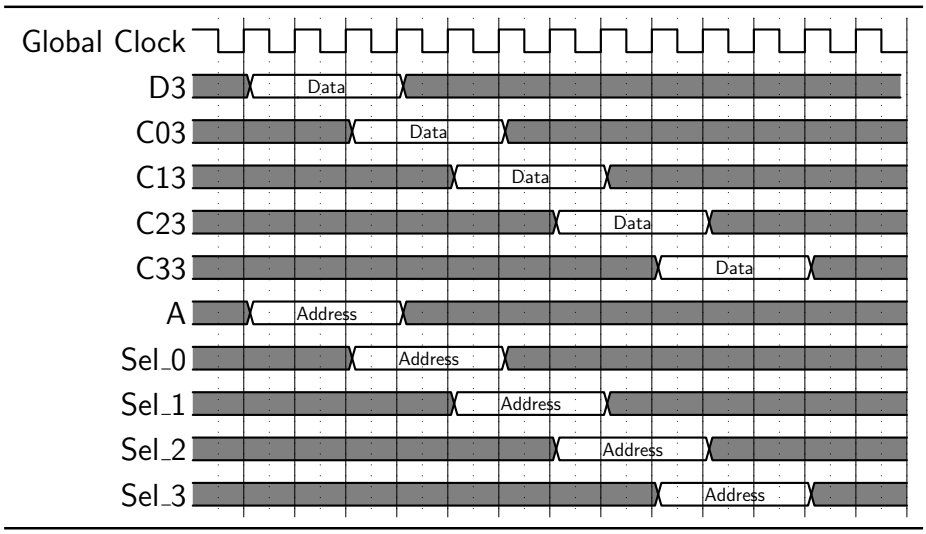


Figure 3.22: Timing of memory whose delays have been balanced

where two plains of 8-words 8-bits memories are stacked one over the other. At the bottom, a line of multiplexers, driven by the last bit of the decoder's output,

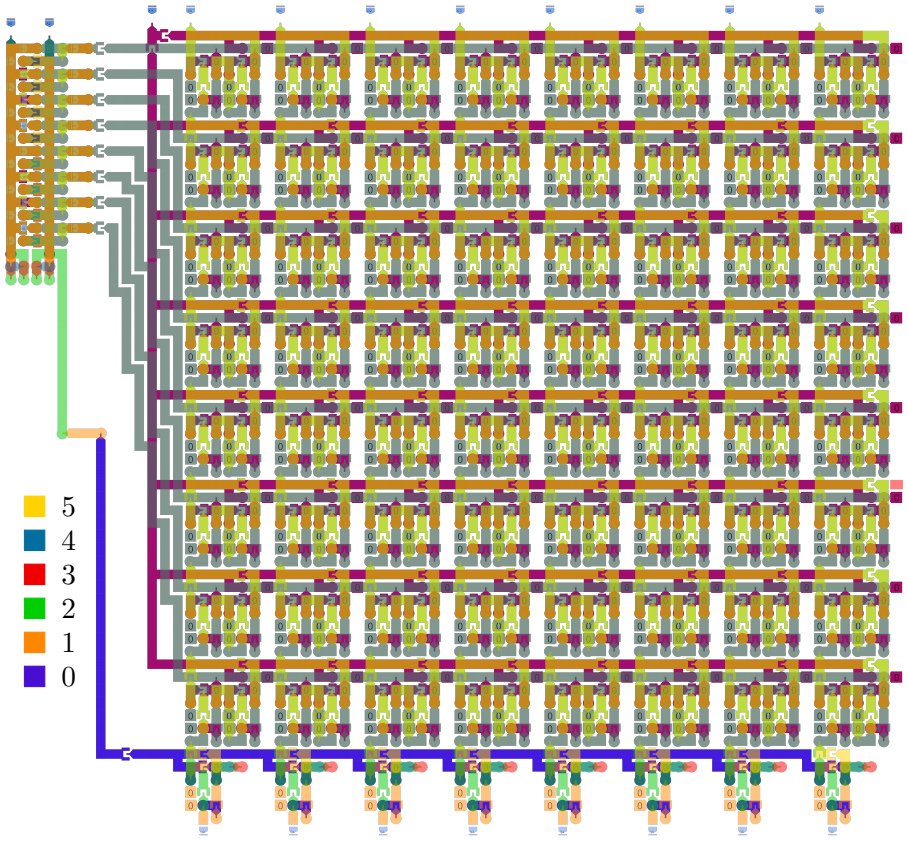


Figure 3.23: Double-plane Memory Array

switches between the two memory planes. This choice avoids the long delay that passing through 16 memory cells would have caused. Anyway, this circuit is just an extreme test of the opportunities given by the multi layer features, but is rather unlikely to be found in practical use.

Areas for these two memories are respectively $70 \cdot 57 = 3990$ for the single plane version, and $70 \cdot 63 = 4410$ for the two plane version.

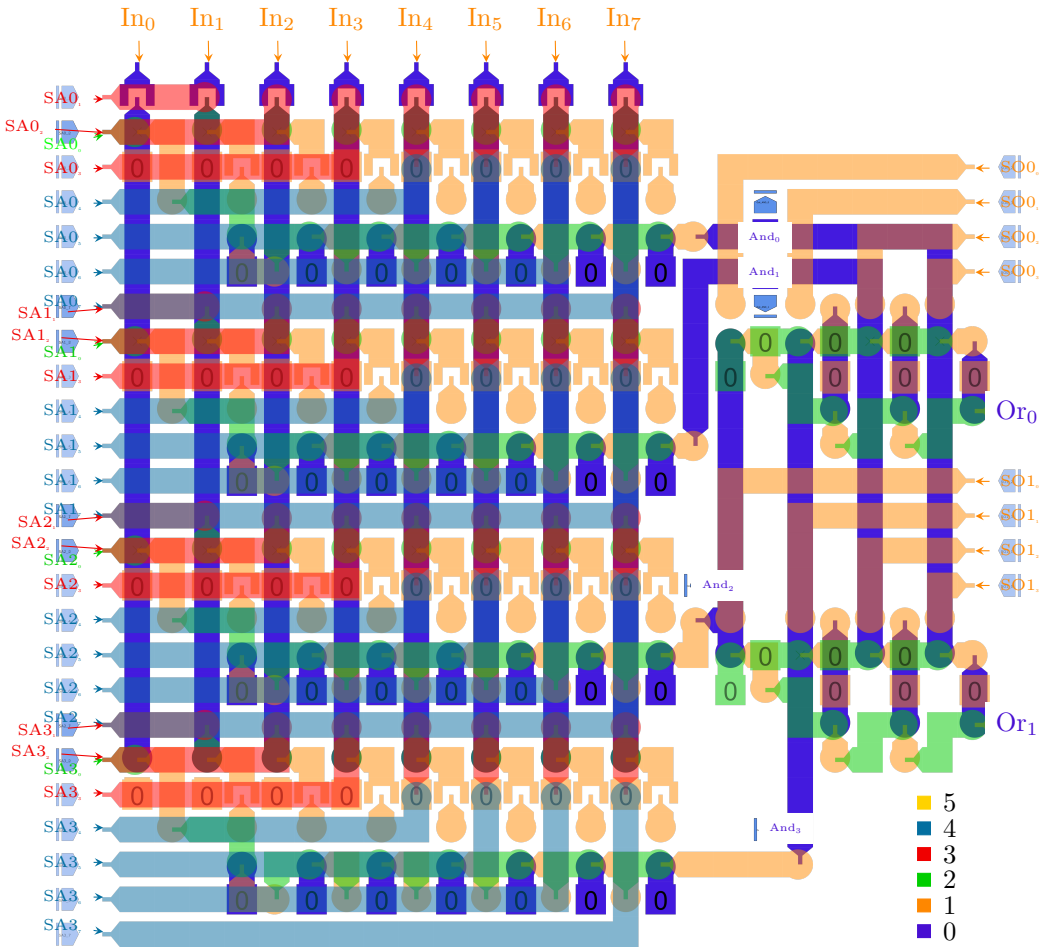


Figure 3.24: Programmable Logic Array

3.8 Programmable Logic Array

A PLA has been designed, following the same criteria (maximum use of the third dimension). Even though a PLA is seldom used in hand-crafted circuits such as the ones studied here, it could nevertheless be considered as an attempt to produce a regular combinational circuit, as more dense and three-dimensional as possible. Figure 3.24 and 3.25 show a PLA with 4 ANDs and 2 ORs; a complete list of the delays is presented in table 3.7, along with the area occupation.

The circuit is technologically compliant; the most interesting thing to point out is the presence of long domain walls, which slow down the clock frequency. Given the relative rarity of use of the circuit, trying a "scaled delay" version would increase

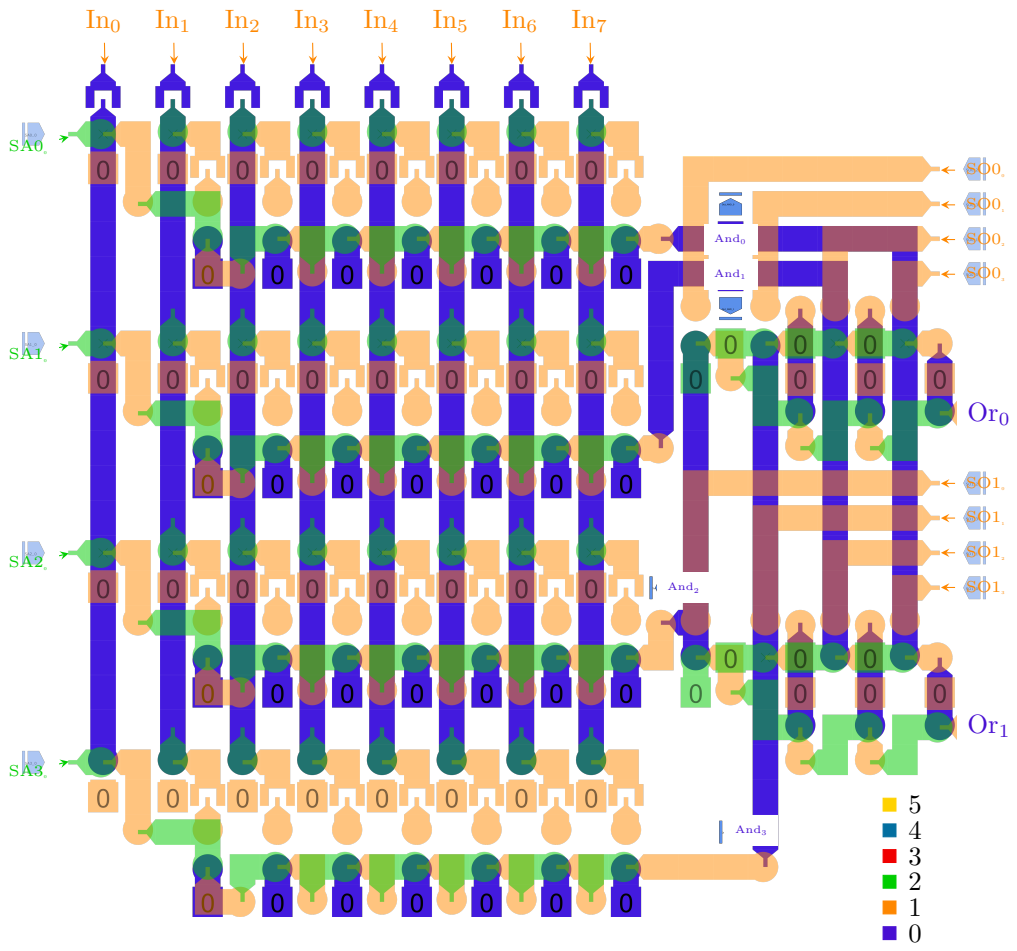


Figure 3.25: Programmable Logic Array, two layers removed from top for a better view

the area and possibly the delay, and here is not worth it. However, in chapter 6, section 6.5, when some of these circuits are redesigned in only two layers, an example of PLA built according to that principle is proposed.

Start point	End point	Delays [Half clock cycles]
In_0	ANDs output	30
In_1	ANDs output	30
In_2	ANDs output	26
In_3	ANDs output	22
In_4	ANDs output	18
In_5	ANDs output	14
In_6	ANDs output	10
In_7	ANDs output	6
AND_0 output	ORs output	8
AND_1 output	ORs output	12
AND_2 output	ORs output	16
AND_3 output	ORs output	16
$^1SA_0$	ANDs output	31
$^1SA_1$	ANDs output	29
$^1SA_2$	ANDs output	27
$^1SA_3$	ANDs output	25
$^1SA_4$	ANDs output	21
$^1SA_5$	ANDs output	17
$^1SA_6$	ANDs output	13
$^1SA_7$	ANDs output	9
$^2SO_0$	ORs output	7
$^2SO_1$	ORs output	11
$^2SO_2$	ORs output	15
$^2SO_3$	ORs output	15
Area	$27 \cdot 25 = 675$	

[1] These are the AND plane switches
[2] These are the OR plane switches

Table 3.7: Delays of the *PLA*, expressed in half clock cycles. Area (expressed in squares) is reported at the bottom line

3.9 Finite State Machine

The last circuit designed was also the most advanced from the conceptual point of view, namely, a Moore finite state machine. The function implemented is a sequence identifier, chosen for its simplicity and for its one-input-one-output property. No matter what technology is used, a FSM is fundamentally a series of memory elements that encode the state, plus some logical networks that make the next-state logic.

Many possible state encoding are available, but the one that makes the hand-made layout the easiest to be designed is the one-hot encoding. Furthermore, this style tends to make simpler the combinational networks at the cost of a higher number of memory elements that represent the state. However, the increase of these memory elements is not tremendous when the number of states is low (about 10). In pNML, notches are used to implement the memory elements. The use of notches to make a FSM is suggested in [10]: the circuits might become unstable otherwise.

The circuit can be seen in figure 3.26: from the topological point of view, the notches are placed on the right side (green box), whereas on the left side the combinational networks carry out the operations to determine the evolution of the circuit (blue box). This is just a rough scheme, some of the logic is placed on the right side as well. In the middle it has been placed the domain wall holding the current datum value, making it available for the combinational networks across the whole circuit width. Besides that domain wall runs the domain wall holding the *Reset* value (the two red lines). The circuit is less regular than all the other ones seen so far, because the task it carries out is arbitrary, so there is no particular symmetry in its logical function, and, hence, in its layout. Consequently, the area exploitation turns out to be less effective. Besides that, in this layout the chance to grow the circuit on multiple layers is probably leveraged in the most regular way: each notch, representing a state of the FSM, is located on a different layer, in order to keep apart the wires needed to carry out the combinational operations. These are located mainly at the bottom layers: nevertheless, drawing the wires from the notch output to the logic gate is easier when the notches are placed on different layers (less entangling and twisting). This circuits contains both signs for the fixed magnets, so it would need some rearranging before actual technological manufacturing. The timing and area analysis are reported in table 3.8. The notches are considered equivalent to standard flip flop, so that all the paths running to their input or from their output are listed.

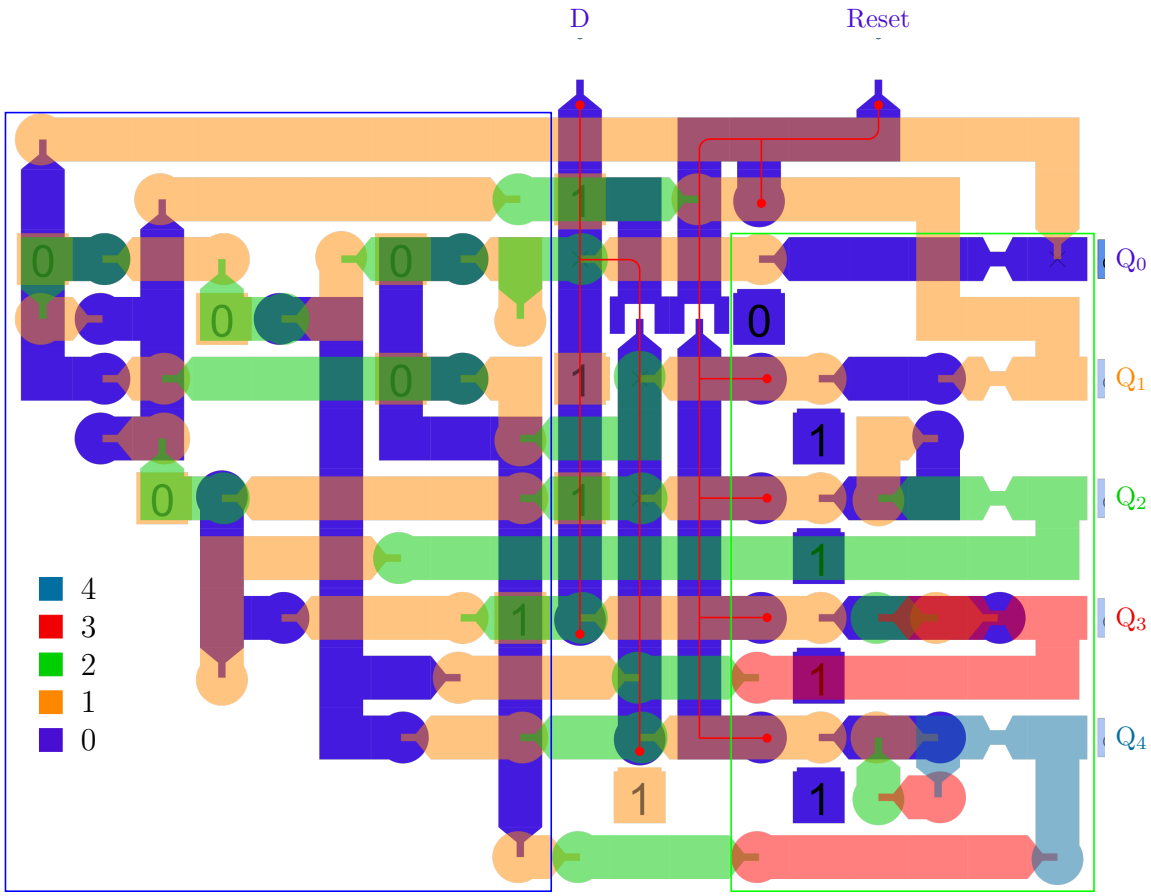


Figure 3.26: Layout of the FSM sequence identifier

From	To	Delay [Half clock cycles]
D	Notch Q0	5
	Notch Q1	8
	Notch Q2	10
	Notch Q3	11
	Notch Q4	14
Reset	Notch Q0	2
	Notch Q1	5
	Notch Q2	7
	Notch Q3	9
	Notch Q4	11
Q0	Notch Q0	20
	Notch Q1	18
Q1	Notch Q0	26
	Notch Q2	22
Q2	Notch Q2	16
	Notch Q3	18
Q3	Notch Q0	18
	Notch Q4	22
Q4	Notch Q0	16
	Notch Q1	18
Area		$18 \cdot 14 = 252$

Table 3.8: Delays of the *FSM*, expressed in half clock cycles. Area (expressed in squares) is reported at the bottom line

Chapter 4

Implementation of the Summed Area Table Algorithm

The SAT (Summed Area Table) algorithm involves a two-dimensional array of numerical values, and returns another numerical value for every element of the array. The value of each location is given by the sum of all the locations' contents, excluding the ones that are either below or to the right of the one considered. From a geometrical point of view, the locations included belong to the rectangle whose opposite corners are the top left corner of the array and the location considered. In formula:

$$I(x,y) = \sum_{x'=0}^{x} \sum_{y'=0}^{y} i(x',y')$$

The algorithm lends itself very well to an "incremental" implementation, because the SAT value for every location can be thought to be:

- The SAT value of the location above, plus the sum of the values of the locations in the same row as the location considered, from the left edge to the location considered (included);
- The SAT value of the location just on the left of the location considered, plus

the sum of the values of the locations in the same column as the location considered, from the top edge to the location considered (included);

A more elaborate way to calculate the SAT values might be the following one:
The SAT value of each location is equal to the SAT value of the location above plus the SAT value of the location just on the left of the location of interest, minus *the SAT value of the location above* and *just on the left of the location of interest*
The previous statement is simply expressed with the formula:

$$I(x,y) = I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$

Therefore, it leaps to the eye that a modular or incremental routine implementing the algorithm is conceivable and might also be very effective, since the value of every location is strictly related to the one of the neighboring ones.

Consider figure 4.1

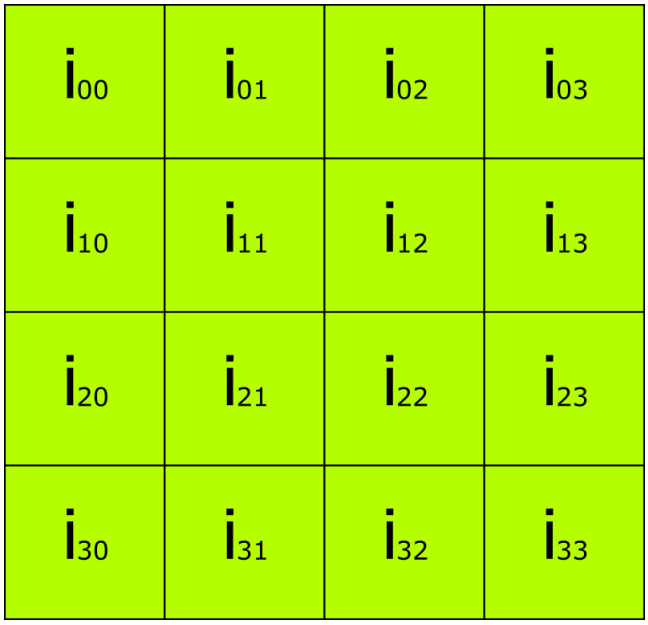


Figure 4.1: Example of a matrix where SAT algorithm can be applied

Now suppose the matrix is split in four groups, each one with four locations; then, the SAT algorithm is applied locally for each group, disregarding the others. It is easy to realize that the sub-matrix on the top left corner already contains the

Location	SAT value
I_{00}	i_{00}
I_{01}	$i_{00}+i_{01}$
⋮	
I_{10}	$i_{00}+i_{10}$
⋮	
I_{22}	$i_{00} + i_{01} + i_{02} + i_{10} + i_{11} + i_{12} + i_{20} + i_{21} + i_{22}$
⋮	
⋮	
I_{33}	$i_{00} + i_{01} + i_{02} + i_{03} + i_{10} + i_{11} + i_{12} + i_{13} + i_{20} + i_{21} + i_{22} + i_{23} + i_{30} + i_{31} + i_{32} + i_{33}$

Table 4.1: How the SAT values are calculated in terms of the original values of the matrix

SAT values both when considered alone and as part of the main matrix. This is due to its being the top left corner of the whole matrix. The locations in the other submatrices do not contain the value that they would contain if the SAT algorithm had been applied to the whole matrix. However, some of the missing values are already calculated in other locations. For example, if location 01 and location 02 are added together, the resulst is the value location 02 would have after having applied the SAT algorithm globally. The same thing holds true for every location, and the rest of the steps are explained in table 4.1. These 2 by 2 matrices represent the building blocks of this SAT routine, and from now on might be referred to as "Cells", especially when discussing the hardware. A size other than 2 by 2 could have been chosen and would work, but in this whole chapter the 2 by 2 size is never changed. There is a reason for this, and will be given later on.

Figure 4.2 represents the state of the matrix just after the algorithm has been applied to each submatrix. The black text shows what elements are inside that location(each one summed to the other). The red text shows what elements are still missing (meaning that they should be added to the black colored value) to get the value of the SAT when applied to the matrix as a whole.

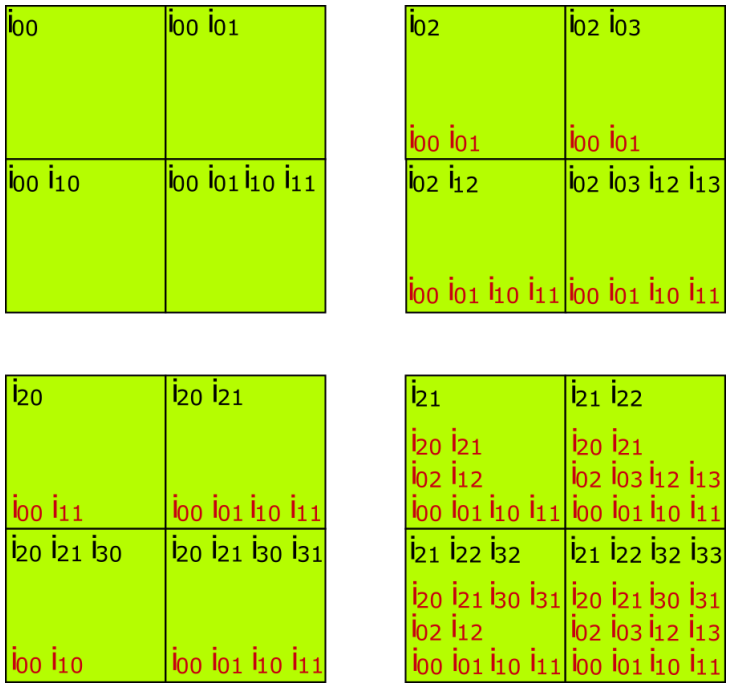


Figure 4.2: SAT Table split in four matrices and after the first step of the algorithm

Notice that:

- In the top left submatrix, "local" or "global" SAT are the same thing;
- Location 02 and 03, that is, the top row of the top right submatrix, are missing the content of the 01 location;
- Location 12 and 13, that is, the bottom row of the top right submatrix, are missing the content of the 11 location;
- Location 20 and 30, that is, the lefthand column of the bottom left submatrix, are missing the content of the 01 location;
- Location 21 and 31, that is, the righthand column of the bottom left submatrix, are missing the content of the 02 location;
- The bottom right submatrix is missing values from all the other three submatrices (each one written in a different row); more specifically, from locations 11, 12, 13, 21 and 23. The situation is more complicated, for the time being, let us put it aside;

Now each cell delivers, if needed, its content to the cell besides. Figure 4.3 shows the final outcome.

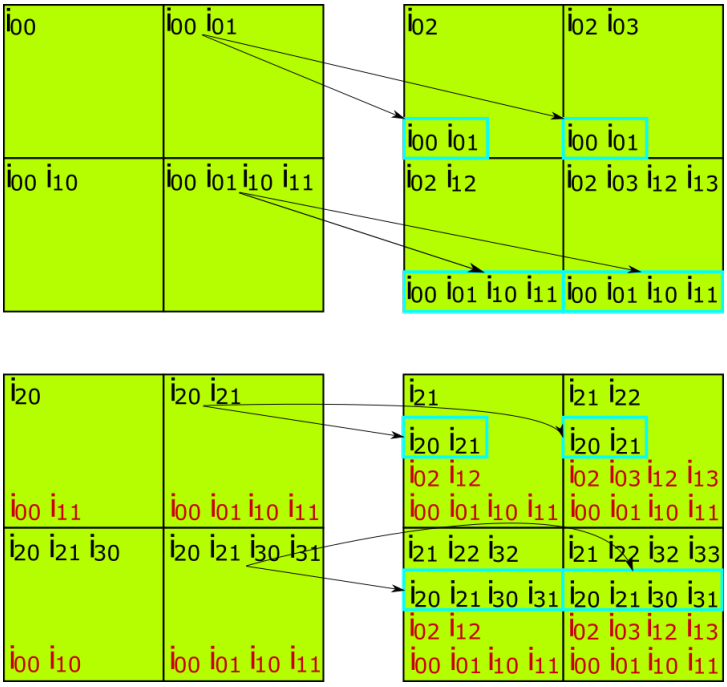


Figure 4.3: SAT Table after the second step of the algorithm

Now the top right submatrix now contains the final values, which are those that a global SAT application would yield. The bottom left submatrix is left unchanged, whereas the bottom right one is closer to the final values, but is not yet there. The same sums as before are made, but this time, vertically. Figure 4.4 shows the final result.

Of course, the algorithm can be applied iteratively as many times as needed. Suppose there were other three 4 by 4 matrices, aside and below this one, and suppose also that those matrices had gone through the same steps carried out on this one. At this point, the same horizontal and vertical passing of the values on the cells on the boundary would produce the same results.

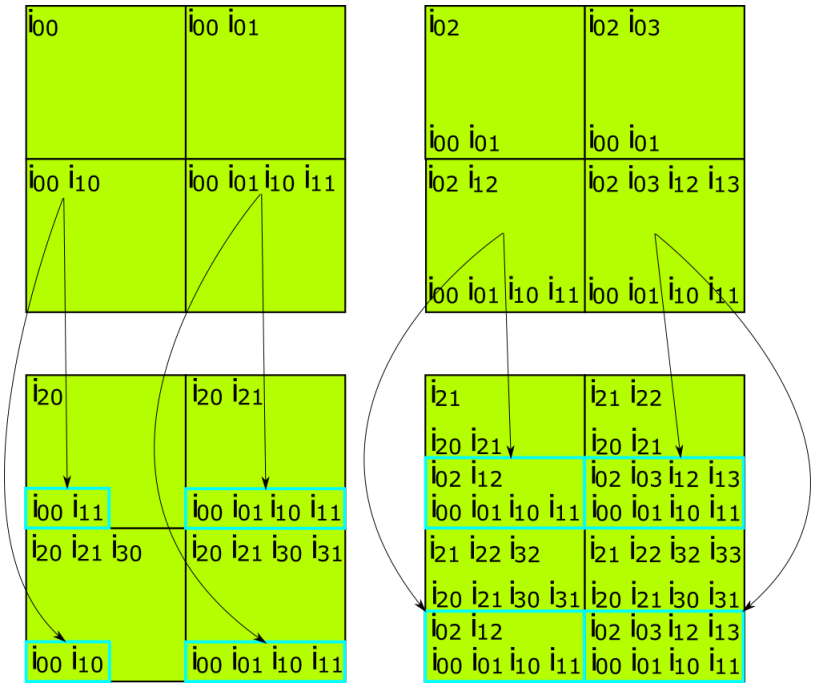


Figure 4.4: SAT Table after the last step of the algorithm

4.1 Datapath Hardware

In this section the hardware needed to implement a basic 2 by 2 matrix able to perform all the steps of the SAT algorithm shown above is presented. The bit parallelism is four, because the adder chosen (see subsection 4.1.2) is a ripple carry one, and a greater number of bits would have excessively slowed it down. On top of that, such a small number of bits makes easier checking whether the outcomes of the operations are right. Once the architecture structure is tested, the number of bits can be chosen at will.

4.1.1 Memory

The first thing needed is some sort of memory. It holds the values of the four cells, outputs them when needed, and takes in new values, either from the external inputs or from local calculations. Its layout can be seen in figure 4.5. It can be noted that it has changed a little with respect from the one presented in chapter 3. The memory has the decoder placed in the middle (in the red box). This makes the footprint of

the circuit more square-shaped, and cuts in half the long horizontal domain walls delivering the control signals, which reduces the total critical path. Furthermore, the cells now contain two notches each, making the general handling of the memory easier.

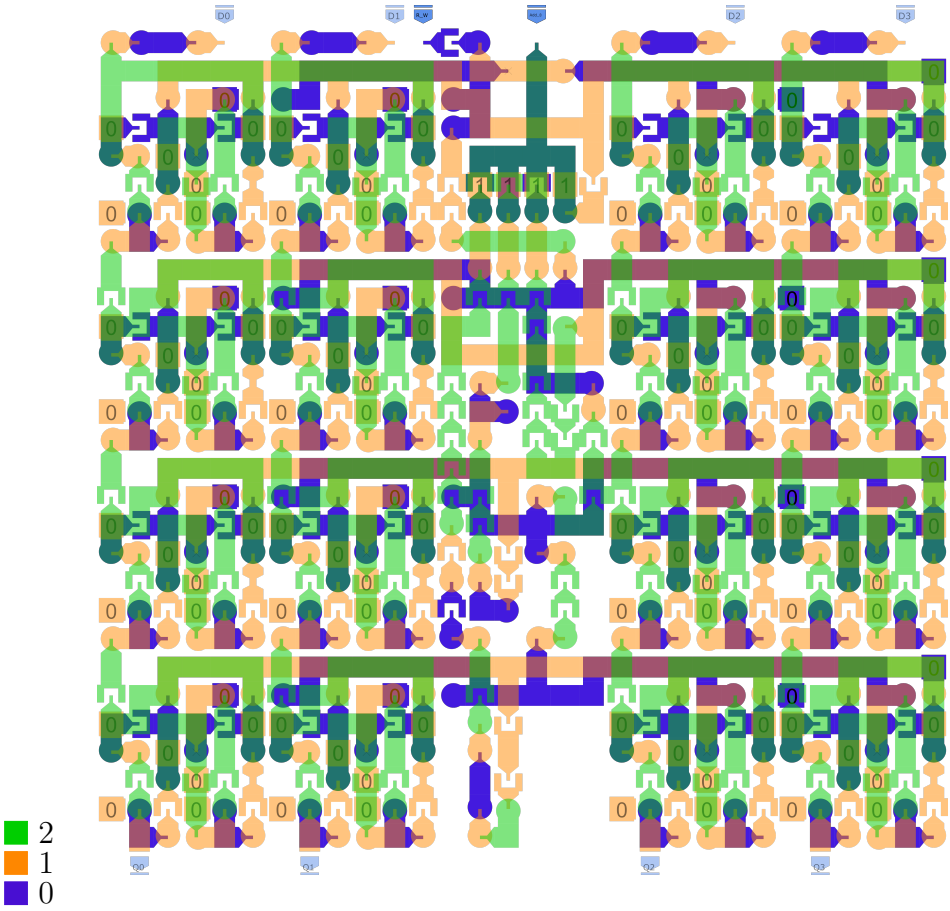


Figure 4.5: Memory element, it is able to store four values of the matrix seen in figure 4.1, or one of the four 2 by 2 matrices in figure 4.2

4.1.2 Adder

The algorithm requires some adding up of numbers; hence, an adder is necessary. In order to keep the design simple, the type of adder chosen is a ripple carry adder. On one hand, it is quite easy to design in a majority voter based logic. On the other hand, its delay is not excessive, given that the bit parallelism is four bits. More effective solutions can be analyzed later on. The circuit can be seen on figure 4.6. On the right there is the only half adder.

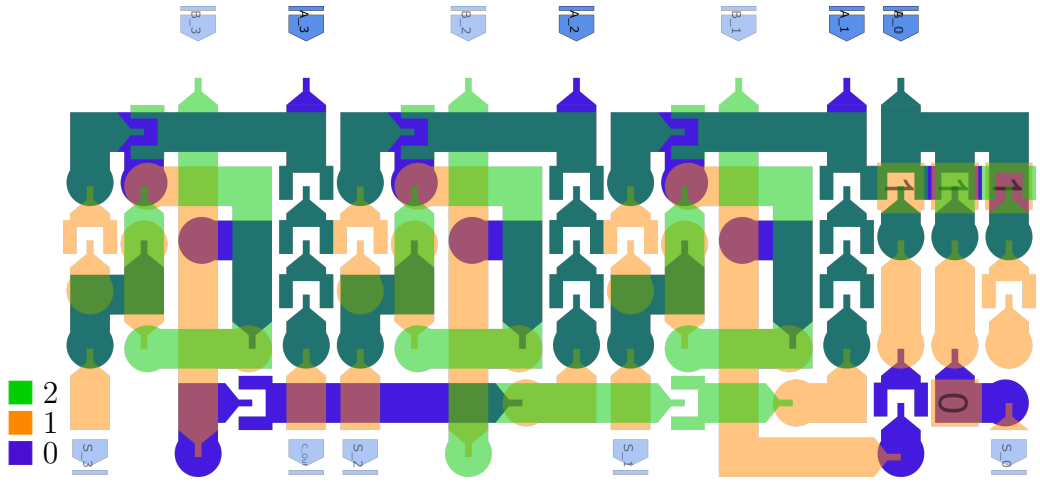


Figure 4.6: Full Adder used in the SAT circuit

4.1.3 Full Datapath

The union of the adder shown in subsection 4.1.2 and the memory seen in subsection 4.1.1 makes all the datapath hardware needed to run this SAT algorithm. Of course, the two building blocks are equipped with some adapting hardware. In more detail, the adder becomes an adder-accumulator, and now is able to store the result and, if needed, use it as input for the next sum. A multiplexer is now at the input of the memory part, switching between input coming either from the outside or from the adder. Last, the control signals that handle the reset and switching of inputs and memories are added. The datapath is pictured in figure 4.7.

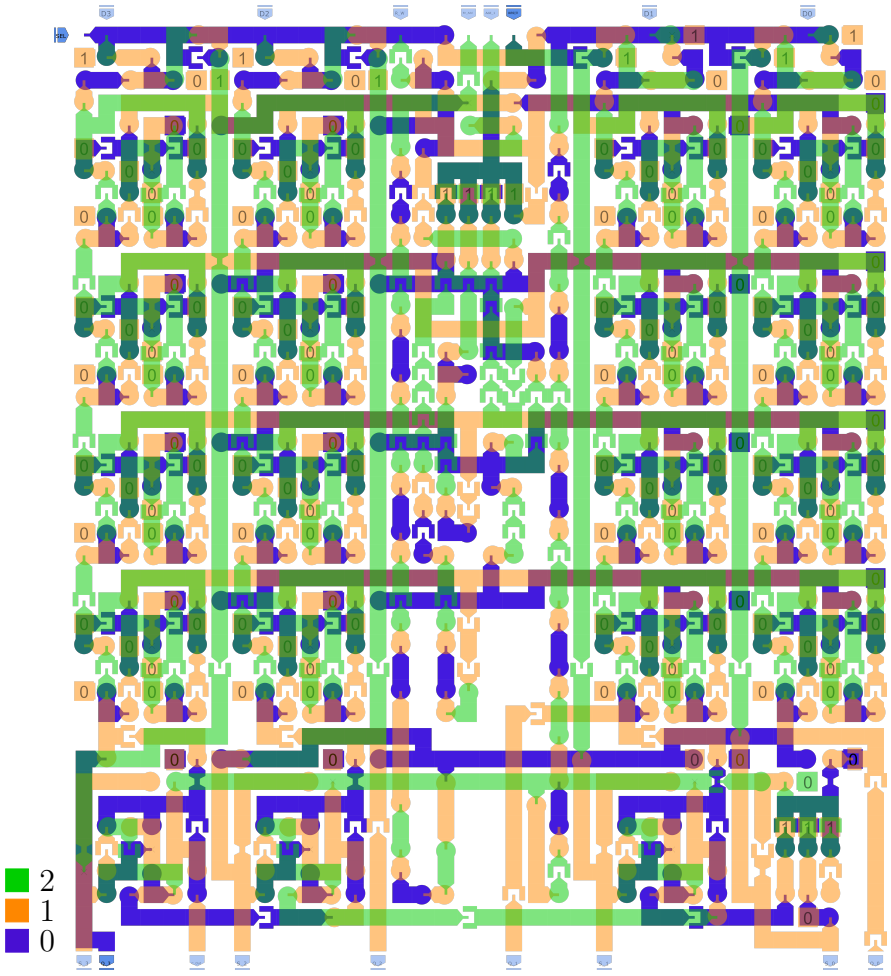


Figure 4.7: Datapath for the SAT algorithm

The circuit exhibits some technologically incorrect elements, namely inverters

right above or below domain walls. In this case, this has been done on purpose after having designed the circuit, that was originally technology compliant. It has been done because, once one achieves a working circuit, he or she might be interested in assessing how the critical path would change when trying to shorten the wires. A new design of it sometimes is not worth it: if the delay reduction is not relevant, quite a lot of time would be wasted. This is what was meant in section 3.1 by the "inverter insertion".

4.2 Control Hardware

The hardware seen in section section 4.1 is able to carry out the SAT algorithm over a cell, but it is still lacking the control part. In principle, a single finite state machine would be needed; however, given that all the hardware here is hand-crafted, it is much easier to deal with the complexity when there are multiple, simpler FSMs rather than a single one much more complex. This choice will turn out to be more flexible too.

Hence, there is an FSM for each step of the algorithm, namely:

- RESET: The initial values are placed into the memory locations;
- LOCAL: The 2 by 2 matrix carries out the SAT algorithm over the 4 locations;
- INPUT: The 2 by 2 matrix takes in the values calculated by an adjacent matrix, and sums it to the local contents, either in horizontal or in vertical direction;
- OUTPUT: The 2 by 2 matrix places at its output the values of specific cells, for the other cells to update their values.

Four operations are identified; nonetheless, the INPUT and OUTPUT step can be performed in two different directions, with different cells involved. Hence, six FSMs should be needed: however, the choice made consists in designing only four FSMs, with a combinational network that switches the values of the outputs when needed. Simplicity reasons caused the two FSMs to be designed as Mealy machines.

This is usually extremely harmful and should be avoided; nonetheless, here we are in total control of the data running through the architecture, and signals going outside of the circuits are known. Anyway, several notches are placed within the circuit; luckily, this completely overcomes the issues of a Mealy machine.

4.2.1 Control Signals

Five signals are enough to control the datapath. They are:

- *R/W*: Read/Write. The input datum (either coming from the outside or from the adder) is to be stored into the memory according to this signal;
- *Sel*: Selection. The input datum for the memory comes either from the outside or from the adder, depending on the value of this signal.
- Add_0,Add_1: The two address bits, both for the read and for the write mode;
- *En Add*: Adder Enable. When high, the adder sums the value coming from the memory to the previously stored result;
- *Reset*: Adder Reset. It clears the content of the adder's accumulator when a new addition chain has to be started

These signal are all "active high". Therefore, they can be just OR-ed all together before reaching the datapath: as long as just one FSM is active at once, the datapath is driven correctly. Thus, it is of the utmost importance preventing FSMs from running at the same time.

4.2.2 Reset FSM

This finite state machine's purpose is to reset the memory, meaning that the initial values to be processed are fed to the memory. Of course, if need be, the values can be all zeros, performing this way a "regular" reset, which might be useful at testing time. The FSM is made with the already mentioned strategy: every state has its own notch, and each notch is placed on a different layer. The FSM is also quite compact, due to its simpleness and to the fact that no condition is evaluated to pass to the next state, and it has just one input, which is the reset. This way, it will start

as soon as the *Reset* signal is pulled down, and stop, regardless of its current state, when the *Reset* signal is pulled up again. Figure 4.8 is a picture of the FSM. From

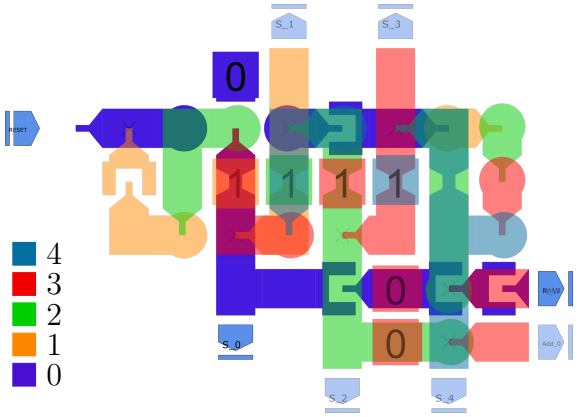


Figure 4.8: *Reset FSM* for the SAT algorithm

the point of view of the layout, it can be seen here that the spacing between notches of consecutive states is narrower than in section 3.9. There, it was two squares, here the notches are adjacent. The main reason is that in this FSM (and in all but one of the others in this chapter) no condition is evaluated before moving to the next state. This makes considerably simpler the next state network, resulting in a more compact one.The state dataflow diagram is shown in figure 4.9. As previously said, it is sensitive only to the *FSM Reset* signal. The outputs signals of the FSM are

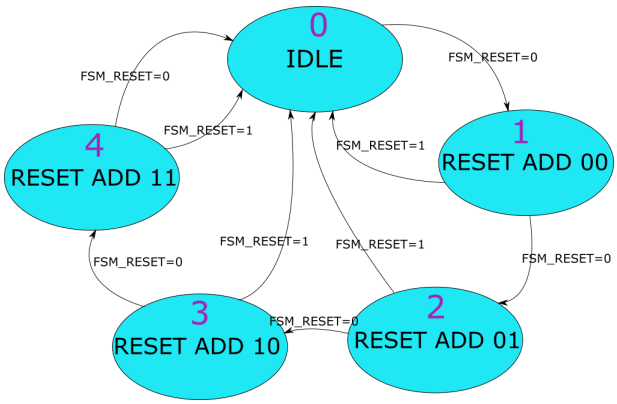


Figure 4.9: State flow chart of *Reset FSM*

shown in table 4.2.

Signal	Logic value				
	State 0	State 1	State 2	State 3	State 4
R/W	0	1	1	1	1
Add_0	0	0	1	0	1
Add_1	0	0	0	1	1

Table 4.2: Outputs of the *Reset FSM*

4.2.3 Local FSM

This finite state machine performs the SAT algorithm among the four values of the cell. The design style is the same as the previous, and all the others, FSMs. It is made of 12 states, always passing to the next one without conditions evaluated, and driven only by the *Reset* signal. The FSM is pictured in figure 4.10.

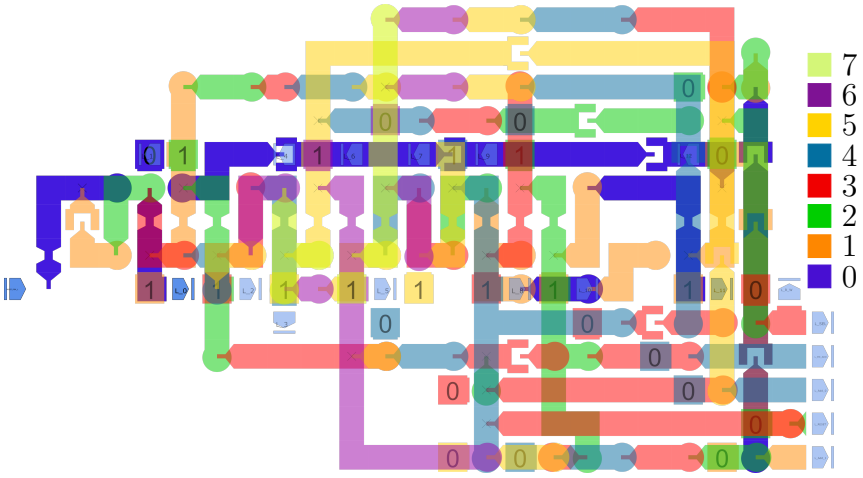


Figure 4.10: *Local FSM*, that drives the SAT algorithm in a single cell

In figure 4.11 is represented the state dataflow diagram of the FSM, whereas table 4.3 shows the outputs' configuration.

4.2.4 Input FSM

This finite state machine writes the values coming from another cell into the proper locations of its own cell. This FSM does not evaluate any conditions through states, however, it issues an output signal called *Go*, needed to trigger the FSM described

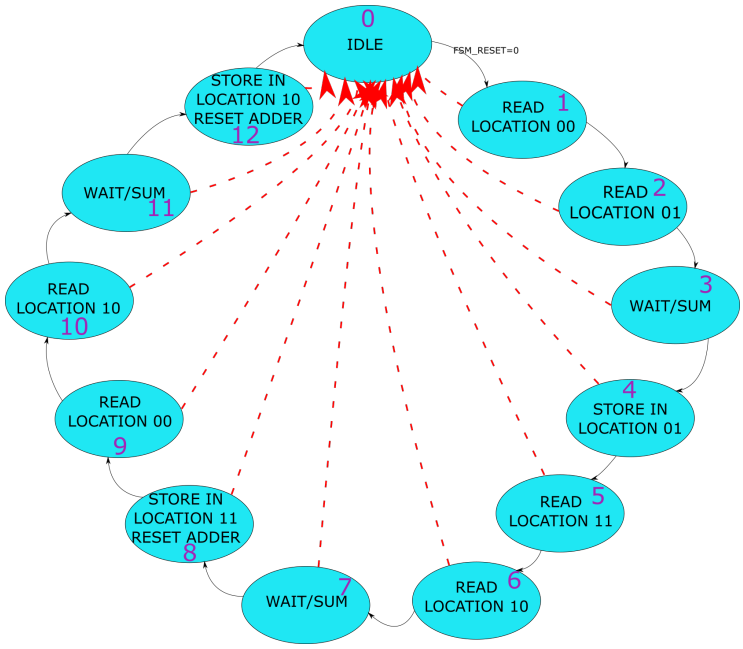


Figure 4.11: State flow chart of *Local FSM*. Red dotted lines indicate that, from any state, if signal *FSM Reset* is high, the FSM is set back to the *IDLE* state

State	Output Signals					
	R/W	*Sel*	Add_0	Add_1	*En Add*	*Reset*
State 0	0	0	0	0	0	0
State 1	0	0	0	0	1	0
State 2	0	0	1	0	1	0
State 3	0	0	0	0	0	0
State 4	1	1	1	0	0	0
State 5	0	0	1	1	1	0
State 6	0	0	0	1	1	0
State 7	0	0	0	0	0	0
State 8	1	1	1	1	0	1
State 9	0	0	0	0	1	0
State 10	0	0	0	1	1	0
State 11	0	0	0	0	0	0
State 12	1	1	0	1	0	1

Table 4.3: Outputs of the *Local FSM*

in subsection 4.2.5 (*Output FSM*), that will move into the next state. Therefore, *Input FSM* and *Output FSM* always work in pairs and at the same time. Every

time an *Input FSM* is running on a cell, an *Output FSM* is running in the cell to its left or above. That's why they need synchronization signals. This is certainly the most articulated FSM. Basically, two FSMs are in place: one (referred to as the master FSM) phases the operations that allow the input data to be stored, summed to the local ones, and then stored; the other one (the salve FSM) defines the address. Hence, for every run of the first FSM the second one changes address. Moreover, a XOR network takes as input the address bit, so that they can be inverted when needed, in order to distinguish between an horizontal and a vertical data transfer. Figure 4.12 shows an overview of the FSM, with two vertical lines drawn to identify the three components (left the one that drives the algorithm, center the one that defines the addresses, right the XOR network). This is a Mealy Machine: State 0 is both the "IDLE" state and the first active state. If the signal *FSM Reset* (the main input reset for this FSM) is high, all the outputs are zero, otherwise the outputs' state is the one shown in column "*State 0*" of the table 4.4.

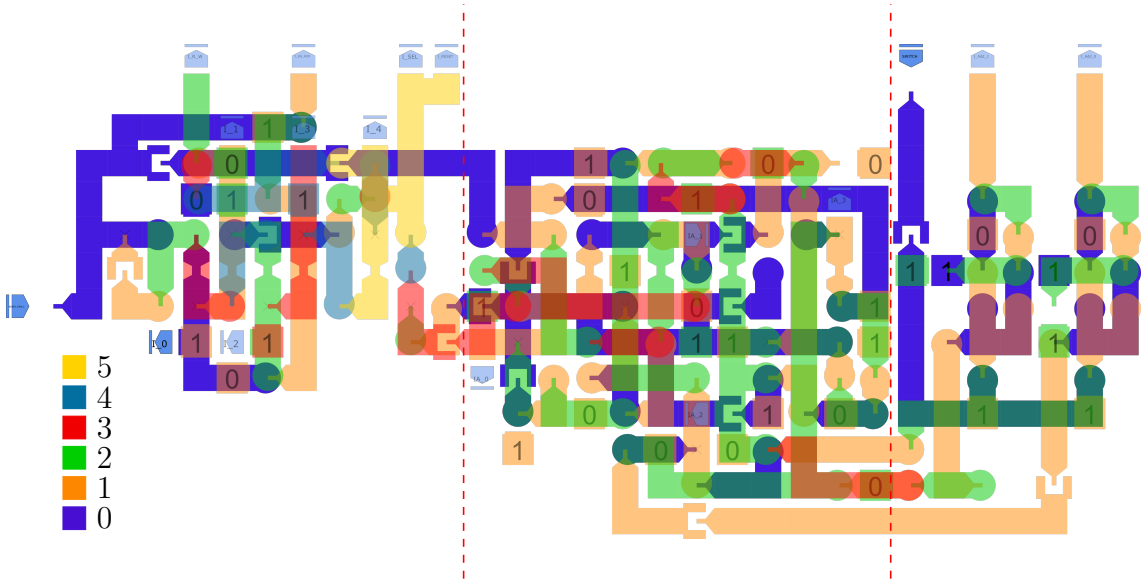
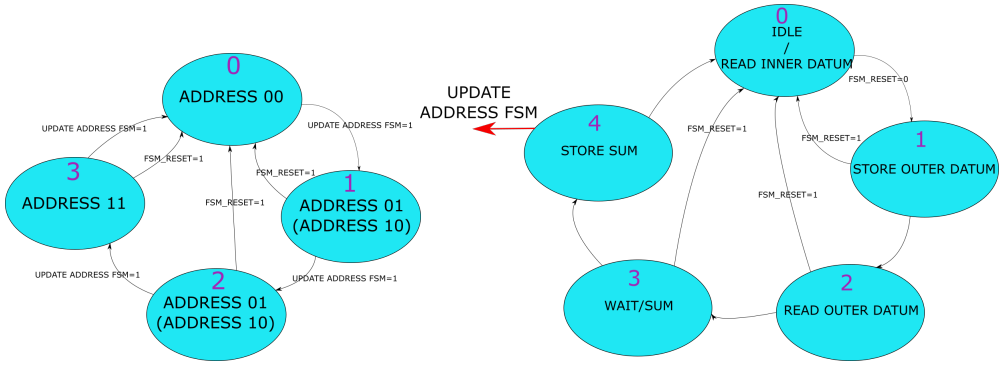


Figure 4.12: *Input FSM*, red lines drawn to distinguish the three components of it

Figures 4.13 show the State flow chart of this FSM. Notice that this is just a logical representation: the circuit is only one, even if, as it can be seen in 4.12 the two parts are clearly defined and easy to identify.

Table 4.4 shows the outputs for each state. Addresses are not indicated, for they



(a) *"Address" FSM*, addresses in brackets are the ones outputted when signal *Switch* is 1

(b) *"Main" FSM*

Figure 4.13: The two "logical" *Input FSM*

are clear enough in figure 4.13.

Signal	Logic value				
	State 0	State 1	State 2	State 3	State 4
R/W	0	1	0	0	1
Sel	0	0	0	0	1
En Add	1	0	1	0	0
Reset	0	0	0	0	1

Table 4.4: Outputs of the *Input FSM*

4.2.5 Output FSM

This finite state machine brings out of the cell the value to be passed to an adjacent cell. It is the FSM with the lowest number of states, but also the only one needing a signal to step to the next state. The signal, called here *Move*, is matched with the *Go* signal coming from the *Input FSM* described in subsection 4.2.4. A picture of the FSM can be found in figure 4.14, its flow chart is in figure 4.15. A vertical, dotted, red line is drawn in the middle of the picture. On its left there is the pure FSM, while on its right there is the XOR network that switches between the vertical data transfer and the horizontal data transfer. Notice that this is the only FSM that evaluates conditions before moving to the next state, and is also the only

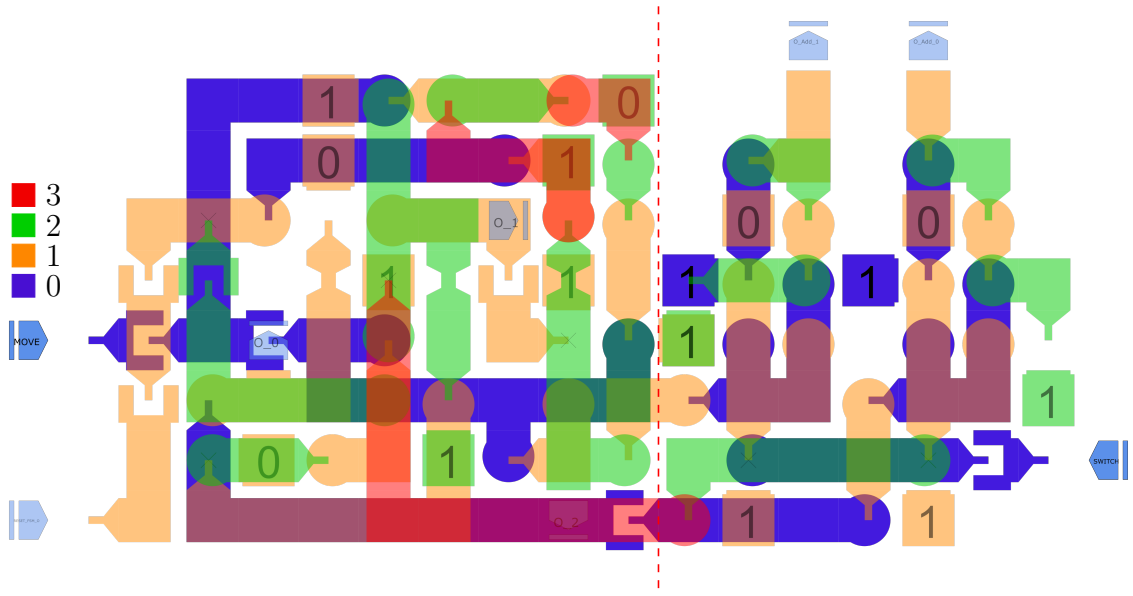


Figure 4.14: *Output FSM*, a red line marks the boundary between actual FSM and XOR network

one that does not have the notches adjacent. However, they are closer than the ones of section 3.9.

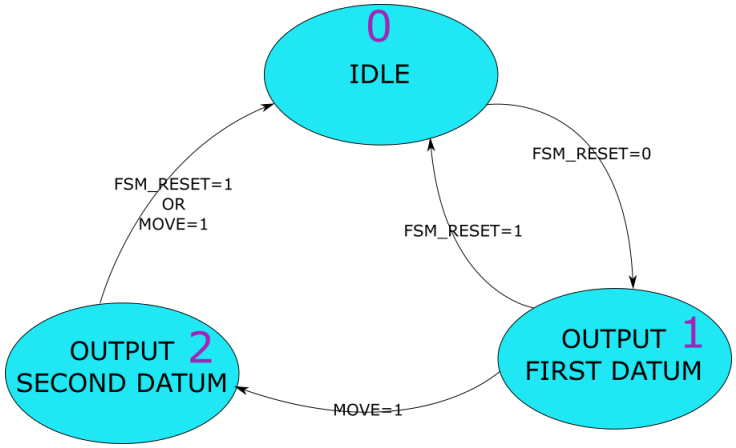


Figure 4.15: State flow chart of the *Output FSM*

4.2.6 Vertical/Horizontal data transfer

The description in subsection 4.2.4 and subsection 4.2.5 mentioned the presence of a XOR network switching the outputs, and provided the values of the outputs, but did

Signal	Logic value ($Switch = 0$)			Logic value ($Switch = 1$)		
	State 0	State 1	State 2	State 0	State 1	State 2
Add_0	0	0	1	0	1	1
Add_1	0	1	1	0	0	1

Table 4.5: Outputs of the *Output FSM*

not elaborate anything in further detail. Here the matter is elaborated a little. The addresses are assigned to the four cell's locations from left to right and from top to bottom, starting from 00 in the top left location, as shown in figure 4.16. Figure 4.25 (third and fourth image) and figures 4.3 and 4.4 shows how the transfers between neighboring cells are done, and table 4.6 list the addresses. Consider the column

	Starting location	Ending location
To the cell on the right	01	00
	01	01
	11	10
	11	11
To the cell below	10	00
	10	10
	11	01
	11	11

Table 4.6: Addresses when data are exchanged between cells

Starting locations: row 1-2 and 5-6 are one the bit-wise inversion of the other. Then, consider the other column: row 2-3 and 6-7 are again equal after bit-wise inverting one of them. This is exactly why having a two input FSMs and two output FSMs did not sound worth it. Inverting the value of some addresses will be enough, and this is the point where the choice of using many subFSMs rather than a single one pays off. A XOR gate is placed after the Add_0 and Add_1, and is driven by a specific signal called *Switch*. This network makes the two FSMs, *Input FSM* and *Output FSM*, Mealy machines. However, the notch stages added for control reasons overcome all the possible side effects of this kind of finite state machines. This trick is also the reason why a 2 by 2 cell is really convenient: inverting the bits is not complicated. Had the cell been a 4 by 4, rearranging the bits of the addresses would not have

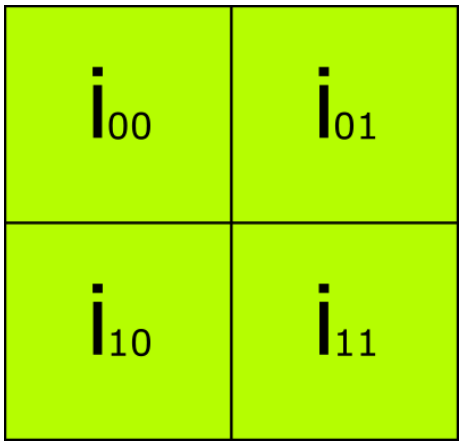


Figure 4.16: Address assignement within a cell

been that easy.

4.3 2 by 2 cell

Once all the circuitry described in section 4.2 is ready, the cells can be built. As previously mentioned, OR-ing all the corresponding outputs of each FSM is enough. These control signals are eventually placed as inputs of the datapath. Remember that no signal coming from the datapath is needed to run the FSMs. This cell alone is not able to deliver useful calculations besides the SAT among its four elements, so that only two of the FSMs are put to work: the *Reset FSM* and the *Local FSM*. On the other hand, when arranged with other similar cells, they can all together carry out a SAT algorithm among a number of cell greater than 4. Here, in figure 4.17, is placed a representation of the three bottom layers, composing the datapath; figure 4.18 contains a view of the four FSMs OR-wired together, making up the whole control circuit. Unlike figure 4.7, now all the pipeline stages (in hardware, notches) have been placed. Once again, the circuit is not perfectly compliant with the technological requirements, both fixed-one and fixed-zeros have been used, and some inverter over or under a domain wall might be found

This circuit is a good example of the space-saving chances offered by the layer stacking: supposing that no more than three layer had been allowed, and that control and datapath circuits were roughly the same size, the area occupation would have

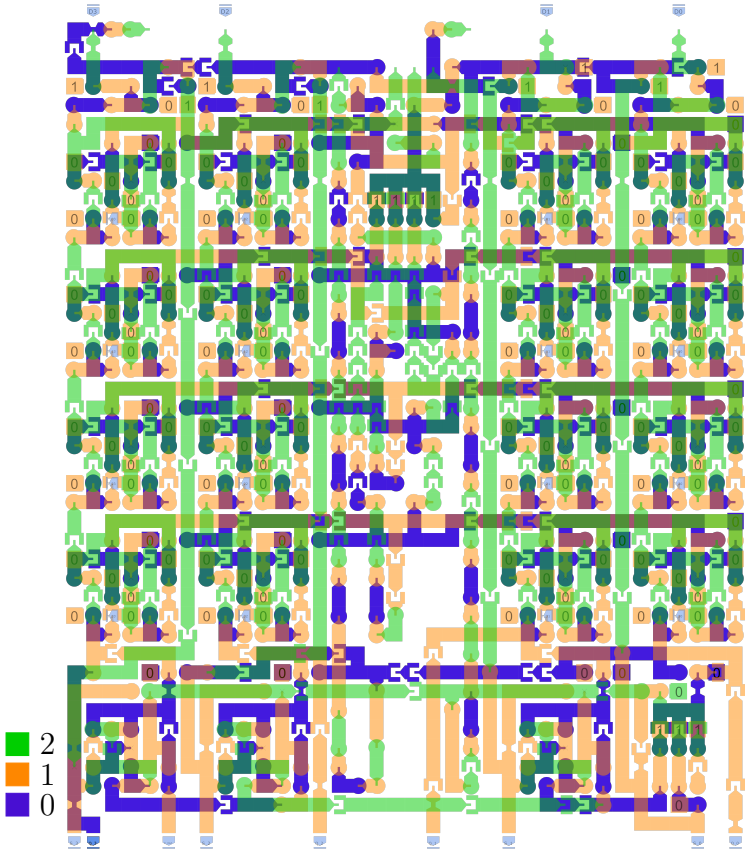


Figure 4.17: Basic cell datapath, made on the three bottom layers

been approximately three or four time as much. The fourth layer (counting from the bottom) is just a link-layer, meaning that the datapath lies on 0 to 2 layers, and control on layers 4 to 11. The reason, pretty straightforward, is that unwanted magnetic coupling between datapath and control circuit has to be avoided.

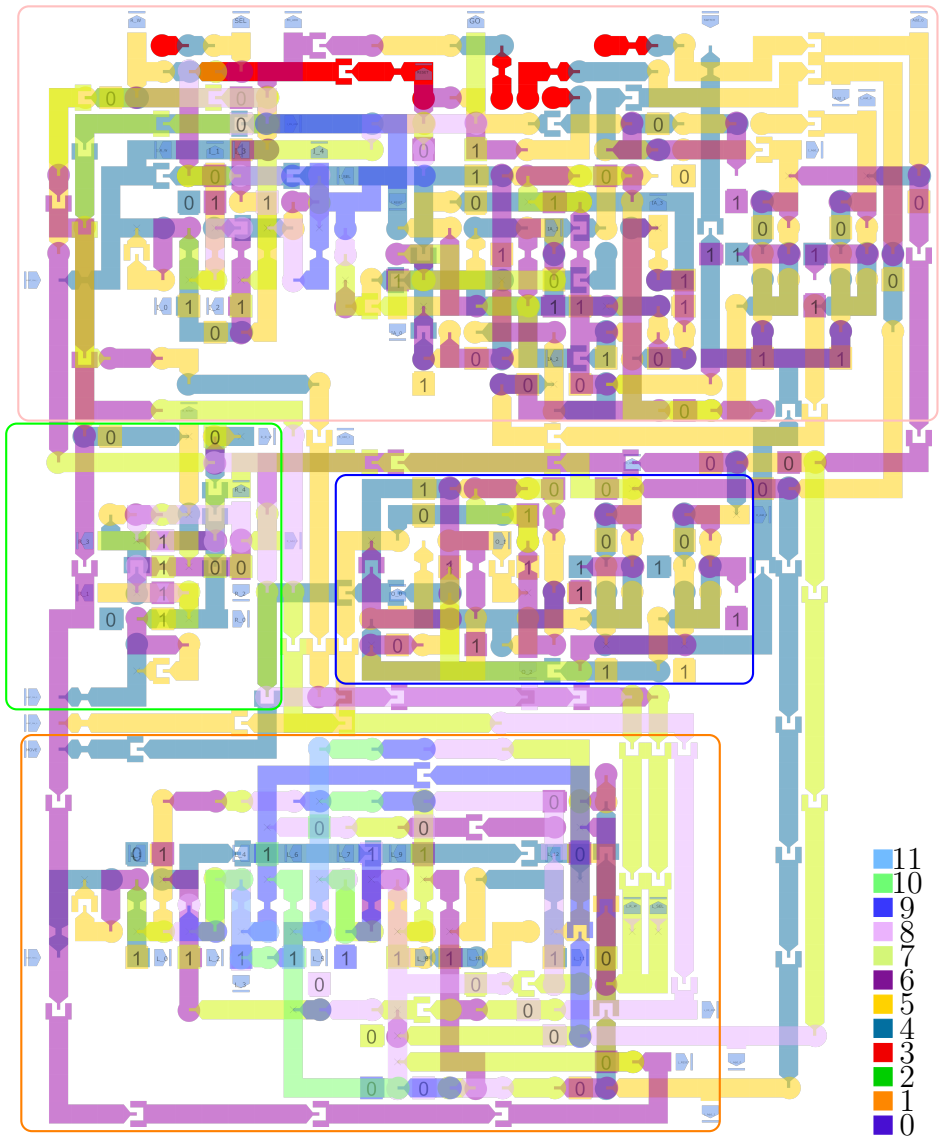


Figure 4.18: Basic cell control, made on the seven top layers. Pink rectangle marks the input FSM; green one marks the reset FSM; blue one marks the output FSM; orange one marks the local FSM

4.4 Putting cells together

When the number of locations is greater than four, more than one cell is needed. Of course, these cells need to exchange some of their values, and this implies that some sort of synchronization/coordination must be be arranged. In other words, the control problem must be tackled.

4.4.1 Hierarchical control

A first control style might be a hierarchical one. The whole matrix is parted into four sub-matrices. These four sub-matrices will carry out a certain amount of calculations, **always involving only locations internal to the sub-matrix**. At the end, a vertical and a horizontal (just one for every direction) transfer of data completes the process. If, after the 4-part split of the main matrix, the sub-matrices are not 2 by 2, they are iteratively split exactly the same way, until a "minimum block", namely, a 2 by 2 cell, is reached. The main steps of the dataflow are:

1. Split the matrix in four sub-matrices;

2. Calculate the local SAT within the four sub-matrices. If their size is not 2 by 2, split them again and calculate the local SAT for each sub-matrix. To calculate it, follow this same dataflow. If the size is 2 by 2, calculate the SAT and carry on with the algorithm;

3. Perform a horizontal and a vertical data transfer. Now the matrix split in four at point 1 has its local SAT calculated.

The dataflow above needs as precondition the ability to calculate the SAT for a 2 by 2 matrix, which is exactly what a cell is able to do. Nevertheless, from a circuital point of view, this dataflow is very unconvenient. The reason lies in wiring problems. Have a look at figure 4.19. The matrix is larger than 2 by 2. It has to be split. Figure 4.20 shows that. Sub-matrices are larger than a 2 by 2 size too. A further (and final) split needs to be done. Figure 4.21 shows the outcome. Each one of the red matrices are 2 by 2, so they fit exactly in an hardware cell and the local SAT algorithm is easily carried out. Then, a local SAT is done involving the matrix made by the four red 2 by 2 matrices: all one has to do is to perform a vertical and horizontal data transfer between the four blocks. Of course these same steps are taken at the same time in the other cells. Once the four 4 by 4 matrices have their local SAT ready, they should perform the well-known vertical and horizontal data transfer in order to conclude the algorithm. Now, consider the blue cell in

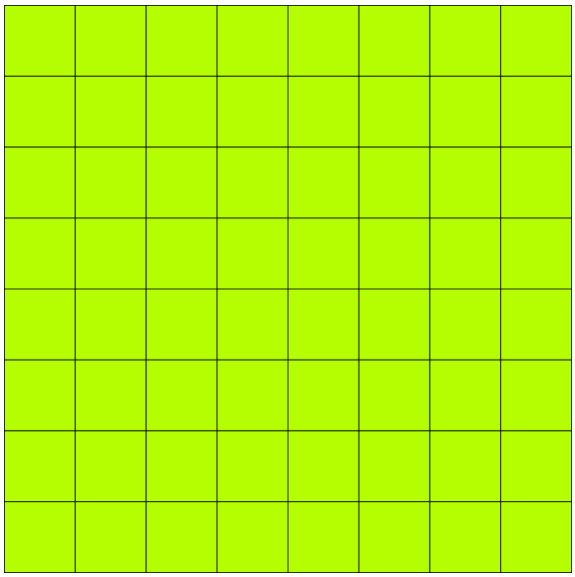


Figure 4.19: A generic matrix that will undergo SAT algorithm

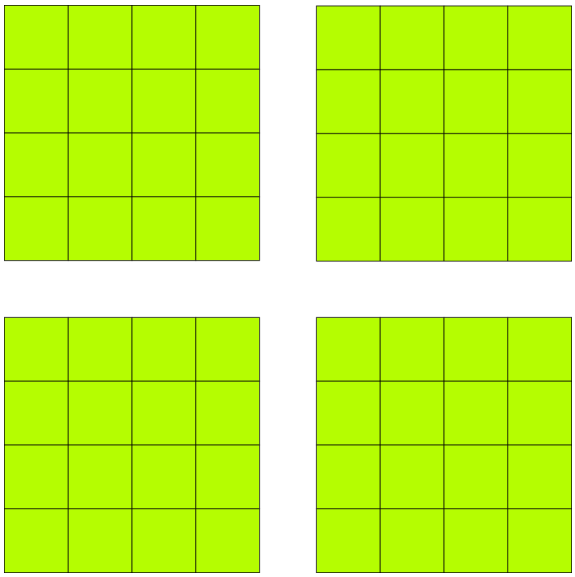


Figure 4.20: Matrix of figure 4.19 after the first split

figure 4.22. In the last horizontal data transfer, it needs data from the border cells of the top left 4 by 4 matrix, and precisely from the green locations. There are two options to do this:

- A direct wire between the locations involved;

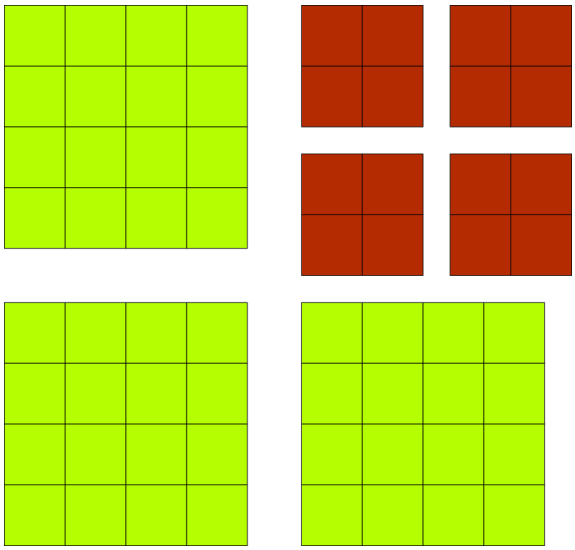


Figure 4.21: Matrix of figure 4.19 after the second split

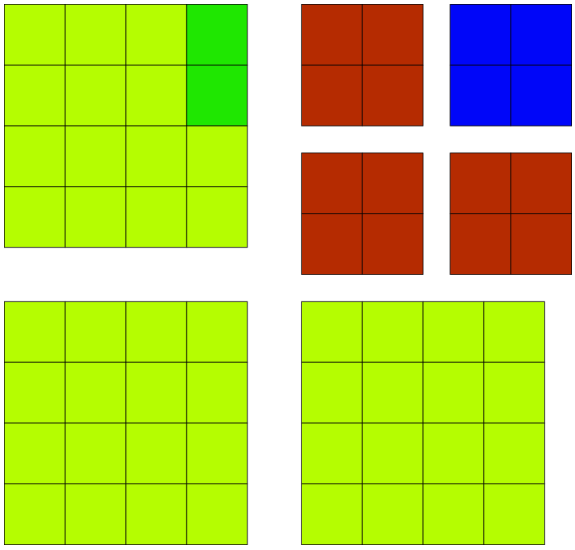


Figure 4.22: Matrix of figure 4.19. Marked locations are the ones involved in the data transfer currently under consideration

- The locations in between must be able to pass the values from green locations to the red ones.

The first solution implies a huge amount of wiring (consider a case with much more locations than in this one, the problem shows up iteratively). The second one might be put in place, but would give up all the benefits of such a hierarchical control.

Anyway, a second control strategy, which is the one used, exists.

4.4.2 "Flat" control

In this control style, there are only two levels: the one encompassing the four locations cell, and the one including the whole set of cells, with no further division. Briefly explained, it works like this: at every algorithmic step, one horizontal and then one vertical transfer are done. The first horizontal transfer encompasses the cells lying across the leftmost vertical border; the first horizontal transfer encompasses the cells lying the top horizontal border. At next step, the border involved will be the one just on the right (for the horizontal transfer) and just below (for the vertical transfer). Figure 4.23 and 4.24 clearly show how it works. This control style is probably less effective than the hierarchical one, at least from an algorithmic point of view, because it can be parallelized much less. However, for the reasons already said, it is less complex to cast in hardware. Nevertheless, should it be necessary to speed up the calculation, this implementation might be pipelined: after the first step,the content of the top left locations is no longer needed. If needed, it could be saved, and then a new value (belonging to a new matrix) might be stored. Anyway, this is beyond the scope of this thesis. There is just one point to make a little clearer: the example did not mentioned any grouping of memory locations; however, it can be shown that if cell are grouped in 2 by 2 blocks, and they perform a "local" SAT among those 4 locations, then they can go through the same steps shown in figures, and the final values will be correct. This is how it is done in the hardware presented in this section, and is briefly illustrated in figure 4.25.

4.4.3 Implementation

Four basic SAT cells have been gathered in order to arrange a 16-locations structure. The control step to take with such a tiny array are:

- Perform the local SAT on each of the four cells
- Horizontal transfer: North-Western cell delivers some of its contents to the North-Eastern one; South-Western cell delivers some of its contents to the South-Eastern one;

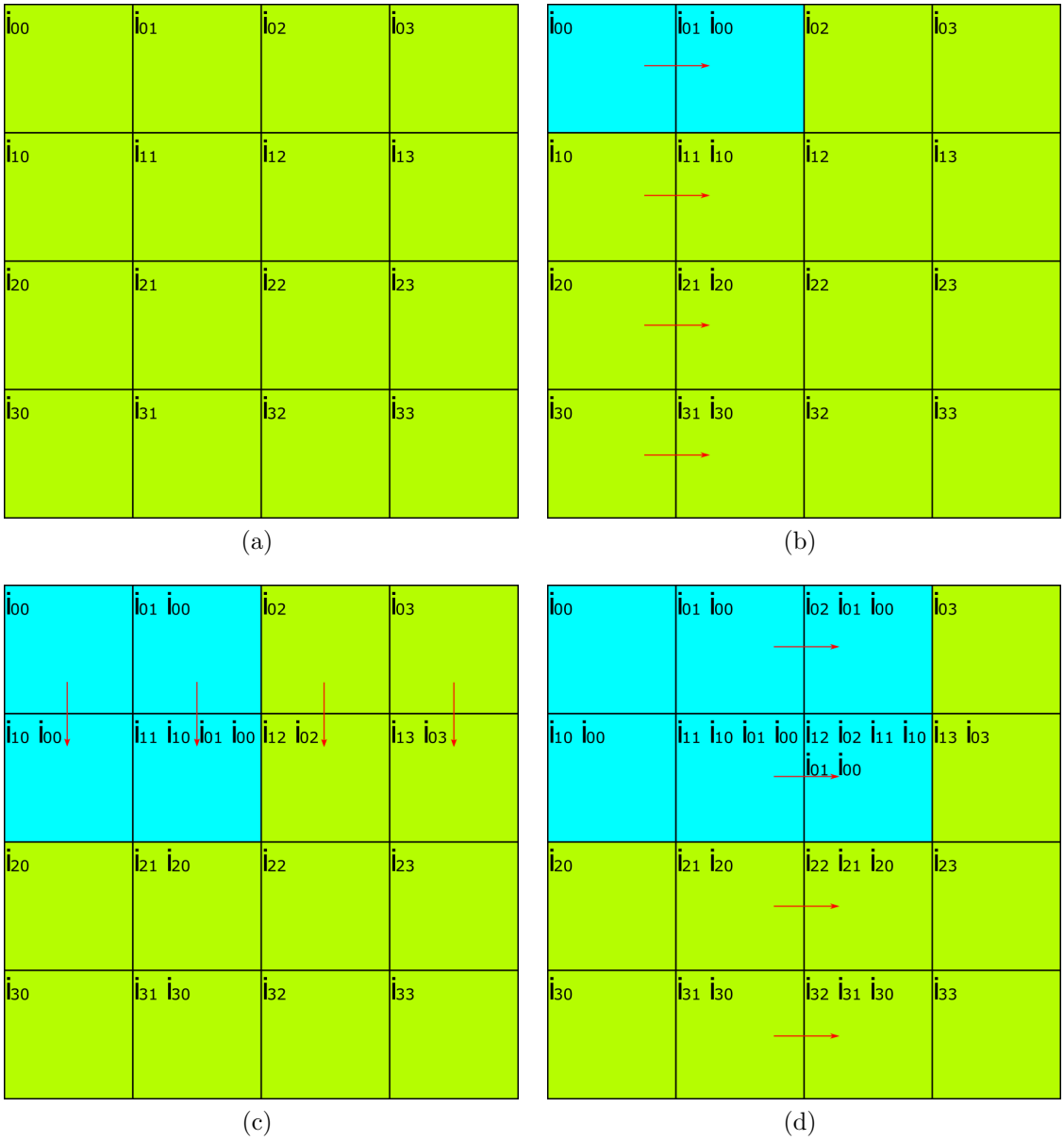


Figure 4.23: Example of flat control on a 4 by 4 array, first four steps. Light blue locations are the ones whose value is the final SAT value

- Vertical transfer: North-Western cell delivers some of its contents to the South-Western one; North-Eastern cell delivers some of its contents to the South-Eastern one;

Figure 4.25 clearly shows all these actions.

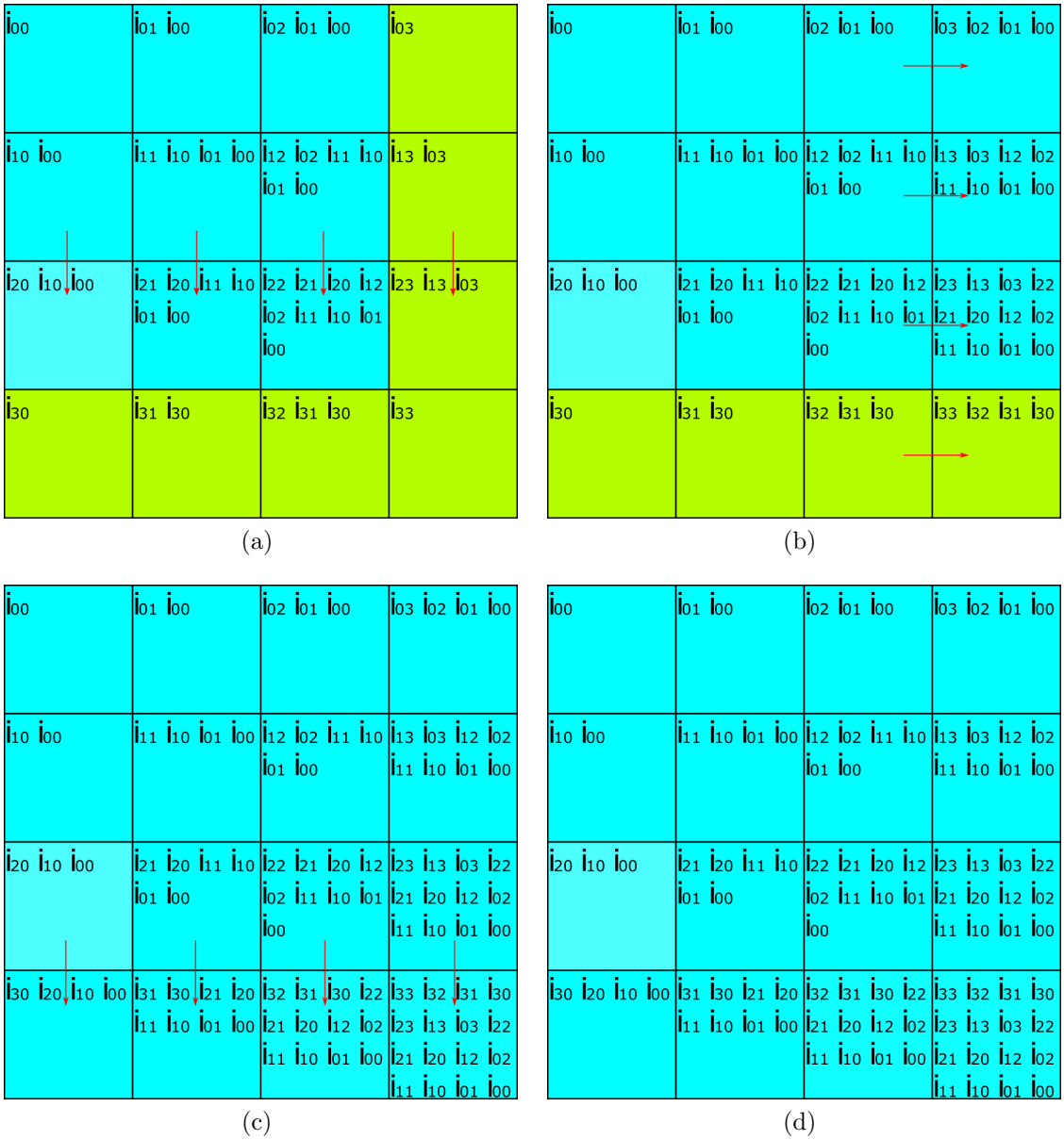


Figure 4.24: Example of flat control on a 4 by 4 array, last four steps. Light blue locations are the ones whose value is the final SAT value

Global FSM

A control circuit able to carry out the steps previously described must be designed, and figure 4.26 contains that very circuit. Figure 4.27 shows its state-flow diagram. This FSM might be referred to as "Global FSM" in future.

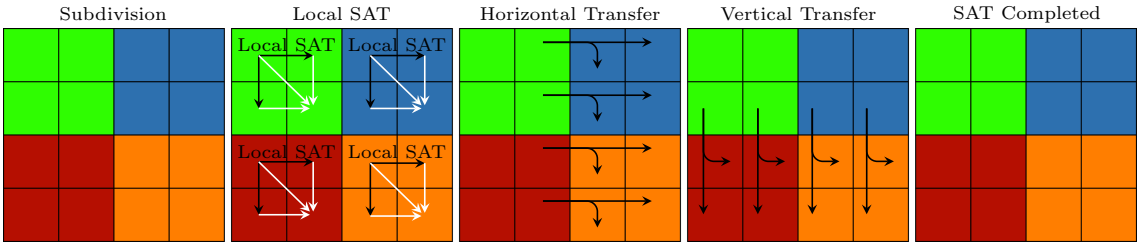


Figure 4.25: Steps of the algorithm as they are carried out in the designed hardware

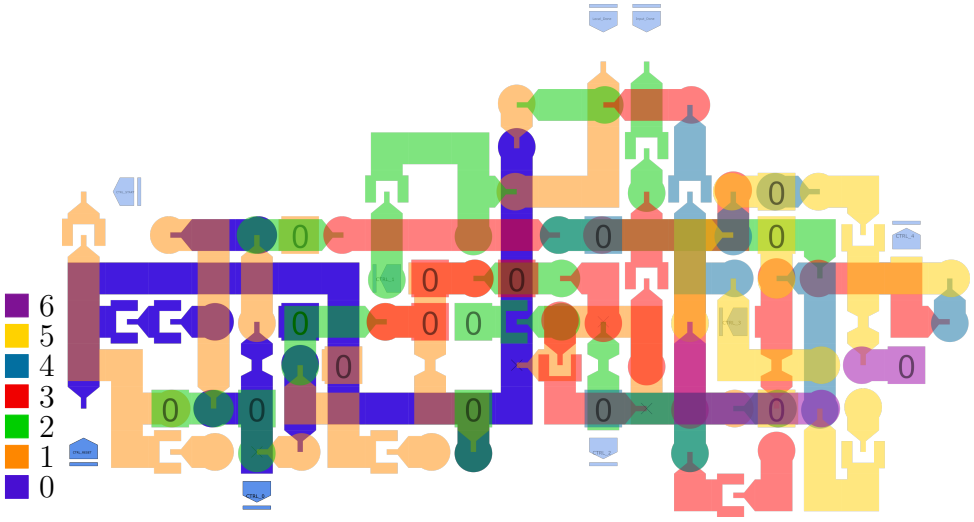


Figure 4.26: *Global FSM*, which controls the SAT over a 2 by 2 array of 4-locations cells

The FSM does not have any output network, because of its function. It defines five time-steps, and every time a FSM within the cells must be active in that time-step, the output of the state-notch in the *Global FSM* is connected to the *Reset* signal of that FSM (possibly OR-end, if more than one signal drives that FSM). The exact timing these signals are driven is shown in table 4.7. Table 4.7 unveils that the state number 2 is there just for timing reasons, and it is not an actual algorithmic step.

2 by 2 cell array

The circuit made up with four cells and the *Global FSM* is shown in figure 4.28. The four cells are placed as "components" in MagCAD, and this is why they are represented as black-boxes. Beware: a cell is made of four locations, so that a 2 by

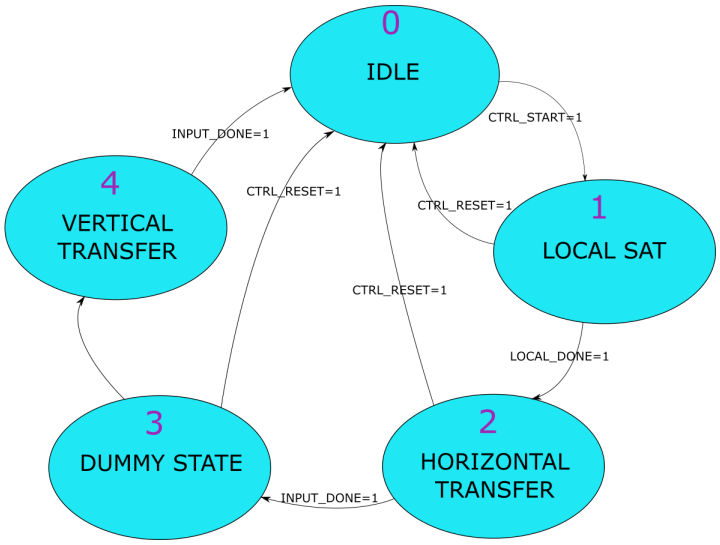


Figure 4.27: State flow chart of the Global FSM

Cell	Signal	State				
		State 0	State 1	State 2	State 3	State 4
North-West	*FSM Reset*	1	0	0	0	0
	FSM Local	1	0	1	1	1
	FSM Input	1	1	1	1	1
	FSM Output	1	1	0	0	0
North-East	*FSM Reset*	1	0	0	0	0
	FSM Local	1	0	1	1	1
	FSM Input	1	1	0	1	1
	FSM Output	1	1	1	1	0
South-West	*FSM Reset*	1	0	0	0	0
	FSM Local	1	0	1	1	1
	FSM Input	1	1	1	1	0
	FSM Output	1	1	0	1	1
South-East	*FSM Reset*	1	0	0	0	0
	FSM Local	1	0	1	1	1
	FSM Input	1	1	0	1	0
	FSM Output	1	1	1	1	1

Table 4.7: "Outputs" of the *FSM Global*. A "1" means that the particular FSM is reset, a "0" means it is running

2 cell array makes 16 locations (a 4 by 4 matrix). The circuit inside the black-box is the basic cell, made up with the two circuits shown in figure 4.17 and 4.18, stacked one on top of the other.

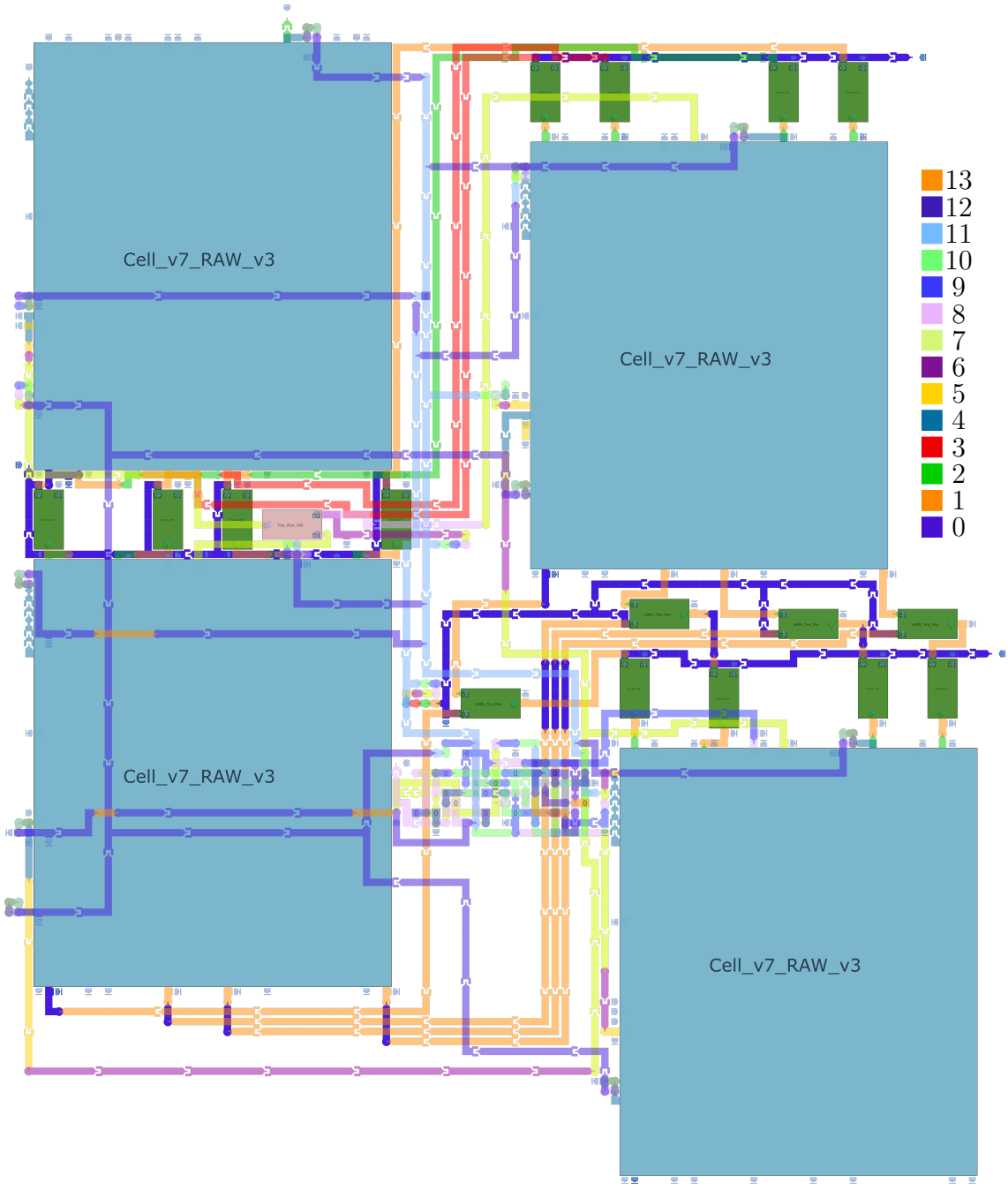


Figure 4.28: 16-location SAT hardware, made up with four 4-location cells

4.5 Use of notches

The arrangement of four cells has shown that sometimes timing must be rearranged by means of register insertions. In this case, their pNML equivalent, the notch, is used. It happens that some signals must be delayed by more than one clock cycle: this would imply the insertion of two notches in series. Figure 4.29 zooms on the specific area where these elements are placed. Two inverters are placed exactly in

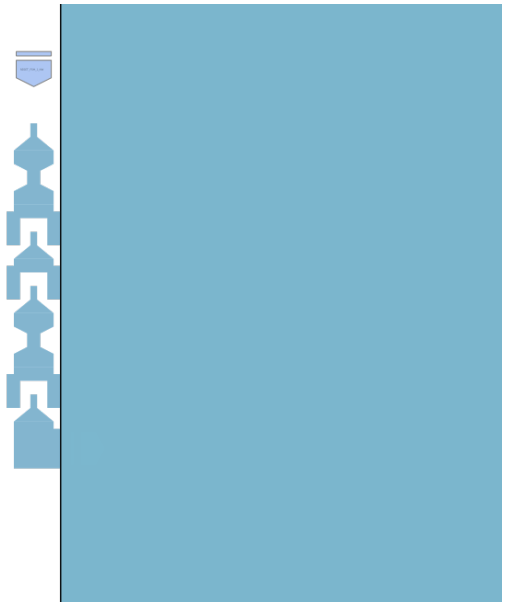


Figure 4.29: Detail of figure 4.28, with the two notches in series and two inverters in the middle

between the two notches. This is absolutely necessary, and it is proved by means of a couple of timing diagrams. Consider figure 4.31, which is referred to two notches in series *without* the pair of inverters in the middle. Refer to figure 4.30 for names.

Figure 4.30: Two notches in series. The signals evolution is shown in figure 4.31

In the example, the notch stays open for such a long time that the second notch too is updated. Since we use notches as registers are used in CMOS, this must be absolutely avoided. A possible solution is shown in figure 4.32. In this case, the *Depinning clock* stays high (that is, opens the notch) for a shorter time-span, just

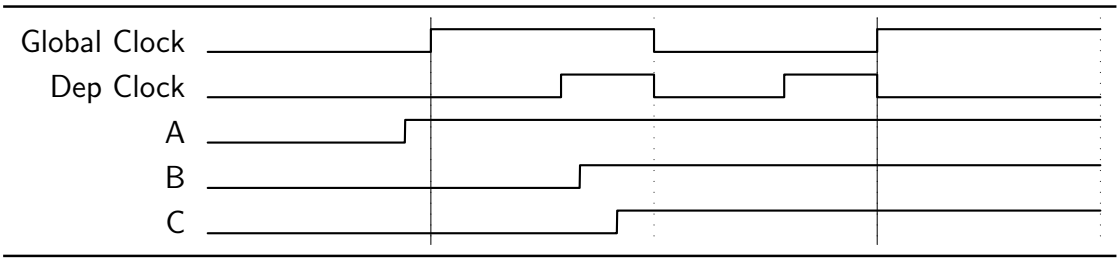


Figure 4.31: Example of signal passing thorough two notches in a single "notch cycle"

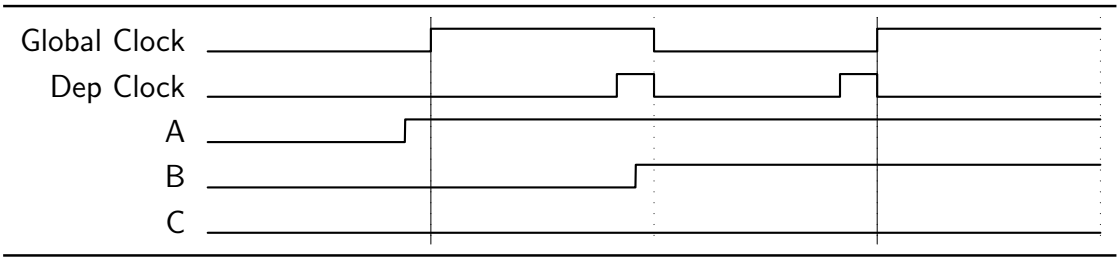


Figure 4.32: Possible solution to the problem of figure 4.31

enough for one notch to let the input propagate towards the output: as it can be seen in the time diagram, the second notch will have to wait for the next notch opening. This solution works and avoids the two inverters, which are not functionally needed and slow down the circuit by one clock cycle. Its only drawback is that it requires a very precise knowledge of the propagation delays of the circuits. Sometimes the designer does not have such a precise knowledge; moreover, delays have their own tolerances, and finding a suitable time length of that depinning clock pulse might be impossible. Using the couple of inverters is much easier and safer. The time diagram in figure 4.34 shows the exact timing of that solution. Names and signal are referred to figure 4.33. Notice that the notch is open for a long time, about half of the

Figure 4.33: Two notches in series with a pair of inverters in the middle. It solves the problems seen in figure 4.31

length of the clock pulse, and yet the second notch has to wait next notch period: before reaching the input of the second notch, the signal must propagate through the two inverters. This propagation needs two clock edges, so that when eventually

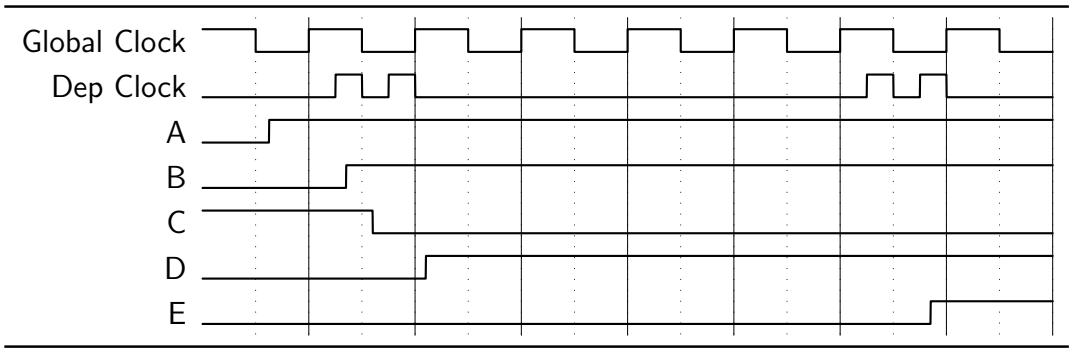


Figure 4.34: Solution to the problem of figure 4.31 with two inverters

the signal has reached the input of the second notch, the latter is no longer open.

Chapter 5

Performance Analysis of the SAT

There are a few motives for which the SAT algorithm has been selected to be tested on hardware. It is an incremental algorithm, that can be optimized in many ways, as the modular one implemented. It does not need complex hardware such as a multiplier, and works with limited word parallelism and number of words. It involves a matrix of memory locations, that might be mapped on an actual matrix of hardware cells able to send and receive values from the neighboring cells. And on top of that, there was another work [11], made in CMOS technology, that could provide interesting data to compare performances. In that work, among many other algorithms, the SAT one has been described in VHDL, simulated and synthesized.

5.1 Independent Design

All the SAT description is obviously the same for this work. The hardware implementation however is quite different. Let us briefly illustrate it. With reference to the two kinds of control approaches to deal with a large number of locations, this work is definitely a hierarchical one (in that work, the flat control is called "Inclusive Scan Algorithm", the hierarchical one "Balanced Trees Parallel Scan Algorithm"). At first the matrix is split in 2 by 2 submatrices, at next step 4 by 4 matrices are

considered, and so on until the algorithm is complete. Moreover, all these steps are assigned to different physical hardware planes, so that each plane gets the values to process from the plane below, computes its part of the algorithm and then passes the results to the plane above. Hence, each of these planes has to do with matrices whose size is always the same. The presence of planes promptly suggests that between everyone of them a pipeline stage could be placed; this is exactly what has been done. The architecture is described in VHDL and is fully parametric: the code takes care of everything, and all it needs in input is the number of bits of the word, the number of columns of the SAT array and the number of rows. It turns out that 16 locations (the highest number of location of the pNML architecture, a 4 by 4 array) are not enough to split the hardware in two pipeline stages. Therefore, the whole operation is performed in just one algorithmic step. Besides the possible pipeline stages, a single plane of the architecture does not contain registers, resulting in a completely combinational circuit that calculates everything in one clock cycle.

The way the two architectures are tested is quite different. First, the CMOS architecture is synthesized in Synopsys Design Compiler, on a Nangate 45 nm cell library [12]. In this step, the parameters are set (word width and number of locations). Design Compiler outputs a circuit that can be simulated on Mentor Graphics Modelsim. By linking the same library in Modelsim, the simulation will consider the delay of each node of the network. The testbench contains instructions whose purpose is measuring and storing the delay. The area can be easily assessed from Design Compiler's reports.

On the other hand, pNML architecture's performances are investigated differently. First of all, there is no synthesis, as it is easy to guess because this is still a pretty new technology. Nevertheless, MagCAD delivers a VHDL description that already includes the delays. These delays are found by means of an analytical model, grounded on the physical description of magnetics nanodomains' motion. Therefore, the simulation of the circuit (still done in Modelsim, or its higher-end equivalent, Questa) will always show the delay of every signal. Anyway, what is to be found for the pNML architecture is the number of clock cycles needed to complete the algorithm. More specifically, since the architecture contains notches that are used as registers, one has to know both how many notch cycles are needed, and how many clock cycles make up a notch cycle. Then, since the clock period is know

in the first place, this number will become a time. This whole process could be done without a simulation: MagCAD provides the minimum clock period without needing simulations, and the length of the notch cycle in terms of clock cycles can be found from an inspection of the circuit (that will find the critical path); last, the number of notch cycles the algorithm needs depends on the algorithm itself and how it is mapped on hardware, it can be found from a circuit inspection as well as the notch cycle. This way the total time needed to perform the algorithm can be found. Despite all these observations, plenty of simulations have been carried out. Two reason are behind them. One is that the correct functionality of the algorithm had to be checked. The same data were put through the two circuits, and the two outcomes were cross-checked. No unmatched cases occurred; furthermore, they were also checked against the results issued by MATLAB. The second reason is that the critical path can be searched within the layout of the circuit, but this way mistakes are very frequent. The paths to analyze are many, and after having found the (supposed) critical one, the circuit must be simulated to check whether it works at that specific frequency, and stops working as soon as the frequency is increased, even slightly. If all of these things happen, the critical path must be the one found. Once the number of cycles is know, any variation either to the area or to the technological timing parameters (see chapter 7) can be made: the clock period will be recalculated and so will the total time. In pNML, a 2 by 2 cell has been designed, as the basic component of an array of arbitrary size, and then four of those cells have been gathered together in a 4 by 4 array. Thus, both the cell alone and the four cell array have been tested. Let us now have a look to the result of this comparison, in table 5.1 and table 5.2. As expected, the comparison shows that the CMOS architecture is much faster than the pNML one. As CMOS is a much older and tested technology, it should not surprise that the number of logical operations per second that it is able to carry out is definitely higher than the one of an emerging technology such as pNML is. Furthermore, the interest in pNML does not reside in its speed, but rather in its being non-volatile and low-power. In other (rough) words, a fast technology is compared with a slow one. This should account for most of the difference, that is 6 or 7 orders of magnitude.

The matter can be elaborated a little. For example, this pNML circuit contains several notches and takes some notch cycles to complete the algorithm, whereas this

	CMOS	pNML
Minimum Clock Period	497 ps [1]	1,676 μs
Notch Period Length	-	29 Clock Cycles
# Notch Periods	-	19
Total	497 ps	923,476 μs
Throughput	2,012 GOps	1,083 KOps
$\left(\frac{Throughput_{pNML}}{Throughput_{CMOS}}\right)$	$5{,}383 \cdot 10^{-7}$	
Area	310 μm^2	169 um^2
$\left(\frac{Area_{pNML}}{Area_{CMOS}}\right)$	0,544	

Ops= Operations per second; KOps= Kilo Operations per secon; MOps= Mega Operations per second; GOps= Giga Operations per second
[1] The delay is the average among 136 simulations

Table 5.1: Delay and area comparison between the pNML architecture and the one of [11], with a 2 x 2 SAT size

	CMOS	pNML
Minimum Clock Period	1,747 ns[1]	1,750 μs
Notch Period Length	-	29 Clock Cycles
# Notch Periods	-	67
Total	1,747 ns	3,4 ms
Throughput	572 MOps	294 Ops
$\left(\frac{Throughput_{pNML}}{Throughput_{CMOS}}\right)$	$5{,}140 \cdot 10^{-7}$	
Area	2485 μm^2	1249 μm^2
$\left(\frac{Area_{pNML}}{Area_{CMOS}}\right)$	0,502	

Ops= Operations per second; MOps= Mega Operations per second
[1] The delay is the average among 28 simulations

Table 5.2: Delay and area comparison between the pNML architecture and the one of [11], with a 4 x 4 SAT size

CMOS one is not pipelined at all and takes a single clock cycle. By "pipeline" here the insertion of notches is meant. It should be reminded that from a theoretical point of view the notches in pNML are never necessary. With a meticulous study of the timing in the circuit, everything can work. This circuit is too complex to do that. Basically, here notches were used to deal with the complexity with a reasonable effort, and they were inserted where it is particularly convenient to break the logical

path. From the timing point of view however, they do not improve in any way the delay. The critical path is greater than most of the other paths; in terms of the classical pipelining theory, here the pipeline is not balanced, and this slows down the circuit's performances. There is also a logical loop (data comes out from the memory, enters the adder and the result gets back to the memory), which makes impossible for the circuit to get improved by means of a pipelining technique.

Another reason that might account, at least to some extent, for the performance gap, is the great diversity of the architectures. The pNML one has an adder-accumulator, that takes time the be loaded because has just an input port, whereas in the CMOS version all the needed adders are present in hardware. The number of sums carried out in the same time by the CMOS architecture is certainly larger. In conclusion, even if CMOS technology is much faster than pNML, the comparison cannot be considered fair. A last point to consider is the library: it plays a very important, and possibly determinant, role in the performances of the CMOS architecture. It should be noted however that this is not the most updated CMOS library at all, and with more recent ones the gap between the two architectures will be likely to increase.

The other characteristic to be discussed is the area. As both tables (5.1 and 5.2) show, pNML implementation takes up about half of the area that the CMOS implementation takes. It would seem that in this aspect pNML is much better than CMOS, but there are at least two points that are in favor of CMOS. First off, CMOS allows to manufacture much smaller circuits, this is certainly not the tiniest technological node. Second, the architectures are very different: the pNML one is "minimal", meaning that is carries out the algorithm with the least possible hardware, because everything is hand-designed. The CMOS one follows a completely different approach: all the needed hardware is deployed, there is no reuse and the computations go ahead in a mono-directional fashion (partial results are never brought back to the same processing element, they always enter another one). This is actually one of the most area-voracious styles of implementation. An implementation with a different technology node can be tried; not in full detail, but just analytically approximating the area. Let us consider the 22 nm and the 14 nm nodes. The approximation could work like this: in this 45 nm technology, the ratio between the technology node typical size and the total area was:

$\frac{310\mu m^2}{0{,}045\mu m\ 0{,}045\mu m} = 1{,}531 \cdot 10^5$ (2 x 2) $\quad\quad$ $\frac{2485\mu m^2}{0{,}045\mu m\ 0{,}045\mu m} = 1{,}227 \cdot 10^6$ (4 x 4)

Now, the roughest estimation of how big the circuit would be if synthesized in those two technological nodes consists in multiplying those numbers by the square of the technological node typical size in use. However, pNML can be shrunk too. The architecture was designed at first with a typical length equal to 330 nm, and one can safely assume to be able to lower that value at least to 200 nm. As extreme physical limit, let us assume 45 nm. With these new values, the data are those in table 5.3. More recent CMOS nodes lead to much smaller circuits, yet larger than a

	2 x 2	4 x 4
CMOS 22 nm	74,1	594
CMOS 14 nm	30,0	249
pNML 200 nm	61,9	459
pNML 45 nm	3,13	23,2

Table 5.3: Area estimates (in μm^2) for implementations with smaller technology nodes

CMOS \ pNML	330 nm	200 nm	45 nm
45 nm	0,544	0,200	0,0101
22 nm	2,28	0,836	0,0423
14 nm	5,62	2,06	0,105

Table 5.4: Area ratios ($\frac{Area_{pNML}}{Area_{CMOS}}$) for a 2 x 2 SAT size for some pNML and CMOS nodes

CMOS \ pNML	330 nm	200 nm	45 nm
45 nm	0,503	0,185	0,00934
22 nm	2,10	0,772	0,0391
14 nm	5,19	1,907	0,0965

Table 5.5: Area ratios ($\frac{Area_{pNML}}{Area_{CMOS}}$) for a 4 x 4 SAT size for some pNML and CMOS nodes

very scaled pNML implementation, as table 5.5 shows. However, pNML architecture is intrinsically smaller. Next section elaborates this point, considering the problem from a different point of view.

5.2 CMOS conversion

Because of what has been said in the previous section, another kind of comparison has been tried. This time, the very same architecture has been described in VHDL, synthesized in CMOS with the same library as before, and tested. In this case, pNML notches become registers in CMOS; hence, the algorithm will not be completed in just one clock cycle. The amount of notch cycles will be equal to the amount of clock cycles in CMOS.

There is a possible source of problem: the synthesis is performed by the CAD tool, and depending on the VHDL coding style, the circuit might be very different from the one in pNML, making meaningless the whole comparison process. This matter is tackled by choosing to write two different VHDL descriptions. The first one just describes the function of the components, leaving to the compiler the task of choosing the most suited hardware implementation. For instance, the memory is written as a for loop, as it would be done in a software language. In the other description, the elements designed throughout all this thesis work are described in the most faithful way, with a large use of the "generation" structure of VHDL. Half adders, full adders, memories and multiplexers are defined and allocated; even the memory cell is made in way similar to the two-multiplexer way seen in section 3.5, figure 3.11. A different way to express the concept could be that the former is a behavioral VHDL description, while the latter is a structural VHDL description. Both versions will be useful and will shed light on different aspects of the comparison. Table 5.6 contains the minimum clock periods for each size and description style. In that table, the lowest clock period is considered. Table 5.6 shows that if the architecture are more alike, the differences are lower. However, even though for these CMOS circuits the delays have grown by about one order of magnitude, the gap is still huge. This proves that the architecture too accounts for the difference, but very little. Someone might legitimately wonder why these two CMOS architectures are considered, since there was one in the previous section that was faster. This is correct: in the end, each design should be free to exploit as much as possible the strong points of the technology in use. What must be compared should be the final outcome of a circuit made trying to achieve the most from the available resources. In other words, the comparison should have the best pNML circuit on one side and

	Behavioral description	Structural description
	Clock period ps	
2 x 2	420	480
4 x 4	470	570
	Total time ns	
2 x 2	7,98	9,12
4 x 4	31,49	38,19
	Throughput	
2 x 2	125 MOps	110 MOps
4 x 4	31,8 MOps	261 MOps
	Total time ratio $\left(\frac{pNML}{CMOS}\right)$	
2 x 2	1,157 $\cdot 10^5$	1,013 $\cdot 10^5$
4 x 4	1,080 $\cdot 10^5$	8,903 $\cdot 10^4$

Ops= Operations per second; MOps= Mega Operations per second

Table 5.6: Delay and throughput of the two CMOS versions with the same architecture used for the pNML implementation

the best CMOS circuit on the other. That having being said, there are reasons that justify this choice too. The most important one is that this is only one algorithm, and it might be much more suitable for one of the two technologies: therefore, new architectures, whose performances are better, might exist. Making the comparison independent from the architecture solves the problem. Furthermore, whatever anyone could think about how two technologies must be compared, analyzing them when the architectures are the same allows to investigate what are the causes of the difference in performance.

So far, delay has been the only parameter considered. The two technologies have been employed to build a circuit able to run the algorithm as fast as possible, regardless of power or area. It may be interesting to check out how goes the comparison when, for example, the two sides have a constrained power budget, or a constrained maximum area. In the early stages of this work, this was one of the objectives. Unfortunately, to date no satisfactory power estimation tool for pNML exists. Therefore, the comparison cannot be made, but the CMOS architectures

power reports have been generated, and the data is listed in Appendix D, ready for the time when such a tool will be developed. In chapter 6, the SAT architecture is designed in a different fashion, using just two layers.

Now let us analyze the area. Table 5.7 shows the area of these CMOS versions of the circuit. pNML proves again to be the smallest one, and in this case, the

	Behavioral description	Structural description
	Area $[\mu m^2]$	
2 x 2	427	471
4 x 4	1710	1799
	$\frac{Area_{pNML}}{Area_{CMOS}}$	
2 x 2	0,375	0,340
4 x 4	0,716	0,681

The areas are the average of the different values, one for each clock period constraint. The circuit with time constraint equal to 500 *ps* was much bigger, and has not been taken into account.

Table 5.7: Area of the two CMOS versions with the same architecture used for the pNML implementation

two technologies have been used to implement the same circuits, even though in two different synthesis styles. The area does not change much between the structural description and the behavioral description. Let us focus on the CMOS version. It was said that the architecture difference could account for the fact that CMOS circuit is larger than pNML. The pNML one is minimal, whereas the CMOS one deploys as many processing elements as needed, with no reuse of any of them. This is actually true, but it turns out that it does not explain the area difference, since using the same architectures makes the CMOS circuit bigger. The memory element should be responsible for that. The first architecture does not include a memory, just combinational logic that carries out its part of the algorithm and then passes the results to the next plane. A pipeline register could be placed between the two planes, but no other memory element is there. Given that in CMOS the memory takes up much more room than the combinational logic, it is reasonable to suppose that the insertion of an element caused the circuit to increase its size.

To sum up all this discussion, in some cases the CMOS circuit was smaller than the pNML version, but the most scaled pNML version considered is likely to be smaller than any CMOS technological node. Notice also that the 200 *nm* version of pNML is smaller than the 14 *nm* CMOS version: it shows that if a pNML and a CMOS circuit are manufactured with the same process (and with the same resolution), the former will probably be much smaller than the other.

In order for these area and delay figures to be checked later on, figure 5.1 shows a screenshot of the parameters set in MagCAD to get the delays listed in this chapter. This report is found in the file *definitions_pnml.vhd*

```
--------------------------------------------------------------------------------
-- Definitions and parameters automatically generate by MagCAD
--
-- Please use MagCAD to change technological parameters
--
--------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

package definitions_pnml is

  --fields constants
  constant H_pulse                  : real         := 560.0;                  --field pulse amplitude [Oe]
  constant H_clock                  : real         := 560.0;                  --clocking field amplitude[Oe]
  constant H_int                    : real         := 190.0;                  --intrinsic pinning field[Oe]

  --time constants
  constant T_prop                   : real         := 1.1e-07;                --Original value 1.0118e-07       effective clock pulse [s]
  constant T_nuc                    : real         := 5.6e-07;                --Original value 5.5752e-07       nucleation time [s]

  --Probability extremes
  constant P_nuc_critical           : real         := 0.95;                   --critical nucleation probability
  constant P_prev_critical          : real         := 0.05;                   --critical nucleation prevent probability

  --base components constants
  constant dw_length                : real         := 3.3e-07;                --length of the domainwall [meters]
  constant dw_width                 : real         := 2.2e-07;                --width of the domainwall [meters]
  constant v_00                     : real         := 1.14e+06;               --numerical prefactor in depinning regime [m/s]
  constant E_pin                    : real         := 12.1;                   --pinning energy barrier (incluedes the K_bT constant)
  constant v_0                      : real         := 31.9;                   --numerical prefactor in flow regime [m/s]
  constant u_w                      : real         := 0.042;                  --domain wall mobility [m/sOe]
  constant C_inv                    : real         := 153.0;                  --Coupling field strength [Oe]
  constant C_mv                     : real         := 48.0;                   --coupling field
  constant f_0                      : real         := 2.0e+09;                --Attempt frequency [Hz]
  constant M_co                     : real         := 1.4e+06;                --Saturation magnetization Co
  constant t_Co                     : real         := 3.2e-09;                --Co thickness [m]
  constant t_stack                  : real         := 6.2e-09;                --stack thickness [m]
  constant K_eff                    : real         := 2.0e+05;                --effective anisotropy [J/m^3]
  constant u_0                      : real         := 1.256e-6;               --permeability constant
  constant g_FIB                    : real         := 0.1416;                 --fib irradiation factor
  constant V_ANC                    : real         := 1.69e-23;               --volume of the ANC
  constant K_B                      : real         := 1.3807e-23;             --Boltzmann constant
  constant T                        : real         := 293.0;                  --temperature in Kelvin

  --notch component constants
  constant A                        : real         := 1.3e-11;                        --exchange stiffness
  constant apex                     : real         :=(51.5* MATH_PI)/180.0;           --apex angle
  constant h                        : real         := 5.4e-08;                        --notch width
  constant V_a                      : real         := 1.26e-23;                       --Activation volume
  constant P_dep                    : real         := 0.98;                           --Used to find the T_sync (1.0 - EXP(-T_sync/Tau_eff_notch) )

  --domainwall via constants
  constant C_via                    : real         := 75.0;                           --coupling field for the via

  --Formula for the parameters
  constant M_s                      : real         := M_Co * (t_Co/T_stack);                              --saturation magnetization
  constant K_anc                    : real         := g_FIB * K_eff;                                      --ANC anisotropy
  constant H_0                      : real         := ((2.9 * K_anc)/(u_0*M_s))*0.01256;                  --coercive field at zero temperature
  constant E_0                      : real         := (K_anc * V_ANC)/(K_B*T);                            --energy barrier at 0 field (incluedes the K_bT constant)
  constant T_eff                    : real         := T_prop + T_nuc;
  constant T_rise                   : real         := T_eff / 4.0;
  constant T_clock                  : real         := T_eff + T_rise;                                     --Effective pulse time

  --NOTCH parameters formulas
  constant d_W                      : real         := MATH_PI*SQRT(A/K_eff);                                                                --characteristic DW width
  constant o_W                      : real         := 4.0 * SQRT(A*K_eff);                                                                  --DW energy density
  constant H_depin                  : real         := H_int + (o_W * SIN(apex))/(2.0*M_s*(h + 0.5*d_W*SIN(apex)))*10000.0;                  --Depinning Field
  constant H_sync                   : real         := H_depin - H_pulse + 1.0;                                                              --In plane field
  constant H_eff_notch              : real         := H_pulse + H_sync;                                                                     --Effective field to depin a notch
  constant T_depin                  : real         := (1.0/f_0) * EXP((M_s*V_a*((H_depin - H_eff_notch)/10000.0))/(K_B*T));                 --Depinning time
  constant E_barrier_notch          : real         := (M_s*V_a*(H_depin - H_eff_notch))/10000.0;                                            --Energy barrier
  constant Tau_eff_notch            : real         := (1.0/f_0) * EXP(E_barrier_notch/(K_b*T));                                             --Effective time to depin a notch
  constant t_sync                   : real         := -Tau_eff_notch*LOG(1.0 - P_dep);                                                      --In plane pulse time


  --generic model constants
  --Set here the number of parameters that you want to analyse: e.g. the critical path.
  constant NParameters              : integer      := 3;                      --number of model parameters
  constant par_TIME                 : integer      := 0;                      --position of propagation time in PARAM_DATA. Here each component add its delay
  constant par_TIME_NUC             : integer      := 1;
  constant par_CR_TIME              : integer      := 2;                      --position of critical propagation time in PARAM_DATA. Here is memorized the current critical path.
```

Figure 5.1: MagCAD report of the technological parameters used in this chapter

Chapter 6

Two-Layer flattening

The objective pursed in chapter 3 consisted in exploring good layout solutions that take advantage of the layers above the first one or two. The assumption was that as many layers as needed were available: in chapter 4, section 4.4 14 layers were eventually used. However, to this day (February 2018), no experimental proof of circuits with more than two layers exists. On one hand, there is no evidence of the impossibility of manufacturing this number of layers; on the other hand, in order not to rely too much in projections about future technological achievements, it is extremely useful analyzing how the multi layer circuits built so far would look like if they were leveled to just two layers. In next sections, some examples of two-layer design will be given. The circuits have been adapted to two layers, then area and timing are compared. In some cases, a few points concerning topological changes are underlined too. When doing either area or length comparisons the units are arbitrary, because they depend on the spatial resolution in use, that depends on the technology. MagCAD draws a grid over the circuit, that represent this spatial resolution. The side of each square is about as long as the width of the domain wall, or as the diameter of the pads. Therefore, the most convenient choice consists in using this square as a reference unit: its side will be the unit length, and its area the unit area. The term "square" will be used in both cases, and the contest will make clear whether it is referred to a length or to an area.

6.1 Multiplexer

The circuit shown in section 3.3 is here redrawn using just two layers. The final result can be seen in figure 6.1. Clearly, the aspect ratio of the circuit has changed.

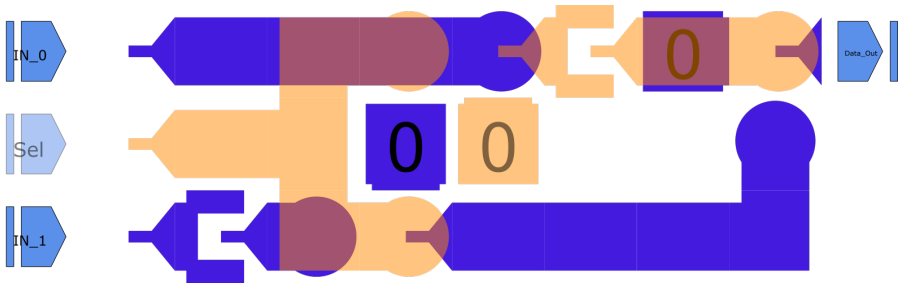


Figure 6.1: Two-layer design of the multiplexer in figure 3.9

Figure 6.2 represents both versions of the circuit; this way, it is easier to spot size differences by just looking at it. Then, table 6.1 contains all the details about

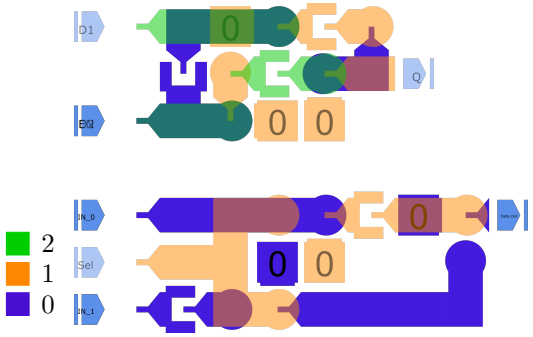


Figure 6.2: Multiplexer in figure 3.9 compared with its two-layer version

timing and area of the two circuits. As it is easy to see from image 6.2, the two-layer version is bigger by one third. Nevertheless, with respect to the timing, the two-layer option outperforms the other one. Unexpected though it might seem, this is normal. In pNML, an half clock cycles delay is unavoidable to invert a signal. A signal moving up or down to another layer undergoes a full clock cycle delay: it means that this operation just adds delay without performing an either useful or needed computation. Thus, something that slows down a signal by a full clock cycle, as a VIA does, is always detrimental for the timing. It is just a waste. The only reason they are used it is because it is a trade-off that saves area. Of course,

	Area		
	3 layer version	2 layer version	Ratio
Height	3	3	1
Width	6	8	1,33
Area	18	24	1,33
	Delay		
	3 layer version	2 layer version	Ratio
D1 → Q	8	6	0,75
D2 → Q	8	4	0,75
En → Q[1]	9 (8)	5 (4)	0,56(0,5)
Clock Period μs	1,448	1,480	1,02
Longest domain wall's length	3	5	1,67

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Delays are expressed in half clock cycles. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.

[1] The path branches in two different ones; hence, there are two different delay counts.

Table 6.1: Area and delays of the multiplexer in figure 3.9 and of its two-layer version

a two-layer implementation of a circuit still has some inter-layer signal passing, but when many layers are used, it is reasonable supposing that they will be on average more frequent. Last, the clock period does not change much, even if the relative difference of the longest domain wall's length is quite high. This is due to the specific technological parameters chosen, that make nucleation delays (independent from domain wall length) the main contribute to the total delay (for further information, refer to chapter 7). The delays and area are listed in table 6.1.

6.2 Full Adder

A second circuit analyzed is the one seen in subsection 4.1.2. Actually, that one is a 4-bit ripple carry adder, whereas the circuit here redesigned in two layers is just one of the three left hand modules, namely, a full adder. In figure 4.6, they are shown all wired together. The layout can be seen in figure 6.3. The circuit is quite compact, and this compactness is partially due to the use of the "oblique pad". It can be seen alone in figure 6.4. A visual comparison between this version and the one of chapter 4 can be made by looking at the tiny figure 6.5. However, the oblique pad

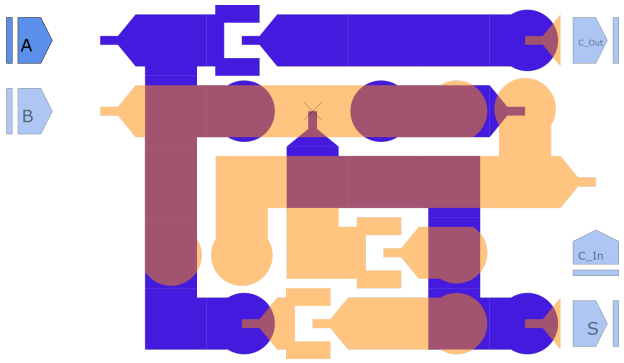


Figure 6.3: Two-layer design of a full adder (with oblique pad)

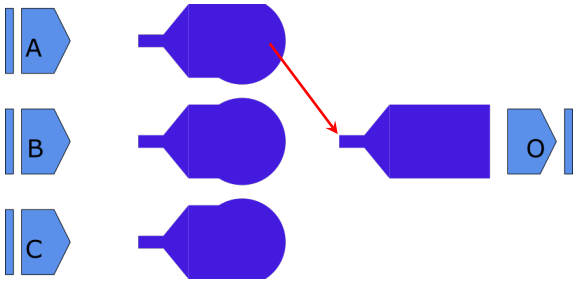


Figure 6.4: Oblique pad, a new technological feature

has never been used so far. Hence, to make the comparison fair, here is presented a version which does not include that specific element. It can be seen in figure 6.6; figure 6.7 draws both versions (the multilayer one and the two layer one); table 6.2 thoroughly compares timing and areas for the three versions. In this case the area increase is similar to the one obtained in section 6.1. Delays are either unchanged or reduced, confirming the assumption that a two-layer version should be faster. Unlike the multiplexer case, the longest domain wall this time is longer in the multi layer version, with the consequent (small) reduction in the total clock period. Anyway, the total clock period is not supposed to change too much: the idea is that any time

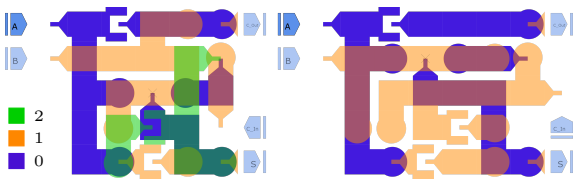


Figure 6.5: Full adder compared with its two-layer version (with oblique pad)

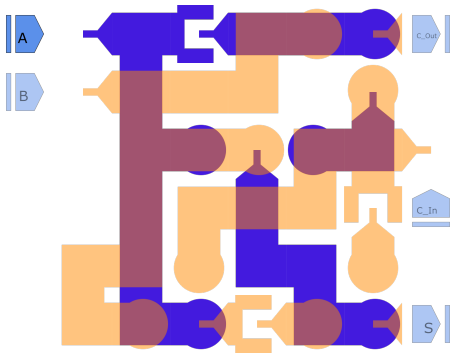


Figure 6.6: Two-layer design of a full adder (without oblique pad)

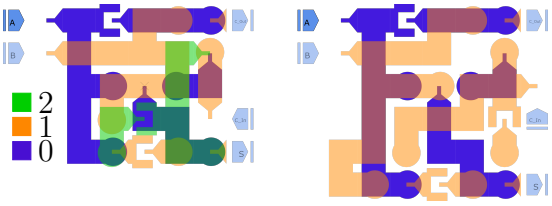


Figure 6.7: Full adder compared with its two-layer version (without oblique pad)

a domain wall gets too long, it is split with a couple of inverters. Hence, the total clock period depends more on the technology than on the layout; a longer domain wall will have a higher delay in terms of clock cycles. This is the most meaningful timing parameter when comparing different version of the same circuit. Therefore, the variations on the clock period will no longer be reported.

So far, simple circuits, with no internal modularity, have been analyzed. In all cases, area came out larger and delay did lower. Basing on this analysis,it can be stated that area increases by about 20-30%; in regard to the timing, it often happened that the same input-output path were shorter in the two-layer version. Nonetheless, since what actually characterizes the circuit is the longest path, it is safer saying that the delay is about the same, sometimes slightly shorter.

	3 layer	Area 2 layer w OP	2 layer w/o OP	Ratio w OP	Ratio w/o OP
Height	5	5	6	1	1,2
Width	6	7	6	1,17	1
Area	30	35	36	1,17	1,2

	3 layer	Delay 2 layer w OP	2 layer w/o OP	Ratio w OP	Ratio w/o OP
$A \rightarrow C_{out}$	4	4	4	1	1
$B \rightarrow C_{out}$	2	2	2	1	1
$C_{in} \rightarrow C_{out}$	2	2	2	1	1
$A \rightarrow S$ [1]	5 (4)	5 (4)	5 (4)	1	1
$B \rightarrow S$ [1]	5 (4)	5 (4)	5 (4)	1	1
$C_{in} \rightarrow S$ [2]	8 (6)(5)	6 (4)(3)	6 (4)(3)	0,75	0,75
Clock Period μs	1,528	1,495	1,512	0,98	0,99
Longest domain wall's length	8	6	7	0,75	0,875

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Delays are expressed in half clock cycles. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$. OP = oblique pad.

[1] The path branches in two different ones; hence, there are two different delay counts.

[2] The path branches in three different ones; hence, there are three different delay counts.

Table 6.2: Area and delays of the three versions of the full adder

6.3 Flat delay decoder

In this section, the same layer-reducing operation is carried out over a decoder, seen in section 3.1, figure 3.4. The decoders are chosen as examples of circuits with many sections repeated, possibly with minor changes, such as the inversion of some inputs according to the output number. They are in fact arrays of basic elements. In the last sections it has been shown that for simple circuits the area increment was about 20-30%. Thus let us analyze figure 6.8, representing the outcome of this flattening operation.

This circuit includes good examples of what is done to fit in just two layers a circuit previously designed for at least three. Keep in mind that the idea is not "redesign the multi layer circuit in two layers" but rather "convert the multi

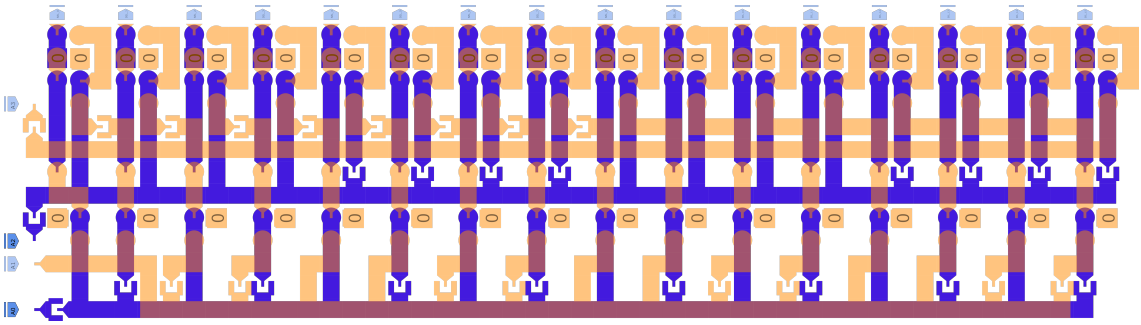


Figure 6.8: Two-layer design of the decoder in figure 3.4

layer circuit in a two-layer one". It means that, as far as the reduced number of layers allows, the two circuits are identical. Nonetheless, any interesting design solution found out while "converting" the multi layer circuit in a two-layer one will be applied, even if completely different from its multi layer counterpart. Figure 6.9 contains both decoder's versions, with some marks. Red dots, circles and squares are ideally located on the layer 0, green ones are located on layer 1. The blue lines do not belong to any layer in particular. The circuits are turned by 90 degrees, so that by "height" it is meant the dimension parallel to the short side of the page, and by "width" it is meant the dimension parallel to the long side of the page. In the two-layer circuit image, all the marks are relative to the second output because the first one is different from all the other (the domain wall must not run above the inverter in the solid red circle on the bottom layer, so it has been diverted). As it can be seen, the circuit certainly has increased in size, both in height and in width. Looking more in detail, it can also be seen that when only two layers are available, some tricks to cross domain wall are needed. In the image it can be seen that in order for the signal in the red dot (layer 0) to reach the green dot (layer 1) the "bypass" made with the domain wall in green rectangle (layer 1) and the one in the red rectangle (layer 0) were needed. This adds to the width of the circuit: the circuit is larger by one or two squares. In the multi layer version of the circuit there was no need to do such a thing: those signals are just laid on a further layer, which of course does not cross with any other layer. The only thing to care about is avoid placing inverters over domain walls. In conclusion, it is fair to say that by giving up the third layer, the circuit has increased in width by one or two squares. The blue double arrow defines the width of the logic circuitry devoted

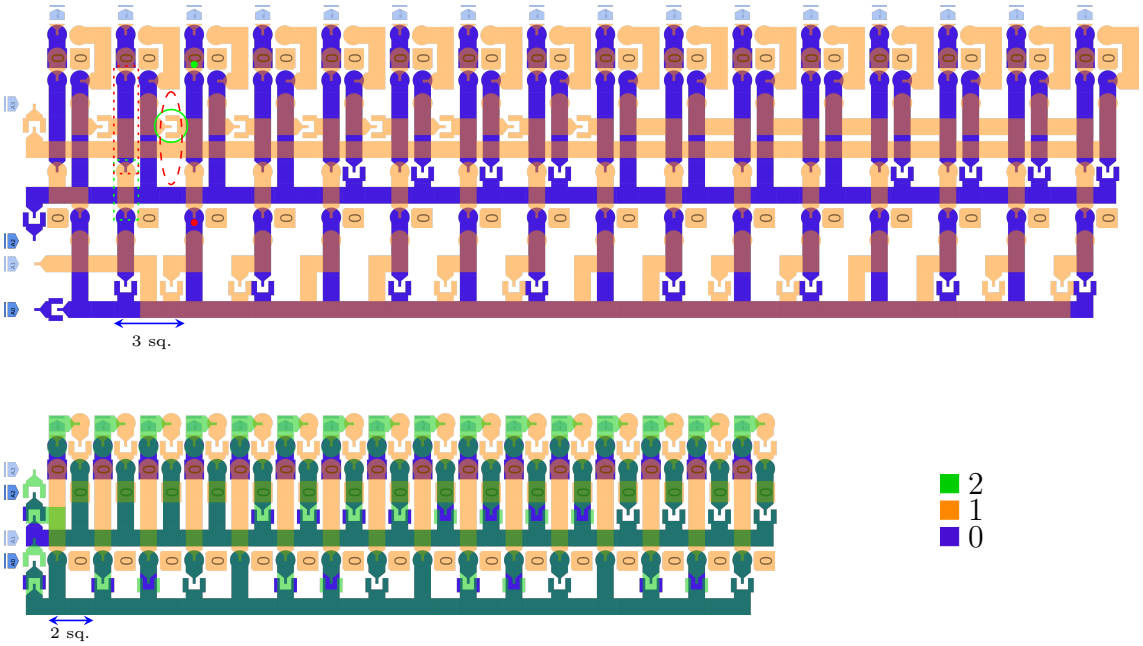


Figure 6.9: Decoder in figure 3.4 compared with its two-layer version

to a single output. This section is one square wider in the two-layer circuit: the outputs are 16, and this account for most of the height increase. This is due to the inverter inside the green circle: it must not couple with domain wall below, so that the "spacing" marked with the red, dashed ellipse, has been inserted. Trying to formulate a general criterion to predict the area increase when converting a three layer circuit into a two-layer version, it can be claimed that, both in this case and in the one in section 6.1, a simple circuit (a full adder, the circuitry that here produces a single output) becomes more or less one square larger in footprint. Since the decoder could be considered a 2 by 16 matrix of such simple circuits, a 16 square increase in height and a 2 square increase in width are expected, which is close to what actually occurred. Table 6.3 shows precisely how timing and area metrics changed. In this case, something unseen before happened. The delay is sometimes greater in the two-layer version, because of the bypass (the trick used to bring input A_0 to the output, see figure 6.9). Anyway, the characterizing delay is always the greatest one, so that, once again, the delay stays unchanged between the two version (the worst case is the same for both of them). The area is larger in the two-layer version, which comes as no surprise; nevertheless, the ratio has considerably increased, and

	Area		
	3 layer	2 layer	Ratio
Height	33	49	1,48
Width	9	13	1,44
Area	297	637	2,14
	Delay		
	3 layer	2 layer	Ratio
A_0	6 (5)	10 (9)	1,67 (1,8)
A_1	6 (5)	8 (7)	1,14 (1,17)
A_2	10 (9)	6 (5)	0,6 (0,56)
A_3	10 (9)	6 (5)	0,6 (0,56)

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Delays are expressed in half clock cycles. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.
Each input reaches every output, either inverted or non inverted. Hence, two delay counts always differing by half a cycle.

Table 6.3: Area and delays of the decoder in figure 3.4 in section 3.1 and of its two-layer version

the surface has doubled. This is due to the mechanism discussed above: on average, each gate increases its size by a few squares, and circuits made of an regular array of gates increase their size proportionally to the number of gates, even if not linearly (more complex circuits show a worse increase ratio).

6.4 Scaled delay decoder

A different kind of decoder has been analyzed too. In this version, seen in figure 3.5, section 3.1, the delays are not equal for every output. The farther from the input the output is, the higher the delay gets. Compared to the previous version of the decoder, this has a greater delay but a slightly smaller area, and it might be interesting checking how things change when it is reduced to only two layers. The two-layer circuit is shown in figure 6.10, whereas figure 6.11 shows both multi layer and two-layer version, in order to visually compare them. Notice that more or less the same techniques used in the other decoder version are used here: the same structure underlined in figure 6.9 can be seen. These two-layer version of the two decoders are very similar indeed, much more than the multi layer versions are. Have

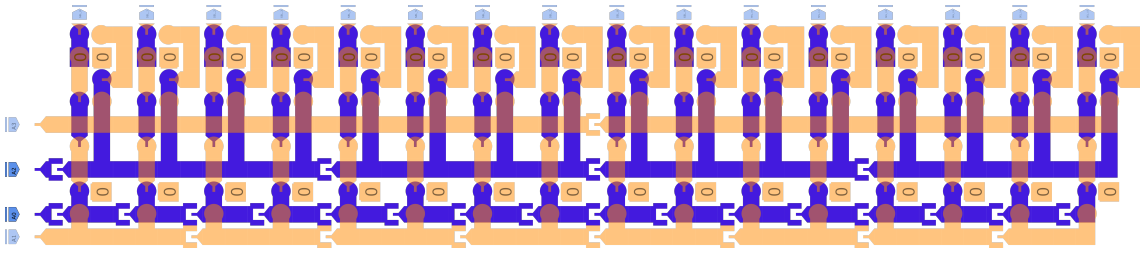


Figure 6.10: Two-layer design of the decoder in figure 3.5 (scaled delay)

a look at figure 6.11. Four rectangles are drawn. As usual, red colored shapes must

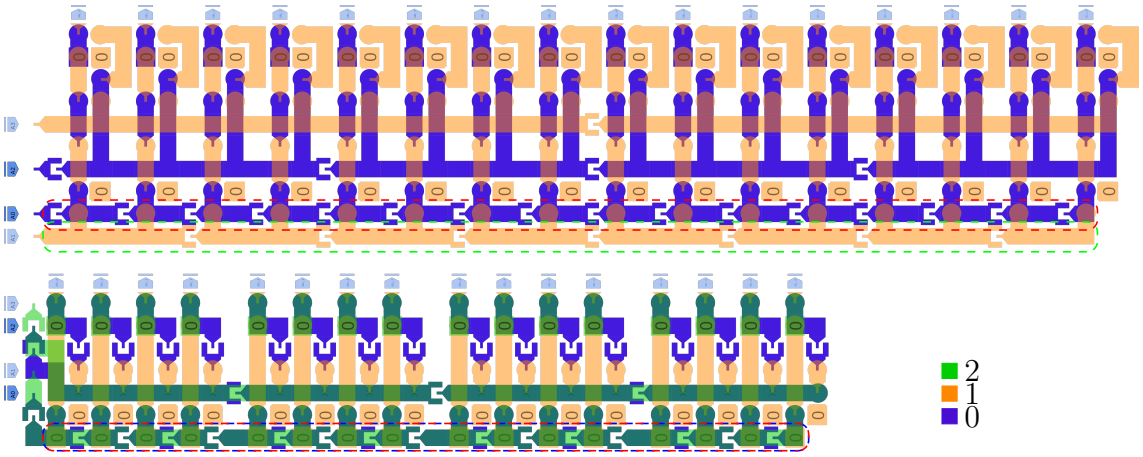


Figure 6.11: Decoder of figure 3.5 compared with its two-layer version

be thought as belonging to layer 0, green color ones to layer 1, and here there is also a blue rectangle, for something placed in layer 2. The whole point of the rectangles is to show that two domain walls, bearing the values of signals A_0 and A_1, are perfectly overlapped in the three layer version, and shifted by one square in the two-layer one. The reason is know, and this is just an example of how leveling in two layers causes the circuit to widen its footprint. However, the inverters needed to rearrange the inputs according to each output are disposed along the height rather than the width, as it was the case in the other decoder. This causes the width increase to be a little more restrained, while the height incrementation is just the same. Table 6.4 lists all the parameters in full detail. By analyzing the exact values for area and timing, it can be seen that the area has increased a little less than before. The delay is larger for inputs 0 and 1, and lower for inputs 2 and 3. This is due to the bypass structures shown in figure 6.9: the delay is usually lower for two-layer implementations, but

	Area		
	3 layer	2 layer	Ratio
Height	36	50	1,38
Width	7	10	1,43
Area	252	500	1,98

	Delay			
	3 layer	2 layer	Ratio	
			Output 0	Output 15
A_0	$5+i$	$9+i$	1,8	1,2
A_1	$5+\lfloor\frac{i}{2}\rfloor$	$7+\lfloor\frac{i}{2}\rfloor$	1,4	1,67
A_2	$9+\lfloor\frac{i}{4}\rfloor$	$5+\lfloor\frac{i}{4}\rfloor$	0,56	0,67
A_3	$9+\lfloor\frac{i}{8}\rfloor$	$3+\lfloor\frac{i}{8}\rfloor$	0,33	0,4

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Delays are expressed in half clock cycles. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.
The delay of each input depends on the particular output, i represents the output number, $0 \leq i \leq 15$. Since the delay varies, both the ratio of input 0 and the one of input 15 has been calculated.

Table 6.4: Area and delays of the decoder in figure 3.5 and of its two-layer version

when these structures are needed, they increase the delay value, which may get even larger (as in here), than the multi layer implementation. Notice also that the two delay values tend to get closer to each other as the output number increases, no matter which implementation is the fastest on output 0. Anyway, because of the usual worst case rule, the two-layer implementation is slower.

There is a last interesting point to analyze briefly about both versions of the decoder. In chapter 3, it has been shown by means of two timing diagrams (in figure 3.6 and 3.7) that the scaled version of the decoder is more affected by glitches than the flat one (which was almost glitch-free). Figure 6.12 shows the time evolution when all the outputs are tested, and represents fairly well the behavior of both the versions: hence, the two-layer leveling has made them similar from this point of view. By comparison to the two timing diagrams of chapter 3, it can be observed that the glitching behavior here is stronger than both of those previous cases.

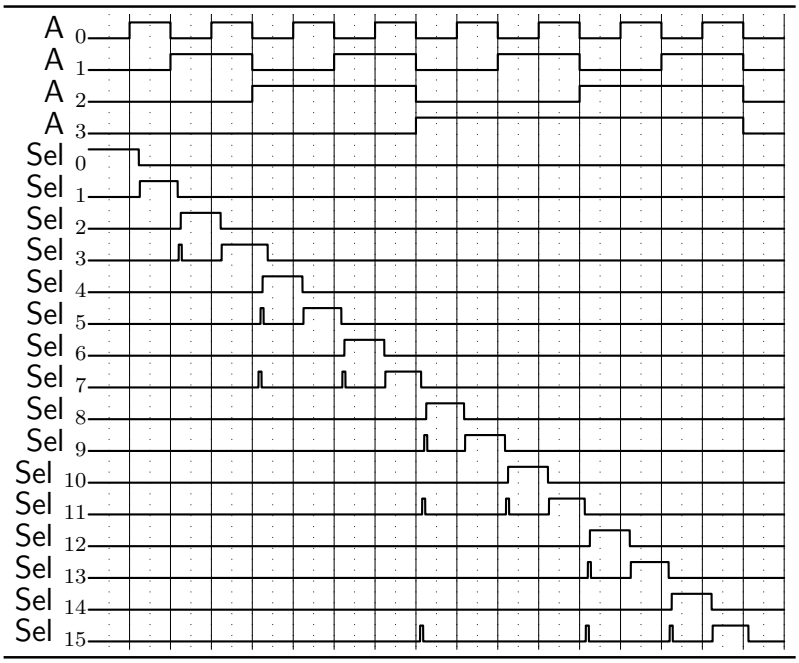


Figure 6.12: Test of the whole set of input combination for the circuit in figure 6.8 and in figure 6.10

So far two kinds of circuits have been analyzed, and they can be considered simple circuits (the full adder and the multiplexer) or array of simple circuits (these two decoders). A general assumption could be that simple circuits undergo an area increase by 20-30%, while the delay changes very little. More complex circuits, such as these two decoders, undergo a greater rise in area, about 100%; due to bypass structures, delays might increase by almost the same factor (70-80%).

6.5 Programmable logic array

A PLA is used to implement an arbitrary combinational logic function; usually in the other technologies the logical function is either one-time programmable or reprogrammable, but not at execution time, while in pNML the PLA can be re-programmed as many times as needed at execution time. Given its function, the PLA contains a great deal of combinational logic, highly structured. The two-layer reduction is pictured in figure 6.13. The first thing one notes regarding the lay-out structure is that it is full of "voids". These voids are chiefly caused by the impossibility to access a nucleation center from many directions (in multi layer it

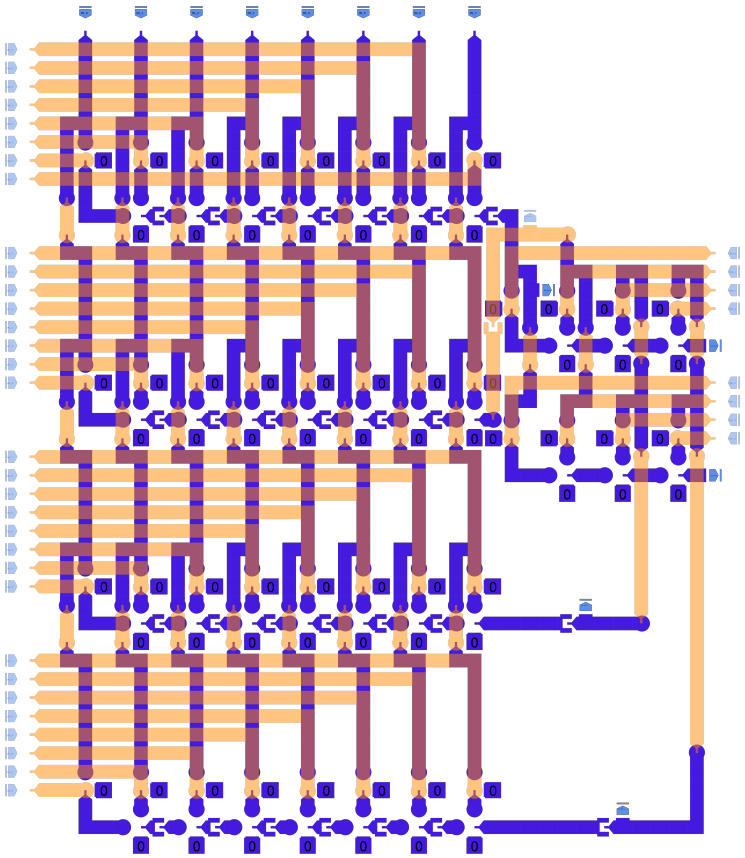


Figure 6.13: Two-layer design of the PLA in figure 3.24

can be accessed from above, below and around, here from around and either below or above) and by the need to fork some paths to make some signals reach farther (the inputs must cross all the circuit and also join some gates along the way). This latter thing was usually made by means of additional layers. Actually, voids could be found in the decoders layouts too, but here are much more cumbersome.

Let us check this point by comparing, in figure 6.14, the two variations of the circuit together. The usual colors are assigned to the drawings: red for the ones on layer 0 and green for the ones on layer 1. All the other colors are "layer independent". The solid-line, light blue rectangle outlines a single minterm within the AND plane in both circuits; the the dashed rectangles do likewise for the max terms within to OR plane. The points worth mentioning are:

- In the multilayer circuit, the inputs have the same delay for every minterm,

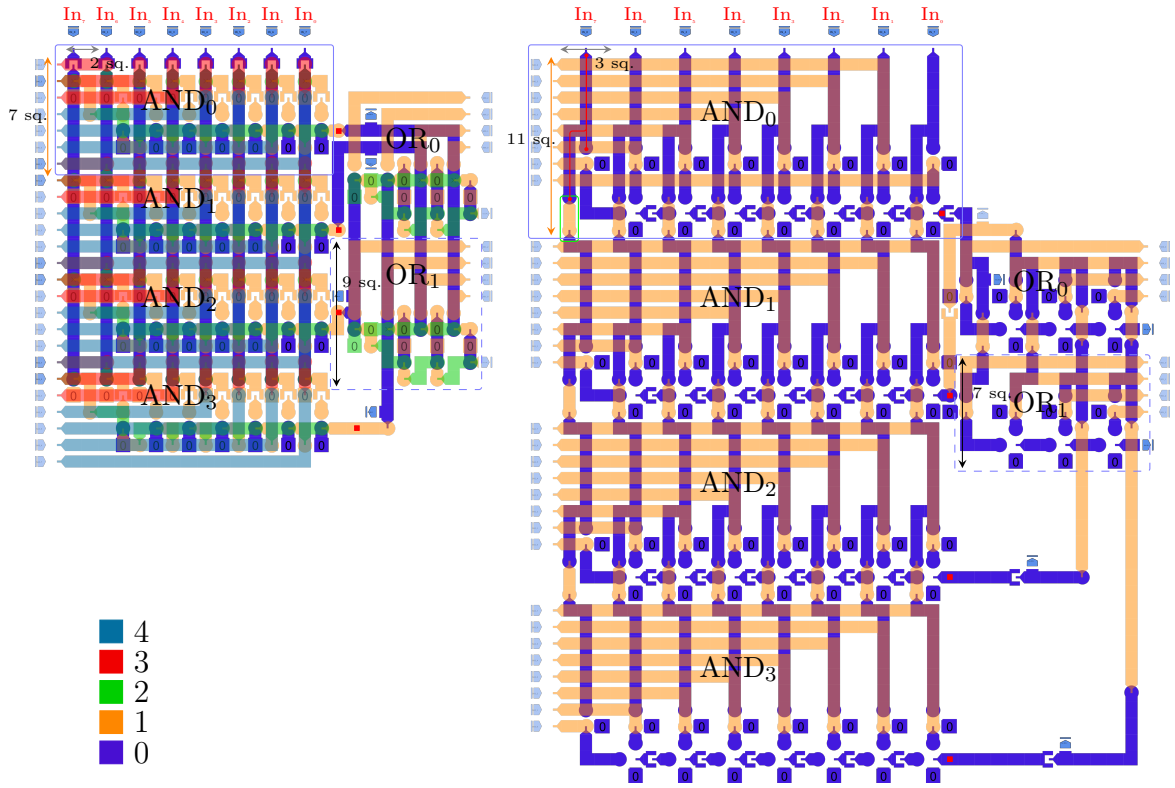


Figure 6.14: PLA in figure 3.24 compared with its two-layer version

whereas in the two-layer version the known bypass structure (green rectangle) causes the bottom minterms to get the input value later with respect to the top ones. However, in the multi layer circuit it was chosen to have all the inputs with the same delay on each minterm, which means having no inverters along the domain wall carrying the input values. A different, and probably more likely choice, could have been breaking these too long domain walls. In any case, the point is that in multi layer design there is a choice, whereas in two-layer design there is no choice;

- In the multilayer circuit, the programming inputs are arranged over multiple layers, resulting in a 7 square height of the single min term; since at most two layers are available in the other circuit, some more vertical space is needed. This results in a 11 square height. Notice that in the multi layer circuit, even if a single minterm takes 7 squares in width, it can be partially overlapped to the neighboring one, so that it is as if every minterm were 6 squares high.

Needless to say, no overlapping can be done in the two-layer version, layers are not enough;

- The same reason that makes necessary the bypass causes the branch off of the domain wall holding the values of the input (red path). This causes a one square width increase for each input (in the left hand circuit, each input needs two squares in width, in the right hand it needs three). In general, it occurs pretty frequently (but not always) that a long, single-layer domain wall in the multi layer circuit becomes in a two-layer version a wire running up and down in both layers;

- Surprisingly, in the OR plane the two-layer version max terms are shorter than their multi layer counterpart. This is probably an exception. Anyway, since the AND plane should always be higher than the OR one, the OR plane height should not matter.

Let us list now the full set of value in table 6.5

	Area		
	3 layer	2 layer	Ratio
Height	26	45	1,73
Width	26	37	1,42
Area	676	1665	2.46

	Delay			
Path	3 layer	2 layer	Ratio/Difference	
			Shortest	Longest
Input → ■	$8+4\,n_{input}$	$3+2\,n_{input}+4\,n_{and}(+1)$	0,5	0,83
■ → Output	$6+4\,n_{input}$	$5+4\,n_{input}+4\,n_{or}(+1)$	1	1,12
SA diff	$-1+2\lceil\frac{n_{...}}{3}\rceil$	$1-4\,n_{and}$	2	-14
SO diff	0	$1-4\,n_{and}+(-1)^{\lfloor\frac{n_{...}}{2}\rfloor}(4\,n_{or})$	1	

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Delays are expressed in half clock cycles. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.
n_{and} and n_{or} is the index number of the particular minterm or maxterm under exam, as defined in figure 6.14

Table 6.5: Area and delays of the PLA in figure 3.24 and of its two-layer version

6.6 SAT hardware in two layers

In section 4.3, a circuit able to carry out the SAT algorithm among four cells was shown. Since it is the most complex module of this thesis work, it is almost compulsory to present a two-layer reduction of it. The main interest in this case was the area and the difficulty of reducing such a complex circuit, originally made in 12 layer, in just two. On the other hand, timing has in part been neglected. There are a few reasons for this. One is the complexity of the circuit: the circuit has not been simulated, and analyzing tens or hundred of paths without a simulation verification is time-consuming and error-prone. Another one is that the analysis of the previous circuits has shown that timing does not change too much, and proving it with too much data will be just a waste of time. A final reason could be that the critical path of the multi layer version is known, and that almost certainly it will be the same in the two-layer version, because all the rest of the circuit is densely pipeline. Saying that the critical path is the same does not mean that the value is the same, but rather that it is the same logical path, whose length might change.

6.6.1 Local FSM

This is the FSM within the SAT cell with the highest number of states and of layers. It is also the only one discussed in full detail. Its layout is pictured in figure 6.15. The design principles are the same as the multi layer structure: the center part of the circuit contains notches, one for each state, with the output of every notch entering the input of the next one. This area is pretty small, due to the nature of the FSM: it moves to next state without conditions to be evaluated, thus no logical operations (besides the reset one) are performed from a state to the next. Then, all around run some domain walls, carrying out logical operations that define the value of each output. Figure 6.16 shows both version together. The dashed black box marks the area where the notches are placed, so that everything else, outside of that box, is the output network. The size of the two circuits may appear not widely different (the ratio is actually around 2.5), but it must be said that the multi layer circuit perhaps could have been narrower: it was designed as a 15 state machine, then three states have been removed. To take this into account, considering the width of the circuit 3 squares smaller should solve the ambiguity. Besides the size, it can be noted straight

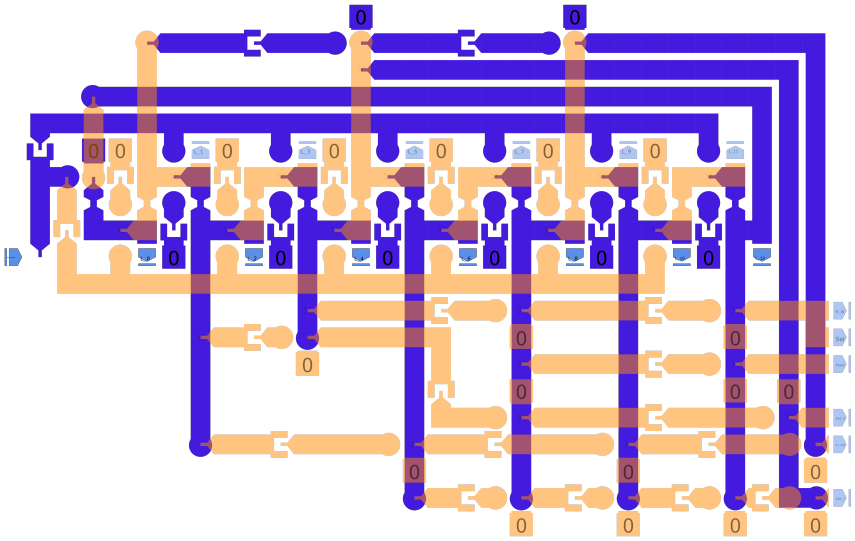


Figure 6.15: Two-layer design of the *Local FSM*

away that the two-layer version is much less dense: this is due to the spacing needed to place fixed magnets and inverters. Furthermore, the area where all the outputs are elaborated (a red box surrounds its, and the structure is the same for all the FSMs) is full of voids: the are lines where just a few fixed magnets can be found (on the top layer, green dashed lines). All the data are detailed in table 6.6.

The table shows that the area ratio is greater than it would seem, but it is within the usual factors. A general analysis of the delays proves that the timing is usually better for the two-layer version, as it was almost always the case; at most, the delay is equal. Given that this FSM design style leverages to a great extent the multi layer technology, it would seem that the area increase depends very little on the number of layers.

6.6.2 Reset FSM

This finite state machine is quite commonplace, at least considering what has already been shown and discussed. It is drawn in figure 6.17, along with its multilayer counterpart. Areas are compared in table 6.7.

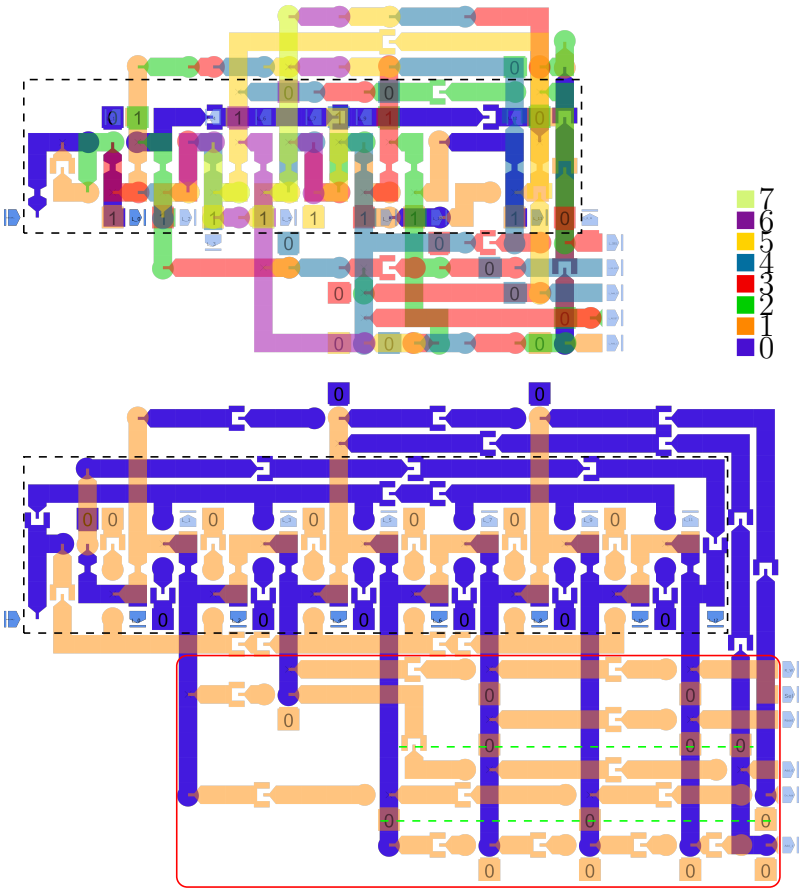


Figure 6.16: *Local FSM* compared with its two-layer version

6.6.3 Input FSM

This finite state machine is probably the most elaborated one. It has two different FSMs inside, one of them (the master) steps through its states any time the *Reset* is low, evaluating no conditions. The other one, the slave FSM, advances by a state every time the master FSM reaches the last state. Moreover, there is a XOR network that negates the output when a certain signal is high, but only while the slave FSM is in some of its states. This makes it a Mealy machine (the input-output direct link is broken by pipelining, but from the point of view of the intrinsic working of the FSM it does not matter). A pretty complex FSM indeed, which is also why it is interesting. Four main areas can be defined, each one inside a box in figure 6.18. They are: the master FSM (bottom left), the slave FSM (all but the output network), in the bottom right box, the output network of the slave

Area	Multi layer	2 layer	Ratio
Height	14	20	1,43
Width	23 (20)[1]	30	1,30 (1,5)[1]
Area	322 (280)[1]	600	1,86 (2,14)[1]

Delay

	R/W			Sel			Add_0			Add_1			En_{add}			Reset		
	ML	2L	R	ML	2L	R	ML	2L	R	ML	2L	R	ML	2L	R	ML	2L	R
S_0	-	-		-	-		-	-		-	-		-	-		-	-	
S_1	-	-		-	-		-	-		-	-		18	10		-	-	
S_2	-	-		-	-		12	8		-	0		10	8		-	-	
S_3	-	-		-	-		-	-		-	-		-	-		-	-	
S_4	10	6		10	6		4	6		-	-		-	-		-	-	
S_5	-	-		-	-		12	8		12	8		10	8		-	-	
S_6	-	-		-	-		-	-		18	10		8	6		-	-	
S_7	-	-		-	-		-	-		-	-		-	-		-	-	
S_8	8	4		4	4		4	4		10	8		-	-		4	4	
S_9	-	-		-	-		-	-		-	-		4	6		-	-	
S_{10}	-	-		-	-		-	-		10	6		4	4		-	-	
S_{11}	-	-		-	-		-	-		-	-		-	-		-	-	
S_{12}	4	2		4	2		-	-		10	4		-	-		4	2	

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Delays are expressed in half clock cycles. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.
ML= Multi Layer version; 2L= 2 Layer version; R=Ratio
[1] In this value the multi layer circuits is considered 3 squares narrower, because of what has been said about the reduction of the states in this FSM

Table 6.6: Area and delays of the *Local FSM* and of its two-layer version

FSM (top right box), the XOR switching network (top left). At design time, the hardest problem is the fact that some conditions are evaluated to decide whether or not leave the current state. This means that four signals (*Reset*, the current state signal, the previous state signal, the trigger signal, this one both negated and non negated) contribute to the determination of each state. Therefore, the solutions found consists in passing the next-state wires in the same way the output wires are drawn. This considerably stretches the layout. Alternative solutions might be found, but the easiest to find are likely to be less regular than this one. Basically, the design would be made by drawing the wires where free room can be found, by

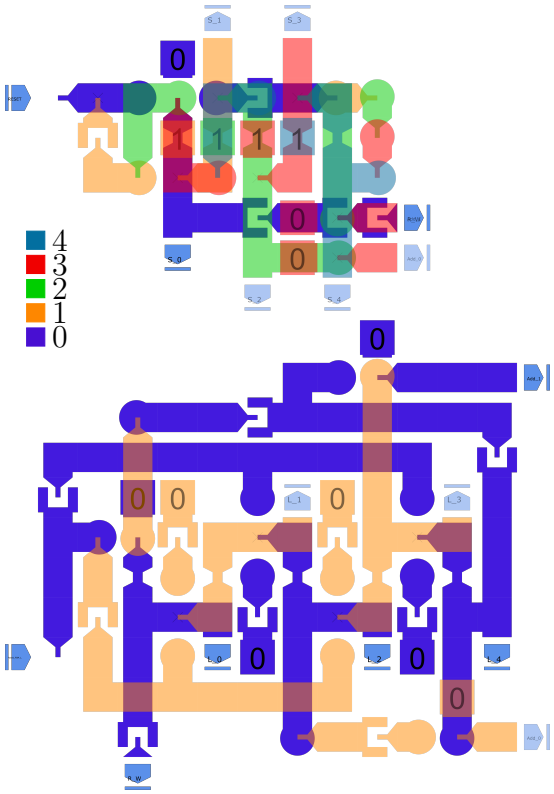


Figure 6.17: *Reset FSM* compared with its two-layer version

	Area		
	Multi layer	2 layer	Ratio
Height	6	11	1,83
Width	9	12	1,33
Area	45	132	2,93

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.

Table 6.7: Area and delays of the *Reset FSM* and of its two-layer version

a sort of "shortest distance" rule between the points to be connected; if successful, this method might yield a result even smaller than the one shown here, but it would be a trial-and-error method, and the solution would not be reproducible. Moreover, should a change be made, it would imply reconsider the whole circuit. And last, it is not guarantee that the design will be feasible: at a certain point, there might not

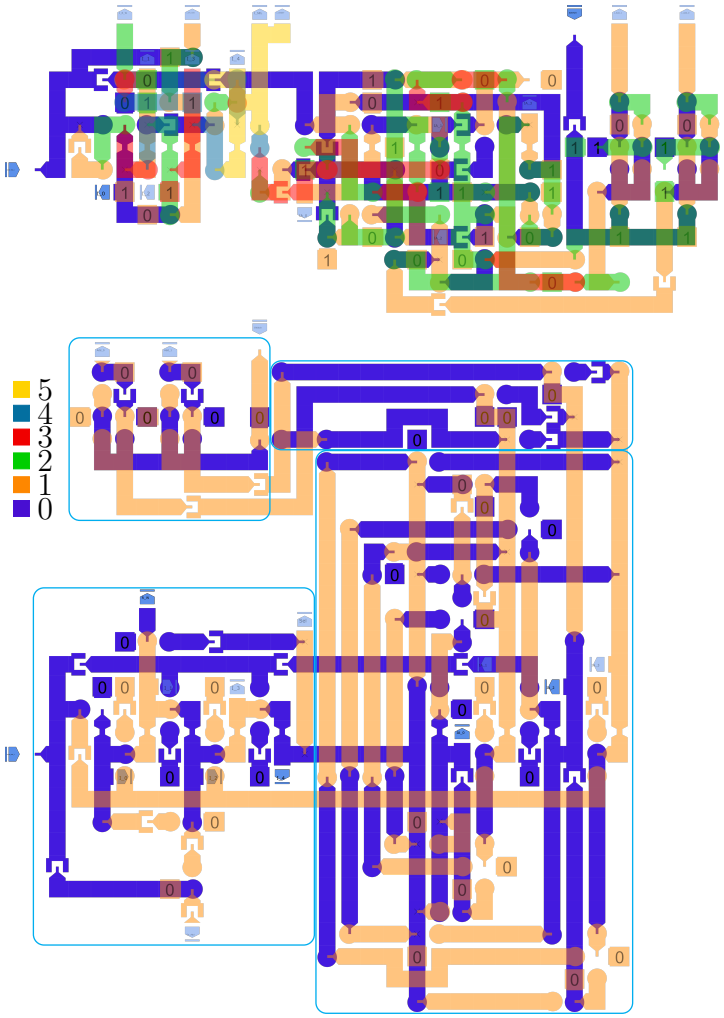


Figure 6.18: *Input FSM* compared with its two-layer version

be any more room to design the whole circuit. Table 6.8 shows the area parameters.

6.6.4 Output FSM

This FSM (in figure 6.19, with parameters in table 6.9)is much simpler than the previous one. It has just two outputs and three states. Unfortunately, it still evaluates conditions to proceed to the next state. Therefore, the same method used in the *Input FSM* has been used. There is just an interesting point: since the states are three, and all the FSMs are made such that odd-number states and even-number

	Area		
	Multi layer	2 layer	Ratio
Height	13	30	2,31
Width	31	27	0,87
Area	403	810	2,01

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.

Table 6.8: Area and delays of the *Input FSM* and of its two-layer version

states are on different layers, the next-state network between the last and the first state has been a little trickier to design, because the two states were on the same side. The details would be meaningless and complicated; this point just proves how apparently silly things become problems when layers are just two. Usually the solution impacts the area.

Another thing to point out is that, with respect to the previous FSM, this is less "regular", meaning that in this case the design rules are broken more often. This is a general trend: the more complex the circuit is, the more structured it must be. As said in the previous section, a non-structured design flow might be more effective than a structured one, and if the case is not too complex, it can carried out easily. This is a circuit so simple that a less structured approach is worth while. Table 6.9 contains the area values.

	Area		
	Multi layer	2 layer	Ratio
Height	8	13	1,63
Width	17	24	1,41
Area	136	312	2,29

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.

Table 6.9: Area and delays of the *Output FSM* and of its two-layer version

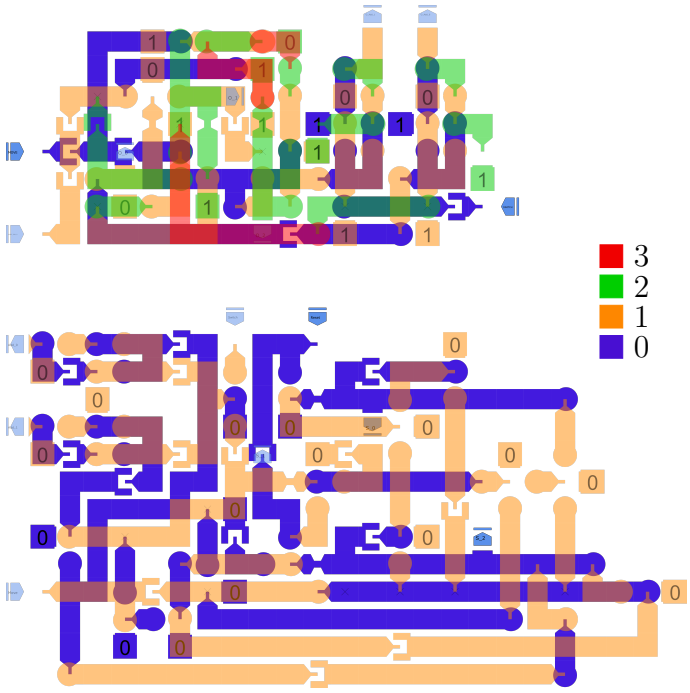


Figure 6.19: *Output FSM* compared with its two-layer version

6.6.5 SAT Datapath

The datapath of the SAT cell, originally made in three layers, has been redrawn. It is made with two of the components described: the full adder and the multiplexer. The basic memory cell is also a variation of two cascaded multiplexer, so that in practice the datapath is only made of multiplexers and full adders. Actually, the only thing missing is an half adder, but there is always just one of the, regardless of the bit parallelism. Since it is a composition of elements, it offers very little insight. The assembling of the component has been quite smooth, maybe the only thing worth mentioning involves the wires: in two or three cases there are wires running in parallel (as it is shown in figure 6.20, green rectangles) that increase the area of the circuit. In multi layer, they could be placed on independent layers, could have been stacked and some area saving would have been possible. Remember also that, even if here they have not been placed, usually long wires have inverters along them; this makes unavailable even the area over/under them in the only layer left. The inverters needed to balance the delays are missing very often, because these circuits have not been simulated. Comparing area and the critical path is the main

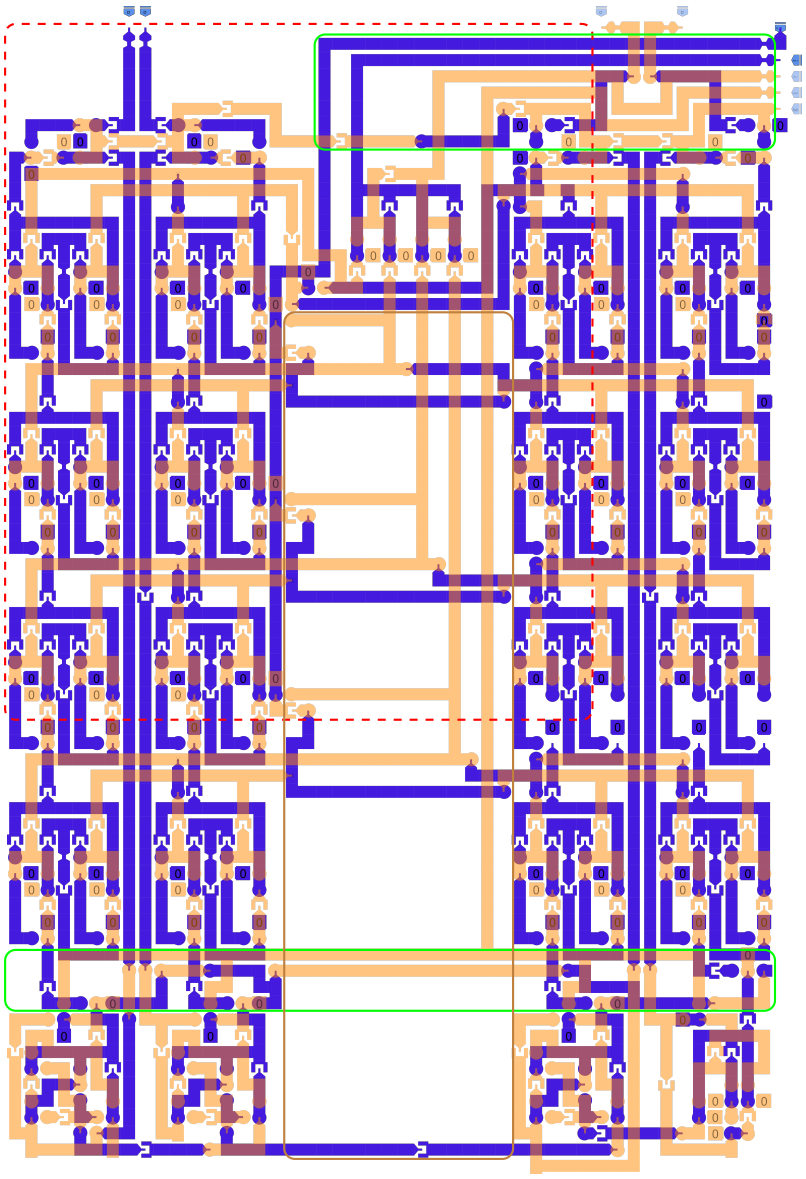


Figure 6.20: Two-layer design of the SAT cell datapath

objective. Along the critical path, however, they are placed, to have a more precise estimate of the its length. In any case, not having simulated them requires to allow a little of "tolerance". The brown box outlines a very large empty space. This is a bad feature of the circuit, but has a good consequence too: the room to allocate the inverters that trim and balance the delays would hardly be missing. This is one more reason not to worry too much about the precise timing of the non critical

paths of the circuit, and a path where inverters must be inserted to balance the delays cannot ever be a critical path. There is no picture with both versions here: given the size of them, it would have been unclear. However, a red dashed rectangle is drawn in figure 6.20, marking what would have been the perimeter of the multi layer version. Table 6.10 shows the area parameters and the length of the critical path.

	Area		
	Multi layer	2 layer	Ratio
Height	43	71	1,65
Width	36	48	1,33
Area	1548	3408	2,20
Critical path Critical path [half clock cycles]	58	39	0.67

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.

Table 6.10: Area and delays of the datapath of the SAT cell datapath and of its two-layer version

6.6.6 Full SAT Cell

Once all the elements have been assembled together, the result is that of figure 6.21. There are just a few comments to make about this circuit. The most interesting point is about the area. It is shown in detal in table 6.11, but from the image one can tell that the area has more than doubled. Also, placing elements and connecting them when just two layers are available, and when the elements themselves are made in two layers (so that there is no chance to tailor their shape exploiting layers above), yields a layout with a lot of wasted room. Table 6.11 shows the area parameters. The increase in area is the greatest ever seen in the chapter, because a circuit whose components where piled up one over the other has been spread all on the same level. A good amount of notches have been inserted along the control paths, implying that none of them might ever be longer than the critical path within the datapath, neither there is a chance to sum paths of the control section with paths of the datapath, because a line of notches separates them. Thus, the critical path is the same as the datapath alone.

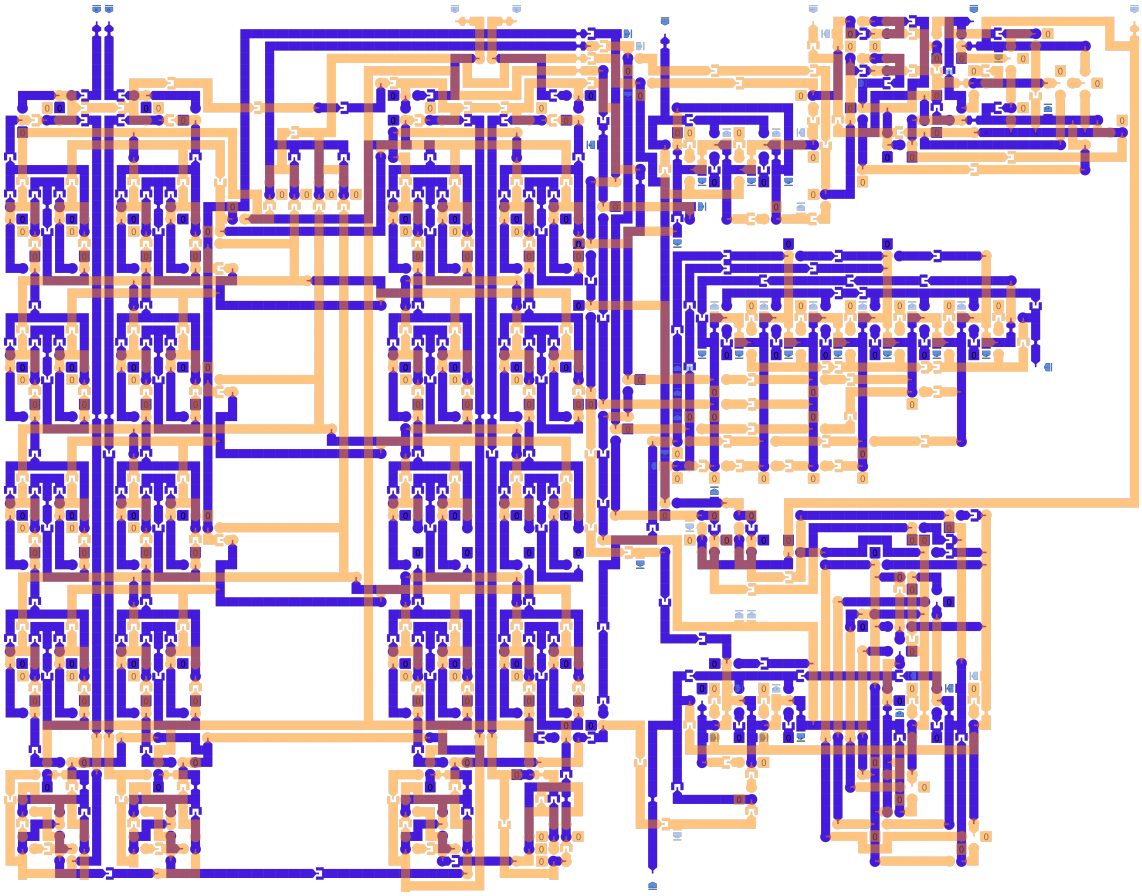


Figure 6.21: Two-layer design of the SAT cell

	Area		
	Multi layer	2 layer	Ratio
Height	43	71	1,65
Width	36	92	2,63
Area	1548	6532	4,22
Critical path [half clock cycles]	58	39	0.67

Height, width and domain wall length are expressed in arbitrary length units. Areas are expressed in (arbitrary length units)^2. Ratios are expressed as $\frac{2\ layer\ version}{multi\ layer\ version}$.

Table 6.11: Area and delays of the SAT cell and of its two-layer version

To summarize the most important findings of the analysis in this chapter, the reduction in two layers of circuits originally made in many layers (up to 12 in this

chapter) is always possible. The area increases, not too much for simple gates, but the ratio gets higher and higher as the circuit gets more complex. In the examples seen in here, the greatest factor was 4,33. Complex though it may seem, it just carries out a very simple algorithm and has a bit parallelism equal to four. It should not be unlikely a 10 or 20 factor when the design comes to more articulated algorithms or words as wide as 8, 16 or 32 bits. Nonetheless, the circuits have been redrawn with the lowest number of layers (besides the single layer case). One or two layers more would certainly reduce the area. Things are much better for the timing: to a first approximation, it does not change in one way or the other. It might change, but usually very little. A very conservative and safe factor could be the two-layer version delays as the double of the original one. Quite often, this is a huge overestimation; in any case, it does not take much, at the first stages of the design, to take a better guess.

Chapter 7

Technological Parameters Tuning

This chapter is devoted to a technological analysis in order to find out how the designs made so far could increase their speed. The aim is finding which technological parameters influence the clock period, and check whether they can be set to the desired value. Changing these parameters does not affect the layout of the circuit, that would keep working in any case; on the other hand, some modifications (namely, reduction of the length of the domain walls) joined with the tuning of the technological parameters, could remarkably improve the performances. In order to understand how the technological parameters influence the clock speed, a brief analysis of the theory of domain walls motion must be done. The sources for this part are the works [13] and [14], but here only the theoretical elements strictly necessary for the purpose said above will be mentioned, and their description will not be rigorous. The reader interested in a deeper analysis is kindly addressed to those works.

7.1 Delay Contributes

There are two contributes to the delay of a switching nanomagnet: it takes some time for the clock field, along with the magnetic fields of the adjacent nanomagnets,

to reverse the nucleation in the ANC spot; then, this reversed magnetization vector must propagate and reach the far end of the nanomagnet. This end will be coupled with the ANC of the next nanomagnet, and so on. These two contributions are called respectively *Nucleation delay* and *Propagation delay*. Thus, the delay is described completely by means of these two parameters, but there is a further constraint. The description of the nucleation delay is a probabilistic one: one can say that, after a certain amount of time, with definite conditions, the ANC has a certain probability to have its magnetization vector reversed. Let us consider a nanomagnet in an inverter, where the ANC is just sensitive to the magnetic field of one other nanomagnet (it could be called "input nanomagnet"). Let H_{clock} and $C_{nanomagnet}$ be the magnetic field in the ANC spot, where the contributions are respectively the one of the clock and the one of the neighboring nanomagnet. If the direction of the two fields is the same, the absolute value of the overall magnetic field is $H_{clock} + C_{nanomagnet}$, if the directions are opposite, that same quantity is $H_{clock} - C_{nanomagnet}$. In the former case, the ANC magnetization vector must be reversed; in the latter case, the ANC magnetization vector must *not* be reversed [15],[16]. In simpler words, the field of the nanomagnets coupled with the ANC under consideration must be determinant for it to nucleate: if their direction matches the one of the clock, the ANC nucleates, otherwise, the directions are opposite, and the clock must not be able to nucleate them. In formulas:

$$P(t_{nuc}, H_{clock} + C_{nanomagnet}) \to 1 \tag{7.1a}$$

$$P(t_{clock}, H_{clock} - C_{nanomagnet}) \to 0 \tag{7.1b}$$

For a majority voter, where more nanomagnets are involved, the worst case is considered (which is two nanomagnets matching the clock direction, one opposing it). This condition is a very tight constraint: the clocking field cannot just be set so that the time it takes to nucleate is very short, because most likely that would make the nucleation quite probable even when it should not occur. Let us write down the equations that described these delays. They will be full of terms not yet explained. The most important ones will be briefly described later, whereas the ones that are not mentioned can be considered constant values that, for whatever reason, cannot

be changed. The propagation delay can be calculated by means of:

$$v_{propagation} \begin{cases} 1 \ \frac{m}{s} & \text{if } H_{pulse} < H_{int} \\ v_{00} \ e^{E_{pin} \frac{H_{int}}{H_{pulse}}} & \text{if } H_{int} < H_{pulse} < 330 \ Oe \\ v_0 + \mu_v (H_{pulse} - H_{int}) & \text{if } 330 \ Oe < H_{pulse} < 750 \ Oe \\ 57 \ \frac{m}{s} & \text{if } H_{pulse} > 750 \ Oe \end{cases} \tag{7.2}$$

Where H_{int} a the so called "intrinsic pinning field" (the field needed to release a pinned domain wall at zero temperature); H_{pulse} is the external magnetic field applied to the circuit (for sake of simplicity, just the clock field is taken into account); μ_v is a proportionality constant between field and speed; and E_{pin} is the energy barrier. None of these quantities are particularly interesting, except H_{pulse}. In our applications, its value is above 500 Oe, so that the two bottom equations are the interesting ones for us. They show that the domain wall speed is proportional to the field in a linear fashion, then speed saturates for $H_{pulse} = 750 \ Oe$. The propagation delay will be:

$$t_{propagation} = \frac{DW_{length}}{v_{propagation}} \tag{7.3}$$

Which is the reason why too long nanomagnets increase the delay: the numerator becomes higher. Nucleation delay, on the other hand, behaves according to this equation:

$$t_{nucleation} = -\frac{1}{f_0} \ e^{E_0 \left(1 - \frac{H_{eff}}{H_0}\right)^2} \ \log\left(1 - P\right) \tag{7.4}$$

It is easier to understand broken down in two equations:

$$P_{nuc}(t, H_{eff}) = 1 - exp\left(-\frac{t}{\tau(H_{eff})}\right) \tag{7.5a}$$

$$\tau(H_{eff}) = \frac{1}{f_0} exp\left(\frac{E_0 (1 - \frac{H_{eff}}{H_0})^2}{k_b T}\right) \tag{7.5b}$$

The terms E_0 and H_0 can be expanded as:

$$E_0 = V_{anc} K_{eff} \tag{7.6a}$$

$$H_0 = \frac{2K_{anc}}{\mu_0 M_s} \tag{7.6b}$$

Rough definitions of all these parameters could be:

- $P_{nuc}(t,H_{eff})$ is the probability of nucleating an ANC by exposing it to an external field equal to H_{eff} for the time t;
- V_{anc} is the volume of the ANC;
- K_{eff} is the effective anisotropy constant. As said in the introduction, the anisotropy is the property of a generic field to "prefer" a particular spatial direction, where by "prefer" it is meant that that direction is the most energetically favorable. It is expressed as Jm^{-3}, an energy density. When multiplied by V_{anc}, it could be interpreted as the additional energy needed for the magnetic field to align the magnetization vector of the ANC to a direction perpendicular to the easy axis. Therefore, the higher K_{eff}, the more is the energy difference between the directions. E_0, thus, is considered an energy barrier related to the anisotropy of the magnet;
- K_{anc} is a quantity linearly proportional to K_{eff};
- H_0 is the coercive field (or coercivity), the magnetic field needed to flip the magnetization vector of the ANC at zero temperature;
- M_s is the saturation magnetization of the nanomagnet, and depends on the geometrical characteristic of it;
- f_0 is called "attempt frequency", it is important because it is the inverse of the minimum possible value of τ, but here it will never be changed.

H_{eff} is the external field the ANC senses. It includes both the clock field and the stray fields of the neighboring nanomagnets, so that its expression could be:

$$H_{clock} \pm C \tag{7.7}$$

Where C is the overall contribution to the field sensed by the ANC and coming from the nanomagnets. It must include all the contributes in a majority voter, taking into account whether their sign is the same or the opposite, and so on. Now, the conditions of 7.1 must be substituted into the 7.5. The simplest case is the inverter, where there is just one input nanomagnet coupled with the ANC; this coupling field could be either opposing to clock or supporting it. In the former case, $H_{eff} = H_{clock} - C_{inv}$, in the latter, $H_{eff} = H_{clock} + C_{inv}$. The substitution results in these conditions:

$$\begin{cases} P_{nuc}(t_{nuc},H_{eff}) = 1 - exp\left(-\frac{t_{nuc}}{\tau(H_{eff})}\right) \rightarrow 1 & H_{eff} = H_{clock} + C_{inv} \\ P_{\overline{nuc}} = P_{nuc}(t_{clock},H_{eff}) = 1 - exp\left(-\frac{t_{clock}}{\tau(H_{eff})}\right) \rightarrow 0 & H_{eff} = H_{clock} - C_{inv} \end{cases} \tag{7.8}$$

In the first row P_{nuc} is the probability to nucleate when the input nanomagnet and the clock field have the same direction. The time span considered is t_{nuc}, the nucleation delay. The second row has $P_{\overline{nuc}}$, the probability not to nucleate the ANC. In this case, the ANC must not nucleate, because the clock field and the input nanomagnet's field have opposite directions. The time span considered is now half clock cycle (t_{clock}), because the ANC is exposed to that field for that time. Notice that the two time constants (τ) are different. The one related to P_{nuc} should be as short as possible (the circuit would spend less time nucleating the ANC and will be faster) and the one related to the $P_{\overline{nuc}}$ should be as long as possible (it would take very long to nucleate the ANC, so that even if the clock period is long, the ANC does not reverse its magnetization). These time constants could be called respectively "nucleation time constant" and "non nucleation time constant". These conditions will be mentioned very often, so that they will be called for brevity "probability conditions" or "feasibility conditions".

7.2 Clock Field

The first parameter that could be considered is the clock field. The nucleation delay has a minimum for a particular value of the clock delay, $H_{clock} = H_0 - C$. However, that value is the minimum just for the nucleation delay, not for the overall delay. Moreover, it could be the case that the conditions 7.1 are not met for that value.

And last, there are actually three different C values, so that the scenario is much more complex. Let us have a look at figure 7.1. The vertical line shows where the

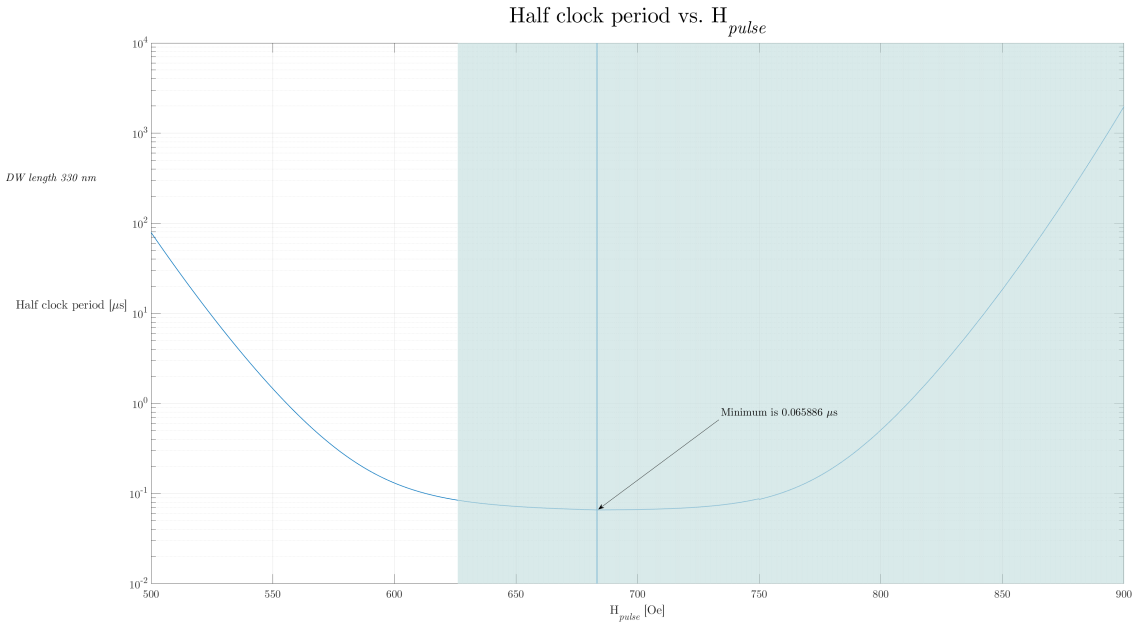


Figure 7.1: Half clock period vs. clock field, critical path equal to 8 squares

minimum delay is found. Unfortunately, it is inside the shaded area, that represents the clock values where the 7.1 conditions are not met. The other area could be called in future "safe area", and the clock field values belonging to it "allowed field values". Figure 7.2 represents a plot made with the same conditions, but with shorter nanomagnets. The minimum is lower, but the most important thing is that in this case the shaded area is smaller. The nanomagnets are shorter, so that they are crossed in a shorter time, and the clock period too becomes shorter. Therefore, the ANC is exposed to the field configuration that must not nucleate it for less time, and the nucleation is less likely. This just provides a further reason not to have too long nanomagnets.

7.2.1 Delay components

It could be interesting to check separately how the two components of the delay, namely the nucleation delay and the propagation delay, change with the clock field intensity. Of course the general trends are known from the formula, but also the

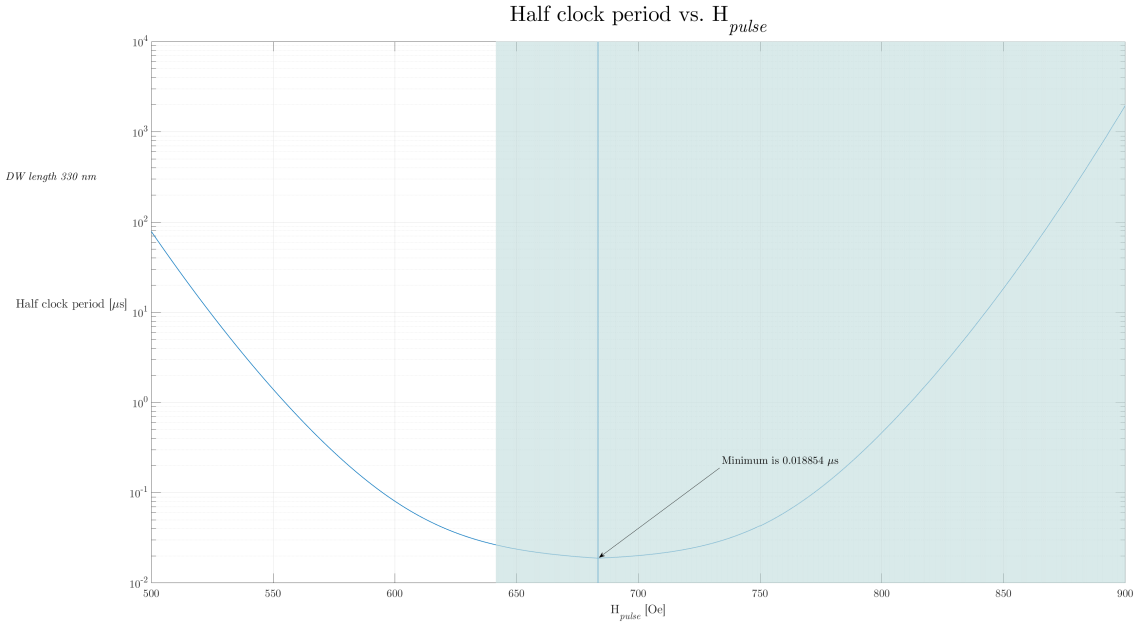


Figure 7.2: Half clock period vs. clock field, critical path equal to 2 squares

numerical values are needed, in order to know if something is negligible, for which field values, and so on. Figure 7.3 shows it. The minimum value of the nucleation delay is much smaller than the one of the propagation delay, by about a factor of 10. Nevertheless, the propagation delay depends on the length of the nanomagnets. In this cases the maximum length has been 8 squares: it is not an extremely high value, and yet in a more performance oriented design could be halved. Notice also that the minimum of the nucleation delay is in a region where the probability conditions are not met. Last, there is a cusp in the minimum. The reason is that the nucleation delay includes the factor C in its definition. This term represents the overall magnetic field (only the one coming from other nanomagnets, the clock field is separately accounted for) that an ANC is sensitive to. Since pNML is a 3D technology, a single ANC might be coupled with two other magnets placed in the same plane and one above, or just one in the same plane, or any other possible configuration. However, there are just three kinds of coupling: the one of an inverter, the one of a via (two nanomagnet coupled one over the other), the one of a majority voter. By "kinds of coupling", it is meant that for manufacturing reasons, the distances between the nanomagnets are different for each one of these

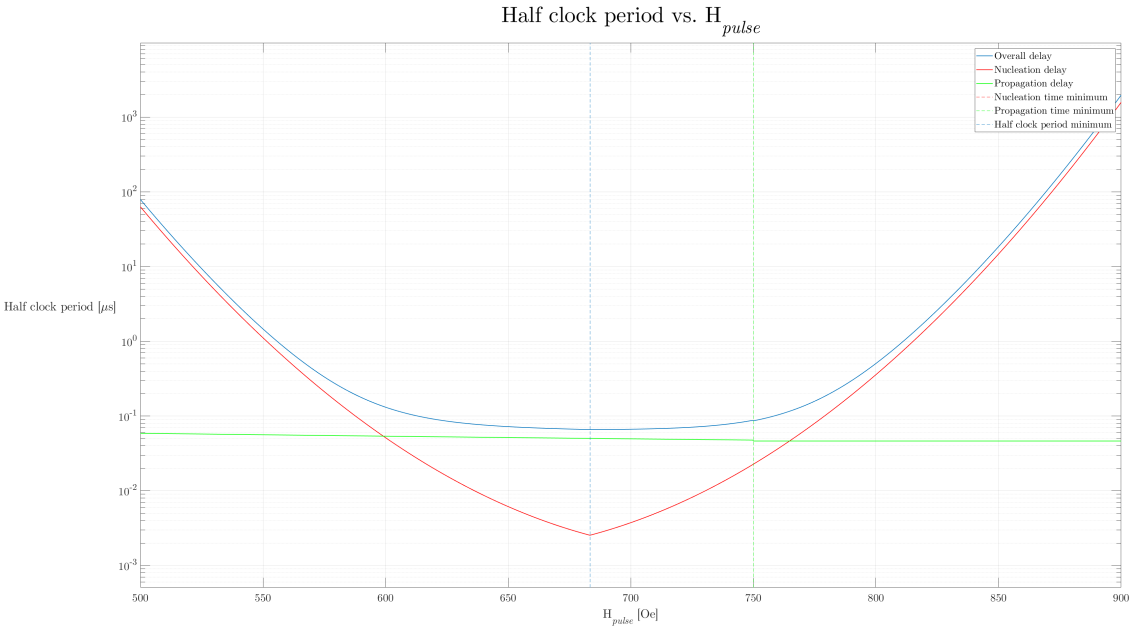


Figure 7.3: Half clock period vs. clock field, by each delay component. The slight discontinuity for $H = 750\ Oe$ is due to the model approximation when passing from the linear region to the saturation region

cases, and the coupling fields are different. Hence, the nucleation delay must be calculated separately for the three kinds of coupling, and then the worst case is selected. Figure 7.4 shows the nucleation delay separately for each of these three elements. The green, thicker line is maximum value for each clock field value. By looking at the plot it is very easy to understand why there is that cusp. This difference in the coupling factors of the three elements is what makes an analytical study of these expressions quite complicated.

7.3 Propagation Delay

Studying the propagation delay is crucial, because is the only one of the two delay components that can be determined when designing the layout. However, it is quite obvious that the shorter the critical path, the shorter the propagation delay becomes. The critical path might also get shorter because the nanomagnets used are smaller (meaning that the technology is scaled). In this case, another factor is to be considered. In equation 7.5b, the volume of the ANC is inside the exponent. If the

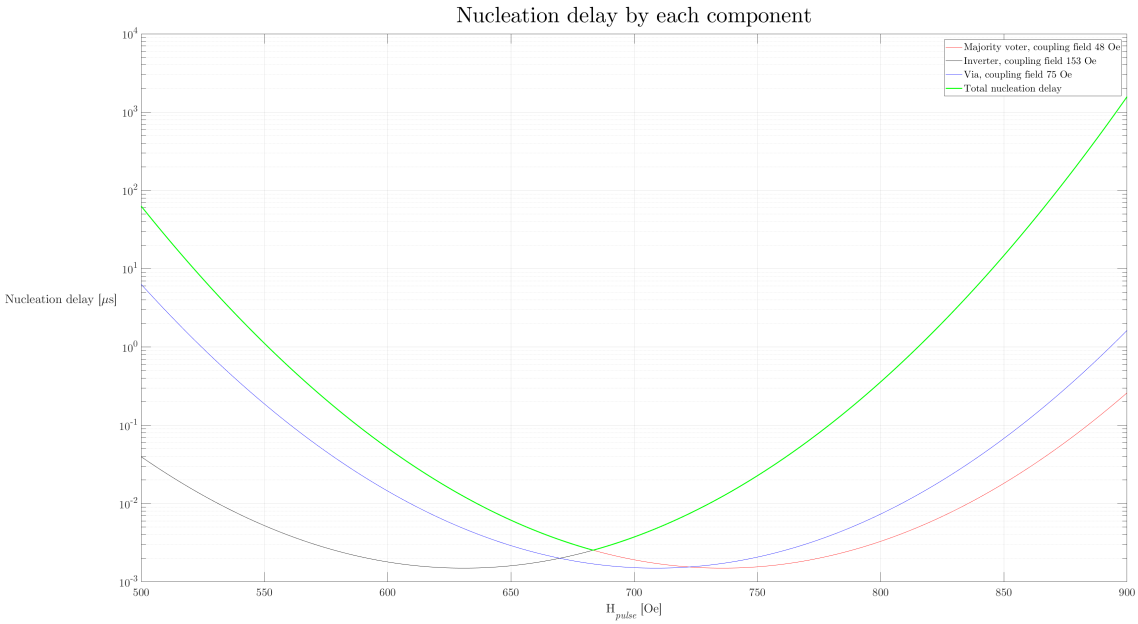


Figure 7.4: Nucleation delays vs. clock field, by each delay component

technology is scaled, the volume of the ANC could be expected to scale accordingly. This reduces both time constants; the nucleation time constant can be minimized in other ways (as will be shown later on), so that the main effect of this ANC volume scaling is decreasing the non nucleation time constant. This makes the probability conditions harder to meet. Figure 7.5 contains a plot where the ANC is scaled along with the nanomagnets' size. In the plot the scaling values where the 7.1 conditions are not met are highlighted with a red, dotted line. It is clearly shown that the as soon as the ANC volume is reduced, the conditions are no longer met. Therefore, when scaling the technology the ANC volumes shoud be kept as big as possible.

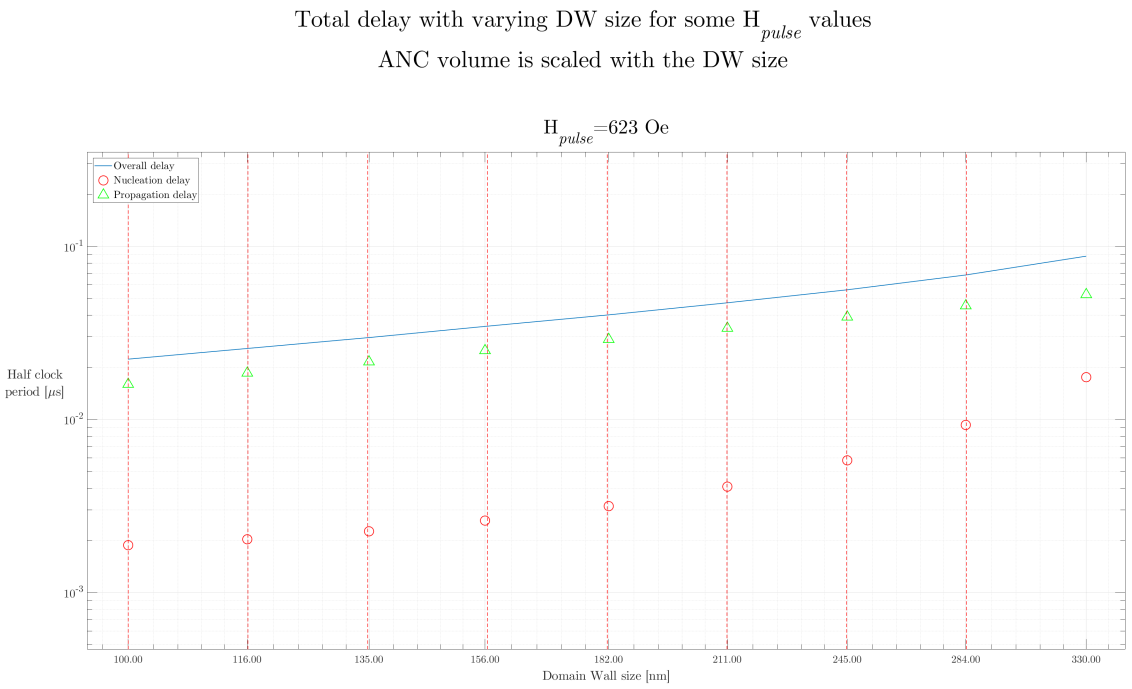


Figure 7.5: Half clock period vs square size, with ANC volume scaled by the same factor

7.4 Anisotropy

As already mentioned, the anisotropy is one of the key features behind the working mechanism of this technology. Equations 7.5b and 7.6b show that if $K_{eff} \to 0$, the nucleation time constants increase; so they do when $K_{eff} \to \infty$. Thus, there should be a minimum between these extremes. In this work, $K_{eff} = 2 \cdot 10^5 Jm^{-3}$, so let us draw a plot around this anisotropy value, as done in figure 7.6. The plot proves that the trends for very high and very low values is the one just stated, and also proves that the minimum discussed above actually exists. As happened for the field plots in figure 7.1 and 7.2, the minimum is located in a region where the probability requirements are not satisfied. Besides that, the standard anisotropy value used seems to be very close to the minimum. Since the range of that plot is too large, figure 7.7 shows that area. The standard value used is actually close to the minimum, and given that the delay is really sensitive to the anisotropy, it is advisable not to change its value: the only region where the sensitivity decreases is the one containing the standard value. Tyring to set an anisotropy value where the sensitivity is high might cause uncertainty-related issues.

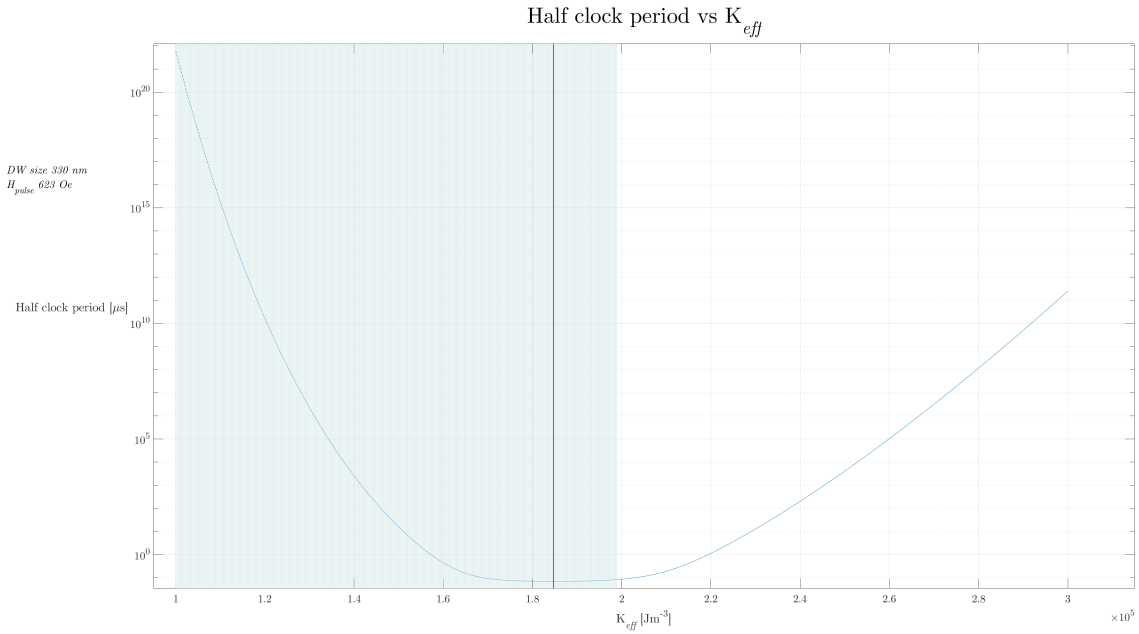


Figure 7.6: Half clock period vs. anisotropy constant

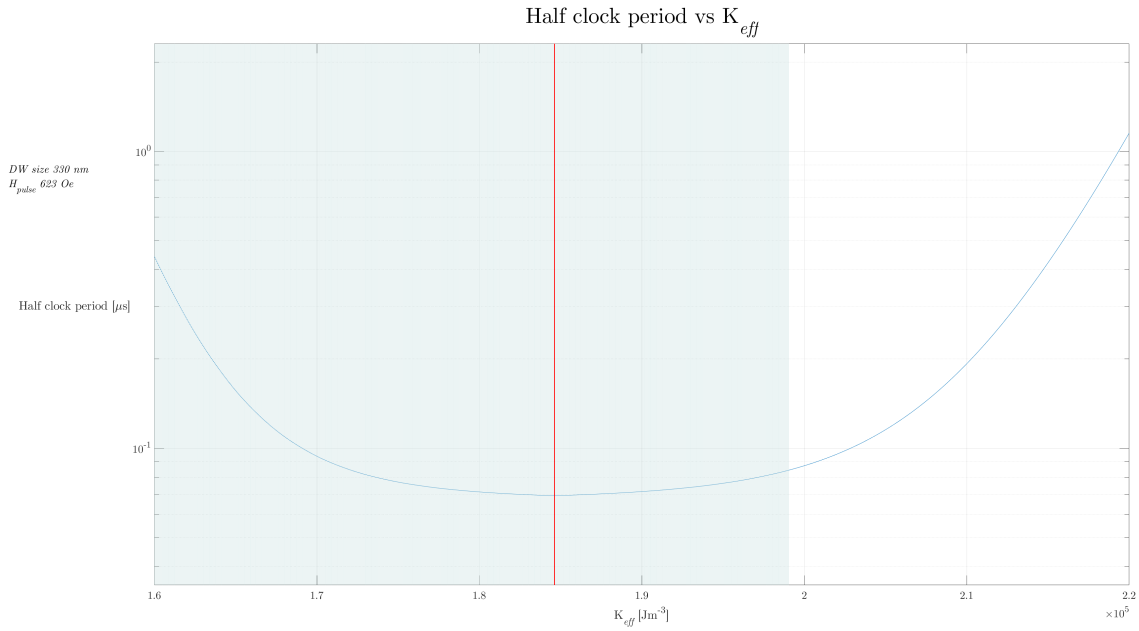


Figure 7.7: Half clock period vs. anisotropy constant, zoomed

The two previous plots have been drawn with the clock field value that minimized the delay according to the plot 7.1. If the anisotropy is changed, that value could

be no longer the best one. Figure 7.8 draws a two dimensional plot, with both the anisotropy and the clock field as free variables. The plot has been drawn only where the delay value is not too high, and where the probability conditions are met. By

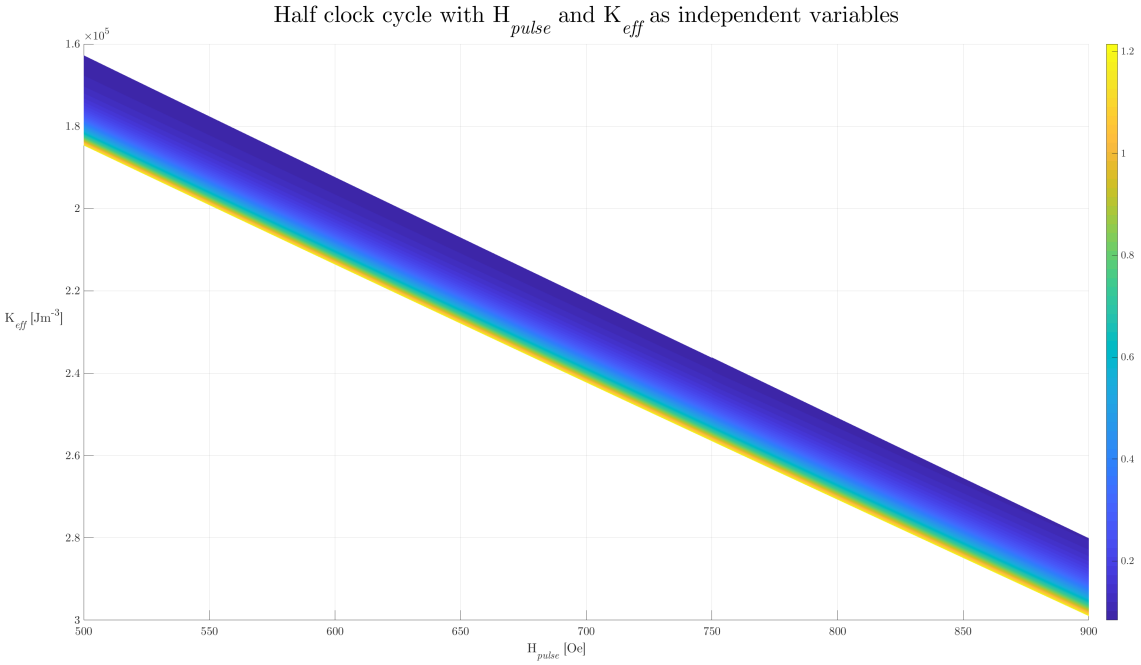


Figure 7.8: Half clock period vs. anisotropy constant and clock field

looking at the plot one gathers that any delay value can be obtained by different pairs of the two free variables. Therefore, there is no need to consider the effect of both of them. One could choose to use for example, the anisotropy value that suits best the manufacturing process in use, and change only the clock field, or the other way around.

7.5 Coupling Fields

The intensity of magnetic field generated by a nanomagnet and sensed by an ANC (that is, the coupling field) plays a crucial role. The reason can be found with a simple analytical study of equation 7.5b, rewritten here for convenience:

$$\tau(H_{eff}) = \frac{1}{f_0} exp\left(\frac{E_0(1-\frac{H_{eff}}{H_0})^2}{k_b T}\right) \tag{7.9}$$

If the C were always the same for all the kinds of coupling, the two time constants would have been:

$$\tau_{nuc} = \frac{1}{f_0} exp\left(\frac{E_0(1-\frac{H_{clock}+C}{H_0})^2}{k_b T}\right) \tag{7.10a}$$

$$\tau_{\overline{nuc}} = \frac{1}{f_0} exp\left(\frac{E_0(1-\frac{H_{clock}-C}{H_0})^2}{k_b T}\right) \tag{7.10b}$$

τ_{nuc} is minimum for $H_{clock} + C = H_0$, and the resulting $\tau_{\overline{nuc}}$ would be:

$$\frac{1}{f_0} exp\left(\frac{E_0(\frac{2C}{H_0})^2}{k_b T}\right) \tag{7.11}$$

Then, by tuning of either C or E_0, the probability condition for $\tau_{\overline{nuc}}$ could be satisfied. The problem is much more complicated by the fact that three different C values exist, and the solution must encompass all of these cases. Trying to reduce the difference among the C values could be a solution: according to this analysis, it should cause more clock field value to satisfy the conditions. Figure 7.9 shows a multi window plot where two of the three C values progressively approach the third one. The value all the coupling parameters tend to is the one of the inverter, which is also the greatest one. The number above each plot is a difference factor, equal to one when the differences have not been changed, and equal to zero when the three elements exhibit all the same coupling field. This plot suggests that if the coupling field is similar for all the cases, the safe area increases: notice that if the difference is halved, the safe area includes the minimum. Even if the difference becomes just the 80% of the original value, a delay very close to the minimum will be obtained. However, the value the coupling fields tend to must be defined. Figure 7.10 shows what happens if the coupling field value tend to the smallest one rather than the greatest one. The four plots are almost identical. This means that the value the coupling fields tend to (let us call it *target value*) should be chosen with some care. For example, if the target value is lower than the lower coupling field value, the region where the probability conditions are satisfied might become smaller, as in figure 7.11. It could be guessed that the target value should at least be greater or equal to the greatest coupling field. Actually, some trials have shown that any

Delay at varying H_{pulse} with C values converging to 153 Oe

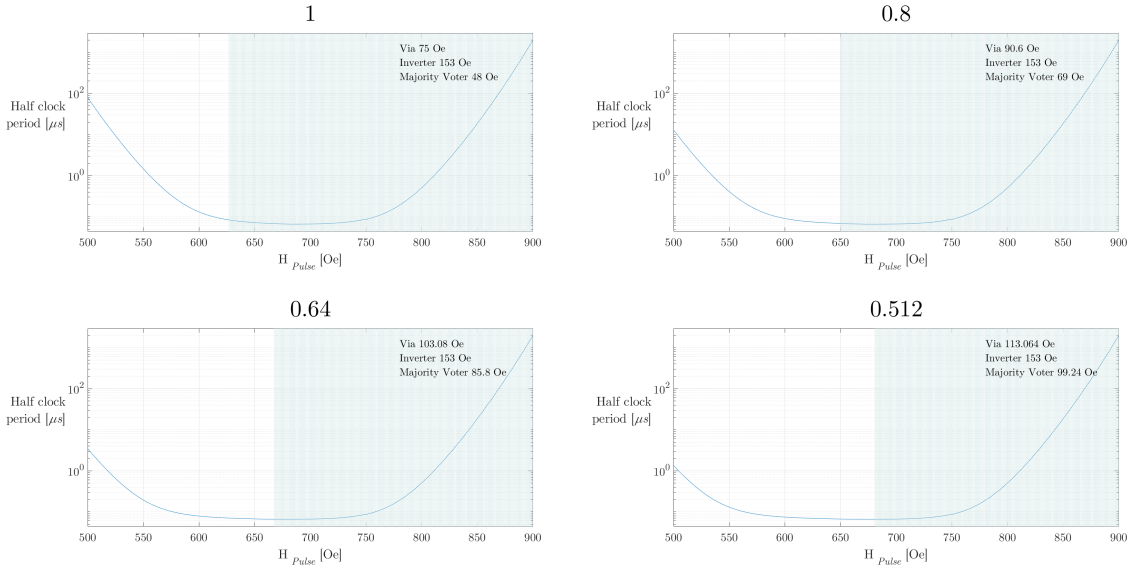


Figure 7.9: Half clock period vs. clock field, with C fields tending to 153 Oe

Delay at varying H_{pulse} with C values converging to 48 Oe

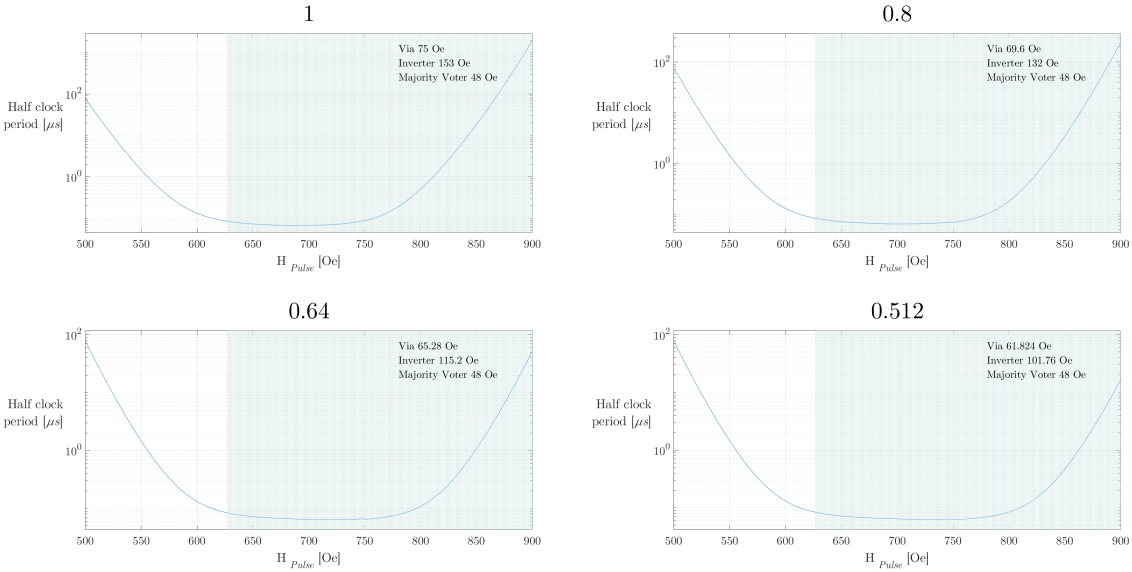


Figure 7.10: Half clock period vs. clock field, with C fields tending to 48 Oe

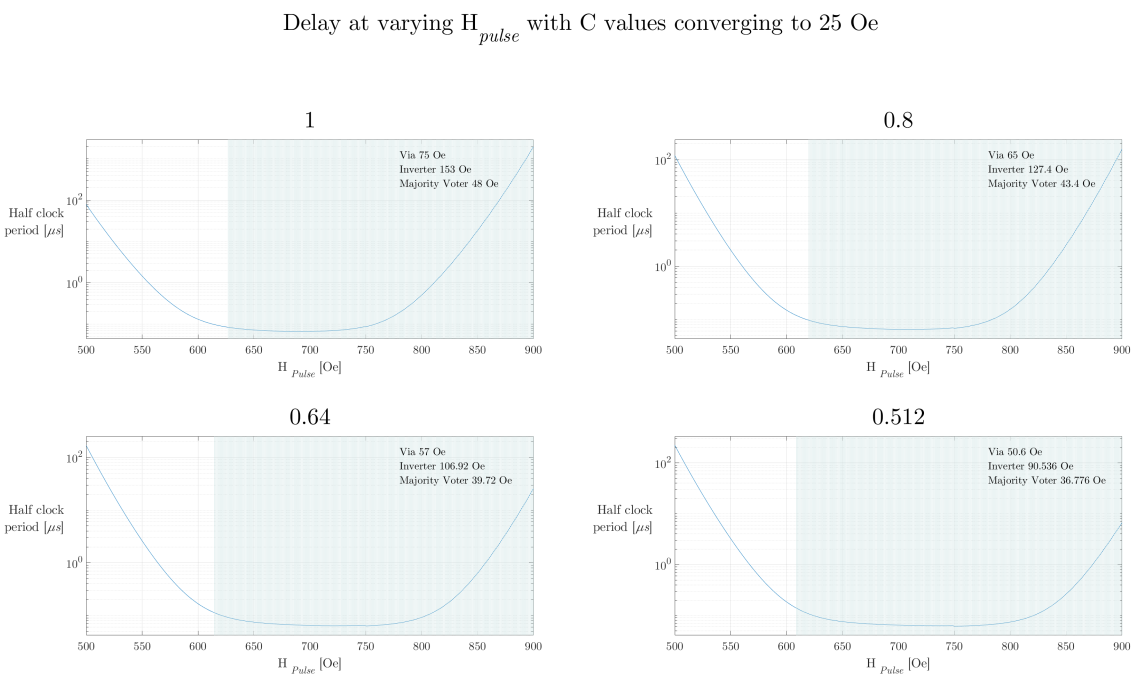


Figure 7.11: Half clock period vs. clock field, with C fields tending to 25 Oe

value greater than the lowest coupling field should be fine, but exceeding the value of the greatest coupling field is probably safer. Anyways, each design case could be studied in its particular details, the aim here is just finding roughly the trends. A last test could be done increasing the values of the coupling fields, without having them converging to a common value. The outcomes are in figure 7.12, and are quite interesting. The number above each plot is the factor each coupling field is multiplied by. The safe area does not seem to move much, but the whole curve shifts towards the left, with the minimum point ending up inside the allowed clock field values. Therefore, just increasing the coupling field, without evening it out among all the elements could be worth a try.

All this analysis should allow to reduce, in one way or the other, the nucleation delay to its minimum value. Nonetheless, before taking too much effort to do so, the contribute of the propagation delay must be considered. If it is the dominant one, engineering the nucleation delay is completely pointless. For example, if the scenario is the one in plot 7.3, the first delay to consider is the propagation delay; this is also the most common situation.

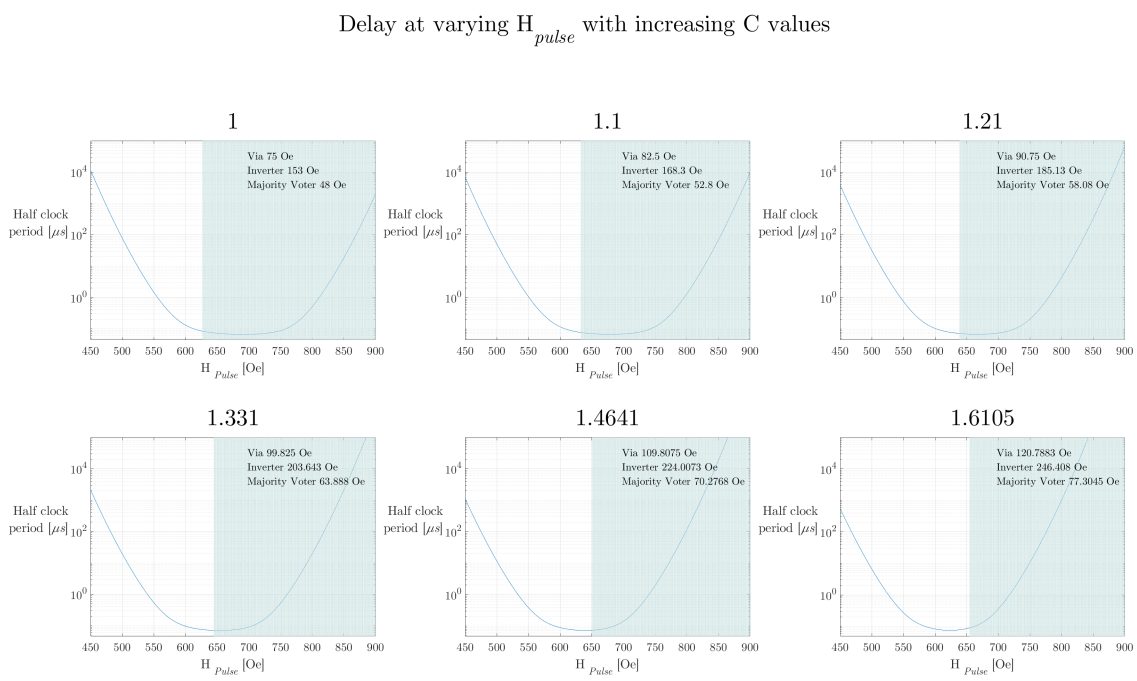


Figure 7.12: Half clock period vs. clock field, with progressively increasing C fields

To conclude the chapter, figure 7.13 shows the situation for the SAT architecture designed in chapter 4, with the longest path equal to 22 squares, and the square size equal to 330 nm. When it had been designed, the parameters were such that the nucleation delay was the most important contribute, so that it was not strictly needed to keep the nanomagnets short. This example suggests that, when a design is started, it would be advisable to consider in the first place the technological optimization allowed, in order to know whether drawing short domain walls is worth it or not.

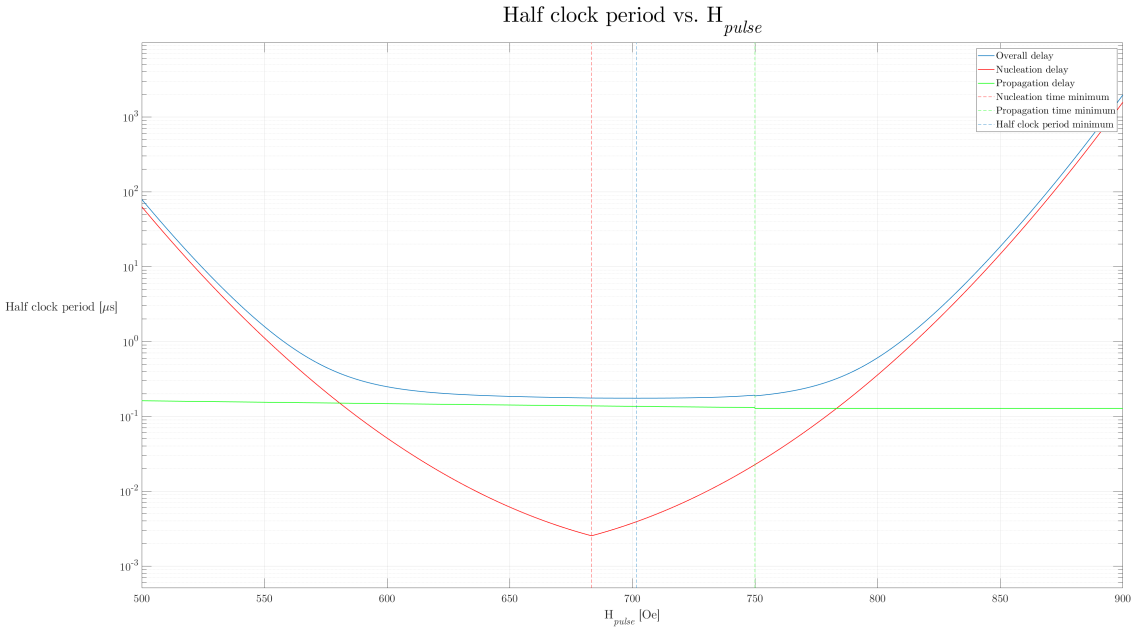


Figure 7.13: Delay components for the SAT architecture

Chapter 8

Conclusions and Future Work

A few conclusions can be drawn from this thesis work. The first one involves the design feasibility. The three main kinds of circuits, namely logic, memory and finite state machines have been tested at length; as [10] suggested, all of them were feasible. Here further proof of this assumption has been given. There does not seem to be any reason preventing whatever digital circuit from being implemented in pNML.

Once feasibility is ascertained, performances should be taken into account as well. First of all, from the point of view of the area this is a very good technology: it has proven to be smaller than a 45 nm CMOS technology node. If scaled, though not as extremely as CMOS has been, and if the multi layer feature will be supported by technological progresses, pNML could easily become smaller than most, or even all, CMOS technological nodes. In this regard, the 3D design possibility should not be just considered a method to reduce area, but also to tune it, meaning that with the same fabrication process, the number of layers used determines the final area. In practice, a further design variable is available. There is also a more specific trait when discussing area that makes pNML quite interesting: as shown in chapter 5, CMOS architectures grew larger when memory elements were included. This is a well-known characteristic of the CMOS: memory are much larger than pure logic, and this influences design choices. Contrariwise, the difference is much less relevant in pNML, dodging a problem that has affected digital design for years. Moreover,

since pNML is not larger than CMOS, it can be stated that pNML solves the problem without area penalties.

On the other hand, delay has not been that good. The delay gap that came to light was huge, with CMOS technology being faster by some orders of magnitude. Something has been tried to reduce delay, both on the design side and on the technology side, but the gap remains very large. Furthermore, the tests have shown that it does not seem to depend on the architectural choices. Therefore, how fast pNML could get will be probably determined by future technological achievements. In any case, the findings of this thesis work depicts pNML as belonging to the field of very low power, area constrained applications, not having very tight requirements about delay performances. pNML could really find its way into this sort of applications, and become perhaps a leading technology. It must be stressed that it is just at its first design stages.

Of course, a lot of work has to be done. The number of designed architectures is still limited, more and more data must be piled up in order to know precisely what specific applications are more suited for pNML. Having "two levels of clock", as said in chapter 1, and having notches allows creative and alternative design styles. The nanomagnets retain their state even without power supply, so that mixed solutions, where the same elements implement both logic and memory must be implemented. The notch element is fundamental in this regard, and should be used in as many different ways as possible. Alternative clocking styles, with respect to one used in this thesis (that replicates the standard CMOS clocking) exist, and should all be explored.

The logic synthesis could be studied in more detail. Since the main logic gate is the majority voter, some ways to effectively synthesize the logical functions with the majority voter as building block must be found, and some researches are already addressing this problem [17]. Even though in this work it has always been used with three inputs, the majority voter can have any odd number of inputs, and this makes it more flexible. Moreover, a five-way majority voter can implement a three input AND (or an OR) gate, whereas in this thesis it would have been achieve with two cascaded two-input AND. This would have saved both area and delay. To sum up, there are many known design variables that have not been tested.

Last, a way to measure power consumption should be found. pNML is a low

power technology, and it is expected to be less power demanding than CMOS; but this must be analyzed with care.

Appendix A

Caveat on the Delay Model

In all the circuits shown in this thesis work, a nanomagnet coupled with another one in a different layer (basically, a VIA) is always modeled as having one clock cycle delay. For example, suppose a wire is coupled with another one on the layer just above. The input wire (the bottom one) is holding a 1, the top wire is holding a 0, and the clock is in its 1 phase. In the model used, the domain wall traveling through the bottom wire would reach the end of it, and stop right there. The clock will then switch to 0. When it switches back to 1, the nucleation of the ANC of the top layer occurs, and the domain wall starts traveling through the top wire. Therefore, the signal moving from a layer to the other undergoes a full clock cycle delay.

Physically, this model is questionable. From the most general point of view, a model where the nucleation of the top layer occurs without having the clock switching twice would be much more representative of the actual phenomenon. On the other hand, the first model is much easier to handle, and depending on the clock period, it could be quite realistic.

A detailed discussion on this issue is off-topic in this work. All is needed to know is that if the second model were used, all the delay figures of these circuits would be reduced; hence, the delay performances have been measured with a sort of worst-case method. The other point is about the comparison between two layer and multi layer structures. The model used impacts a great deal the outcomes of that comparison, possibly leading to opposite conclusions; nevertheless, since the comparison showed that the delay is basically the same, in practice the model used

should not seriously affect the delay in one way or the other.

Appendix B

Design Fundamentals of pNML

This chapter tries to provide practical and useful advices to draw the layout of a logical function or a circuit. At first, looking at a pNML circuit and understanding the function is not easy at all; pretty much every logical element inverts the value of the input, so that getting something wrong is annoyingly frequent. At design time, it goes the same way. Next, the layout from the topological point of view is addressed. Some cases occur quite often, so that an analysis of the most common solutions will be helpful and will spare some effort.

B.1 Majority Voter

The main, an almost unique, logical operator in QCA technology is the majority voter, and pNML is no exception. Even though the majority voter is not a basic element outside of the QCA field, its function, and its reduction to the known logical operations (AND, OR) is quite intuitive; hence, it does not represent a problem. In pNML, to some extent, it does. The reason is that if all the domain walls coupled to the nucleation center are coplanar, the coupling is antiferromagnetic. This means that the circuit is, in fact, a minority voter. A minority voter is far less intuitive than a majority voter, and much more puzzling. For some reason, a circuit made

in negative logic is much more misleading than a regular one, and a minority voter could be thought as negative logic. When counting the number of inversion, which is the core of the analysis, a mistake in the count might occur frequently. Moreover, if the inputs come from the outside of the circuit, to get into a magnetic circuit they have to nucleate a nucleation center, "forcing" into the domain wall a value opposite to their own. To tackle the problem from the right perspective, let us start with the AND/OR reduction of a majority voter, in table B.1. Thus, a very easy way to

Equivalent function	Inputs			Output
	In_1	In_2	In_3	
And	0	0	0	0
	0	0	1	0
	0	1	0	0
	0	1	1	1
Or	1	0	0	0
	1	0	1	1
	1	1	0	1
	1	1	1	1

Table B.1: Truth table of a majority voter

get an AND or an OR gate is available: an input fixed to 1 (OR) or to 0 (AND) is enough. Associating 1 with the OR function and 0 with the AND function matches very well the basic intuition.

B.2 Logical analysis

The previous section somehow associated the 0 value with the AND gate, and the 1 values with the OR gate. Hence, seeing a majority voter with a fixed-zero could make someone think it is an AND gate, but sometimes this is not the case. Have a look at the circuits in figure B.1; The circuit B.1a looks like an AND (because of the 0), but it is actually a NAND. B.1b might seem an OR (because of the fixed-one), and yet it is a AND. The one in B.1c is still a NAND, identical to the one the left. These three examples are aimed at showing that in pNML, the kind of coupling, whether there are the nucleations centers for the inputs (B.1b and B.1c) or not (B.1a) and

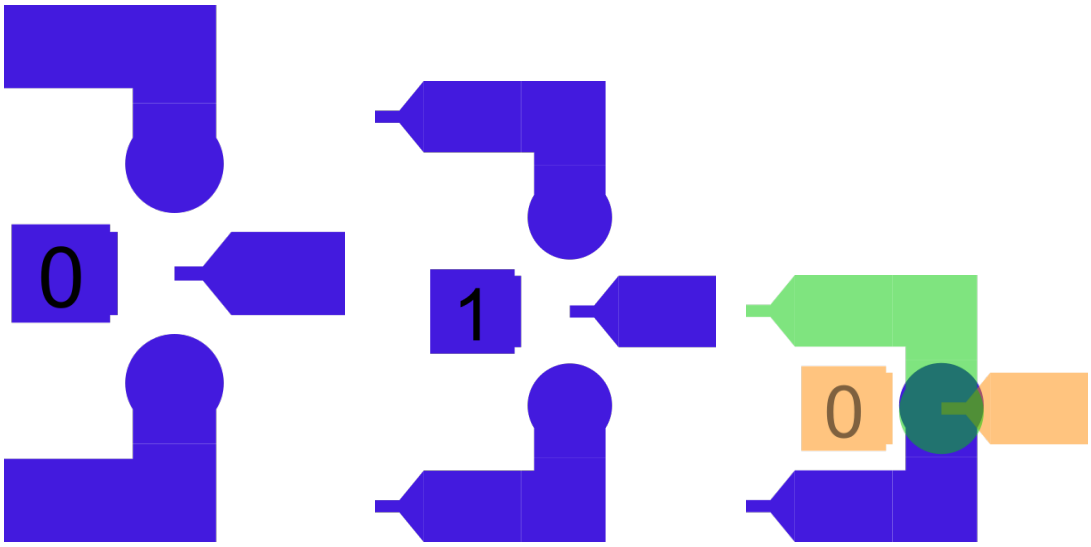


Figure B.1: Examples of circuits that might look like AND gates but actually are not

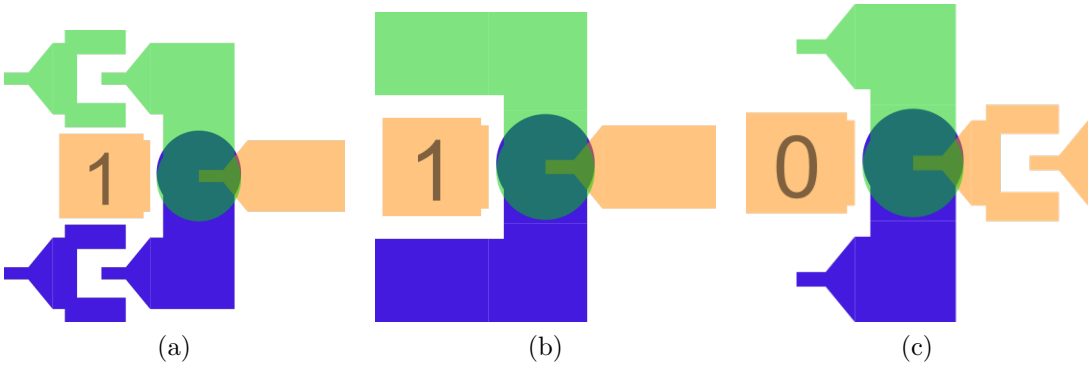


Figure B.2: Example of circuits that are AND gates

the sign of the fixed magnet must be considered with great care, and are almost always misleading. The reason why the circuits are misleading is that the majority voter is "disguised" within the frequent inversions occurring inside a pNML circuit. The correct way to make an AND gate is shown in figure B.2. Three possibilities are shown, with different features: some have a zero fixed-magnet (that forces a one, and this might seem incredible), some have either inputs or outputs inverted. In order to correctly identify the logical function of a circuit, we must somehow "bring to light" the "regular" majority voter function underneath the circuit. A good way

to do so consists in checking the relationship between an input and the value it forces into the nucleation center, rather than just looking at what the whole circuit looks like. For example, consider the circuit in figure B.1b. The inputs nucleate the domain wall, which will hold the negated value of the input. Then the signals run along the domain wall and nucleate again the nucleation center of the gate, getting inverted again. Therefore, we can say that the inputs force their non negated value. The circuit in figure B.1c has the inputs that first nucleate their domain wall, so that it gets their value inverted, but then each domain wall forces that value in the nucleation center of the gate. Therefore, we can say that the inputs force their inverted value in the gate nucleation center. In the analyzing method proposed, every time a circuit is to be analyzed, a table like table B.2, with all the input combinations, is written. The circuit considered is the on in figure B.1c. The table

Inputs	
A	B
0	0
0	1
1	0
1	1

Table B.2: Possible input configuration for a two-input gate

is filled in with the values that each input forces into the gate nucleation center, as in table B.3. Every input may force either the direct or the inverted value into the nucleation center. In this case, for each input, the *Forced value* column is the

Inputs		Forced value	
A	B	A	B
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	0

Table B.3: Values forced by each input

opposite of the corresponding *Inputs* column, since it has been shown that the inputs force their inverted value. Now, in that column everything is accounted: the number

of signal inversion along the path, the kind of coupling and so on. Of course, the fixed magnet is not different from the two inputs, so let us add it in the table, as in table B.4. The fixed magnet value is zero, and is antiferromagnetically coupled.

Inputs		Forced value		
A	B	A	B	Fixed input
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

Table B.4: Values forced by each input and by the fixed magnet

Therefore, it forces a one. At this point, to get the final output of the whole gate, the three rightmost columns can be considered the inputs of a regular majority voter. Or, alternatively, A and B can be AND-ed. The reason why they must be AND-ed is that the fixed magnet forces a zero. Had it forced a one, A and B had to be OR-ed. Everyone will chose the way that suits him/her best. The table that one eventually gets is table B.5. The table shows clearly that the circuit in figure B.1c is a NAND.

Inputs		Forced value			
A	B	A	B	Fixed input	Output
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	1	0

Table B.5: Output value eventually calculated

Needless to say, any other analysis (provided that it is correct and consistent with respect to the definitions) will work, but this one is quite quick to carry out, and seems more effective than just trying to remember the particular arrangement of every kind of gate. It could be considered a good entry-level method, then after a little of practice one finds his/her own methods and insight to the problem. This table method is also easy to update in case some changes are made. For example,

suppose the input A is inverted (or, as it often happens, the table is made, but later on a mistake is spotted and the table has to be corrected). To update the table, just swap the rows with $A = 0$ with the ones with $A = 1$, leaving B unchanged, just for the "forced value" column, and for the "output" column. With reference to table B.5, it means swapping row 1 with row 3, and row 2 with row 4. Table B.6 shows the result outcome of this permutation. s a lNotice that now the column

Inputs		Forced value			
A	B	A	B	Fixed input	Output
0	0	0	1	1	1
0	1	0	0	1	0
1	0	1	1	1	1
1	1	1	0	1	1

Table B.6: Example of row switch for an input

Inputs (A) and the column *Forced value* (A) are identical, whereas previously they were one the inverted of the other. After all, the analysis boils down to just checking whether the input signal forces its direct or its inverted value. And the main point in this analysis is that: one must always consider the forced value in the nucleation center, and focus on that. Never try thinking about anything else, such as for example the multi layer version of the gate (like the on in B.2c) associated with the "direct" logical function and the single layer with the inverted logical function, or the any other possible association layout-function. Then, after having had to do with a certain amount of circuits, some configurations are likely to be identified at once, naturally, without any explicit classification. This method of row switching works even if both inputs are inverted (as usual, either because an inverter is actually placed along the domain wall, or because a mistake occurred in the analysis). All the rows must be swapped: row 1 with row 4, row 2 with row 3. It is probably easier to understand if considered as the inversion of one of the inputs first, and then of the other one. It can be seen in table B.7, where the gate described in table B.5 has both inputs inverted. Table B.7 is both the inversion of both inputs with respect to table B.5, but also the inversion of input B with respect to table B.6, proving that the row swapping can be performed either for both inputs at the same time or one

at once. If the fixed magnet is changed (or it forced value was considered the wrong

Inputs		Forced value			
A	B	A	B	Fixed input	Output
0	0	0	0	1	0
0	1	0	1	1	1
1	0	1	0	1	1
1	1	1	1	1	1

Table B.7: Example of row switch for two inputs

way), the column of its forced value must be inverted, but this time the output must be recalculated, whereas when the inputs are changed, the row switching operation is enough in order to update also the output. For example, consider table B.8, which is the same table as B.5 but with the fixed magnet switched. The output column

Inputs		Forced value			
A	B	A	B	Fixed input	Output
0	0	1	1	0	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	1	0	0	0

Table B.8: Example of fixed-magnet switched

cannot be obtained from the one in the original table by means of row permutation, neither it can by means of inversion. It is something different, and this is why it is much better to carry out again the majority voting function among the forced values. Anyway, the fixed input is not a "special" input at all: the simple reason why its switching does not cause a row swap in the output column is that not all the input combinations are written in the table. The missing combinations are exactly the one where the fixed input has the opposite value. If one considers the whole set of input combinations, every input inversion cause an output row permutations. Simply stated, in the truth table of a majority voter, the output corresponding to the inversion of an input can always be obtained by means a row permutation. But

this is just a theoretical consideration, always writing the full truth table of a gate with a fixed input is not worth it.

B.3 Singular fixed-magnet

From the logical point of view, the third input of a majority voter can be kept stable either to one or to zero, and the choice is made basing on mere functional reasons. Nevertheless, when it comes to circuit manufacture, it turns out that technology is not always able to provide both kinds of fixed magnets. Therefore, it could be advisable to design the circuit with just one of the two fixed magnets. It might sound impossible, but it can be done, and with far less effort than expected. Before showing the possible ways to do so, let us say something about *which* of the two values must be kept. In this work, the choice fell on the zero, just because some circuits had already been designed, and it the most complicated one had only fixed-zeros. Thus, anyone can choose any one of the two values: keep in mind that two circuits with different signs of the fixed magnets must have separated clocking control circuits, which makes manufacturing them together very cumbersome. Therefore, a designer should choose one of the two values and always stick to that choice. In an hypothetical world with a widespread use of pNML technology, there would probably be an universal standard in one way or the other.

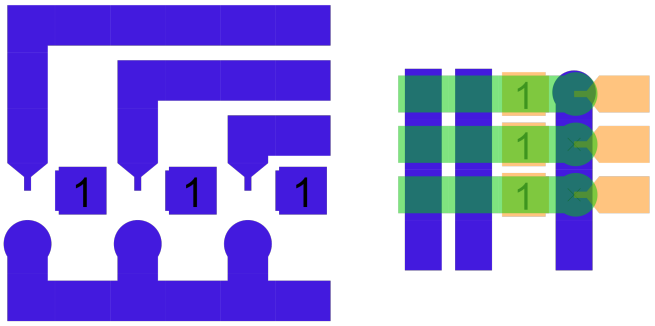
B.3.1 Inverter insertion

Suppose to have only the fixed-zero magnet available. The simplest solution consists in using that magnet when needed, and the modified magnet shown in figure B.3 when a fixed-one magnet is needed. This solution is easy and does not require any

Figure B.3: Alternative way to produce a fixed-one magnet, sometimes referred to as “alternative fixed-one”

elaboration. A circuit designed with magnets of both types can have its fixed-one

magnets changed with this one, and will work. This magnet is however three times bigger than the traditional one. In practice, really often the room to allocate a triple-sized element is not available. Scenarios like the ones in figure B.4 are not rare. In case B.4a, there is not room enough to place this "alternative fixed-one".



(a) Not enough room (b) Fixed-magnet would get coupled

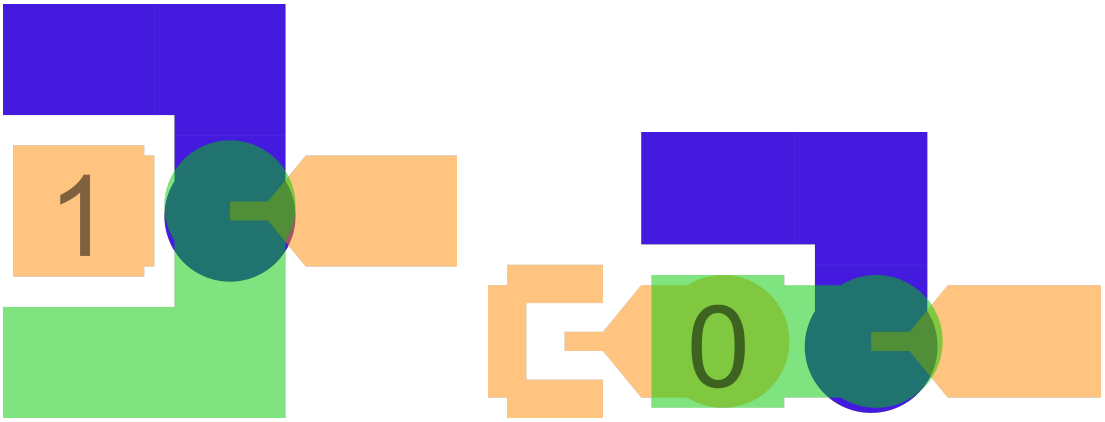
Figure B.4: Example of circuits where the "alternative fixed-magnet" cannot be used

In case B.4b, there would be room enough, but the domain walls placed all around would influence the inverter of the fixed magnet. In some cases, both of these adverse conditions might occur. One of the main aims of this work was packing some pNML logic into the tightest possible space, and this makes these situations quite common. In conclusion, this could be an interesting, simple solution, but other alternatives are needed.

B.3.2 Layer switch

The second idea leverages the multi layer feature of pNML. A nucleation center is antiferromagnetically coupled with a coplanar domain wall; when the domain wall come from above or below, the coupling is ferromagnetical. This entails that the same input forces two opposite values depending on whether it is placed on the same plane or in another one with respect to the nucleation center. Thus, when the logical operation needs a one-forcing fixed magnet, a fixed-zero magnet can be placed in the same plane as the nucleation center, whereas when a zero-forcing fixed magnet is needed the same fixed-magnet can be placed on the layer above or below the nucleation center. Figure B.5 shows two circuits whose logical function is the

same. The solution seems very effective, but has some disadvantages. The first one



(a) Original circuit (b) Adjusted version

Figure B.5: Rearrangement of a gate when just one fixed-magnet is available

is that sometimes layers are limited. Almost all this thesis work takes for granted the availability of an arbitrary number of layers, but technology does not offer more than two or three layers. When only two layers are available, this method sets a very tight constraint on the design possibilities: if the fixed-magnet has to be placed on the layer above the nucleation center, all the other signals, possibly also coming from the layer above the nucleation center, have to reach to the layer below. If the fixed-magnet has to be placed on the same layer as the nucleation center, it occupies one of the three (sometimes five) slots available around the nucleation center. This might not seem a big deal, but sometimes (because of domain walls tightly packed, angular positions and so on) the geometry prevents some of the slots around the nucleation center from being accessible, so that even just one more occupied might makes thing very difficult. The second issue concerns the other signals reaching the nucleation center. Consider figure B.5. The signal on the top layer goes to the middle one (from the green layer to the orange one). Its coupling is changed, and the signal must be inverted in order for it to force the same value into the nucleation center. Therefore, an inverter is needed, as B.5b shows. However, most of the multi layer gates shown here just as examples are made with the layers not overlapping with each other: this is just a workaround to make the image easier to understand. In fact, a real circuit will be more like the one in figure B.6. As it can be seen, the domain walls on the different layers tend to overlap, so that the gates take up less

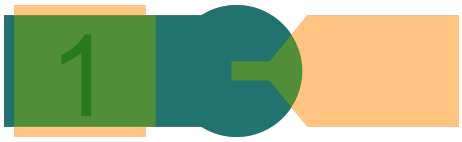
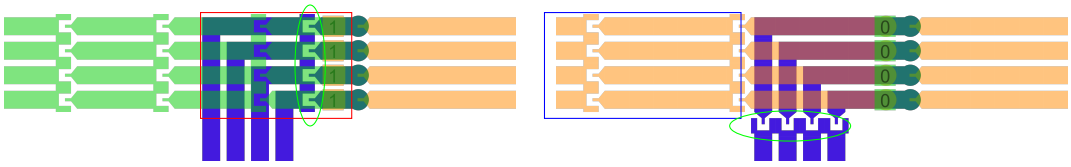


Figure B.6: Standard way to build a 3D gate

volume. In these conditions, there is room for the inverter needed to rearrange the input. The circuit would be the one in figure B.7 That rearrangement would work

Figure B.7: Technologically incorrect gate

in MagCAD, but is not technologically correct: the inverter is placed over a domain wall, and will be influenced by it. The gate could happen to have domain walls all around (even though here are not drawn), making difficult for it to be arranged like in figure B.5a. In some cases, the input comes through a long domain wall, along which is periodically inserted an inverter (there must even number of them in overall) to reduce the critical path. In this case, this solution works very well: if the inverters are used to break periodically the domain wall, it means that somewhere the room for them has been found. All one has to do is eliminating or adding an inverter and everything will be fine. The strong point of this solution, when compared with the previous one, is that some times and inverter can be removed, if there already is one, and that the inverter insertion/removal can be done far away from the nucleation center, if the domain wall holding the input value is long enough. On the other hand, the other solution, in subsection B.3.1 needs that the room for the new element is found very close to the nucleation center. An example of this is given in figure B.8 On the B.8a, the red box outlines the "off-limits" area: no inverter can be placed here. The green ellipse contains a row of inverters that are placed in the *bottom* layer (the color of the shapes drawn over the figure are not related with the colors of the layers). All the others are on the top layer. On side B.8b, all the inverters are removed from the "forbidden" area. The ones that were into the the green ellipse are still into a green ellipse, but on a different spot.



(a) (b)

Figure B.8: Example of layer switch with the removal of an inverter

The blue box outlines the area available to place the inverters needed. Actually in this case one inverter was removed (one for every gate). Notice that the maximum length between two inverters (that defines the critical path) has increased. This might happen, but it is seldom critical.

B.3.3 Boolean logic rearrangement

The most used solution to design a circuit when only a kind of fixed-magnet is available is neither the one with the inverter (which is rather poor) nor the one that switches places between the fixed magnet and one of the signals (a good one, but with some limitations). The analysis in section B.2 proves that a logic gate such as the AND can be made with two inputs forcing their value and a fixed-magnet forcing a zero. This holds true no matter what is the actual coupling, the relative position of layers and so on, because, as demonstrated above, the forced value accounts for everything. In a technology where signal inversion does not occur as often as it does here (other QCA technologies, or even a CMOS majority voter) this would be the most natural way to produce an AND. Therefore, this might lead to think that in a majority voter, forcing a zero is necessary to produce an AND, but this is not true. Let us use the method seen in section B.2 to show it, and let us start with table B.9. The table contains only know information: the fixed input forces a one, and the output must be the one of an AND (if a solution does exist). Therefore, inputs A and B have to force a value so that the majority voting operation yields that output. Two things must be said beforehand:

- The operation must be carried out coherently. In other words, the unknown columns can be either equal to the corresponding input column or equal to the negation of it: they cannot be filled with arbitrary values. It makes four

Inputs		Forced value			
				Fixed	Needed output
A	B	A	B	input	
0	0	?	?	1	0
0	1	?	?	1	0
1	0	?	?	1	0
1	1	?	?	1	1

Table B.9: Study to find a way to make an AND gate with a fixed-magnet that forces a logical one

total possible patterns. This is actually a little obvious;

- The AND is a "symmetrical" operation. This means that the inputs cannot be distinguished: they can be swapped and nothing changes. If one has a black box that performs the AND operations and must plug two inputs in it, it does not matter which input goes in which socket. All this is to say that A and B must force a value in the same way: either both force their value or the negated one. This discards two of the four previous possible patterns.

The second point simplifies considerably the matter: after two tries all the possibilities are explored. Let us do it, in table B.10. It seems there is no way to get an AND

Inputs		Forced value			
				Fixed	Output
A	B	A	B	input	
Forcing non negated value					
0	0	0	0	1	0
0	1	0	1	1	1
1	0	1	0	1	1
1	1	1	1	1	1
Forcing negated value					
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	1	0

Table B.10: The two possible cases for table B.9

from a majority voter with a forced one, and so the first intuition was correct, but it does not take long to realize that the second pattern is the one needed provided that it is negated. To recap, an AND can be made at least in two ways:

- Forcing a zero and the non negated value of the two inputs;
- Forcing a one and the negated value of the two inputs, and then negating the output.

This method requires nothing but possible inverter insertions. It must be considered an alternative to the method of layer switching. With thee two methods the designer can choose with much more freedom where to place the fixed magnet, regardless of which logic function is to be implemented (this operation can be done for any logical function, even if here only the AND case is illustrated in detail). The drawbacks of the method are more or less the same as the ones of the layer switching method: room for an inverter insertion must be found. Actually, the layer switching method affected just one of the inputs, whereas this one affects the two inputs and the output, meaning that three inverters must be added. In any case, with two methods to choose between, the difficulty in inserting inverters could be one of the factors that determine the choice. No images are shown of this method, because the considerations are pretty much the same as the ones made for the layer switching method. To finish off this section, some rules that someone might find useful to mentally process the boolean functions and the gates are put in evidence. Two expressions are used, and they have to be defined:

- "Dual gate" means the other kind of gate, so that the dual gate of an AND is an OR, the dual gate of a NOR is a NAND;
- "Opposite gate" just means that the output of the gate is negated. Hence, the opposite gate of a NAND is an AND.

Now, the rules are:

- If in a gate the fixed-magnet is switched (meaning that, in whatever way, it ends up forcing the opposite value), the gate changes in its dual gate;
- If in a gate the inputs are negated, the gate changes in its dual and opposite gate (e.g. an OR becomes a NAND);

- If in a gate the output is negated, the gate changes in its opposite gate. This is trivial;
- As a consequence, if in a gate the fixed-magned is switched, and both inputs and the output is negated, the gate turns into its dual gate. This is the case discussed above;

It is fair to say that all this playing with the boolean logic comes right from the two De Morgan laws, as it is almost always the case when handling boolean logic.

B.4 General rules and tricks for layout drawing

Hand-making layout is a particular job. It is considered usually suboptimal, an activity to resort to only when either computer CADs fail or the gate is so critical that it has to be analyzed with the utmost care. In any case, in CMOS digital design it is usually limited to a very little part of the circuit (the cell of a library, a critical point of a CAD designed circuit). In pNML, the chances of automatically designing a circuit are very few. Therefore, everything is hand-crafted, keeping in mind that the optimization level of the circuit is inversely proportional to its complexity. Anyway, after a little of practice, recurrent cases are identified, and reasonable solutions for them are found. This section offers an illustration of the cases found during this thesis work. They should be considered as a starting point or a series of advices, that anyone, depending on his/her skills and knowledge, can either follow or ignore in complete freedom.

B.4.1 Choosing layer to reach a nucleation center from

When many layers are available, the designer can choose if the input and the nucleation center must be coplanar, or if the input comes from a layer above/below to the one where the nucleation center lies, and couples with it from there. Some considerations can be made. The first point is probably the most general: for a domain wall reaching the nucleation center from above/below the direction does not matter, whereas a domain wall one the same plane as the nucleation center is directional. Thus, if the domain wall comes from above, all the solutions in figure B.9 work,

whereas if the domain wall comes from the same layer, it must be turned in order to approach the nucleation center from the correct direction, as in figure B.10, and this sometimes (when the input comes from the same direction as the output goes) takes up too much room. Any time room occupation is discussed, it is to be supposed

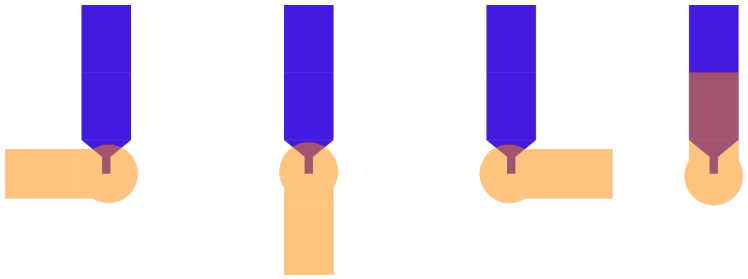


Figure B.9: Example of VIAs, where the direction of the input does not matter

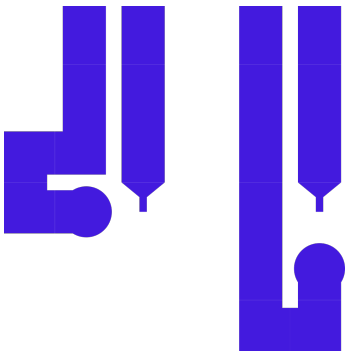


Figure B.10: Example of coplanar coupling, where the coupling takes place only in certain directions

that other domain walls run all around the point considered, and sometimes must reach that same nucleation center. The images are often kept minimal in order not to be too confusing. Also, almost all the examples here are made in two layers (the bottom one is blue, the top one is orange, and sometimes there could be a third one on top of it, green-colored) because the two-layer constraint causes a very thorough exploration of the design possibilities. Actually, choosing on which layer the wires should be placed is a problem that might be elaborated much further. For example, consider figure B.11. It shows two signals that are used as input of a gate. Suppose that the output has to be on the bottom. At least two solutions exist, those shown in figure B.12. The solution B.12a increases the width (consider the domain walls running along the height direction, or the images turned clockwise by 90 degrees),

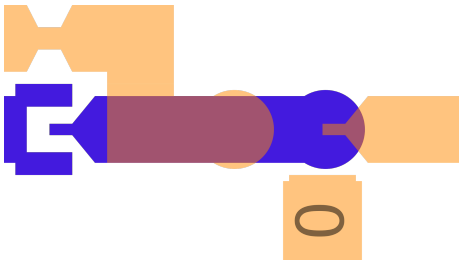
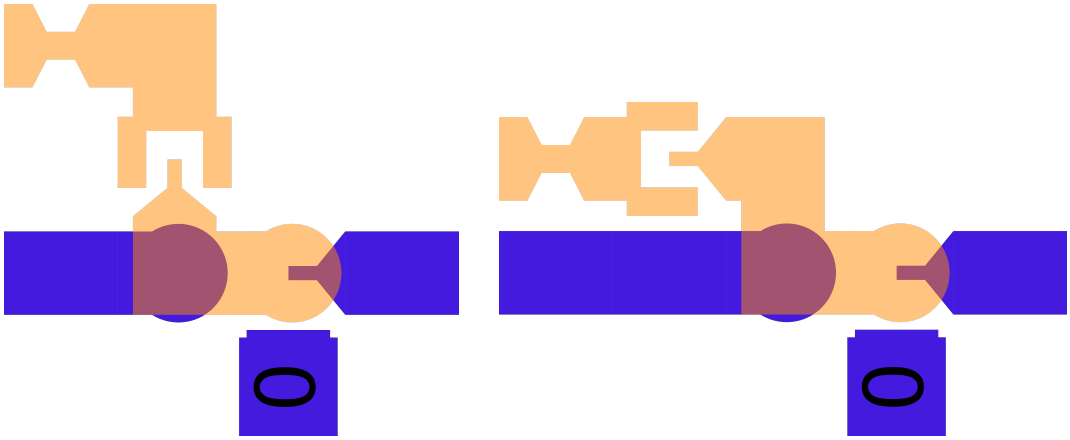


Figure B.11: Case study: the output must be moved to the bottom layer



(a) (b)

Figure B.12: Possible solutions to the problem of figure B.11

the B.12b increases the height. The one to be preferred depends on the particular case, that is, if it is less expensive increasing the width or the height. For example, if this piece of circuitry is replied many times along the height, but just two along the width, it is probably less expensive the B.12a solution (namely, the circuit area is smaller). Let us see another variation: suppose the signal from the notch must reach other points of the circuit too. Figure B.11 will become something like figure B.13. Solutions of figure B.12 turn into the ones of figure B.14. In this case there is something more to take into account. The solution on the B.14a is larger, but does not change in any way the path from the point of view of the red square. The solution B.14b is narrower, but adds two inverters to the path going towards the red square. And these new problems must be tackled along with the considerations of the previous example, so that now both area and timing are affected: each variable will carry a certain weight, and eventually a choice must be made. For example, if, from the previous analysis, it has been ascertained that increasing the width of the

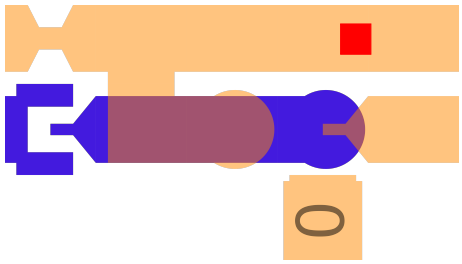
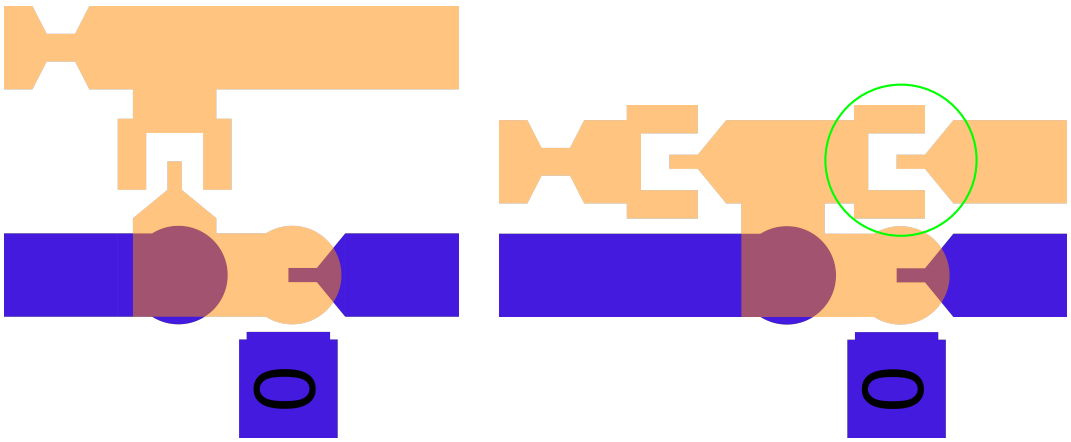


Figure B.13: Variation of the case study in figure B.11: the output must be on the bottom layer, and the signal from the notch must reach farther away



(a) Increased width (b) Increased delay

Figure B.14: Possible solutions to the problem of figure B.13

circuit does not increase too much the overall area, B.14asolution is the best one. If the two inverters are not determinant (possibly because the critical path is elsewhere), the B.14b solution in fact does not impact timing. Hence, different metrics and options are on the table. Two particular points are worth discussing, as they might not be evident from the images. Assume that the domain wall where the red square is placed travels very far. It would be very likely to be broken periodically with inverters, as was said above. In this case the new inverter (the one in the green circle) could be removed along with another one, and the path rather than being an half cycle slower, gets an half cycle faster. The second point concerns what is above/below the orange layer. The solution B.14b does not allow another domain wall to be placed just above or below it, because there is an inverter (unless it can be eliminated), that would be influenced. In case passing this hypothetical domain wall is absolutely necessary, the solution will be discarded. And this thought prompts

another variation to this problem. The starting point is again figure B.11, but this time there is a further constraint: there cannot be any inverter along the domain wall in the top layer, probably because there is another domain wall (not shown) below. Actually this is unlikely, because the notch too would be influenced by the domain wall, but this is just an example. If no inverter can be placed, the signal on the notch will have to force into the nucleation center its value, and not the negated one as it would be required from figure B.11. Therefore, the designer must resort to the same workaround used to make an AND gate even if a fixed-one is forced into the nucleation center. Nevertheless, if the constraints that just allows one of the two fixed magnets still holds, it means that either the fixed-zero is placed on third layer, or that the solution seen in subsection B.3.1 must be used. And if just two layers are available, this last solution is the only one feasible. That solution has an inverter too, which might undergo the same coupling problems: in that case, the wire can be made as long taking up more room, as shown in figure B.15. The wire can be as long as needed, since, due to the steady nature of the input, it does not cause any timing concerns. This case might be very rare happen in real design, but it shows that, even if very strong constraints are set, there still are feasible choices.

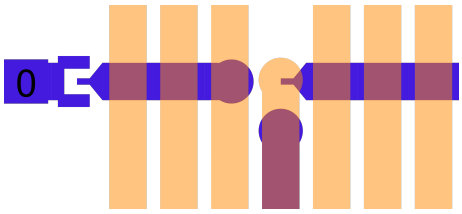


Figure B.15: Possible alternative to place an "alternative fixed-one"

B.4.2 Sensible layer organization

When multiple layers are available, they must be used in the most structured possible way, otherwise the design of a moderately complex circuit can be impossible. And, even if one manages to complete the design in an unstructured way, later modifications might be prohibitive. Before drawing wires and gates, it is necessary to conceive some sort of frame for the components of the circuit, so that there is already a rough idea of where every gate should be placed. The frame changes depending on the kind of circuits to be designed. For decoders and memories, which

are basically an array of the same basic elements, possibly with some changes, it should be defined approximately how the array will be structured. For simple gates, such as a full adder or a multiplexer, the idea is trying to keep the design as small as possible, but the particular arrangement depends on the logical function. Just to give an example, a full adder is made of three majority voters, so that a regular arrangement of them can be found. A multiplexer has two AND sharing a signal (one of the ANDs has the signal negated) which are then OR-ed. Here too exist a regular structure, as section 3.3 has endeavored to prove. So far, every time a circuit has shown a certain regularity, it has been pointed out. Here some further examples will be given.

FSM states

The first example has already been mentioned throughout the description of some structures, but here it will be seen again, devoid of unnecessary details. It is about a possible way an FSM with many states exploits the multi layer design feature. It can be seen in figure B.16. Only three states are there, because the drawing would

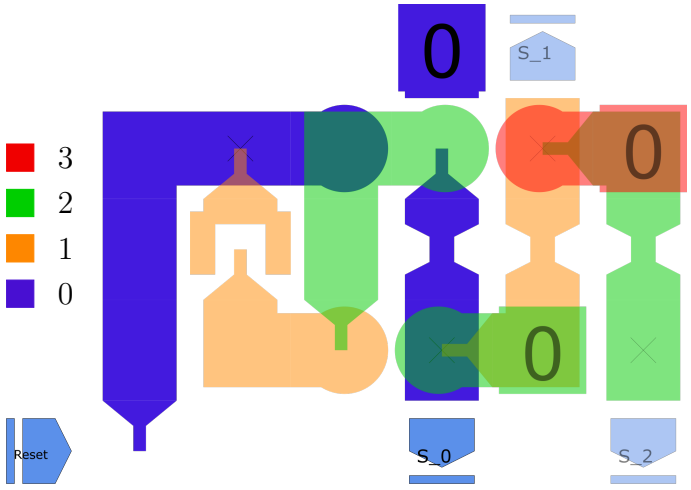


Figure B.16: Possible arrangement on notches in a FSM

be too complicated and unreadable. The wire going from the last state notch back to the first one (in blue) is not drawn either. Every notch is placed on a different layer, in a sort of staircase-like structure. The structure includes the *Reset* network. This is a useful method, that, as already said, works very well with FSMs that run

through the states without conditions to be evaluated. If that is not the case this arrangement can still be useful, but less effective and regular. Even if here only three states are shown, the structure has been used to allocate up to 15 states, with no particular issue. Nevertheless, when the number of states is quite high, a variation of this structure comes in handy, since it is pointless having 15 layers on a circuit just because of a single finite state machine. The variation consists in moving down with the layers after a certain point, or to use again the staircase similarity, to make the staircase climbing up and then down. The image is the B.17. Again, few layers

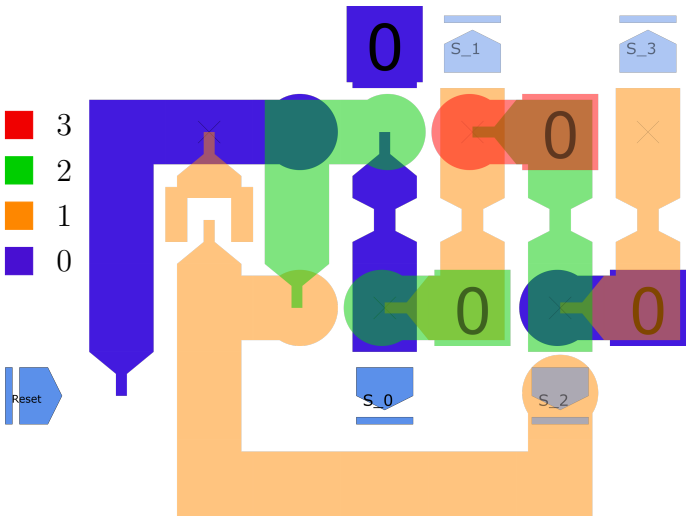


Figure B.17: Variation to the structure in figure B.16

are drawn to make the image simpler. The wire that brings the signal *Reset* to state 3 is made wider than needed, just to show how the signal can be brought to the "second flight" of stairs.

Structured paths

When layers are just two, the designer may tend to use just one of them, saving the other one as a sort of "bypass layer", that is, using it just when wires in the main layer cross with each other (by the way, it is interesting that, despite all the interesting features pNML technology has, even when compared to other QCA technologies, it does not allow coplanar wire crossing). Of course, this is not a good practice: the circuit in figure B.18 is a good example of regular use of two layers. It

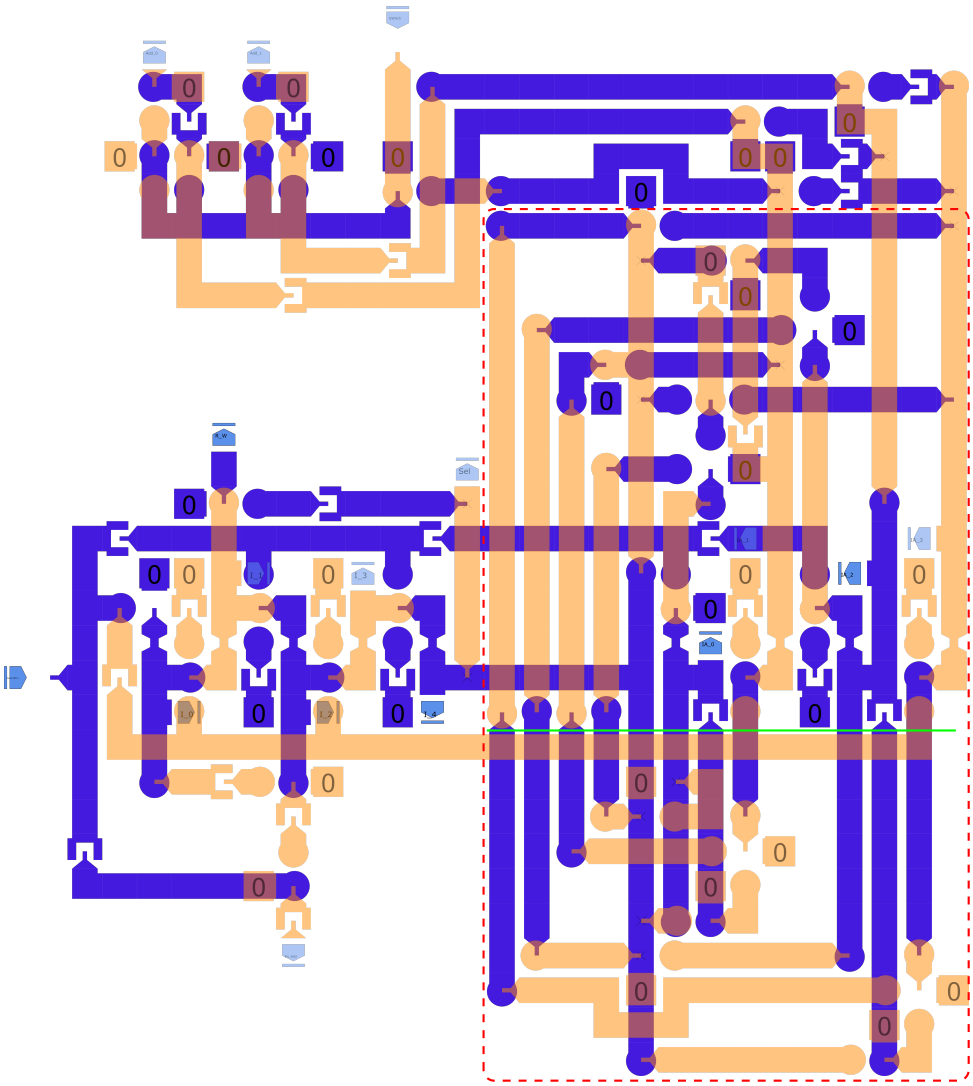


Figure B.18: Example of a regular use of the two available layers

shows a circuit with very interesting details. It has already been seen in Figure 6.18, and has been described as an FSM made with two "subFSMs", a master one which automatically moves to the next state each time the reset signal is low, and a slave one, which steps into the next state as soon as the master reaches its last state. There is something more, but not interesting here. The first thing that should draw the attention is the notches' arrangement. It can be considered a particular case of the staircase method, with just two steps. Nonetheless, given that neither the notches nor the inverters can have domain walls below or above, the notch horizontal

spacing is greater. The most interesting feature of this FSM consists in its highly structured arrangement of the wires within the layers depending on direction and position. Let us be clear. Focus just on the section of the circuit inside the red dashed rectangle. As already said, this part contains the logic necessary to move through the steps of the slave FSM. A green horizontal line is drawn in the middle of the circuit. Notice that below that line, almost all the vertical wires are on the bottom layer, and almost all the horizontal wires are on the top layer. On the other hand, above that line the bottom layer houses almost all the horizontal domain walls, whereas the top layers includes most of the vertical domain walls. This is a very good example of a structure that rationally finds a way for each needed wire, avoiding congestion, twisting and wire crossing. It has a drawback: the room it occupies is pretty large. But finding a more compact (but less regular) way to route all those signals would have taken much more time, with no guarantee of success. As explained in the SAT chapter, this is one of the two FSMs that evaluates conditions to decide whether moving to next state. This increases the amount of wires, and that is why a regular structure has been scrupulously followed.

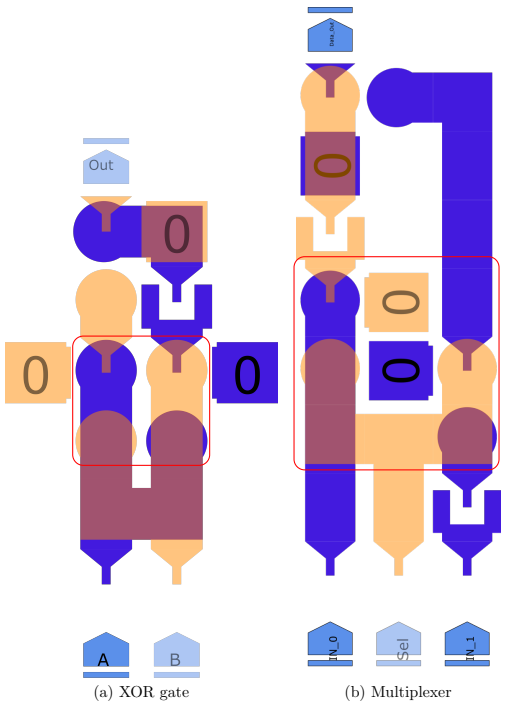
Leveraging Coupling Mode

Another criterion that is almost necessary in two layer designed, but it is also a very good rule in any case, has to do with the kind of coupling between the domain wall and the nucleation center, which can be, as amply shown through all this work, either ferromagnetic or antiferromagnetic. Since this means, in digital logic terms, forcing respectively the value of the domain wall or the negated value, it would be very nice to take advantage of it. Let us see two circuits, illustrated in figure B.19. They are a XOR gate (B.19a) and a multiplexer(B.19b). A certain portion of the circuit is outlined in a red box in both circuits, but first let us get to their logical expression:

$$\bar{A}B + A\bar{B} \qquad \text{XOR gate} \tag{B.1}$$

$$In_0\bar{Sel} + In_1Sel \qquad \text{Multiplexer} \tag{B.2}$$

In the XOR gate is made with two AND whose outcome is OR-ed, both AND get the two inputs, but for one of them the input is negated; in the multiplexer, the signal



(a) XOR gate (b) Multiplexer

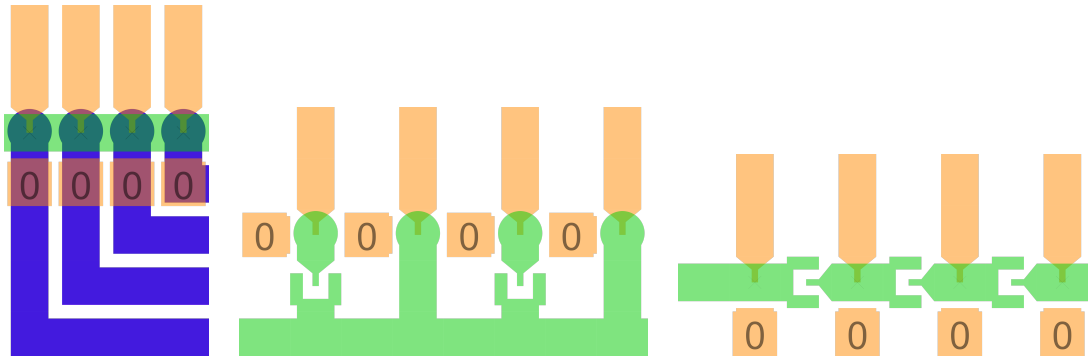
Figure B.19: Example of circuits with two ANC on the same pad coupled in two opposite ways

Sel reaches two different inputs being negated in one of them. Every time a signal in used once negated and once non negated, a single domain wall holding that value can be coupled with two distinct nucleation center, once ferromagnetically, and once antiferromagnetically. This is what is done in figure B.19, in the area within the rectangles. This works very well when a signal is needed both in its negated value and in its non negated one, and usually makes the layout very dense and compact.

Passing-through signal

A common design situation consists in delivering a signal that is used in several gates. There are a lot of examples: the reset signal, the inputs of a decoder, the inputs of a PLA, and so on. These examples suggest that this situation is common in circuits that are arrays of basic elements, as the decoder or the PLA are. Sometimes the signal is inverted periodically along its path, either for functional reasons (a decoder) or just to split the domain wall and decrease the critical path. Here the main solutions to this common problem are shown. A first observation could be: this

case is one where passing the signal in further layer is absolutely worth it. When the signal does not have to be inverted, and it does not cross a large area, a straight wire solves the problem. An example of this is found in all the decoders of section 3.1. To put in evidence the important domain walls, they have been drawn schematically in figure B.20, where the unneeded circuitry has not been drawn. When many



(a) (b) (c)

Figure B.20: Possible solution to deliver a signal to many ANC

layers are available, this is practically the only way this need is answered. Solution B.20a is the most compact one, but it is also rare, because usually the signal must be inverted periodically, as in B.20b or B.20c. These two have already been discussed in section 3.1: one of them is larger but with regular delays, the other one is extremely small but slower, meaning that the signal is inverted (that is, delayed) along the path. The way the *Reset* signal is brought to each notch in figure B.16 can be considered a solution too, but is probably suited just to that structure. When the design must be confined within two layers, these solutions have to be adjusted. In some cases, they are applied basically in the same way, is in figure B.18. The dashed paths deliver the *Reset* signal, and method B.20b is used, with the domain wall periodically broken with a pair of inverters. In other cases, a different approach can be tried. A good study case is the one of figure B.21, which is a detail of figure 6.14. The solution is marked with the red path. Let us examine the conditions. The input cannot run through the top layer, because it is already used for other signals. A solution that travels along just one layer might be conceived, replicating solution B.20b but is not viable here, because there are other domain walls blocking the path, the ones in the black rectangle. Therefore, the piece of domain wall on the top layer (inside the green dashed rectangle) that links the two red paths on the bottom layer is

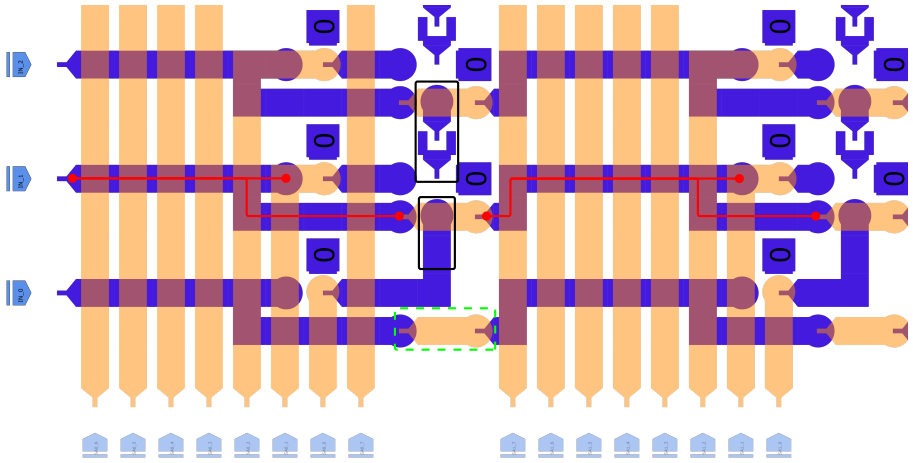


Figure B.21: Possible solution to deliver a signal to many ANC with just two layers available

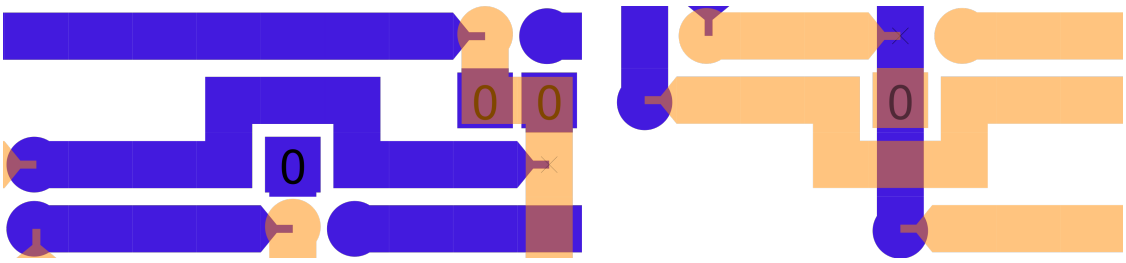
necessary. It must not be neglected that the "branch" of the domain wall must reach to a spot where there are no domain walls on the top, so that the top layer can be "assigned" to the continuation of that path. From a theoretical standpoint, this way to deliver a signal to multiple gates is the same as the one in figure B.20b (imagine the inverters are taken off), except for the fact that here it runs on two different layers. From a practical point of view, the solutions do not look like the same, so it seemed better analyzing them separately.

The oblique pad

The oblique pad is an element first introduced in section 6.2, but then never used again. This does not mean that it is not a useful item: on the contrary, it is extremely useful. It has been used rarely because it was not available when the SAT circuits were designed, and from that point on most of the work consisted in comparing new solutions with those circuits. Hence, for consistency reasons it has not been used. If all the circuits in this thesis work that do not contain that particular item were to be redesigned using it, they would come out smaller and more regular. A general rule should be this: always try using it. If it cannot be used, then the designer will resort to bend the domain all in the direction needed.

General advices

This chapter is concluded with small hints and tricks, not very much articulated but anyway worth discussing. The first one concerns the design dataflow with big, modular and complicated circuits such as the one in figure 4.28. The dataflow should mimic the typical CMOS one: first components are placed one the workspace, and this should be done with concern towards the relative position of them, so that the connection are as short as possible, and the area as small as possible. Then connections are made: sometimes no logic goes in between the connections, some other many signals reach the same input, and they must be OR-ed, or multiplexed, or elaborated in a suitable way. At this point the circuit should be fully working, but a further action must be taken: the shrinking. It is extremely likely that somewhere some space can be saved, and once the wires are routed it is much easier to notice it. This way, the "routing" phase can be done with no concern for space saving, focusing only on the topology, putting off the shrinking task for later. In this regard, the two layer version of the SAT Input FSM serves a double purpose: when discussing how to structure regularly the circuit, it provided a very powerful example. Now, it will show how, after the regular design of the circuit, some changes to the latter, in a direction that make it "less regular and structured", can also make it more compact. Figure B.22 contains two details of the circuit. Two domain walls have been bent



(a) (b)

Figure B.22: Example of wires bent to reduce room occupation

so that a fixed magnet could lie where originally they were. This allows to reduce the size of the circuit by one square, taking advantage of the free space beside the domain wall.

Of course, this is a description of a fairly reasonable approach to produce a good project, keeping in mind that a completely hand-made design of a complex circuit very rarely can be perfectly optimized.

Another point involves layer number: it is probably a good rule using one more layer only if it will host a good amount of circuitry, not just one or two domain walls. However, if the structure is based on layers (as the examples about FSMs), and it cannot be adapted easily, the only alternative could be considering using another structure, and this might become mandatory if the number of layers is a critical parameter of the technological process.

Appendix C

3D Rendering of the pNML circuits

Here is explained how to render in 3D the nanomagnetic circuits made with MagCAD. Two more programs are needed: Blender and Inkscape, both of them free.

- Inkscape (https://inkscape.org/it/release/0.92.2/), for svg handling;
- Blender (https://www.blender.org/download/), for 3D images creation

First thing to do, open the project in MagCAD. Place a fake input always in the same spot in all the layers of the circuit, as in figure C.1. It will serve as reference point to align layers later on.

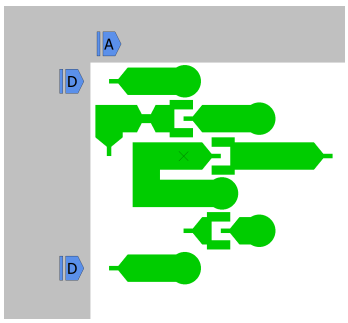


Figure C.1: Fake input as "orientation mark"

Then, export every layer separately. To do so, hide all the layers but the one to

be exported, and select *File* → *Export image*. Choose PNG image format, as in figure C.2.

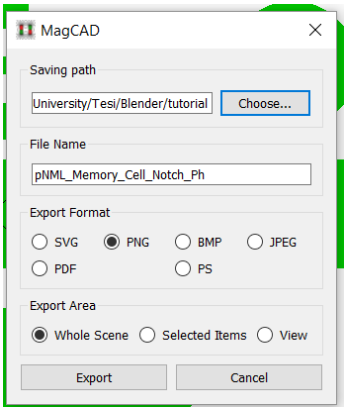


Figure C.2: PNG export

MagCAD can export in SVG format too, but for some reason its SVG causes some issues when imported in Blender. It is better exporting the image in PNG format and the converting it in SVG within Inkscape. Once all the layers are exported, close MagCAD. Open Inkscape, then click *File* → *Open*, or use the shortcut $Ctrl + O$. Select the first layers within your folders, and open it. The window in figure C.3 will pop up. The parameters in figure should do, but none of them are critical, so feel free to test different combinations. If something catastrophic occurs, roll back to these ones.

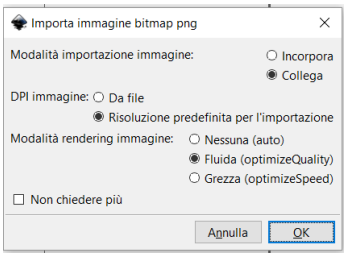


Figure C.3: Inkscape PNG import

At this point, Inkscape contains the image, but it is not yet an SVG file. To convert it, select the image, either with a mouse drag or with $Ctrl + A$, and then strike $Alt + Shift + B$. The menu in figure C.4 shows up.

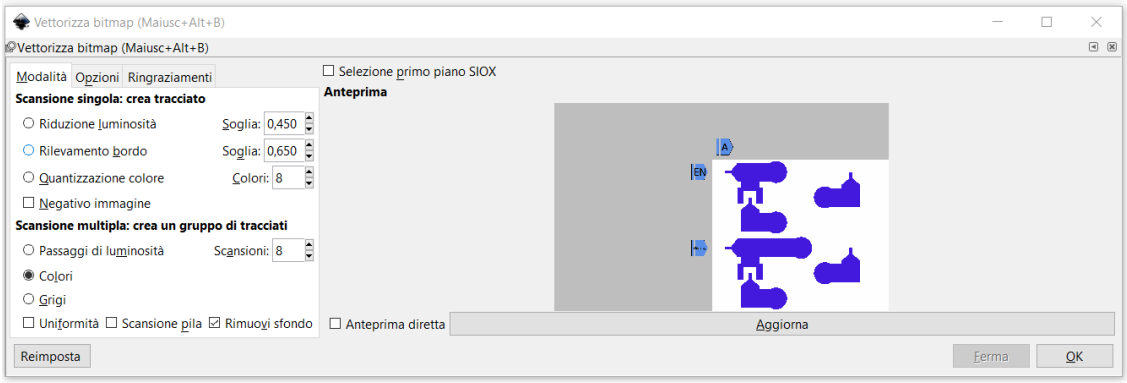


Figure C.4: SVG conversion

Check the same options and click *Ok*. An SVG version of the image is generated and is placed *over the other one*. The two images are perfectly overlapped. Remember, the one on the background is the PNG one, the other one is the SVG one. If they are overlapped, the one that gets selected when clicking over them is the one on the foreground, which is the SVG version. Drag it aside, as in figure C.5 (where the selected image is the SVG version, the one to be kept), and delete the other one (select it and then hit *Canc*, or *Double mouse click* and then *Delete*).

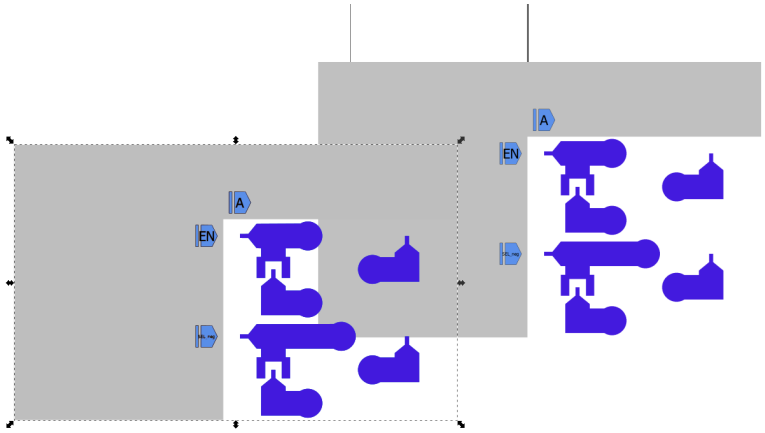


Figure C.5: Separation of the two (SVG and PNG) overlapping images

The screen should look like in figure C.6.

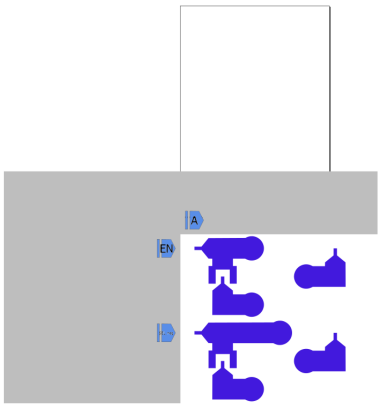


Figure C.6: SVG image after deletion of the PNG one

Now the SVG image needs some polishing. The SVG conversion chosen maps the colors of the PNG image into paths of the SVG image. Just two colors are needed, the one of the magnet (that changes for each layer) and the one of the inputs/outputs (light blue). They will become two different objects in Blender. However, the conversion adds some tiny spots of different color, that would hinder the 3D extrusion in Blender. Thus, they must be eliminated. To do so, zoom in the input zone. Click one the text until it is selected, and then *Right mouse click* and *Delete*. *Canc* here does not work, for no known reason. When the text has been canceled, the input appears as in figure C.7.

Figure C.7: Text deletion

Some unwanted colors are still there. Select them (as in the circle in figure C.8) and eliminate them. Do the same until those areas can be found, and eventually, when the circuit has just two colors, save it in SVG format (*File* → *Save as*, or

Shift Ctrl S).

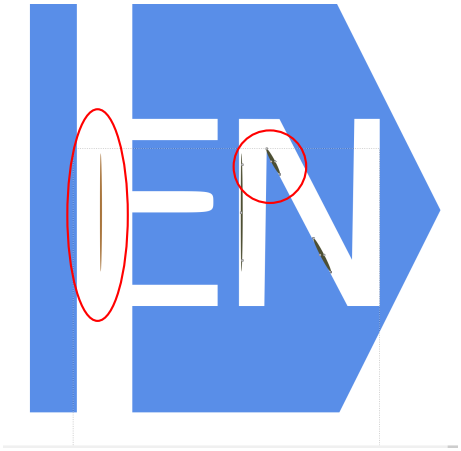


Figure C.8: Detail removal

These operations must be performed for each PNG file (one for every layer). Close Inkscape and open Blender. In Blender, the first thing to do is to get rid of the cube in the middle of the scene. Click on it with the right mouse button, a menu pops up, click on it or strike *Enter*, as in figure C.9

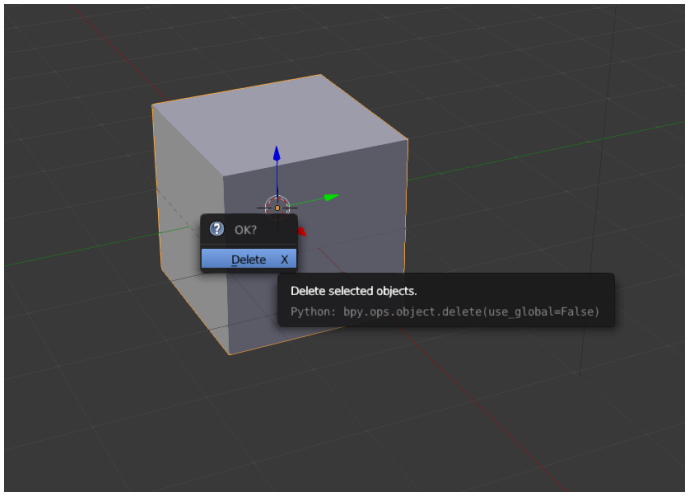


Figure C.9: Cube deletion in Blender

Once the scene is free from useless things, import the first layer. To do so, click *File* and then *Import*, and select SVG. Figure C.10 shows it.

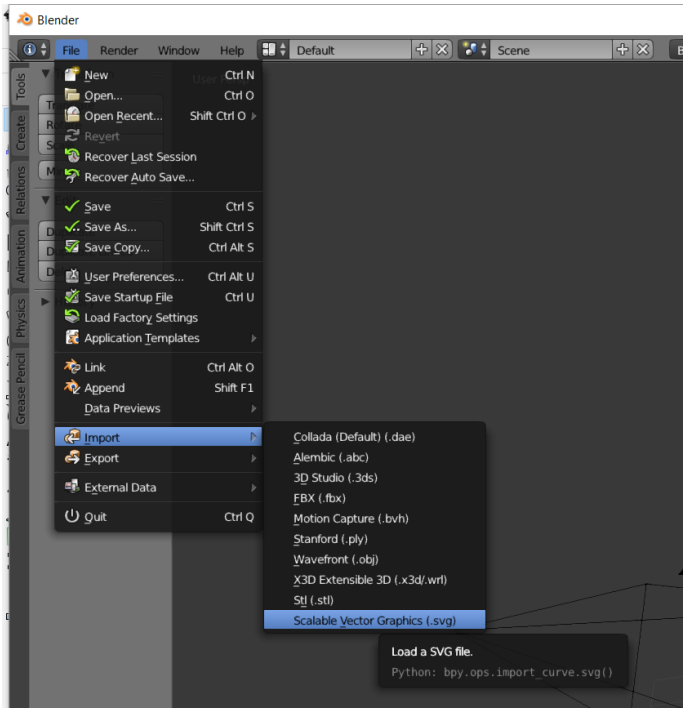


Figure C.10: Blender SVG import

The image of the first layer appears on the "floor" of the scene, as in figure C.11.

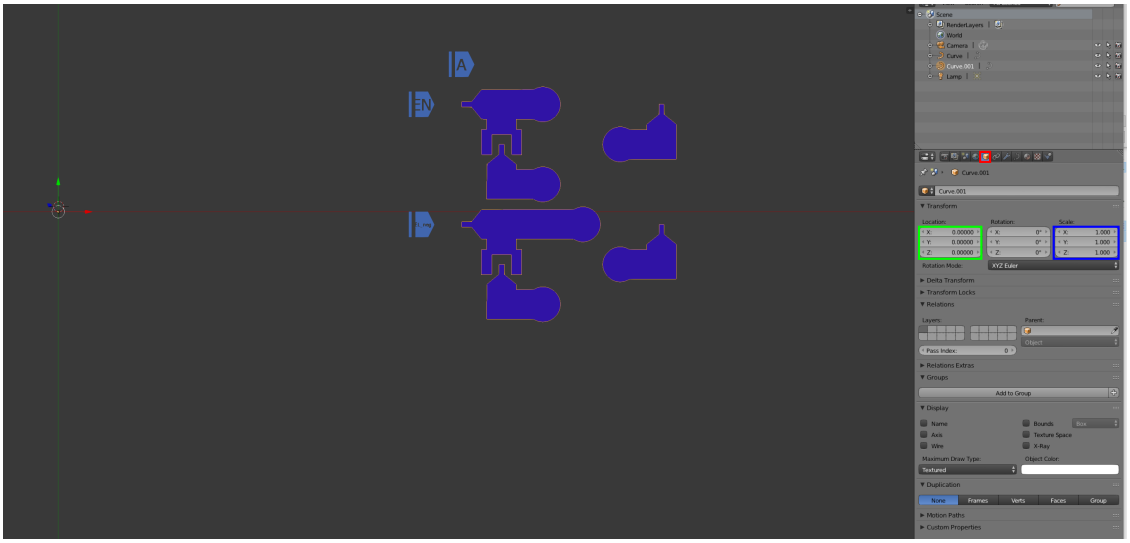


Figure C.11: SVG imported in Blender and scaled

Let us list some useful commands that allow to navigate the scene in Blender:

- The mouse wheel zooms in and out;
- *Shift* + *Center mouse button* to strafe left or right;
- *Center mouse button* + *mouse* to move around the scene with the mouse;
- *NumPad* 7 to look from the top (the view in figure C.11);
- *NumPad* 3 to look from the side.

Now select with the right mouse button the magnets (selection in Blender is always with right mouse button). When something is selected, its border becomes yellow, as the magnets in figure C.11 or in C.12. Look at the window on the right. Click on the icon within the red circle and set the scale in X and Y directions (blue box). A factor around a 10-20 should be fine. Then, set the same scaling factor for the inputs/outputs, after having selected them. This scaling step is not strictly necessary, but the origin of the objects (see the rest of this tutorial) is set manually, and a bigger objects reduces the error when doing that operation. Next step is the one where the third dimension is generated in the circuit.

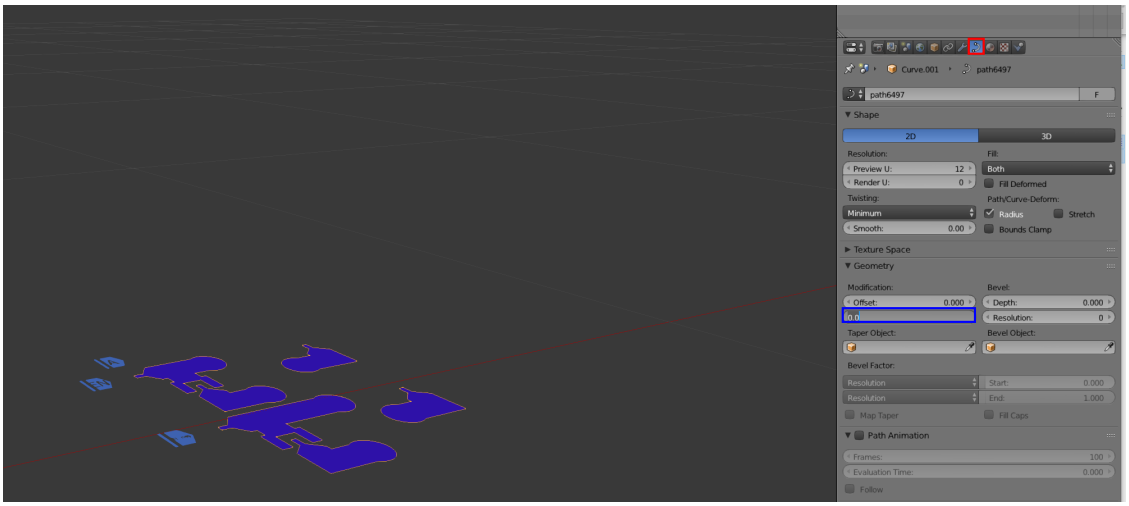


Figure C.12: Extrusion

Click on the icon inside the red box in figure C.12, and set a value for the *Extrusion* field. This set the thickness of the layer. Typical values lie between 0.05 and 0.2. The magnets extruded can be seen in figure C.13.

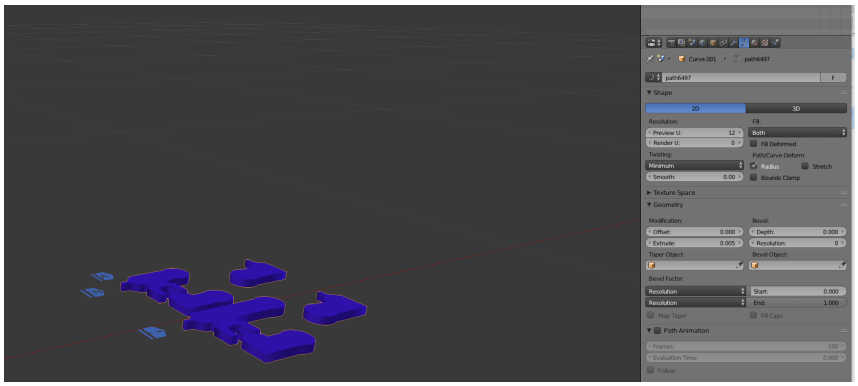


Figure C.13: Layer extruded

Now the origin of the two objects (the magnets and the inputs/outputs) must be set, so that they will be aligned with the other layers. Move around the circuit, find a point of view where that input added to the circuit can be clearly seen. Choose one of its vertices, and click, with the *left* mouse button, as in figure C.14.

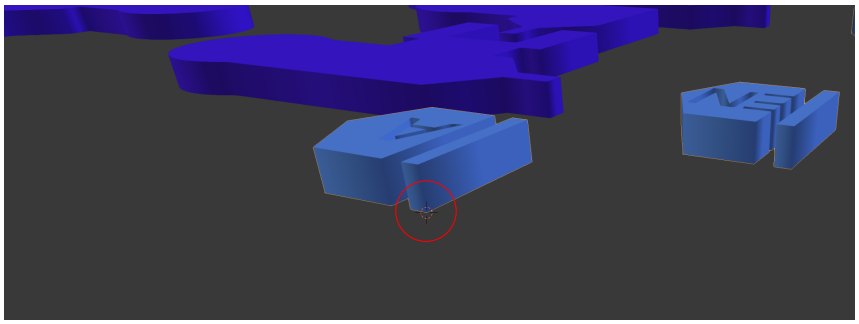


Figure C.14: Choice of the origin point

That cursor (in the red circle) is where the reference point of the objects will be. To set it, select the magnets, then on the left menu, click *Set origin* and select *Origin to 3D Cursor*. Figure C.15 shows the drop down menu with the magnets selected.

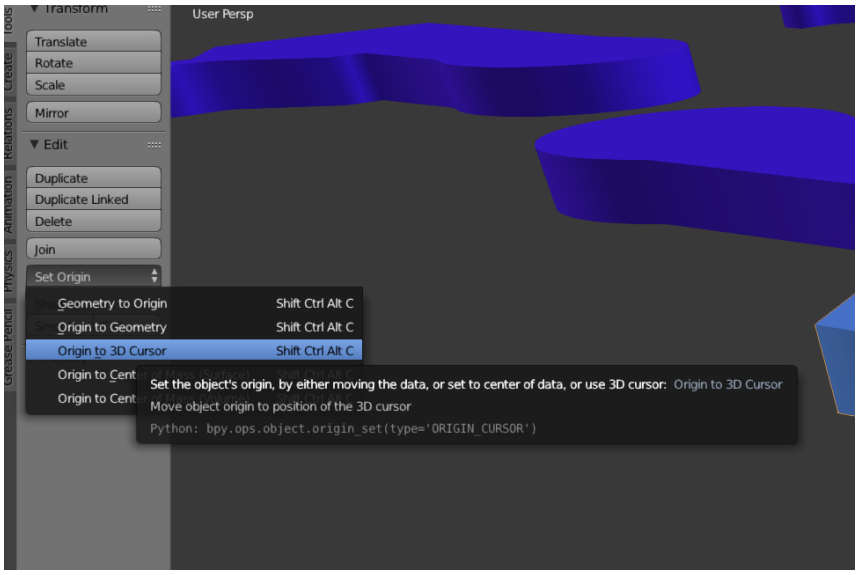


Figure C.15: Setting of the origin point (magnets)

Select the other object and repeat this same procedure *keeping the 3D cursor in the same position as before*. In order not to move it, avoid any left click of the mouse within the scene should be enough. Figure C.16 shows the procedure, with the magnets selected.

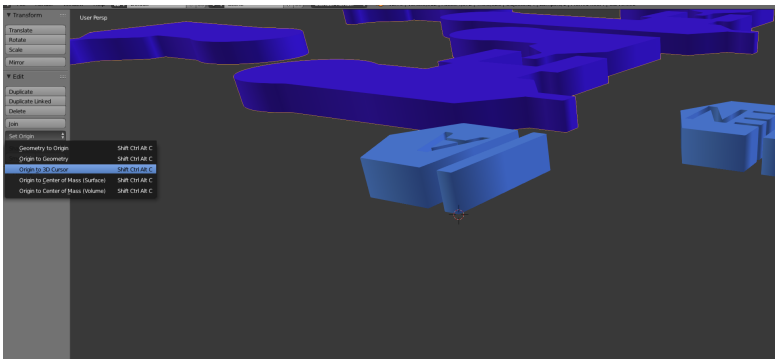


Figure C.16: Setting origin point (inputs/outputs)

Now the two objects have the correct origin, and they can be moved in any place. The idea is to move every layer in the same position, so they will be correctly aligned. To do so, the origin of the objects must be positioned at the same coordinates for every layer. Of course, these coordinates can be whatever, but the choice $X = 0$ and $Y = 0$ is very easy to remember and to digit. Move the two objects at point

(0,0). The coordinates can be set by clicking, on the right hand menu the icon in the red box in figure C.11, and then by chosing the coordinates in the fields within the green box. Figure C.17 shows the scene when the magnets have been moved to that point, whereas the inputs/outputs have not.

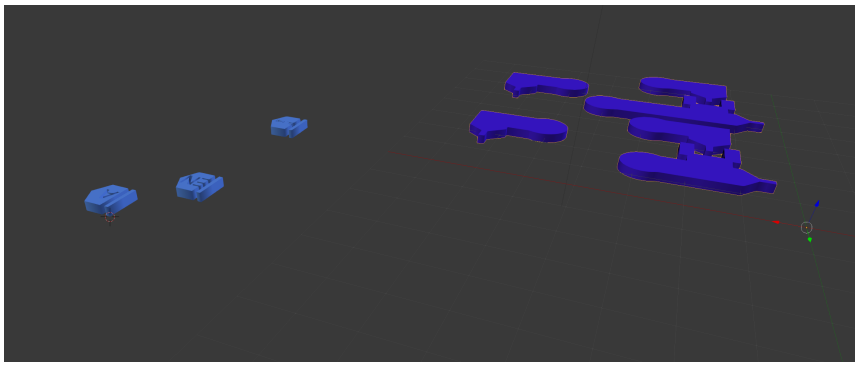


Figure C.17: Magnets moved to the origin

After having moved the two objects, hit *NumPad* 7. The view becomes the one in figure C.18. Select the inputs/outputs and hit *Tab*. The object "inputs/outputs" enters the edit mode.

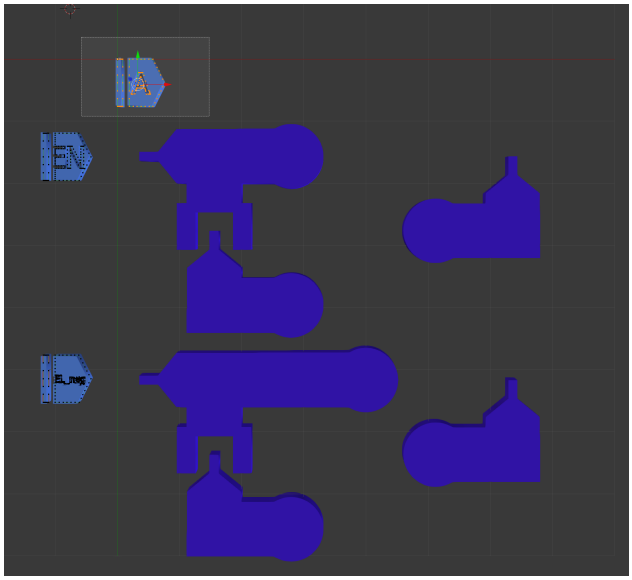


Figure C.18: Selection of the fake input

Hit *B*, which actives drag selection, and select all the vertices that make that

input used as reference point,as in figure C.18. Once it is selected, hit *Canc*. The window in figure C.19 shows up. Select *vertices* and click.

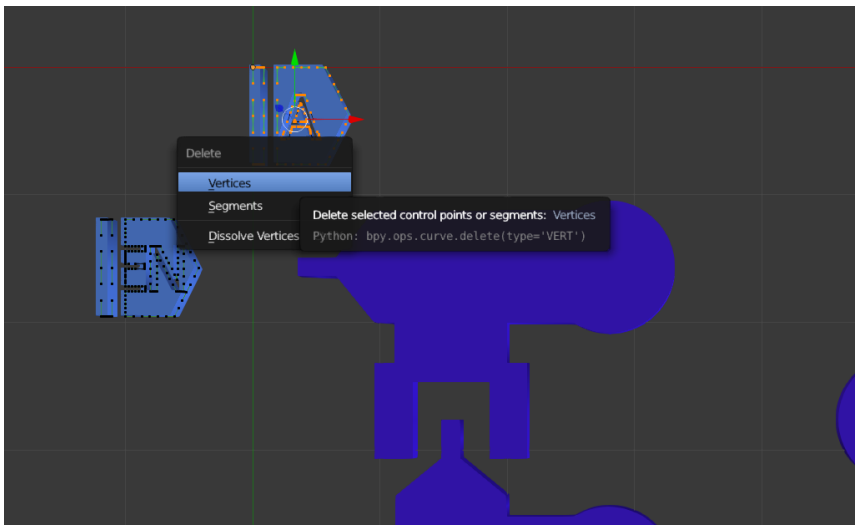


Figure C.19: Deletion of the fake input

Now the reference input is gone, and the work with the first layer is almost over. The last operation is raising. To raise an object, refer again to figure C.11, and choose a Z value other than 0. The value is not arbitrary. It works like this: when an object is extruded, the reference point for the Z axis is in the middle. It means that when an SVG is imported, the 2D objects are placed at the coordinate $Z = 0$; then, when a Δh quantity is set in the field of the blue box in figure C.12, the object is extruded towards both Z directions. In other words, the bottom face of the object is at $Z = -\Delta h$, and the top face is at $Z = \Delta h$. The first layer will be moved to $Z = \Delta h$, so that its bottom face is at $Z = 0$ and its top face at $Z = 2\Delta h$. Therefore, the second layer will be extruded by the same quantity, and its Z will be $2\Delta h$, so that its bottom face is at $Z = \Delta h$, right over the top face of the first layer, and its top face at $Z = 3\Delta h$. The two layers will look like in figure C.20.

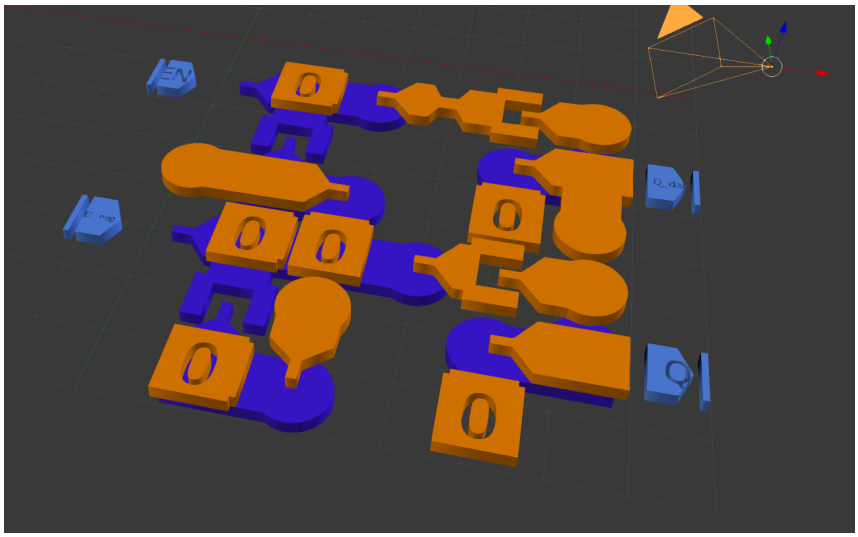


Figure C.20: Second layer placement

Then, third layer will undergo these same operations, and its Z will be $5\Delta h$, and so on, and a odd-multiples progression, until the top layer is reached.

Appendix D

Power Reports Of SAT Architecture Turned Into CMOS

As said in chapter 5, the CMOS versions of the SAT hardware were characterized also with respect to power consumption. This little appendix chapter provides the results obtained and explains the method. When doing power measurements over a circuit, it is crucial how the switching activity of the nodes is determined. A large variety of approaches to the matter exists; in this work the backannotation method was the one chosen. Design Vision creates a netlist for the circuit, which will be then filled out by Modelsim after a simulation. The simulation must have as input vectors values representing fairly well the real use case; the whole process depends on this particular point. When Modelsim has generated the file with the switching activity of each node, Design Vision can elaborate it to eventually come out with a value for the power consumption. In Modelsim, the simulation can be either pre-synthesis or post-synthesis; in this case it has been a pre-synthesis, less accurate but quicker. As for the input vectors, this is a pretty simple case: a number great enough of random inputs will do. Since simulations are very quick, and the bit parallelism is just four bits, thousands of input vectors can be tested. Four circuits are to be analyzed, because each one could be either a four-locations array or a 16-locations array, and

either a behavioral model or a structural model. Every one of these four circuits is synthesized seven times, every one with a different maximum clock period allowed. These values were set as constraints:

500 ps	1 ns	10 ns	100 ns	1 μs	10 μs	100 μs

Those values are not the clock period, but rather the constraint set on Design Compiler. Anyway, the actual clock period is quite close to that. Thus, every circuit is synthesized in seven different versions (all this work was automated by means of scripts). For every version, Design Vision tries to reduce to the minimum possible value the dynamic and the leakage power. Actually the library used is not a low-power one, and does not allow supply voltage tuning or any other low-power technique; hence, these power reduction attempts always leads to almost no improvement. The script includes these steps because with a different library great improvements could result. For a better reading, the power figures obtained are plotted with MATLAB. The three plots are in figure D.1, D.2 and D.3. The raw data the plots are made with are in table D.1 and D.2:

Power [μW] / Clock constraint	Behavioral			Structural		
	Dynamic	Leakage	Total	Dynamic	Leakage	Total
500 ps	917	9,21	927	1024	10,26	1034
1 ns	453	8,43	462	499	8,74	507
10 ns	45,7	8,62	54,3	49,4	8,67	58,0
100 ns	4,56	8,58	13,1	4,94	8,71	13,6
1 μs	0,453	8,48	8,93	0,493	8,70	9,19
10 μs	0,0460	8,56	8,61	0,0494	8,72	8,77
100 μs	0,00452	8,55	8,56	0,00494	8,68	8,69

Table D.1: Power consumption figures for the CMOS SAT architecture, with size 2 by 2

A possible way to compare power could be matching the delays, so that the two circuits have the same speed, and check which one absorbs the lowest amount of power. As said, pNML power consumption for a specific circuit cannot yet be assessed. For what concerns CMOS the data shows that *for this specific technology, which is not low-power oriented*, the value of power settles around 10 μW for the

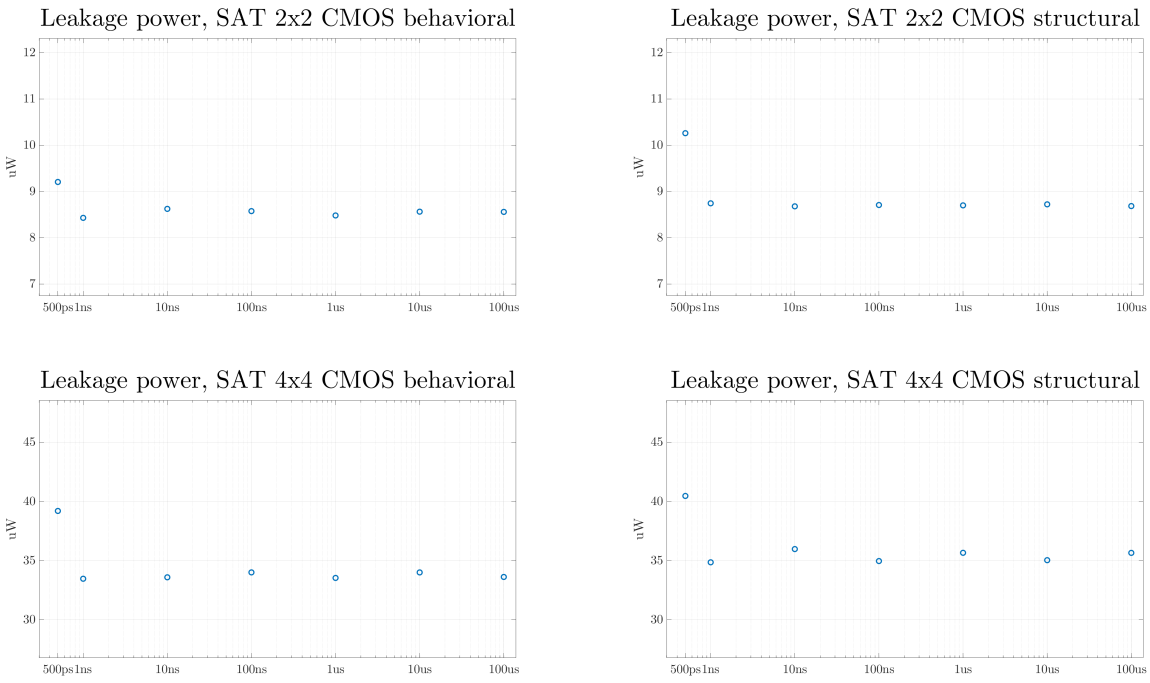


Figure D.1: Dynamic power for the CMOS SAT architecture

Power [μW] / Clock constraint	Behavioral			Structural		
	Dynamic	Leakage	Total	Dynamic	Leakage	Total
500 ps	3767	39,2	3807	3899	40,45	3940
1 ns	1725	33,5	1758	1747	34,8	1782
10 ns	172	33,6	205	179	36,0	215
100 ns	17,1	34,0	51,1	17,4	34,9	52,3
1 μs	1,72	33,5	35,2	1,79	35,6	37,4
10 μs	0,172	34,0	34,2	0,174	35,0	35,2
100 μs	0,0171	33,6	33,6	0,0179	35,6	35,7

Table D.2: Power consumption figures for the CMOS SAT architecture, with size 4 by 4

4-locations circuit and around 40 μW. The corresponding clock value is in the μs range, 10 or even 100. A lower clock period would make the CMOS much faster than the pNML: the latter has a clock period approximately equal to 1 or 2 μs, but the comparison must be made between the CMOS clock period and the pNML

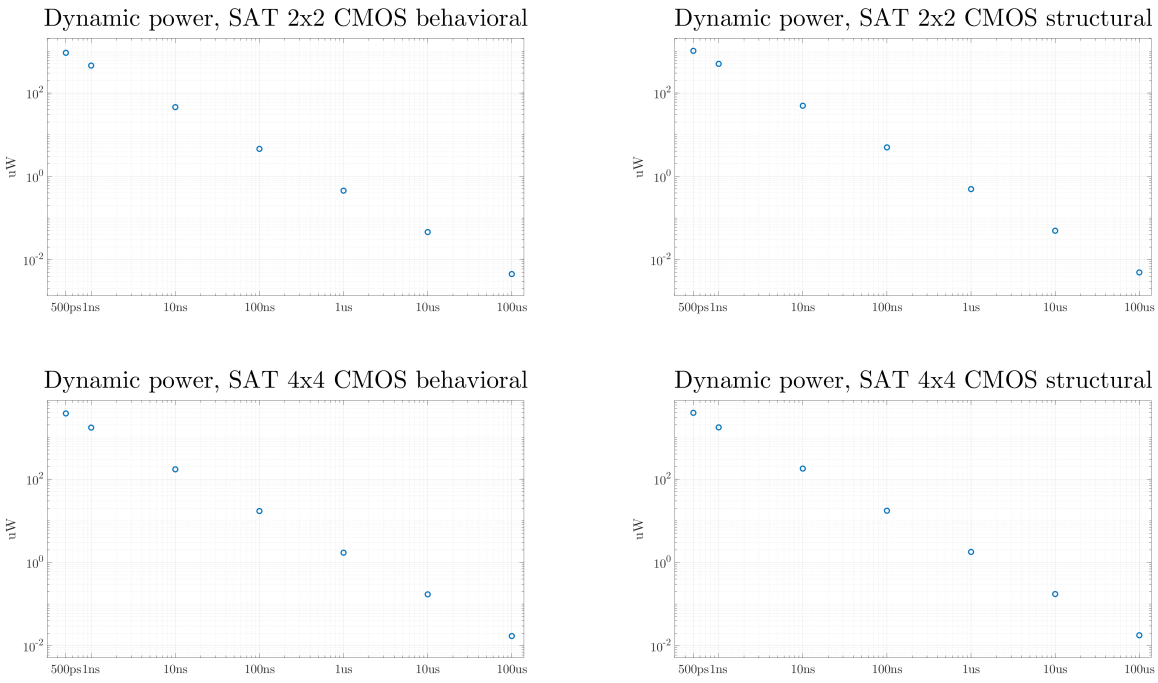


Figure D.2: Leakage power for the CMOS SAT architecture

notch period. The notch period depends on the circuit, the SAT one has a notch period as long as almost 30 clock periods. Therefore, the CMOS circuit is as fast as the pNML one when its clock period is equal to about 30 pNML clock periods.

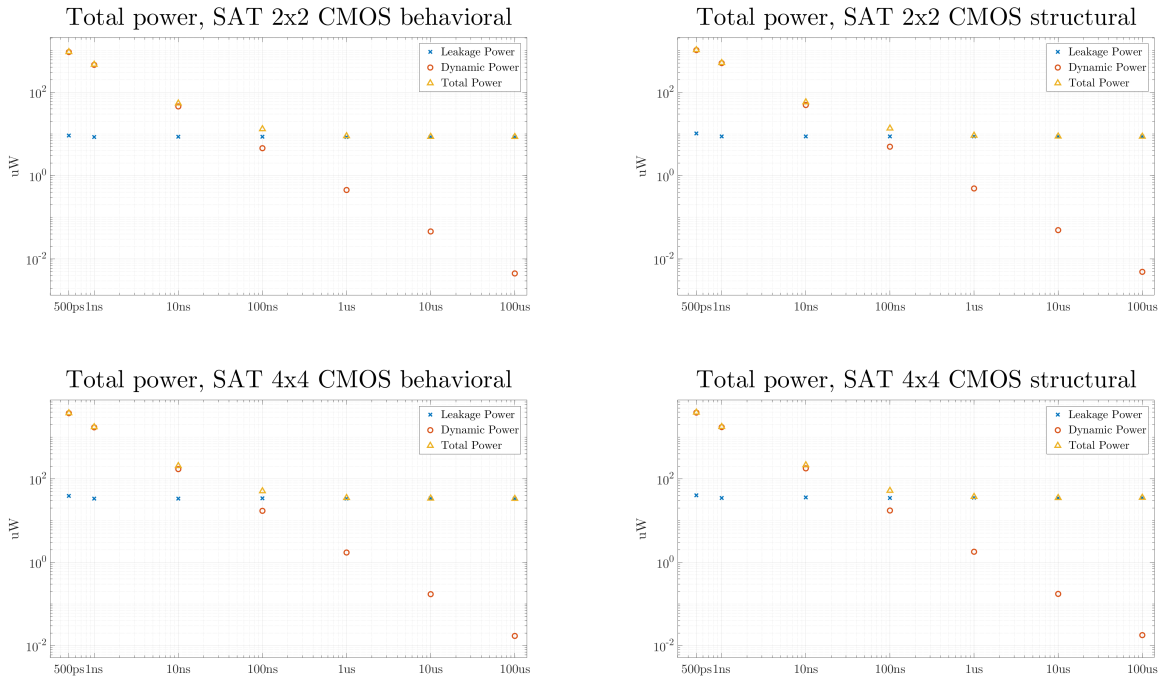


Figure D.3: Total power, with the leakage and dynamic component, for the CMOS SAT architecture

Appendix E

MagCAD Hints

MagCAD [18] is the software used to design all the pNML circuits of this work; it has been entirely developed in Turin's Polytechnic. It has a graphical interface, with drag and drop elements that can be placed on the work space; in overall, it is fairly intuitive and easy to use. Once the design is done it produces a VHDL description that includes delays. This tutorial will just include some peculiarities of the program, that might come in handy when designing a pNML circuits. Moreover, this technology will be the only one described, even if the tool allows also iNML design. For a more complete description of the software, the reader is addressed to the manual in the website.

E.1 Geometrical Parameters

When starting a new design, the first thing the tool requires, after having selected the technology (red oval in figure E.1) is the definition of the geometrical parameters (in that same window). The two fields must be filled with the desired grid size, which is what throughout this work has been called "square", and the width of the nanomagnet which is slightly narrower than the grid size. The magnet width size will determine the appearance of the circuit: in figure E.2, two different values have been chosen, and the wire aspect ratio changes accordingly.

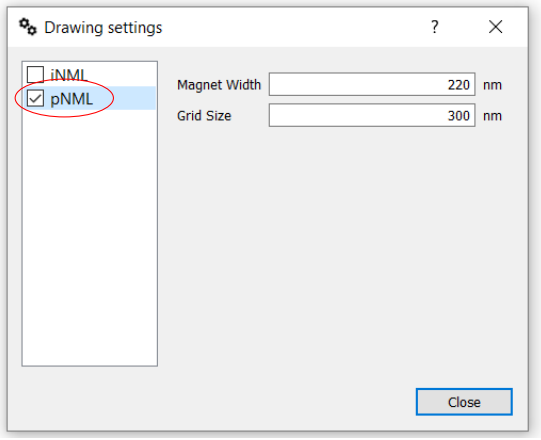
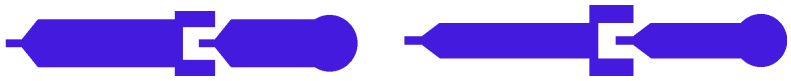


Figure E.1: Technology choice and geometry definition



(a) 330 *nm* (b) 150 *nm*

Figure E.2: Nanowires with two different nanomagnet width

E.2 Forbidden configurations

Once these parameters are set, the design can be started by dragging and dropping elements on the workspace. Describing the elements one by one is absolutely unnecessary, but some distinctive traits should be pointed out. Consider figure E.3, where an input is placed and coupled with a nucleation center. The input behaves

Figure E.3: Input coupled with an ANC

in a way totally similar to a coplanar nanomagnet: it means that it will force the opposite of its own value. Keep in mind this point, that at first is definitely misleading. By the way, if the input is arranged in a different fashion, as in figure E.4, the nanowires would get the same value as the input. But this configuration *is absolutely forbidden*. Even if MagCAD allows it, such a configuration cannot be manufactured. MagCAD does not issue any warning or error because it could be useful when designing circuits to be exported as models (it will be described later

Figure E.4: Forbidden input configuration

on). On the other hand, the arrangement in figure E.5 is the standard way to draw an output.

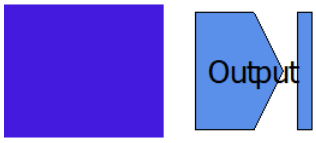


Figure E.5: Output placement

There are other configurations that do not trigger any software error but are not physically correct. Figure E.6 shows some of them. Case E.6a and E.6b are

(a) ANC over nanowire (b) Inverter over nanowire (c) Notch over nanowire

Figure E.6: Examples of forbidden configurations (unwanted coupling)

actually the same thing, because an inverter is nothing more than a nanomagnet coupled with the ANC of the next one. Therefore, this is why the arrangement is forbidden: the blue wire on the bottom would influence the ANC. For this same reason, no fixed magnet should be placed over or under a nucleation center (unless it actually must influence it) or an inverter. Case E.6c is different, because there is no ANC: that element is a notch. Anyways, the effect is the same: notches are sensitive to the magnetic fields of the elements nearby, and they would get coupled. Sometimes the ANC must be actually influence by the wire below (or above). In

this case, the VIA element must be used, so that MagCAD knows that it is a wanted coupling. Anyways, in none of the forbidden cases mentioned MagCAD warns the user, because sometimes the insertion of inverters is carried out in a later stage of the design, to try to pipeline the circuit. In this stage, the inverters are usually placed with no concern towards their position; then, if the pipelining actually improves the circuit, the inverters are moved to allowed locations. Basically, it is just a matter of convenience.

E.3 Coordinates System

The workspace could be considered a three dimensional discrete space, with every element occupying a well-defined spot. Hovering the mouse over the workspace the x and y coordinates are shown in the bottom right corner of the screen (figure E.7, the red oval). The layer number can be thought as the third coordinate. The blue square

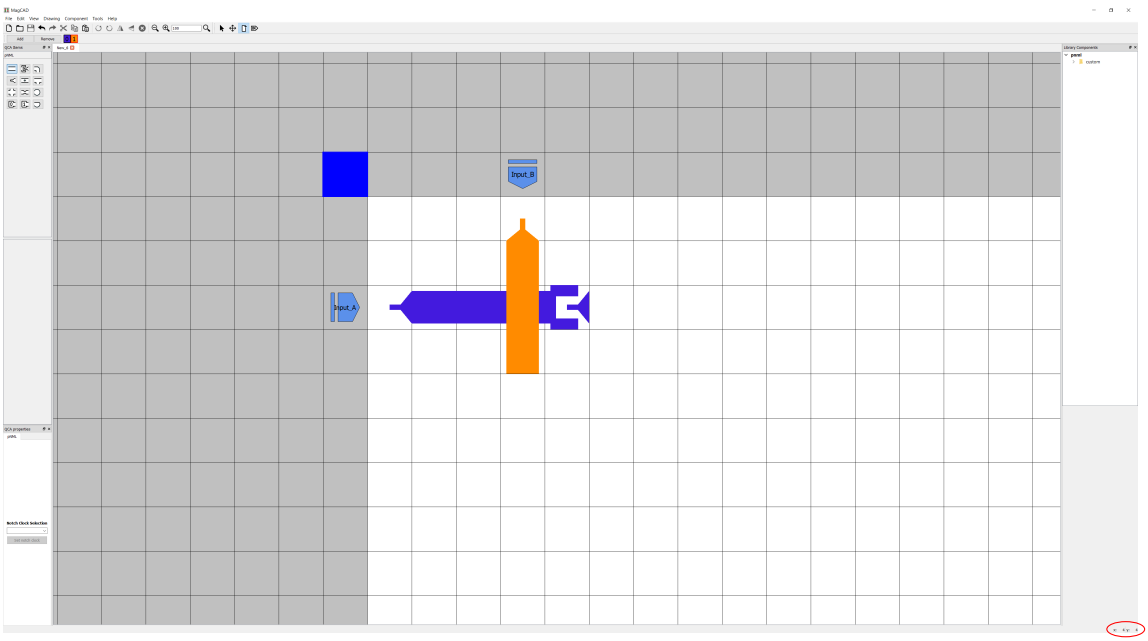


Figure E.7: MagCAD coordinates and forbidden location (the blue square)

is the location $x = -1$,$y = -1$, and nothing can be placed there. Moreover, the design must be aligned to the top left corner of the workspace as in figure E.7, with the leftmost inputs on line $x = -1$ and topmost input on line $y = -1$. If there are no

inputs on the top or left edge, the magnets should anyway be placed as in figure E.7 (topmost elements on $y = 0$, leftmost one on $x = 0$). This is done in order to match the coordinates of the workspace with the ones of the report. If something goes wrong in the VHDL export, the error is reported on a .log file, with the coordinates and layer of the failing element. The report assigns coordinates $x = 0$,$y = 0$ to the top left corner of the design (as said, disregarding possible inputs or outputs), and if the placement rule just stated is ignored, the two reference systems will be shifted with respect to each other. In figure E.7, a grid is drawn over the workspace: to turn it on/off, click on "drawing", then "pNML", and last "grid", as in figure E.8. It might be very helpful.

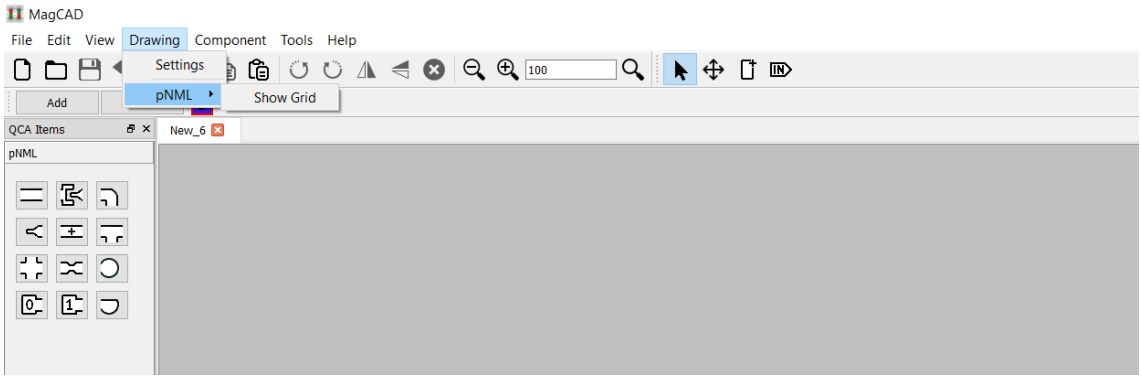


Figure E.8: Grid activation

E.4 Components

If the circuits design is correct, MagCAD is able generates a VHDL description that could be simulated. If the circuit is not correct, MagCAD can export it as "component". A component is useful when a circuit includes many replicas of the same building block: the building block is designed just once, then exported as component and placed on the design as many times as needed. To export as component, the procedure is the same as the one for the VHDL generation; if VHDL cannot be generated, because the circuit could not work alone, the component is generated in any case. To avoid the VHDL export error, uncheck the "export VHDL" box (as in figure E.9). The components are linked in MagCAD library of component, that can

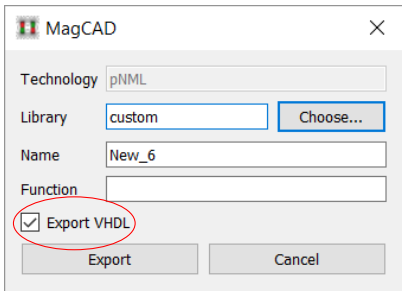


Figure E.9: Disabling VHDL export

be opened by clicking on “Tools”, “Windows”, “Library components” (figure E.10). This way, the library windows appears on the right hand side of the main window,

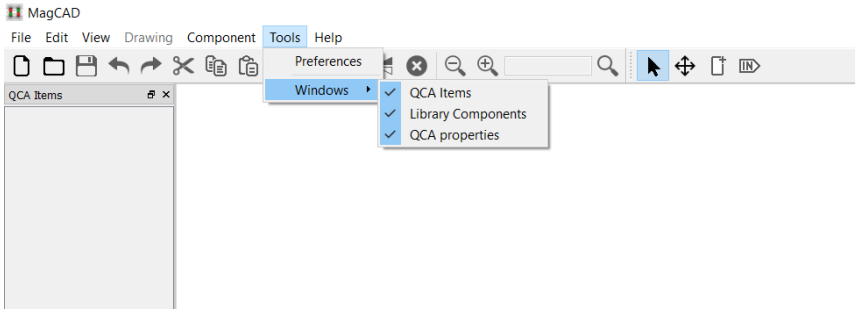


Figure E.10: Component import into the design

and by clicking “custom” (in the red oval), the whole set of exported components pops up. The right mouse button click on one of them opens a windows, as in figure E.11

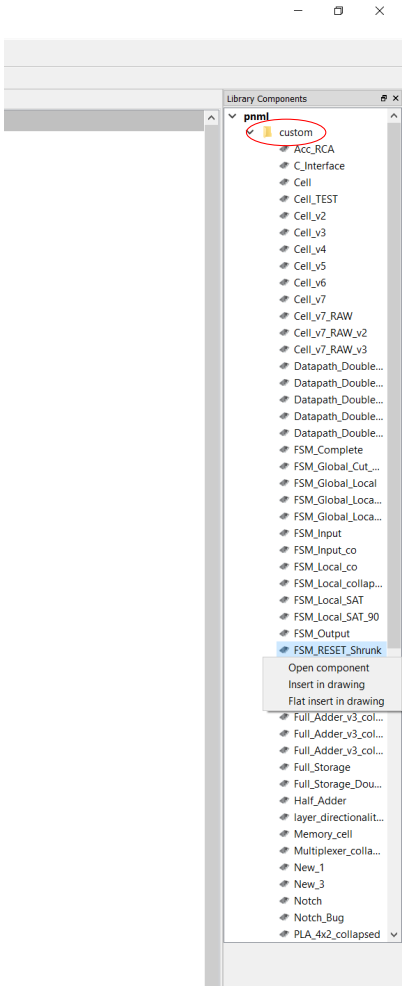


Figure E.11: Components' options

with three options:

- **Open component**: a new window with the layout of the component opens. It does not allow modifications, and not even selections. The only thing that can be done is changing the layer visualization;

- **Insert in drawing**: the component is inserted in the design as a black box (figure E.12). If the component is multi layer, the bottom layer of the inserted component will be placed on the current layer, and all the layers above will be placed accordingly. By clicking on the component, a new window opens as in the previous point (so, no changes can be made);

- **Flat insert in drawing**: the component is placed on the workspace, as a modifiable entity. The layers are sorted following the rule on the previous point. However, in this case inputs and outputs are not exported;

Figure E.12: Component placed in the workspace

The *Insert in drawing* option must be used when the component is already tested and is not expected to undergo any modification or adaptation. Inputs and outputs must be placed along the rectangular border, otherwise would not be reachable. Pretty often, when testing, dummy outputs are placed on some points of the circuits, in order to easily monitor some signals. These outputs must be removed. If, on the other hand, the component is likely to be somehow changed in a non predictable way (should it be predictable, it is better to do it before exporting), the *Flat insert in drawing* is a better option, but the the removal of inputs and outputs must be dealt with.

E.5 Pads coupling directions

The key feature of MagCAD is that nucleation centers placed besides pad are influenced by them. However, the direction and relative orientation of these two elements does matter. A pad couples with an ANC nearby only if the position is one of those shown in figure E.13, and does not couple with nucleation center arranged as in figure E.14

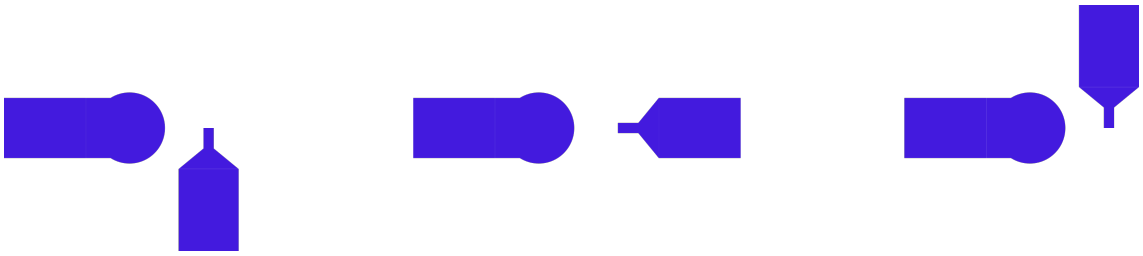


Figure E.13: Example of pads coupled with ANC

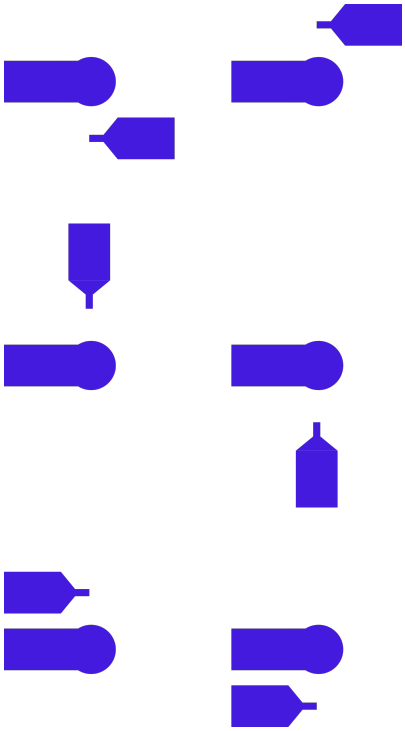
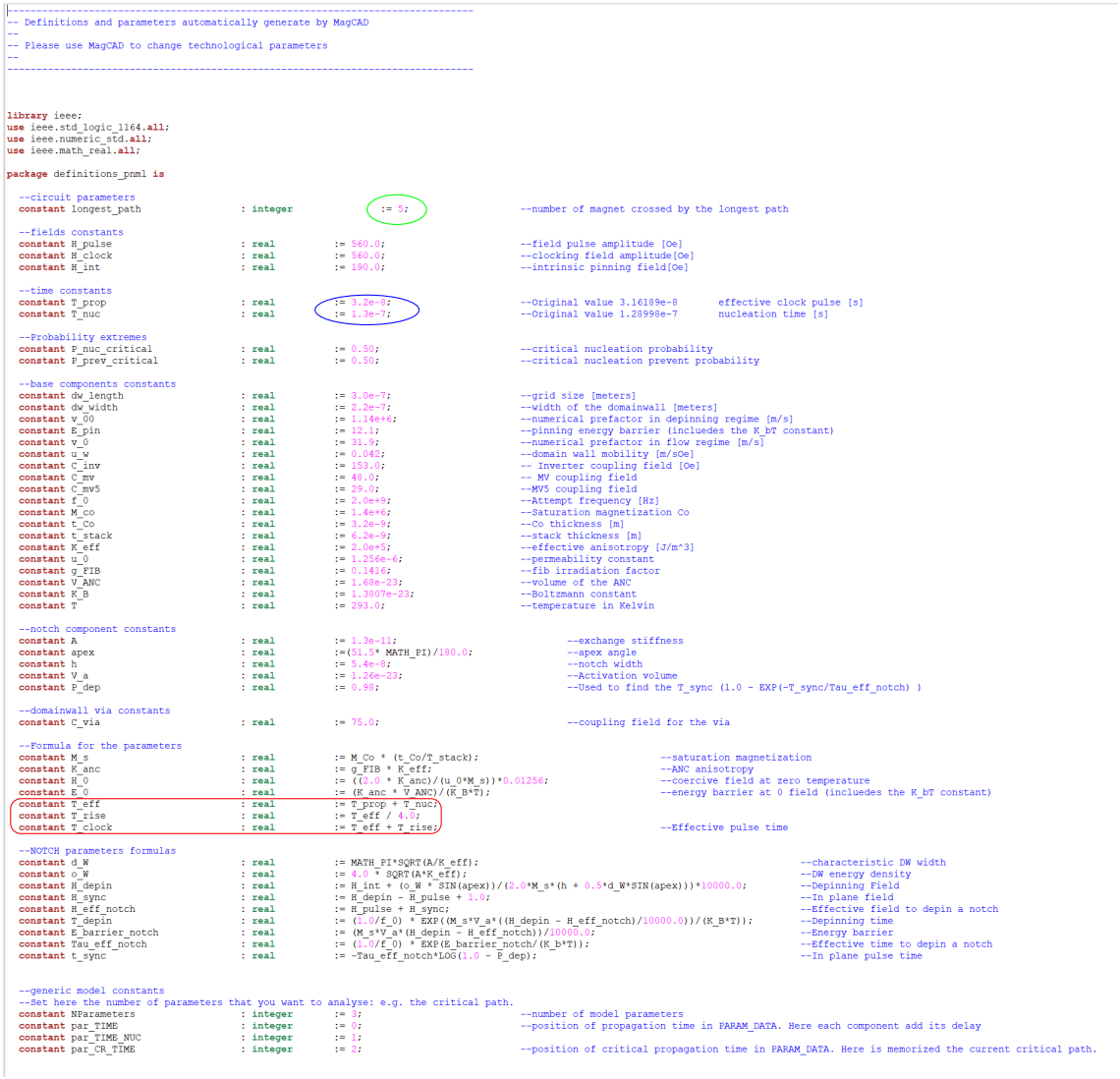


Figure E.14: Example of pads *not* coupled with ANC

E.6 Reports

At the end of a successful compilation, the file "definitions_pnml.vhd" (which is a file actually used by the simulation tool to get the delay parameters) contains a couple of interesting information. The first one is about the maximum length of the path between any two elements that wait the clock edge to propagate (inverters, nucleation centers, VIAs), reporting in fact the critical path in terms of squares. The other one is "hidden", meaning that it must be calculated starting from two other values. A report looks like the image E.15 The critical path is the one in



```
--------------------------------------------------------------------------------
-- Definitions and parameters automatically generate by MagCAD
--
-- Please use MagCAD to change technological parameters
--
--------------------------------------------------------------------------------



library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

package definitions_pnml is

  --circuit parameters
  constant longest_path               : integer             := 5;              --number of magnet crossed by the longest path

  --fields constants
  constant H_pulse                    : real          := 560.0;                --field pulse amplitude [Oe]
  constant H_clock                    : real          := 560.0;                --clocking field amplitude[Oe]
  constant H_int                      : real          := 190.0;                --intrinsic pinning field[Oe]

  --time constants
  constant T_prop                     : real          := 3.2e-8;               --Original value 3.16189e-8       effective clock pulse [s]
  constant T_nuc                      : real          := 1.3e-7;               --Original value 1.28998e-7       nucleation time [s]

  --Probability extremes
  constant P_nuc_critical             : real          := 0.50;                 --critical nucleation probability
  constant P_prev_critical            : real          := 0.50;                 --critical nucleation prevent probability

  --base components constants
  constant dw_length                  : real          := 3.0e-7;               --grid size [meters]
  constant dw_width                   : real          := 2.2e-7;               --width of the domainwall [meters]
  constant v_00                       : real          := 1.14e+6;              --numerical prefactor in depinning regime [m/s]
  constant E_pin                      : real          := 12.1;                 --pinning energy barrier (incluedes the K_bT constant)
  constant v_0                        : real          := 31.9;                 --numerical prefactor in flow regime [m/s]
  constant u_w                        : real          := 0.042;                --domain wall mobility [m/sOe]
  constant C_inv                      : real          := 153.0;                -- Inverter coupling field [Oe]
  constant C_mv                       : real          := 48.0;                 -- MV coupling field
  constant C_mv5                      : real          := 29.0;                 --MV5 coupling field
  constant f_0                        : real          := 2.0e+9;               --Attempt frequency [Hz]
  constant M_co                       : real          := 1.4e+6;               --Saturation magnetization Co
  constant t_Co                       : real          := 3.2e-9;               --Co thickness [m]
  constant t_stack                    : real          := 6.2e-9;               --stack thickness [m]
  constant K_eff                      : real          := 2.0e+5;               --effective anisotropy [J/m^3]
  constant u_0                        : real          := 1.256e-6;             --permeability constant
  constant g_FIB                      : real          := 0.1416;               --fib irradiation factor
  constant V_ANC                      : real          := 1.68e-23;             --volume of the ANC
  constant K_B                        : real          := 1.3807e-23;           --Boltzmann constant
  constant T                          : real          := 293.0;                --temperature in Kelvin

  --notch component constants
  constant A                          : real          := 1.3e-11;                      --exchange stiffness
  constant apex                       : real          :=(31.5* MATH_PI)/180.0;         --apex angle
  constant h                          : real          := 5.4e-8;                       --notch width
  constant V_a                        : real          := 1.26e-23;                     --Activation volume
  constant P_dep                      : real          := 0.98;                         --Used to find the T_sync (1.0 - EXP(-T_sync/Tau_eff_notch) )

  --domainwall via constants
  constant C_via                      : real          := 75.0;                         --coupling field for the via

  --Formula for the parameters
  constant M_s                        : real          := M_Co * (t_Co/T_stack);                                  --saturation magnetization
  constant K_anc                      : real          := g_FIB * K_eff;                                          --ANC anisotropy
  constant H_0                        : real          := ((2.0 * K_anc)/(u_0*M_s))*0.01256;                      --coercive field at zero temperature
  constant E_0                        : real          := (K_anc * V_ANC)/(K_B*T);                                --energy barrier at 0 field (incluedes the K_bT constant)
  constant T_eff                      : real          := T_prop + T_nuc;
  constant T_rise                     : real          := T_eff / 4.0;
  constant T_clock                    : real          := T_eff + T_rise;                                         --Effective pulse time

  --NOTCH parameters formulas
  constant d_W                        : real          := MATH_PI*SQRT(A/K_eff);                                                                  --characteristic DW width
  constant o_W                        : real          := 4.0 * SQRT(A*K_eff);                                                                    --DW energy density
  constant H_depin                    : real          := H_int + (o_W * SIN(apex))/(2.0*M_s*(h + 0.5*d_W*SIN(apex)))*10000.0;                    --Depinning Field
  constant H_sync                     : real          := H_depin - H_pulse + 1.0;                                                                --In plane field
  constant H_eff_notch                : real          := H_pulse + H_sync;                                                                       --Effective field to depin a notch
  constant T_depin                    : real          := (1.0/f_0) * EXP((M_s*V_a*((H_depin - H_eff_notch)/10000.0))/(K_B*T));                   --Depinning time
  constant E_barrier_notch            : real          := (M_s*V_a*(H_depin - H_eff_notch))/10000.0;                                              --Energy barrier
  constant Tau_eff_notch              : real          := (1.0/f_0) * EXP(E_barrier_notch/(K_b*T));                                               --Effective time to depin a notch
  constant t_sync                     : real          := -Tau_eff_notch*LOG(1.0 - P_dep);                                                        --In plane pulse time


  --generic model constants
  --Set here the number of parameters that you want to analyse: e.g. the critical path.
  constant NParameters                : integer       := 3;                    --number of model parameters
  constant par_TIME                   : integer       := 0;                    --position of propagation time in PARAM_DATA. Here each component add its delay
  constant par_TIME_NUC               : integer       := 1;
  constant par_CR_TIME                : integer       := 2;                    --position of critical propagation time in PARAM_DATA. Here is memorized the current critical path.
```

Figure E.15: Report with most important parameters outlined

green oval, whereas the two parameters in the blue one are propagation time and nucleation time. The way the clock half period is calculated is in the three lines in the red box. If the *T_rise* parameter is *T_eff/4.0* (as it is supposed to be) the full clock period is simply $(T_{prop} + T_{nuc}) \cdot 2.5$. Consulting this file is useful when the design must be somehow improved: for example, the design is done and the designer decides that no nanowire should be longer than 5 squares. He/she could trim the nanowires and then check this file. Or, the designer might be looking for a specific maximum clock frequency: he/she could design the circuit and then

check the clock period. If it is too long, an adjustment will be made accordingly. Some technological variations can be done if needed, by simply exporting again the desing and setting different parameters in the window in figure E.16. This allows to check the circuit performances with different sets of technological parameters. The nanomagnet geometry, even if it had already been set as in section E.1 can now be changed.

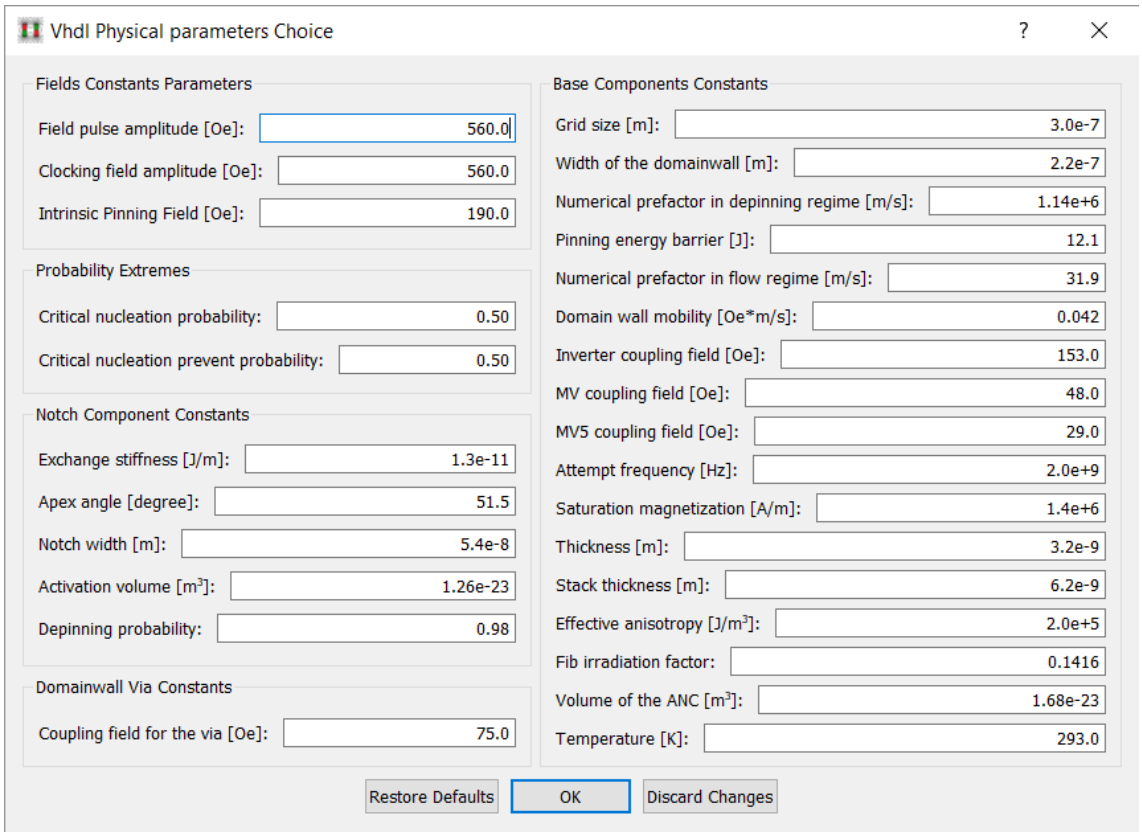


Figure E.16: Technological parameters definition

Bibliography

[1] P. Douglas Tougaw and Craig S. Lent. "Logical Devices Implemented Using Quantum Cellular Automata". In: *Journal of Applied Physics* (1994).

[2] Mariagrazia Graziano et al. "An NCL-HDL Snake-Clock-Based Magnetic QCA Architecture". In: *IEEE Transactions on Nanotechnology* (2011).

[3] M. T. Niemier et al. "Nanomagnet Logic: Progress Towards System-Level Integration". In: *Journal of Physics: Condensed Matter* (2011).

[4] Marco Vacca et al. "Magnetoelastic clock system for NanoMagnet Logic". In: *IEEE Transaction on Nanotechnology* (2014).

[5] Michael Niemier et al. "Boolean logic through shape-engineered magnetic dots with slanted edges". In: *IEEE Transactions on Nanotechnology* (2010).

[6] Stephan Breitkreutz et al. "Nanomagnetic Logic: Demonstration of Directed Signal Flow for Field-coupled Computing Devices". In: *Proceeding of the European Solid-State Device Research Conference* (2011).

[7] Jacques-Olivier Klein et al. "Synthesis of Finite State Machines with Magnetic Domain Wall Logic". In: *IEEE International Symposium on Circuits and Systems* (2007).

[8] Fabrizio Cairo et al. "Domain Magnet Logic (DML): A New Approach to Magnetic Circuits". In: *Proceedings of the 14th IEEE International Conference on Nanotechnology* (Aug. 2014).

[9] S. Breitkreutz et al. "Domain Wall Gate for Magnetic logic and Memory Applications with Perpendicular Anisotropy". In: *IEEE International Electron Devices Meeting (IEDM)* (2013).

[10] Antonino Ferrara. "Study and Design of Memories and Programmable Structures in Emerging Magnetic Technologies". MA thesis. Politecnico di Torino, Mar. 2017.

[11] Gina Jiang. "Implementation and Synthesis of Algorithms Parallelization in ASIC". MA thesis. Politecnico di Torino, 2017.

[12] NanGate Corporation. *NanGate FreePDK45 Open Cell Library*. 2008.

[13] Stephan Breitkreutz. "Perpendicular Nanomagnetic Logic: Digital Logic Circuits from Field-coupled Magnets". PhD thesis. Technical University of Munich, 2015.

[14] Elisa Plozner. "pNML Architecture: Modelling and Analysis". MA thesis. Politecnico di Torino, Oct. 2016.

[15] S. Breitkreutz et al. "Influence of the Domain Wall Nucleation Time on the Reliability of Perpendicular Nanomagnetic Logic". In: *Proceedings of the 14th IEEE International Conference on Nanotechnology* (Aug. 2014).

[16] S. Breitkreutz et al. "Time-dependent Domain Wall Nucleation Probability in Field-Coupled Nanomagnets with Perpendicular Anisotropy". In: *Journal of Applied Physics* (2015).

[17] Rumi Zhang et al. "A Method of Majority Logic reduction for Quantum Cellular Automata". In: *IEEE Transactions on Nanotechnology* (2004).

[18] F. Riente et al. "MagCAD: A Tool for the Design of 3D Magnetic Circuits". In: *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 3 (2017), pp. 65–73. DOI: 10.1109/JXCDC.2017.2756981.