ENS'07 *Paris, France, 3-4 December 2007*

A layered methodology for fast deployment of new technologies

Loïc Lagadec, Bernard Pottier, and Alix Poungou
LESTER FRE 2734, Université de Bretagne Occidentale
loic.lagadec@univ-brest.Fr

***Abstract*—Few efforts have been reported in building computing architectures out of nanodevices by contrast with the extensive research focusing on nanodevices fabrication aspects. Addressing mass market and reducing time-to-market appear now as key issues. We propose a layered approach, taking advantage of FPGAs good characteristics, to offer the promise of fast deployment of new technologies.**

I. Introduction

A variety of nanodevice technologies have been demonstrated. While there are many practical challenges still remaining, a key issue is now "How do we enable fast deployment of new technologies to mass market?"

Despite research projects will go on offering new computing solutions to implement applications, it remains difficult to assess the real interest for final products, as well as to produce some usable development environment for these products.

Applying a methodology similar to the interpreters or the virtual machines met in the early years of microprocessors, can lead to similar benefits in terms of quick availability on new architectures. The original idea was to define portable machines description in order to isolate the compiler, tools and OS from the fast evolution of real architectures. These portable machines had simple specifications so that they could effectively be implemented on a target.

In the case of nano technologies, we propose a CAD open framework that will enable the following porting flow:

- Layer 0, target technology : modeled as tiles, connectivities, logic, switching elements,
- Layer 1, gate array/FPGA : model described on the target technology,
- Layer 2, application : model described on the Layer 1 model,

This flow only makes sense if the CAD algorithms allowing to implement logic functions, to place and route, to floor plan, are easily available: L0 to L1 to implement the micro architecture, L1 to L2 to implement the application. The answer we propose is to use interpretative and generic techniques for the CAD tools.

This paper focuses on implementing a FPGA on a nano-tube technology called NASIC. This implementation is a tiling of cells that own logic and routing capabilities. Once the FPGA is implemented it can act as a L1 target to port L2 applications and to investigate costs and performances.

In the practical case of NASIC the primitives are the cell Look Up Table (LUT), the routing switch, and a register assembled together following the technology rules.

Given an available L1, an estimated time to build up an L0 is within the order of 6 man-months. As the model is parametric, it is easy to vary the L0 definition on details such as tile sizes, routing capabilities ...

II. Nanoscale FPGA

A. Nanoscale FPGA in NASIC

1) Nanodevices: The most mature among all known-nanoelectronic devices are the carbon nano tubes [1], [2] and silicon nano wires [3]. A large part of research in nano electronics entirely focuses on these two technologies due to their CMOS similarity. These filiform technologies are synthesized with few nanometer in diameter and micrometer in length. they can act as the metallic wires, semiconductors or insulating materials [4], [5], [6] and are used to fabricate mechanic switches [7].

2) NASICs technology: The *Nanoscale Application-Specific Integrated Circuits* (NASICs) concept and architectures have been developed at UMAss by Moritz [8]. NASICs rely on nano-arrays of orthogonally crossed nanowires, surrounded by microwires that are used to carry on the control signals.

The NASIC circuits operate based on a temporary storage of data on a nanowire, each nanowire having a transistor at both ends connected to a clock phase. The figure 1 shows a simple realization of the AND function with such a technology. The grid is built out of two orthogonal plans of nanowires with different doping: the red nanowires are p-doped and the blue nanowires are n-doped. Each junction of two nanowires can contain active nanodevices (n-FET or p-FET transistors) or be detached. In this design, signals are expressed in both their original form and their complementary form.

The logic implementation is based on the PLA model, but as the complementary signals must be generated, additional lines appear (in fig. 1, only the bottom line is required for a classical implementation). The product terms of the function are realized with the p-doped nanodevices (left part of the figure) and the sum of the product terms are coupled with the n-doped nanodevices (right part of the figure).

3) NASICs circuit and design rules: The NASIC approach has been demonstrated through implementing a *WIre Streaming Processor* (WISP) [8]. WISP is a simple but complete

  ISBN :978-2-35500-003-4

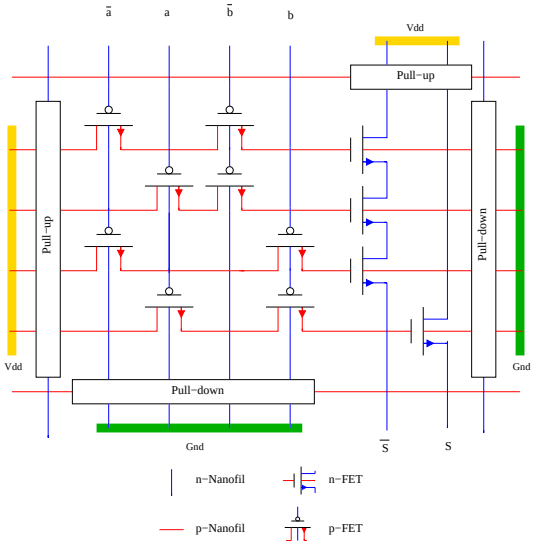

Fig. 1. 1-bit AND gate realized with nanowires FETs

design exercising the principles of NASIC design; the processor is made up of a 5-stage pipeline (fetch, decode, register file, execute and write back) supporting a simple ISA (nop, movi, mov, add, mul).

Some stages among the five seem critical as they highlight key strategies to preserve the density advantage of the technology despite implementation constraints. *Decode* and *Execute* stages are basic computing elements; *Fetch* supports the program counter by providing a local state; *Register File* demonstrates the local storage mechanism (that can be seen as simple RAM implementation); the *Write Back* stage brings the need for a U turn in the data propagation flow.

In addition to offering a demonstrator of a computing architecture based on nanodevices, WISP has mainly been a test-bed for different strategies to be applied when designing computing architectures, that can be wide spread reused.

4) Layered adapter: A preliminary work has targeted the definition of a virtual FPGA architecture [9], coupled with some exploration tools. The implementation flow of an application follows bottom-up approach i.e. a CMOS technology, programming support (physical FPGAs) and a computation model (virtual FPGAs) as shown in the figure 2.

The virtual FPGA concept favors portability of applications from a FPGA family to another at the expense of some performances loss, hence some domain-specific functionalities are also integrated into the virtual FPGAs in order to minimize these costs.

The know-how developed in this scope can benefit to addressing new target technologies such as the NASICs, assuming a layered scheme can fit to. In particular, realizing an FPGA (computation model) based on a programming support (NASIC tiles) implemented on a technology (grid of carbon nano tubes or silicon nanowires) conforms to the proposed layered approach.

5) Reconfigurable architectures: Reconfigurable architectures can be seen as a compute and configuration plans stacking up. The configuration plan stores the behavior of the architecture (switch activation, logic functions, etc...) whereas the compute plan implements the application. Hence, migrating from a circuit to a reconfigurable circuit requires to implement configuration storage mechanisms. These directly benefit from the WISP Register File design.

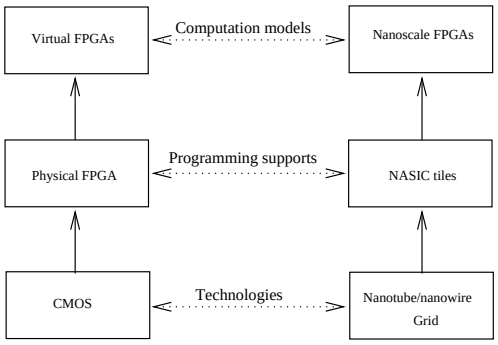

Fig. 2. Computation models in the reconfigurable Area

B. NFPGA architecture

The FPGA (L1 architecture) acting as a demonstrator is wilfully simple but still owns parametric characteristic to favor prospection. The FPGA exhibits regular tiling of cells. The cells are made up from a short list of basic elements. Hence, designing a FPGA relies on design rules for elements composition and on a circuit level description of these elements.

a) Architecture composition: The L1 tile is made up of a LUT and neighbor-to-neighbor directional routing resources.

As illustrated in the figure 3, the schematic view of the NFPGA cell consists of two parts: a LUT which regroups a decoder and RAM memories, and routing representing the input-output multiplexers.

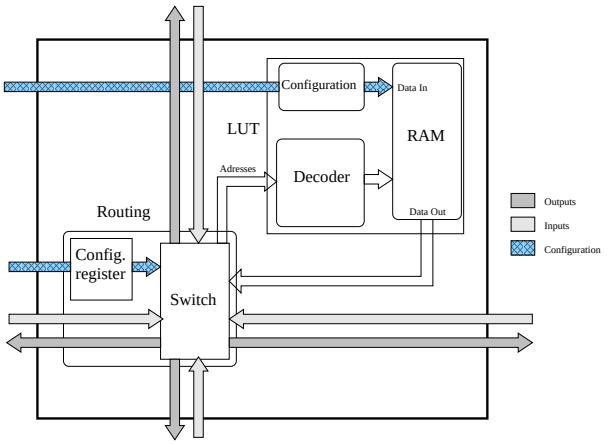

Fig. 3. Schematic of the nano scale FPGA cell. All internal connections between the modules are directional.

b) Basic elements:

- Decoder

  The figure 4 represents the address decoder layout with 4 input bits hence 8 nanowires (the signals a_i and their

complementaries). This figure also refers to figure 1, with a by 90 counter clockwise rotation: black points are p-FET, white ones are n-FET.
It contains 16 nanowires as outputs (the signals s_i and their complementaries, 9 are symbolized in the diagram). The lower part (the black points) shows the various address combinations to select the output line and the higher part makes it possible to activate only one output line.
The inputs of the address decoder comes from the compute plan.

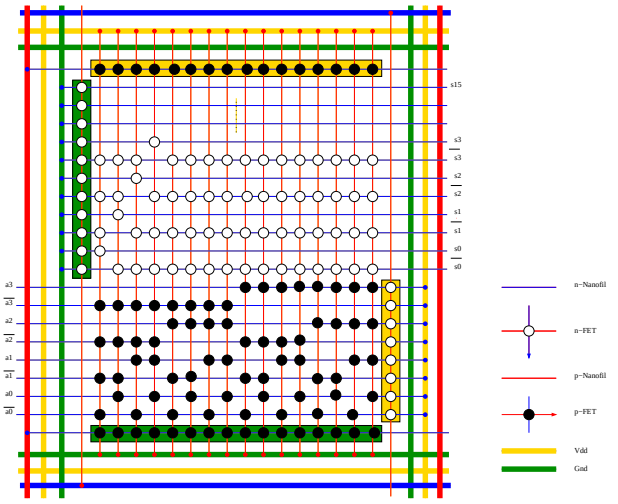

Fig. 4. Address decoder.

- Look-Up Table
  A LUT (figure 5) contains 2^N inputs coming from the address decoder and $2 * K$ bits of data to be stored. The LUT outputs $2 * K$ bits. The LUT inputs are in the configuration plan whereas the LUT's outputs are back in the compute plan.

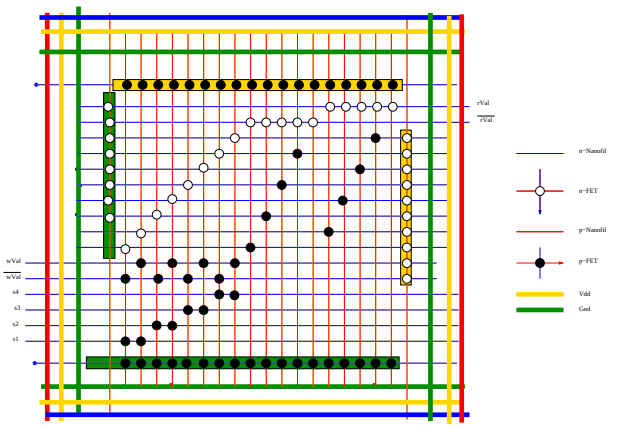

Fig. 5. LUT layout.

- Routing configuration
  The programmable interconnect relies on multiplexers. The routing configuration drives these multiplexers behavior (figure 6) by providing some signals to the switch block. Note that all of the outputs must be concurrently read in this case (2×2 signals per each of the 4 multiplexers) by contrast with the LUT's outputs that can be seen as picking up K signals out of 2^N.

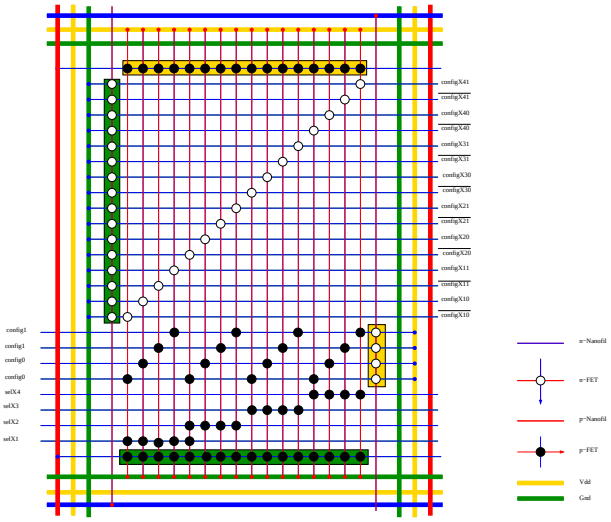

Fig. 6. Routing configuration layout.

- Switch block
  The figure 7 shows a 4-to-1 multiplexer. It takes as inputs the signals F, North, South, East and their complementaries (compute plan), gives the signal Out and its complementary. The inputs a_i and their complementary signals are the address bits coming from the previous routing configuration stage (configuration plan) which select the 4 input signals to be switched towards the output. The 4 white spots in the bottom of the grid are used to direct the North and South signals in the proper computing direction. The top part represents all possible combinations of the multiplexer output.

III. Tools

A. L0 to L1 model

1) L0 model: The L0 model is an extension of the Madeo framework [10]. The Madeo framework originally applied to FPGAs, with technical realizations over JBits/Virtex or some prospective architectures such as LPPGA [11]. Hardware elements are described with their geometric properties and tiling. This description relies on the Madeo HDL, as shown in table I.

The Madeo framework includes a set of back end tools that apply to any L0 assuming it can be described using Madeo HDL : placer router, editor, user interfaces, etc...Hence by enforcing new technologies fit within the Madeo HDL/model framework (see table II), the back end is compatible.

2) L1 model: The L1 model describes the NFPGA as an application to be implemented over the L0 level. L1 appears as a regular tiling of modules. the modules appear as a graph of logic descriptions. As an example, the nano grid f of table

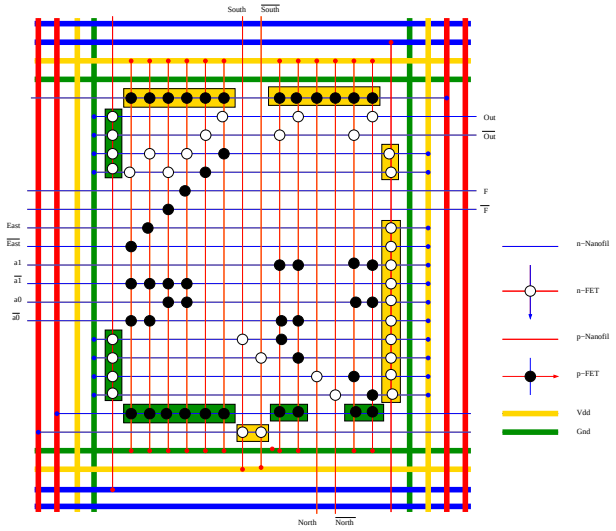

Fig. 7. Switch connection points.

```
( (COMPOSITE
    ( ( (FUNCTION
        (INPUTS  x1 x2 x3)
        (OUTPUTS  out_f))  "END of FUNCTION"
      NAMED function) " end of  NAMED  "

      ( (MULTIPLEXER
        (INPUTS  north south east out_f)
        (OUTPUT  out_east  )) "END of MULTIPLEXER"
      NAMED mux_east) " end of  NAMED  "

[...] "mux_north, mux_west, mux_south"

      ( (MULTIPLEXER
        (INPUTS  north south east west)
        (OUTPUT  out  )) "END of MULTIPLEXER"
      NAMED mux_x1) " end of  NAMED  "
[...] "mux_x2, mux_x3, mux_x4"

    ) "END OF ELEMENTS"
    (CONNECTION
      SRC mux_x1 PIN out
      DEST function PIN x1
[...] "mux to LUT"

      SRC function PIN out_f
      DEST mux_east PIN out_f
[...] "LUT to mux"

    ) "END OF CONNECTION") "END OF COMPOSITE"
PRODUCE NFPGATile) " END of  PRODUCE  "
```

TABLE I
PARTIAL L0 DESCRIPTION OF NFPGA

II can implement the function $function$ of table I, provided it appears as a RTL description (table III). From a practical point of view, the NFPGA is represented by a 2-dimension matrix where the cells contain, each one, a K-LUT and 4xN 4-to-1 multiplexers, where N is the width of the channels. The structure model of the NPGFA cell is extensible due to the variability of both the channel width and both the LUT size. The cell complexity strongly depends on these two parameters.

```
(
 (
  (
   (
    (NANOGRID 36 36) "END of NANOGRID"
    REPRESENTATION
    (DEFAULTCOLOR gray ) "END of DEFAULTCOLOR"
    (COLOR black ) "END of COLOR"
    (TEXT 60 40 'function name') "END of TEXT"
   ) "END of REPRESENTATION"
  NAMED f   ) " end of  NAMED  "
 PRODUCE NanoGrid   ) " END of  PRODUCE  "
CATEGORY ENS07 ) "END of CATEGORY"
```

TABLE II
PARTIAL L0 DESCRIPTION OF NASICS (NANOARRAY)

```
.i 22
.o 2
-------0000 01
------0-001 01
-----0--010 01
----0---011 01
---0----100 01
--0-----101 01
-0------110 01
0-------111 01
-------1000 10
------1-001 10
-----1--010 10
----1---011 10
---1----100 10
--1-----101 10
-1------110 10
1-------111 10
.e
```

TABLE III
A 3 INPUTS LUT: THE LAST THREE BITS ENCODE THE ADDRESS, THE OUTPUT AND ITS COMPLEMENTARY SIGNAL ARE BOTH GENERATED.

The structure of the multiplexers is simplified by the use of the disjoint routing.

We need to consider a regular tiling mechanism for the matrix, agglomerate structuration mechanism for the cell and the basic elements. In each cell, there are N output multiplexers for feeding a LUT, K input multiplexers and a K-LUT.

B. Synthesis

Using the layouts of the FPGA-basic elements designed in the NASIC technology, the NASIC implementation of the aforementioned parts can be realized by combining the topologies of the elements composing each part according to the Moritz-design rules.

1) Basic element synthesis: The first version of the NFPGA synthesis was focused on the synthesis at the first level of the structuring hierarchy (Cell level). To simplify the cell synthesis , the implementation of this one is divided into two implementations: the routing implementation (input and output multiplexers) and the logic implementation (LUT). This technique permit to regard th

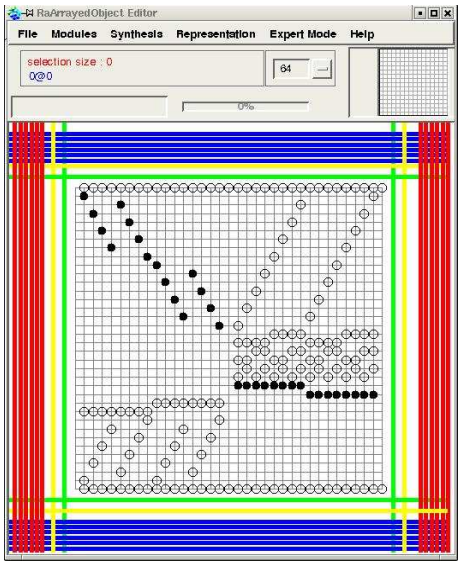

Fig. 8. The L1 layout of a 3-LUT over a Nasic L0

- Part A and A1 : The writing of configuration proceeds as follows: first, part A takes as inputs either the configuration MSB or the LSB (controlled by A1). This design technique serializes the configuration while minimizing the diagonal effect.
- Part B : The configuration is stored on the nanowires and taken as input in part C.
- Part C : The stored values are read and sent to D.
- Part D and D1 : The LUT inputs are the address. Part D1 extracts the output signal of the LUT and its complementary.

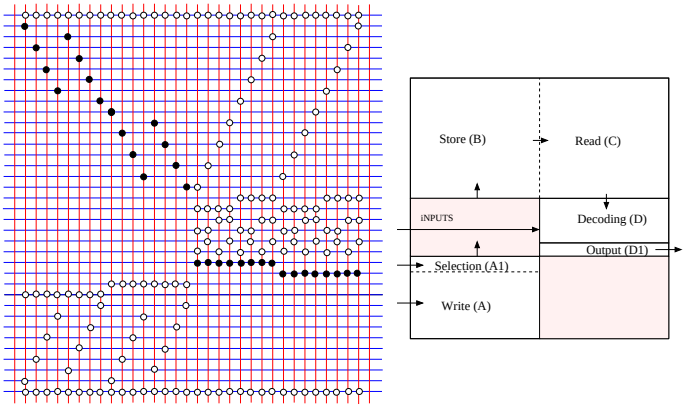

Fig. 9. The NASIC grid of the 3-input LUT.The left diagram illustrates the cutting of the grid in several basic units.

2) Floorplanning: Floorplanning consists in minimizing the global design area by applying some transformation over an assembly of modules (rotation, translation, swap, etc…). This general situation must be specialized in a NASIC scope as both internal floorplanning (see figure 9) and external floorplanning (WISP-0 implementation illustrates such a floorplanning in [12]) are required. However, internal floorplanning can be archived when placing elements (by allocating a nanowire to only one signal, floorplanning shows a diagonal effect). External floorplanning relies on Transitive Closure Graphs (TCG) to ensure non slicing floorplans when performing annealing schedule [13].

IV. Conclusion

Among other challenges the emerging nanodevices have to face, is the lack of stable computing support that prevents from reaching mass market. On the other side, reconfigurable architectures are nowadays widespreadly used due to the ease of tailoring versions to meet application-specific requirements while preserving the cost low by cross recouping them over many applicative fields.

This paper brings a solution to implement, at low cost, well known regular FPGA architectures on top of emerging devices. The key idea behind this work is to offer to application designers a stable layer on top of which to work. This approach has already proved to be highly efficient during the early days of microprocessors. We argue transposing it to nanodevices will be as fruitful.

Acknowledgment

Thanks to Damien Picard for his contribution to the automation of the L1 to L0 translation.

References

[1] C. T. White, D. H. Robertson, and J. W. Mintmire, “Helical and rotational symmetry of nanoscale graphitic tubules,” *Physic review*, vol. 47, no. 9, pp. 5485–5488, March 1 1993.
[2] P.-W. Chiu, “Towards carbon nanotube-based molecular electronics,” Ph.D. dissertation, Walker Schottky Institut, Germany, July 2003.
[3] Y. Cui, X. Duan, J. Hu, and C. M. Lieber, “Doping and electrical transport in silicon nanowires,” *Physical chemistry*, vol. 104, no. 22, pp. 5213–5216, June 2000.
[4] Mintmire and al, “Are fullerene tubes metallic?” *Physical review letters*, vol. 68, pp. 631–634, 1992.
[5] Saito and al, “Electronic structure of graphene tubules based-on c_{60},” *Physical review*, vol. B, no. 46, pp. 1804–1811, 1992.
[6] C. Dekker, “Carbon nanotubes as molecular quantum wires,” *Physics today*, vol. 52, no. 5, pp. 22–28, May 1999.
[7] T. Kueckes, K. Kim, E. Joselevich, G. Tseng, C. Cheung, and C. Lieber, “Carbon nanotube based nonvolatile random access memory for molecular computing,” *science*, vol. 289, pp. 94–97, 2000.
[8] C. A. Moritz and T. Wang, “Latching on the wire and pipelining in nanoscale designs,” in *third Workshop on A non-Silicon Computation*. Allemagne: ISCA, june 2004, pp. 39–45.
[9] L. Lagadec, D. Lavenier, E. Fabiani, and B. Pottier, “Placing, routing and editing virtuals fpgas,” *Computer Science*, vol. 2147, pp. 357–366, 2001.
[10] L. Lagadec, “Abstraction and modélisation et outils de CAO pour les architectures reconfigurables,” Ph.D. dissertation, Université de Rennes 1, 2000.
[11] G. Varghese, H. Zhang, and J. M. Rabaey, “The design of a low energy fpga,” in *ISLPED*, 1999, pp. 188–193.
[12] C. A. Moritz, T. Wang, M. Ben-Naser, and Y. Guo, “Wire-streaming processor on 2-d naniwires fabrics,” in *Nanotech*. NSTI, 2005.
[13] J.-M. Lin and Y.-W. Chang, “TCG: A transitive closure graph-based representation for non-slicing floorplans,” in *Design Automation Conference*, 2001, pp. 764–769. [Online]. Available: citeseer.ist.psu.edu/lin01tcg.html