Emerging Technologies and Nanoscale Computing Fabrics

Ian O'connor, Junchen Liu, Jabeur Kotb, Nataliya Yakymets, Renaud Daviot, David Navarro, Pierre-Emmanuel Gaillardon, Fabien Clermidy, Maïmouna Amadou, Gabriela Nicolescu

Emerging Technologies and Nanoscale Computing Fabrics

Ian O'Connor[1a,3], Junchen Liu[1a], Jabeur Kotb[1a], Nataliya Yakymets[1a], Renaud Daviot[1b], David Navarro[1a], Pierre-Emmanuel Gaillardon[2], Fabien Clermidy[2], Maïmouna Amadou[3], Gabriela Nicolescu[3]

[1]University of Lyon, Lyon Institute of Nanotechnology UMR 5270
[a]Ecole Centrale de Lyon, 36 avenue Guy de Collongue, F-69134 Ecully cedex, France
[b]CPE Lyon, 43, boulevard du 11 novembre 1918, F-69100 Villeurbanne, France

[2]CEA – LETI – MINATEC
17, rue des Martyrs, F-38054 Grenoble, France

[3]École Polytechnique de Montréal, Computer Science Department
Montréal, QC, H3C 3J7, Canada

ian.oconnor@ec-lyon.fr

Abstract. This chapter describes a reconfigurable computing architecture based on clusters of regular matrices of fine-grain dynamically reconfigurable cells using double-gate carbon nanotube field effect transistors (DG-CNTFET), which exhibit ambivalence (p-type or n-type behaviour depending on the back-gate voltage). Hierarchical function mapping methods suitable for the cluster of matrices structure have been devised, and various benchmark circuits mapped to the architecture. This work shows how circuit and architecture designers can work with emerging technology concepts to examine its suitability for use in computing platforms.

Introduction

Computing power recently broke the petaflop/s barrier within a single machine and is expected to continue to scale to exacomputing over the next decade [1] (fig. 1). The main hardware vectors behind this spectacular evolution have been a) increase in intrinsic chip functionality through scaling and b) massive parallelism and increasingly efficient interconnect topologies. While scaling has now for a few years been mainly limited to improving the number of functions per chip rather than clock speed, other factors (such as cost, reliability, static power) render necessary the exploration of other technologies and computing paradigms to pursue the quest for performance.

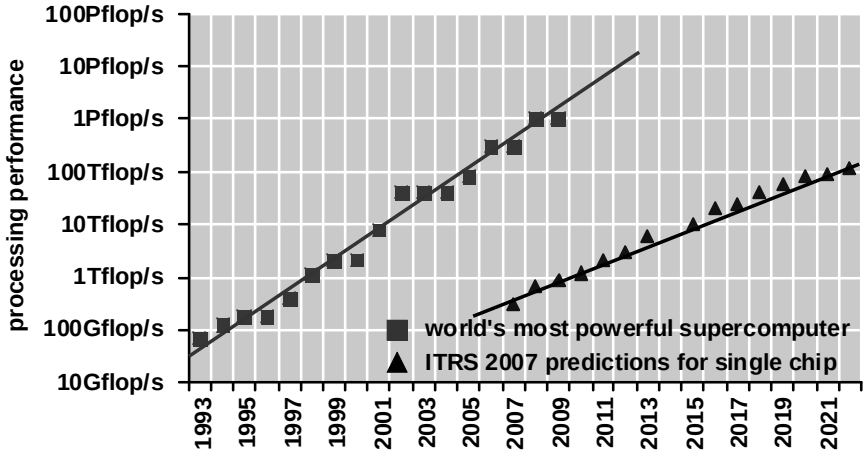


Fig. 1. Best observed processing performance for a single machine [1] (a) and predictions for processing performance in a single chip [2] (b)

Indeed, it is widely recognized that transistor scaling, as a vector for the pursuit of performance levels predicted by Moore's Law and required by future applications, will not last through the next decade. Alternatives must be found, be they at the architectural level (e.g. exploring multiple core architectures) or at the device level (heterogeneous or nanoelectronic devices). In this context, the emergence of new research devices offers the opportunity to provide novel logic building blocks and to elaborate non-conventional techniques for digital design. Ultimately it will be possible to reconsider the paradigms of computing architectures to achieve orders of magnitude improvements in the conventional figure of merit (MIPS / volume*power). In this way, future computing platforms are likely to cover broad ranges of applications, from traditional number-crunching (counting and calculating) to emerging neuromorphic (recognizing and reasoning). These very different classes of algorithm will require suitable hardware platforms, with the additional constraints of occupying small volume and low-power.

It is also expected that the necessary structuring of the projected tens of billions of elementary, unreliable, nanometric devices to achieve the computing capacities necessary for future software applications will lead to the emergence of reconfigurable platforms as the principal computing fabric before the end of the next decade. The reconfigurable approach allows volume manufacturing and reduces the impact of the evolution of mask costs, projected to move above the $10M mark in 2010. It can also efficiently cover a broad range of applications while exceeding performance levels of programmable systems, and couples naturally to fault-tolerant design techniques for robust architectures. Reliability is clearly an increasingly important issue given the lack of reliability at individual device level: leading to the rise of self-x (self-configuration, self-repair …) at architectural and/or software levels

However, the organization of such reconfigurable cells in a system is uncertain – integration density and switchbox overhead concerns are a growing issue. These point

to the rising probability of fixed interconnect topologies between individual cells organized into clusters, and the use of switchboxes or network approaches only between clusters of cells. In parallel, the unreliability of individual devices will lead to a loss of accessible functions in certain cells. This can be circumvented by reformulating cluster configurations based on the incomplete set of identified operators. In this context, the development of methods capable of mapping complex functions onto clusters of reconfigurable cells with incomplete sets of operators is a key milestone to exploiting the full potential of future reconfigurable systems.

In addition, recent technological breakthroughs have led to the proposal of area- and power-efficient reconfigurable cells based on emerging devices such as double-gate carbon nanotube transistors (CNTFET) with incomplete operator sets [3]. CNTFETs have attracted much attention in recent years, and benchmark figures against state-of-the-art planar and non-planar silicon logic transistors are favourable. They have shown in particular that the high mobility, achievable current density, theoretical transition frequency and Ion/Ioff ratio place CNTFETs among the most promising nanodevices in line to succeed the MOS transistor from the standpoint of their integration into future nanoelectronic systems on chip [4]. Some work has been carried out to explore the use of the unique properties of multiple diameter and ambipiolar CNTFETS with respect to CMOS for new computing paradigms ([5], [6]). The emergence of double gate devices, with four accessible terminals, also opens the way to solutions specifically exploiting the additional terminal for reconfigurability purposes. In the case of the double gate CNTFET [7], completely new prospects for reconfigurability are possible due to its ambivalent (n- and p-type) behaviour. Using this property, logic cells can be built that offer fine-grain reconfigurability not available with MOSFET technology, at comparable or better speed and power figures, and improving over current reconfigurable systems in terms of the number of devices used to realize a single function.

These considerations have recently led to the emergence of the concept of nanofabrics [8], or nanoscale computing fabrics. A nanoFabric can be defined as an array of connected nanoscale logic blocks (nanoBlocks), where a nanoBlock is a circuit block containing programmable devices to compute boolean logic functions and means to route data. From a technological point of view, such systems are usually based on a hybrid approach (on a silicon die, or with CMOS compatibility). They are a combination of a bottom-up structure, using chemical self-assembly for dense and regular arrangement of elements, and a top-down structure, using conventional process options for interconnect or for computation. In this work, we do not consider memory issues but it is clear that memory integration is also paramount.

This chapter begins by describing the structure and properties of a DG-CNTFET based dynamically reconfigurable logic cell to be used in a computing nanofabric. We then explore ways in which this cell can be used to form a regular and dense matrix structure, as well as a method to map function graphs to such matrices, and clusters of matrices, of reconfigurable cells for on-the fly and partial reprogrammability. The method is applied for various benchmarks in order to evaluate the capability of the architecture to execute complex functions. We finally conclude with a discussion on the insights of this work, and future challenges.

Carbon-based nanofabrics

For high-performance FETs, short gate lengths and high channel mobility are required. Since nanotubes typically exhibit very small diameters (allowing excellent gate control) without suffering from mobility degradation, they are promising candidates to overcome the limitations of nanometric silicon devices. Fig. 2 shows the structure of a novel DG-CNTFET [7], fabricated with an aluminium front gate placed under the nanotube between the contacts of the source and the drain and controlling the electrostatics and switching of the nanotube bulk channel in region B. The Schottky barriers (SB) at the nanotube/metal contacts are controlled by the silicon back gate (substrate), which also prevents the electrostatics in region A from being influenced by the front gate. The SBs at the contacts are not affected by the front gate voltages.

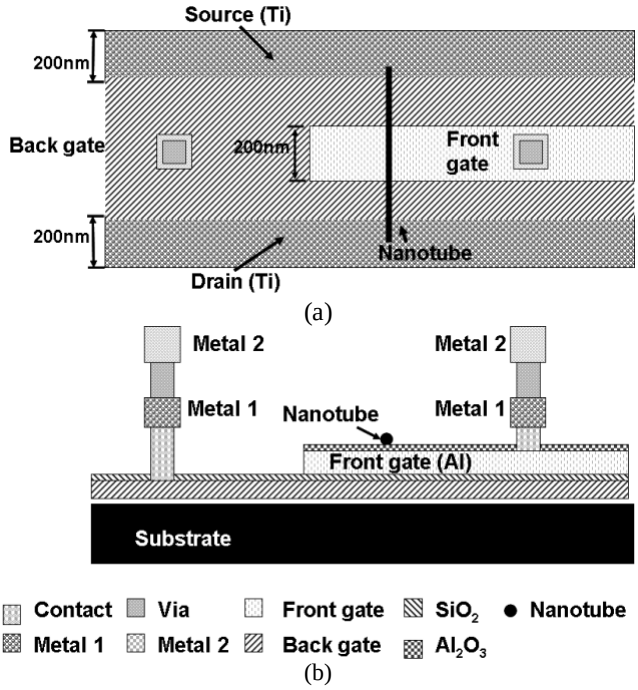


Fig. 2. Double-Gate CNTFET structure [7] (a) top view (b) cross-sectional view

The behaviour of this DG-CNTFET device is strongly dependent on the potential of the silicon back gate, which we call $V_{gs\text{-}bg}$:

- when $V_{gs\text{-}bg}$ is sufficiently negative (some hundreds of mV), the device functions like a p-type FET with a negative threshold voltage.
- when $V_{gs\text{-}bg}$ is sufficiently positive (some hundreds of mV), the device functions like an n-type FET with a positive threshold voltage;

- when $V_{gs\text{-}bg}$ is floating, the sub-bands with the contacts are not affected by the bias of the front gate, and the device is in the off state with a very weak current (I_{off}<100fA).

The impact of the back gate voltage polarity on the transistor channel transport characteristics opens up new opportunities for using CNTFETs in logic circuits. We have built a reconfigurable logic block which can be configured to any one of fourteen basic binary operation modes. This functionality is impossible to achieve in CMOS technology without resorting to far more complex circuitry (and therefore silicon real estate and system power) than the cell structure described here. The polarity (n-type or p-type) of each transistor is controlled by the back-gate bias voltage values. The dynamically reconfigurable logic cell (DRLC_7T) is shown in fig. 3; while tab. 1 describes the 3-input configuration, corresponding basic binary logic functions and power figures for operation at 4GHz and 250MHz extracted from transient simulations using a Verilog-A model adapted from [3] and parasitic capacitances extracted from layout estimations.

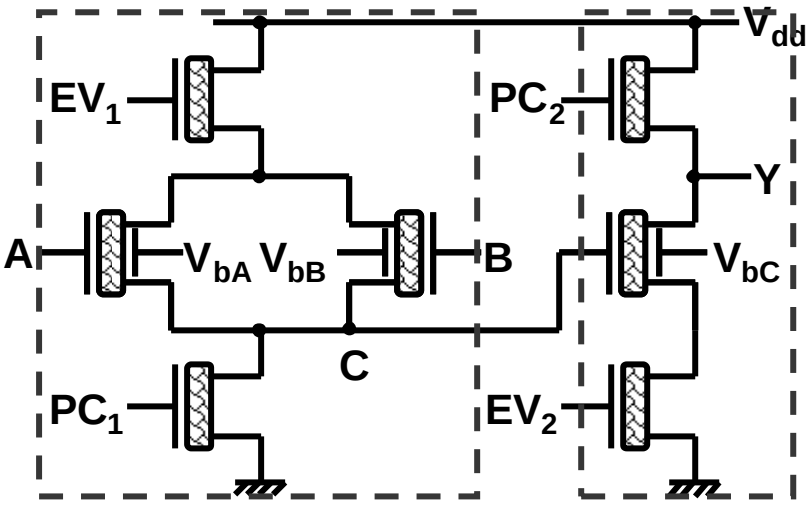


Fig. 3. Dynamically reconfigurable logic cell (DRLC_7T) transistor-level schematic

V_{bA}	V_{bB}	V_{bC}	Y	P_{tot}@4GHz (nW)	P_{tot}@250MHz (nW)
+V	+V	+V	A↓B	1.87	1.076
+V	+V	-V	A∨B	1.85	0.99
+V	0	+V	¬A	1.83	0.84
+V	0	-V	A	1.81	0.82
-V	-V	+V	A∧B	1.84	0.9
-V	-V	-V	A↑B	1.82	0.814
+V	-V	+V	B⇏A	1.86	1.05

+V	-V	-V	B→A	1.84	0.96
0	+V	+V	¬B	1.82	0.8
0	+V	-V	B	1.79	0.79
0	0	0	1	1.12	0.04
0	0	-V	0	1.82	0.2
-V	+V	+V	A⊼B	1.84	1.03
-V	+V	-V	A→B	1.82	0.95

Table 1. 3-input configurations for reconfigurable cell with 3 logic levels (+V, 0, -V) and corresponding 14 basic binary logic functions

DRLC_7T is made up of 7 CNTFETs arranged in two logic stages: the first stage performs an elementary logical operation and the second stage works either in follower or inverter mode.

- A and B are boolean data inputs (voltages at A and B vary between 0V and 1V);
- V_{bA}, V_{bB}, V_{bC} are control inputs which configure the circuit according to tab. 1 (control bias voltages may take one of three values at -1V, 0V and 1V);
- PC_1, PC_2 (pre-charge) and EV_1, EV_2 (evaluation) are four non-overlapping clocking inputs with pre-charge and evaluation periods as in classical CMOS dynamic logic gates;
- Y is the circuit output.

We can see that V_c is evaluated between EV_1 (evaluation of the first logical stage) and the next PC_1 (pre-charge of the first logical stage) according to the value of inputs A and B; and Y is evaluated and maintained between EV_2 (evaluation of the second logical stage) and the next PC_2 (pre-charge of the second logical stage). This clocking scheme is illustrated in fig. 4.

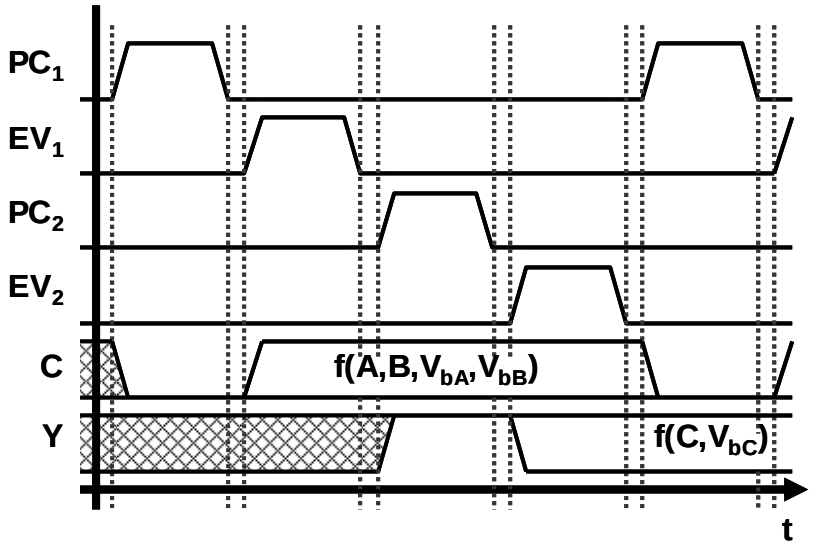


Fig. 4. Two-stage dynamic logic clock signal scheme

An example illustrates how this logic gate works. When $V_{bA}=V_{bB}=V_{bC}=1V$, CNTFETs T_{c1}, T_{c2} and T_{c3} (shown in fig. 3) are all configured as n-type FETs, as indicated in the previous section. When PC_1 is enabled, the first stage is pre-charged, and the voltage of the internal node C (V_c) is discharged to 0V. If for example either of the data inputs A or B=logic "1", then when EV_1 is enabled, the first layer evaluates its output such that the internal node C is set to logic "1". Then PC_2 is enabled (pre-charge of the second stage), and the output Y is charged to logic "1"; and when EV_2 is enabled, the output is evaluated and Y is evaluated to logic "0". In fact in this configuration, the only situation where C is not set to logic "1" and Y therefore evaluates to logic "1" (since Tc_3 is off) is when both A and B=logic "0". This shows that for $V_{bA}=V_{bB}=V_{bC}=1V$, DRLC_7T is configured as a NOR operator, as specified in tab. 1. Simulation results of DRLC_7T in this configuration are shown in the left half of fig. 5 (up to 8ns) at 500MHz operation. After this point, V_{bA},V_{bB} and V_{bC} change to -1V, such that CNTFETs T_{c1}, T_{c2} and T_{c3} are all configured as p-type FETs. When PC_1 is enabled, the first stage is pre-charged, and the voltage of the internal node C (V_c) is discharged to 0V. If for example either of the data inputs A or B=logic "0", then when EV_1 is enabled, the first layer evaluates its output such that the internal node C is set to logic "1". Then PC_2 is enabled (pre-charge of the second stage), and the output Y is charged to logic "1"; and when EV_2 is enabled, the output is evaluated and Y is evaluated at logic "1". The only situation here where C is not set to logic "1" and Y therefore evaluates to logic "0" (since Tc_3 is on) is when both A and B=logic "1". This shows that for $V_{bA}=V_{bB}=V_{bC}=-1V$, DRLC_7T is configured as a NAND operator.

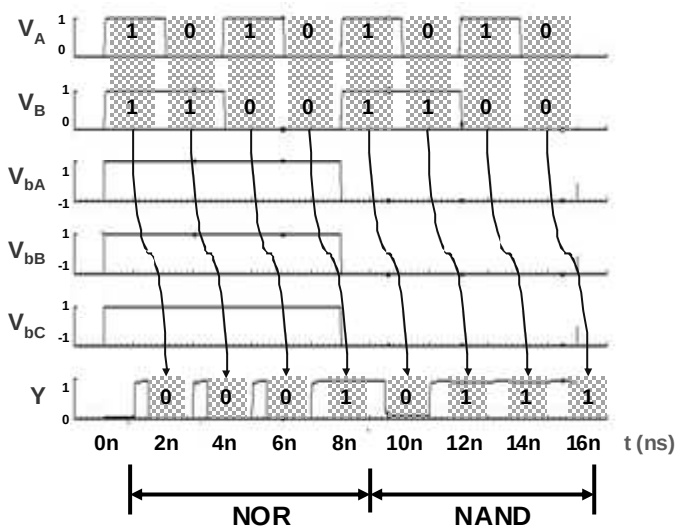


Fig. 5. Simulation results of dynamic reconfiguration of reconfigurable cell DRLC_7T from NOR operator to NAND operator

It is thus clear that this gate can realize several functions and can be dynamically reconfigured during the calculation.

Tab. 1 also gives the power consumption when DRLC_7T (working with 2-logic-stage control bias voltage) operates at 250MHz and 4GHz.

It should be noted that as this is prospective work, no technology as yet exists to build this circuit, although a single DG-CNTFET has been fabricated and characterized [7]. Further, significant technological advances have been made recently to achieve 95-98% horizontally aligned semiconducting CNTs [9] and, separately, hybrid integration with CMOS [10]. This enables us to envisage systems using "substrates" of many aligned semiconducting CNTs with conventional metallization and lithography techniques creating interconnections. In terms of device design, much work has focused on improving drive current (and therefore maximum frequency and insensitivity to noise) and reliability by using an array of parallel single-walled carbon nanotubes as multiple channels in a single transistor with good directional and spatial control [11]. This device can pass currents of up to 1.5mA and has achieved a record current gain cutoff frequency of 8GHz. These performances are still dominated by parasitics but recent advances project that this device should reach a current gain cutoff frequency of 31GHz. Double-gate transistors using the same principle for the channel should not pose any technological obstacle.

Here we have given only one example of the family of dynamically reconfigurable logic cells. By changing the number of the transistors we have developed 3 other logic cells which can realize different logic function sets [3].

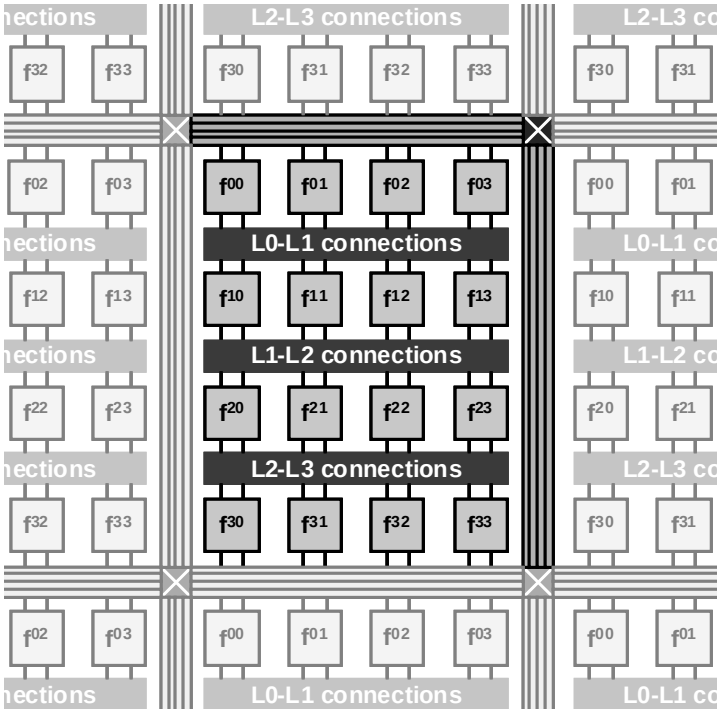
Clusters of cell-matrices architecture

Such fine-grain reconfigurable gates open the way towards structures which can be configured dynamically, for on-the-fly and partial system reprogrammability. In this section, the way elementary building blocks are connected and programmed is explored to achieve increased efficiency at the application level.

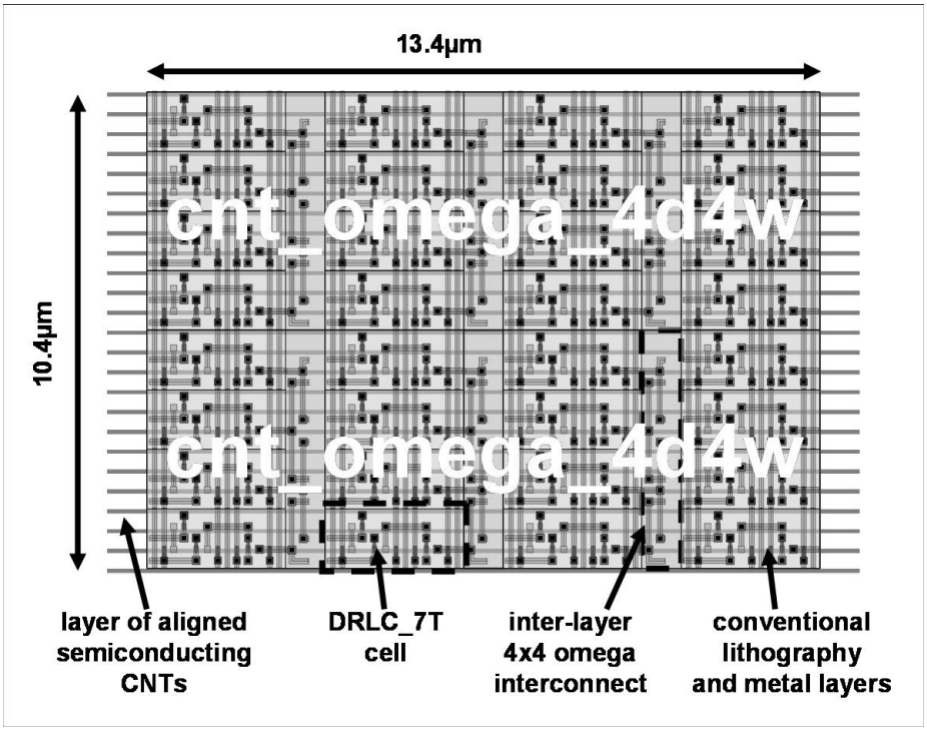
Fixed intra-matrix interconnect topology strategy

In a conventional architecture, each calculation cell would be connected to the switch box directly. However, in the case of fine-grain logic cells, this approach would lead to a loss of efficiency due to a large overhead in terms of device complexity. In the case of DRLC_7T, 7 transistors are used in the cell, while a similar number of transistors (at least 6) are used for a 1-bit switchbox. In order to avoid this overhead problem, we propose a cluster-based approach as shown in fig. 6, which consists of assembling cells in a matrix pattern, with the use of fixed intra-matrix interconnect between layers of cells. Here, the identifier f^{xy} corresponds to the configured function of the cell and the {x,y} coordinates of the cell within the matrix. Inputs A and B are shown, as is the output Y (duplicated); precharge and evaluation connections are not shown to avoid making the figure overly cumbersome. Considering the whole of this cluster set as a new coarse grain element, switchboxes

could be used for inter-matrix interconnect. It is interesting to note that matrix architectures are also particularly well-suited to CNTFET-based logic cells, since it is possible for single nanotubes to span several cells in the same column.



(a)



(b)

Fig. 6. Tile-based approach for the integration of regularly structured matrices of reconfigurable cells (a) conceptual schematic (b) layout for two tiles of 4d4w Modified-Omega topologies

For intra-matrix interconnect, and taking wiring complexity into account, we eliminate any total interconnectivity topologies at the outset. Instead, and through analogy to computer networks, we adapt incomplete interconnection sets to the matrix architecture. In fact, Multistage Interconnection Networks (MIN) are designed to interconnect layers in an efficient way and can be applied in this context.

Of course, there are many topologies or combinations, but we focus principally on 4 typical permutations [12]: Banyan (fig. 7(a)), Baseline (fig. 7(b)), Flip (fig. 7(c)) and Modified-Omega (fig. 7(d)), where the modifications to standard Omega maximize the shuffling in this topology. In computer science, MINs are used to interconnect layers of switchboxes in order to route information packets only. In this application, the main difference is that switchboxes have been removed and replaced by logic cells, introducing computing directly inside the network.

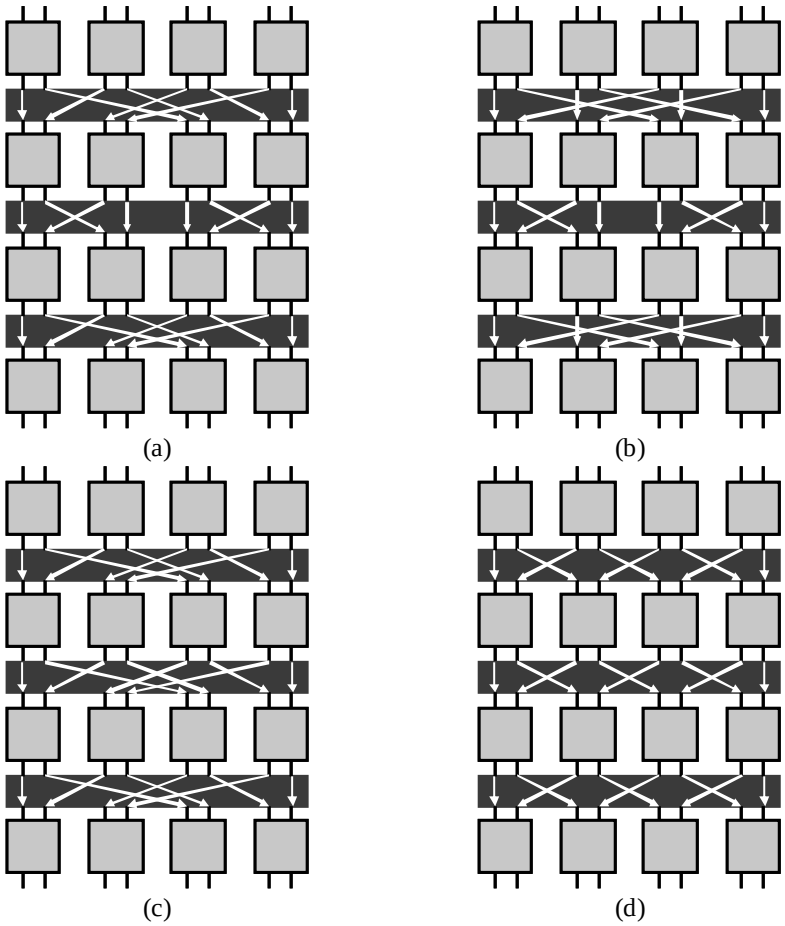


Fig. 7. Matrix of 16 reconfigurable gates with various interconnect topologies: (a) Banyan (b) Baseline (c) Flip (d) Modified-Omega

Matrix programming

It is useful to combine such novel types of nanodevice-based reconfigurable cell with the exploration of new function mapping methods in anticipation of the deployment of incomplete-operator cluster-based systems. A primary objective is to analyze the limits of such architectures when mapping a complex software application onto it. Many parameters must be considered to program the nanodevice-based architectures:

- the number of cells in matrix
- the topology of cells interconnections
- potentially, the faults present in the matrix

Moreover, several metrics have to be optimized (computation speed, area, etc). Therefore, new CAD tools meeting these requirements are mandatory in order to explore the potential of nanodevicebased architectures during the prototyping phase. One of the key issues is the automatic mapping of complex functions onto nanodevice-based architectures. Several mapping methods defined for conventional architectures have been proposed. However, these methods fail to reach the ultra-fine granularity specific to nanodevice-based architectures. Furthermore, they do not consider connectivity restrictions and dynamic reconfiguration opportunities.

While the application is quite close to logic synthesis and network routing, the fact that we have introduced computing inside the matrix means that we cannot use routing algorithms or synthesis algorithms directly. We present in this section a mapping method designed to map a logic function graph onto the architecture described above. First, we will describe the method, and then we will give an example to show how this method works.

Functional description

The described method inputs the function graph to map and the physical connectivity matrix and outputs the map of logic elements onto physical cells. The algorithms work using adjacency matrices. In such matrices, (*i*,*j*) refers to the intersection of the row *i* and column *j*. A 1 at the position (*i*,*j*) means that the point *i* is connected to the point *j*. These matrices are essential to subsequent processing steps, described in fig. 8.

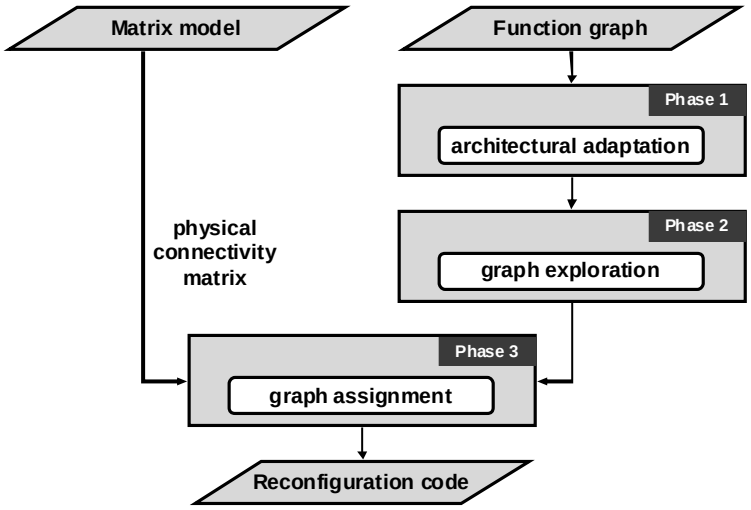


Fig. 8. Mapping method functional stream

Phase 1: Pre-processing of function graph

In a first operation, we have to adapt the logic function to our architecture. Due to the layered structure, the system is pipelined. Function graphs have to be processed by

adding necessary synchronization elements (to extend input and output paths of data), as well as removing jumps over logic layers (to conform to the physical topology). To identify logic layers in function graphs, we divide the associated adjacency matrix into small matrices C_{nm}. C_{nm} is the adjacency matrix between the points in the logic layer n and those in the logic layer m. We therefore pay particular attention to matrices where $m \neq n+1$ because the presence of any non-zero element in these three matrices identifies a connection which "jumps" at least one logic layer. Such a direct connection cannot be realized in the topology since the interconnect topology is physically fixed. The solution is to add synchronization elements (i.e. create a path instead of a jump), then repeat the process until no more connections jump logic layers.

Phase 2: Recursive exploration

The processed function graph is then analyzed in depth, meaning that for each node in the structure, child branches are identified and recursively explored.

Phase 3: Node assignment

Logic nodes are then assigned to cells. This is done according to the physical interconnections. Each layer's connections are compared to the relevant inter-layer connectivity matrix – allowing (or not) the assignment of functions to cells. Branching (i.e. the exploration of the immediately preceding alternative) is used when the arbitrary choice leads to a dead-end, and the process is repeated until all functions are assigned to cells. The algorithm of this step is shown in fig. 9.

```
Current node = First node
LOOP
  IF (children/parents of current node already placed) THEN
    IF (empty cell connected to childs or parents) THEN
      Map the node at the first empty position
      Store other solutions
      IF (Remaining node to map) THEN
        Current node = Next node
      ELSE
        MAPPING FINISHED - END LOOP
    ELSE
      IF (Previous node has a non explored solution) THEN
        Try next solution
      ELSE
        NO SOLUTIONS - END LOOP
  ELSE
    IF (free cell at the corresponding stage) THEN
      Map the node at the first empty position
      Store other solutions
      IF (Remaining node to map) THEN
        Current node = Next node
      ELSE
        MAPPING FINISHED - END LOOP
    ELSE

      NO SOLUTIONS - END LOOP
```

Fig. 9. Mapping algorithm (pseudo-code)

The algorithm has been implemented in Matlab, and executes in under 0.1s for a complete 16-node mapping operation using a standard 2GHz PC. While this approach enables the mapping of simple functions to the fixed-interconnect matrix based on the reconfigurable cells, function partitioning and merging methods will be required to map more complex functions over several matrices.

Mapping example

As an example, we consider a matrix which is 4 cells deep and 4 cells wide (4d4w) using a Banyan interconnect topology (fig. 7(a)). Individual cross-connectivity matrices X_{nm} (X_{01}, X_{12} and X_{23}) between logic cell stages of depth n to m are shown in eq. 1.

$$X_{01} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, X_{12} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}, X_{23} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \textbf{(1)}$$

The function to map is represented by a graph, generated by a random graph generator for test purposes. During the graph adaptation step, we pay particular attention to the matrices C_{02}, C_{03} and C_{13} containing 1's. As mentioned previously, such matrices represent connections which “jump” at least one logic layer, and are impossible to realize. In this example, the connections between points (2, 6), (5, 8) and (4, 9) are the three connections to be adjusted.

The graph exploration is then launched and we obtain the following sequence (p* represents synchronization nodes):

p_1-p_4-p*-p_9-p_5-p_7-p_3-p*-p*-p*-p_8-p_2-p*-p_6-p*

Finally, the graph assignment is performed. In the example, the first point p_1 is assigned to the cell f^{00}. According to the path defined in the previous step, p_4 is the next point to assign to a cell in the matrix. Since f^{00} is physically connected to f^{10} and f^{12}, the cell with lower y-index (here f^{10}) is arbitrarily chosen for p_4 assignment, and the other possibility is memorized. In our example, the final programmed matrix is shown in fig. 10. In this figure, we can see the nodes of the logic function graph and the nodes added for synchronization purposes (circles with no names) correctly placed on the cell matrix.

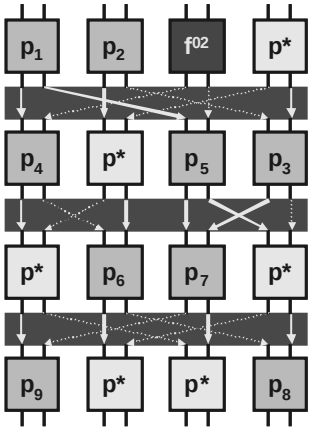


Fig. 10. Matrix after function mapping

Evaluation methodology

The aim of this part of our work was to evaluate and compare performance metrics for the 4 interconnect topologies. Our study was made on a 4d4w matrix using the previously mentioned intra-matrix interconnection topologies. 4d4w matrices have been chosen because of a good balance between complexity and simulation time. We evaluated various metrics: the success rate of mapping function graphs, the fault tolerance and the average interconnection length. We have carried out detailed analyses to compare the efficiency of the different intra-matrix interconnect topologies. We use a random graph generator to generate static sets of function graphs containing 6-16 points, in order to have fixed comparison criteria between topologies. No graphs contain isolated nodes, as here we focus on fixed interconnect layers, which are severely penalized by isolated nodes. We consider therefore these cases to be an overload issue to be solved by specific architectural customization. Each set, corresponding to a given number of points in the function graph, contains 1000 samples. Using the previously described mapping method, each function is programmed onto the 4d4w matrix using the various intramatrix interconnect topologies, ideal or faulty, and metrics are calculated. Fig. 11 summarizes the evaluation methodology and the associated parameters.

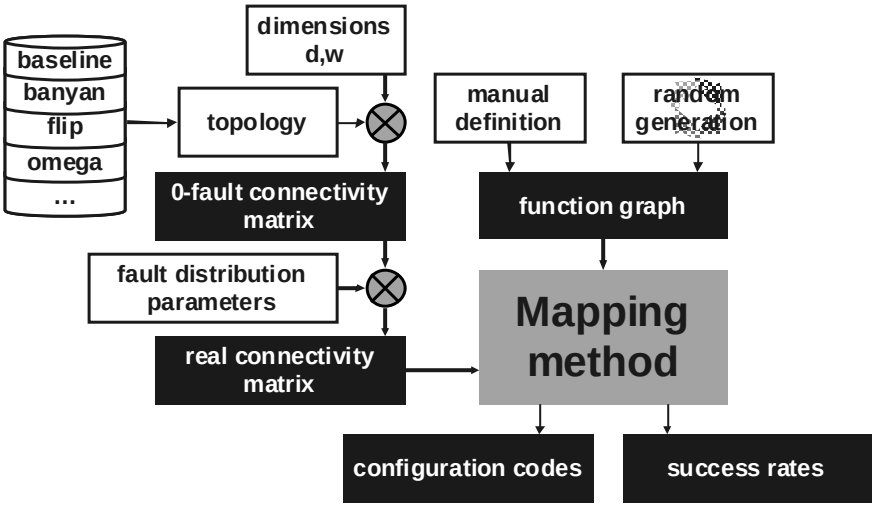


Fig. 11. Evaluation method

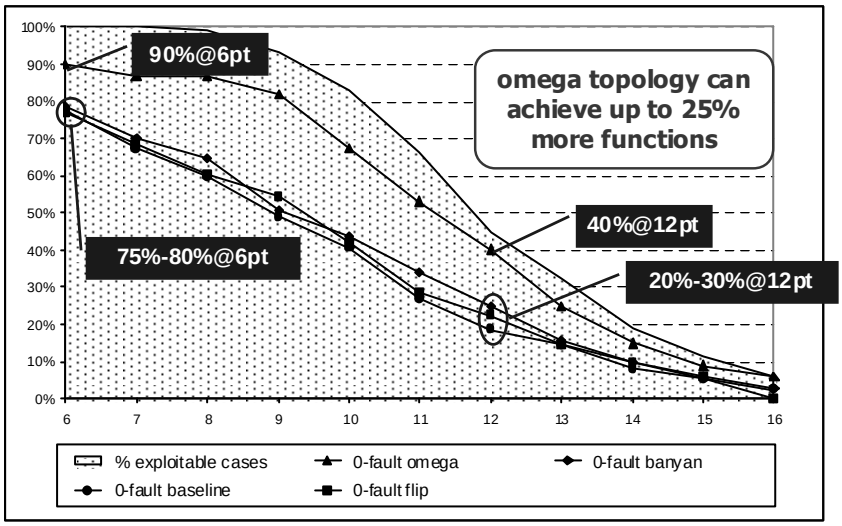


Fig. 12. Programmability success rates for Banyan, Omega, Flip and Baseline interconnect topologies within 4d4w matrices

Applying static sets to ideal interconnect topologies, we can test the ability of the matrix-topology ensemble to have complex functions mapped onto it. Considering the percentage of function graphs successfully mapped onto matrix with respect to the number of samples in the set, we obtained the success rate. Fig. 12 shows the comparison of success rates for 4d4w Banyan, Omega, Flip and Baseline topologies. For Banyan, Flip and Baseline interconnect topologies, the success rate is about 80% when the function graphs have 6 points. At 12 points, the success rate is about 25%. The difference between these two topologies is thus relatively small. However for the

Omega interconnect topology, the success rate is about 90% for 6-point function graphs and about 40% for 12-point graphs. This clearly shows that the Omega interconnect topology is more suitable for this type of matrix.

This is because this topology is has less symmetric redundancy than the other topologies and spreads calculations over cells occupying less width, which seems to correspond better to typical function graphs. In fact in the matrix, there are pairs of cells which have the same inputs. For two cells which have the same inputs, the sum of the number of functions they can achieve is 14. For two cells which do not have the same inputs, the sum of the number of functions they can implement is 14+14 = 28. In the Banyan topology for example, there are 6 pairs of cells which have the same inputs, while in the Omega topology, there are only 2. This is the main reason why the Omega topology has the potential to realize more functions than other topologies.

It is worth noticing that the use of a MIN reduces the number of mapped logic functions, compared to traditional LUT approaches. Such a problem is managed at a higher hierarchical level. For example, if a function graph cannot be mapped onto a single matrix, we can split it and map the subgraphs onto different matrices.

Cluster programming

At the cluster level, an extra layer of parameters and additional flexibility is introduced. It is possible to consider clusters of varying matrix size, as well as the execution of functions in parallel, and the dynamic (potentially cycle-level) reconfiguration of each matrix to achieve highly optimized graph execution. In order to explore these aspects, a cluster-level mapping model is proposed. This mapping model places applications onto the complete architecture composed of several matrices, such that multiple metrics are optimized. These metrics are:

- communication cost
- configuration cost
- execution time
- number of unused logical cells.

The objective of the mapping model is to optimize the placement of a complex function onto the architecture. It considers the structure of the architecture, the scalability requirement as well as the dynamic reconfiguration implying a high-level of pipelining and parallelism. It combines GA and partitioning approaches.

Model Description

The proposed model is shown in fig. 13. It takes as inputs a function graph and the architecture model and generates the reconfigurable code of the mapping as output. In general, complex functions cannot be mapped onto a single matrix. For this purpose, functions are partitioned into sub-functions. Thus, in order to map a complex function onto a cluster of matrices, two mapping levels are performed:

- Firstly, each sub-function is mapped onto a matrix using the method described in the previous section;
- Then, the dependency graph of sub-functions is also mapped onto the cluster of matrices.

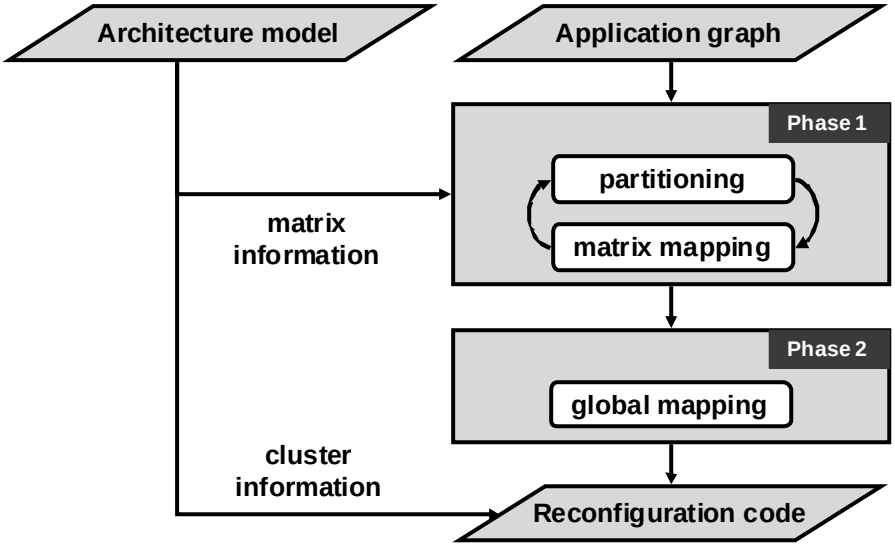


Fig. 13. Mapping model

The function to map is represented as a Direct Acyclic Graph (DAG) $G_f = (E_f, V_f)$ where V_f is a set of nodes representing logic operations, and E_f a set of direct edges dependency relations between nodes.

Phase 1: Partitioning and matrix mapping

The aim of the first step is to partition a function graph into smaller sub-graphs so that each sub-graph fits in a single matrix. For this purpose, partitioning and matrix mapping methods are processed. To be valid, a partitioning result must respect the following constraints:

- The total number of operation in the sub-graphs is the same as the one in the initial graph
- There is no cyclic dependency.

Each sub-graph obtained during partitioning is mapped onto the matrix using the matrix mapping method defined previously. An exhaustive research is performed to ensure that the best mapping is found. If the matrix mapping fails, the partitioning is modified and new matrix mappings are performed. This exploration loop, represented in fig. 13 through a feedback arrow, is performed until the sub-graphs fit in a single matrix. The result of the first step is a new graph composed of subgraphs and a set of mapping solutions associated to each subgraph.

Phase 2: Global mapping

The second phase maps the set of sub-graphs obtained during the phase 1 onto the cluster of matrices. The objective is to find a mapping and an execution order minimizing the communication cost, the configuration cost, the number of cells non-

used and the execution time. A Fast Elitist Non-Dominated Sorting Genetic Algorithm (NSGA II) [13] is used to minimize these metrics.

A mapping solution is encoded as a two part chromosome (fig. 14): the first part represents the execution order of the subgraphs; the second part gives the matrix on which sub-graph are executed. For example, in fig. 14, the set composed of subgraphs Sb0, Sb1, Sb2, Sb3 and Sb4 is executed on matrices Gm1 and Gm2. Thus, Sb1 and Sb4 are executed in parallel on matrix Gm1 respectively Gm2 at cycle 0. The execution of sub-graphs Sb0, Sb 2 and Sb 3 is pipelined on Gm2. For this purpose, Gm2 is dynamically reconfigured two times.

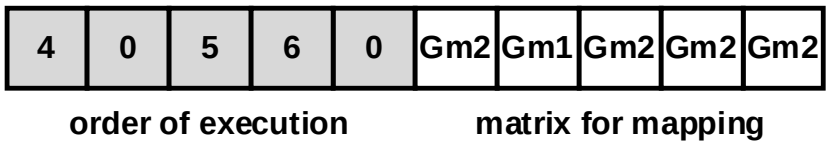


Fig. 14. Global mapping solution encoding

Experimental results

To evaluate the efficiency of the proposed approach, we used Xilinx-Virtex-4 [14] schematic designs. Each circuit is characterized by the number of operations, the number of dependencies and the depth (i.e. the length of the longest path in the circuit). The designs are modified to reach the granularity required for the logic cell (two 1-bit inputs and two 1-bit outputs). The target architecture is based on Modified-Omega clusters of 4d4w matrices. The experimental results are given in tab. 2, in terms of the number of sub-functions obtained after the partitioning phase, the number of dependencies between sub-functions, the ratio between the number of logic operations and the number of sub-functions (fill factor) and finally the number of failures for matrix mapping during partitioning.

These results depend on the following factors:

- the size of the circuit: In the partitioning, no cyclic dependency is allowed between sub-functions. Moreover, the restriction of connectivity must be respected in order to route data coherently in matrices. So, partitioning results depend on number of dependences and the size of the circuit.
- failures on matrix mapping: During partitioning when a sub-graph cannot be mapped onto a matrix, the sub-graph has to be partitioned until it can be mapped. This case occurred twice for CMP8, i.e. the corresponding graph was reduced three times.

Benchmk circuit	No. operations	No. dependencies	Logical depth	No. sub-functions	No. dependencies	Fill factor (%)	No. hw placement failures
ALU2	45	64	14	6	9	46.9	1
CMP8	59	75	11	9	14	40.6	2
ADD8	101	132	22	11	18	57.4	0
ADSU8	133	180	25	18	30	45.8	0
CMP16	142	189	22	19	41	46.7	2
ADD16	197	260	38	21	36	58.6	0

ADSU16	261	356	41	27	49	60.5	1

Table 2. Experimental cluster mapping results for Xilinx-Virtex-4 benchmark functions

Insights and future challenges

In this work, we have firstly explored some of the opportunities opened up by the electrical behaviour of the double-gate CNTFET. We focused primarily on a specific property, namely the enabling of p-type or n-type device behaviour to be achieved in the same CNTFET according to the voltage applied to the back-gate. We have developed a family of dynamically reconfigurable logic cell, based on these devices and configured by the set of back gate bias voltages. These fine-grain reconfigurable cells have been considered as a universal reconfigurable cell enabling the synthesis of any Boolean function.

We then introduced a cluster-based matrix architecture useful for fine-grain reconfigurable logic cells based on emerging devices. This cluster-based architecture uses fixed interconnection topologies in order to reduce the overhead induced by conventional approaches. We have proposed a method to map specific functions to the matrices of reconfigurable cells. This method has been used to analyze intra-matrix topologies with respect to various metrics and shown that the mapping success rate is about 90% for 6-point function graphs and about 40% for 12-point graphs when using the Modified-Omega interconnect topology in a 4x4 matrix. This new function mapping method is a key step towards using fine-grain reconfigurable cells for the on-the fly and partial reprogrammability.

Finally, we proposed a methodology for mapping applications onto the cluster-based architectures. This model enables the exploration of the potential of this type of architectures for future applications and was used successfully to map complex benchmark functions onto the architecture. The model consists of two main steps accomplished through three complementary methods: partitioning, matrix mapping and cluster mapping.

The experience gained through this work lead us to believe that carbon-based computing fabrics can be a suitable support for pervasively deployment in many industries such as communications, energy, transport and healthcare. Tomorrow's computing platforms must achieve high computing throughput at very low power, while maintaining a level of robustness and capacity to be redeployed to new applications via software programming on very flexible hardware. In this way, engineers and scientists can benefit from almost unlimited computing power and can concentrate on developing imaginative and high added-value applications, while seamlessly supported by highly flexible and automatically configurable hardware and software programming models.

Such design of nanofabrics is at the interface between two scientific communities: that of nanoscience and nanotechnology on one hand, and that of data processing and embedded systems on the other. Critical challenges from the design point of view are to be able to understand how such devices can best be used in architectures and indeed if they can be expected to deliver significant benefits at this level, and to

extend existing design and simulation approaches to take into account the nanoelectronic approach. Close collaboration between designers and technologists is key to the strengthening of mutual design approaches necessary for the development of nanoelectronic systems and the generation of truly original designs exploiting the specific properties of nanodevices. The outcome of such collaboration is a clearer understanding of choices among the broad spectrum of potential devices and possible technologies capable of challenging conventional approaches in future nanoscale applications.

In our view, technology is evolving at such a rate that it is necessary to break the traditional technology – device – compact model – circuit – architecture development cycle by focusing on the fast-track integration of new devices into many-core computing platforms, through the implementation of a vertical and integrated research approach. In this way, circuit and architectural design activities are based on reasonable hypotheses issuing from device and technology work, and the development of the aforementioned devices and technology is focused towards the needs of high-level architectures.

References

1 TOP500 Supercomputing Sites, 2009. Available at http://www.top500.org
2 International Technology Roadmap for Semiconduc tors, 2007 edition. Available at http://public.itrs.net
3 O'Connor, I. et al. (2007), 'CNTFET Modeling and Reconfigurable Logic-Circuit design', IEEE Trans. Circuits and Systems, Vol. 54, No. 11, pp.2365-2379
4 Chau, R. et al., (2005) 'Benchmarking Nanotechnology for High-Performance and Low-Power Logic Transistor Applications', IEEE Trans. Nanotechnology, Vol. 4, No. 2. 153.
5 Raychowdhury, A., Roy, K., (2005) 'Carbon-Nanotube-Based Voltage-Mode Multiple-Valued Logic Design', IEEE Trans. Nanotechnology, Vol. 4, No. 2, pp. 168-179.
6 Sordan, R., Balasubramanian, K., Burghard, M., Kern, K., (2006) 'Exclusive-OR gate with a single carbon nanotube', Appl. Phys. Lett., Vol. 88, 053119.
7 Lin, Y., Appenzeller, J., Knoch, J., Avouris, P., (2005) 'High-Performance Carbon Nanotube Field-Effect Transistor With Tunable Polarities', IEEE Trans. Nanotechnology, Vol. 4, No. 5, September 2005
8 Moritz, C.A. et al., (2007) "Fault-Tolerant Nanoscale Processors on Semiconductor Nanowire Grids", IEEE Trans. CAS I, vol. 54, no. 11, p. 2422
9 Ding, L., Tselev, A., Wang J., et al., (2009) 'Selective Growth of Well-Aligned Semiconducting Single-Walled Carbon Nanotubes', Nano Lett., Vol. 9, No. 2, p. 800.
10 Akinwande, D., Yasuda, S., Paul, B., Fujita, S., Close, G., Wong, H.S.P., (2008) 'Monolithic integration of CMOS VLSI and carbon nanotubes for hybrid nanotechnology applications', IEEE Trans. Nanotechnology, Vol. 7, No. 5, p. 636.
11 Beyhoux, J.-M., Happy, H., Dambrine, G., Derycke, V., Goffman, M., Bourgoin, J.-P., (2006) 'An 8-GHz ft Carbon Nanotube Field-effect transistor for Gigahertz Range Applications', IEEE Electron Device Letters, Vol. 27, No. 8, pp. 681-683.
12 Adams, G.B., Agrawal, D.P., Siegel, H.J., "A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks," Computer, vol. 20, no. 6, pp. 14-27, June 1987.
13 Kalyanmoy, D., "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II", IEEE Trans. Evol. Comput., vol. 6, no. 3, p. 149, 2002
14 Xilinx, "Virtex-4 Libraries Guide for Schematic Designs," 2009