Topology Aware Mapping of Logic Functions onto Nanowire-based Crossbar Architectures [*]

Wenjing Rao
UC San Diego
CSE Department
wrao@cs.ucsd.edu

Alex Orailoglu
UC San Diego
CSE Department
alex@cs.ucsd.edu

Ramesh Karri
Polytechnic University
ECE Department
rkarri@poly.edu

ABSTRACT

Highly regular, nanodevice based architectures have been proposed to replace pure CMOS based architectures in the emerging post CMOS era. Since bottom-up self-assembly is used to build these architectures, regular nanowire crossbars are emerging as a promising candidate.

While these regular structures resemble CMOS programmable logic arrays (PLAs), PLA logic synthesis methodologies fail to solve the associated problems since the length and connectivity constraints imposed by individual nanowires in these crossbars translate into challenges hitherto not considered. These strict topological constraints should be considered while mapping Boolean functions onto nanowire crossbars during logic synthesis. We develop a mathematical model for this problem, an algorithm to solve it and three heuristics to improve the algorithm runtime.

Categories and Subject Descriptors: B.6.0 [Logic Design]
General Terms: Algorithms, Design
Keywords: Nanoelectronic, PLA, Crossbar, Logic synthesis

1. INTRODUCTION

Nanoelectronic devices based on quantum mechanical effects at the nanometer scale have been projected as a replacement for CMOS in the next generation electronic systems so as to extend Moore's law into the sub-nanometer regime [1, 2]. Unlike CMOS based systems, it is projected that nanoelectronic systems will be fabricated via bottom-up self-assembly of nanoelectronic devices [2, 3, 4]. It has been shown that regular logic architectures such as PLA-like crossbars is a promising choice in this context [5, 6, 7, 8].

Nanoelectronic crossbars are significantly different from CMOS PLAs. In a nanoelectronic crossbar, length of individual nanowires is limited and connectivity between nanowires is constrained. Finally, the bottom-up self-assembly process may result in a large number of defects that further constrain the effective connectivity of the crossbar. Logic synthesis targeting nanoelectronic crossbars should consider these topology constraints. In traditional CMOS logic synthesis, the last step of mapping a logic function onto a regular PLA is straightforward. However, in nanoelectronic crossbars, this step turns out to be a difficult problem.

In this paper, we develop a mathematical model for mapping a logic function onto a nanoelectronic crossbar subject to topological constraints. We then propose an algorithm for logic mapping that guarantees to find a solution if one exists. We then develop a number of heuristics that improve the algorithm run time.

2. MOTIVATION

Nanowire-based crossbar structures have been proposed due to their inherent regularity. Previous research in this area has shown that two terminal molecular diodes can be self-assembled at the crosspoints of such a crossbar [9, 10, 11, 12, 13], thus facilitating diode-based logic. Furthermore, three-terminal FET devices have also been demonstrated in [14]. Consequently, mapping of logic functions onto the nanoelectronic crossbars will be very similar to the CMOS PLA mapping due to a common underlying regular crossbar structure. Notwithstanding this apparent similarity, a number of constraints at the nanoscale impose significant differences between the nanowire-based crossbars and CMOS PLAs.

First, it is extremely difficult to transfer signals over a long distance due to signal attenuation in nanowires. This strictly limits the length and connectivity of nanowires. In a hybrid CMOS-nanowire-nanodevice crossbar architecture proposed in [11], it has been shown that (vertical and horizontal) nanowires of limited length can be weaved into an arbitrarily large and connected fabric. On one hand, each nanowire crosses a limited number of nanowires in the orthogonal layer. On the other hand, every nanowire has a unique set of wires in the orthogonal layer crossing it. Thus no two wires in the same direction has the same set of perpendicular crossing wires. Contrast this with a CMOS PLA wherein interconnect is offered between every pair of perpendicular wires. Secondly, even if a CMOS PLA architecture can be extended into the nanoscale, the large number of defects due to the defect-prone self-assembly will create breaks in the crossbar. These breaks translate into topological constraints on the underlying otherwise regular crossbar [5, 6, 7, 8],

Evidently, mapping a logic function onto a regular PLA is quite straightforward since the PLA structure provides the full flexibility of mapping a logic variable to any vertical wire and mapping a product term to any horizontal wire. However, in a nanoelectronic crossbar with breaks in wires (either periodic or arbitrary), this mapping has to obey certain constraints. In the first case, the nanowires are broken periodically and the pattern of a crossbar is available at the fabric time. In the second case, the breaks are introduced by the manufacturing defects and are rather arbitrary, thus demanding a post-fabric testing process to identify the crossbar structure pattern.

Figure 1 provides an example of a logic function being mapped onto a traditional PLA and a nanowire-based crossbar with a number of breaks in wires. If in the first step we map variable a to the vertical wire A, then the selection for the product terms ab and acd is immediately constrained to the horizontal wires $\{\beta, \delta\}$ since these are the only two wires crossing wire A.

[*]The work of the first two authors is supported in part by NSF Grant 0082325.

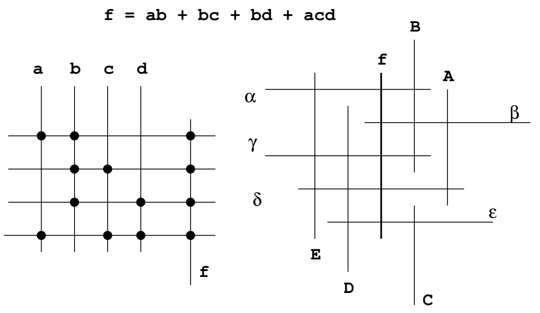

Figure 1: Logic function mapping in traditional PLA versus nanowire crossbars

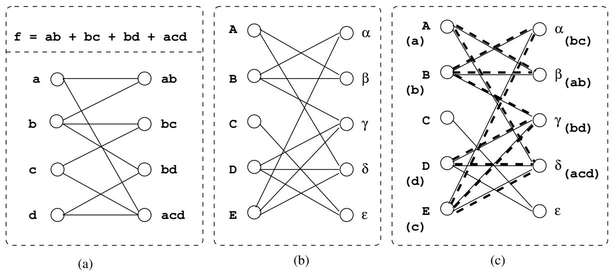

Figure 2: Bipartite graph representations of (a) a sum-of-product logic function and (b) a nanowire crossbar with (c) a successful embedding

It can be seen from this example that mapping a variable or a product term onto a specific nanowire imposes constraints on the subsequent mapping. Therefore, (i) determining whether a logic function can be successfully mapped to a specific crossbar with topology constraints and (ii) finding a mapping, if one exists, constitute important new challenges for logic synthesis on nanelectronic crossbars.

3. LOGIC MAPPING IN NANO CROSSBARS

3.1 Bipartite graph model

Definition: A *bipartite graph* $B(N, N', E)$ is a 3-tuple where N and N' are two disjoint sets of nodes and E is a set of edges. Every edge $e \in E$ connects a node $n \in N$ with a node $n' \in N'$.

Consider a two-level logic function in a sum-of-product form. The relationship between the logic variable set and the product term set can be represented by a bipartite graph, with node set N_{var} representing the logic variables and node set $N_{product}$ representing the product terms. In the logic function bipartite graph, there exists an edge between a node in N_{var} and a node in $N_{product}$ if and only if the corresponding product term contains the variable. Figure 2(a) illustrates the bipartite graph of the logic function shown in figure 1.

A crossbar architecture can also be represented by a bipartite graph, with one set of nodes representing the horizontal nanowires and the other set of nodes representing the vertical nanowires. A crosspoint connection between two orthogonal nanowires can then be represented as an edge in the bipartite graph. A nanowire crossbar with breaks in the regular structure (either due to the topology constraints or due to fabrication time defects) is shown in figure 1 and its bipartite graph representation is shown in figure 2(b).

When implementing a two-level logic function in a crossbar structure, the relationship between a product term containing a variable manifests as the connection between a horizontal wire and a vertical wire in the crossbar. The problem of mapping a two-level logic function onto a target nanowire crossbar entails matching each variable and each product term with specific nanowires in the crossbar structure, such that the relationships among the variables and product terms can be represented by existing connections in the crossbar. This logic function to crossbar mapping problem can be formulated as embedding the logic function bipartite graph into the nanowire crossbar bipartite graph.

Definition: *bipartite graph embedding:* Given a logic function bipartite graph $B1(N_{var}, N_{product}, E1)$ and a crossbar nanofabric bipartite graph $B2(N_{ver}, N_{hor}, E2)$, find a node mapping $(M : N_{var} \rightarrow N_{ver}; N_{product} \rightarrow N_{hor})$ such that $\forall (n, n') \in E1, (n \in N_{var}, n' \in N_{product})$, we have $(M(n), M(n')) \in E2$. The dotted lines on Figure 2(c) illustrates a valid mapping from the logic function bipartite graph in figure 2(a) into the crossbar bipartite graph shown in figure 2(b).

On one hand, a CMOS PLA represents a *complete bipartite graph* with an edge between every pair of nodes in the horizontal wire and vertical wire sets. Embedding an arbitrary bipartite graph onto a complete bipartite graph is straightforward. Mapping a specific node in the logic function bipartite graph onto a node in the PLA complete bipartite graph does not impose any constraints on the subsequent embedding steps and consequently, mapping a logic function onto the PLA structure is trivial. On the other hand, nanowire crossbar bipartite graphs are not complete due to breaks in wires as discussed above and hence mapping a specific node in the logic function bipartite graph onto a node in the nanowire crossbar bipartite graph imposes a number of constraints on the subsequent embedding steps.

3.2 Embedding a logic function bipartite graph into a crossbar bipartite graph

We develop a recursive algorithm to embed the logic function bipartite graph $B1(N_{var}, N_{product}, E1)$ into the nanowire crossbar bipartite graph $B2(N_{ver}, N_{hor}, E2)$, such that all the edges in $B1$ can be mapped to the existing edges in $B2$.

```
EMBED ($N1 \subseteq (N_{var} \cup N_{product})$, $N2 \subseteq (N_{ver} \cup N_{hor})$)
//taking as input $B1$'s unmapped node set N1 and $B2$'s unmapped node set N2
  1. if ($N1 = \emptyset$): //no more unmapped nodes in $B1$
        return success
  2. if ($N2 = \emptyset$): //no more unmapped nodes left in $B2$ to map $N1$, thus fail
        return fail
  3. select one unmapped node $n1 \in N1$
  4. if($n1 \in N_{var}$): //variable → vertical wires
        repeat step (a) - (d) for every node $n2 \in (N_{ver} \cap N2)$
     else ($n1 \in N_{product}$): //product term → horizontal wires
        repeat step (a) - (d) for every node $n2 \in (N_{hor} \cap N2)$
          (a) if $n1$ can be mapped to $n2$, mark $n1 \rightarrow n2$ //n1 mapped to n2
          (b) EMBED ($N1 - n1$, $N2 - n2$)
          (c) if (success):
                return success
          (d) else (fail): //backtrack, try a different n2
                unmap $n1$ with $n2$
  5. return fail //failed to map n1 to any possible nodes in B2
```

The above algorithm embeds $B1$ to $B2$ in two steps. First, it maps one node from $B1$ to $B2$. Then it recursively solves the reduced embedding problem with the remaining unmapped nodes. The critical part of the ***EMBED*** algorithm to guarantee the correctness is the justification step in 4(a). Assume $E1$ is the edge set of $B1$ and $E2$ is the edge set of $B2$; the following sufficient condition justifies that a node $n1$ in $B1$ *cannot* be mapped to a node $n2$ in $B2$:

$\exists(\tilde{n1} \in B1$ and $\tilde{n2} \in B2)$, $\tilde{n1}$ already mapped to $\tilde{n2}$, $(n1, \tilde{n1}) \in E1$ while $(n2, \tilde{n2}) \notin E2$.

For an already mapped node $\tilde{n1}$ that is connected to $n1$ in $B1$, if the edge $(n1, \tilde{n1})$ cannot be embedded in $B2$, then such a mapping of node $n1$ does not lead to a valid solution. If all the unmapped nodes in $B2$ are justified as unmappable to $n1$, then the algorithm fails.

3.3 Heuristics to prune impossible mappings

For a chosen $n1$ in $B1$, the recursive ***EMBED*** algorithm explores a number of solution subspaces, each determined by a possible selection of $n2$ from $B2$. The algorithm is capable of exploring all these subspaces for a solution by performing a sequence of trial mappings from $n1$ to each possible $n2$. We propose three heuristics to quickly prune the subspaces which have no solution, thus improving the algorithm runtime.

3.3.1 Fanout embedding heuristic

According to the constraint imposed in the mapping process, when $n1 \in B1$ is mapped to $n2 \in B2$, the fanout nodes of $n1$ are deemed to be mapped within the fanout nodes of $n2$. Therefore, if the number of edges connected to $n1$, i.e., the degree of $n1$, is more than the degree of $n2$, then such a mapping is deemed to fail as it cannot lead to any valid solution. In the example of figure 2, node c in 2(a) can be mapped to node A, B, D, E, but not C in 2(b) according to this criteria, since node c in 2(a) has a degree of 2 while node C in 2(b) has a degree of 1.

Furthermore, during the mapping process, it is possible that a subset of the fanout nodes of $n1$ as well as a subset of the fanout nodes of $n2$ have been mapped already. Therefore, a more precise justification should consider the fanout nodes that have not yet been mapped. Basically, if the number of unmapped fanout nodes in $n1$ is larger than the number of unmapped fanout nodes of $n2$, then $n1$ cannot be mapped to $n2$. Otherwise, we denote that $n1$ can be *fanout embedded* to $n2$. The justification process of $n1$ to $n2$ can be performed in time $O(m)$, provided that m is the number of nodes in $B1$ and $B2$.

3.3.2 Fanout-fanout embedding heuristic

When $n1$ in $B1$ is mapped to $n2$ in $B2$, then the unmapped fanout nodes of $n1$, denoted as $\Phi(n1)$, are to be mapped within $n2$'s unmapped fanout nodes, denoted $\Phi(n2)$. If $n1 \rightarrow n2$ leads to a valid solution, then there should exist a valid mapping $\Phi(n1) \rightarrow \Phi(n2)$. If there is no mapping to guarantee a *fanout embedding* from every node in $\Phi(n1)$ to $\Phi(n2)$, then $n1 \rightarrow n2$ does not lead to any valid solution.

Consider a simple example in figure 2. Node ab in figure 2(a) has 2 unmapped fanout nodes a and b with degree 2 and 3. Node ε in figure 2(b) has 2 unmapped fanout nodes C and D with degree 1 and 3. Although ab and ε have the same number of unmapped fanout nodes, mapping ab to ε does not lead to any valid solution, since there is no way to map $\Phi(ab)$ to $\Phi(\varepsilon)$.

The justification process for $n1$ with $n2$ has a time complexity of $O(m^2)$. Going through sorted $\Phi(n1)$ and $\Phi(n2)$ takes $O(m \log m)$, which is dominated by the calculation for the degree numbers in $\Phi(n1)$ and $\Phi(n2)$, which takes $O(m^2)$.

3.3.3 Fanout chain embedding heuristic

Suppose we have a bipartite graph $B1(N1, N1', E1)$ with node $n1 \in N1$; then the fanout node set of $n1$, denoted as $\Psi(n1)$, is a subset of $N1'$. We can observe that the set $\Psi(n1)$ has a further fanout set that contains nodes that are connected to any member of $\Psi(n1)$. The fanout set of a node set N is defined as $\widehat{\Psi}(N) = \bigcup_{n \in N} \Psi(n)$.

Assume that $n1$ is mapped to $n2$ in $B2(N2, N2', E2)$; then not only is $\Psi(n1)$ deemed to be mapped within $\Psi(n2)$, but also the fanout set of $\Psi(n1)$ is to be mapped within the fanout set of $\Psi(n2)$. Therefore, $n1 \rightarrow n2$ implies $\Psi(n1) \rightarrow \Psi(n2)$, which further implies:

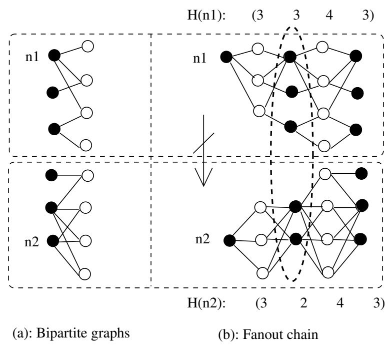

Figure 3: Fanout chain embedding example

$\widehat{\Psi}(\Psi(n1)) \rightarrow \widehat{\Psi}(\Psi(n2))$, $\widehat{\Psi}(\widehat{\Psi}(\Psi(n1))) \rightarrow \widehat{\Psi}(\widehat{\Psi}(\Psi(n2)))$, and so on. For ease of representation, we define the following: $\widehat{\Psi}^1(N) = \widehat{\Psi}(N)$; $\widehat{\Psi}^{i+1}(N) = \widehat{\Psi}(\widehat{\Psi}^i(N))$.

The sequence of $\widehat{\Psi}^i(N)$ is denoted as a *fanout chain* and an example is shown in figure 3. Figure 3(a) shows the two bipartite graphs and figure 3(b) shows the fanout chains of node $n1$ and $n2$ for the two bipartite graphs. We define *height chain*:

$$H(n1) = (|\widehat{\Psi}^1(n1)|, |\widehat{\Psi}^2(n1)|, |\widehat{\Psi}^3(n1)|, ...)$$

as is shown in figure 3(b). If the mapping of a node $n1$ to a node $n2$ leads to a valid solution, a necessary condition for this is that the fanout chain of $n1$ must be mapped into the fanout chain of $n2$. The height chain of $n1$ consequently needs to be smaller at every point than the height chain of $n2$. As is shown in the example of figure 3, $n1$ cannot be mapped to $n2$ since $H_2(n1) = 3$, which is larger than $H_2(n2) = 2$.

The following condition justifies that node $n1 \in B1$ *cannot* be mapped to a node $n2 \in B2$: $\exists i$ such that $H_i(n1) > H_i(n2)$. Basically, if at a any point the height chain of $n1$ is greater than the height chain of $n2$, then the fanout chain of $n1$ cannot be embedded into the fanout chain of $n2$. Otherwise, there is a *fanout chain embedding* from $n1$ to $n2$.

For any node $n1 \in N1$ in $B1(N1, N1', E1)$; it can be shown that $H_{2i}(n1)$ and $H_{2i-1}(n1)$ are two monotonically increasing functions, with the upper bound for $H_{2i}(n1)$ being $|N1|$ and the upper bound for $H_{2i-1}(n1)$ being $|N1'|$. Furthermore, these two functions reach their maximum values within $MAX(2|N1|, 2|N1'|)$ steps. This implies that the height chain for any node in a bipartite graph reaches a maximum value in a limited length bounded by $MAX(2|N1|, 2|N2|)$ and remains constant afterwards. Therefore, calculating the height chain of any node takes time $O(m)$ where m is the number of nodes in the two bipartite graphs $B1$ and $B2$. In fact, since the degree of each node is constant, the Ψ height chain needs to be calculated only once before the algorithm starts.

4. EXPERIMENTAL RESULTS

The proposed algorithm is implemented in C++. While the *basic* version uses the *fanout embedding heuristic* alone, the *advanced* version uses all the three heuristics to cut down on backtracking significantly. The algorithm is run on a computer with 2.8GHz CPU, 1G Byte memory, under the Linux operating system. Table 1 summarizes the experimental results of embedding the logic function bipartite graph onto a nanowire-based crossbar bipartite graph for four logic synthesis benchmarks from the LGSynth93 benchmark set [15].

For each of the benchmarks, we use multiple nanowire-based crossbars (each representing a different defect map or topology constraints) by using the corresponding bipartite graph. The size of the crossbar (size column in table) is determined according to the number of variables and product terms in the logic function. The connectivity of

PLA circuit	# of inputs	Crossbar size	Crossbar degree	find mapping?		run-time (s)	
				basic	adv	basic	adv
rd53	10	11	4	$\times$	$\times$	17	1
		13	5	$\times$	$\times$	149	16
		15	6	$\times$	$\times$	909	114
		17	7	$\surd$	$\surd$	1	1
		21	9	$\surd$	$\surd$	3	1
		25	11	$\surd$	$\surd$	6	0
misex1	16	16	4	$\times$	$\times$	581	81
		18	5	$\times$	$\times$	> 1200	> 1200
		20	6	$\surd$	$\surd$	1072	4
		22	7	$\surd$	$\surd$	768	80
		24	8	$\times$	$\surd$	> 1200	203
		26	9	$\times$	$\surd$	> 1200	441
5xp1	14	18	3	$\times$	$\times$	1	1
		20	4	$\times$	$\times$	103	1
		22	5	$\surd$	$\surd$	19	13
		24	6	$\surd$	$\surd$	13	12
		26	7	$\surd$	$\surd$	6	2
		28	8	$\surd$	$\surd$	7	2
bw	10	13	3	$\times$	$\times$	1	1
		15	4	$\times$	$\times$	1	12
		17	5	$\times$	$\times$	> 1200	> 1200
		19	6	$\times$	$\times$	> 1200	> 1200
		21	7	$\times$	$\times$	> 1200	> 1200
		23	8	$\times$	$\surd$	> 1200	66
		25	9	$\times$	$\surd$	> 1200	32
		27	10	$\times$	$\surd$	> 1200	52
		29	11	$\times$	$\surd$	> 1200	79
		31	12	$\times$	$\surd$	> 1200	115

Table 1: Experimental results for selected logic synthesis benchmarks

the crossbar (degree column in table) is defined as the fixed number of nodes that each node is connected to. The crossbar size increases with the crossbar degree. The *find mapping* column lists whether the algorithm has successfully found the mapping and the *run-time* column illustrates the number of seconds spent for the algorithm. We set a runtime limit of 1200 seconds for both versions of the algorithm. When this limit is exceeded, the algorithm returns fail and is denoted as "> 1200" in the table.

It can be seen from table 1 that, a successful mapping of a logic function onto a nanowire-based crossbar and the running time of the embedding algorithm depend heavily on the connectivity of the crossbar. Both the basic and the advanced versions exhibit similar behavior as a function of the connectivity of the crossbar.

- When the connectivity is exceedingly low, the search space for exploration is limited. Therefore, it is easy to justify at the decision point that none of the search spaces leads to a solution. Obviously, the algorithm fails to return a successful mapping within a very low run-time under this case.
- As connectivity increases the search space grows exponentially. Backtracking is required to explore the solution space for a possible solution. The runtime of the algorithm then sharply increases with the increased connectivity. Under this case, where the search space is huge with only limited number of solutions, the searching process takes exceedingly long time. When connectivity further increases, the number of valid solutions in the search space is boosting. Therefore, it becomes easy to find solutions while exploring the solution space.

Although both versions show similar behavior with increasing connectivity levels for the underlying crossbar, the *advance* algorithm, which utilizes all three heuristics, cuts off backtracking processes and is significantly faster than the *basic* algorithm.

5. CONCLUSIONS

With nanoelectronic technologies evolving at a rapid clip towards practical realization of nanoelectronic computing systems, it is important to identify and propose research on the new challenges emerging during their design process. This paper identifies a new challenge of mapping a logic function onto a nanowire crossbar under the inherent constraints of limited connectivity and unreliability. This paper sets up an important initial contribution in the logic synthesis for nanoelectronics. The future research directions include the decomposition of large logic functions onto the nanoelectronic crossbar structures and an incremental bipartite graph creation integrated with the defect testing procedure of the nanofabrics.

6. REFERENCES

[1] European Commission, *Technology Roadmap for Nanoelectronics*, 2001.

[2] ITRS, *International Technology Roadmap for Semiconductors Emerging Research Devices*, 2004.

[3] M. S. Montemerlo, J. C. Love, G. J. Opitech, D. G. Gordon and J. C. Ellenbogen, *Technologies and Designs for Electronic Nanocomputers*, MITRE, July 1996.

[4] V. V. Zhirnov and D. J. C. Herr, "New Frontiers: Self-Assembly and Nanoelectronics", *IEEE Computer*, vol. 34, n. 1, pp. 34–43, January 2001.

[5] P. J. Kuekes, D. R. Stewart and R. S. Williams, "The Crossbar Latch: Logic Value Storage, Restoration, and Inversion in Crossbar Circuits", *Journal of Applied Physics*, vol. 97, n. 3, pp. 034301, July 2005.

[6] A. DeHon, "Array-Based Architecture for FET-Based, Nanoscale Electronics", *IEEE Transactions on Nanotechnology*, vol. 2, n. 1, pp. 23–32, 2003.

[7] R. Rubin and A. DeHon, "Nanowire-based Sublithographic Programmable Logic Arrays", in *FPGA*, pp. 123–132, 2004.

[8] G. Snider and W. Robinett, "Crossbar Demultiplexers for Nanoelectronics Based on n-Hot Codes", *IEEE Transactions on Nanotechnology*, vol. 4, pp. 249–254, 2005.

[9] C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams and J. R. Heath, "Electronically Configurable Molecular-Based Logic Gates", *Science*, vol. 285, pp. 391–394, July 1999.

[10] Y. Luo, C. P. Collier, J. O. Jeppesen, K. A. Nielsen, E. DeIonno, G. Ho, J. Perkins, H. Tseng, T. Yamamoto, J. F. Stoddart and J. R. Heath, "Two-Dimensional Molecular Electronics Circuits", *ChemPhysChem*, vol. 3, pp. 519–525, 2002.

[11] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits with Two-terminal Nanodevices", *Nanotechnology*, vol. 16, pp. 888–900, Apr 2005.

[12] S. C. Goldstein and M. Budiu, "NanoFabrics: Spatial Computing Using Molecular Electronics", in *ISCA*, pp. 178–191, 2001.

[13] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach and M. M. Ziegler, "Molecular Electronics: From Devices and Interconnect to Circuits and Architecture", *Proceedings of the IEEE*, vol. 91, n. 11, pp. 1940–1957, November 2003.

[14] G. Snider, P. J. Kuekes and R. S. Williams, "CMOS-like Logic in Defective, Nanoscale Crossbars", *Nanotechnology*, vol. 15, pp. 881–891, Aug 2004.

[15] Collaborative Benchmarking Laboratory, *1993 LGSynth Benchmarks*, North Carolina State University, Department of Computer Science, 1993.