A Reconfigurable Architecture for Hybrid CMOS/Nanodevice Circuits

Dmitri B. Strukov and Konstantin K. Likharev
Stony Brook University
Stony Brook, NY 11794-3800, U.S.A
dstrukov@ic.sunysb.edu, klikharev@cc.notes.sunysb.edu

ABSTRACT

This report describes a preliminary evaluation of performance of a cell-FPGA-like architecture for future hybrid "CMOL" circuits. Such circuits will combine a semiconductor-transistor (CMOS) stack and a two-level nanowire crossbar with molecular-scale two-terminal nanodevices (programmable diodes) formed at each crosspoint. Our cell-based architecture is based on a uniform CMOL fabric of "tiles". Each tile consists of 12 four-transistor basic cells and one (four times larger) latch cell. Due to high density of nanodevices, which may be used for both logic and routing functions, CMOL FPGA may be reconfigured around defective nanodevices to provide high defect tolerance. Using a semicustom set of design automation tools we have evaluated CMOL FPGA performance for the Toronto 20 benchmark set, so far without optimization of several parameters including the power supply voltage and nanowire pitch. The results show that even without such optimization, CMOL FPGA circuits may provide a density advantage of more than two orders of magnitude over the traditional CMOS FPGA with the same CMOS design rules, at comparable time delay, acceptable power consumption and potentially high defect tolerance.

Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design styles—*logic arrays*; B.7.1 [**Integrated Circuits**]: Types and Design Styles— *advanced technologies*

General Terms

Architecture, Design, Algorithms

Keywords

Programmable logic, integrated hybrid circuits, nanoelectronics, programmable interconnect

FPGA'06, February 22–24, 2006, Monterey, California, USA.

1. INTRODUCTION

It is now believed that the growing problems with scaling of CMOS technology [12, 20] may be only overcome by a radical paradigm shift from lithography-based fabrication to the so-called "bottom-up" approach - see, e.g., Ref. 33. In this approach, the smallest active devices of integrated circuits are not defined lithographically but assembled from parts with fundamentally reproducible size and structure, e.g., few-nm-scale molecules. This procedure may be rationally envisioned only for two-terminal nanodevices. Since such devices have limited functionality, most efforts in the development of nanoelectronic architectures are focusing on hybrid CMOS/nanodevice circuits - see, e.g., Refs. 9, 13, 17, 19, 27, 30, and also recent reviews [10, 16, 21, 28]. In most of the proposed hybrid circuits, relatively large silicon MOSFETs are used for signal restoration, long-range communications, I/O, testing/bootstrapping, and some other critical functions, while a set of dense nanodevices provides most of information storage and signal processing.

We believe that the most promising species of CMOS/nanodevice hybrids are "CMOL" circuits [20, 21]. As in several earlier hybrid proposals, in CMOL circuits the two-terminal nanodevices are formed at each crosspoint of a "crossbar" array, consisting of two levels of nanowires (Fig. 1). However, in order to overcome the CMOS/nanodevice interface problems pertinent to earlier concepts, in CMOL circuits the interface is provided by sharp-tip pins that are distributed all over the circuit area, on the top of the CMOS stack. By activating two pairs of perpendicular CMOS lines, two pins (and two nanowires they contact) may be connected to CMOS data lines (Fig. 1b).

If the nanodevices have a sharp current threshold, like in the usual diodes, such access allows each of them to be tested. Moreover, if such a diode is programmable, i.e. may be switched between two internal states, e.g., as the single-electron latching switch [11, 20], each device may be turned on or off by applying voltages $\pm V_{\rm W}$ to the selected nanowires, so that voltage $V = \pm 2V_{\rm W}$ applied to the selected nanodevice exceeds the corresponding switching threshold, while half-selected devices (with $V = \pm V_{\rm W}$) are not disturbed [21].

As Fig. 1c shows, interface pins of each type (reaching to the lower and upper nanowire level) are arranged in a square array with side $2\beta F_{\rm CMOS}$, where $F_{\rm CMOS}$ is the half-pitch of the CMOS subsystem, and β is a dimensionless factor larger than 1 that depends on the CMOS cell complexity. Relative to the CMOS pin array, the nanowire crossbar is rotated by angle $\alpha = \arcsin(F_{\rm nano}/\beta F_{\rm CMOS})$, where $F_{\rm nano}$ is the

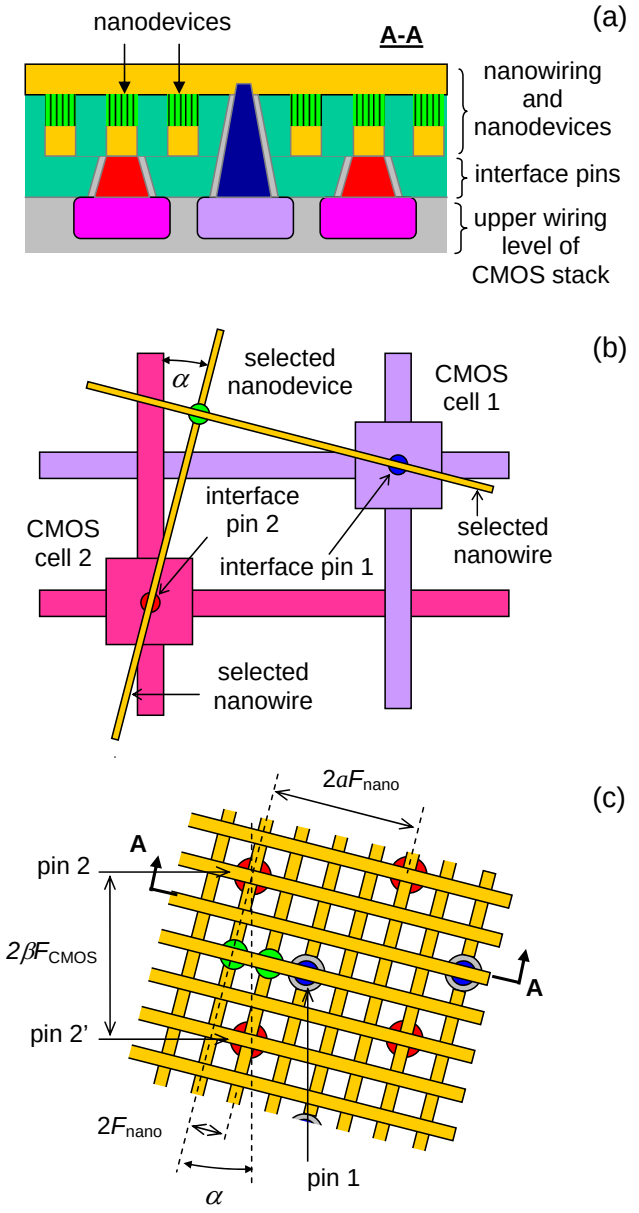

Figure 1: Low-level structure of the generic CMOL circuit: (a) schematic side view (A-A cross-section); (b) the concept of addressing a particular nanodevice, and (c) zoom-in on several adjacent pins. The last panel shows that any nanodevice may be addressed via the appropriate pin pair (e.g., pins 1 and 2 for the left of the two shown devices, and pins 1 and 2' for the right device). On panel (b), only the activated CMOS lines, cells, and nanowires are shown, while panel (c) shows only two devices. (In reality, similar nanodevices are formed at all nanowire crosspoints.) Also disguised on panel (c) are CMOS cells and wiring.

nanowiring half-pitch. Figure 1c illustrates the fact that this approach allows a unique access to any nanodevice, even if $F_{\rm nano} \ll F_{\rm CMOS}$ - see Ref. 21 for a detailed discussion of this point.[1]

Earlier we have shown that CMOL circuits may be used to build several types of highly defect-tolerant circuits including terabit-scale memories [21, 29, 31] and mixed-signal neuromorphic circuits capable of advanced information processing, e.g. fast classification of large patterns such as few-megapixel images [18, 34]. However, the most important application of CMOL technology may be in reconfigurable Boolean logic circuits [30] whose structure resembles the so-called cell-based FPGAs [24]. In these "CMOL FPGA" circuits (Fig. 2a,b) the basic cell includes two pass transistors and one inverter, and is connected to the nanowire/ nanodevice crossbar via two pins. During the configuration process the inverters are turned off, and the pass transistors may be used for setting the binary state of each molecular device, just as in CMOL memories [21, 29, 30]. By turning programmable diodes ON or OFF, each pin of a basic cell may be connected through a nanowire-nanodevice-nanowire link to each of $M = a^2 - 2$ other cells within a near square-shaped "connectivity domain" (painted light-gray in Fig. 2a). Figures 3 and 4 show how such fabric may be configured for the implementation of NOR gates. This is already sufficient to implement any logic function, though other gates (e.g., NAND) are clearly possible.

In our previous work [30] we analyzed defect tolerance and performance of two combinational (latch-free) logic circuits which were manually mapped on a simple CMOL FPGA fabric: a 32-bit Kogge-Stone adder and a 64-bit full crossbar. The results implied that CMOL FPGA may provide area-delay advantage beyond two orders of magnitude over purely CMOS FPGA circuits, at manageable power consumption, simultaneously with high defect tolerance (above 20% of bad nanodevices). Later, an almost similar density advantage was reported for CMOL FPGA implementation of an advanced encryption algorithm [22]. However, these results were still insufficient to evaluate the benefits of the CMOL FPGA concept for general-purpose computing. The goal of this report is to describe our next step in this direction: an analysis of CMOL FPGA performance for all circuits of the Toronto 20 benchmark set [2]. For this purpose, we have developed semi-custom software for an automated CMOL FPGA circuit design.

[1] For this work, we have made two changes in the CMOL geometry. First (and most importantly), in the initial version of CMOL circuits [21], interface pins leading to the upper nanowire level would pass between nanowires of the lower level. This had restricted the maximum CMOL circuit yield (without nanoscale alignment) to 50%. This work is based on an improved version of CMOL, with insulator-covered pins intentionally interrupting the lower nanowires - see Fig. 1a. While keeping our prior results on CMOL FPGA [30] valid, this modification improves the circuit yield substantially, raising its theoretical upper bound to 100%. Second, in this paper the nanowire crossbar is not rotated by the additional angle of 45°, which was convenient for manual circuit mapping in Ref. 30. In this case angle α is given by $\tan\alpha \equiv 1/a$, where a is an integer defining the range of cell interaction (Fig. 2a). For fixed fabrication technology parameters $F_{\rm CMOS}$, $F_{\rm nano}$ and β_{min}, the lower bound on a is given by equation $a_{\rm min} = \sqrt{(\beta_{\rm min} F_{\rm CMOS}/F_{\rm nano})^2 - 1}$.

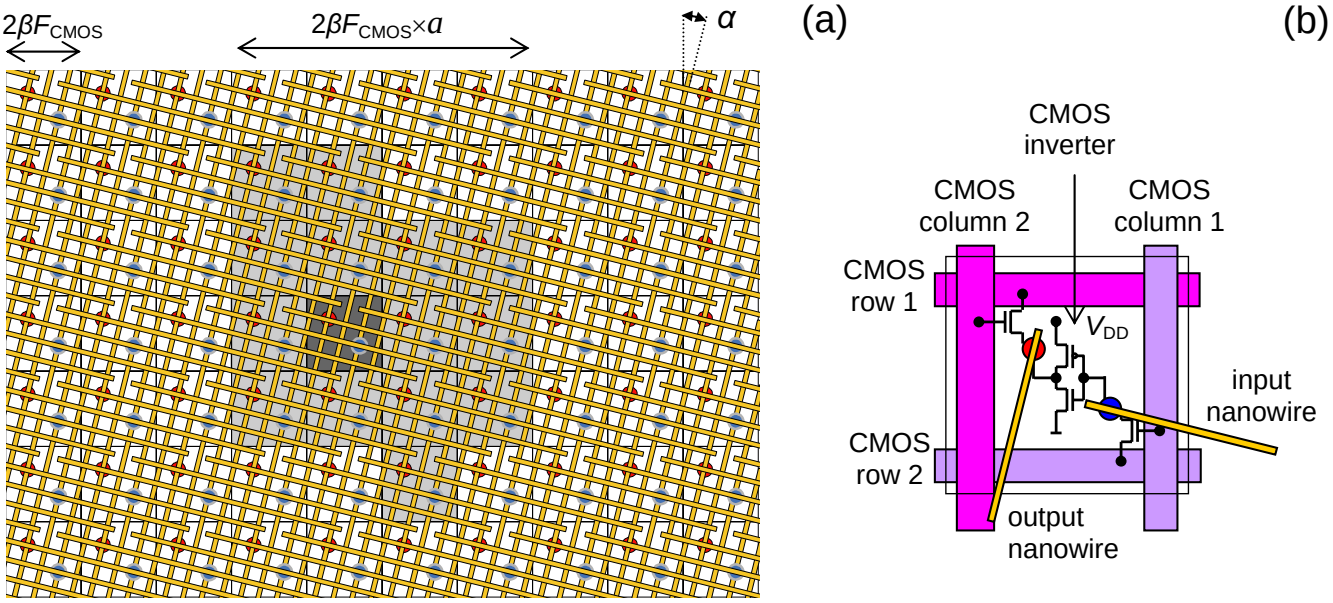

Figure 2: CMOL FPGA: (a) the general structure of the circuit (for the particular case $a = 4$) and (b) the basic cell. In panel (a), output pins of $M = a^2 - 2 = 14$ cells painted light-gray may be connected to the input pin of a specific cell (shown dark-gray) via a nanowire-nanodevice-nanowire links. For the sake of clarity, panel (b) shows only the two nanowires which are contacted by interface pins of the given cell.

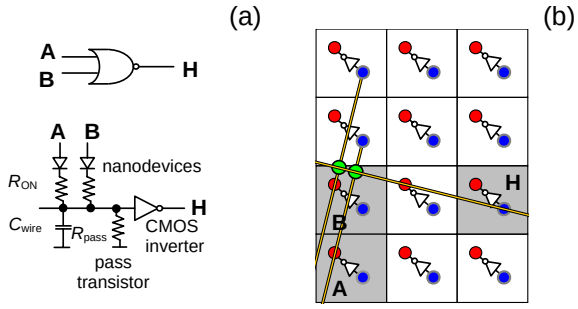

Figure 3: Fan-in-two NOR gate: (a) equivalent circuit and (b) physical implementation in CMOL. Note that in panel (b) only two (shown) nanodevices on the input nanowire of cell H are set to the ON state, while others (not shown) are set to the OFF state. For the sake of clarity panel (b) shows only the nanowires used for the gate.

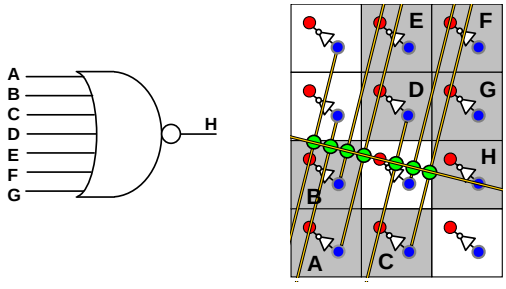

Figure 4: Example of CMOL implementation of a 7-input NOR gate. Just as in Fig. 3, only the engaged nanowires and nanodevices are shown.

2. HARDWARE ARCHITECTURE

A genuine optimization of CMOL FPGA circuit architectures would require a completely new set of CAD tools, whose development is a challenging task. At this preliminary stage, our choice was instead to get as much leverage as possible from the existing software used for mapping and architecture exploration of semiconductor logic, in particular, from the design automation tools for island-type CMOS FPGAs [5].

In order to use these tools, we have restricted our design to a specific, simple two-cell-species CMOL fabric. (Though such fabric is a generalization of the single-cell CMOL FPGA structure considered in Ref. 30, it is still a small subset of all possible CMOL architectures, so that the results described below may be certainly improved in future.) The fabric is a uniform mesh of square-shaped "tiles" (Fig. 5a). Each tile consists of a shell of T basic cells (Fig. 2b) surrounding a single "latch" cell (Fig. 5b). The latter cell is just a level-sensitive latch implemented in the CMOS subsystem, connected to 8 interface pins, plus two pass transistors used for circuit configuration. Note that all four pins of each (either input or output) group are always connected, so that the nanowires they contact always carry the same signal. This means that at configuration, groups of four nanodevices sitting on these wires may be turned on or off only together. A simple analysis shows that this does not impose any restrictions on the CMOL FPGA fabric functionality.

CMOS layout estimates assuming a compact layout from, e.g., Ref. 14 have shown that the latch cell requires an area approximately four times larger than that of the basic cell. As a result, for this analysis we have accepted $T = 12$, so that the total tile area is $T + 4 = 16 = 4 \times 4$ basic cells (Fig. 5a). This provides a latch/logic resource ratio comparable to those of the conventional FPGAs. In fact, the

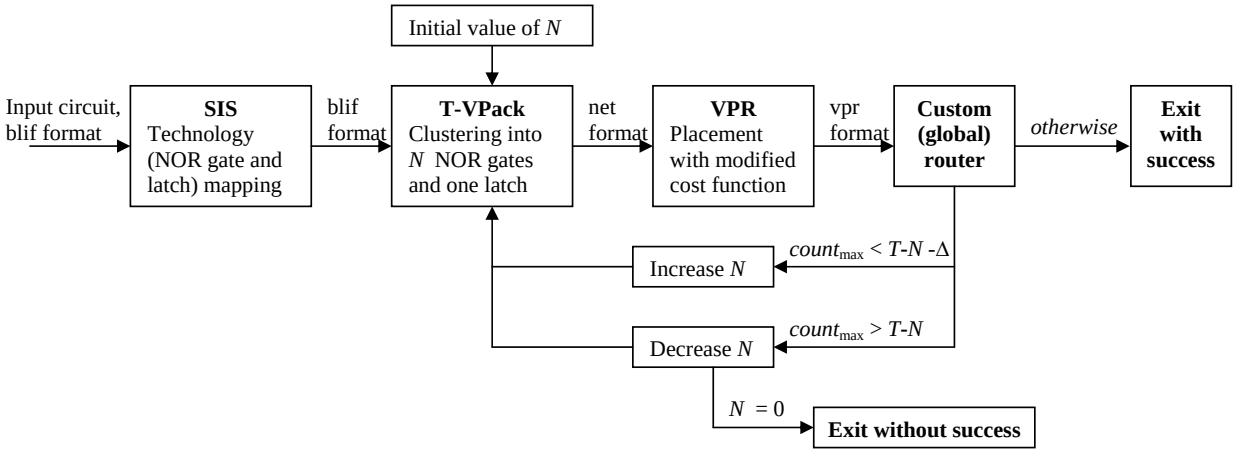

Figure 6: CMOL FPGA design flow used in this work.

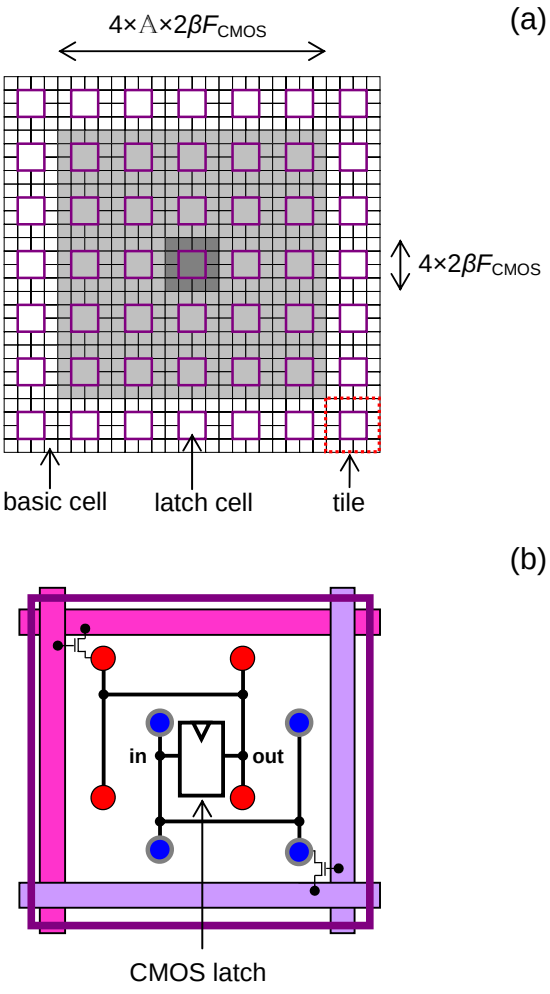

Figure 5: (a) Latched CMOL FPGA fabric and (b) latch cell structure. On panel (a), the tile connectivity domain with linear size $\mathbb{A} = 5$ is painted light gray. Any cell in this domain can be connected to any cell of the central tile (shown dark-gray) via a single nanowire-nanodevice-nanowire link.

4-input parity function (the worst-case Boolean function of 4 inputs) can be implemented with 14 four-input NOR gates, while an average 4-input Boolean function requires much less (6 to 8) of such gates. Hence each CMOL tile is crudely similar in functionality to the basic logic element consisting of a four-input LUT and one latch [5].

3. DESIGN AUTOMATION

3.1 Technology Mapping and Clustering

The convenience of the proposed CMOL hardware structure is that, from the design point of view, the CMOL tile can be treated in the same way as that of the island-type CMOS FPGA. Indeed, consider the design flow shown in Fig. 6. Using the SIS package [26], we first map the original pre-optimized logic circuit onto a network of NOR gates (with a certain maximum fan-in) and latches (if any), to produce a netlist in blif format. Next, we reserve a certain number (N) of basic cells inside each tile to perform logic operations, while the rest ($T - N$) basic cells are left for routing. The netlist of NOR gates is then partitioned into logic clusters of N gates with the help of the T-VPack program [5]. This code, while originally written to pack LUTs, can work as well for NOR gates, since the representations in the blif format for both gate libraries are the same, and any CMOL NOR gate occupies exactly one basic cell regardless of its fan-in [21, 30]. The only modification which has been made to T-VPack is the addition of a latch counter to prevent packing of more than one latch into one cluster.

3.2 Cluster Placement

The logic clusters are then mapped on the CMOL tile fabric (one cluster per tile) using VPR tool [4,5]. However, the original linear congestion cost function [4] is modified, since it was not adequate for CMOL FPGA where connections within the connectivity domain are not limited by CMOS resources. More specifically, the cost function for some cluster (tile), located in position x_0, y_0, whose outputs are connected to $\mathbb{N}_{\text{clusters}}$ other clusters, is calculated as

$$Cost = \sum_{i=1}^{\mathbb{N}_{\text{clusters}}} \left\lfloor \frac{2(\max(|x_0 - x_i|, |y_0 - y_i|) - 1)}{\mathbb{A} - 1} \right\rfloor, \qquad (1)$$

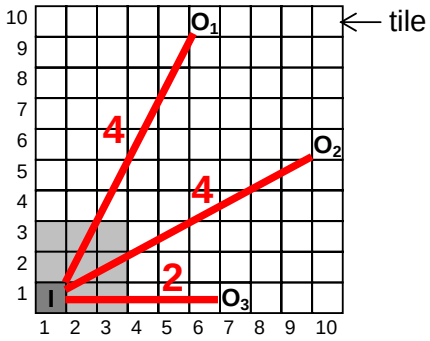

Figure 7: Cost function calculation example. Assuming the linear size of the tile connectivity domain $\mathbb{A} = 5$, the cost function of the connections between cluster I and clusters O_1, O_2 and O_3 is 10. The numbers shown in red are the contributions of specific connections to the cost function. In this particular example, these numbers also give parameters $Hop_{\min}$ used in the global routing procedure (see Sec. 3.3 below).

where x, y is the position of a cluster inside the array (defined exactly as in Ref. 5), while $\mathbb{A} = 2\lfloor a/8 \rfloor - 1$ is the linear size of the "tile connectivity domain". This domain is such a set of tiles that any cell from each tile of the domain can be connected to any cell of the initial tile directly, i.e. via one nanowire-nanodevice-nanowire link. For instance, Fig. 5a shows a tile connectivity domain for the case $\mathbb{A} = 5$. (In more realistic cases $a = \beta F_{\rm CMOS}/F_{\rm nano} \approx 40$, i.e. $\mathbb{A} \approx 9$.) Figure 7 gives an example of placement cost calculation for $\mathbb{A} = 5$. In this example, one input (I) and three output (O_1, O_2, and O_3) clusters are located in tiles with positions (1, 1), (6, 10), (10, 6), and (7, 1), correspondingly. Hence, according to Eq. (1) the cost function is $Cost = 4+4+2 = 10$.

Using the new cost function, VPR tries to place connected logic clusters close to each other, ideally within each other's tile connectivity domains.

3.3 Global Routing

The next step, global routing[2], which is required to interconnect clusters located outside of each other's tile connectivity domain, is performed using a custom tool. This tool views the whole routing as a set of "nets", where each net is a set of connections between a particular output of a certain cluster (or an input pad) and all its recipient clusters (and/or output pads). Each net is routed by adding an even number of routing inverters by configuring logic-free basic cells (there are at least $T - N$ of them in each tile) into one-input NOR gates, which are further called routing inverters (cells). Each pair of adjacent components in the net, i.e. any connection between the input cluster and the first routing inverter, a routing inverter and the next routing inverter, or the last routing inverter and output cluster, should be within the tile connectivity domain of each other (e.g., see the last step of Fig. 9).

[2]At the global routing stage, the specific location of basic cells inside the tile is not defined. For a perfect CMOL fabric with no defective nanodevices any placement inside the tile may be implemented; otherwise one can use the defect tolerance procedure described in Ref. 30.

(a)

```
INPUT:
A)  description of cluster connections (T-VPack format)
B)  mapping of clusters to tiles (VPR format)

START:
1:  Generate input_netlist from input data.
2:  Initialize counter (count) of unused basic cells for each tile.
3:  Create sorted_netlist by sorting input_netlist by the number of outputs
    (largest first) while sorting entries with the same number of outputs by
    the cost function (lowest first)
4:  For each net (curnet) from the sorted_netlist
5:    RouteNet(input, outputlist, count, true),
      where input, outputlist are the coordinates of an input and
      output clusters of curnet, respectively
6:  If there are any congestions (for some tiles count > T - N ) {
7:    For each tile (curtile) with count {
8:      Sort all nets (tile_sorted_netllist) which are routed using basic
        cells from curtile by the number of outputs while sorting entries
        with the same number of outputs by cost function (lowest first)
9:      For each net (curnet) from the tile_sorted_netlist {
10:       if (RouteNet(curnet_input_cluster,
                       curnet_outputlist_clusters, count, false) false
11:           Exit without success }
12: }} Exit with success
```

(b)

```
Function RouteNet(input, outputlist, count, congallowed)

INPUT:
A) input (input cluster coordinates)
B) outputlist (list of coordinates for output clusters)
C) count  (array of tile counters, each with the number of unused basic cells)
D) congallowed (Boolean value specifying whether the function should terminate
 (false) or not (true) if for some tile count > T-N )

START:
1:  For each tile (curtile) from input's connectivity domain {
      curtileoutputcount = 0
      curtileoutputlist = Ø
2:    For each output (curoutput) from outputlist {
3:      If(Hop(curtile, curoutput) + 1 = Hop(input, curoutput) ) {
4:        curtileoutputcount ++
5:        Add curoutput to curtileoutputlist }
6:      Among tiles with largest curtileoutputcount choose the tile
        which has the largest count and increment count for this tile
7:      If (count > T-N && congallowed is false)
8:        return false
9:      For each output (curoutput) from curtileoutputlist
10        If(Hop (curtile, curoutput)=0)
11            Delete curoutput from curtileoutputlist
12:     If(curtileoutputlist != Ø)
13:       if RouteNet(curtile, curtileoutputlist, count, congallowed) is false
14:           return false
15:   outputlist = outputlist - curtileoutputlist
16:   If(outputlist != Ø)
17:       if RouteNet(input, outputlist, count, congallowed) is false
18:           return false
19: } return true
```

Figure 8: The global routing pseudocode: (a) the whole algorithm and (b) the subroutine used to route a single net.

The formal description of the algorithm is presented in Fig. 8. Its general idea is that the nets are processed one at a time, and each one is routed with the minimal number of hops physically possible for a given placement, therefore ensuring the smallest number of routing cells for the successful circuit mapping. Moreover, if the net has more than one "output" cluster, the program tries to minimize the total number of routing cells by sharing them among different connections. Such problem is equivalent to finding the shortest-path Steiner tree [15], and, consequently, its exact solution is exponentially hard. In a context of VLSI design, several (both exact and heuristic) algorithms had been suggested for finding shortest path Steiner tree [7, 25]. Our method (formally presented in Fig. 8b) is close to the so-called RSA heuristic algorithm [25]. It is based on the recursive function (RouteNet) which, in a single iteration, finds the quasi-optimal position for the routing inverter in the connectivity domain of the input cluster (***input***) for the given set of output clusters (***outputlist***).

The algorithm can be best explained using the example shown on Fig. 9. Let us consider the case $\mathbb{A} = 5$ and suppose that the algorithm needs to route input cluster I to three output clusters O_1, O_2, and O_3 (corresponding to the tiles colored yellow and cyan, respectively, in Fig. 9). At first, for each pair of ***input*** and output clusters from ***outputlist***, the algorithm determines the minimal even number $Hop_{\min}$ of routing inverters required for routing. For the case shown in Fig. 9, these numbers are shown in red in Fig. 7. Then RouteNet function ranks all tiles of the ***input*** tile connectivity domain (step I in Fig. 9). The rank of a tile shows how many output clusters from ***outputlist*** can be routed to the ***input*** cluster with the minimal path, i.e. with $Hop_{\min}$ inverters, using a routing cell in the given tile. In the new iteration, the routing cell tile location (R_1), chosen randomly ("greedily") among the tiles with maximum rank, is considered as new ***input*** tile and the set of output clusters, which contributes to the rank (for R_1 in step II of Fig. 9, clusters O_1 and O_2) becomes new ***outputlist***, etc. Once these outputs have been routed (steps III, IV in Fig. 9) they are not considered during the ranking of the rest of output clusters, e.g., cluster O_3 (step V in Fig. 9).

Note that at this stage some congestion is possible, i.e. the number of routing cells assigned to a tile may be larger than the one physically available. Our algorithm tries to avoid the congestion by keeping an occupation counter of the total number of routing cells (*count*) requested by the algorithm in each tile. At the start of the routing procedure, the counter value is set to either zero (if the cluster, assigned to the given tile, is fully packed by T-VPack) or to the corresponding negative number in the opposite case (i.e. if the cluster has less than N logic cells). If for a certain iteration there are several tiles with the same rank, the preference is given to the tiles with the least utilized routing cells. Also, routing nets in a specific order, i.e. nets having fewer outputs and larger cost last, helps assigning routing cells more evenly throughout the tile array.

After processing all the nets, the algorithm makes the second step: it identifies the nets which were routed using tiles with the maximum number of requested routing cells ($count_{\max}$) and tries to reroute them by using the same algorithm as used at the first step, starting with the longest nets (in a hope that those nets have more paths to choose from, and hence may be rerouted around the congestion). This step yields a more even spread of routing resources and thus improves $count_{\max}$.

Finally, when $count_{\max}$ can be no longer improved, it is compared with $(T - N)$, and if $count_{\max}$ is larger, the whole design flow (starting from the T-VPack stage) is repeated with a reduced value of N (Fig. 6).[3] Alternatively, if $count_{\max} < T - N - \Delta$, where Δ is a small integer (set to 2 in our calculations), the cycle is repeated with an increased value of N. In case if $T - N - \Delta \le count_{\max} \le T - N$, the algorithm stops successfully.

4. PERFORMANCE CALCULATION

Generally, our performance analysis follows that described in detail in Ref. 30, with some simplifications. In particular, at this stage we have not yet optimized the nanowire pitch, but rather have simply calculated the performance for case $F_{\rm CMOS}$ = 45 nm, $F_{\rm nano}$ = 4.5 nm which seems technologically plausible at the initial stage of CMOL technology development [21]. Neither was the power supply voltage optimized; we have just accepted the value $V_{\rm DD} = 0.3$ V, which is in the ballpark of the results of $V_{\rm DD}$ optimization for the two circuits analyzed in Ref. 30 for these values of $F_{\rm CMOS}$ and $F_{\rm nano}$.

4.1 Area

We will show below that the typical current through molecular devices in the ON state is of the order of 1 μA. With a saturation current density of 1 mA/μm, typical for the long term CMOS projections [1], such current may be provided with a MOSFET channel as narrow as 1 nm. Hence, we can safely assume that all four transistors of the basic cell are of the minimum width.[4] Using SCMOS design rules [23], we estimate the smallest basic cell area to be about $A_{\rm cell} = 64(F_{\rm CMOS})^2$, i.e., $\beta_{min} \approx 4$. The additional area overhead associated with auxiliary circuitry such as clock buffers, peripheral logic for reconfiguration, etc. has not been taken into account at this stage, but is probably negligible[5].

4.2 Nanowires and Nanodevices

Even taking into account the additional diffusive scattering at nanowire surface [30], the estimated resistance between the center and the end of a nanowire fragment, of the length $(\beta F_{\rm CMOS})^2/F_{\rm nano} = 7.2$ μm, is about 20 KΩ. This resistance is negligible, because it is connected in series with that of a crosspoint device (Fig. 10), which is an order of magnitude larger, even in the ON state - see Section 5 below. With the wire capacitance per unit length, which was calculated earlier [30], to be close to 0.2 fF/μm, capacitance $C_{\rm wire}$ of the full nanowire fragment is about 3 fF.

[3] In some cases, e.g., for very large circuits with rich connectivity, $count_{\max}$ may be larger than $(T-N)$ even for $N = 1$, though this has never happened for any of the circuits from the considered benchmark set. Such situation would require a change in hardware parameters - say, a reduction of $F_{\rm nano}$ to increase a and hence the tile connectivity domain.

[4] The minimum-width CMOS inverter in the cell can provide a very large (> 20) fan-out without any latency degradation.

[5] For example, using results from our previous work [31], the overhead associated with peripheral logic for reconfiguration for 50 × 50 CMOL FPGA array of tiles should be less than 20%, while the shift-register reconfiguration circuitry which is discussed in Ref. 30, will make this overhead even much smaller.

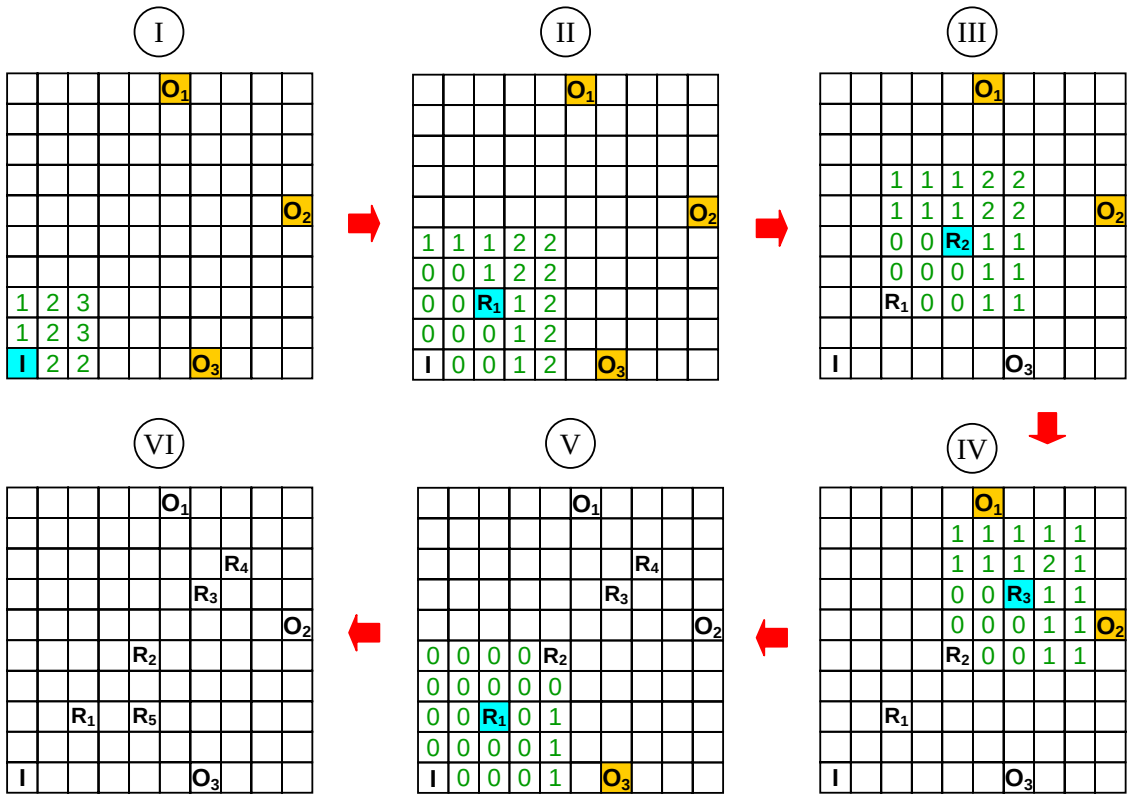

Figure 9: Example of global routing for a single net for the case $\mathbb{A} = 5$.

As in our previous calculations, the smallest acceptable resistance $R_{\rm ON}$ of a single molecular device in ON state is limited by the maximum manageable power density $p_{\rm max} = 200$ W/cm^2 [1]. For our estimates we have taken into account only the static power dissipated in nanodevices turned ON. (See Fig. 15a of Ref. 30 and its discussion for a justification of this assumption.) Hence, $R_{\rm ON}$ can be found from

$$R_{\rm ON} = \frac{DN_{\rm cell}(V_{\rm DD})^2}{2A_{\rm cell}p_{\rm max}}, \qquad (2)$$

where $N_{\rm cell}$ is the average number of crosspoint nanodevices turned ON per one basic cell, while D is the number of parallel molecules in each nanodevice. Based on the experimental data for self-assembled monolayers (see, e.g., Ref. 35), the footprint of a single molecule may be estimated as 0.25 nm^2; so for D we have used the value $F^2_{\rm nano}/(0.25\ {\rm nm}^2) \approx 80$.

Though we have not optimized $V_{\rm DD}$, we have checked that the ratio of resistances in the OFF and ON states provided by the second-order quantum effect of elastic cotunneling ($R_{\rm OFF}/R_{\rm ON} \approx R_{\rm ON}/R_{\rm Q}$) is less than the maximum value of this ratio, which is limited by the classical thermal activation ($R_{\rm OFF}/R_{\rm ON} \leq \cosh^2(V_{\rm DD}/2k_{\rm B}T)$), where $R_{\rm Q} \equiv \hbar/e^2 \approx 4.1$ kΩ is the quantum unit of resistance. Hence, for our parameters the cotunneling effect may be safely ignored.

4.3 Delay

In order to decrease the NOR gate delay τ_0 we minimize the signal voltage swing at the input of the CMOS inverter $V_{\rm in}$ by choosing an appropriate resistance of the pass transistor $R_{\rm pass} \backsimeq V_{\rm in}/V_{\rm DD} \times R_{\rm ON}/D$ (Fig. 10). More specifically, we use for $V_{\rm in}$ a condition similar to that used in Ref. 30, i.e.

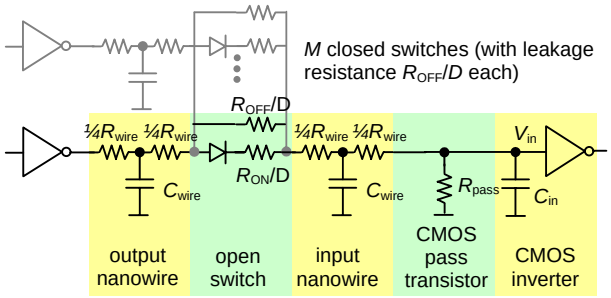

Figure 10: The equivalent circuit of a CMOL logic stage.

require a bit error rate of one gate less than 10^{-28}. (This corresponds to the mean time between failures of the whole VLSI circuit of about 10 000 hours [1].) For this, $V_{\rm in}$ has to be substantially larger than the r.m.s. of the sum of thermal noise (mostly contributed by the pass resistor) and the shot noise:

$$V_{\rm in} \geq \Delta V_{\rm T} = 23\sqrt{k_{\rm B}(T + T_{ef})/C_{\rm wire}} \approx 40\ {\rm mV}, \qquad (3)$$

where $T_{\rm ef} \approx (eV_{\rm in}/2k_{\rm B}) \times \coth(eV_{\rm DD}/2k_{\rm B}T) \approx 250$K is the effective temperature contributed by the shot noise. The digital noise resulting from the galvanic coupling of the input of basic cell to output of others basic cells through nanodevices turned OFF may be neglected since it is much less than the thermal and shot noise - for details, see Eq. (8) of Ref. 30.

Finally, for our set of parameters we could use the following simplified formula for gate delay:

$$\tau_0 \approx \ln(2I) \times (C_{\rm wire} R_{\rm ON}/D) \times (V_{\rm in}/V_{\rm DD}), \qquad (4)$$

where I is the circuit fan-in [30].

5. RESULTS AND DISCUSSION

We have applied our methods to analyze possible CMOL FPGA implementation of the Toronto 20 benchmark circuit set [2]. The largest value of the average nanodevice utilization factors among all circuits of the set has turned out to be about 1.5 nanodevices turned ON per basic cell. Plugging $N_{\rm cell}$ into Eq. (2), we find that $R_{\rm ON} = 21$ MΩ[6] and the ON resistance of a crosspoint nanodevice is $R_{\rm ON}/D = 260$ KΩ. These values justify the simplifications described in the previous sections.

Since most of the circuits benefited from mapping on NOR gates with large fan-in, we have chosen the maximum value allowed by T-VPack, $I = 7$. According to Eq. (4), the delay of a 1-input NOR gate turns out to be about 70 ps. The full delay of the considered circuits was calculated from the critical path, which had been found after circuit placement and global routing.

Table 1 summarizes the performance results for the benchmark circuits. Note that in contrast with earlier nanoelectronics work, the results for different circuits are obtained for the CMOL FPGA fabric with exactly the same operating conditions and physical structure for all the circuits, thus enabling a fair comparison with CMOS FPGA. For this comparison, the same benchmark circuits have been synthesized into cluster-based island-type logic block architecture [5]. This was done with the original T-VPack and VPR tools using the architecture designed for the optimal area-delay product, specifically the cluster size of 4, 4-input LUTs [3], and the VPR's default architecture file (4x4lut-sanitized.arch) with technology parameters corresponding to the 0.3 μm CMOS process. We had first found the worst case segment width required to route every circuit successfully, which has turned out to be 70 for pdc.blif circuit. Then, using an architecture with such segment width we have mapped and routed all circuits, and then extracted their delay and area (for the optimistic case of buffer sharing). Assuming the $1/s$ scaling for the delay and assuming the area of the minimum-width transistor to be $25(F_{\rm CMOS})^2$, we have obtained the results shown in the left part of Table 1. (As a sanity check, the delays before scaling are close to those obtained in Ref. 5 for CMOS FPGA with a similar architecture.)

Table 1 shows very clearly that CMOL FPGA circuits may be much denser than the purely CMOS FPGA circuits fabricated with the same CMOS design rules. The benchmark circuit area for CMOL FPGA also favorably compares with that implemented using the nanoPLA concept [9], taking into account the fact that the latter results had been calculated assuming a smaller nanowire half-pitch $F_{\rm nano} = 2.5$ nm.[7] Concerning speed performance, the delays calculated in this work for all benchmark circuits are comparable to those of their CMOS FPGA counterparts.

It is safe to expect that the improvement in area will be even larger if CMOL FPGA is used for much larger circuits, because the area of CMOS FPGA is always determined by the worst-case routing requirement. On the other hand, a distinctive feature of the CMOL FPGA fabric suggested in this work is that the same resources, basic cells, are used to perform both logic and interconnect functions.[8] Using the proposed CAD flow, the resources can be allocated flexibly according to the specific logic-to-routing ratio of the circuit. For example, in order to synthesize the relatively large pdc.blif circuit, only about 15% of the cells have been allocated for logic operation, while this number is about 60% for the smaller dsip.blif (Table 1).

Another evident resource for improvement is the optimization of $F_{\rm nano}$ and $V_{\rm DD}$. Next, an optimization of the maximum fan-in for each circuit may also give substantial results. For example, the area of the s298.blif circuit would be one half its current size, and its pre-mapped depth by 30% lower, if the maximum fan-in was 16 (rather that 7). Finally, our routing may be evidently improved further using better algorithm, e.g., described in Ref. 7. For instance, five routing cells is the best solution in the example shown in Fig. 9, while this number could be seven in a worst case, provided that the greedy algorithm picks different highest ranked tiles at each step.

While we have not performed a defect tolerance analysis in this work, the fact that the NOR gate locations inside the tiles are not fixed gives the gate placement the freedom similar to that employed in our first work to ensure high defect tolerance. In this analogy, the linear size of the tile has to be compared with the difference $(r - r')$ [30], where r and r' are physical and artificially confined connectivity radiuses, respectively. (For $a \gg 1$, r is related to our current parameter a as $r = a/\sqrt{2}$.) In Ref. 30 we have shown that even a modest difference $r - r' = 2$ allows the defect tolerance above 20% (for defects similar to "stuck-on-open" faults). Since in our current architecture the linear size of the tile is 4, equivalent to $r - r' \approx 3$, we may expect the defect tolerance of these circuits to be even higher. A verification of this hypothesis is one of our next goals.

To summarize, we believe that even the preliminary results presented in this report show a possible dramatic impact of the FPGA circuit transfer from CMOS to CMOL technology.

6. ACKNOWLEDGMENTS

The authors would like to thank Alan Mishchenko and Deming Chen for providing a pre-optimized benchmark set,

[6]Such resistances, well above $R_{\rm Q}$, may be readily implemented in molecular electronics - see, e.g., Ref. 33.

[7]Also, the nanoPLA results might be worse if all circuits had been mapped on a hardware fabric of fixed geometry, as we did for the CMOL architecture. Finally, the fabrication technology assumed in the nanoPLA architecture is much more demanding than CMOL, requiring (besides programmable diodes at crosspoints) nanoscale field-effect transistors, which are inherently irreproducible [21].

[8]For CMOS technology, similar ideas have been developed in several works. For example, Triptych FPGA architecture [6] is based on the universal cell which is used both for routing and logic operations. The authors of Refs. 8, 32 suggested to avoid the worst-case routing limitation in CMOS FPGA during mapping by deliberately underusing logic resources. However, both in Triptych FPGA (in which the universal cell has physically different CMOS hardware for routing and logic operations), and in the latter approach the CMOS circuitry utilization suffers. On the contrary, in CMOL FPGA, the CMOS subsystem utilization may be close to 100%.

Circuit	CMOS FPGA (F_{CMOS} = 45 nm)					CMOL FPGA (F_{CMOS} = 45 nm, F_{nano} = 4.5 nm, max fan-in = 7)						Comparison	
	Depth	LUTs	Array size (clusters)	Area (μm^2)	Delay (ns)	Depth	Array size (clusters)	N	Nano-devices	Area (μm^2)	Delay (ns)	A_{CMOS}/A_{CMOL}	$A_{nanoPLA}/A_{CMOL}$
alu4	7	1274	19×19	137700	5.1	23	22×22	5	9788	1004	4.0	137	0.28
apex2	8	1602	21×21	166050	6.0	26	21×21	6	11365	914	4.6	182	3.09
apex4	6	1147	34×34	414619	5.5	19	18×18	6	7781	672	3.6	617	0.58
bigkey	3	1810	22×22	193388	3.1	20	20×20	6	10207	829	2.7	233	1.82
clma	16	6779	42×42	623194	13.1	75	67×67	2	48746	9308	10.2	67	1.74
des	6	1263	19×19	148331	4.2	28	23×23	6	12610	1097	4.5	135	3.21
diffeq	14	987	16×16	100238	6.0	73	24×24	6	10799	1194	10.4	84	2.27
dsip	3	1362	19×19	148331	3.2	26	20×20	7	9905	829	3.4	179	1.63
elliptic	18	2142	24×24	213638	8.6	81	47×47	4	25415	4581	12.7	47	1.63
ex1010	8	4050	33×33	391331	9.0	43	41×41	3	28746	3486	5.7	112	0.28
ex5p	7	950	16×16	100238	5.1	27	20×20	4	6875	829	4.3	121	0.19
frisc	23	2320	25×25	230850	11.3	114	45×45	4	25869	4199	17.6	55	2.64
misex3	7	1178	18×18	124538	5.3	24	22×22	4	9211	1004	3.6	124	0.56
pdc	9	3901	32×32	369056	9.6	54	49×49	2	14841	4979	6.8	74	0.15
s298	15	1682	21×21	166050	10.7	45	20×20	4	10161	829	8.1	200	1.33
s38417	11	4773	36×36	462713	7.3	52	67×67	3	53156	9308	7.2	50	1.24
s38584	9	4422	35×35	438413	4.8	64	69×69	3	50275	9872	8.8	44	-
seq	7	1427	20×20	151369	5.4	23	25×25	4	11027	1296	4.0	117	1.15
spla	8	3331	30×30	326025	7.3	40	38×38	3	24808	2994	5.8	109	0.12
tseng	13	781	14×14	78469	6.3	75	24×24	6	4918	1194	11.5	66	2.48

Table 1: Performance results for Toronto 20 benchmark set.

as well as Jacob Barhen, Shamik Das, Andre DeHon, Dan Hammerstrom, Ramesh Karri, Phil Kuekes, Alex Orailoglu, Greg Snider, Mircea Stan, and Stan Willams for valuable discussions. Suggestions of anonymous Referee #4 have been also extremely useful. This work was supported in part by AFOSR, DTO, and NSF.

7. REFERENCES

[1] *International Technology Roadmap for Semiconductors. 2003 Edition, 2004 Update*. Available online at http://public.itrs.net/.

[2] *FPGA place-and-route challenge*. 1999. Available online at http://www.eecg.toronto.edu/∼vaughn/challenge/challenge.html/.

[3] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Trans. on VLSI*, 12(3):288–298, 2004.

[4] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Proc. of* 7^{th} *Int. Work. on Field-Programmable Logic and App.*, pages 213–222, London, UK, September 1997.

[5] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for deep-submicron FPGAs*. Kluwer Int. Series in Eng. and Comp. Science 497. Kluwer Academic, Boston; London, 1999.

[6] G. Borriello, C. Ebeling, S. A. Hauck, and S. Burns. The Triptych FPGA architecture. *IEEE Trans. on VLSI*, 3(4):491–501, 1995.

[7] J. Cong, A. B. Kahng, and K. S. Leung. Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design. *IEEE Trans. on Computer-Aided Design*, 17(1):24–39, 1998.

[8] A. DeHon. Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization). In *Proc. of* 7^{th} *Int. Symp. on FPGAs*, pages 69–78, Monterey, CA, Febrary 1999.

[9] A. DeHon. Design of programmable interconnect for sublithographic programmable logic arrays. In *Proc. of* 13^{th} *Int. Symp. on FPGAs*, pages 127–137, Monterey, CA, Febrary 2005.

[10] A. DeHon and K. Likharev. Hybrid CMOS/nanoelectronics digital circuits: Devices, architectures, and design automation. In *Proc. of Int. Conf. on Computer-Aided Design*, pages 375–382, San Jose, CA, November 2005.

[11] S. Fölling, Ö. Türel, and K. K. Likharev. Single-electron latching switches as nanoscale synapses. In *Proc. of Int. Joint Conf. on Neural Networks*, pages 216–221, Washington, DC, July 2001.

[12] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H. S. P. Wong. Device scaling limits of Si MOSFETs and their application dependencies. *Proc. of the IEEE*, 89(3):259–288, 2001.

[13] S. C. Goldstein and M. Budiu. NanoFabrics: Spatial computing using molecular electronics. In *Proc. of*

28^{th} *Int. Symp. on Computer Architecture*, pages 178–191, Götenberg, Sweden, July 2001.

[14] S. Heo, R. Krashinsky, and K. Asanović. Activity-sensitive flip-flop and latch selection for reduced energy. In *Proc. of* 19^{th} *Conf. on Advanced Research in VLSI*, Salt Lake City, UT, March 2001.

[15] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Annals of discrete mathematics; 53. North-Holland, Amsterdam; London, 1992.

[16] P. J. Kuekes, G. S. Snider, and R. S. Williams. Crossbar nanocomputers. *Scientific American*, 293(5):72–76, 2005.

[17] P. J. Kuekes, D. R. Stewart, and R. S. Williams. The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. *J. Appl. Phys.*, 97(3):034301, 2005.

[18] J. H. Lee and K. K. Likharev. CMOL CrossNets as pattern classifiers. In *Proc. of* 8^{th} *Int. Work-Conference on Artificial Neural Networks*, pages 446–454, Barcelona, Spain, June 2005.

[19] K. Likharev, A. Mayr, I. Muckra, and O. Turel. Crossnets - high-performance neuromorphic architectures for CMOL circuits. *Ann. NY Acad. Sci.*, 1006:146–163, 2003.

[20] K. K. Likharev. Electronics below 10 nm. In *Nano and Giga Challenges in Microelectronics*, pages 27–68. Elsevier, Amsterdam, 2003.

[21] K. K. Likharev and D. B. Strukov. CMOL: Devices, circuits, and architectures. In G. Cuniberti, G. Fagas, and K. Richter, editors, *Introducing Molecular Electronics*, pages 447–478. Springer, Berlin, 2005.

[22] M. Masoumi, F. Raissi, M. Ahmadian, and P. Keshavarzi. Design and evaluation of basic standart encryption algorithm modules using nanosized complementary metal-oxide-semiconductor-molecular circuits. *Nanotechnology*, 17:89–99, 2006.

[23] C. Mead and L. Conway. *Introduction to VLSI systems*. Addison-Wesley, Reading, MA; London, 1980.

[24] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic. *Digital integrated circuits: A design perspective*. Pearson Education, Upper Saddle River, NJ, 2nd edition, 2003.

[25] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear steiner arborescence problem. *Algorithmica*, 7(2-3):277–288, 1992.

[26] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, 1992.

[27] G. Snider, P. Kuekes, and R. S. Williams. CMOS-like logic in defective, nanoscale crossbars. *Nanotechnology*, 15(8):881–891, 2004.

[28] 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. *Proc. of the IEEE*, 91(11):1940–1957, 2003.

[29] D. B. Strukov and K. K. Likharev. Architectures for defect-tolerant CMOL memories. Paper in preparation, 2006.

[30] D. B. Strukov and K. K. Likharev. CMOL FPGA: A cell-based, reconfigurable architecture for hybrid digital circuits using two-terminal nanodevices. *Nanotechnology*, 16:888–900, 2005.

[31] D. B. Strukov and K. K. Likharev. Prospects for terabit-scale nanoelectronic memories. *Nanotechnology*, 16:137–148, 2005.

[32] M. Tom and G. Lemieux. Logic block clustering of large designs for channel-width constrained FPGAs. In *Proc. of* 42^{nd} *Design Automation Conference*, pages 726–731, San Diego, CA, June 2005.

[33] J. Tour. *Molecular Electronics*. World Scientific, Singapore, 2003.

[34] O. Türel, J. H. Lee, X. L. Ma, and K. K. Likharev. Neuromorphic architectures for nanoetectronic circuits. *Int. J. Circ. Theory App.*, 32(5):277–302, 2004.

[35] N. Zhitenev, W. Jiang, A. Evbe, Z. Bao, E. Garfunkel, D. M. Tennant, and R. Cirelli. Control of topography, stress, and diffusion of molecule-metal interface. Preprint, 2005.