CMOL FPGA circuits

Dmitri B. Strukov and Konstantin K. Likharev

Stony Brook University
Stony Brook, NY 11794-3800, U.S.A
Email: dstrukov@ic.sunysb.edu, klikharev@cc.notes.sunysb.edu

***Abstract*— This paper describes an architecture of FPGA-like fabric 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. We have developed a custom set of tools for CMOL FPGA circuit design automation, and used it for the evaluation of performance of these circuits for the Toronto 20 benchmark set, so far without optimization of several parameters including the power supply voltage, nanowire pitch and maximum NOR fan-in. 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.**

I. Introduction

It is now believed that the growing problems with scaling of CMOS technology [1]–[3] may be only overcome by a radical paradigm shift from lithography-based fabrication to the so-called "bottom-up" approach - see, e.g., Ref. [4]. 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. [5]–[15], and also recent reviews [16]–[19]. 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/nano-device hybrids are "CMOL" circuits [3], [16]. 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 a-few-nm-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 [16], [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 [16].

As Figs. 1, 2 show, 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 turned by angle $\alpha = \arcsin(F_{\rm nano}/\beta F_{\rm CMOS})$, where $F_{\rm nano}$ is the nanowiring half-pitch.[1] This approach allows a unique access to any nanodevice, even if $F_{\rm nano} \ll F_{\rm CMOS}$ - see Refs. [7], [16] for a detailed discussion of this point.

Earlier we showed that CMOL circuits may be used to build several types of highly defect-tolerant circuits including terabit-scale memories [15], [16], [21] and mixed-signal neuromorphic circuits capable of advanced information processing, e.g., fast classification of large patterns such as few-megapixel images [22], [23]. However, the most important application of CMOL technology may be in reconfigurable Boolean logic circuits [6], [7] whose structure resembles the so-called cell-based FPGAs [24]. The goal of this report is to describe the CMOL FPGA concept and review our most recent results on design automation and performance analysis of these circuits.

II. Hardware Architecture

CMOL FPGA circuits are formed on a uniform mesh of square-shaped "tiles" (Fig. 2). Each tile consists of a shell of T basic cells (Fig. 2b) surrounding a single latch cell of a larger area (Fig. 2c). (For the considered parameters, and no defective basic cells inside the tile, $T = 12$.)

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

[1] Actually, the most intrinsic parameter of CMOL circuits is the integer $a \equiv 1/\tan\alpha$ which determines the range of CMOS cell interaction. For fixed fabrication technology parameters $F_{\rm CMOS}$, $F_{\rm nano}$ and $\beta_{\rm min}$, the lower bound on a is given by equation $a^2 > (\beta_{\rm min} F_{\rm CMOS}/F_{\rm nano})^2 - 1$.

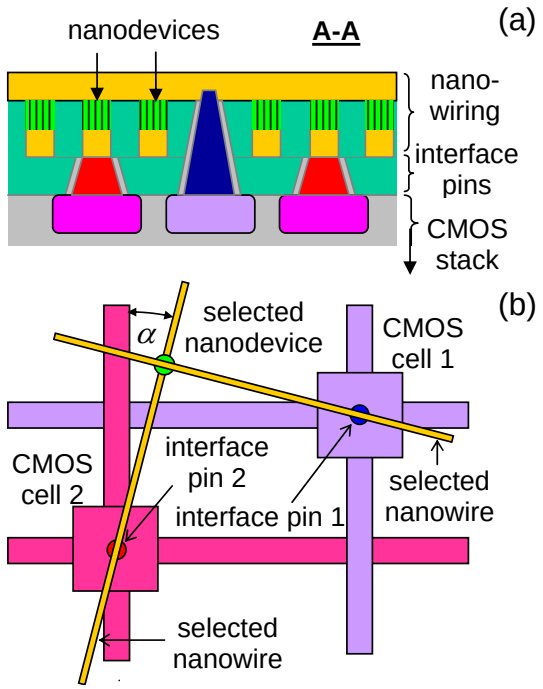

Fig. 1. Low-level structure of the generic CMOL circuit: (a) a schematic side view, and (b) the concept of addressing a particular nanodevice.

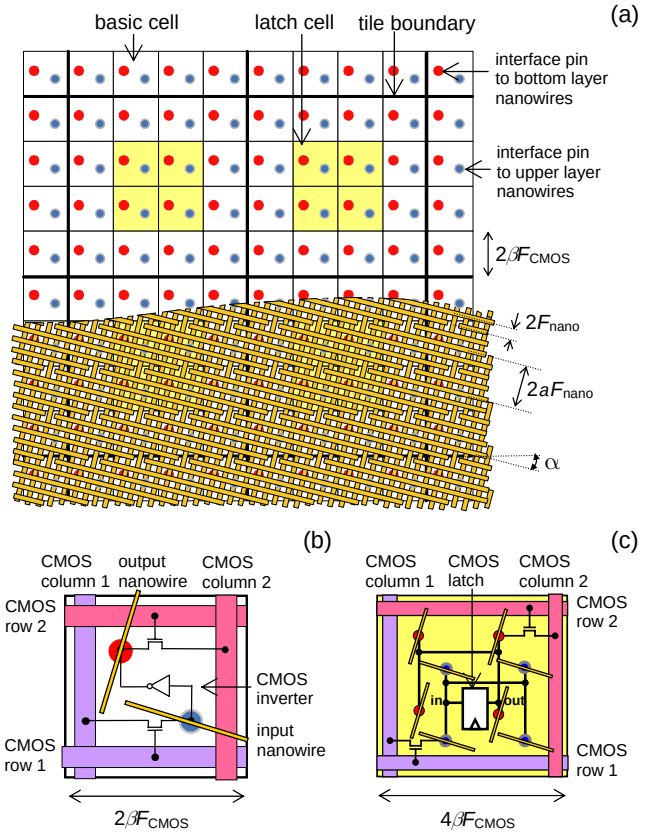

Fig. 2. CMOL FPGA: (a) The fragment of the fabric for the particular case $a = 4$, and the structures of (b) the basic cell and (c) the latch cell.

are turned off, and the pass transistors may be used for setting the binary state of each molecular device, just as in CMOL memories [15], [16], [21]. 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 "cell connectivity domain" [7]. Figure 3 shows 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.

CMOS layout estimates assuming a compact layout have shown that the latch cell requires an area approximately four times larger than that of the basic cell. As a result, latch/logic resource ratio comparable to those of the conventional FPGAs. In fact, the 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 [25].

III. Performance Model

In general, it is possible to optimize the power supply voltage V_{DD} and nanowire pitch F_{nano} to achieve the best performance of a CMOL FPGA circuit (at fixed power). However, for the considered benchmark circuits we have not yet performed such optimization, but rather assumed the values $F_{CMOS} = 45$ nm, $F_{nano} = 4.5$ nm which seem technologically plausible at the initial stage of CMOL technology development [16]. Similarly we have accepted

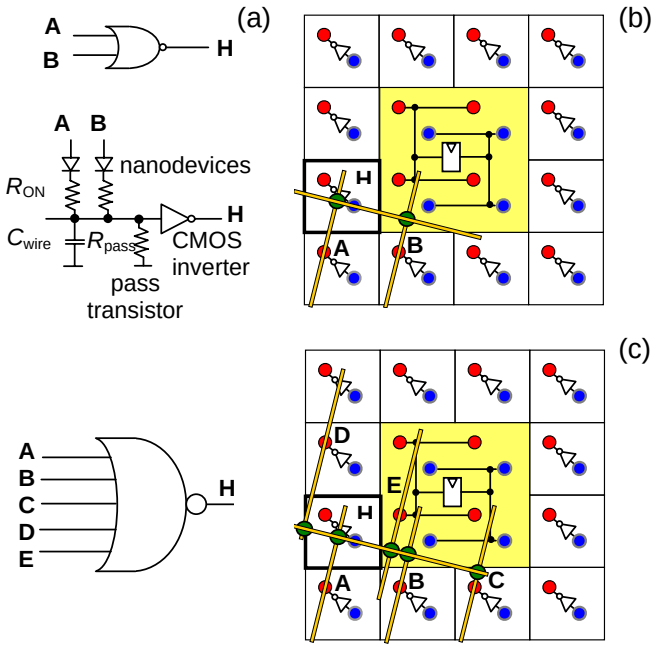

Fig. 3. NOR gate: (a) equivalent circuit and (b) physical implementation of a fan-in-two NOR gate in CMOL, and (c) same for a fan-in-five NOR gate. Note that for the sake of clarity, panels (b) and (c) show only the nanowires used for the gate operation, and only those nanodevices on the input nanowire of cell H which are set to the ON state.

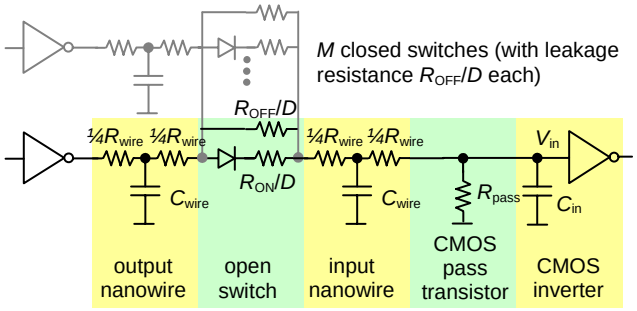

Fig. 4. The equivalent circuit of a CMOL logic stage.

the value $V_{DD} = 0.3$ V which is in the ballpark of the results of V_{DD} optimization for the two circuits analyzed in Ref. [6] for these values of F_{CMOS} and F_{nano}.

The typical current through molecular devices in the ON state, resulting from the performance optimization, is of the order of 1 μA [6]. 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. Using SCMOS design rules [24], we estimate the smallest basic cell area to be about $A_{cell} = 64(F_{CMOS})^2 \approx 0.13$ (μm)^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 [7].

From equivalent circuit (Fig. 4) the gate delay is approximately:

$$\tau_0 \approx \ln(2I) \times (C_{wire} R_{ON}/D) \times (V_{in}/V_{DD}), \qquad (1)$$

Here C_{wire} (equal to $\sim$ 3 fF in our case) is a capacitance of the full nanowire fragment, R_{ON}/D is a resistance of the nanodevice at the crosspoint in the ON state[2], V_{in} is a voltage swing on the input of the CMOS inverter, and I is the circuit fan-in [7]. Note that nanowire resistance R_{wire} is neglected since it is typically much smaller than R_{ON}/D.

To reduce gate delay we minimize the signal voltage swing V_{in} and resistance R_{ON}/D. The former parameter can be reduced by choosing an appropriate resistance of the pass transistor $R_{pass} \approx V_{in}/V_{DD} \times R_{ON}/D$ (Fig. 4). We have shown that even for a very strict requirement for the bit error rate V_{in} can be as small as 40 mV [7].

The smallest acceptable resistance R_{ON} is limited by the maximum manageable power density $p_{max} = 200$ W/cm^2 [1]. Since the power consumption in our case is mostly contributed by its static component [6], R_{ON}/D can be found from

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

where N_{cell} is the average number of crosspoint nanodevices turned ON per one basic cell. For the considered benchmark circuits the maximum value of N_{cell} is about 1.6 (Table I). Hence from Eq. (2) we find that the $R_{ON}/D \approx 280$ KΩ. Such crosspoint resistance, for example, may be readily implemented in molecular electronics, especially assuming that each crosspoint can house tens of molecular devices [7]. Finally, from Eq. (1) the delay of the inverter (NOR-1 gate) is about 80 ps.

[2] D is the number of parallel molecular-scale devices (each with R_{ON} resistance) in a single crosspoint nanodevice.

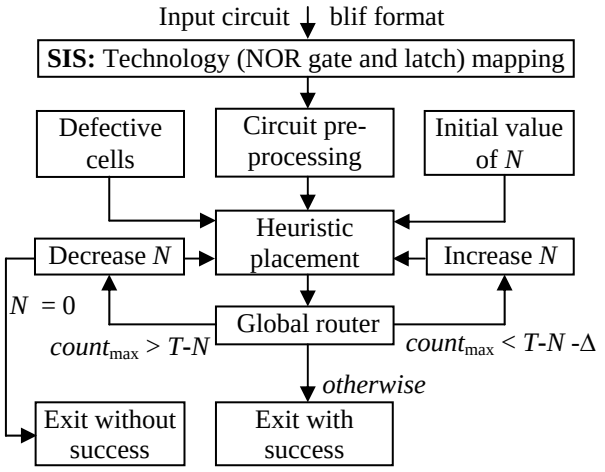

Fig. 5. CMOL FPGA design flow.

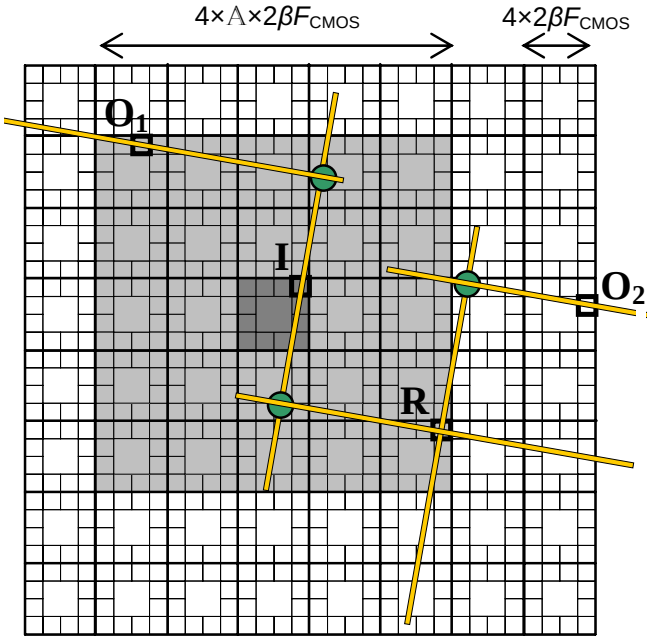

Fig. 6. Tile connectivity domain: Any cell of the central tile (shown dark gray) can be connected with any cell in the tile connectivity domain (shown light gray) via one nanowire-nanodevice-nanowire link (e.g., cells I and O $_1$). Cells outside of each other's tile connectivity domain (e.g., I and O $_2$) can be connected with additional routing inverters (e.g., R). Note that nanowire width and nanodevice size are boosted for clarity. For example, for the considered CMOL parameters, 1600 crosspoint nanodevices may fit on one CMOS cell area.

IV. Design Automation

The main idea of the proposed design flow for CMOL FPGAs (Fig. 5) is to reserve some number of basic cells

$(T - N)$ inside each tile for routing purposes, while use the rest (N) cells for logic during the placement step. The placer tries to put gates into such locations (with maximum one latch and N NOR gates per tile) so that their interconnect is local or, equivalently, is within tile connectivity domain of each other.[3] At the global routing step, idle cells inside each tile are used to interconnect global connections. If there is a congestion after the global routing step, i.e. the number of requested basic cells during routing $count_{\max}$ is larger than the actual number of idle cells $T - N - \Delta$ (here Δ is parameter which allows to trade-off the number of iterations with the mapping quality), then we decrease N and repeat the flow again until there is no congestion.

More specifically, every circuit is first mapped into the network of NOR gates (so far with a rather artificial maximum fan-in of 7) and latches, using the SIS package [26]. After that, at the circuit processing step, all inverters (1-input NOR gates) are removed from the circuit. (For the considered benchmark set, such inverters comprise about 28% of the total number of gates on the average.) The idea behind the processing step is that inverters which are synthesized by SIS are not different logically from the routing inverters and can be more efficiently (in terms of performance and CAD running time) used during the global routing step (as opposed to the situation when they are treated as logic gates and have fixed positions after the placement step). Instead of the removed inverters, every net is assigned a "polarity" property, which is just an additional information for a placer and global router to specify whether an odd or even number of routing inverters should be used when interconnecting nets.

The placement algorithm follows very closely the heuristic one used in the VPR tool [25], [27]. It is based on a simulated annealing algorithm with some additional techniques employed for a faster and better solution. According to our preliminary results, the best area and delay in mapping circuits to CMOL FPGA can be achieved using the timing-driven placement algorithm with $\lambda = 0.5$, the criticality exponent equal to 1, the time-analysis interval of about 10000 successful swap/moves, and with similar temperature and D_{lim} schedules. (For a detailed explanation of these parameters and the algorithm, see Ref. [27], in particular Fig. 2.)

In this algorithm, the wiring cost is calculated as a sum of costs for all connections in the circuit, with a cost of a single connection obtained using the function *LogicHop* from Fig. 8. Here x, y is the location of a tile with input/ouput gate of the considered connection, while $\mathbb{A}$ is a linear size of the tile connectivity domain, which is, for the CMOL parameters accepted in this paper, equal to 9. The timing cost was calculated according to the Eqs. (6-8) of Ref. [27], using the slack (normalized with respect to inverter delay), also obtained through Eqs. (4), (5) of that work.

[3]Here, similarly to the cell connectivity domain [6], the tile connectivity domain of a given tile is defined as such fabric fragment that any cell within it can be connected to any cell of the initial tile directly, i.e. via one nanowire-nanodevice-nanowire link (Fig. 6). Note that all tile connectivity domains are similar and have square shape.

```
INPUT:
A)  mapping of gates to tiles
B)  map of defective cells (def)
START:
1:  Generate input_netlist from input mapping
2:  Initialize counter (count) of unused basic cells for each tile and
    adjusting accordingly to def
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:    RouteNetGreedy(input, outputlist),
      where input, outputlist  are the coordinates of an input and
      output gates of curnet, respectively
6:  If there are congestions (for some tiles count  > T - N )  {
7:    For each tile (curtile)  {
8:      For each curgate from curtile {
9:        If (curgate from curtile is inverter and fanout = 1) {
10:         Remove curgate and all consecutive inverters with
            (fanout =1) connected to curgate's input and output
11:         If (RouteNetExhaustively(curnet) is false)
12:           Exit without success
13: }}}} Exit with success
```

Fig. 7. Global routing algorithm

The next operation, global routing, which is needed to connect the gates which are not within tile connectivity domains of each other, performed in two steps (Fig. 7). At the first step the algorithm connects global nets by allocating the routing inverters, so far without any tile capacity limitations. In this step, each connection of the net is routed with the smallest possible number of inverters. Moreover, if the net has more than one output, the algorithm tries to minimize the number of routing inverters by sharing them among connections of the same net. Since this problem is equivalent to finding the shortest-path Steiner tree [28], which is exponentially hard, our algorithm is heuristic.

The algorithm is formally presented in Fig. 8. It is based on the recursive function (*RouteNetGreedy*) which, in a single iteration, finds the quasi-optimal position for the routing inverter in the tile connectivity domain of the input gate (***input***) for the given set of output gates (***outputlist***).

The algorithm can be better explained using the example shown in Fig. 9. Let us consider the case $\mathbb{A} = 5$ and suppose that the algorithm needs to route input gate I to three output gates O_1, O_2, and O_3 (corresponding to the tiles colored yellow and cyan, respectively. At first, for each pair of ***input*** and output gates from ***outputlist***, the algorithm determines the minimal number $Hop_{\min}$ of inverters required for routing, taking into account the polarity of each connection (shown with "+" and "-" superscripts). Then *RouteNetGreedy* function ranks all tiles of the ***input*** tile connectivity domain (step I in Fig. 9). The rank of a tile shows how many output gates from ***outputlist*** can be routed to the ***input*** gate 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 "greedily" among the tiles with maximum rank, is considered as a new ***input*** tile, while the set of output gates, which contributes to the rank (e.g., for R_3 in step III of Fig. 9, gates O_1 and O_2) becomes new ***outputlist***, etc. Once these outputs have been routed (step IV in Fig. 9) they are not considered during the ranking of the

```
Function RouteNetGreedy (input, outputlist)
1:  For each tile (curtile) from input's connectivity domain {
      curtileoutputcount = 0
      curtileoutputlist = Ø
2:    For each output (curoutput) from outputlist {
3:      If(PhysicalHop(curtile, curoutput) + 1 =
          LogicalHop(input, curoutput, curoutput's polarity) ) {
4:        curtileoutputcount ++
5:        Add curoutput to curtileoutputlist }
6:      Among tiles with largest curtileoutputcount choose the one
        which has the largest count and increment count for this tile
7:      For each output (curoutput) from curtileoutputlist
8:        If(LogicalHop(curtile, curoutput, curoutput 's polarity)=0)
9:          Delete curoutput from curtileoutputlist
10:     If(curtileoutputlist != Ø)
11:       RouteNetGreedy(curtile, curtileoutputlist)
12:   outputlist = outputlist - curtileoutputlist
13:   If(outputlist != Ø)
14:     RouteNetGreedy(input, outputlist) }

Function RouteNetExhaustively (curnet)
1:  Calculate slack for curnet
2:  For curhops = LogicalHop(curnet's input, curnet's output, curnet's polarity)
    to slack steps of 2 {
3:    Find all tiles (tilelist) which might be used to route curnet using curhops
      routing inverters
4:      For each tile (curtile) from tilelist
5:        For hops = PhysicalHop(curnet's output, curtile) to
          slack - PhysicalHop(curnet's input, curtile)
6:          RankTile(curtile, hops, curnet's output, count)
7:    Choose tile (besttile) in curnet's input tile domain with the largest rank
8:    If(rank != 0) {
9:      Route curnet by traversing tiles using bestpath starting from besttile
10:     Return true
11: }} Return false

Function RankTile(inputtile, hops, outputtile)
1:  Let tiledomain be tile domain of inputtile
2:  Eliminate all tiles (curtile) from tiledomain such that
    PhysicalHop(curtile, outputtile) ≥ hops
3:  For each tile (curtile) from tiledomain {
4:    If(rank (curtile, hops -1) is undefined) {
5:      If(hops =1) {
6:        rank (curtile, hops -1) = count (curtile)
7:        bestpath (curtile, hops -1) = outputtile
8:      } Else RankTile(curtile, hops -1, outputtile) }
9:  Choose tile (besttile) from tiledomain with largest rank (hops -1)
10: If(rank (besttile, hops -1) = 0 or count (inputtile) = 0 ) {
11:   rank (inputtile, hops) = 0
12: Else {
13:   bestpath (inputtile, hops) = besttile
14:   rank (inputtile, hops) = rank (besttile, hops -1) + count (inputtile) }
15: Return rank (inputtile, hops)

Function PhysicalHop (tile1, tile2)
1:  Let x1, y1 and x2, y2 be tile1 and tile2 coordinates, respectively
2:  Return floor(2*max(abs(x1-x2),abs(y1-y2)-1)/(𝔸-1))

Function LogicalHop (tile1, tile2, polarity)
1:  hops = PhysicalHop (tile1, tile2)
2:  If((hops is even and polarity is odd) or (hops is odd and polarity is even)) {
3:    Return hops + 1
4:  } Else Return hops
```

Fig. 8. Subroutines used in the placement and global routing algorithms.

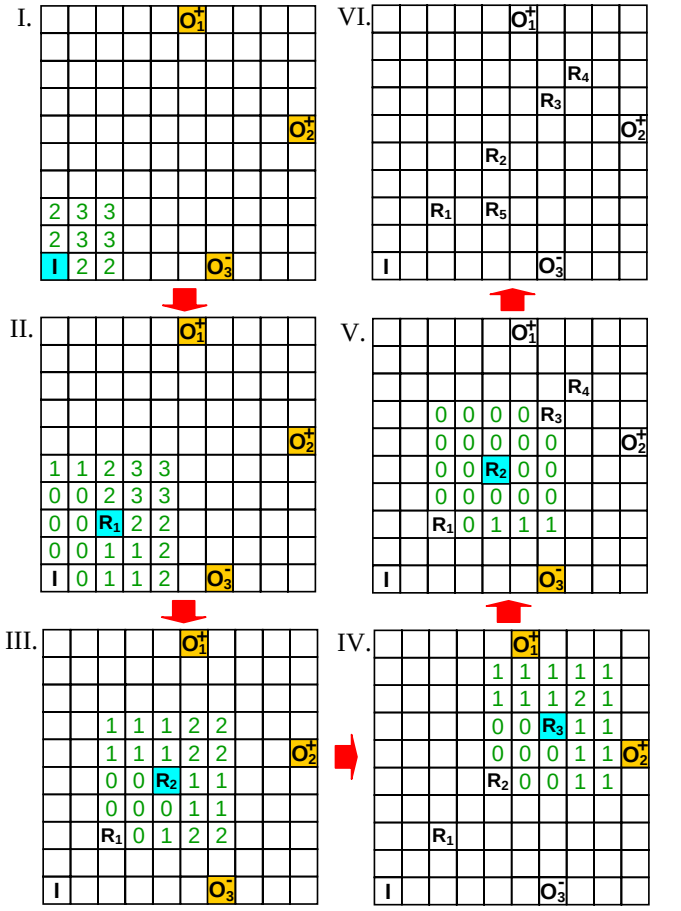

Fig. 9. Example of global routing for a single net for the case $\mathbb{A} = 5$.

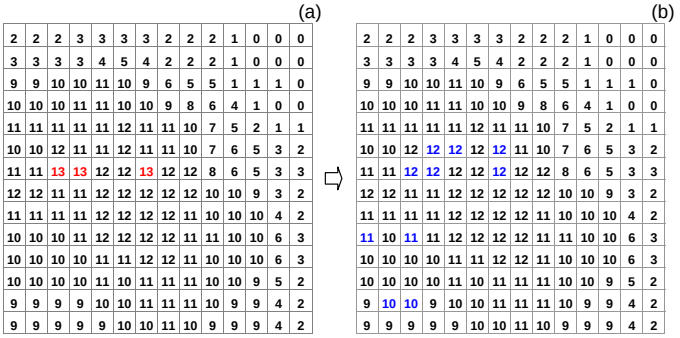

Fig. 10. Tile capacity for s298.blif circuit: (a) after step 1 and (b) after step 2 of the global routing algorithm.

rest of output gates, e.g., gate O_3 (step V in Fig. 9).

Note that after the first step some congestion is possible (Fig. 10a), i.e. the number of routing cells assigned to a tile may be larger than the one physically available. Congestions are dealt with at the second step, using recursive function *RouteNetExhaustively* (Fig. 8). The idea is to reroute exhaustively the nets whose routing inverters are located in the congested tiles. To make this step more powerful, we perform slack analysis of the congested nets, and (provided that the critical path stays the same) allow for more flexible rerouting with the number of routing inverters larger than its minimum.

Finally, after finding the largest N enabling successful mapping, all circuits are functionally verified. This is done by first converting the mapped circuit into the blif format, and then comparing them with the original circuit with the help of the ABC verification tool [29].

V. Results and Discussion

We have simulated performance of the Toronto 20 benchmark circuit set [30] mapped on CMOL FGPA. Table I summarizes the performance results for each of 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 mapped into the cluster-based island-type logic block CMOS architecture [25]. This was done with T-VPack and VPR tools [25] using the architecture designed for the optimal area-delay product, specifically a cluster size of 4, 4-input LUTs [31], 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 the 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 the area of the minimum-width transistor to be $25(F_{\text{CMOS}})^2$, we have obtained the results for the CMOS FPGAs shown in Table I.

The table 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 [5], taking into account the fact that the latter results had been calculated assuming a smaller nanowire half-pitch $F_{\text{nano}} = 2.5$ nm.[4] Concerning speed performance, the delays calculated 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 the CMOL FPGA technology 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 our CMOL FPGA fabric is that the same resources, basic cells, are used to perform both logic and interconnect functions. 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 30% of the cells have been allocated for logic operation, while this number is about 75% for the smaller dsip.blif (Table I). Furthermore, there is still some room for a further improvement via, e. g., optimization of F_{nano}, V_{DD}, and increasing the maximum fan-in [7].

Finally, Table I also shows preliminary results of the defect tolerance analysis (so far for a rather limited number of trials). For each circuit, in each trial, defective cells have been generated with the same probability q. Keeping the same defect map in each trial we have performed the mapping, finding the maximum possible N for each case. For example,

[4]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 [3], [16].

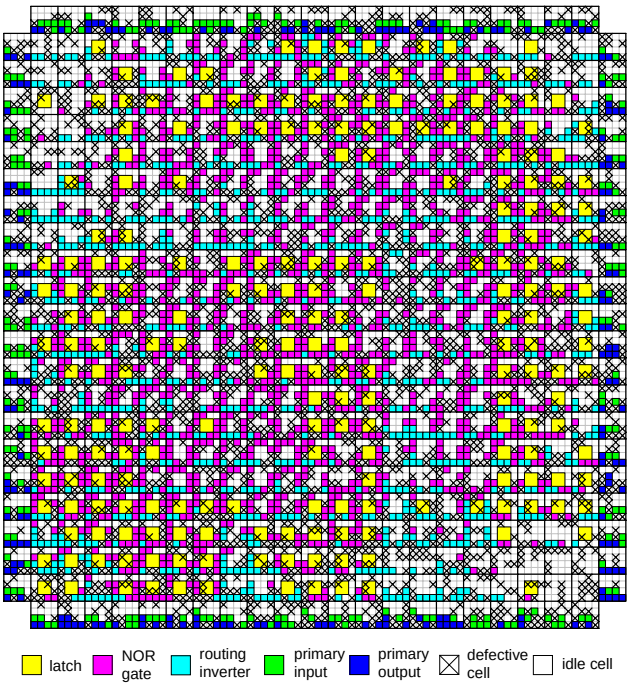

Fig. 11. Example of CMOL mapping: dsip.blif circuit of the Toronto 20 set, mapped on the (21+2)×(21+2) tile array with 30% defective cells. The additional layer of tiles at the array periphery is used exclusively for I/O functions.

Fig. 11 shows successful mapping for dsip.blif circuit on the fabric with $q = 0.3$. Table I shows that the average swelling of the circuit area is rather limited: only about 20% and 80% for, respectively, $q = 0.1$ and $q = 0.3$. This means, in particular, that faulty interface pins, nanowires and/or CMOS cells can be very effectively tolerated. On the other hand, the tolerance to stuck-on-close crosspoint defects is rather low (equivalent to about 0.02% of defective nanodevices for $q = 0.3$), so that some other defect tolerance mechanism should be used to reduce the effects of such faults. On the other hand, the preliminary defect tolerance results with respect to stuck-on-open kind of defects in crosspoint nanodevices are very encouraging. For example, we have shown that 32-bit Kogge Stone adder can be successfully mapped on CMOL FPGA with up to 50% of such defects [6].

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

VI. Acknowledgments

Useful discussions of various aspects of CMOL FPGA with R. Brayton, S. Chatterjee, S. Das, A. DeHon, D. Hammerstrom, R. Karri, P. Kuekes, G. Lemieux, A. Mishchenko, A. Orailoglu, T. Polishchuk, V. Polishchuk, G. Snider, M. Stan, and R. S. Williams are gratefully acknowledged. The work has been supported in part by AFOSR, DTO, and NSF.

References

[1] *International Technology Roadmap for Semiconductors. 2005 Edition*, available online at http://public.itrs.net/.

Circuit	CMOL FPGA (F_{CMOS} = 45 nm, F_{nano} = 4.5 nm)									CMOS FPGA (F_{CMOS} = 45 nm, no defects)		Comparison (no defects)	
	No defects					10% defective cells		30% defective cells					
	Array size (tiles)	N	Nano-devices	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	A_{CMOS}/A_{CMOL}	$A_{nanoPLA}/A_{CMOL}$
alu4	19 × 19	7	5468	749	3.3	915	3.3	1745	3.9	137700	5.1	184	0.37
apex2	20 × 20	7	6241	830	4.2	1004	4.1	1297	4.2	166050	6.0	200	3.41
apex4	16 × 16	7	4203	531	3.0	600	3.0	915	3.0	414619	5.5	781	0.73
bigkey	18 × 18	9	6589	672	1.8	749	1.8	1098	1.9	193388	3.1	288	2.25
des	22 × 22	7	6955	1004	3.5	1098	3.5	1403	3.5	148331	4.2	148	3.51
diffeq	20 × 20	9	6279	830	7.3	830	7.3	1195	7.3	100238	6.0	121	3.27
dsip	17 × 17	9	5392	600	2.1	672	2.1	915	2.1	148331	3.2	247	2.25
elliptic	34 × 34	7	16403	2399	9.7	3488	10.0	11362	11.6	213638	8.6	89	3.12
ex1010	29 × 29	6	13540	1745	4.1	1994	4.0	2996	4.5	391331	9.0	224	0.56
ex5p	16 × 16	6	3551	531	3.3	600	3.3	749	3.3	100238	5.1	189	0.30
frisc	35 × 35	6	16393	2542	13.6	2841	13.6	5187	14.3	230850	11.3	91	4.36
misex3	17 × 17	7	4798	600	2.7	749	2.7	1004	2.9	124538	5.3	208	0.93
pdc	41 × 41	4	21804	3488	5.4	4982	6.0	9594	7.7	369056	9.6	106	0.22
s298	15 × 15	7	5891	467	6.9	531	7.2	749	7.5	166050	10.7	356	2.36
s38417	55 × 55	4	32430	6277	5.9	6980	6.3	9038	6.5	462713	7.3	74	1.84
s38584	45 × 45	5	27360	4202	5.9	4781	5.9	6050	5.9	438413	4.8	104	-
seq	21 × 21	6	6003	915	3.4	1004	3.4	1513	3.7	151369	5.4	165	1.63
spla	38 × 38	4	18562	2996	5.4	4390	6.1	8499	8.0	326025	7.3	109	0.12
tseng	20 × 20	8	5610	830	8.7	830	8.7	1098	8.7	78469	6.3	95	3.57

TABLE I

PERFORMANCE RESULTS FOR TORONTO 20 BENCHMARK SET.

[2] D. J. Frank, *et al.*, "Device scaling limits of Si MOSFETs and their application dependencies," *Proc. of the IEEE*, vol. 89, no. 3, pp. 259–288, 2001.

[3] K. K. Likharev, "Electronics below 10 nm," in *Nano and Giga Challenges in Microelectronics*. Amsterdam: Elsevier, 2003, pp. 27–68.

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

[5] A. DeHon, "Nanowire-based programmable architectures," *J. Emerg. Technol. Comput. Syst.*, vol. 1, no. 2, pp. 109–162, 2005.

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

[7] ——, "A reconfigurable architecture for hybrid CMOS/Nanodevice circuits," in *FPGA'06*. New York, NY: ACM Press, 2006, pp. 131–140.

[8] 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.*, vol. 97, no. 3, p. 034301, 2005.

[9] G. Snider, P. Kuekes, and R. S. Williams, "CMOS-like logic in defective, nanoscale crossbars," *Nanotechnology*, vol. 15, no. 8, pp. 881–891, 2004.

[10] G. Snider, *et al.*, "Nanoelectronic architectures," *App. Phy. a-Mater. Sci. & Proc.*, vol. 80, no. 6, pp. 1183–1195, 2005.

[11] S. C. Goldstein and M. Budiu, "NanoFabrics: Spatial computing using molecular electronics," in *ISCA-2001*, Götenberg, Sweden, July 2001, pp. 178–191.

[12] K. Likharev, *et al.*, "Crossnets - high-performance neuromorphic architectures for CMOL circuits," *Ann. NY Acad. Sci.*, vol. 1006, pp. 146–163, 2003.

[13] C. A. Moritz and T. Wang, "Latching on the wire and pipelining in nanoscale designs," in *NSC-3*, 2004, pp. 1–7.

[14] J. M. Tour, *et al.*, "Nanocell logic gates for molecular computing," *IEEE Trans. on Nanotechn.*, vol. 1, no. 2, pp. 100–109, 2002.

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

[16] K. K. Likharev and D. B. Strukov, "CMOL: Devices, circuits, and architectures," in *Introducing Molecular Electronics*, G. Cuniberti, G. Fagas, and K. Richter, Eds. Berlin: Springer, 2005, pp. 447–478.

[17] A. DeHon and K. Likharev, "Hybrid CMOS/nanoelectronic digital circuits: devices, architectures, and design automation," in *ICCAD-2005*, 2005, pp. 375–382.

[18] M. R. Stan, *et al.*, "Molecular electronics: From devices and interconnect to circuits and architecture," *Proc. IEEE*, vol. 91, no. 11, pp. 1940–1957, 2003.

[19] P. J. Kuekes, G. S. Snider, and R. S. Williams, "Crossbar nanocomputers," *Scientific American*, vol. 293, no. 5, pp. 72–76, 2005.

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

[21] D. B. Strukov and K. K. Likharev, "Defect tolerant architectures for nanoelectronic crossbar memories," *J. Nanosci. Nanotechnol.*, 2006, submitted for publication. Available at http://rsfq1.physics.sunysb.edu/∼likharev/nano/.

[22] O. Türel, *et al.*, "Neuromorphic architectures for nanoetectronic circuits," *Int. J. Circ. Theory App.*, vol. 32, no. 5, pp. 277–302, 2004.

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

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

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

[26] E. M. Sentovich, *et al.*, "SIS: A system for sequential circuit synthesis," Tech. Rep., 1992.

[27] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *FPGA*, 2000, pp. 203–213.

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

[29] *ABC: A System for Sequential Synthesis and Verification*, 2005, available online at http://www.eecs.berkeley.edu/∼alanmi/abc/.

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

[31] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Trans. on VLSI*, vol. 12, no. 3, pp. 288–298, 2004.