Automated Design Flow for Diode-Based Nanofabrics

KUSHAL DATTA, ARINDAM MUKHERJEE, and ARUN RAVINDRAN
The University of North Carolina Charlotte

We present an automated design flow for minimizing the use of diodes and switches (active devices) in design implementations on a nanofabric based on chemically self-assembled electronic nanotechnology as proposed in Goldstein and Budiu [2001]. Connectivity and logic in the nanofabric are realized using the switch and diode behaviors of molecular devices, unlike very large scale integrated (VLSI) circuits where complementary metal-oxide semiconductor (CMOS) gates are used. Similar to the optimization goal of reducing the number of gates in VLSI designs to minimize area, power dissipation, and delay, decreasing the number of switches and diodes used in the nanofabric can potentially minimize design implementation area and power dissipation, besides reducing the delay and signal drop between latched stages in order to improve performance. An integrated placement, topology selection, and routing approach for design implementation on the nanofabric is proposed. Note that this problem is fundamentally different from CMOS VLSI placement and routing because of the inherent routing-dependent logic realization in our target nanofabric. To the best of our knowledge this is the first reported work on automated integrated placement, topology selection, and routing for diode-based nanofabrics. A practical and scalable simulated annealing-based placement and routing algorithm has been implemented. On average, the integrated placement and routing approach achieves a reduction of 12% in the number of switches and diodes used for MCNC benchmarks, compared to separate placement and routing optimization results. The maximum reduction achieved in the number of active devices using our approach is 24%, and in general, we observed that the bigger the benchmark, the larger the improvement achieved.

Categories and Subject Descriptors: B.6 [**Logic Design**]

General Terms: Design, Algorithms

Additional Key Words and Phrases: Automatic synmthesis, optimization

1. INTRODUCTION

1.1 Motivation

Although complementary metal-oxide semiconductor (CMOS) technology is expected to dominate in the next 10 years [Semiconductor Industries Association

Authors' address: Department of Elecrical and Computer Engineering, The William States Lee College of Engineering, The University of North Carolina at Charlotte, 9201 University City Blvd., Charlotte, North Carolina 28223; email: amukherj@uncc.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2006 ACM 1550-4832/06/0700-0219 $5.00

Roadmap (SIAR)], alternative cheaper technologies are expected to become viable thereafter. Besides reaching the physical limits of scaling in current CMOS technology, the high cost associated with chip masks and future fabrication plants poses an insurmountable economic challenge to commercial nanometer-scale lithography. A more likely development in the near future is CMOL, the integration of CMOS with molecular-scale nanodevices [Strukov and Likharev].

Chemically self-assembled electronic nanotechnology, henceforth referred to as nanofabric, presents another alternative to the CMOS technology [DeHon 2005; Goldstein and Budiu 2001; Mishra and Goldstein 2003]. Chemical processes are used, often in conjunction with lithography, to self-align and self-assemble nanoscale molecules such that they exhibit switching behavior. Low cost nanofabrics with density of active devices as high as 10^{12} per cm^2 has been predicted in Ma et al. [2005]. Although these nanofabrics do not exist as yet, they are expected to provide unparalleled operation performance of up to 10^{20} operations per cm^2/second with manageable power dissipation. Unlike CMOS VLSI, the stochastic nature of the chemical self-assembly process limits the reliability of nanofabrics. Nanofabrics based on two terminal nanodevices such as diodes are expected to be more reliable than those based on three terminal devices [Mishra and Goldstein 2003].

1.2 Our Research

In this work, we propose an integrated placement, topology selection, and routing scheme to minimize the number of diodes and switches required for design implementations on the chemically self-assembled molecular diode-based nanofabric proposed in Goldstein and Budiu [2001]. In traditional CMOS VLSI circuits, one of the main optimization goals is to reduce the number of gates because this typically results in high performance designs. Connectivity and logic in the nanofabric are realized using the switch and diode (active devices) behaviors of molecular devices and decreasing the number of such active devices can potentially minimize design implementation area and power dissipation, besides reducing delay and signal drop between latched stages (to improve performance). We show that placement, net topology selection, and routing together determine the total number of switches and diodes required to implement a design on the nanofabric and present a simulated annealing [Kirkpatrick et al. 1983]-based nanofabric automated layout solution to minimize this number.

Our contributions in this article include formulation of an integrated placement, topology selection and routing problem, and development of an automated design flow for design implementation on a diode-based nanofabric. We identify constraints unique to the nanofabric such as routing-dependent logic realization, which arise from its physical properties. We incorporate these constraints in a simulated annealing-based placement and routing algorithm. The effectiveness of our integrated placement, topology selection and routing technique has been demonstrated on MCNC benchmark circuits. Future work in this area can be benchmarked against our results.

1.3 Article Organization

The remainder of this article is organized as follows. Recent work relevant to this article is discussed in the next section, followed by a description of the target nanofabric in Section 3. The effects of placement, topology selection and routing on the number of diodes and switches used, is also discussed in this section. Automated flows for implementing designs on the nanofabric are introduced in Section 4, and, in Section 5, we explore the variables and constraints that should be satisfied for design implementations on the nanofabric. In Section 6, we incorporate the constraints in an integrated placement, topology selection, and routing simulated annealing algorithm. Results are discussed in Section 7, followed by the conclusions in Section 8.

2. BACKGROUND

The first step in manufacturing bottom-up chemically self-assembled nanofabrics is to create the devices and wires. Researchers have created a variety of interesting molecular devices including resonant-tunneling diode (RTD) devices [Chen et al. 1999; Chen and Reed 2000], programmable molecular switches [Collier et al. 2000], carbon nanotube transistors [Cui and Lieber 2001; Rueckes et al. 2000], and diodes [Huang et al. 2001a]. Wires which have diameters of only a few nanometers have also been fabricated, for example, single-crystal nanowires [Cui et al. 2001; Huang et al. 2001b; Kamins et al. 2000; Morales and Lieber 1998] and carbon-nanotube wires [Soh et al. 1999]. Some nanoscale wires can be used as more than just conductors, they can also be used as active devices [Collier et al. 1999; Cui and Lieber 2001; Derycke et al. 2001; Trans et al. 1998; Wind et al. 2000]. During chemical self-assembly, dimensions of nanowires and spacing between them are controlled to nanometer scales using molecular seed catalysts.

An interesting consequence of all these devices is the ability to store state and implement switching at a wire crossing, that is, the switch device itself holds its state. Even if 35nm silicon feature sizes (which might imply 70–90nm wire pitches) can be achieved in silicon circuits, the density difference between 20nm-spaced nanotubes or silicon nanowires [Cui et al. 2001; Huang et al. 2001b; Kamins et al. 2000; Morales and Lieber 1998] and the 35nm silicon wires will be greater than the (roughly 80nm/20nm) wire feature size difference. This implies a 16$\times$ reduction in area, and, consequently, a similar reduction in power dissipation. The difference in relative costs also has an impact on architecture. Whereas full crossbars in silicon are switch-dominated, motivating their depopulation for compactness, crossbars in this technology can be fully populated with no density penalty.

Using the current success and understanding of chemical self-assembly, Luo et al. [2002] has manufactured a working 8×8 memory array in collaboration with researchers at HP Labs. DeHon [2003] proposes an architecture based on crossed arrays of nanowires and carbon nanotube transistors. These crossed arrays can act as memory cores, programmable logic array planes for computation, and crossbars for interconnection—all the key elements needed to implement designs. However, the effective crosspoint density of nanofabrics is

reduced due to the CMOS and address support needed for each subarray. To overcome this problem, DeHon et al. [2003] propose an address decoder that uses a small number of microscale control wires to selectively activate one of a large number of nanowires.

Chemical self-assembly as a stochastic process will not always produce precise alignment of structures. It is much easier to bring two terminals into precise proximity than to do the same with three terminals. Hence, two terminal devices such as diodes are more reliably fabricated on a nanofabric compared to three terminal transistors [Mishra and Goldstein 2003]. Goldstein and Budiu [2001] and Stan et al. [2003] suggest an architecture based on molecular-scale electronic building blocks with only two terminal devices. For the design optimization flow presented in this article, we consider the nanofabric architecture suggested in these works. This nanofabric will be referred to as our *target nanofabric* in the remainder this article.

Our target nanofabric has a lot of similarity with VLSI field programmable gate arrays (FPGAs). Similar to the crosspoint logic and switching elements in the nanofabric, FPGAs have slices for logic implementation, and switching blocks and interconnect tracks for routing. FPGAs are broadly categorized as island-style with logic slices separated from each other by routing resources, and nonisland-style with clustered logic slices. VPR [Betz and Rose 1997; Betz et al. 1999] is the current public domain state-of-the-art FPGA placer and router. VPR's core is a simulated annealing algorithm [Kirkpatrick et al. 1983] that uses a net semiperimeter metric to estimate the routability of a placement. VPR consistently produces high quality placements. Due to a strong prevalence of routing rich island-style FPGA architectures, VPR's placement algorithm is primarily targeted to island-style FPGAs. Existing design automation flows for FPGAs are quite mature, and this motivates us to build on these design flow algorithms while optimizing design implementations on the target nanofabric. Since the simulated algorithm in VPR gives the best result for island-style FPGAs, which are architecturally similar to the target nanofabric, the design flow that we present in the article is based on simulated annealing.

In [Datta et al. 2005], we presented a simulated annealing-based topology selection and routing algorithm for placed designs on a molecular diode-based nanofabric to reduce the number of diodes and switches. However, the method is limited by the initial placement which is optimized for VLSI FPGAs and not the target nanofabric. In this article, we extend the work of [Datta et al. 2005] by integrating the placement, topology selection and routing phases for more optimal results.

3. MOLECULAR DIODE-BASED NANOFABRICS

Our target nanofabric [Stan et al. 2003] is manufactured in a bottom-up fashion, where the basic components such as wires and switches are first chemically self-assembled, and then aligned and grouped into regular arrays through self-assembly to form complete systems. Two planes of aligned wires are combined to form a 2D grid (of the order of a few microns) with configurable molecular switches at the crosspoints. The self-assembly process does not allow for

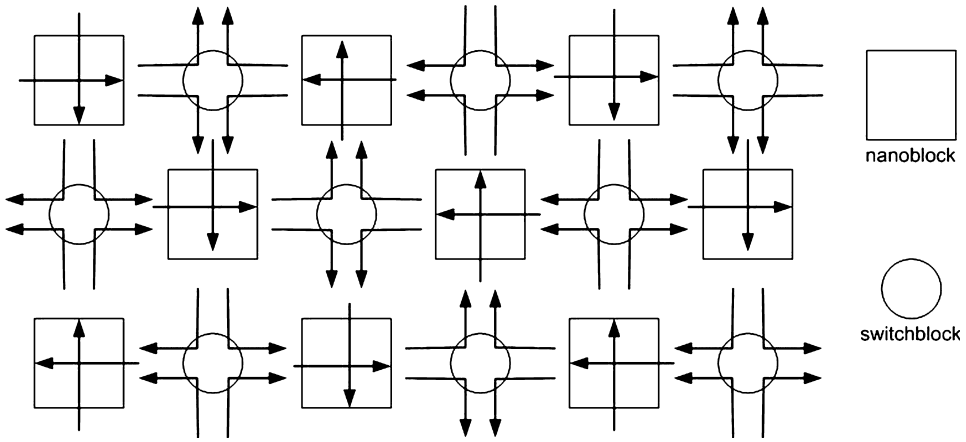

Fig. 1. Nano architecture.

precise end-to-end connections between nanowires, and hence, all connections are made at crosspoints of orthogonally aligned wires. Molecular latches based, on resonant tunneling diodes, are used for saving states and restoring signals [Goldstein and Rosewater 2002]. A post fabrication configuration step is used to realize circuits on the nanofabric.

Similar to an FPGA, our target molecular diodebased nanofabric is a regular 2D mesh of interconnected computing units (nanoblocks) and routing switches (switch blocks). A nanoblock can be programmed after fabrication to implement logic functions. As dictated by fabrication constraints, outputs of a nanoblock can either go out through the south or east sides (SE), or through the north and west sides (NW). The respective inputs can only come in through the sides unused by the outputs. The switch block is the area between nanoblocks where the input and output wires of the nanoblocks overlap, and it can be configured to route signals between nanoblocks. In Figure 1, we show a nanofabric where NW and SE nanoblocks are arranged along alternate diagonals. A switch block between two SE (left) and NW (right) nanoblocks in a row will have inputs coming in through the east and west sides and outputs going out through that north and south sides (NS switchblock), while that between two NW (left) and SE (right) nanoblocks will have inputs coming in through the north and south sides and outputs going out through the east and west sides (WE switch block). This arrangement of nanoblocks and switch blocks allow for the realization of signal flow from any point on the fabric to any other point.

3.1 Nanoblock

As shown in Figure 2(a), a nanoblock consists of (i) the molecular logic array (MLA), (ii) the molecular latches (useful for saving states and restoring signal levels), and (iii) the I/O area used to connect the nanoblock to its neighbors [Goldstein and Budiu 2001]. The MLA is composed of two orthogonal sets of wires. At each crosspoint lies a configurable molecular switch, which acts like a diode if configured to be on [Goldstein and Budiu 2001]. The direction of

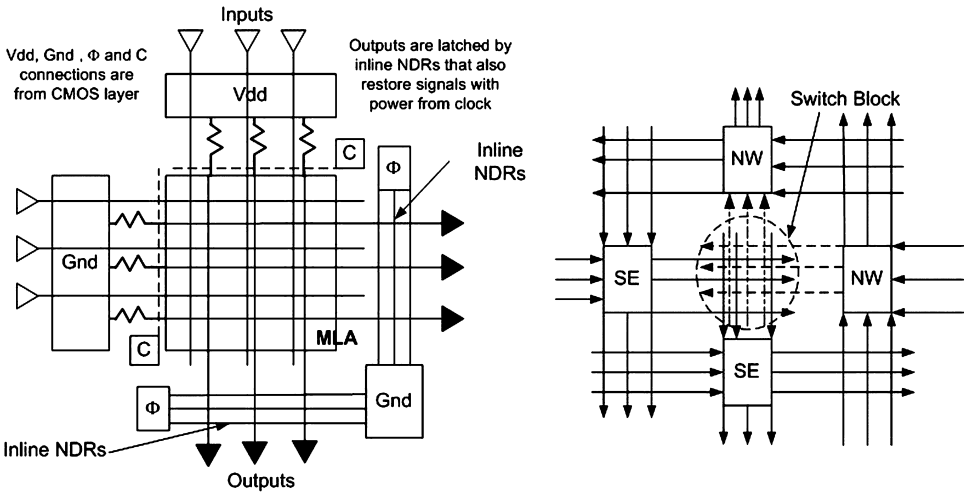

Fig. 2. Nanoblock and switch block [Goldstein and Budiu 2001].

current flow through the diode is determined during fabrication and is non-reconfigurable. A vertical wire drives the anode of the diode, while the cathode is connected to a horizontal wire; this is determined by fabrication constraints. The MLA implements Boolean functions using diode-resistor logic. The drawbacks of this logic style are that signals degrade and have to be restored using latches and that complements of signals have to be separately generated and cannot be obtained by inversion of the signal, as inverters cannot be realized.

3.2 Switch Block

A switch block is similar to an MLA without power and ground connections. As shown in Figure 2(b), a switch block is formed by 4 nanoblocks; crossing horizontal and vertical wires from the surrounding nanoblocks are connected by configurable molecular switches. If the nanoblocks have R rows and C columns each, the number of vertical wires in the switch block is 2C, and the number of horizontal wires is 2R. For the configuration of a switch block shown in Figure 2(b), a maximum of 4RC crosspoints can be formed.

3.3 Layout

Figure 3 shows the realizations of different basic functionalities using the diode-resistor logic in an SE nanoblock. AND and OR functions can be realized using signals to drive either the cathodes or the anodes of the diodes in the nanoblock (figures 3(b) and 3(d)). Using the complements of the signals and DeMorgan's law, NOR and NAND functions can be realized from the AND and OR functions, respectively (Figures 3(c) and 3(e)). Due to the fabrication constraints, signals have to enter the nanoblock from the west side for an AND (product) term realization and from the north side for an OR (sum) term realization. Note that a product term is available in a vertical wire connected to the anodes

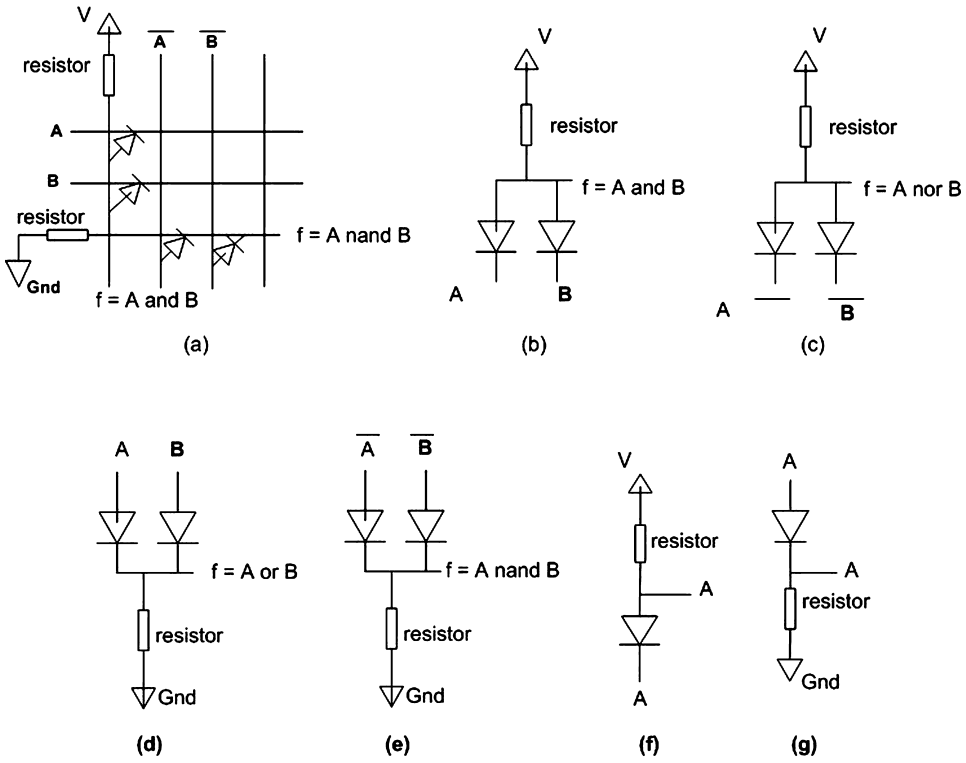

Fig. 3. Functional mapping in a nanoblock.

of the diodes, while a sum term is available in a horizontal wire connected to the cathodes of the diodes. However, a signal entering the nanoblock vertically can be rerouted horizontally within the nanoblock for use in a product term. This is shown in Figure 3(g) where the signal A enters the nanoblock's north side to drive the anode of a diode, and the same signal is available on the horizontal wire connected to the cathode of the diode. Similarly, a signal entering the nanoblock horizontally can be rerouted vertically within the nanoblock as shown in Figure 3(f). This signal can then be used in a sum term. Similar arguments exist for the NW nanoblocks. Note that topology selection and routing determine the direction of entry of signals in a nanoblock, and hence affect the number of diodes required to reroute signals for functional realizations.

Figure 4 shows branching inside nanoblocks and switch blocks. In Figure 4(a), we have show how a signal entering an SE nanoblock from the west side is rerouted to drive a vertical wire. At the output of the nanoblock, we have the same signal as output from both the east and south sides. Similarly, 2-way branching of a signal entering a nanoblock from the north side is shown in Figure 4(b). Branching of signals inside NS and WE switch blocks are shown for a signal A entering the switch blocks from the west and north sides, respectively, in Figures 4(c) and 4(d). The crosspoints inside the switch blocks are molecular switches that can be configured after fabrication.

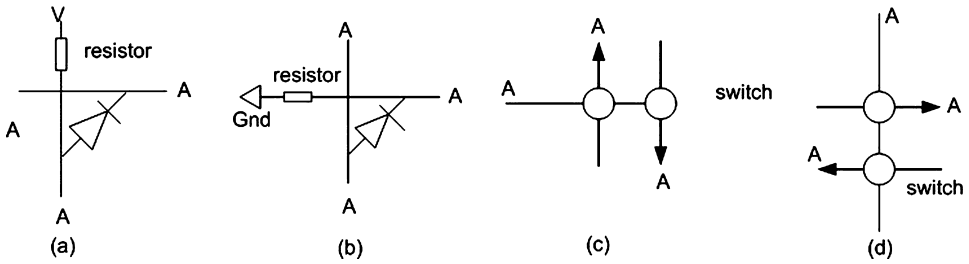

Fig. 4. Branching inside nanoblocks and switch blocks.

Similar to VLSI circuits, net lengths are determined by the placement of logic blocks on nanofabrics. Nanoblocks can be used for pass-through signals which do not require use of any diode if no signal rerouting is involved; a net can enter a SE nanoblock through the north or the west side and go out through the south or east side, respectively. However, a net cannot be routed through a switch block without changing direction at least once (refer to Figure 1). This is because of the input-output direction constraints on switch blocks and hence, routing a net through a switch block involves the use of a switch. Note that a switch block is always used to route a net between any two nanoblocks. A net can also be routed between two switch blocks along a diagonal. Referring to Figure 1, consider a net to be routed between two NS switch blocks on the same diagonal. The net going out north from one switch block enters a NW nanoblock from the south side, goes out of the nanoblock through the west side after a directional rerouting, and enters the other NS switch block from the east side. As explained in Figure 3, a directional rerouting inside a nanoblock uses a diode. Hence, routing of a net between two nanoblocks or two switch blocks requires one or more switches and/or diodes. Since placement determines net lengths, it directly affects the number of diodes and switches used for routing nets.

4. NANOEDA DESIGN FLOWS

A nanofabric is similar to an FPGA because of the regular 2D array structure and reconfigurability. Hence, FPGA-inspired electronic design automation flows can be developed for automatic implementations of complex designs on nanofabrics. We refer to these as NanoEDA design flows.

We start with the *blif* description of an MCNC benchmark circuit (www.mcnc.org) that has been minimized in terms of literals using SIS [www-cad.eecs.berkeley.edu]. FlowMap [http://ballade.cs.ucla.edu/software_release/rasp/htdocs] is used to decompose the blif netlist into 4-input 1-output functions. VPack is used next to pack the 4-input 1-output blocks and flip flops together. (http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html)

In our first NanoEDA design flow, the netlist file generated by VPack is placed using a tool that we developed. This placer uses a simulated annealing (SA)-based algorithm to minimize a cost function that takes into account both the wire length and routing density. SA is a very well known optimization technique that falls into the category of hill climbing heuristics. This method accepts temporarily inferior solutions with the hope of overcoming locally optimal

solutions and obtains globally optimum ones. A parameter used in the SA algorithm, temperature, starts off initially at a high value and gradually cools down during the heuristic—the higher the temperature, the higher the tendency of the algorithm to accept inferior solutions. Of course, each solution point has to be feasible and satisfy all constraints. A detailed description of our placement algorithm is given in Section 6. For each net in the placed nanofabric, we then proceed to find a feasible topology and routing solution that minimizes the number of diodes and switches used. An optimal scalable SA algorithm for routing is presented in Section 6. The result of this step is an optimized nanofabric layout.

Our second NanoEDA design flow is obtained by the integrated optimizations of placement, topology selection and routing such that the number of diodes and switches used is minimized. This design flow also starts from the VPack netlist, but combines the placement, topology selection and routing steps into one. Section 6 explains this approach in detail, and, as shown by the results in Section 7, the integrated optimization flow gives us the best results.

The next section establishes the design variables and constraints that need to be satisfied by the NanoEDA design flows discussed previously. Section 6 discusses implementations of the NanoEDA design flows based on simulated annealing.

5. DESIGN SPACE DESCRIPTION

The targeted design space has multiple placements or mapping of logic blocks to nanoblocks and multiple potential routes and topologies for different nets in any placement. For a given placement, the nanofabric is mapped to a directed graph where all nanoblocks and switch blocks are vertices, and their interconnections are directed edges. Vertices corresponding to nanoblocks with mapped logic are marked with input and output net names. For each net we do a depth-first search (DFS) on the graph, starting from the vertex corresponding to the net source. The search space is limited by a user-defined bounding box, which is initially set equal to the dimension of the net bounding box. In case no viable route for the net is found, the bounding box is adaptively increased.

Consider a SE nanoblock N where inputs can only enter through the north or west sides. We shall use the word "equation" to generally include inequations as well. For any signal l_i entering N, there are three integer variables (with values 0 or 1) w_i, n_i, and b_i, respectively, denoting whether l_i enters N through the west, north, or through both north and west sides. Since only one of these entries is possible, we have

$$\forall l_i \in N : w_i + n_i + b_i = 1. \tag{1}$$

The signal l_i has a one-to-many mapping to l^x_{kj}, where x denotes a sum-of-product (SOP) function f^x in N that depends on l_i, and k denotes the corresponding literal in the jth product term. All the associated variables w_i, n_i, and b_i are similarly mapped to w^x_{kj}, n^x_{kj}, and b^x_{kj}. If l_i is involved in a product-of-sum (POS) function f^y in N, it is mapped to l^y_{pm}, where p denotes a literal in the mth sum term. The variables w_i, n_i, and b_i will be respectively mapped to w^y_{pm},

n^x_{pm}, and b^x_{pm}. The following equation shows our representation of an SOP and a POS function. The SOP function f^x has J^x product terms in the sum, and K^x_j represents the number of literals in the jth product term. Similarly, the POS f^y has M^y sum terms in the product, and P^y_m represents the number of literals in the mth sum term.

$$f^x = \sum_{j=1}^{J^x} \left(\prod_{k=1}^{K^x_j} l^x_{kj} \right) \quad f^y = \prod_{m=1}^{M^y} \left(\sum_{p=1}^{P^y_m} l^y_{pm} \right) \tag{2}$$

The total number of diodes required to implement f^x is given by

$$D^x_{sop} = \sum_{j=1}^{J^x} \sum_{k=1}^{K^x_j} \left(w^x_{kj} + 2n^x_{kj} + b^x_{kj} \right) + J^x. \tag{3}$$

Explanation of Equation (3) follows. For any signal l_i, one and only one of the variables w^x_{kj} or n^x_{kj} or b^x_{kj} will be "1", depending on whether l_i enters N through the west, or north, or both north and west sides. If l_i enters N through the west side ($w^x_{kj} = 1$ or $b^x_{kj} = 1$), only one diode will be required to realize l_i as a literal l^x_{kj} in the jth product term. If l_i enters N only through the north side, one diode will be required to change its direction, and another diode will be required to realize l_i as a literal l^x_{kj} in the jth product term. Thus if $n^x_{kj} = 1$, we need two diodes; and hence, the multiplier 2 in the expression $(w^x_{kj} + 2n^x_{kj} + b^x_{kj})$. Now there are K^x_j literals in the jth product term, and therefore, a summation of the diodes over all K^x_j literals gives us the number of diodes required to implement the jth product term $(\sum_{k=1}^{K^x_j} (w^x_{kj} + 2n^x_{kj} + b^x_{kj}))$. The outermost summation in Equation (3) calculates the number of diodes required to implement all the different J^x product terms in the SOP function f^x. Finally, J^x diodes are required to realize the output sum term of f^x.

Similarly, the total number of diodes required to implement f^y is

$$D^y_{pos} = \sum_{m=1}^{M^y} \sum_{p=1}^{P^y_m} \left(2w^y_{pm} + n^y_{pm} + b^y_{pm} \right) + M^y. \tag{4}$$

Again, two diodes are required if a signal enters N through the west side and is involved in a sum—one to change the signal direction, and the other to realize the sum term. The inner summation over P^y_m literals calculates the number of diodes in the pth sum term, and the outer summation over M^y calculates the number of diodes required to implement all the sum terms in f^y. Finally, M^y diodes are required to realize the output product term. The total number of diodes required to realize X SOP functions and Y POS functions, all independent of each other, is given by:

$$D = \sum_{x=1}^{X} D^x_{sop} + \sum_{y=1}^{Y} D^y_{pos}. \tag{5}$$

Let R(N) be the maximum number of horizontal rows in N that constrains the use of horizontal wires by all signals associated with N as follows:

$$\forall l_i \in N : \sum_i (n_i \cdot t_i) + \sum_i (w_i + b_i) + X + \sum_{y=1}^{Y} M^y \leq R(N). \tag{6}$$

Here t_i is an integer variable in (0, 1), which is 1 if l_i is involved in any product term. So if l_i enters N from the north side such that $n_i = 1$, and if l_i is involved in a product term such that $t_i = 1$, then l_i needs to be routed horizontally to drive the cathode of a diode as shown in Figure 3. Hence the term $(n_i \cdot t_i)$ evaluates to 1 in Equation (6) and accounts for one row that is used up to reroute the signal l_i horizontally. The summation of similar terms over all signals entering N from the north side and involved in a product computation is the first term in Equation (6). The second term in Equation (6) is the summation over all signals entering N from the west side (including signals that enter N from both north and west sides). Since such a signal uses up one horizontal row in N, the second summation term in Equation (6) calculates the number of horizontal rows required by these signals. The third term X in Equation (6) is the total number of final sum outputs (one per SOP function) in N, while the fourth term is the summation over all intermediate sum terms in all the Y POS functions in N. Note that M^y is the number of sum terms in any particular POS function f^y.

A similar constraint for the vertical wires in N is given by

$$\forall l_i \in N : \sum_i (w_i \cdot s_i) + \sum_i (n_i + b_i) + Y + \sum_{x=1}^{X} J^x \leq C(N). \tag{7}$$

Here s_i is an integer variable in (0, 1), which is 1 if l_i is involved in any sum term. C(N) is the maximum number of columns in N.

Subfunction Sharing. Consider an SOP function f^x in an SE nanoblock N, with a subfunction f as one of its sum terms (f can be potentially shared among several functions in N). If f is evaluated outside N, it is treated as an input and nothing changes with respect to our previous discussions. If f is evaluated inside N, two situations arise. First, f is a POS, in which case it is available on a vertical wire. In this case, just 1 diode is required to realize the final sum term of the SOP function. This diode has already been considered in Equation (3). Second, if f is an SOP function, however, it is available on a horizontal wire, and it has to be rerouted vertically to realize f^x. In this case, an extra diode is required, and an extra column is used up. If there are Γ such functions, D^x_{sop} is modified as shown in Equation (8), and the constraint Equation (7) is modified as shown in Equation (10).

Similarly, for sharing Θ POS subfunctions as product terms in a POS function f^y, D^y_{pos} is modified as shown in Equation (8), and constraint Equation (6) is modified as shown in Equation (9). Here *LHS* stands for left-hand side. In this case, extra diodes and rows are used up to reroute vertical signals horizontally so that they can be product terms for f^y. The total number of required diodes

D has been modified as shown in Equation (8).

$$\tilde{D}^x_{sop} = D^x_{sop} + \Gamma, \tilde{D}^y_{pos} = D^y_{pos} + \Theta$$

$$D = \sum_{x=1}^{X} \tilde{D}^x_{sop} + \sum_{y=1}^{Y} \tilde{D}^y_{pos} \tag{8}$$

$$LHS(6) + \Theta \leq R(N) \tag{9}$$

$$LHS(7) + \Gamma \leq C(N) \tag{10}$$

Pass-Through Signals. Let us consider an SE nanoblock N, where l_v represents a signal that passes through N, but is not involved in its functionality. We define the following integer variables in (0, 1): u_v^w (which is 1 if l_v enters N through the west side), u_v^n (which is 1 if l_v enters N through the north side), u_v^{we} (which is 1 if l_v enters N through the west side and exits N through the east side), u_v^{ws} (which is 1 if l_v enters N through the west side and exits N through the south side), u_v^{ne} (which is 1 if l_v enters N through the north side and exits N through the east side), and u_v^{ns} (which is 1 if l_v enters N through the north side and exits N through the south side). We do not allow a pass-through signal to enter a nanoblock from both sides; this prevents the formation of cycles in a route. This constraint is implemented as

$$u_v^w \geq u_v^{we}, u_v^w \geq u_v^{ws}, u_v^n \geq u_v^{ne}, u_v^n \geq u_v^{ns}, u_v^w + u_v^n \leq 1. \tag{11}$$

Also, Equation (11) relates the exit variable(s) of a signal to the corresponding entry side variable.

We have allowed for 2-way branching of pass-through variables inside N—this will lead to automatic selection of net topology. The following constraint ensures that a signal may enter N through either the west or the north side, and then go out through the east or south sides, or both.

$$\forall l_v \in N : \left(u_v^w + u_v^n\right) \leq \left(u_v^{we} + u_v^{ws} + u_v^{ne} + u_v^{ns}\right) \leq 2 \tag{12}$$

A pass-through variable will use up a row if it either enters through the west side, or if it exits through the east side of N. This is shown in Equation (13), which modifies Equation (9).

$$LHS(9) + \sum_v \left(u_v^w + u_v^{ne}\right) \leq R(N) \tag{13}$$

Similarly, a pass-through variable will use up a column if it either enters through the north side, or if it exits through the south side of N. This is shown in Equation (14), which modifies Equation (10).

$$LHS(10) + \sum_v \left(u_v^n + u_v^{ws}\right) \leq C(N) \tag{14}$$

Since a single diode is required to change signal directions, but not to pass through the signal, the total number of diodes used is incremented by the number of pass-through literals that are required to be directionally rerouted inside N.

$$D = \sum_{x=1}^{X} \tilde{D}^x_{sop} + \sum_{y=1}^{Y} \tilde{D}^y_{pos} + \sum_v \left(u_v{}^{ne} + u_v{}^{ws}\right) \tag{15}$$

The entire analysis in this section has been for SE nanoblocks. The equations presented will also hold for NW nanoblocks if east is replaced by west and vice versa, and north is replaced by south and vice versa.

Switch Block Variables. The e variables for signals routed through a switch block correspond to the u variables of pass-through signals through nanoblocks. Consider a WE switch block (SB) with inputs coming in from the north and south sides, and outputs going out through the east and west sides. The total number of switches used, S, is equal to the number of exit points of a signal from SB over all signals in the set $\{l_v\}$ routed through SB.

$$\forall l_v \in SB : S = \sum_v \left(e_v^{ne} + e_v^{nw} + e_v^{se} + e_v^{sw}\right) \tag{16}$$

The following constraint ensures a fanout of at most 2 inside a switch block. Variables involved with branching lead to automatic topology selection during the optimization.

$$\left(e_v^n + e_v^s\right) \leq \left(e_v^{ne} + e_v^{nw} + e_v^{se} + e_v^{sw}\right) \leq 2 \tag{17}$$

Similar to the cycle breaking constraints for pass-through signals for nanoblocks, the following constraints ensure that a certain signal enters either through the north or the south side of a switch block.

$$e_v^n \geq e_v^{ne}, e_v^n \geq e_v^{nw}, e_v^s \geq e_v^{se}, e_v^s \geq e_v^{sw}, e_v^n + e_v^s \leq 1 \tag{18}$$

Since a switch block is formed by wires entering and leaving a nanoblock, the maximum number of north-entering wires in the WE switch block SB is equal to the number of columns ($C(N_n)$) of an SE nanoblock N_n above it, and the number of south-entering wires in SB is equal to the number of columns ($C(N_s)$) of an NW nanoblock N_s below it:

$$\sum_v e_v^n \leq C(N_n), \sum_v e_v^s \leq C(N_s). \tag{19}$$

Similarly, the maximum number of wires leaving SB through the east side is limited by the number of rows ($R(N_e)$) of its east side SE nanoblock N_e, and the maximum number of wires leaving SB through the west side is limited by the number of rows ($R(N_w)$) of its west side NW nanoblock N_w as given by

$$\sum_v e_v^{ne} + \sum_v e_v^{se} \leq R(N_e), \sum_v e_v^{nw} + \sum_v e_v^{sw} \leq R(N_w). \tag{20}$$

In practice, all nanoblocks will have the same number of rows and columns, but the number of rows need not be the same as the number of columns. The analysis in this section has been for a WE switch block. The equations will hold for NS switch blocks as well if south is replaced by east and vice versa, and north is replaced by west and vice versa.

The cost function to minimize is (D+S) over all the nanoblocks and switch blocks used in the design implementation. Given a nanofabric architecture, available space on the nanofabric, and a placement, some of the w, n, b, u, and e variables will be constant values. For example, a net might have a unique

route such that the b variables of the corresponding signal will be 0, certain n, w, u, and e variables will be set to 1's and 0's, and Equations (11) and (16) for the signal will be upper bounded by 1 and not 2.

6. OPTIMIZATION ALGORITHMS

6.1 Placement

The placement problem is stated as follows. Given a nanofabric architecture and a design mapped to k-input nanoblocks, place the design in the architecture to minimize a cost function that takes into account both the wire length and routing density. First, the nanofabric is divided into equally sized tiles whose size is empirically determined and depends on the design to be implemented on the nanofabric. Each tile has several nanoblocks and switch blocks. Traditionally, placement minimizes wire lengths. However, for nanofabrics, we have observed that just minimizing the wire length during placement results in overly congested designs which (i) require more vertical and horizontal wires per nanoblock and switch block, and/or (ii) results in net detours during routing. In the first case, most of the vertical and horizontals wires are pass-through signals and use diodes and switches for directional changes as explained in Section 3.3. In the second case, net detours lead to longer net lengths and ultimately results in the same effects as in the first case. Hence, our placement cost function is:

$$\sum_{n=1}^{N_{nets}} \sum_{l}^{L_n} [wire\_length_{nl} * routing\_density^{l}]. \tag{21}$$

Here $wire\_length_{nl}$ is the length of the segment of a net n which passes through the lth tile, and L_n is the set of all tiles through which n passes. Also, $routing\_density^{l}$ denotes the current routing density of the lth tile. N_{nets} is the total number of nets in the design.

We have used the SA algorithm [Kirkpatrick et al. 1983] to provide an optimized solution to the placement problem. For the initial placement, all the input/output nodes are placed on the boundary nanoblocks in an ordered fashion as shown by the arrows along the nanofabric boundary in Figure 5. Thereafter, the logic blocks are placed inside the nanofabric, following a diagonal pattern dictated by the input/output directional constraints of the nanoblocks and switch blocks with the order of placement indicated by the zig-zagging arrow in Figure 5. Note that diagonal placement of logic blocks is intuitively expected to reduce wire lengths in the nanofabric where all nanoblocks along a diagonal are either of the type SE or of the type NW, and all switch blocks along a diagonal are either of the NS type or of the WE type (refer to Figure 1). Although this placement can potentially lead to reduced net lengths and hence, a reduced number of switches and diodes used during the initial routing phase, it quickly builds up congestion in the layout as routing progresses. For nets that are routed later during the routing phase, this congestion requires net detours and longer net lengths. Therefore, it is possible that an initial placement does not guarantee 100% routing for a certain circuit.

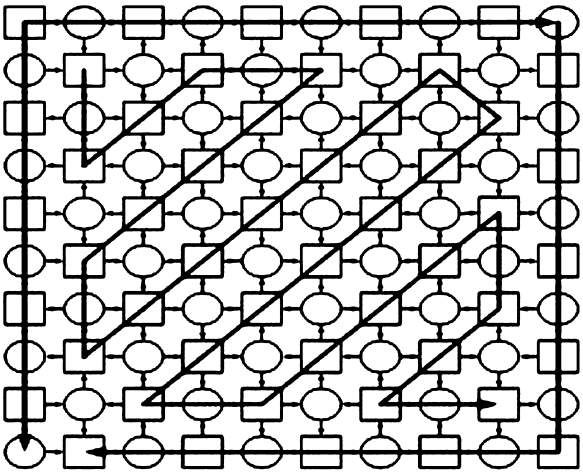

Fig. 5. Initial placement.

The pseudo code of our simulated annealing (SA) algorithm follows.

```
SA_Place( )
        Iteration = I0;              /* initial # of iterations ≥ 1 */
        SA_Counter = 1000;           /* SA run time */
        T = T0;                      /* initial temperature */
        Init_Place( );
        CFN_OLD = Calculate_Cost_function( );
        Loop SA_Counter times
                Loop Iteration times
                 Perturb_Placement( ) ; /* Change placement */
                 CFN_NEW = Calculate_Cost_function( )
                 If ( CFN_NEW < CFN_OLD or random < e^((CFN_OLD − CFN_NEW)/T) ) then
                        If (CFN_NEW < CFN_OLD ) Update _best_placement( );
                                CFN_OLD = CFN_NEW;
                End
                End Loop        /* inner loop ends *
      T = α * T; Iteration = β * Iteration;          /* α < 1, β > 1 */
      End Loop          /* Simulated Annealing is out of time */
End
Perturb_Placement( )
        (old_x,old_y) = Choose_random_placed_nano_block( );
        (new_x,new_y) = Choose_random_empty_nano_block( );
        While ( Isaboundarynode( old_x,old_y ) and not Isaboundarynode( new_x,
new_y ) )
                (new_x,new_y) = Choose_random_empty_nano_block( );
            Swap[ (new_x,new_y),(old_x,old_y) ];
End
```

The first three lines of the SA function initialize the number of iterations and the temperature. Thereafter, the initial placement is done as shown in Figure 5, and the cost of the placement is calculated using Equation (21). In the inner loop of the SA, the first call is to the function *Perturb_Placement* which randomly selects a nanoblock that has logic mapped to it and a nanoblock which is empty. The logic is then remapped to the empty nanoblock. Note that

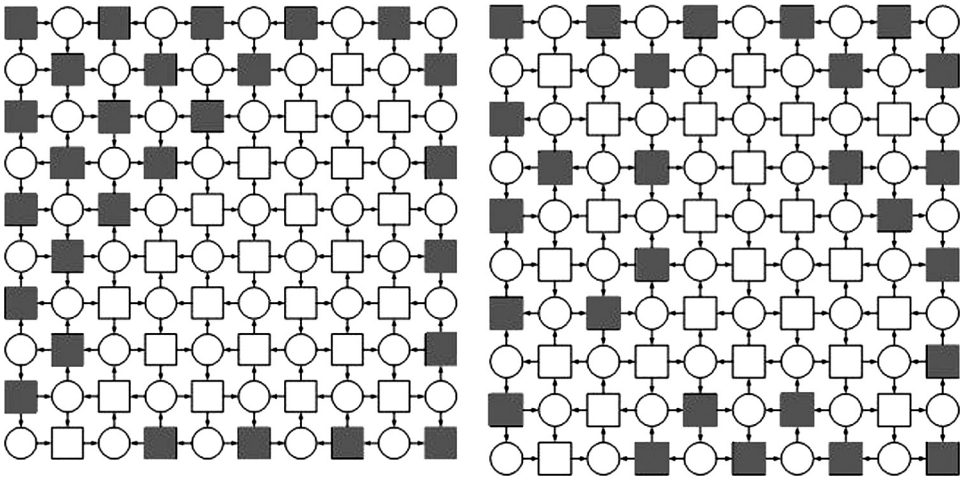

Fig. 6. Initial and optimized placement of the cm138a in a 10×10 nanofabric grid.

a boundary node is swapped only with another empty boundary node. Then *Calculate_Cost_function* does a bounding box estimation of the routings of all nets affected by the remapping, updates the routing densities of the affected tiles involved with the affected nets, and then returns the new cost function calculated by Equation (21). If (i) the new cost function is lesser than the old one, or (ii) a randomly generated number is smaller than a threshold selected by the increase in cost function and the current temperature, the perturbation is accepted. Note that the perturbed placement is stored as the best only if the new cost is less than the old one and after verifying that a valid net routing exists for the perturbed placement. Condition (ii) helps in hill climbing so that a solution which is temporarily degraded can be selected in the hope of escaping a local minimum and ultimately finding the globally optimum solution. The higher the temperature, the higher the threshold, and the higher the probability that a random number is less than the threshold to allow hill climbing. Initially, the temperature is set high to encourage hill climbing, and, as SA proceeds, the temperature is reduced using the constant cooling factor $\alpha < 1$, while the number of iterations in the inner loop is increased by a factor of β. Finally, the algorithm terminates when the outer loop time outs.

Figure 6 shows the result of placing the cm138a MCNC benchmark circuit using the SA algorithm. The shaded nanoblocks have logic implementations or input/output pads in them. In Figure 6(a), we have the preplacement scenario, while Figure 6(b) shows the distributed occupancy of logic to reduce routing congestion. The reduction in the use of diodes and switches for the placement of Figure 6(b) was discussed in Section 7.

6.2 Topology Selection and Routing

Given a placed design in a nanofabric, the next problem to solve is net routing. Using the variables of Section 5, we do a design space exploration to select net

topologies and route nets subject to the different constraints of Section 5. The measure of design quality is the total number of diodes (from Equation (15)) and switches (from Equation (16)) used. We use a modified version of the placement SA algorithm for simultaneous topology selection and net routing. Note that the design space exploration as discussed in Section 5 simultaneously explores different topologies and routes for different nets. The SA algorithm for topology selection and routing follows:

```
SA_Route( )
      Iteration = I0;            /* initial # of iterations ≥ 1 */
      SA_Counter = 1000;     /* SA run time */
      T = T0;              /* initial temperature */
      Init_Route( );
      D_old = Calculate_diodes_used( );
      S_old = Calculate_switches_used( );
      CFN_routeOLD = D_old + S_old;
      Loop SA_Counter times
      Loop Iteration times
            Perturb_Routing( ); /* Change routing */
   D_new = Calculate_diodes_used( );
      S_new = Calculate_switches_used( );
   CFN_routeNEW = D_new + S_new;
            If (CFN_routeNEW < CFN_routeOLD or random < e^((CFN_routeOLD − CFN_routeNEW)/T_NEW))
   If (CFN_routeNEW < CFN_routeOLD) Update _best_routing();
      CFN_routeOLD = CFN_routeNEW;
           End
      End Loop          /* inner loop ends *
      T = α * T ; Iteration = β * Iteration;          /* α < 1, β > 1 */
      End Loop          /* Simulated Annealing is out of time */
End

Perturb_Routing( )
      Select a net n randomly;
   New_route = route of net n;
      Num_blotted_nodes = random_number( );
      Get_bounding_box( New_route );
      Loop Num_blotted_nodes times within bounding box of New_route
           (x,y) = Choose_random_node_in_grid( );
           COLOR( x,y ) = BLACK;
      End Loop
      Do
           Reroute( n );
           If route not found, increase bounding box;
      While route not found by DFS;
   Return New_route = route of net n;
End
```

The differences between the SA techniques for placement and routing are the initial starting point selection functions, the cost calculation functions, and the perturbation functions. Since the best placement from Section 6.1 guarantees routing, the initial placement and routing for this step are determined from the results of the optimization in Section 6.1. The cost of the topological selection and routing is calculated using Equations (15) and (16), as explained earlier. The *Perturb_Routing* function selects a net n randomly, finds its route bounding box,

blacks out or prohibits routing through certain randomly chosen nanoblocks and switch blocks, and runs a depth-first search inside the bounding box to find an alternate topology and route, while satisfying the constraints of Section 5. If a new route is not found, the size of the bounding box is adaptively incremented until a route is found.

6.3 Integrated Optimization

The integrated optimization of placement and topology selection and routing uses the SA approach described in the following. The initial placement and routing is the same as that for the routing optimization described in Section 6.2, and the cost function is calculated similar to the case of routing optimization. However, for the integrated optimization, after the placement is perturbed using the same functions as that for the SA-based placement optimization of Section 6.1, the nets affected by the perturbation are actually routed to satisfy the constraints of Section 5 in every iteration. If the new cost function calculated using Equations (15) and (16) is smaller than the old one, the best placement and routing solution is saved. The rest of the SA method is similar to the previous optimization algorithms.

```
SA_PandR( )
        Iteration = I0;                 /* initial # of iterations ≥ 1 */
        SA_Counter = 1000 ;             /* SA run time */
        T = T0;             /* initial temperature */
        Init_placement_and_route( );
        D_old = Calculate_diodes_used( );
        S_old = Calculate_switches_used( );
        CFN_OLD = D_old + S_old;
        Loop SA_Counter times
                Loop Iteration times
                        Perturb_Placement( ) ;          /* Change placement */
                        Get_route_for_current_placement( );
                        D_new = Calculate_diodes_used( );
        S_new = Calculate_switches_used( );
                        CFN_NEW = D_new + S_new;
                        If (CFN_NEW < CFN_OLD or random < e^((CFN_OLD − CFN)/T_NEW) ) then
  If (CFN_NEW < CFN_OLD ) Update _best_placement_routing();
        CFN_OLD = CFN_NEW;
                        End
                 End Loop           /* inner loop ends *
        T = α * T ; Iteration = β * Iteration;          /* α < 1, β > 1 */
        End Loop        /* Simulated Annealing is out of time */
End
```

7. EXPERIMENTS AND RESULTS

The benchmark circuits we have chosen are from the MCNC (www.mcnc.org) suite. These circuits are first placed as shown in Figure 5. Note that this initial placement does not always guarantee routing. Thereafter the placement is optimized using the SA algorithm of Section 6.1 which guarantees routability, followed by the SA-based routing as discussed in Section 6.2. The results of this approach are tabulated in Table I. For all our experiments, the SA parameters

Table I. SA Placement Followed by SA Routing

Circuit	# Placed Nodes	#Routes	Grid Size	Tile Size	# Diodes (D)	# Switches (S)	Run Time (secs)
9sym	282	601	100 × 100	25 × 25	22752	21642	2
alu4	2084	5408	300 × 300	75 × 75	2864678	1947219	1459
apex1	1356	2918	200 × 200	50 × 50	253120	188127	28
apex2	2520	6740	300 × 300	75 × 75	4957648	1954684	8051
apex4	1919	4771	200 × 200	50 × 50	509565	396629	43
C17	18	26	10 × 10	4 × 4	39	43	0
C1908	743	1255	100 × 100	25 × 25	30992	28713	2
C432_N	261	372	50 × 50	12 × 12	5369	6148	1
C6288	2608	4896	250 × 250	60 × 60	233420	221615	13
C880	493	748	100 × 100	25 × 25	26472	24346	5
Cordic	2593	5221	300 × 300	75 × 75	632877	517453	4500
Count	200	270	50 × 50	12 × 12	4498	4358	1
cm138a	27	40	10 × 10	4 × 4	102	90	0
decod	50	81	60 × 60	15 × 15	783	752	0
ex5p	1645	4206	200 × 200	50 × 50	432012	410948	45
example2	495	702	100 × 100	25 × 25	75740	60240	4
majority	18	22	10 × 10	4 × 4	60	64	0
misex3	2163	5110	250 × 250	60 × 60	2074896	1578837	1911
sqrt8ml	222	370	60 × 60	15 × 15	7524	7619	1
tseng	1619	3598	300 × 300	75 × 75	635761	522543	84
z4ml	58	79	15 × 15	4 × 4	250	298	2

that produce optimal results are empirically found to be initial temperature $T_0 = 3000$, cooling factor $\alpha = 0.75$, iteration amplifier $\beta = 1.6$, initial number of iterations $I_0 = 100$, and total annealing runtime of 1,000 units, which is the number of times the outer loop in the algorithm is executed. In Table I, the first column shows the benchmarks, followed in column 2 by the number of nanoblocks to which the logic is mapped. The number of source-sink pairs for the nets in a design is shown in column 3. The size of the nanofabric grids used for the layout in terms of number of nanoblocks and switch blocks is shown in the next column, followed by design-specific empirically-determined sizes of tiles (refer to Section 6.1) in column 5. The size of each tile that the nanofabric is divided into has been optimally found to use the minimum number of diodes and switches, when

$$Size = (number\ of\ rows\ in\ nanofabric/4) * (number\ of\ columns\ in\ nanofabric/4).$$

The next two columns in Table I show the number of diodes and switches used in the design implementation. Runtimes for our optimization in seconds are shown in the last column. Since most of our benchmarks have high routing densities, we found that a nanoblock with 100 horizontal and 100 vertical wires is suitable for completely routing all circuits. Note that in more congested benchmarks as many as 50% of the wires are used for passing variables.

In Table II, we have compared the number of diodes and switches used for design implementations with the initial zig-zagging placement of Figure 5 and with no placement or routing optimization. For a placed circuit, routes are found using the method described in the first paragraph of Section 5, and the

Table II. Results of Initial Placement and No Optimization

Circuit	#Diodes (D)	#Switches (S)	%Increase (D + S)
9sym	26405	22389	9.91
alu4	Routes not found		—
apex1	Routes not found		—
apex2	Routes not found		—
apex4	Routes not found		—
C17	45	46	10.98
C1908	32596	30471	5.63
C432_N	6505	6395	12.01
C6288	Routes not found		—
C880	32864	28473	20.7
Cordic	Routes not found		—
Count	4627	4452	2.52
cm138a	114	91	6.77
decod	892	802	10.37
ex5p	Routes not found		—
example2	Routes not found		—
majority	70	71	13.71
misex3	Routes not found		—
sqrt8ml	8631	7971	14.86
tseng	Routes not found		—
z4ml	252	315	3.47

optimizations of Section 6.2 are not used. The first valid route found for a net is chosen. Note that in some cases no routes were found, and hence the number of diodes and switches used cannot be calculated. For those benchmarks for which routing can be completed, we saw that the approach of Table I reduced the number of diodes and switches used by 10% on average. This is not an indication of the true effectiveness of the optimization method used to generate Table I because for the bigger and more complex benchmark circuits where our optimizations are more effective, it was not possible to complete routing.

Finally, the MCNC benchmark circuits are placed and routed on a nanofabric using our integrated placement and routing algorithm of Section 6.3. As shown in Table III, a reduction of as much as 24% in the number of diodes and switches used is achieved using the integrated optimizations compared to the method of Table I. Note that the largest improvements are in the bigger benchmarks; this implies the effectiveness and scalability of our methodology. On average, a 12% reduction in the number of diodes and switches used is achieved by the integrated method compared to an approach with separate placement and routing optimizations. The average number is pessimistic because it has been lowered by the inclusion of several smaller benchmarks where the optimizations could not improve results significantly. For both Tables I and III, we have kept the tile size and the grid size for respective circuits unchanged so that a comparative analysis can be done.

8. CONCLUSIONS AND ONGOING WORK

In this work, we present an integrated placement, net topology generation, and routing algorithm for minimizing the number of diodes and switches used for

Table III. Integrated SA Placement and Routing

Circuit	# Placed Nodes	# Routes	Grid Size	Tile Size	# Diodes (D)	# Switches (S)	Run Time (secs)	% Red (D + S)
9sym	283	602	100 × 100	25 × 25	20792	20973	5	6.33
alu4	2085	5409	300 × 300	75 × 75	1799041	1847219	2500	24.22
apex1	1357	2919	200 × 200	50 × 50	190950	188061	55	14.11
apex2	2521	6741	300 × 300	75 × 75	3419311	2164693	12000	19.22
apex4	1920	4772	200 × 200	50 × 50	375892	396514	84	14.76
C17	19	26	10 × 10	4 × 4	31	37	0	17.07
C1908	743	1255	100 × 100	25 × 25	26978	28662	4	7.31
C432_N	261	372	50 × 50	12 × 12	5089	6092	2	2.92
C6288	2608	4896	250 × 250	60 × 60	208163	211614	27	7.75
C880	493	748	100 × 100	25 × 25	23834	24002	10	5.88
Cordic	2593	5221	300 × 300	75 × 75	500695	510288	9000	12.11
Count	200	270	50 × 50	12 × 12	4304	4328	3	2.53
cm138a	27	40	10 × 10	4 × 4	80	82	0	15.63
decod	50	81	60 × 60	15 × 15	638	636	1	17
exp5	1645	4206	200 × 200	50 × 50	371938	334777	90	16.16
example 2	495	702	100 × 100	25 × 25	66069	60222	9	7.13
majority	18	22	10 × 10	4 × 4	49	53	0	10.53
misex3	2163	5110	250 × 250	60 × 60	1957194	1368838	3800	8.97
sqrt8ml	222	370	60 × 60	15 × 15	6943	7511	2	4.55
tseng	1619	3598	300 × 300	75 × 75	570753	452496	170	13.89
z4ml	59	80	15 × 15	4 × 4	192	235	5	22.08

design implementations on molecular diode-based chemically self-assembled nanofabrics. Constraint equations that reflect the unique properties of the underlying nanofabric such as the relations between routing and the number of diodes used for realizing logic inside nanoblocks are identified. A scalable and practical approach based on simulated annealing is implemented for the optimization of the algorithm, while incorporating these constraints. Experimental results using MCNC benchmark circuits show that, on average, the integrated optimization reduces the number of diodes and switches by 12% when compared to designs where placement and routing optimizations are done separately. The reduction achieved is as high as 24% for the *alu4* benchmark, and, in general, we observed that the bigger the benchmark, the larger the reduction in the number of active devices. Although we have selected a nanofabric architecture based on Goldstein and Budiu [2001] and the nanoblock sizes are based on the benchmark characteristics, our NanoEDA design flows enable different nanoarchitectural explorations. Similar routing-dependent logic constraints are thought to exist in other nanofabric architectures. The integrated placement and routing techniques proposed in this work can potentially be extended to these emerging nanoarchitectures. Future work in this area can be benchmarked against our results.

Our current work is focused on developing a nanofabric-specific logic synthesis frontend tool for the NanoEDA flow of Section 4. The other area of related research we are doing is in introducing fault tolerance in the NanoEDA flow. In this article, we are assuming all nanoblocks and switch blocks operate without faults. Our future work will incorporate probabilistic distributions for both

transient and permanent faults in our SA-based algorithm. The placement and routing algorithm will be modified to achieve improved performance and area under faulty conditions.

REFERENCES

BETZ, V. AND ROSE, J. 1997. VPR: A new packing, placement and routing tool for FPGA research. In *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*. 213–222.

BETZ, V., ROSE, J., AND MARQUARDT, A. 1999. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Boston, MA.

CHEN, J., REED, M. A., RAWLETT, A. M., AND TOUR, J. M. 1999. Observation of a large on–off ratio and negative differential resistance in an electronic molecular switch. *Science 286*, 1550–1552.

CHEN, J. AND REED, M. A. 2000. Molecular wires, switches and memories. In *Proceedings of the International Conference on Molecular Electronics*, December 2000, Kona, HI.

COLLIER, C. P., MATTERSTEIG, G., WONG, E., LUO, Y., BEVERLY, K., SAMPAIO, J. S., RAYMO, F., STODDART, J., AND HEATH, J. 2000. A catenane-based solid state reconfigurable switch. *Science 289*, 1172–1175.

COLLIER, C. P., WONG, E. W., BELOHRADSKY, M., RAYMO, F. M., STODDART, J. F., KUEKES, P. J., WILLIAMS, R. S., AND HEATH, J. R. 1999. Electronically configurable molecular-based logic gates. *Science 285*, 391–394.

CUI, Y. AND LIEBER, C. 2001. Functional nanoscale electronic devices assembled using silicon nanowire building blocks. *Science 291*, 851.

CUI, Y., LIEBER, C., LAUHON, L., GUDIKSEN, M., AND WANG, J. 2001. Diameter controlled synthesis of single crystal silicon nanowires. *Appl. Physics Lett. 78*, 15, 2214–2216.

DATTA, K., MUKHERJEE, A., AND RAVINDRAN, A. 2005. Routing for reliability in molecular diode-based nanofabrics. In *Proceedings of the 8th Military and Aerospace Programmable Logic Device International Conference* (Sept.).

DEHON, A. 2005. Design of programmable interconnect for sublithographic programmable logic arrays. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays* (Feb.). 127–137.

DEHON, A. 2003. Array-based architecture for FET-based, nanoscale electronics. *IEEE Trans. Nanotech. 2*, 1, 23–32.

DEHON, A., LINCOLN, P., SAVAGE, J. E. 2003. Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Trans. Nanotech. 2*, 3, 165–174.

DERYCKE, V., MARTEL, R., APPENZELLER, J., AND AVOURIS, P. 2001. Carbon nanotube inter- and intramolecular logic gates. *Nano Lett. 1*, 9, 453–456.

GOLDSTEIN, S. C. AND BUDIU, M. 2001. Nanofabrics: Spatial computing using molecular electronics. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*. Gotenbörg, Sweden (June). 178–189.

GOLDSTEIN, S. C. AND ROSEWATER, D. 2002. What makes a good molecular scale computer device? Tech. rep. School of Computer Science, Carnegie Mellon University, CMU-CS-02-181 (Sept.).

GUDIKSEN, M. S., WANG, J., AND LIEBER, C. M. 2001. Synthetic control of the diameter and length of semiconductor nanowires. *J. Physical Chem. B 105*, 4062–4064.

HUANG, Y., DUAN, X., CUI, Y., LAUHON, L., KIM, K.-H., AND LIEBER, C. 2001a. Logic gates and computation from assembled nanowire building blocks. *Science 294*, 1313.

HUANG, Y., DUAN, X., WEI, Q., AND LIEBER, C. 2001b. Directed assembly of one dimensional nanostructures into functional networks. *Science 291*, 630–633.

KAMINS, T. I., WILLIAMS R. S., CHEN, Y., CHANG, Y.-L., AND CHANG Y. A. 2000. Chemical vapor deposition of Si nanowires nucleated by TiSi2 islands on Si. *Appl. Physics Lett. 76*, 562–564.

KIRKPATRICK, S., GELATT, C. JR., AND VECCHI, M. 1983. Optimization by simulated annealing. *Science 220*, 671–680.

LUO, Y., COLLIER, C. P., NIELSEN, K., JEPPESEN, J., PERKINS, J., DEIONNO, E., PEASE, A., STODDART, J. F., AND HEATH, J. R. 2002. Molecular electronics random access memory circuits. *J. ChemPhysChem 3*, 519.

MA, X., STRUKOV, D. B., LEE, J. H., AND LIKHAREV, K. K. 2005. Afterlife for silicon: CMOL circuit architectures. In *Proceedings of the 5th IEEE Conference on Nanotechnology 1*, 11–15 (May), 175–178.

MCNC. url: www.mcnc.org.

MISHRA, M. AND GOLDSTEIN, S. C. 2003. Defect tolerance at the end of the roadmap. In *Proceedings of International Test Conference*. 1201–1210.

MORALES, A. AND LIEBER, C. 1998. A laser ablation method for the synthesis of crystalline semiconductor nanowires. *Science 279*, 208–211.

RASP-SYN. url: http://ballade.cs.ucla.edu/software_release/rasp/htdocs.

RUECKES, T., KIM, K., JOSELEVICH, E., TSENG, G. Y., CHEUNG, C.-L., AND LIEBER, C. M. 2000. Carbon nanotube based nonvolatile random access memory for molecular computing. *Science 289*, 94–97.

SEMICONDUCTOR INDUSTRIES ASSOCIATION ROADMAP. http://public.itrs.net.

SOH, C., QUATE, C., MORPURGO, C., MARCUS, C., KONG, C., AND DAI, C. 1999. Integrated nanotube circuits: Controlled growth and ohmic contacting of single-walled carbon nanotubes. *Appl. Physics Lett. 75*, 5, 627–629.

STAN, M. R., FRANZON, P. D., GOLDSTEIN, S. C., LACH, J. C., AND ZIEGLER, M. M. 2003. Molecular electronics: From devices and interconnect to circuits and architecture. In *Proceedings of the IEEE Conference on Molecular Electronics* (Nov). 1940–1957.

STRUKOV, D. B. AND LIKHAREV, K. K. CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *IEEE Trans. Nanotechn.* To appear.

TRANS, S. J., VERSCHUEREN, A. R. M., AND DEKKER, C. 1998. Room-temperature transistor based on a single carbon nanotube. *Nature 393*, 49–51.

VPR AND VPACK. url: http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html.

WIND, S. J., APPENZELLER, J., MARTEL, R., DEYCKE, V., AND AVOURIS, P. 2002. Vertical scaling of carbon nanotube field-effect transistors using top gate electrodes. *Appl. Physics Lett. 80*, 20, 3817–3819.

SIS LOGIC SYNTHESIS PACKAGE. url:www-cad.eecs.berkeley.edu.

Received November 2005; revised June 2006; accepted July 2006 by Krishnendu Chakrabarty