Int J Parallel Prog (2009) 37:389–416
DOI 10.1007/s10766-009-0105-x

Design and Evaluation of a Carbon Nanotube-Based Programmable Architecture

Scott Chilstedt · Chen Dong · Deming Chen

Received: 17 January 2009 / Accepted: 16 April 2009 / Published online: 28 May 2009

Abstract In the hunt to find a replacement to CMOS, material scientists are developing a wide range of nanomaterials and nanomaterial-based devices that offer significant performance improvements. One example is the Carbon Nanotube Field Effect Transistor, or CNFET, which replaces the traditional silicon channel with an array of semiconducting carbon nanotubes (CNTs). Given the increased variation and defects of nanometer-scale fabrication, and the regular nature of bottom-up self-assembly, field programmable devices are a promising initial application for such technologies. In this paper, we detail the design and evaluation of a novel nanomaterial-based architecture called FPCNA (Field Programmable Carbon Nanotube Array). New nanomaterial-based circuit building blocks are developed and characterized, including a lookup table created entirely from continuous CNT ribbons. To accurately determine the performance of these building blocks, we create variation-aware physical design tools with statistical timing analysis that can handle both Gaussian and non-Gaussian random variables. When the FPCNA architecture is evaluated using this CAD flow, we see a 2.75× performance improvement over an equivalent CMOS FPGA at a 95% yield. In addition, FPCNA offers a 5.07× footprint reduction compared to the baseline FPGA.

S. Chilstedt (✉) · C. Dong · D. Chen
Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: chilste1@illinois.edu

C. Dong
e-mail: cdong3@illinois.edu

D. Chen
e-mail: dchen@illinois.edu

Keywords FPCNA · FPGA · Nanoelectronics · Variation-aware CAD · Discretized SSTA

1 Introduction

At and below the 22 nm process node, the conventional top-down manufacturing process faces serious challenges due to physical device limitations and rising production costs. Shifting to a bottom-up approach, in which self-assembled nanoscale building blocks such as carbon nanotubes (CNTs) are combined to create integrated functional devices, offers the potential to overcome these challenges and revolutionize electronic system fabrication.

The nature of the chemical synthesis process allows smaller feature sizes to be defined, but offers less control over individual device location. This leads to very regular structures, making it ideal for creating the repetitive architectures found in Field Programmable Gate Arrays (FPGAs). In addition, the programmability of FPGAs allows reconfiguration around the large number of fabrication defects inherent in nanoscale processes, which helps to provide the high level of fault tolerance needed for correct nanocircuit operation.

In this paper, we propose a new carbon nanotube based FPGA architecture called FPCNA (Field Programmable Carbon Nanotube Array). We detail the building blocks of FPCNA, including the carbon nanotube lookup table which makes up its programmable logic. We also describe a high-density routing architecture using a recently proposed nanoswitch device. In our design, special considerations are made to mitigate the negative effects of nano-specific process variations. We characterize our components considering these variations, as well as circuit-level delay variations.

To evaluate the performance of our architecture, we adopt a typical FPGA design flow and develop variation-aware placement and routing algorithms. These algorithms are enhanced from the popular physical design tool VPR [1], and use statistical timing analysis (SSTA) to improve the performance yield. We perform SSTA with both normal and non-Gaussian variation models. Our results show that FPCNA offers significant performance and density gains compared to the conventional CMOS FPGA, demonstrating potential for the use of CNT devices in next-generation FPGA circuits.

This paper is organized as follows: In Sect. 2, we review the related works in nano-FPGA architecture. An overview of our design flow is provided in Sect. 3. Section 4 introduces the nanoelectronic devices used in our designs. In Sect. 5, we present the FPCNA architecture. Section 6 discusses the fabrication of our CNT-based lookup table. Circuit-level characterization results are presented in Sect. 7. These results provide meaningful guidance for our CAD flow, which is detailed in Sect. 8. In Sect. 9 we present experimental results showing the advantages of FPCNA over a conventional CMOS FPGA, and Sect. 10 concludes this paper.

2 Related Work

There have been a number of nanomaterial-based programmable architectures proposed in past literature. In an early work [2], Goldstein and Budiu presented an

island-style fabric in which clusters of nanoblocks and switch blocks are interconnected in an array structure. Each nanoblock consists of a grid of nanowires which can be configured to implement a three-bit input to three-bit output Boolean function. Routing channels are created between clustered nanoblocks to provide low-latency communication over longer distances. A comprehensive PLA-based architecture known as NanoPLA was presented by DeHon in [3]. This architecture builds logic from crossed sets of parallel semiconducting nanowires. Axial doping is used to address the individual nanowires, and OR-plane crossbars are programmed by applying a voltage across a pair of crossed nanowires. Nanowire field-effect transistor (FET) restoring units are used at the output of the programmable OR-planes to restore the output signals and provide inversion. Snider et al. proposed a CMOS-like logic structure based on nanoscale FETs in [4], where nanowire crossbars were doped to create arrays of either n-type and p-type semiconductors. These crossbars are made from horizontal metallic wires and vertical semiconducting wires and can be connected using undoped programmable switch arrays to create AND-OR-INVERT functions.

CMOS-Nano hybrid FPGAs have also been explored. In [5], an architecture was presented by Gayasen et al. that combines CMOS logic blocks/clusters with different types of nanowire routing elements and programmable molecular switches. The results show that this architecture could reduce chip area by up to 70% compared to a traditional CMOS FPGA architecture scaled to 22 nm. Using an opposite approach, Rad et al. presented a FPGA with nanowire logic clusters in [6], where the inter-cluster routing remains in CMOS. Similar to Gayasen et al., up to 75% area reduction is seen, with performance comparable to a traditional FPGA. In reference [7], an innovative cell-based architecture was introduced by Strukov and Likharev. This architecture, called CMOL, uses specially doped silicon pins on the upper layers of a CMOS stack to provide contacts to a nanowire array interconnect layer. Logic functions are implemented by a combination of CMOS inverter arrays and nanowire OR-planes based on molecular-switches. Interconnect signals can also be routed through the nanowires by connecting specific crosspoints. A generalized CMOL architecture called FPNI was proposed by Snider et al. in [8]. Unlike CMOL's inverter array architecture, the logic in FPNI is implemented in CMOS arrays of NAND/AND functions paired with buffers and flip-flops, and the nanowires are used only for routing. FPNI also improved upon the manufacturability of CMOL by requiring less alignment accuracy between the CMOS and nanowire layers. In reference [9], Dong et al. proposed a 3D nano-FPGA architecture that distributes the components of a 2D FPGA into vertically stacked CMOS and nanomaterial layers. To connect the layers, vias made from CNT bundles were used to transmit signals and dissipate heat. Researchers have also proposed using carbon nanotube based memory (i.e., NRAM [10–12]) as block storage for FPGA configuration data [13], interconnect switch memory [9], and FPGA LUT memory [14].

While many of the aforementioned studies demonstrate the use of nanowire crossbars for logic and interconnect, none of them explore the possibility of using carbon-based nanoelectronics to implement FPGA logic. In addition, none of the previous designs were evaluated using a variation-aware CAD flow to predict the performance yield under the large variations in nanoscale fabrication.

3 Design Flow

The development of a nanomaterial-based programmable device involves a number of design steps. To show the relationships between these steps, we illustrate the overall flow in Fig. 1. In this figure, our contributions are highlighted by the dashed box that bridges the development of nanoelectronic devices with the fabrication of nanocircuits.

Using the latest advances in nanoelectronic device research, we design low-level circuit elements such as logic and memory, and high-level architectures such as programmable logic tiles and interconnect structures. Circuit characterization is then used to model the behavior of these structures including their delay under variation. Because these tasks are highly dependent on the nanomaterials used, we refer to them as nano-centric design. CAD evaluation enhances nano-centric design, allowing us to test benchmark circuits on these architectures and explore the trade-offs of architectural parameters. The end results are new nanoelectronic designs with satisfactory performance in terms of area, delay, and yield. These designs provide an early assessment of the feasibility of nanomaterial-based devices and circuits.

4 Nanoelectronic Devices

4.1 Carbon Nanotubes

Single-walled carbon nanotubes (SWCNTs) are molecular devices with a diameter of roughly 1 nm. They are composed of hexagonal carbon rings which form a seamless

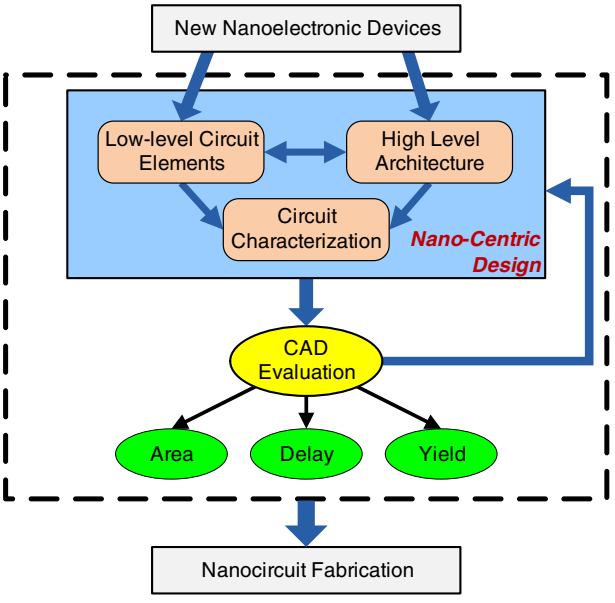


Fig. 1 Overall design flow

cylinder, and depending on the chirality of the lattice can be either metallic or semiconducting [15].

Metallic SWCNTs have high electron mobility and robustness, making them attractive for uses in nanoscale interconnect and nano-electromechanical systems (NEMS). Semiconducting SWCNTs, on the other hand, have ideal characteristics for use in field-effect transistors [16]. In the past, it was difficult to mass produce CNT-based circuits because of the inability to accurately control nanotube growth and position. However, recent research has demonstrated the fabrication of dense, perfectly aligned arrays of linear SWCNTs [16], and wafer scale CNT-based logic devices [17,18].

4.2 CNT Field Effect Transistors

The structure and operation of a CNT-based field effect transistor (CNFET) is analogous to that of a silicon-based device. A semiconducting CNT forms the conducting channel between the source and drain contacts, and is controlled by a gate electrode. In the past decade, several groups have reported that CNFETs are a promising post-silicon technology [19,20]. Based on intrinsic CV/I gate delay, CNFET devices can be up to 13× and 6× faster than pMOS and nMOS devices of the same gate length, when local interconnect capacitances and CNT imperfections are not considered [21].

4.3 NRAM

NRAM is a nonvolatile NEMS memory device formed by suspending a metallic CNT over a trench which contains a base electrode. The off state is characterized by the CNT lying flat across the trench where elastic energy keeps the tube in place. The on state occurs when the CNT is bent into the trench and makes contact with the base electrode. In the on state, the van der Waals force between the CNT and the trench floor creates a strong molecular attraction, overpowering the elastic energy. Since these interactions are purely molecular, no power is consumed when the memory is at rest. Programming is accomplished by applying either attractive or repulsive voltages at the CNT and base electrode. This creates an electro-mechanically switchable, bi-stable memory device with well-defined off and on states [10,11].

4.4 CNT-Bundle Interconnect

As integrated circuit dimensions scale down, the resistivity of copper (Cu) interconnect increases due to electron surface scattering and grain-boundary scattering, leading to a communication bottleneck. Metallic CNTs are a promising replacement because they offer superior conductivity and current carrying capabilities [22,23]. Since individual SWCNTs can have a large contact resistance, a rope or bundle of SWCNTs is used to transfer current in parallel. SWCNT bundle interconnect can outperform copper interconnect for propagation delay, especially for intermediate and long interconnects [24].

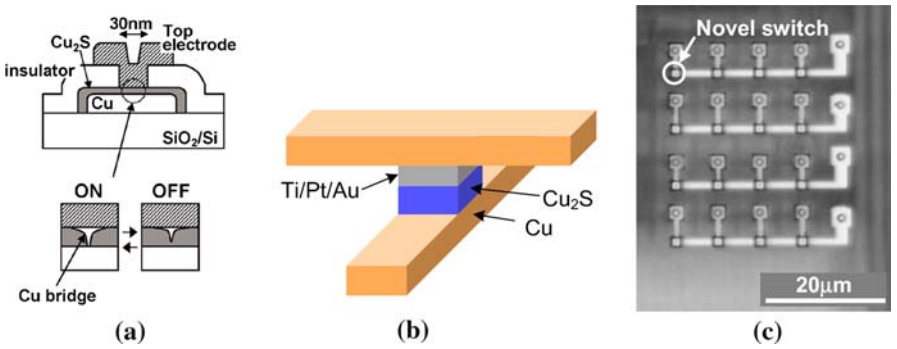


Fig. 2 **a** Solid-electrolyte switch, **b** Implementation in metal interconnect, **c** SEM image of a 4×4 crossbar switch array

4.5 Solid-Electrolyte Nanoswitches

Solid-Electrolyte switches are a new type of nano-scale switch developed by [25]. A Solid-Electrolyte switch is created by sandwiching a layer of Cu_2S between two metals, a top electrode (Ti, Pt, or Au) and bottom layer of Cu (Fig. 2a).

When a negative voltage is applied at the top electrode, Cu ions in the Cu_2S are electrochemically neutralized by the electrons coming from that electrode and a conductive bridge between the two electrodes is created, turning the switch on. An on-state resistance of as low as 50 ohms can be achieved by continually applying negative voltage to make the nano-bridge thicker. Similarly, the bridge can be ionized and dissolved by applying a positive voltage to the top electrode, turning the switch off.

Because this design does not depend on a substrate, the switches can be manufactured between the higher layers of metal interconnect that are used for routing, as shown in Fig. 2b. This figure shows how a Cu interconnect line can serve as the bottom layer in the electrolyte switch. In addition to individual devices, crossbars can be made from switch arrays. An SEM image of a prototype 4×4 crossbar is shown in Fig. 2c, from [25].

5 FPCNA Architecture

In this section we describe the FPCNA architecture in detail. We begin with the introduction of a new LUT design based on carbon nanotubes. Then we describe a Basic Logic Element (BLE) that can be created using this LUT design. Finally, we discuss FPCNA's high level architecture, including the design of local and global routing.

5.1 CNT-Based LUT

A K-input lookup table (K-LUT) is the basic unit of programmable logic in modern FPGAs. For FPCNA, we propose a novel K-LUT design based entirely on carbon nanotube devices. Profile and overhead views of this device are shown in Fig. 3. This

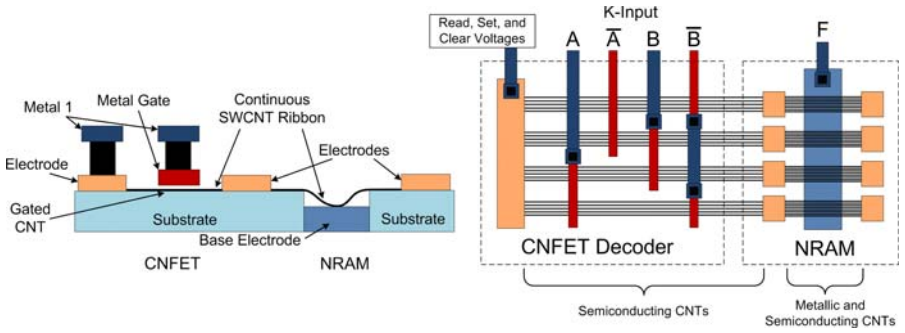


Fig. 3 CNT-based LUT

design uses parallel ribbons of SWCNTs held in place by metal electrodes and crossed by metal gates. PMOS CNFET devices are formed at the crossing points of the CNT ribbons and the metal gates, creating a CNFET decoder. At points where the CNT ribbons pass over a trench in the substrate, NRAM memory devices are formed. This CNT memory is used to store the truth table of the BLE's logic function. By applying K-inputs to the decoder, a reading voltage will be sent to the corresponding memory bit whose output can then be read from the base electrode.

One of the key innovations of this LUT design is that it builds the decoding and memory on the same continuous CNT ribbons. This structure allows for high logic density and simplifies the manufacturing process. For comparison, the work in [14] uses an LUT memory based on individually-crossed nanotubes that is addressed by a CMOS multiplexor tree. In addition to being more costly in area, this design suffers from fabrication issues because it requires the alignment and interfacing of individual nanotubes in two dimensions.

By using CNT ribbons, we also have the advantage that each device will contain multiple tubes. This adds fault tolerance from the high defect rates of nanotube fabrication, and increases the chance that a CNFET or NRAM device will contain functioning nanotubes. Thus, the design is more reliable than in [14] where a device will fail if either of the two nanotubes is defective.

5.2 BLE Design

In Fig. 3, a 2-to-4 (2 input, 4 NRAM cell) LUT was shown for illustration purposes. In modern FPGAs, each Basic Logic Element (BLE) typically contains a 4-to-16 LUT, as well as a flip-flop (FF) and multiplexor (MUX) to allow registered output. When scaled to K inputs, our LUT will contain 2^K CNT ribbons. The BLE design we use for FPCNA is shown in Fig. 4. In this figure, we expand the LUT to four inputs and add supporting CMOS logic for voltage control, address line inversion, and registered output.

In the decoder, we use Gray address decoding to minimize the number of gate-to-metal1 transitions. Compared to binary decoding, this reduces the number of vias by 46% (from 48 to 26). Since our LUT depends on both normal and complemented inputs, inverters are added for each of the address line inputs. A buffer is used to restore the output signal before it passes to the flip-flop and MUX.

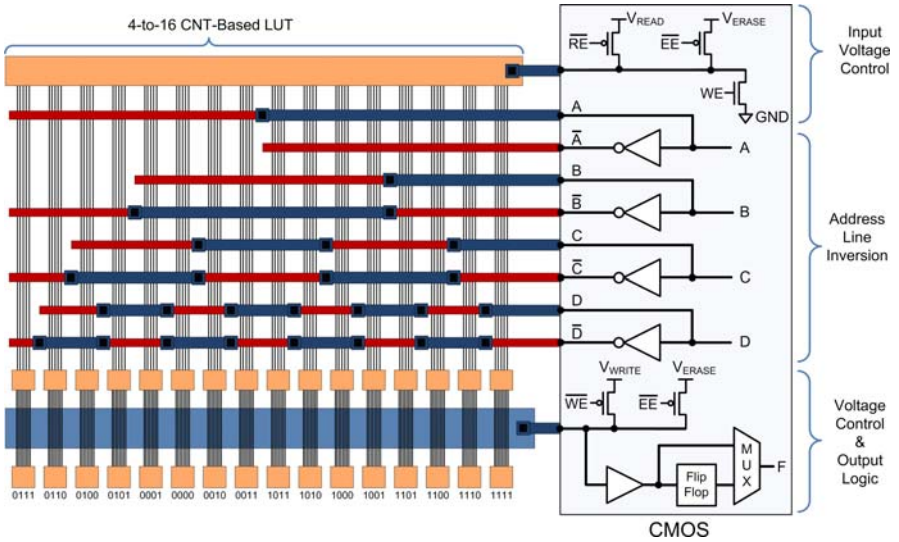


Fig. 4 FPCNA BLE with a 4-to-16 CNT-based LUT

Table 1 NRAM operating modes

Mode	RE	WE	EE	Ribbon voltage	Base electrode voltage
Reading	High	Low	Low	V_{READ} (1 V)	Output
Writing	Low	High	Low	Ground (0 V)	V_{WRITE} (1.6 V)
Erasing	Low	Low	High	V_{ERASE} (+2.5 V)	V_{ERASE} (+2.5 V)

To read and program NRAM, three different voltage configurations are needed. CMOS pass transistors are used to configure the three modes. Table 1 shows the pass transistor enable signals and voltages during each mode. Most often, the circuit will be in the reading mode with the RE (read enable) signal set. This allows V_{READ} to pass through the decoder and select the appropriate NRAM bit. If the NRAM bit is set, the signal will pass through the relatively low resistance of the nanotubes contacting the base electrode (logical 1). If the NRAM bit is not set, a multiple GΩ resistance will prevent transmission (logical 0).

To program a value, RE is deactivated, and either WE (write enable) or EE (erase enable) is activated. When WE is set, the selected CNT ribbon is grounded, and V_{WRITE} is applied to the base electrode. The difference in potential creates an attractive force, which pulls the ribbon down into the trench. In [11], Nantero measured a threshold voltage of $1.4\,V \pm 0.2\,V$, so we assume a V_{WRITE} of 1.6 V to switch a majority of the tubes. When erasing, the same voltage (V_{ERASE}) is applied to both the ribbon and the base electrode. The like voltages repel each other, releasing the ribbon from the trench floor and allowing it to return to an unbent state. V_{ERASE} must be somewhat larger than V_{WRITE} [10,11], so we assume a value of +2.5 V. There is a risk that this voltage applied to the base electrode could attract an unselected ribbon, causing an unintentional write. This can be avoided by erasing all of the NRAM bits. As each

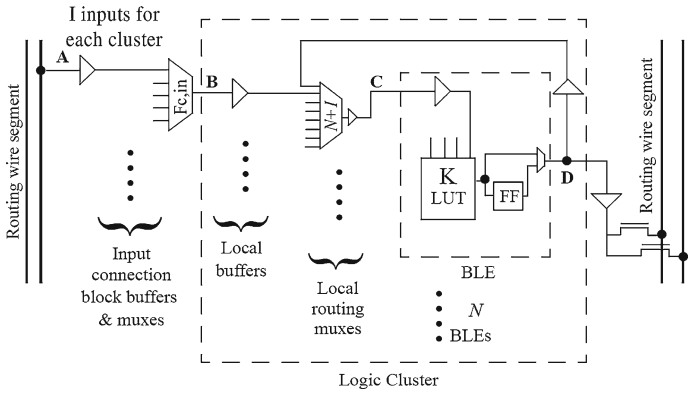


Fig. 5 Schematic of a CLB (logic cluster)

bit is erased, its ribbon will be temporarily charged to +2.5 V, repelling it from the electrode during the erasure of the remaining bits. Then the individual bits can be written.

5.3 Logic Block Design

For FPCNA, we use a cluster-based configurable logic block design. Each configurable logic block (CLB) contains N of the BLEs described in the previous section (where N is the cluster size), as well as the local routing used to connect the BLEs together. In conventional CMOS FPGA designs, the routing is often multiplexor-based. An example of this is shown in Fig. 5, a CLB schematic from [1]. While we could adopt the same approach, using CMOS for the MUXs and NRAM to store multiplexor configuration bits, we believe a greater logic density can be achieved by using solid-electrolyte switch crossbars.

Figure 6 shows a simplified CLB design to illustrate this technique. The CLB in this figure contains four BLEs made from CNT-based LUTs. The local routing is created with solid-electrolyte switches created at the crosspoints of the vertical and horizontal routing wires. By programming the nanoswitch points, a BLE output can be routed to any BLE input. In Fig. 6, one of the input signals to BLE 1 is identified with a dashed line labeled 'Input to BLE'. The black dots at crosspoints indicate that solid-electrolyte switches at those locations are turned on. By using more switches, the same signal can be routed to multiple BLE inputs. Output from a BLE can connect to the inputs of other BLEs or be output from the CLB. Note that Fig. 6 shows the local routing positioned between BLEs for clarity. In an actual implementation, the local routing and routing switches can be made above the BLEs, and our area calculations reflect this.

5.4 High Level Architecture and Global Routing

We adopt a conventional island-based FPGA architecture for the high level organization of FPCNA. The basic structural unit is a tile, consisting of one programmable

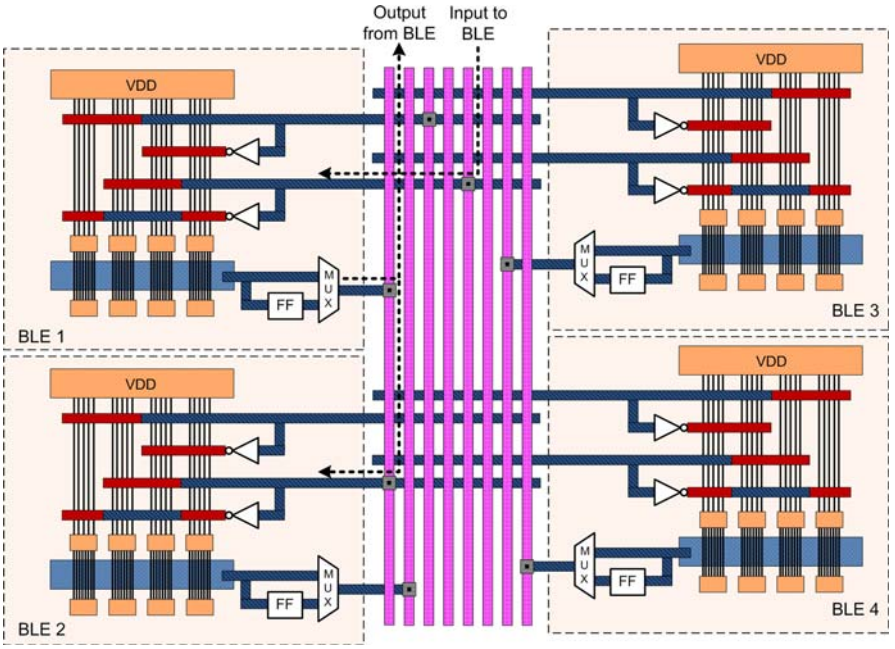


Fig. 6 CNT-based CLB with nanoswitch local routing

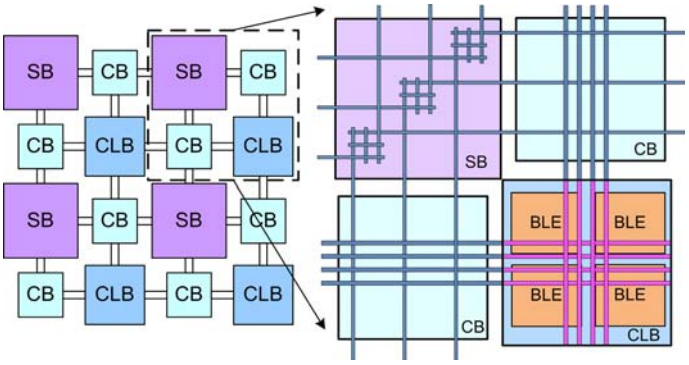


Fig. 7 High-level layout of FPCNA

switch block (SB), two connection blocks (CB), and one configurable logic block (CLB). This tile is replicated to create the FPGA fabric, as shown in Fig. 7.

The global routing structure consists of two-dimensional segmented interconnects connected through programmable SBs and CBs. The CLBs are given access to these channels through connections in the CBs. The parameter I represents the number of inputs to a CLB, and Fc defines the number of routing tracks a CLB input can connect to. We use CNT-bundle interconnects for global routing because they have been shown to be superior to copper in terms of current density and delay [22].

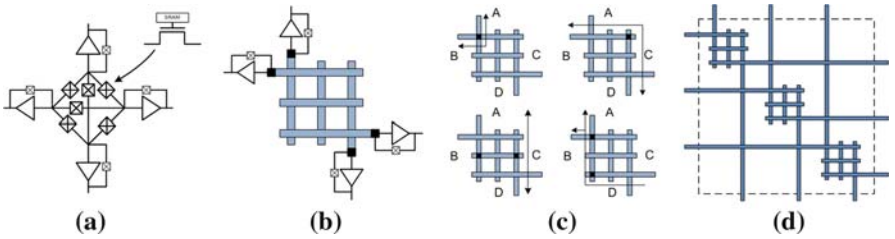


Fig. 8 **a** CMOS switch point, **b** Nanoswitch-based switch point with CMOS driving buffers, **c** Example switching scenarios, **d** 3 × 3 switch block (driving buffers not shown)

In a traditional CMOS-based FPGA, the SBs and CBs take up the majority of the overall area [26]. For example, if the CLB size is 10 and the BLE size is 4 (popular parameters for commercial FPGA products), the global routing takes 57.4% of the area, with the CLBs occupying the remaining 42.6% [26]. To reduce the size of the global routing in FPCNA, we replace the traditional CB with a solid-electrolyte switch crossbar, and propose a new nanoswitch-based SB design.

The new SB design is shown in Fig. 8. Instead of using six SRAM-controlled pass transistors for each switch point as in conventional CMOS designs (Fig. 8a [1]), we use six perpendicular wire segments with solid-electrolyte nanoswitches at the crosspoints. In this design, we keep the driving buffers and input control pass transistors in CMOS, as shown in Fig. 8b. By programming nanoswitches at the crosspoints of the wire segment array, a signal coming from one side of the block can be routed to any or all of the other three sides. To demonstrate how routing connections can be made, four switching scenarios are illustrated in Fig. 8c. In the figure, arrows represent signal directions and black dots indicate the activated switches. The upper left scenario shows how signals A and B are connected using a single switch. A multipath connection is demonstrated in the lower right scenario, where a signal from C is driving both A and B. By turning on the appropriate nanoswitches, any connection of signals can be made. Using these switch points, larger switch blocks can be constructed. For example, the 3 × 3 universal-style switch block in Fig. 8d is made from three nanoswitch-based switch points. This design can be scaled to any routing channel width, and significantly reduces the SB area. In a conventional CMOS switch point (Fig. 8a, center), six 10× pass transistors are controlled by six SRAM cells, which normally requires an area of $88.2T$ (where T is the area of a minimum-size transistor). When using nanoswitch-based switch points, the same routing function can be achieved in approximately $9T$.

6 Nanotube Lookup Table Fabrication

Recent progress in the fabrication of CNTs has enabled us to explore the use of CNT-based structures in FPCNA's LUT design. To demonstrate the feasibility of this design at the 32 nm technology node, we address some of the fabrication issues involved.

The first step in manufacturing the LUT is to define the NRAM trench in the silicon wafer using a process similar to the one described in [11]. Then the nanotubes are grown on separate quartz wafers using chemical vapor deposition. Since the desired

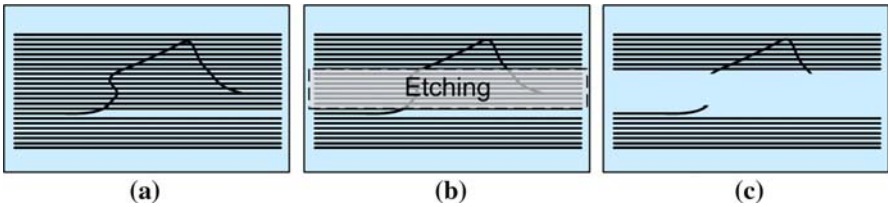


Fig. 9 CNT ribbon etching

CNT ribbons are all aligned in the same direction, an array-based CNT growth process can be used. In [16], researchers report a technique for fabricating dense, perfectly aligned arrays of CNTs using photolithographically defined catalytic seeds, which achieves an alignment of up to 99.9%. The aligned nanotubes can then be transferred to a silicon wafer using a stamping process like the one developed in [27]. These techniques create nanotubes that are suitable for the transistors and NEMS devices used in our LUT. In addition, it is possible to improve nanotube density on the silicon wafer by performing multiple consecutive transfers. In this study, we assume a multiple transfer process is used that provides a CNT pitch of 4 nm.

After the nanotubes have been transferred to the substrate, parallel ribbons are then made from the continuous nanotube array by using an etching process similar to the one used in [17]. We set the distance between ribbons as 96 nm to allow spacing for contacts, and assume this resolution can be achieved in the target photolithographic process. By using etching to define the ribbons, we have the added advantage of making the ribbons misalignment immune. This is because any nanotubes crossing the border of a ribbon will be removed during the etching process. Figure 9 demonstrates this concept, where (a) shows a misaligned tube, (b) shows the etched area, and (c) shows the resulting CNT ribbons.

The next major step in fabrication is to disable the metallic nanotubes inside the decoder region. Since metallic CNTs act as a short between source and drain, they need to be removed to create CNFET transistors with desirable on/off current ratios. Electrical burning [28] is an effective method to selectively disable the metallic CNTs. In this technique, a large voltage is applied across the array which heats the conducting metallic nanotubes to a breakdown temperature of $\sim$600°C and causes irreversible oxidization. Because this is done when the CNTs are still exposed to air, a minimum power dissipation of 0.05 mW is needed to achieve breakdown [28].

Since metallic tubes are used for NRAM operation, the burning must only be done in the decoder region. One way to remove the metallic CNTs from the decoder but keep them for the NRAM devices is shown in Fig. 10. In this figure, (a) shows vertical ribbons of mixed metallic and semiconducting CNTs held in place by horizontal metal electrodes. The middle and bottom electrodes are used to hold the ribbons in place during NRAM operation. In (b), a thermal breakdown voltage, V_{BURN}, is applied between the top and middle electrodes. This burns away the metallic tubes in the decoder region but leaves them in the NRAM region. Because the NRAM memory devices need to be individually addressable, the electrode is segmented to provide electrical isolation (c).

After the CNT ribbons are defined and processed, the gate and source/drain formation is similar to a regular CMOS process. Based on these techniques and the existing

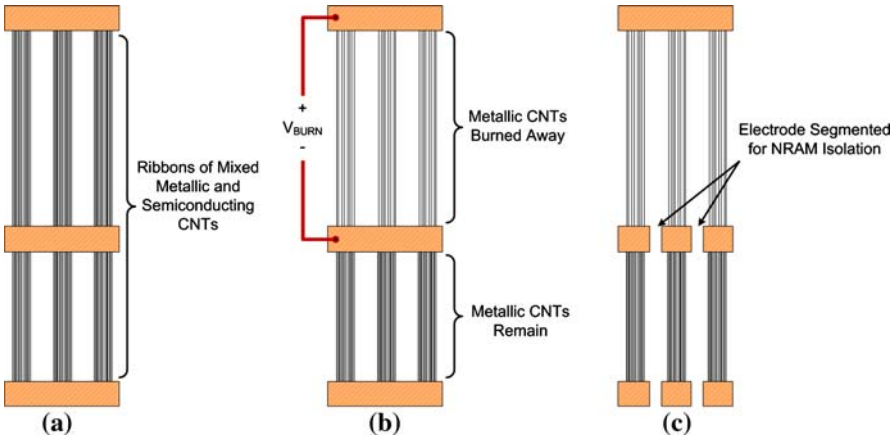


Fig. 10 Metallic CNT removal

CNT fabrication work [16–18], we believe the proposed nanotube-based LUT design is implementable.

7 Circuit Characterization

7.1 CNFET and CNT-Based LUT Variation

As mentioned in Sect. 4, CNFETs have many properties that make them attractive for use in future electrical circuits. Ideally, the channel region of these CNFETs would consist of identical, well-aligned semiconducting CNTs with the same source/drain doping levels. However, it is difficult to synthesize nanotubes with exactly controlled chirality using known fabrication techniques. HiPco synthesis techniques yield around $50\% \pm 10\%$ metallic CNTs [19]. This means the number of semiconducting CNTs per device is stochastic, causing drive current variations even after the metallic CNTs are burned away. Meanwhile, CNFETs are also susceptible to variations in diameter and source/drain region doping [21].

In a traditional MOSFET, Gaussian distributions are often assumed when modeling variation sources such as channel length and gate width. These models are then used in the delay or power characterization of the MOSFET. We can use a similar approach to characterize CNFETs. To quantify the effects of CNFET variations, we perform a

Table 2 Sources of CNFET variation

Parameter	Mean	Variation (3σ)
CNTs per channel case 1	8	± 3
CNTs per channel case 2	6	± 2
CNT diameter	1.5 nm	± 0.3 nm
Doping level	0.6 ev	± 0.03 ev

Monte Carlo simulation of CNFET devices with 2,000 runs. We consider the sources of variation listed in Table 2, evaluating two scenarios for the number of CNTs in a channel: 8 ± 3 and 6 ± 2, both normally distributed. The diameter range, doping level range, and CNFET model are suggested in [21].

The results of the simulation show that the delay distribution of a CNFET device under these variations fits the Gaussian distribution. Figure 11 illustrates this distribution for a CNFET with 8 ± 3 semiconducting nanotubes in its channel.

Using the CNFET model, we can also evaluate the performance of our CNT-based LUT design. The LUT decoder consists of multiple stages of *p*-type CNFETs, simulated under the variations mentioned in Table 2. The contact resistance between an electrode and a single nanotube is assumed to be 20 KΩ based on [22]. In a ribbon, multiple CNTs are operating in parallel, so we consider the ribbon contact resistance to be inversely proportional to the number of semiconducting nanotubes. For NRAM devices, we assume a contact resistance between a bending nanotube and the base electrode of 20 KΩ, based on the measurements in [12]. Since these CNTs also operate in parallel, we treat the total ribbon NRAM contact resistance as inversely proportional to the number of metallic nanotubes in the ribbon. The resulting LUT delay distribution generated by Monte Carlo simulation in HSPICE is shown in Fig. 12.

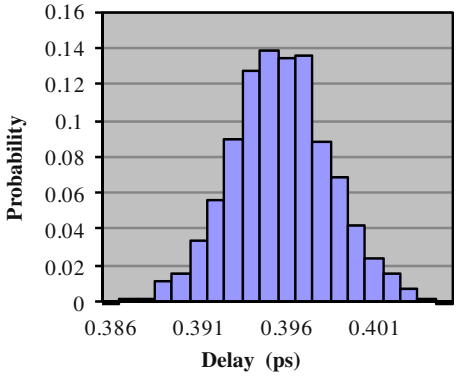


Fig. 11 Delay distribution of a CNFET under process variation

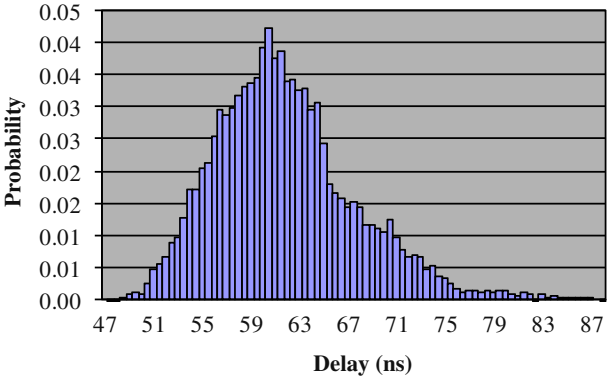


Fig. 12 CNT-based LUT delay considering variation

The average delay is 60.94 ps, which is 41% faster than a traditional 32 nm CMOS LUT, which has a delay of 103.8 ps. Unlike the CMOS LUT, the delay of the nanotube-based LUT has a distribution similar to log-normal.

7.2 Crossbar Characterization

As described in Sect. 5, the routing in FPCNA is implemented using crossbars. We capture the delay and variation of these crossbars using HSPICE. CNT bundle interconnect is assumed to be 32 nm in width, with an aspect ratio of 2. We set the dielectric constant of the insulation material around the crossbar at 2.5, and derive a unit resistance of 10.742 Ω/μm and capacitance of 359.078 aF/μm for the carbon nanotube bundles. The interconnects are evaluated for 10% geometrical variation of wire width, wire thickness, and spacing according to [29]. CNT bundle interconnect variation also considers a 40–60% range on percentage of metallic nanotubes inside a bundle. The solid-electrolyte switches between interconnect layers are considered with a 100 Ω ON resistance [25] and 10% variation to capture via contact resistance.

7.3 Timing Block Evaluation

To support the evaluation CAD flow, various circuit models are needed to capture characteristics of the FPCNA architecture. In the architecture specification file of VPR, we specify the delay values for certain combinational circuit paths to enable accurate timing analysis. For example, in Fig. 5, there are paths A $\rightarrow$ B, B $\rightarrow$ C, and D $\rightarrow$ C, etc. In FPCNA, these paths contain buffers, metal wires, and solid-electrolyte switches, making them susceptible to process variations.

We compute the delay and variation of these paths in FPCNA by performing a Monte Carlo simulation of 1,000 runs, varying the CNFET parameters and CNT contact resistance for each run. Figure 13 illustrates the resulting delay distributions of wire track to CLB inpin connections (A $\rightarrow$ B) and subblock opin to subblock inpin connections (D $\rightarrow$ C).

Based on these results, we observe that the timing blocks follow a normal-like distribution. We therefore calculate the mean (μ) and variation (σ) of each delay path, as shown in Table 3. An equivalent design in CMOS is measured as a baseline for

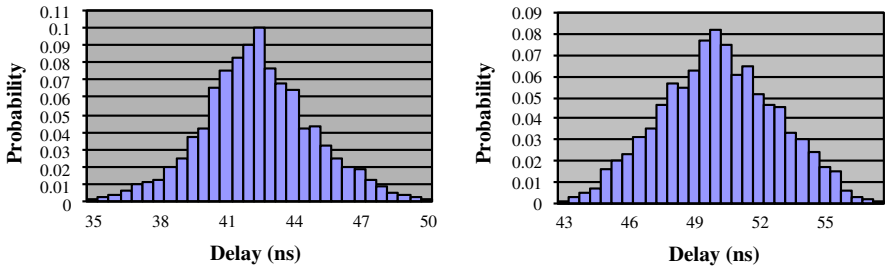


Fig. 13 Delay distribution of wire track to CLB inpin (*left*) and sub-block opin to sub-block inpin (*right*)

Table 3 Delay comparison between baseline CMOS and FPCNA

Paths	CMOS-baseline		FPCNA	
	μ(ps)	σ(ps)	μ(ps)	σ(ps)
A → B	141.66	7.13	42.24	2.48
B → C	107.59	5.37	30.45	2.21
D → C	107.59	5.37	49.96	2.92
D → Out	28.48	1.22	29.91	2.28

comparison, assuming 12% channel width variation, 8% gate dielectric thickness variation, and 10% doping variation (values from [30] for 32 nm CMOS), and these values are also shown in the table.

8 CAD Flow

In the previous sections, we show that both deep sub-micron CMOS and nano-scale devices are susceptible to variation. In traditional static timing analysis, it is assumed that all circuit elements have deterministic delay. This approach cannot correctly capture the variability of the fabrication process. The worst-case analysis commonly used by industrial designs satisfies yield but is overly pessimistic. On the other hand, the nominal case produces low yield due to variation-based timing failures. To maximize yield without sacrificing performance, it is necessary for CAD tools to consider the statistical information of circuit elements during timing analysis.

In this work, we use the timing-driven, variation-aware CAD flow shown in Fig. 14. Each benchmark circuit goes through technology independent logic optimization using SIS [31] and is technology-mapped to 4-LUTs using DAOmap [32]. The mapped netlist then feeds into T-VPACK and VPR [1], which perform timing-driven packing

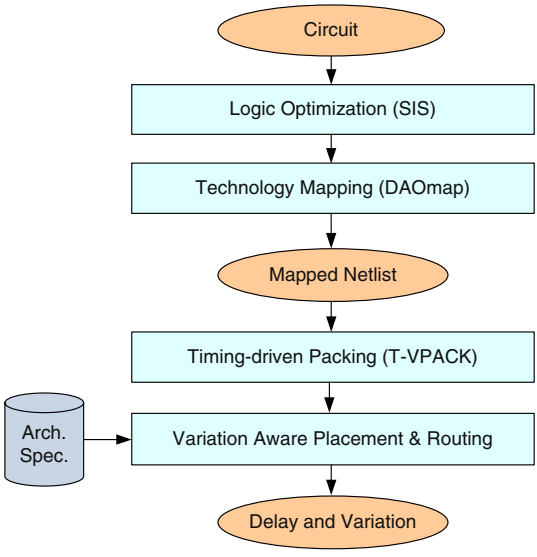


Fig. 14 FPCNA evaluation flow

(i.e., clustering LUTs into the CLBs), placement, and routing. To take variation into consideration, we enhance the VPR tool [1] to make it variation aware.

Existing works have shown that statistical optimization techniques are useful during the physical design stage. Variation aware placement is implemented in [33] and variation aware routing is developed in [34]. Based on the ideas presented in these works, we implement a complete variation aware physical design flow. In this holistic solution, the placer calls the variation aware router to generate delay estimates for its timing cost calculations.

From the Monte Carlo simulation results in Sect. 7, we observe that the CNT-based LUT delay follows a non-Gaussian distribution. Reference [22] also reports a non-Gaussian distribution for CNT bundle interconnect. However, all of the existing CAD work targeting CMOS assumes normally distributed random variables [33–35]. The Gaussian-based SSTA algorithms that these works use to evaluate CMOS are not suitable for modeling the non-normal variables of our molecular-based architecture. Therefore, in this work we develop a new statistical timing analyzer that can handle an arbitrary distribution, based on discretization techniques adapted from [36,37].

One such technique is the probabilistic event propagation developed in [36], in which discretized random variables of cell delays are used for timing analysis. As illustrated in Fig. 15, a non-Gaussian probability density function can be represented as a set of delay-probability pairs which contain the time t and the probability a signal will arrive at time t. In [37], ADD and MIN operations are developed for propagating multiple event groups. We use these operations, and additionally define a MAX operation for our statistical timing analyzer. In Fig. 16, we illustrate how the discretized MAX operation is performed using an example point.

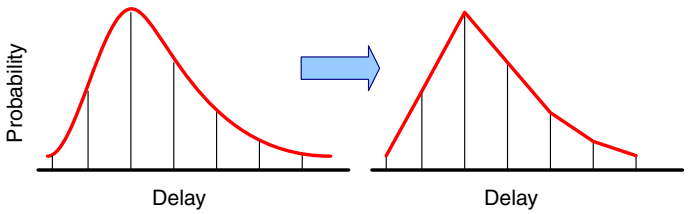
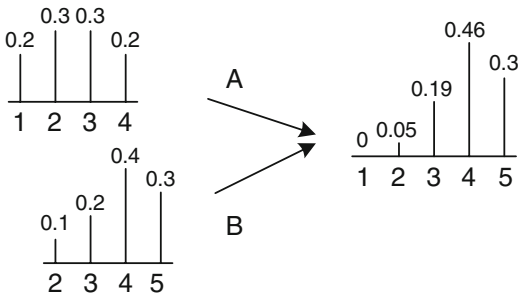


Fig. 15 Discretization process of a log-normal probability density function

Fig. 16 The discretized MAX operation



During the MAX operation, all possible timing points at the output are evaluated, computing their probability based on the input sets of delay-probability pairs. For each timing point t, we define the probability that both inputs arrive as $P(t)$. $P(t)$ can be derived using conditional probability as the sum of:

1. The probability that both A and B arrive at t
2. The probability that A arrives at t and B arrived before t
3. The probability that B arrives at t and A arrived before t

Take $t = 1$ for example. The earliest arrive time for B is 2, so $P(1) = 0$. The probability that both A and B arrive at $t = 2$ is $0.1 \times 0.3 = 0.03$. The probability that A arrives at $t = 2$, and B arrived before $t = 2$ is $0.3 \times 0 = 0$. The probability that B arrives at $t = 2$, and A arrived before $t = 2$ is $0.2 \times 0.1 = 0.02$. Therefore, $P(2)$ sums up to 0.05. The remaining points are processed in a similar fashion. The accuracy of this technique is dependent on the number of points used for piecewise linear approximation. It is shown in [36] that seven points are sufficient to obtain an accuracy of less than 1% error compared to Monte Carlo. Therefore, we use 7-point sampling throughout our discretized SSTA.

Figure 17 shows the pseudo-code of our variation-aware router. The routing is iterative. During the first iteration, the criticality of each pin in every net is set to 1 (highest criticality) to minimize the delay of each pin. For the CMOS architecture, the Gaussian delay mean (μ) and standard deviation (σ) of each path are computed during the routing of each net. For FPCNA, the discretized delay distribution of each path is computed. If congestion exists, more routing iterations are performed until all of the overused routing resources are resolved. At the end of each routing iteration, criticality and congestion information are updated before the next iteration starts.

To consider variation, new formulas to capture the criticality of sink j of net i are derived. For the CMOS architecture with a Gaussian distribution, we express the arrival time of pin j in net i as $arr(i, j) = (t_a, \sigma_a)$ and the required time as

```
Set all nets i and sinks j, Crit(i,j) =1.0;
while (overused routing resources exist) do
    for (each net, i) do
        Rip-up routing tree of net i ;
        for (each sink j of net i in decreasing Crit (i,j) order) do
            Find the least cost route of sink j ;
            for (all nodes in the path from i to j ) do
                Update congestion;
            end
            Update delay and standard deviation of the route ;
            (Update discretized delay for FPCNA ;)
        end
    end
    Update historic congestion ;
    Compute mean and standard deviation of delay for each net (i);
    (Compute discretized delay for each net (i) in FPCNA;)
    Update Crit(i,j);
end
```

Fig. 17 Pseudo-code of the modified VPR router

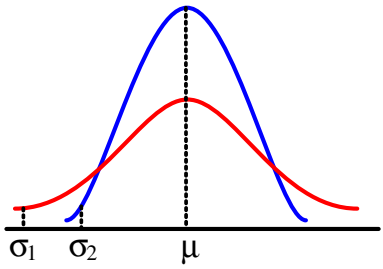


Fig. 18 Criticality estimation

$req(i, j) = (t_r, \sigma_r)$. The mean and standard deviation of slack $slack(i, j) = (t_s, \sigma_s)$ can be derived as:

$$t_s = t_r - t_a \quad \sigma_s = \sqrt{|\sigma_a^2 + \sigma_r^2|}$$

The criticality of pin j in net i can then be computed by taking both slack and slack variation into consideration:

$$Crit(i, j) = 1 - \frac{slack(i, j) - 3\sigma_s(i, j)}{t_{crit} + 3\sigma_{crit}}$$

We modify the original VPR cost function in this way so when two slacks have a similar mean but different variations, the $slack(i, j) - 3\sigma_s(i, j)$term assigns a larger criticality to the path with the greater variation to weight it more heavily in the next routing iteration. This is illustrated in Fig. 18, where the distribution with slack variation σ_1 will be assigned a higher criticality than the distribution with slack variation σ_2, even though they have the same mean. This cost function also considers the critical path variation with the $t_{crit} + 3\sigma_{crit}$ term.

In the discretized routing, the expected values of the slack and critical path discretized points are computed and used in the following criticality function:

$$Disc\_Crit(i, j) = 1 - \frac{E[disc\_slack(i, j)]}{E[disc\_t_{crit}]}$$

After each routing iteration, SSTA is executed by traversing the updated timing graph to calculate the new slack and critical path delay. Our variation-aware placer also uses these criticality functions to calculate the timing cost of each move during simulated annealing. In the placement cost function, the critically value is raised by the exponent β. We determined the optimal value of β to be 6 for our design. This differs from the original VPR method of incrementing β from 1 to 8, and from [33] where a β value of 0.3 was used. As in [33], we calculate the variation during the delta array creation, and store these pre-calculated values in the delta arrays. The main difference is that we use a variation aware router to generate the delay and store sets of discretized delay-probability points for each delay value in addition to the mean and variation.

9 Experimental Results

9.1 Experimental Setup

Because our CAD flow is flexible, we can experiment with various architecture parameters and determine their impact. To evaluate FPCNA, we use a fixed LUT input size $K = 4$, and explore logic cluster sizes of $N = 4$, 10, and 20. We also explore the difference between using an average of 16 CNTs per ribbon and 12 CNTs per ribbon, to see the effect on area. We set the number of CLB inputs based on the cluster size so that it is equal to $2N + 2$. We keep Fc at 0.5, a typical value which connects the CLB input to half of the routing tracks in the channel.

It is shown in [1] that a mixture of different length interconnects can provide improved performance. We evaluate two popular wire length mixtures in our experiments: an equal mixture of length-4 and length-8 wire segments (wires crossing either four CLBs or eight CLBs), and a mix of 30% length-1, 40% length-2, and 30% length-4 wire segments.

For each configuration of the above parameters, we perform a binary search to determine the routing channel width needed to successfully route the largest benchmark, and then use that width to evaluate all of the benchmarks.

9.2 Area Reduction

Due to the high density CNT-based logic and solid-electrolyte switch-based routing, the footprint of FPCNA is significantly smaller than the equivalent CMOS FPGA. To calculate the area, we use the architecture parameters defined above, and assume a transistor feature size of 32 nm (2λ) for both CNT and CMOS-based transistors. The area of our CNT-based lookup table (Fig. 4), is determined by the size and spacing of our CNT-ribbons and addressing lines. Since we assume an average CNT pitch of 4 nm, the CNT ribbons are 64 nm wide for the 16-tube per ribbon experiments and 48 nm wide for the 12-tube per ribbon experiments. To accommodate gate layer to metal-1 layer vias, the nanotube ribbons are spaced 96 nm (6λ) apart. LUT addressing gate metal is 32 nm (2λ), with a spacing of 80 nm (5λ) between adjacent lines. Gate to metal-1 via size is assumed to be 64 nm (4λ) square. The nanotube memory, NRAM, offers a much smaller area than an SRAM cell. We assume an NRAM trench that is 180 nm in width and 18 nm in height. These are conservative dimensions based on proven fabrication in [27]. Trench to electrode spacing is set to 90 nm for each side. All of the LUT electrodes are assumed to be 64 nm wide. The area of the 32 nm CMOS components in our BLE are calculated using a technique from [38]. Total BLE logic area is the sum of both the CMOS logic area and CNT-based LUT area.

Local CLB interconnect crossbars are assumed to have a line thickness of 64 nm (4λ), and spacing of 64 nm (4λ). The routing crossbars are created on the metal layers above the CLB logic, so they do not add to the overall CLB area (assuming the crossbar area is smaller than the logic area, which was true in all of our experiments). Since the routing path is controlled by non-volatile solid-electrolyte switches, the SRAM cells used in the baseline CMOS FPGA can be eliminated in FPCNA. By replacing

the MUX-based routing with crossbars and switching to CNT-based LUTs, a large overall area reduction is seen. For an architecture with a cluster size of 10 and wire segmentation of length 4 and 8, we estimate the footprint of a baseline CMOS FPGA tile to be 34,623T. Using a minimum width transistor area of T = 0.0451 μm^2 for a 32 nm transistor gives us a tile area of 1561.5 μm^2. When we calculate the area for an equivalent FPCNA tile under the area assumptions above, only 307.99 μm^2 is used. These calculations show that FPCNA can achieve an area reduction of roughly 5× over CMOS.

The area breakdowns for a single LUT and an architecture tile are shown in Table 4. In this table the CB area is the sum of both connection blocks. Table 5 shows the area breakdowns of various FPCNA architectures. The first row describes global routing with 30% length-1, 40% length-2 and 30% length-4 wire segments. The second row is for 50% length-4 and 50% length-8 interconnects. As seen in the table, routing occupies the majority of FPCNA's overall area. Due to the size of the SB Area, wire segmentation has a significant impact on the overall area. Shorter wire segments have

Table 4 Area reduction of FPCNA

	CMOS FPGA	FPCNA	Reduction
Single LUT area	10.88 μm^2	2.15 μm^2	5.06×
LUT addressing area	5.68 μm^2	1.52 μm^2	3.73×
LUT memory area	5.20 μm^2	0.63 μm^2	8.24×
Tile area	1561.5 μm^2	307.99 μm^2	5.07×
CLB area	665.2 μm^2	63.290 μm^2	10.51×
CB area	337.7 μm^2	82.5 μm^2	4.1×
SB area	558.6 μm^2	162.2 μm^2	3.4×

Table 5 Area of various FPCNA architectures

	Cluster 4		Cluster 10		Cluster 20	
	16 tubes	12 tubes	16 tubes	12 tubes	16 tubes	12 tubes
1–2–4 wire segments						
CLB area (μm^2)	25.316	23.784	63.29	59.46	126.58	118.921
CB area (μm^2)	23.46	23.46	75.01	75.01	203.438	203.438
SB area (μm^2)	205.06	205.06	444.49	444.49	829.437	829.437
Total tile area (μm^2)	253.84	252.31	582.79	578.96	1159.46	1151.8
Tile edge length (μm)	15.932	15.884	24.141	24.062	34.051	33.938
4–8 wire segments						
CLB area (μm^2)	25.316	23.784	63.29	59.46	126.58	118.921
CB area (μm^2)	27.07	27.07	82.5	82.5	435.94	435.94
SB area (μm^2)	83.94	83.94	162.2	162.2	1087.86	1087.86
Total tile area (μm^2)	136.33	134.8	307.99	304.16	1650.38	1642.72
Tile edge length (μm)	11.676	11.61	17.55	17.44	40.625	40.531

better flexibility during routing, but require a larger number of switch points in each SB, which greatly increases the tile size.

9.3 Performance Gain

In this section, we evaluate the experimental CAD flow presented in Sect. 8, quantifying the overall performance improvement of FPCNA from the baseline CMOS counterpart. When considering variation, performance evaluation becomes complicated. The critical path delay can no longer serve as the absolute measure of performance. Due to variations, near-critical paths may actually be statistically critical. This is illustrated by PO3 in Fig. 19. In addition, setting a clock period based only on the most statistically critical path is not appropriate. Consider the case in Fig. 19, where the target clock period is set to a 95% guard-band of PO3. This means that for 95% of chips made, PO3 will not generate a timing failure. However, at this clock period, the other POs may also fail due to variation, making the overall yield less than 95%. Because of this phenomenon, it is necessary to consider the statistical delay of every path in yield analysis. We express the performance yield as a delay-probability pair (t, p), so that by setting the clock period t, we can evaluate the system yield p. This allows us to compare the performance of the statistical information generated by our experiments.

We calculate performance yield for both Gaussian and non-Gaussian distributions using the flow in Fig. 20. After selecting a target clock period T_c, the yield of each of the POs is computed. For a Gaussian distribution, the yield is calculated by computing the inverse cumulative distribution function (CDF) of the delay random variable. In a non-Gaussian delay distribution, the delay is represented by a group of points, so the yield is computed by converting the piecewise linear PDF into a piecewise linear CDF (Fig. 21). The overall system yield is determined by multiplying all of the path yields. If the system yield is not satisfied, we increase the T_c and repeat the process until the desired yield is obtained. We then report the final clock period, which guarantees the targeted yield.

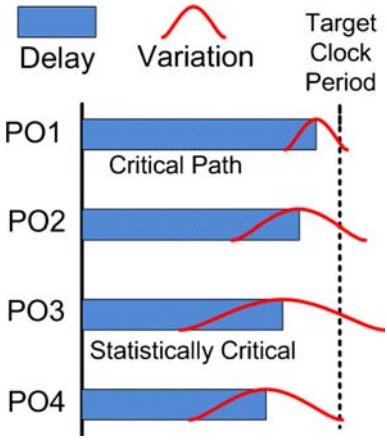


Fig. 19 The effect of variation on critical path and yield

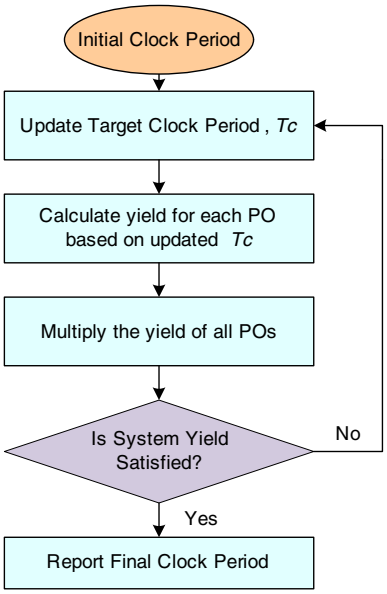


Fig. 20 Performance yield estimation

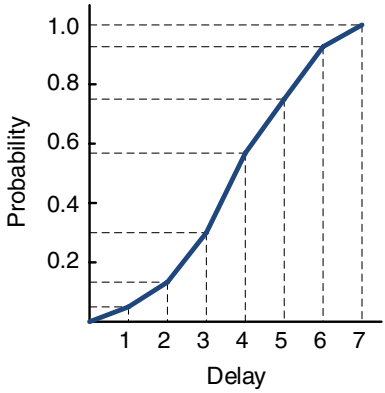


Fig. 21 Piecewise linear CDF in discretized timing analysis

Using our variation-aware CAD flow, we evaluate the achievable clock period of 20 MCNC benchmarks and report the results in Table 6. For a rough comparison to a deterministic solution, we evaluate the CMOS design using VPR [1], with a worst case delay guard-band of 3σ added to each component in VPR's architecture file. This equates to a component yield of roughly 99%.

The table shows the deterministic CMOS results, variation aware CMOS results, and two versions of variation aware FPCNA results; one with 16 nanotubes per CNT ribbon, and the other with 12 nanotubes per ribbon. For each variation aware flow, we calculate the clock period for performance yield at both 95% and 99%. We also generate the performance gain of the FPCNA architectures over the baseline CMOS. In this table, both CMOS and FPCNA are configured with a cluster size of 10 and interconnect wire segmentation of 50% length-4 and 50% length-8. Average delays

Table 6 System clock period needed to achieve target performance yield

MCNC benchmark	CMOS with deterministic CAD flow	CMOS with variation-aware CAD flow		FPCNA with variation-aware CAD flow (16 CNTs per ribbon)			FPCNA with variation-aware CAD flow (12 CNTs per ribbon)		
	99% Component yield (ns)	95% performance yield (ns)	99% performance yield (ns)	95% performance yield (ns)	99% performance yield (ns)	Perf. gain over CMOS at 95% yield	95% performance yield (ns)	99% performance yield (ns)	Perf. gain over CMOS at 95% yield
alu4	9.262	7.338	7.469	2.559	2.698	2.87×	2.678	2.812	2.74×
apex2	10.51	8.444	8.587	3.235	3.313	2.61×	3.263	3.307	2.59×
apex4	9.796	7.602	7.726	3.666	3.706	2.07×	3.460	3.756	2.2×
bigkey	4.580	4.336	4.416	1.480	1.502	2.93×	1.474	1.495	2.94×
clma	20.55	18.98	19.18	5.666	5.720	3.35×	6.790	6.818	2.8×
des	8.900	8.853	8.994	2.884	2.921	3.07×	3.027	3.058	2.92×
diffeq	7.241	6.351	6.448	2.736	2.978	2.32×	2.827	3.070	2.25×
dsip	4.790	4.856	4.954	1.643	1.647	2.96×	1.668	1.682	2.91×
elliptic	14.87	11.26	11.39	3.342	3.483	3.37×	3.810	3.967	2.96×
ex1010	16.39	12.99	13.15	5.215	5.363	2.49×	4.801	5.046	2.71×
ex5p	9.885	8.693	8.847	3.760	3.812	2.31×	4.500	4.554	1.93×
frisc	16.11	14.99	15.15	3.908	4.367	3.84×	5.114	5.316	2.93×
misex3	8.284	6.543	6.649	3.092	3.284	2.12×	2.709	2.899	2.42×
pdc	17.25	16.13	16.32	4.637	4.863	3.48×	4.770	4.957	3.38×
s298	15.14	14.10	14.25	3.822	3.857	3.69×	4.029	4.134	3.5×
s38417	10.97	10.62	10.74	4.314	4.370	2.46×	3.463	3.590	3.07×
s38584.1	8.456	7.024	7.140	2.816	2.894	2.49×	2.884	3.019	2.44×
seq	10.78	7.859	7.987	3.203	3.344	2.45×	3.634	3.757	2.16×
spla	15.20	12.04	12.20	4.643	4.730	2.59×	4.826	4.864	2.49×
tseng	8.851	6.700	6.804	2.692	2.785	2.49×	2.835	2.917	2.36×
Average	10.59	9.070	9.203	3.293	3.404	2.75×	3.417	3.536	2.65×

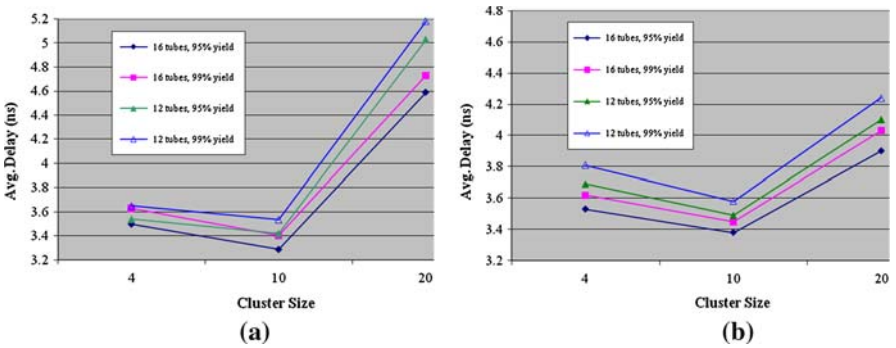


Fig. 22 Average delay for different architecture parameters at 95% and 99% yield. **a** 4–8 Wire segmentation, **b** 1–2–4 Wire segmentation

are calculated using the geometric mean. At a 95% performance yield, the FPCNA designs have an average gain of 2.75× and 2.65× over the CMOS counterpart, for 16 and 12 ribbons, respectively. This significant improvement in performance is achieved by the synergistic combination of CNT logic, CNT bundle interconnects, and routing crossbar design in FPCNA.

As shown in Table 5, reducing number of tubes inside nanotube ribbon can reduce tile footprint, which will reduce the length of global interconnect and should therefore enhance performance. However, as we see in Table 6, the overall performance is actually degraded. This is because with fewer tubes, each CNFETs has less driving capability, which increases the LUT delay enough to overcome any global interconnect savings. To develop a better understanding of how the FPCNA architecture affects performance, we evaluated different architecture combinations of wire segmentation, cluster size, and nanotube ribbon size for the 20 benchmarks. The average results, again using the geometric mean, are plotted in Fig. 22.

As seen in Fig. 22a, for small and medium cluster sizes (4 and 10), long interconnects are preferable, because they can make connections to CLBs which are far away. For the larger cluster size of 20, shorter wire segments are preferred (Fig. 22b). Note that in Fig. 22a, the performance degrades rapidly at cluster size 20 because there are an increased number of connections between neighboring CLBs, and a limited number of short wire segments. The experiments also show that medium sized clusters with longer interconnects have the best performance for FPCNA. This is because a medium sized cluster will take advantage of both carbon nanotube bundle interconnect and local routing.

10 Conclusion and Future Work

In this paper, we presented a novel FPGA architecture called FPCNA. We designed CNT-based and nanoswitch-based building blocks such as the CNT-based LUT, and characterized them under nano-specific process variations such as CNT diameter and CNT doping level. We also addressed some issues in CNT device fabrication. An effective variation-aware CAD flow was developed which handles arbitrary delay

distributions using variation-aware placement and routing. Experimental results show that FPCNA offers a 5.07× footprint reduction and a 2.75× performance gain (targeting a 95% yield) compared to a baseline CMOS FPGA at the same technology node. This clearly demonstrates potential for using nanomaterials to build the next-generation FPGA circuits.

While we only studied random variation in this work, a more accurate variation model could be made by considering correlated variation in future works. Another consideration is that the universal memory market is still in its infancy. In upcoming years, alternatives to solid-electrolyte nanoswitches and NRAM might become widely adopted and cost effective, and therefore should also be considered.

Acknowledgements This work is partially supported by NSF Career Award CCF 07-46608, NSF grant CCF 07-02501, and a gift grant from Altera Corporation. We also appreciate the helpful discussions with Prof. John Rogers of the University of Illinois at Urbana Champaign and Prof. Subhasish Mitra of Stanford University.

References

1. Betz, V., Rose, J., Marquardt, A.: Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, New York, NY (1999)
2. Copen Goldstein, S., Budiu, M.: NanoFabrics: spatial computing using molecular electronics. In: Proceedings of the 28th Annual International Symposium on Computer Architecture, pp. 178–189 (2001). doi:10.1109/ISCA.2001.937446
3. DeHon, A.: Nanowire-based programmable architectures. ACM JETC **1**(2), 109–162 (2005). doi:10.1145/1084748.1084750
4. Snider, G., Kuekes, P., Williams, R.S.: CMOS-like logic in defective nanoscale crossbars. Nanotechnology **15**, 881–891 (2004)
5. Gayasen, A., Vijaykrishnan, N., Irwin, M.J.: Exploring technology alternatives for nano-scale FPGA interconnects. In: DAC '05: Proceedings of the 42nd Annual Conference on Design Automation, Anaheim, CA, pp. 921–926. ACM, New York (2005). http://doi.acm.org/10.1145/1065579.1065820
6. Rad, R.M.P., Tehranipoor, M.: A new hybrid FPGA with nanoscale clusters and CMOS routing. In: DAC '06: Proceedings of the 43rd Annual Conference on Design Automation, San Francisco, CA, pp. 727–730. ACM, New York (2006). http://doi.acm.org/10.1145/1146909.1147094
7. Strukov, D.B., Likharev, K.K.: CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. Nanotechnology **16**, 888–900 (2005)
8. Snider, G., Williams, S.: Nano/CMOS architecture using a field-programmable nanowire interconnect. Nanotechnology **18**, 035204–035215 (2007)
9. Dong, C., Chen, D., Haruehanroengra, S., Wang, W.: 3-D nFPGA: a reconfigurable architecture for 3-D CMOS/nanomaterial hybrid digital circuits. IEEE Trans. Circuits Syst. I **54**(11), 2489–2501 (2007)
10. Rueckes, T., Kim, K., Joselevich, E., Tseng, G.Y., Cheung, C., Lieber, C.M.: Carbon nanotube-based nonvolatile random access memory for molecular computing. Science **289**(5476), 94–97 (2000). doi:10.1126/science.289.5476.94
11. Ward, J.W., Meinhold, M., Segal, B.M., Berg, J., Sen, R., Sivarajan, R., Brock, D.K., Rueckes, T.: A nonvolatile nanoelectromechanical memory element utilizing a fabric of carbon nanotubes. In: Non-Volatile Memory Technology Symposium, Nov. 2004, pp. 34–38 (2004)
12. Smith, R.F., Rueckes, T., Konsek, S., Ward, J.W., Brock, D.K., Segal, B.M.: Carbon nanotube based memory development and testing. In: IEEE Aerospace Conference, March 2007, pp. 1–5 (2007). doi:10.1109/AERO.2007.353104
13. Zhang, W., Jha, N.K., Shang, L.: NATURE: a hybrid nanotube/CMOS dynamically reconfigurable architecture. In: 43rd ACM/IEEE Design Automation Conference, pp. 711–716 (2006). doi:10.1109/DAC.2006.229333

14. Zhou, Y., Thekkel, S., Bhunia, S.: Low power FPGA design using hybrid CMOS-NEMS approach. In: ISLPED '07: Proceedings of the 2007 International Symposium on Low Power Electronics and Design, Portland, OR, pp. 14–19. ACM, New York (2007). http://doi.acm.org/10.1145/1283780.1283785
15. McEuen, P., Fuhrer, M., Park, H.: Single-Walled Carbon Nanotube Electronics. IEEE Trans. Nanotechnol. **1**(1), 78–85 (2002)
16. Kang, S.J., et al.: High-performance electronics using dense, perfectly aligned arrays of single-walled carbon nanotubes. Nat. Nanotechnol. **2**(4), 230–236 (2007)
17. Patil, N., Lin, A., Myers, E.R., Wong, H.-S.P., Mitra, S.: Integrated wafer-scale growth and transfer of directional carbon nanotubes and misaligned-carbon-nanotube-immune logic structures. In: Symposium on VLSI Technology, June 2008, pp. 205–206 (2008). doi:10.1109/VLSIT.2008.4588619
18. Zhou, W., Rutherglen, C., Burke, P.: Wafer scale synthesis of dense aligned arrays of single-walled carbon nanotubes. Nano Res. **1**, 158–165 (2008). doi:10.1007/s12274-008-8012-9
19. Li, Y., et al.: Preferential growth of semiconducting single-walled carbon nanotubes by a plasma enhanced CVD method. Nano Lett. **4**, 317 (2004). doi:10.1021/nl035097c
20. Liu, X., Han, S., Zhou, C.: Novel nanotube-on-insulator (NOI) approach toward single-walled carbon nanotube devices. Nano Lett. **6**(1), 34–39 (2006). doi:10.1021/nl0518369
21. Deng, J., Patil, N., Ryu, K., Badmaev, A., Zhou, C., Mitra, S., Wong, H.-S.P.: Carbon nanotube transistor circuits: circuit-level performance benchmarking and design options for living with imperfections. In: IEEE International Solid-State Circuits Conference (ISSCC 2007), Digest of Technical Papers, Feb. 2007, pp. 70–588 (2007). doi:10.1109/ISSCC.2007.373592
22. Massoud, Y., Nieuwoudt, A.: Modeling and design challenges and solutions for carbon nanotube-based interconnect in future high performance integrated circuits. ACM J. Emerg. Technol. Comput. Syst. **2**, 155–196 (2006). doi:10.1145/1167943.1167944
23. Wei, B.Q., Vajtai, R., Ajayan, P.M.: Reliability and current carrying capacity of carbon nanotubes. Appl. Phys. Lett. **79**(8), 1172–1174 (2001). doi:10.1063/1.1396632
24. Srivastava, N., Banerjee, K.: Performance analysis of carbon nanotube interconnects for VLSI applications. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD-2005), Nov. 2005, pp. 383–390 (2005). doi:10.1109/ICCAD.2005.1560098
25. Kaeriyama, S., et al.: A nonvolatile programmable solid-electrolyte nanometer switch. IEEE J. Solid-State Circuits **40**(1), 168–176 (2005). doi:10.1109/JSSC.2004.837244
26. Ahmed, E., Rose, J.: The effect of LUT and cluster size on deep-submicron FPGA performance and density. IEEE Trans. VLSI **12**(3), 288–298 (2004). doi:10.1109/TVLSI.2004.824300
27. Kang, S.J., Kocabas, C., Kim, H.S., Cao, Q., Meitl, M.A., Khang, D.Y., Rogers, J.A.: Printed multilayer superstructures of aligned single-walled carbon nanotubes for electronic applications. Nano Lett. **7**(11), 3343–3348 (2007). doi:10.1021/nl071596s
28. Pop, E.: The role of electrical and thermal contact resistance for Joule breakdown of single-wall carbon nanotube. Nanotechnology **19**, 295202–295207 (2008)
29. Boning, D.S., Nassif, S.: Models of process variations in device and interconnect. In: Design of High Performance Microprocessor Circuits. IEEE Press (2000)
30. International Technology Roadmap for Semiconductors: http://www.itrs.net/
31. Sentovich, E.M., et al.: SIS: A System for Sequential Circuit Synthesis. Dept. of Electrical Engineering and Computer Science. University of California, Berkeley, CA (1992)
32. Chen, D., Cong, J.: DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs. In: International Conference on Computer-Aided Design, pp. 752–759. IEEE Computer Society, Los Alamitos (2004). http://doi.ieeecomputersociety.org/10.1109/ICCAD.2004.1382677
33. Lin, Y., Hutton, M., He, L.: Placement and timing for FPGAs considering variations. In: FPL '06: International Conference on Field Programmable Logic and Applications, Aug. 2006, pp. 1–7 (2006). doi:10.1109/FPL.2006.311192
34. Sivaswamy, S., Bazargan, K.: Variation-aware routing for FPGAs. In: FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays, Monterey, CA, pp. 71–79. ACM, New York (2007). http://doi.acm.org/10.1145/1216919.1216930
35. Visweswariah, C., Ravindran, K., Kalafala, K., Walker, S.G., Narayan, S.: First-order incremental block-based statistical timing analysis. In: DAC '04: Proceedings of the 41st Annual Conference on Design Automation, San Diego, CA, pp. 331–336. ACM, New York (2004). http://doi.acm.org/10.1145/996566.996663

36. Devgan, A., Kashyap, C.: Block-based static timing analysis with uncertainty. In: International Conference on Computer Aided Design (ICCAD-2003), Nov. 2003, pp. 607–614 (2003). doi:10.1109/ICCAD.2003.1257873
37. Liou, J.-J., Cheng, K.-T., Kundu, S., Krstic, A.: Fast statistical timing analysis by probabilistic event propagation. In: DAC '01: Proceedings of the 38th Conference on Design Automation, Las Vegas, NV, pp. 661–666. ACM, New York (2001). http://doi.acm.org/10.1145/378239.379043
38. Lemieux, G., Lewis, D.: Design of Interconnection Networks for Programmable Logic. Kluwer Academic Publishers (2004)