Design tools for a DNA-guided self-assembling carbon nanotube technology

C Dwyer[1], V Johri[1], M Cheung[2], J Patwardhan[1], A Lebeck[1], D Sorin[2]

1: Department of Computer Science, Duke University, Durham, NC 27708.

2: Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708.

Corresponding author's e-mail: dwyer@cs.duke.edu

Abstract

The shift in technology away from silicon CMOS to novel nanoscale technologies requires new design tools. In this paper, we explore one particular nanotechnology: carbon nanotube transistors that are self-assembled into circuits by using DNA. We develop design tools and demonstrate how to use them to develop circuitry based on this nanotechnology.

1. Introduction

The rapid advance of silicon technology toward single-nanometer device feature sizes and the foreshadowed difficulties with these developments are driving research into alternative technologies and architectures that can either replace or supplement existing silicon technologies [7]. Researchers are exploring novel nanoscale components, such as carbon nanotube transistors, as well as techniques for integrating these components into circuits. Existing top-down fabrication technology (i.e., photolithography) cannot resolve dimensions as small as desired, which has spurred research in self-assembling fabrication processes.

We are exploring one particular set of nanotechnologies, carbon nanotube FETs (CNFETs), that are self-assembled using DNA as a scaffolding. CNFETs have been demonstrated to exhibit excellent switching properties [1, 10, 2, 8, 11]. To fabricate a circuit out of CNFETs, which are on the order of 1.7nm in diameter (far smaller than current photolithographic capabilities), we plan to exploit the self-assembly properties of DNA. Strands of DNA connect to each other if the base pairs on the strands are complementary. Our proposed technology is fabricated by using lattices of DNA as shown in Figure 1 [15][1].

Then, by attaching a tag of single-strand DNA (ssDNA) to a specific point on the lattice and the complementary ssDNA tag to one end of a nanotube (which is called *functionalization* of the nanotube), we could programmably attach that end of the nanotube to the scaffold [5]. The precursor work for this has recently been demonstrated by making a back-gated CNFET with source and drain leads using metallized DNA strands [9].

We plan to use multiple pairs of DNA tags to create different connections on the lattice, in order to connect both ends of multiple nanotubes to the scaffold. If we attach two nanotubes such that they are perpendicular and cross each other, the top one (chosen to be metallic) acts as a gate and the bottom one (chosen to be semiconducting) acts as a channel in a CNFET. Figure 2 shows cross-connected nanotubes spanning cavities in the DNA lattice. We also plan to attach single nanotube wires to each side of each cavity. To implement a connection between two nanotube wires, we will specify a DNA tag at the gap between the wires that attracts a metallic nanosphere that later serves as a nucleation site for chemical electroless plating. By programmably attaching CNFETs and specifying wire connections, we will fabricate circuits with this technology.

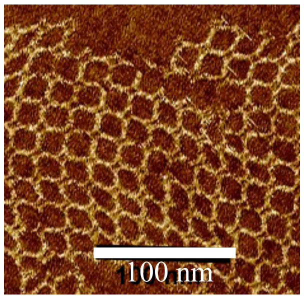


Figure 1. AFM image of DNA lattice

However, due to current limitations in DNA self-assembly technology we do not believe that we can create lattices greater than ~2 microns on a side. Thus, the functionality on any given lattice is limited. To develop large-scale systems, we plan to interconnect multiple lattice nodes. We are currently exploring computer architectures that are amenable to this kind of computational substrate. The focus of this paper is on the computer-aided design tools needed for this work.

To design lattice circuitry, we need various custom design tools to facilitate the new technology. For example, we need to be able to specify where nanotubes assemble to the lattice and specify their connectivity. Since placement and routing are produced by DNA self-assembly, these processes require design tools that can choose DNA tags (which sequence of

[1] Image courtesy of Thomas LaBean.

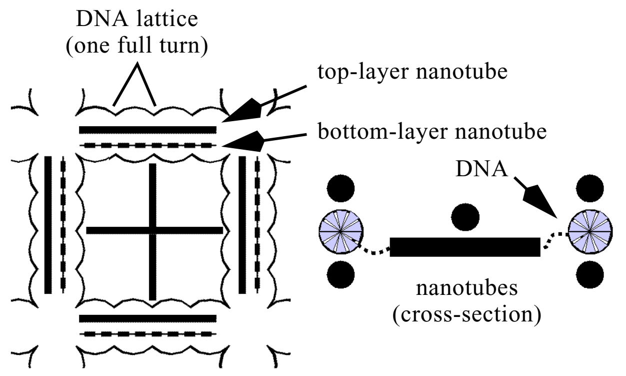


Figure 2. Schematic of the DNA lattice scaffold

base pairs) for the scaffolding and for functionalizing the nanotubes. Unlike existing design tools for silicon CMOS technology, our tools must produce DNA tag sequences rather than mask artwork.

In this paper, we present the design tools and flow we have developed for the design of self-assembled circuitry. Section 2 describes our method and illustrates where we have introduced custom tools to handle the new technology. We also briefly describe a computer architecture that we have developed that is amenable to the self-assembling technology and the behavioral simulator we are using to explore architectural mechanism design. Section 3 describes the application of our design tools to several logic circuits. These results demonstrate a new capability in designing self-assembled circuitry.

2. Methodology

In this section, we present our design methodology. We begin with a high-level overview of the design flow, and then we discuss the architectural, circuit, and DNA level design flows.

2.1 Overview

Figure 3 illustrates the design flow we have applied to the self-assembling process. The design flow begins with an architectural description that is used to manually create a behavioral simulator that can verify the high-level procedural operations of the system. Once verified, the behavioral description can be used to manually capture gate-level modules for input to a complementary transistor synthesis tool. We have focused our efforts on the generation of layout for our process assuming a transistor level module description. Once a feasible layout has been generated and verified it is used to create an ordered set of DNA sequence allocations for fabrication. The allocation process orders the assembly of the circuit so that a constant number of DNA sequences can be used independently of the circuit size.

The remainder of this section discusses the methods we use to design systems and circuits in this self-assembling technology. Section 2.2 describes our architectural design process and an example architecture. Section 2.3 discusses our circuit design process and leads into the discussion in Section 2.4 on device design and a discussion of the self-assembling process in Section 2.5.

2.2 Architectural Design

The larger context of our project is to develop self-assembled computing systems and this requires behavioral design and simulation tools. We have developed an *active network* architecture that is amenable to self-assembly, and we briefly describe the system here to motivate our device-level design tool work.

The limited circuit size in our self-assembling technology precludes making single circuits that can perform all operations. Instead, we assemble several different circuit (node) types (e.g., ALU, memory) into a larger network. For ease of fabrication, nodes are randomly interconnected with bit-serial links. The large scale network of nodes is divided into *cells*, each of which has a *via* that connects it to the micro-scale. A configuration phase at system startup imposes some limited structure on this otherwise random sea of nodes. Configuration maps out defective nodes and links, organizes the memory nodes into a memory system, and establishes routing options within the network.

In our active network execution model, an execution packet traverses this network in search of the resources it needs to perform its instructions. For example, a packet needing to execute an add would search for an ALU node. Each execution packet contains instructions, an accumulator, and all the

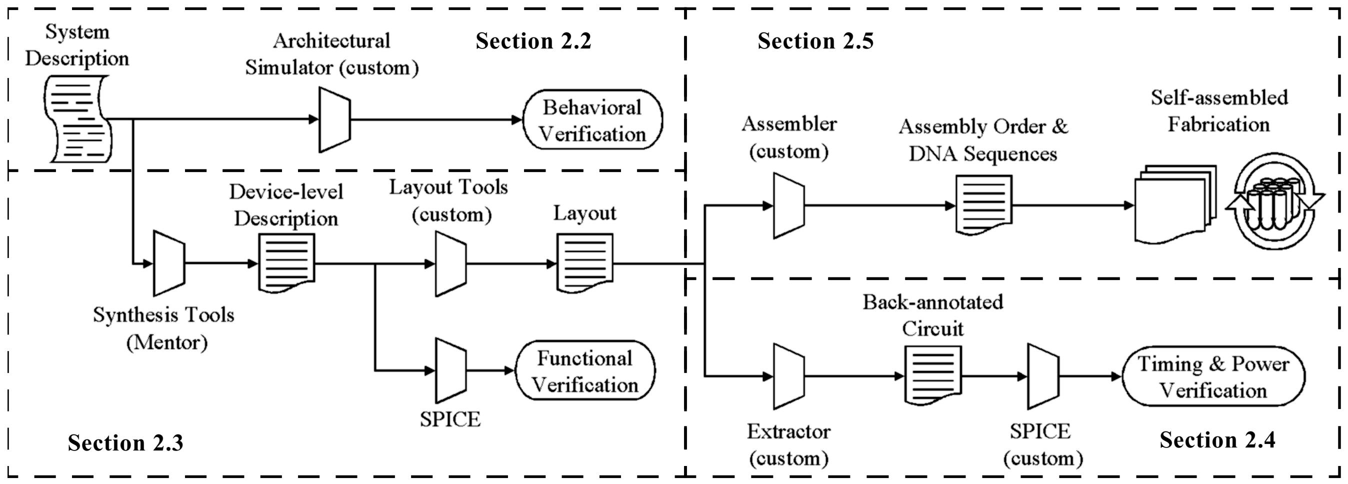


Figure 3. Design tool flow

operands necessary for executing the instructions. By storing everything in the packet rather than at specific nodes, we reduce the per-node storage requirements. After executing an instruction, the packet integrates the result into its accumulator for use by subsequent instructions in the packet. The active network execution model enables us to encode a series of dependent instructions within a single packet, and it enables execution to take an arbitrary path through the network.

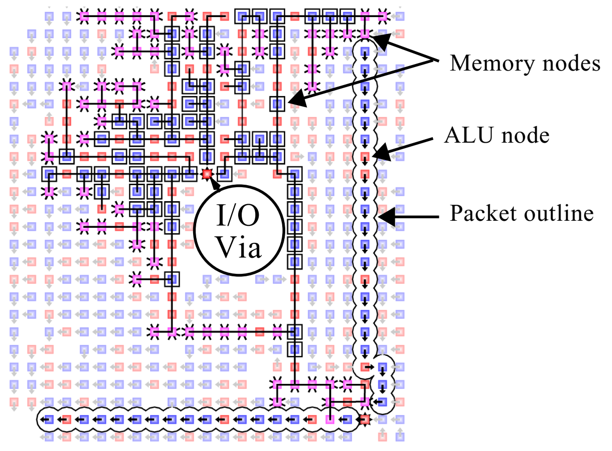


Figure 4. Network of circuit nodes illustrating the behavioral execution of a packet.

The memory system is composed of the memory nodes in the network. The configuration phase allocates unique identifiers to each memory node. To access memory (e.g., to perform a load), an execution packet searches for a *memory port node*, which is similar to a memory node but has been configured to service memory requests. The memory port node creates a *memory packet* that it then sends to the cell's serialization point, a specific node attached to the via, from which it is broadcast to all memory nodes in the cell. The corresponding memory node responds by broadcasting the result, and the memory port that issued the request receives this response and passes it to the execution packet.

We have developed a custom simulator to model a single cell at the bit-serial link level to capture network communication behavior. We also use the simulator to verify architectural behavior and to gather insight into execution mechanisms and optimizations. Figure 4 illustrates the execution of a simple series of instructions (LOAD X, LOAD Y, ADD, STORE Z). The execution begins in the upper right corner at a memory port node and follows a path toward the lower left corner as the execution packet searches for the appropriate nodes.

A detailed treatment of the system and its configuration and execution model can be found elsewhere [12]. We have included this brief overview here as motivation for our tool development and as an example behavioral simulation tool that addresses some of the issues of designing nanoscale self-assembled systems. The behavioral simulator models system-level aspects of this self-assembling technology including node interconnections. The remainder of our tools focus on the design of individual nodes.

2.3 Circuit Design

The complete circuit description for the architecture described in Section 2.2 is not present here. Instead we demonstrate our process for a NAND gate, full-adder, and SR-latch. The circuit level flow begins with a device (transistor) level description of the system generated by the synthesis tools; in our case, a suite of tools from Mentor Graphics, Inc. produces this output. The transistor netlist is used to verify the functional properties of the circuit using a switch-level simulator. The unique constraints of our self-assembling process (~100 x 100 FETs) make large circuits infeasible. Fortunately that means that a high resolution SPICE simulation can be used to verify circuit functionality before the layout process. In this limited sense, the self-assembling technology simplifies the design flow compared to a conventional technology because it forces system architects to explicitly partition their designs.

The layout process can include automated tools and/or manual full-custom steps. Our custom layout tools provide an interface for manual layout and the exploration of design spaces through automated techniques. The constrained circuit sizes in our technology appear to make it more feasible to apply fully automated layout generation to the entire system than with conventional technologies.

The generated layout is used by a custom circuit extractor to back-annotate the original circuit with wire models and additional parasitics derived from the geometry of the layout. This requires the extractor to model the as-fabricated structure of the circuit and use geometric relationships to refine the circuit. The back-annotated circuit is simulated by the modified SPICE kernel and empirical device models as described in Section 2.1. The results of this simulation are used to verify the timing and power constraints of the circuit and can be used to make decisions that feedback to the system description and earlier design process.

2.4 Device Design

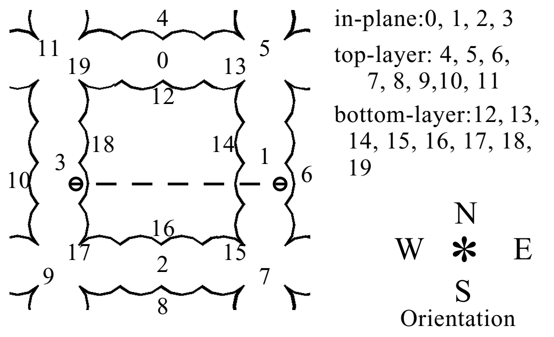
The device level flow begins with the transistor level description of the logic module that has been synthesized and optimized by a logic design tool. This description is first verified using a switch-level simulator and then fed to a custom automated place and route layout generator. A SPICE deck is then extracted from the generated layout to estimate performance including wire delays and other parasitics. We use a modified SPICE 3f5 kernel similar to [6] and a custom semi-empirical device model for the CNFETs [3, 11] and parasitics [3] to estimate the performance of the circuitry. The details of this model are currently under review for publication and are only briefly mentioned here for completeness. Our results are consistent with the reference data used in our models and recent observations of CNFET performance [1, 13].

2.5 Self-assembled DNA Design

In an analogous fashion to the generation of masks from layout artwork, DNA sequences are used by custom DNA strand manufacturers to create the DNA tags that will be attached to carbon nanotubes and direct the assembly of the circuit during the fabrication process.

The layout is used by a custom *assembler*. The assembler is a tool that renders the layout into an ordered sequence of assembly steps and DNA tag identifiers as well as the self-assembling components (nanotubes or nanoparticles) to which the tags must be attached.

The assembler uses the stitching algorithm illustrated in Figure 5 to order the DNA tag allocations. This pattern is


in-plane:0, 1, 2, 3
top-layer: 4, 5, 6, 7, 8, 9,10, 11
bottom-layer:12, 13, 14, 15, 16, 17, 18, 19
N
W
E
S
Orientation

E.g. (E3, W6) represents the dashed line which connects from in-plane location 3 to top-layer location 6.

Figure 6. Tag location legend. Each number represents the location of a DNA tag. N, E, S, and W designate an orientation with respect to the location.

better than simple line scans because it minimizes the span of the lattice at all stages which appears to be important in forming planar DNA lattices [14].

Our assembly ordering assumes a single "active" cavity (i.e. one available for binding nanotubes, etc.) in the scaffold at each step in the process. That is, we assume the scaffold is extended in the direction of the next cavity (as specified by the assembly pattern) before each assembly step. To prevent components from binding to previously assembled cavities the lattice can be passivated with short DNA fragments that bind to unused locations on the cavity after each assembly step.

At each step the assembler generates the tags and components specified in the layout for that cavity. Figure 6 illustrates the positions of each tag in the cavity.

Each position has four associated orientations: north, east, south, and west. That is, a DNA tag can bind a component at any of its orientations. Further, nanoparticles bind to a reserved portion of the DNA tag and do not prevent nanotubes from binding to the same tag (i.e., nanotubes have their own portion as well).

A tag and orientation are specified for each attachment point of a component. For example, a nanoparticle can be specified with a single tag and orientation. The specifier N7 represents the north (side) of location 7. A nanoparticle at N7 and a nanotube connecting (N7, S5) will fuse during the metallization process.

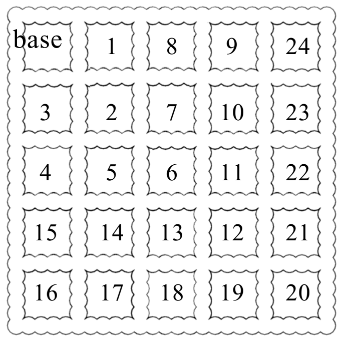


Figure 5. The assembly order follows a radial boustrophedonic pattern to build the lattice.

The exact sequence of the DNA used for each tag is taken from a pool of sequences that are known not to interfere with each other. The pool is generated using binding energy minimization techniques to avoid cross-hybridization [4]. The size of this pool using our technology and this assembly ordering is 20 sequences. However, the number of DNA sequences required to form one additional active cavity in the scaffold lattice scales as the perimeter of the device grows and minimizing this is a topic of future research.

The output from the assembler, the assembly ordering and DNA tag allocation, is then used to direct the self-assembly of the circuit. The remainder of this paper describes the application of our tools to several simple logic structures.

3. Case Studies

We demonstrate our design methodology by designing a NAND gate, full-adder, and SR-latch. While these circuits are trivial, the process we have developed introduces the tools needed by this self-assembling technology.

Each circuit layout was generated manually and converted to a SPICE netlist for switch-level verification. The NAND layout is illustrated in Figure 7.

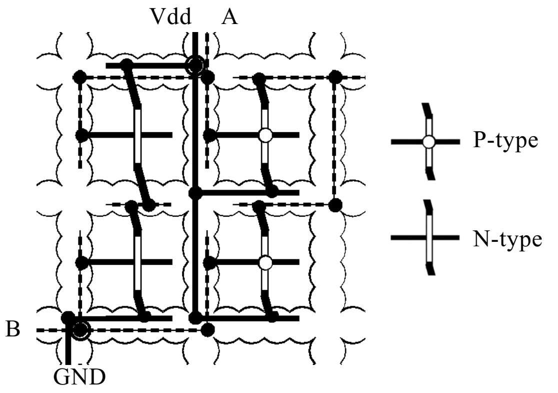


Figure 7. Self-assembled NAND layout (boundary cavities have been removed) Only nanotubes, particles, and scaffold DNA are shown.

The other two circuits were designed in a similar fashion and Table 1 lists several simple measures of their layout.

Table 1. Layout Features

	Total occupied area (cavities)	Span (cavities)	Layout efficiency (FETs/cavity)
NAND	9	4 x 4	0.44
full-adder	65	10 x 11	0.43
SR-latch	12	5 x 4	0.60

Each cavity has an area of 4 x 10^{-4} μm^2 determined by the spacing of the DNA scaffold, which has been designed to span a 20 nm pitch.

Table 2 lists the transition energies ($\mathbf{E_t}$) and switching delay ($\mathbf{t_d}$) for each circuit. These results were obtained by loading each device output with a FO-4 inverter tree and conditioning each square wave input through a series of four CNFET inverters. The results in Table 2 are from the worst-case single-input transition event for each circuit.

Table 2. Circuit Performance

	E_t (10^{-18} J)	t_d (ps)
NAND	38.8	10.3
full-adder	119	16.7
SR-latch	48.6	14.4

In Table 3, we illustrate the assembly sequence of the NAND gate as generated by our tool. Each row represents a single active cavity (indicated by the number in braces as specified in Figure 5) and the assembly actions needed to complete it. For example, steps 0-2 are used to extend the lattice in a particular direction to follow the stitching pattern and steps 4-14 assemble the components for the cavity (cavity 2 is active). In this table, 'V' indicates a Vdd connection (1.0 V) and 'G' indicates a ground connection (0 V). These connections could be made out of the plane of the lattice (or node) to microscale contacts above and below the lattice.

Some cavities are not used in this layout due to the placement of the circuit on an overly large lattice (in this case a 5 x 5 cavity lattice). Table 3 illustrates how tangled the self-assembly process can become even for trivial design problems (e.g., a NAND gate). This underscores the need for the continued development of design tools capable of handling this emerging technology.

4. Conclusions

The development of self-assembling technologies that can either replace or supplement existing silicon technologies requires new design automation tools because of the distinctions between self-assembling and conventional photolithography processes. This fundamental change in technology motivates the development of design tools that address these differences.

In this paper, we have presented the design tools for one such self-assembling circuit technology. The technology we explore, which uses DNA to programmably self-assemble the circuit components, requires different tools than those used for silicon CMOS design. Starting with a circuit description, we use tools for placement, routing, and electrical simulation to develop a viable circuit. We then use a custom tool to generate DNA tag sequences that will enable this circuit to be fabricated using self-assembly.

Future work will extend this tool chain beyond individual circuit nodes in order to encompass large-scale designs, including computer architectures.

Table 3. DNA sequence allocations and assembly order for a NAND gate. The number in braces indicates the active cavity as specified in Figure 5. Each step is numbered.

Lattice extension	CNFETs	metallic CNTs	nanoparticles
[2] 0. base 1. East 2. South	N-type: 5. (S0, N2)	3. (E3, W1) 12. (S19, N17) 13. (E11, W5) 14. (E19, W13)	4. E18, 6. N16, 7. S4 8. W13, 9. W5, 10. E19, 11. S19
[4] 15. West 16. South			17. G15
[5] 18. East	N-type: 21. (S0, N2)	19. (E3, W1) 25. (S19, N17)	20. E18, 22. N8 23. S12, 24. N17
[6] 26. East	P-type: 29. (S0, N2)	27. (E3, W1) 37. (S11, N9) 38. (S19, N17) 39. (E11, W5) 40. (E19, W13)	28. E18, 30. N8 31. S12, 32. W13 33. E11, 34. S11 35. N9, 36. N17
[7] 41. North	P-type: 44. (S0, N2)	42. (E3, W1) 51. (S11, N9) 52. (S19, N17) 53. (E19, W13)	43. E18, 45. N8 46. S12, 47. W13 48. S19, 49. S11 50. N9
[8] 54. North		61. (S11, N9) 62. (S19, N17) 63. (E19, W13)	55. W13, 56. E19 57. S19, 58. N9 59. N17, 60. V9
[9] 64. East			65. G19
[10] 66. South		72. (S19, N17) 73. (E19, W13)	67. W13, 68. E19 69. S19, 70. N17 71. G13
[13] 74. South 75. South 76. West		78. (E11, W5)	77. E11
[14] 79. West		85. (S11, N9) 86. (S19, N17) 87. (E11, W5) 88. (E19, W13)	80. W13, 81. E11 82. E19, 83. S11 84. N17
[15] 89. West		92. (E19, W13)	90. W13 91. G15

5. References

[1] J. Appenzeller and D. J. Frank. Frequency dependent characterization of transport properties in carbon nanotube transistors. *Applied Physics Letters*, 84(10):1771–1773, Mar. 2004.

[2] P. Avouris, J. Appenzeller, V. Derycke, R. Martel, and S. Wind. Carbon nanotube electronics. In *International Electron Devices Meeting Digest*, pages 281–284, Dec. 2002.

[3] P. J. Burke. An RF Circuit Model for Carbon Nanotubes. *IEEE Transactions on Nanotechnology*, 2(1):55–58, Mar. 2003.

[4] R. Deaton, J. Chen, H. Bi, and J. A. Rose. A software tool for generating non-cross hybridizing libraries of DNA oligonucleotides. In *DNA Computing: 8th International Workshop on DNA-Based Computers, DNA8 Sapporo, Japan, June 10-13, 2002. Revised Papers*, volume 2568, pages 252–261, 2003.

[5] C. Dwyer, M. Guthold, M. Falvo, S. Washburn, R. Superfine, and D. Erie. DNA Functionalized Single-Walled Carbon Nanotubes. *Nanotechnology*, 13:601–604, 2002.

[6] C. Dwyer, L. Vicci, and R. M. Taylor. Performance Simulation of Nanoscale Silicon Rod Field-Effect Transistor Logic. *IEEE Transactions on Nanotechnology*, 2(2):69–74, June 2003.

[7] M. Forshaw, R. Stadler, D. Crawley, and K. Nikolic. A short review of nanoelectronic architectures. *Nanotechnology*, 15:S220–S223, 2004.

[8] M. S. Fuhrer, J. Nygard, L. Shih, M. Forero, Y.-G. Yoon, M. S. C. Mazzoni, H. J. Choi, J. Ihm, S. G. Louie, A. Zettle, and P. L. McEuen. Crossed Nanotube Junctions. *Science*, 288:494–497, Apr. 2001.

[9] K. Keren, R. S. Berman, E. Buchstab, U. Sivan, and E. Braun. DNA-Templated Carbon Nanotube Field-Effect Transistor. *Science*, 302:1380–1382, Nov. 2003.

[10] R. Martel, V. Derycke, J. Appenzeller, S. Wind, and P. Avouris. Carbon nanotube field-effect transistors and logic circuits. In *Proceedings of the 39th Design Automation Conference*, pages 94–98, June 2002.

[11] P. L. McEuen, M. S. Fuhrer, and H. Park. Single-Walled Carbon Nanotube Electronics. *IEEE Transactions on Nanotechnology*, 1(1):78–85, Mar. 2002.

[12] J. P. Patwardhan, C. Dwyer, A. R. Lebeck, and D. J. Sorin. Circuit and System Architecture for DNA-Guided Self-Assembly of Nanoelectronics. In *Foundations of Nanoscience: Self-Assembled Architectures and Devices*, Apr. 2004.

[13] S. Rosenblatt, Y. Yaish, J. Park, J. Gore, V. Sazonova, and P. L. McEuen. High Performance Electrolyte Gated Carbon Nanotube Transistors. *Nano Letters*, 2(8):869–872, 2002.

[14] H. Yan, S. H. Park, L. Feng, G. Finkelstein, J. H. Reif, and T. H. LaBean. 4x4 DNA Tile and Lattices: Characterization, Self-Assembly, and Metallization of a Novel DNA Nanostructure Motif. In *Proceedings of the Ninth International Meeting on DNA Based Computers (DNA9)*, June 2003.

[15] H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean. DNA Templated Self-Assembly of Protein Arrays and Highly Conductive Nanowires. *Science*, Sept. 2003.