NATURE: A Hybrid Nanotube/CMOS Dynamically Reconfigurable Architecture *

Wei Zhang
Dept. of Electrical Engineering
Princeton University,
Princeton, NJ 08544
weiz@princeton.edu

Niraj K. Jha
Dept. of Electrical Engineering
Princeton University,
Princeton, NJ 08544
jha@princeton.edu

Li Shang
Dept. of Electrical Engineering
Queen's University, Kingston,
ON K7L 3N6
li.shang@queensu.ca

ABSTRACT

Recent progress on nanodevices, such as carbon nanotubes and nanowires, points to promising directions for future circuit design. However, nanofabrication techniques are not yet mature, making implementation of such circuits, at least on a large scale, in the near future infeasible. However, if photolithography could be used to implement circuits using these nanodevices, then hybrid nano/CMOS chips could be fabricated and the benefits of nanotechnology could be utilized immediately. A startup company, called Nantero, has developed and implemented a non-volatile nanotube random-access memory (NRAM) using photo-lithography that is considerably faster and denser than DRAM, has much lower power consumption than DRAM or flash, has similar speed to SRAM and is highly resistant to environmental forces (temperature, magnetism). In this paper, we propose a novel high performance reconfigurable architecture, called NATURE, that utilizes CMOS logic and NRAMs. Use of the highly-dense NRAMs allows large on-chip configuration storage, enabling fine-grain run-time reconfiguration and temporal logic folding of a circuit before being mapped to the architecture. This can significantly increase the logic density of NATURE (by over an order of magnitude for larger circuits) while remaining competitive in performance. Compared to traditional reconfigurable architectures, NATURE also allows the designer the flexibility to adjust the level of logic folding in order to improve performance or perform area-performance trade-offs. Experimental results establish its efficacy and give comparisons with today's mainstream FPGA technology which does not allow logic folding.

Categories and Subject Descriptors: B.7.1 [**Types and Design Styles**]: Advanced technologies
General Terms: Design, performance
Keywords: NRAM, run-time reconfiguration, logic folding

1. INTRODUCTION

Since CMOS is expected to approach its physical limits in the coming decade [1], to enable future technology scaling, intensive research has been directed towards the development of nanoscale molecular devices, such as carbon nanotubes [2, 3], which demonstrate superior characteristics to MOSFETs in terms of integration density, performance, power consumption, etc. However, lack of a mature fabrication process is a roadblock in implementing chips using these nanodevices. In addition, if nanoelectronic circuits/architectures are implemented using bottom-up chemical self-assembly techniques, then the chip defect levels are expected to be high (between 1% and 10%) [4]. To be able to deal with such high defect levels, regular architectures, such as an FPGA, are favored. In addition to being regular, reconfigurable architectures allow reconfiguration around fabrication defects as well as run-time faults. Researchers have presented various designs of programmable nanofabrics, such as the molecular logic array (MLA) [5], nanowire programmable logic array (PLA) [6], and CMOS/nanowire hybrid architecture CMOL [7]. However, even though such self-assembly based fabrics may be feasible in the next decade, they are not ready for large-scale fabrication today.

In contrast, in this paper, a hybrid CMOS/NAnoTUbe REconfigurable architecture, called NATURE, is presented, which can be fabricated with CMOS-compatible manufacturing processes and leverage the beneficial aspects of both technologies immediately. NATURE is based on CMOS logic and NRAMs. An NRAM is a non-volatile, high-density, high-speed and low-power nanomemory. It is being developed by a company called Nantero [8]. Nantero has already prototyped 10 Gbit NRAMs. Such chips are expected to be commercially available in the near future. Besides the outstanding properties of NRAMs mentioned above, an important advantage of NRAMs is that they can be fabricated using photolithography. Thus, NATURE can also be fabricated with currently-available processes, alleviating the concerns related to the ability to fabricate large chips and defect levels.

In NATURE, we exploit the excellent properties of NRAMs by distributing them in the reconfigurable fabric to act as the main storage for storing large sets of reconfiguration bits. Thus, the logic implemented in the logic elements (LEs) of the reconfigurable architecture and interconnects can be reconfigured every few cycles, or even every cycle, making both coarse-grain and fine-grain run-time reconfiguration possible. Each LE can be reconfigured independently. Note that traditional reconfigurable architectures do allow partial dynamic reconfiguration [9], in which only a part of the architecture is reconfigured at run-time. However, the reconfiguration latency is still high, allowing it to be used only sparingly. They may also allow only coarse-grain [10] or multi-context reconfiguration (with very few contexts) [11], due to the area overhead associated with the configuration SRAMs. In NATURE, NRAMs store a relatively large set of configuration bits for every programmable element at a relatively small area overhead. At the same time, on-chip NRAMs make reconfiguration very fast (of the order of $160ps$). Hence, NATURE makes it possible to perform fine-grain cycle-level reconfiguration for the first time.

The ability to reconfigure NATURE every few cycles leads to the concept of temporal logic folding, i.e., the possibility of folding the logic circuit in time and mapping each fold to the same LEs in the architecture. This provides significant gains (an order of magnitude or more for larger circuits) in the area-delay product compared to traditional reconfigurable architectures, while allowing the flexibility of trading

*This work was supported by NSF under Grant No. CCF-0429745.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

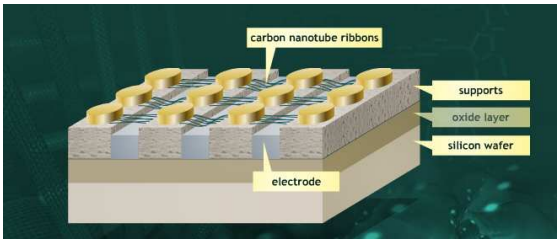

Figure 1: Structure of an NRAM [8]

area for performance. The high logic density provided by logic folding makes it possible to use much cheaper chips with smaller capacities to execute the same applications, hence, making them attractive for use in cost-conscious embedded systems. Note that other reconfigurable architectures, such as PipeRench [12], allow later stages of a pipeline to be folded back and executed in a set of LEs that executed an earlier stage. This can be thought of as coarse-grain temporal folding. However, such architectures are largely limited to stream media or DSP applications. NATURE does not have these limitations and is very general in the applications that can be mapped to it.

It is worth pointing out that the success of NATURE is not tied to NRAMs. There are other contenders for the emerging non-volatile universal memory market, such as phase change RAM [13], magnetoresistive RAM [14] and ferroelectric RAM [15], that could also be considered instead of NRAMs.

2. BACKGROUND

In this section, we provide some background material on the properties of nanotubes and operation of NRAMs.

2.1 Carbon nanotubes

Carbon nanotubes are hollow cylinders composed of one or more concentric layers of graphite [16]. The diameter of a nanotube is usually a few nanometers and length up to millimeters. Nanotubes exhibit unique electronic, mechanical and chemical properties which make them very attractive building blocks for molecular electronics. For example, carrier transport in nanotube is ballistic in the micrometer range and it can carry current densities as high as $10^9 A/cm^2$. Moreover, nanotube has thermal conductivity that is twice that of diamond.

2.2 NRAM

The theory behind NRAMs is discussed in [17]. Fig. 1 shows the basic structure of an NRAM [8]. Memory cells are fabricated in a two-dimensional array using photo-lithography. Each memory cell comprises multiple suspended nanotubes, support and electrode. The memory state is determined by the state of the suspended nanotubes – whether they are bent or not leads to well-defined electrostatically switchable ON/OFF states. When opposite voltages are applied to the support and electrode of a memory cell, the suspended nanotubes are bent due to Coulombic force, reducing the resistance between the nanotubes and electrode to as low as several hundred ohms, corresponding to the "1" state. On the other hand, when the same high voltage is applied to the support and electrode, the nanotube remains straight or "springs" back from the "1" state, resulting in a resistance of several Gigaohms, which is defined as the "0" state. Such ON/OFF states have been shown to be both electrically and mechanically very stable. The above discussion illustrates the writing process to NRAM cells. After writing, the voltages can be removed without changing the cell state, which is preserved by van der Waals forces. Hence, NRAMs are non-volatile. Reading from an NRAM is similar to reading from traditional memories. A single NRAM cell can be addressed using the word and bit lines. When the reading voltage is applied to the support and electrode, different resistances corresponding to different states result in different output voltages.

3. TEMPORAL LOGIC FOLDING

The basic idea behind logic folding is that one can use NRAM-enabled run-time reconfiguration, which is described in Section 4, to realize different Boolean functions in the same LE every few cycles. Let us consider an example. Suppose a

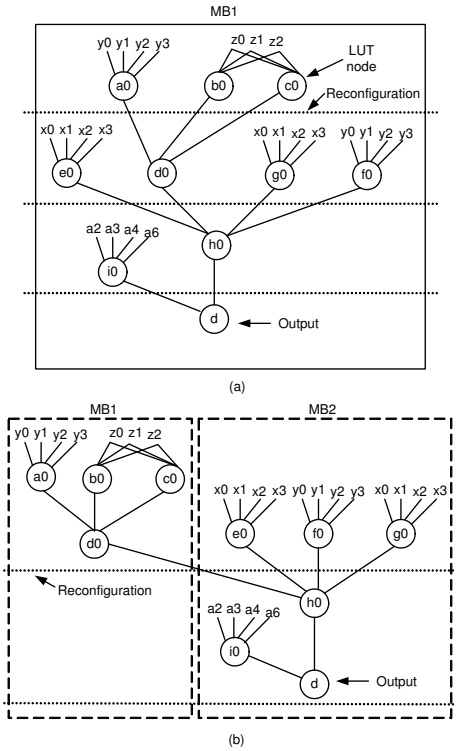

Figure 2: (a) Level-1 folding and (b) level-2 folding

subcircuit can be realized as a series of n serially connected look-up tables (LUTs). Traditional reconfigurable architectures will need n LUTs to implement this subcircuit. However, using run-time reconfiguration, at one extreme all these LUTs can be mapped to a single LE, which is configured to implement LUT1 in the first cycle, LUT2 in the second cycle, and so on, requiring n cycles for execution. In this case, we would need to store n configuration sets in the NRAM associated with this LE. Moreover, all communications between the LUTs mapped to the same LE are local. Hence, global communication is reduced and the routing delay is significantly reduced too. Note that traditional reconfigurable architectures only support partial dynamic reconfiguration or coarse-grain multi-context reconfiguration, and do not allow such fine-grain temporal logic folding. The price we pay for logic folding is reconfiguration time. However, through nanotube SPICE simulation [18], we have found that the time required to output the reconfiguration bits from an NRAM to the SRAM, i.e., the reconfiguration time to switch from one LUT to another, is only around $160ps$. This is small compared to the routing delays.

Logic folding can be performed at different levels of granularity, providing flexibility to perform area-performance tradeoffs. As an example, consider the LUT graph (in which each node denotes a LUT) shown in Fig. 2(a), which denotes what we call level-1 folding. Such a folding implies that the LEs are reconfigured after the execution of each LUT mapped to it. Thus, in this example, four LEs are enough to execute the whole LUT graph. Here, we have assumed that four LEs belong to a macroblock (MB) that is described in Section 4. On the other hand, Fig. 2(b) shows level-2 folding which implies reconfiguration of the LE after the execution of two LUT computations. In this case, two MBs are required. A level-p folding can be similarly defined. The case of no folding corresponds to mapping of circuits to a traditional reconfigurable architecture in which spatial partitioning of the LUT graph is feasible, but not temporal folding. In this case, 10 LEs spread over three MBs would be required.

There are various trade-offs involved in the choice of the folding level. First, when the folding level is large, the cycle period increases since a larger amount of computation needs to be executed in one cycle. The number of LEs needed also increases since they are not time-shared enough. However, the

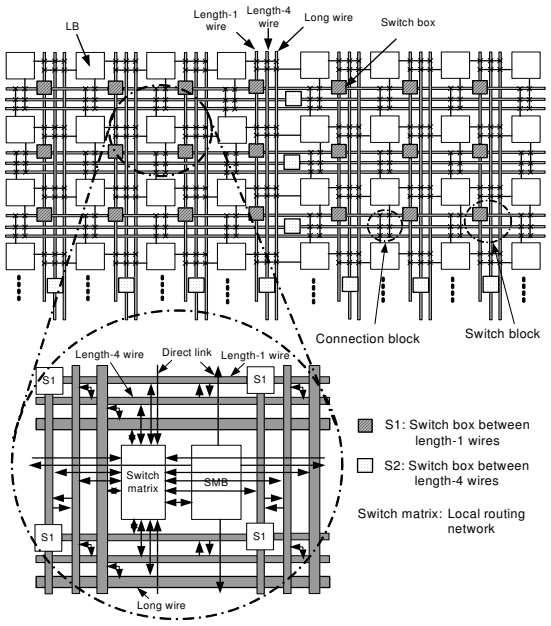

Figure 3: High-level view of the architecture of NATURE

total number of cycles decreases. This fact coupled with the reduction in reconfiguration time may reduce total delay of the circuit. However, this would generally be true when communications between LEs are still local in the folded circuit. If the area required for implementing the circuit is such that long global communication is required in some cycle, then a small folding level may give better performance.

Another important advantage of logic folding is that a much larger circuit can be mapped to NATURE in the same chip area as that of a conventional reconfigurable architecture. To provide more flexibility, NATURE can be even further divided into several sections, each implementing a circuit with a different folding level, running at different clock frequencies. Note that when the number of available LEs is smaller than the number of LUTs in the LUT graph being mapped to them, the best folding level should be the one that best utilizes the available LEs.

4. ARCHITECTURE OF NATURE

A high-level view of NATURE's architecture is shown in Fig. 3. It contains island-style logic blocks (LBs) [19], connected by various levels of interconnect, supporting local and global communications among LBs. Connection and switch blocks are as shown in the figure. $S1$ and $S2$ refer to switch boxes that connect wire segments. An LB contains a super-macroblock (SMB) and a local switch matrix. The inputs and outputs of an SMB are connected to the interconnection network through a switch matrix. Neighboring SMBs are also connected through direct links. We discuss the design issues involving the SMB and interconnect in detail next.

4.1 The SMB architecture

We use two levels of logic clusters in an LB to facilitate temporal logic folding of circuits and enable most inter-block communications to be local, as will be clearer later. The first (i.e., lower) level, called the MB level, is shown in Fig. 4. An MB contains n_1 m-input reconfigurable LEs (in this figure, $n_1 = 4, m = 4$). In the second level, n_2 MBs comprise an SMB, as shown in Fig. 5 (in this figure, $n_2 = 4$). Within an MB or SMB, communications among various components take place through a local crossbar. We use a crossbar instead of a multiplexer at this level in order to speed up local communications. Since a crossbar requires more SRAM control bits, we do pay a slight price in area to obtain the speed-up. As shown in Fig. 4, the m inputs of an LE can arrive from the outputs of other LEs in the MB or the inputs to the MB. Similarly, the inputs of an MB can arrive from the outputs of other MBs or the inputs to the SMB through the switch matrix. The

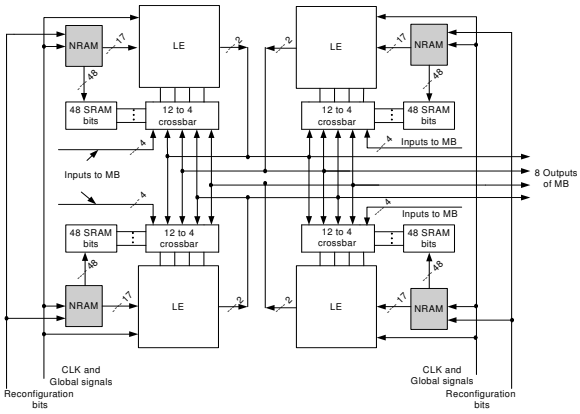

Figure 4: Architecture of an MB

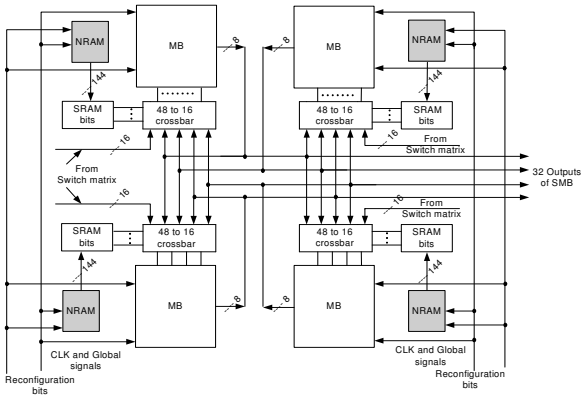

Figure 5: Architecture of an SMB

outputs (two in this instance) from an LE can be used within the MB or go to the upper level SMB or go to other SMBs through the routing network. This logic/interconnect hierarchy maximizes local communications and provides a high level of design flexibility for mapping circuits to the architecture.

An LE implements a basic computation. The structure of an LE is shown in Fig. 6. It contains an m-input LUT and a flip-flop. The m-input LUT can implement any m-variable Boolean function. The flip-flop stores the internal results for future use (the result of a previous stage is often needed by a subsequent stage). A pass transistor is used to decide if the internal result will be stored or not.

4.2 Support for reconfiguration

In this section, we dwell on how NRAMs provide support for reconfiguration. As shown in the previous section, an NRAM is associated with each reconfigurable block and stores its run-time reconfiguration bits. If k configuration sets are stored in an NRAM, then the associated components can be reconfigured k times during execution. As an example, for the MB architecture shown in Fig. 4, 65 reconfiguration bits are required for one configuration set (when $m = 4$). In this set, 16 bits are required for each 4-input LUT, and one bit for determining whether to store the internal result or not, and the remaining 48 bits to reconfigure the crossbar. Hence, when $n_1 = 4$, $m = 4$, and k configuration sets are used, the total number of NRAM bits required for one MB is $65kn_1$. Note that the configuration sets for different LE-crossbar pairs are different, providing immense flexibility to NATURE for reconfiguring such pairs independently of each other.

Let us see next how reconfiguration is performed. Reconfiguration bits are written into NRAMs only at the time of initial

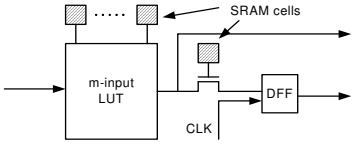

Figure 6: Architecture of an LE

configuration from off-chip storage. For run-time reconfiguration, reconfiguration bits are read out of NRAMs sequentially and placed into SRAM cells to configure the LEs and crossbar/switches to implement different logic functionalities or interconnections. At the rising edge of the clock signal CLK, reconfiguration commences, followed by computation. At the same time, the prior computation result is stored into a register to support computation in the next cycle. If different sections of the chip use different folding levels, their clock frequencies will also differ. In this case, each such section can be controlled by a different clock signal. A limited number of global clock signals can be generated by a clock manager module, as in current commercial reconfigurable chips. Since reconfiguration bits for different logic configurations are read out in order, counters can be used instead of decoders.

Inclusion of NRAMs in the LB incurs an area overhead. To compute this overhead, we manually drew the layout of an LB (with NRAMs and associated interconnects included - see Fig. 3). We used information from [8] and [20] to obtain the relevant parameters for an NRAM cell. Assuming a $100nm$ technology for implementing CMOS logic, $100nm$ nanotube length, and $k = 16$, the NRAMs occupy roughly 20.5% of the LB area. However, through NRAM-enabled logic folding, the number of LBs required to implement a circuit is reduced nearly k-fold. To account for these facts, it is helpful to define the concept of relative logic density. This is the ratio of the amount of logic NATURE can implement in a given amount of area compared to the amount of logic a traditional reconfigurable architecture can. When $k = 16$ and assuming the circuit being implemented can use 16 configurations (as most large circuits would), the relative logic density can be calculated as $16(1 - 0.205) = 12.72$. This means that in the same area, NATURE can implement roughly 12 times more logic than a traditional architecture or equivalently needs 12 times less area to implement the same functionality. Moreover, this relative logic density should remain roughly the same with the scaling of CMOS technology and NRAM cell size.

It can be seen that both the NRAM size and relative logic density vary with the value of k. If k is too small, more global communication may be needed. If it is too large, it may not be possible to make use of the extra configurations, thus leading to wasted NRAM area that could have been put to other use. Since the best k value varies with the specific design, it can be obtained through design space exploration. We have found $k = 16$ works well in practice.

To further improve the performance of the architecture at the expense of increased area, one can use a shadow reconfiguration SRAM to hide the reconfiguration latency for transferring bits from the NRAMs to the SRAMs. This allows one group of SRAM bits to load reconfiguration bits from NRAMs, while another SRAM group supports current computation. The performance improvement due to this feature will depend on the level of logic folding. However, we have currently not implemented this feature.

4.3 Interconnect design

Interconnect design is very important because of its large routing delay and area. Consequently, the routing architecture must be designed to be both fast and area-efficient. In our case, it should also aid logic folding and local communication as much as possible. In our design, we used a hybrid interconnect. Within the SMB, the interconnect is hierarchical to aid logic clusters and local communication. To connect SMBs, we use wire segments of various lengths.

To give a better picture of the interconnect structure, we next address the following routing architecture features [19]:

- The length of each routing wire segment (i.e., how many LBs a routing wire spans before terminating).
- The number of wires (tracks) in each routing channel.
- Location of routing switches and which routing wires they connect.
- The type of each routing switch: pass transistor, tristate buffer or multiplexer (MUX).
- Size of transistors in the switches, and the metal width and spacing of the routing wires.

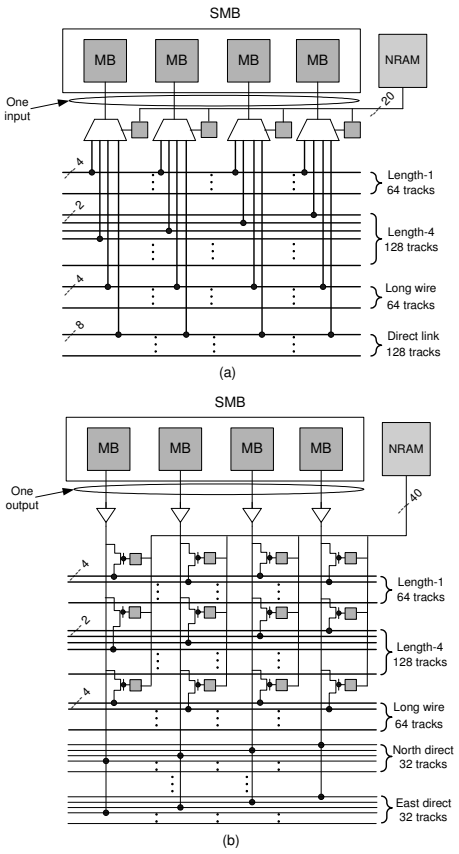

Figure 7: Connection block for (a) one input of an MB, and (b) one output from an MB

For the first feature, since too many short wires decrease circuit performance, while too many long wires provide little routing flexibility and may waste area, we implemented a mixed wire segment scheme including length-1, length-4 and long wires. Length-1 (length-4) wire segments span one (four) LB(s) before connecting to a switch block, while long wires traverse the chip horizontally and vertically, connecting to each LB along the way. There are also direct links from the outputs of one LB to its four neighboring LBs, further facilitating local communications.

To address the second feature, based on the observations in [19], for the architecture instance in which $m = n_1 = n_2 = 4$, $I = 64$, and $O = 32$ (where I/O refers to the number of inputs/outputs of an SMB), we choose 128 horizontal and vertical tracks and assume a $25\%, 50\%$, and 25% distribution for length-1, length-4 and long wires, respectively. In addition, we provide 32 tracks for direct links between adjacent SMBs (since $O = 32$). Fig. 7 illustrates how the inputs/outputs of an SMB are connected to the routing network.

Next, let us consider the design of the connection block, characterized by F_c, and switch block, characterized by F_s (F_c refers to the number of adjacent tracks a pin of an LB can connect to and F_s the number of tracks to which each track entering the switch block can connect [21]). Higher values of F_c and F_s result in higher routing flexibility, however, at the expense of a higher number of switches and hence more routing area. According to [22], for a cluster of N LUTs, F_c can be chosen as $1/N$ of the total number of tracks and F_s should be greater than three in order to be able to achieve routing completion while maintaining area efficiency. We use $F_c = 1/N$ and $F_s = 6$ in our design. Another related important issue is whether or not the internal connection blocks or switch blocks should be populated [19, 23] (such a block is said to be populated if it is possible to make connections from the middle of the block to LBs or other such blocks). When

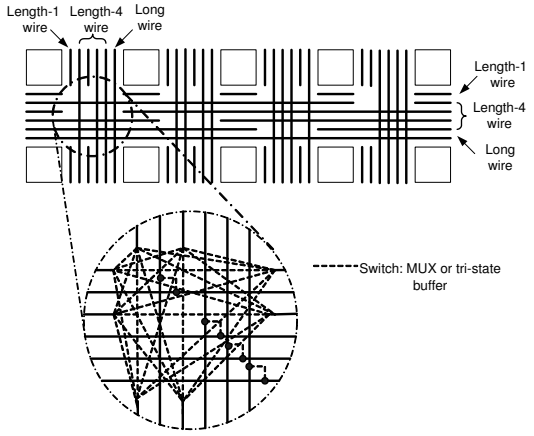

Figure 8: Illustration of a switch block

they are both fully populated, the number of routing tracks required to achieve routing completion can be reduced, at the expense of a larger number of switches being attached to a wire (resulting in more capacitance and, hence, decrease in speed). Based on the investigation in [23], we decided to depopulate the connection blocks and populate the switch blocks to provide the best performance-area advantage. Fig. 8 shows an example connection of disjoint switch blocks.

Next, we come to the types of switches. There are typically three types of switches: pass transistor, MUX and tri-state buffer. Since using pass transistors results in fastest implementations of small crossbars, we choose pass transistors for the local crossbars within the MB and SMB. A MUX has longer delay, but needs fewer reconfiguration bits. Therefore, it is used in the switch matrix to connect to the inputs of an SMB. The outputs of an SMB are connected to long interconnects through tri-state buffers in the switch matrix.

For the last feature, we use pass transistors that are 10 (5) times the size of a minimum-sized transistor for tri-state buffers (MUXes) [19]. Minimum width and spacing are used for the metal wires.

5. EXPERIMENTAL RESULTS

In this section, we present experimental results for various MCNC benchmarks and arithmetic circuits to illustrate the benefits of run-time reconfiguration and logic folding in NATURE. In Section 4.1, we proposed a family of NRAM-based reconfigurable architectures at different levels of granularity in terms of the number of LEs in an MB (n_1), number of MBs in an SMB (n_2), number of inputs per LE (m), number of configuration sets stored in the NRAM (k), etc. It is possible that different architectural instances are best for different types of circuits. Since prior work [22] suggests that a cluster of four 4-input LUTs provides one of the best area-delay trade-offs, we use the architecture instance corresponding to $n_1 = 4$, $n_2 = 4$, and $m = 4$ for all experiments. Parameter k is varied in order to compare implementations corresponding to selected folding levels: level-1, level-2, level-4 and no logic folding (note that the number of NRAM bits increases as we go from no folding to level-4 folding and towards level-1 folding since the number of LE configurations increases). We assume k is as much as needed for each case.

Several small/mid-sized benchmarks were manually mapped to the underlying architecture instance. The depth of the circuit LUT graph, number of LEs, circuit delay, product of number of LEs and delay (this is a proxy for the area-time product; this is reasonable since NATURE is a regular architecture), and frequency are shown in Table 1 for different levels of folding. These results are based on the $100nm$ CMOS and nanotube technology parameters.

We can observe the area/performance trade-offs that become possible because of the use of logic folding. Consider the 64-bit ripple-carry adder. Its LUT graph has 64 LUTs on the critical path. Using level-1 logic folding, the complete adder can be mapped to only two LEs. This, of course, requires reconfiguration of the LEs from the local NRAMs at each cycle. If more LEs are allowed (as in level-2, level-4 and no folding cases), the circuit delay goes down because fewer reconfigurations are required (note that we are not assuming the presence of a shadow SRAM to overlap the reconfiguration and computation times of an LE – if we did, the circuit delay for level-1 folding would go down by roughly 1.6X at the expense of a doubling of the SRAM area). Traditional reconfigurable architectures will require 128 LEs for such an adder (some architectures incorporate a carry generation circuit with each LE; in such a case, they will require 64 LEs although each LE will be larger due to the carry generation circuit overhead) because they cannot perform any temporal logic folding. As the number of required LEs increases, the need for using higher-level (i.e., more global) interconnects to connect them also increases which is one of the reasons traditional reconfigurable architectures are not competitive with ASICs in terms of performance.

Next, consider the area-time product. For larger, more serially-connected circuits of larger depth, the area-time product advantage of level-1 folding relative to no folding is typically larger. For example, for the 64-bit ripple-carry adder, we can see that this advantage is about 35X. This results from a large saving in area while maintaining competitive performance.

We can also see from the table that NATURE can operate at a high frequency. The peak frequency is around 2.86 GHz. From level-1 folding to no-folding, the frequency decreases because increasingly more computation is included in one clock period.

In spite of the fact that traditional reconfigurable architectures devote a vast majority of their area to interconnects, their LE utilization may not be high (an extremely large number of routing tracks may be needed to approach 100% LE utilization [24]). Because of the possibility of cycle-by-cycle reconfiguration in our architecture, one can see why its LE utilization and relative logic density can be very high, with a reduced need for a deep interconnect hierarchy. Thus, NATURE also hints at an evolutionary path for existing reconfigurable architectures in which they use fewer levels in the interconnect hierarchy and use the area thus saved to distribute NRAMs throughout the chip.

6. CONCLUSIONS

In this paper, we presented a novel high-performance run-time reconfigurable architecture, called NATURE. We introduced a recently-proposed high-density, high-speed NRAM into the architecture to enable cycle-by-cycle reconfiguration and logic folding. Choice of different folding levels allows the designer flexibility in performing area-performance trade-offs. The significant increase in relative logic density (more than an order of magnitude for larger circuits) made possible by NATURE can allow the use of cheaper reconfigurable architectures with smaller logic capacities to implement the same functionality, thus giving a boost to their use in cost-conscious embedded systems. We are currently implementing a tool to automatically map different functionalities to NATURE. We are also in the process of extending NATURE to a coarse-grain architecture.

7. REFERENCES

[1] "Semiconductor Industries Association Roadmap." http://public.itrs.net

[2] A. Javey *et al.*, "Carbon nanotube field-effect transistors with integrated ohmic contacts and high-k gate dielectrics," *Nano Letters*, vol. 4, pp. 447–450, Mar. 2004.

[3] W. Zhang and N. K. Jha, "ALLCN: An automatic logic-to-layout tool for carbon nanotube based nanotechnology," in *Proc. Int. Conf. Computer Design*, pp. 281–288, Oct. 2005.

[4] M. Butts, A. DeHon, and S. C. Goldstein, "Molecular electronics: Devices, systems and tools for gigagate, gigabit chips," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2002, pp. 433–440.

[5] S. C. Goldstein and M. Budiu, "Nanofabrics: Spatial computing using molecular nanoelectronics," in *Proc. Int. Symp. Computer Architecture*, June 2001, pp. 178–189.

[6] A. DeHon and M. J. Wilson, "Nanowire-based sublithographic programmable logic arrays," in *Proc. Int. Symp. FPGAs*, Feb. 2004, pp. 123–132.

Table 1: Circuit mapping results for NATURE

Circuit	LUT graph depth	Level-1 folding				Level-2 folding			
		LEs	Delay (ns)	LEs× Delay	Freq. (GHz)	LEs	Delay (ns)	LEs× Delay	Freq. (GHz)
pm1	3	9	1.05	9.45	2.86	11	1.16	12.76	1.72
sct	4	9	1.40	12.60	2.86	16	1.16	18.56	1.72
cm163a	4	5	1.40	7.00	2.86	8	1.16	9.28	1.72
cc	4	14	1.64	22.96	2.44	19	1.51	28.69	1.32
z4ml	5	4	1.75	7.00	2.86	4	1.56	6.24	1.92
poler8	7	17	4.10	69.70	1.71	17	3.02	51.34	1.32
cordic	7	7	2.45	17.15	2.86	12	2.32	27.84	1.72
lal	7	18	2.45	44.10	2.86	28	2.32	64.96	1.72
ldd	7	14	2.87	40.18	2.44	17	2.90	49.30	1.38
9symml	19	23	11.12	255.76	1.71	26	7.55	196.30	1.32
alu2	31	33	18.14	598.62	1.71	37	13.04	482.48	1.23
16-bit ripple-carry adder	16	2	5.60	11.20	2.86	4	4.16	16.64	1.92
32-bit ripple-carry adder	32	2	11.20	22.40	2.86	4	8.32	33.28	1.92
64-bit ripple-carry adder	64	2	22.40	44.80	2.86	4	16.64	66.56	1.92
16-bit carry-lookahead adder	10	13	4.10	53.30	1.71	26	3.78	98.28	1.32
32-bit carry-lookahead adder	18	13	7.38	95.94	1.71	26	6.80	176.80	1.32
64-bit carry-lookahead adder	34	13	13.94	181.22	1.71	26	12.35	321.10	1.32
16-bit carry-select adder	8	30	4.68	140.40	1.71	55	3.02	166.10	1.32
32-bit carry-select adder	14	30	8.19	245.70	1.71	55	5.29	290.95	1.32
64-bit carry-select adder	26	30	15.21	456.30	1.71	55	9.82	540.10	1.32
8-bit multiplier	8	16	3.28	52.48	2.44	32	3.26	104.32	1.23
16-bit multiplier	16	32	9.36	299.52	1.71	64	6.52	417.28	1.23
32-bit multiplier	32	64	18.72	1198.08	1.71	128	13.04	1669.12	1.23
Circuit	LUT graph depth	Level-4 folding				No folding			
		LEs	Delay (ns)	LEs× Delay	Freq. (GHz)	LEs	Delay (ns)	LEs× Delay	Freq. (GHz)
pm1	3	22	0.70	15.40	1.43	22	0.54	11.88	1.85
sct	4	32	1.14	36.48	0.88	32	0.98	31.36	1.02
cm163a	4	14	0.87	12.18	1.15	14	0.71	9.94	1.41
cc	4	17	1.08	18.36	0.93	34	0.92	31.28	1.09
z4ml	5	6	1.72	10.32	1.16	9	0.88	7.92	1.14
poler8	7	17	2.14	36.38	0.93	33	1.46	48.18	0.68
cordic	7	22	2.08	45.76	0.96	28	1.46	40.88	0.68
lal	7	48	1.84	88.32	1.09	73	1.28	93.44	0.78
ldd	7	30	2.32	69.60	0.86	44	1.66	73.04	0.60
9symml	19	38	7.45	283.10	0.67	124	4.71	584.04	0.21
alu2	31	43	11.92	512.56	0.67	211	7.37	1555.07	0.14
16-bit ripple-carry adder	16	8	3.44	27.52	1.16	32	2.87	91.84	0.35
32-bit ripple-carry adder	32	8	6.88	55.04	1.16	64	6.01	384.64	0.17
64-bit ripple-carry adder	64	8	13.76	110.08	1.16	128	12.28	1571.84	0.08
16-bit carry-lookahead adder	10	52	4.17	216.84	0.72	104	2.85	296.40	0.35
32-bit carry-lookahead adder	18	52	6.95	361.40	0.72	208	5.39	1121.12	0.19
64-bit carry-lookahead adder	34	52	12.51	650.52	0.72	416	10.47	4355.52	0.10
16-bit carry-select adder	8	93	2.78	258.54	0.72	134	2.68	359.12	0.37
32-bit carry-select adder	14	93	5.56	517.08	0.72	268	5.22	1398.96	0.19
64-bit carry-select adder	26	93	9.73	904.89	0.72	536	10.30	5520.8	0.10
8-bit multiplier	8	64	2.90	185.60	0.69	128	2.98	381.44	0.34
16-bit multiplier	16	128	5.80	742.40	0.69	512	6.21	3179.52	0.16
32-bit multiplier	32	256	11.60	2969.60	0.69	2048	12.05	24678.40	0.08

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

[8] "Nantero," http://www.nantero.com.

[9] S. Hauck *et al.*, "The Chimaera reconfigurable functional unit," *IEEE Trans. VLSI*, vol. 12, pp. 206–217, Feb. 2004.

[10] B. Mei *et al.*, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *Proc. Field-Programmable Logic and Applications*, Aug. 2003, pp. 61–70.

[11] S. Trimberger *et al.*, "A time-multiplexed FPGA," in *Proc. Symp. FPGAs for Custom Computing Machines*, Apr. 1997, pp. 22–28.

[12] S. C. Goldstein *et al.*, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, pp. 70–77, Apr. 2000.

[13] S. Lai, "Current status of the phase change memory and its future," in *Proc. Int. Electron Devices Meeting*, Dec. 2003, pp. 10.1.1–10.1.4.

[14] S. Tehrani *et al.*, "Magnetoresistive random access memory using magnetic tunnel junctions," *Proc. IEEE*, vol. 91, pp. 703–714, May 2003.

[15] G. R. Fox, F. Chu, and T. Davenport, "Current and future ferroelectric non-volatile memory technology," *J. Vaccum Science Technology B.*, vol. 19, pp. 1967–1971, Sept. 2001.

[16] W. Hoenlein, "New prospects for microelectronics: Carbon nanotubes," *The Japan Society of Applied Physics*, vol. 41, pp. 4370–4374, June 2002.

[17] T. Rueckes *et al.*, "Carbon nanotube-based nonvolatile random access memory for molecular computing," *Science*, vol. 289, pp. 94–97, July 2000.

[18] P. J. Burke, "An RF circuit model for carbon nanotubes," *IEEE Trans. Nanotechnology*, vol. 2, pp. 55–58, Mar. 2003.

[19] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proc. Int. Symp. FPGAs*, Feb. 1999, pp. 59–68.

[20] G. Stix, "Nanotubes in the clean room," *Scientific American*, pp. 82–85, Feb. 2005.

[21] J. Rose *et al.*, "Architecture of field-programmable gate arrays," *Proc. IEEE*, vol. 81, pp. 1013–1029, July 1993.

[22] V. Betz and J. Rose, "How much logic should go in an FPGA logic block," *IEEE Design and Test of Computers*, vol. 15, pp. 10–15, Jan.-Mar. 1998.

[23] P. Chow *et al.*, "The design of an SRAM-based field-programmable gate array - Part I: Architecture," *IEEE Trans. VLSI Systems*, vol. 7, pp. 191–197, June 1999.

[24] R. Tessier and H. Giza, "Balancing logic utilization and area efficiency in FPGAs," in *Proc. 10th Int. Wkshp. Field Programmable Logic and Applications*, Aug. 2000, pp. 535–544.