MBARC: A Scalable Memory Based Reconfigurable Computing Framework for Nanoscale Devices

Somnath Paul and Swarup Bhunia.
Department of EECS, Case Western Reserve University.
{sxp190, skb21}@case.edu

***Abstract*— While the emerging nanoscale devices show promises in terms of integration density and computing power, system design with these devices involve some major challenges, such as bottom-up design approach, effective integration with CMOS and defect tolerance. To address some of these challenges, we propose *MBARC*, a reconfigurable framework using memory as the primary computing element. The proposed framework leverages on the reported advantages of memory array design with nanodevices, which are compatible to fabrication into dense and regular structures. The main idea is to partition a logic circuit, implement the partitions as multi-input multi-output lookup tables in a memory array, and then use a simple CMOS-based scheduler to evaluate the partitions in topological time-multiplexed manner. Compared to existing reconfigurable nanocomputing models, the proposed memory based computing has three major advantages: 1) it minimizes the requirement of programmable interconnects, thus, saving design cost; 2) it minimizes the number of CMOS interfacing elements (required for level restoration and cascading logic blocks); 3) existing techniques for defect tolerance in memory array can be easily extended to this framework. Simulation results for a set of ISCAS benchmarks show average improvement of 32% in area, 21% in delay and 34% in energy per vector compared to nanoscale FPGA implementation.**

Index Terms: *Nanoscale Crossbar, Reconfigurable architecture, FPGA, Memory based computing*

I. INTRODUCTION

In the quest of a potential alternative to CMOS at the end of its roadmap [1], multitude of research efforts have been directed towards investigating novel devices with interesting and unique switching characteristics. Examples of such emerging array of devices include single-electron transistors (SET) [22], carbon nanotube field effect transistor (CNTFET) [23], semiconductor nano-wires, quantum-dot cellular automata (QCA) [19] and chemically assembled electronic nanocomputers (CAEN) [7]. Although most of these emerging nanodevices are still in their infancy, they hold tremendous potential in terms of integration density ($\sim 10^{10}$ devices/cm^2), low power operation and higher switching speed. Molecular electronics is one such promising alternative that has drawn significant attention of the researchers in recent years [8]. These nanoscale circuits comprise of a molecular monolayer of rotaxanes sandwiched between metal nanowires [9]. Researchers at HP and at UCLA have met with experimental success in their efforts to realize crossbar structures using these nanoscale circuits either by self-assembly process or by nano-imprinting method [9, 10]. Such experimental success has been complemented with development of architecture [7, 11], circuit [12, 13] and CAD tools [14] to support computation using these molecular crossbars. Substantial research has also been done to develop efficient testing [15] and application mapping procedures [16] that are able to tolerate high defect rate in these self-assembled structures.

The molecular crossbars under consideration provide an attractive solution to configurable computing. The reason is that rotaxane molecules sandwiched between the Ti/Pt nanowires at each crossbar junction can be switched from a state of high resistance to a state of low resistance and vice versa on the application of proper voltages to these nanowires [9]. Thus, each crossbar junction can be thought of as a 1-bit storage element and the entire crossbar can be used for storing the responses of logic functions. The molecular crossbar circuits are highly favorable for the production of dense and regular fabric, which allows the realization of large and complex functionalities within a small area either in the form of Programmable Logic Array (PLA) or as Lookup Table (LUT) [12].

These molecular electronic systems, however, present several design challenges [11]. They are as follows: a) the bistable rotaxane molecules can be considered as two-terminal diode-like devices, which do not provide signal restoration and need to be interfaced with signal restoring circuits before they can be cascaded. It has been proposed [7, 12] that one can use conventional CMOS devices for the purpose of signal restoration. This requires that the nanowires of the crossbar are interfaced with interconnects whose dimensions are of the order of μm. Therefore, it is extremely important to choose a crossbar interface architecture that preserves the high device density offered by the crossbar circuits [12]. b) Since the fabrication process involves patterning at molecular dimensions, variations in the electrical behavior are observed across the crossbar junctions. Some junctions which become permanently irreversible during fabrication are referred to as defective. Design methodologies attempting to use the nanoscale crossbar as a computational fabric should take into account the high defect rate in such devices. Although methods for LUT and PLA-based logic realization in nano-crossbars are well established [7, 11, 12], sufficient investigations have not been reported on how these structures may be cascaded to realize larger functions. One solution as proposed in [12] is to cascade individual LUTs using signal restoration hardware. Such a solution will only benefit if the size of the individual crossbar is much large so as to offset the area and power requirement of the signal restoration circuitry. This is difficult to achieve considering the mismatch in size between the nano-crossbars and CMOS interfacing logic.

The dense and periodic structures of most emerging nanodevices (including the aforementioned molecular switches) as well as bi-stable nature of these switches make them amenable to large memory array design [9, 19, 22-23]. In this paper, we propose a novel computational framework referred as *Memory BAsed Reconfigurable Computing (MBARC)* that exploits the fact that nanodevices can be effectively configured into a memory array. The main idea is to decompose a logic circuit into a set of partitions, implement the partitions as lookup tables in a nanoscale memory array, and then use a CMOS-based controller to evaluate the partitions in topological and time-multiplexed manner. The partitioning and the mapping of the partitions to memory are performed during the application mapping process. The proposed approach separates the nanoscale memory and CMOS logic, therefore minimizing the requirement of interface hardware (a single set of signal restoring circuitry can be used for all the cycles). Further, contrary to existing implementations, where interfacing logic is distributed between any two connected nano-

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

structures (e.g. crossbar), CMOS interfacing logic in *MBARC* is localized and separate, potentially facilitating CMOS-nano hybridization process. Moreover, the wide array of existing techniques to test, diagnose and achieve defect tolerance in memory [2] can be used for the proposed framework that utilizes a memory array as the primary computing element.

In particular, the paper makes the following major contributions:

1. It proposes a scalable reconfigurable memory-based computing model for nanodevices, which are suitable for memory array design. Compared to existing FPGA-like reconfigurable framework for nanodevices (which we refer as *NanoFPGA*), the proposed model can achieve considerable improvement in area, performance and energy per vector.
2. It presents a complete design flow with efficient partitioning and scheduling algorithms for mapping an application to the proposed computational framework.
3. The proposed method minimizes requirements for programmable interconnects and CMOS interfacing hardware.

II. MEMORY BASED COMPUTING METHODOLOGY

Configurable computing systems capitalize on the strengths of both hardware and software by using software algorithms to set the configuration for the programmable hardware. In the proposed computational framework, depicted in Figure 1a, the partitioning of the target application into smaller multi input-output logic functions, subsequent mapping of those functions to memory modules and finally scheduling them is achieved through software intervention. Information regarding the address, schedule and connectivity among the partitions is stored in a smaller memory array (denoted as *schedule table* in Figure 1a) during the phase of application mapping. The smaller logic functions obtained from the partitioning of the target application are mapped to different memory modules, which we collectively refer to as the *function table*. The memory modules are realized using nanoscale devices. Following are the steps for evaluation of a function using *MBARC*.

In *MBARC*, the behavioral description of the function that is to be realized is first synthesized to obtain an optimized multi-input, single-output LUT representation. A partitioning algorithm is then used for *partitioning* the representation into a number of multi-input multi-output logic *partitions*. The number of inputs and outputs to each partition is dictated by the design constraints such as memory requirement and delay. Before evaluation of a function, the functional behavior (the output values corresponding to all input combinations) of each partition is stored in the function table. We define this as the *memory configuration* or the *memory write* phase. Since the functional behavior is loaded into the memory during the write phase, the addresses for the different partitions are known before the actual evaluation of the function.

After the configuration phase, the partitions are accessed in a sequence so that the topological dependence among the partitions is satisfied. In other words, a partition is evaluated only when all its input values are available. When the evaluation of all the partitions corresponding to a given function is completed, it can proceed to evaluate the function for the next input vector. Thus, *MBARC* can substantially reduce the requirement of expensive programmable switching matrices as required in an FPGA fabric [17] while still achieving easy dynamic reconfigurability. As seen in Figure 1a, in an evaluation cycle, the controller communicates with the memory array, providing inputs to and receiving outputs from the partition(s) that is/are being evaluated. The address for the mapping of a particular partition is provided by the schedule table. Figure 1b explains major steps in the proposed memory based computation flow.

2.1) Circuit Partitioning

As described in Figure 1b, an important step towards realizing a complex function with large number of inputs/outputs in memory is to partition the function appropriately to satisfy one or two of the following objectives: 1) to reduce total memory requirement for storing the partitions in the memory in the form of lookup table and 2) to minimize the evaluation time. Thus the problem of partitioning a circuit into multi-input multi-output representation can be formulated as an optimization problem considering evaluation time as optimization objective and memory requirement as a constraint. The constraint on memory is specified as the number of inputs and outputs of a partition. We have developed a heuristic-based solution for the partitioning problem that ensures no cyclic dependency among the partitions. Conventional hypergraph partitioning techniques [18] widely used in VLSI design typically target minimizing the cut-edges between partitions and do not ensure topological order or minimization of

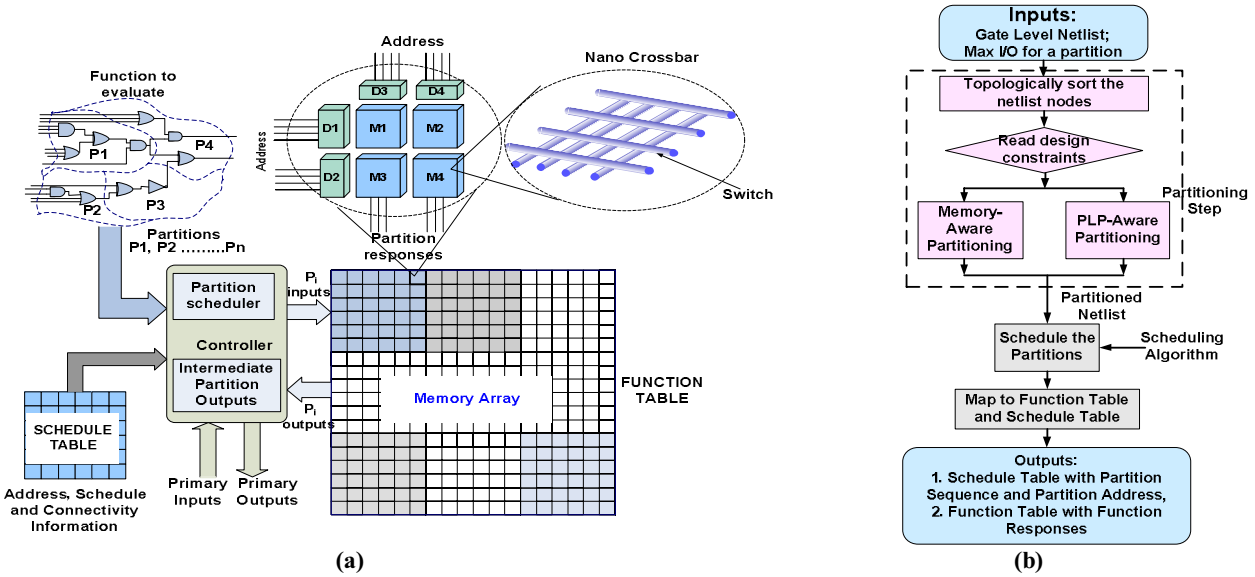

Figure 1: a) Overall memory based computation scheme. A multi input/output logic function to be evaluated is partitioned and the partitions are stored into memory arrays (realized with nanoscale devices using a set of small memory modules). A controller performs the tasks of partition evaluation in topological order and handling of intermediate partition outputs; b) Design flow for *MBARC*.

Figure 2: *Procedure:* *PLP-Aware Partitioning*

INPUT: Circuit netlist, partition size (*M X N*)
OUTPUT: Set of partitions

- *Create hypergraph (G);*
- *Sort vertices topologically;*
- *while vertex v in G not traversed*
  - *Create a new partition P; Include v in P;*
  - *For vertex u in the fanout cone of v*
  - *Include u in P if it satisfies*
    - *topological order*
    - *Size (MXN) limit*
    - *PLP*
  - *Backtrack to include topologically related vertices;*
  - *Complete partition P;*
  - *Mark vertices in P as traversed;*
- *endwhile*
- *Anneal partitions to reduce partition count.*

evaluation time. The pseudo code for the proposed partitioning algorithm is given in Figure 2. It starts with creating a hypergraph from the circuit description and then sorts the vertices in topological order. The sorted vertices are traversed from the primary inputs and considered for inclusion in a partition. A vertex *v* is included in a partition if it satisfies the topological order (among partitions), size limit in terms of number of inputs and outputs of the partitions and *Partition Level Parallelism (PLP)*, which represents the number of independent partitions (i.e. the partitions that can be evaluated in parallel). Since this partitioning algorithm tries to maximize PLP (for improving performance), we refer this as *PLP-aware partitioning*. The fanout cone of vertex *v* included in a partition is traversed to maximize the number of vertices per partition (thus minimizing total number of partitions). If no more vertices can be added to a partition without violating its input/output bounds, the partition is added to the partition pool and vertices in the partition are marked as traversed. Once all the partitions are created, an *annealing* step is performed to reduce the number of partitions. In this step, the partitions are levelized and vertices are shuffled among the partitions of the same level as well as across levels (while maintaining topological dependence). This allows vertices from some partitions to be subsumed inside another partition, thus, resulting in reduced number of partitions. A variant of the above partitioning algorithm was implemented to achieve optimization of memory requirement instead of cycle time. We refer this as *memory-aware partitioning* approach. In the latter approach, during the annealing step, the bigger partitions are broken down to smaller ones to reduce the memory requirement.

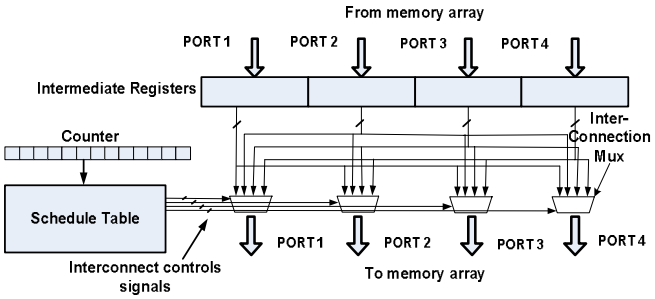

Figure 3: Schematic of the controller hardware that evaluates the partitions in topological order based on static scheduling.

To minimize the impact on evaluation time, we break only those partitions which have little PLP.

2.2) Scheduling the Partitions

For the proposed computational framework, scheduling refers to the order in which the partitions are being evaluated. Since the dependency and the connectivity among the different partitions is predefined during the compilation phase, we refer to the scheduling algorithm as static scheduling. Partition P_{i+1} is dependent on partition P_i if it receives any input from partition P_i. During computation, the controller evaluates the partitions one after another according to the scheduled sequence using the address and connectivity of the partitions stored in the schedule table. Figure 3 shows a schematic of the controller module implementation. The controller interfaces with a memory that has 4 memory banks, each with one read port. As seen in Figure 3, the outputs from the partitions are stored in an intermediate register bank. The register bank stores the partition responses. Depending on the partitions to be evaluated, the select signals for the multiplexer network coming from the schedule table selects the inputs of the partitions from the intermediate register bank. The counter is used to select the schedule table outputs in each clock cycle. Similar to the configurable logic block (CLB) in FPGA, the computational building block for the proposed framework consisting of the schedule table, the function table, intermediate registers and the multiplexer network is, hereafter, referred as *Memory-based Computational Block* or *MCB*.

III. TEST SETUP AND RESULTS

We have validated the proposed computational framework with ISCAS benchmark circuits. Each of the benchmark circuits was first synthesized and technology-mapped using 'RASP' (A FPGA/CPLD Technology mapping and Synthesis Package

Table I: Results for partitioning and scheduling algorithms

ISCAS85 Ckt	Memory Aware Partitioning				PLP Aware Partitioning				
	12 X 12 X 4		12 X 12 X 8		12 X 12 X 4		12 X 12 X 8		
	Mem Req (KB)	Delay (cyc)	Mem Req (KB)	Delay (cyc)	Mem Req (kB)	Delay (cyc)	Mem Req (kB)	Delay (cyc)	Run time (sec)
C432	3.2	9	3.2	7	19.8	7	19.0	5	0.36
C880	19.2	9	11.9	9	33.6	7	33.6	7	1.23
C499	15.5	9	15.4	5	19.8	5	19.8	3	1.24
C1908	18.4	8	14.4	5	32.6	5	32.6	4	3.01
C1355	19.8	7	17.3	5	19.8	5	19.8	3	1.62
C2670	72.9	12	68.8	8	83.1	10	83.1	7	3.19
C3540	61.8	19	52.3	13	79.8	17	79.4	10	8.85
C5315	89.4	19	85.7	10	124.8	18	125.5	11	9.58
C7552	108.2	28	106.2	14	164.1	23	164.1	13	54.05
C6288	78.4	19	62.1	14	78.4	19	78.4	13	18.67

Table II: Design overheads for *MBARC* and NanoFPGA architectures

ISCAS85 Ckt	MBARC Architecture						NanoFPGA Architecture					
	MBARC (12X12X4)			MBARC (12X12X8)								
	Total Area (μm^2)	Delay (Cfg2) (ns)	Energy /Vector (pJ)	Total Area (μm^2)	Delay (Cfg2) (ns)	Energy /Vector (pJ)	# of LUTs in critical path	Total # of CLB used	Total # of CIB used	Total Area (μm^2)	Delay (ns)	Energy/ Vector (pJ)
C432	21168	4.55	39.7	65551	3.4	51.5	13	41	108	35277	13.23	57.16
C880	21733	4.55	45.4	66149	4.76	78.2	7	45	170	45780	7.35	74.31
C499	21168	3.25	28.3	65584	2.04	31.1	5	33	149	53134	5.39	86.2
C1908	21692	3.25	32.1	66108	2.72	44.4	8	33	117	46621	8.33	75.68
C1355	21168	3.25	28.3	65584	2.04	31.1	4	33	152	36817	4.41	59.72
C2670	23760	6.5	94.1	68177	4.76	98.7	4	61	283	867461	4.41	140.82
C3540	23625	11.05	156.6	68025	6.8	138.8	10	116	337	108615	10.29	176.1
C5315	25468	11.7	213.8	69913	7.48	182.7	7	133	551	170640	7.35	276.9
C7552	27078	14.95	326.6	71494	8.84	245.6	7	184	629	213259	7.35	345.4
C6288	23568	12.35	173.5	67984	8.84	179.7	26	275	641	198741	25.97	322.3

developed at UCLA [20]). The gate-level netlist obtained from the synthesis tool contains multiple single output LUTs, which are then grouped into multi-input multi-output partitions according to the partitioning algorithms explained in Section II. The total memory requirement is calculated during the partitioning step on the basis of the number of inputs and outputs to each partition. For example, to evaluate an $N \times M$ partition a total of $(2^N*2N + 2^N*M)$ data points are required in nano-crossbar. The rationale is, 2^N*2N data points are required for implementing the decoder and 2^N*M data points are needed for storing the function responses. The total number of memory accesses required to complete the evaluation of the entire function is obtained from static scheduling.

We have simulated the performance of the proposed computational framework for both Memory and PLP-Aware partitioning algorithms. In each case, the number of inputs and outputs to each partition was restricted to 12. Increasing the number of partitions that may be evaluated in parallel improves the execution time. However, this requires either increasing the number of read ports from a single memory or increasing the number of memory banks, each with one read port that may be accessed in parallel. However, since the partitions communicate among each other only through the intermediate registers, we can distribute the partitions in 4 different memory banks each having a single port (for 4 parallel evaluations). Note that the later configuration is better in terms of design effort and memory access time. Thus hereafter, an *'m'* port configuration refers to *'m'* different memory banks each having a single read port. In order to observe the effect of parallel memory accesses on the proposed framework, results were obtained for both 4 and 8 memory read ports respectively.

Design overhead for MBARC was estimated at 70nm technology node. A behavioral description of the controller was written in Verilog HDL and synthesized using Synopsys Design Compiler. Area, delay and energy per computation for the nano crossbar based FPGA implementation have been estimated using representative values [12]. The number of LUT and Programmable Interconnect Blocks required for a NanoFPGA implementation was estimated by mapping the benchmark circuits to Stratix III FPGA platform using *Altera Quartus Version 7.0*.

3.1) Partitioning and Scheduling Results

Table I lists the simulation results for the proposed computational framework for ISCAS'85 benchmarks. From Table I, it is evident that the memory-aware partitioning algorithm requires less memory compared to the PLP-aware partitioning at the cost of higher number of execution cycles (denoted as delay). Table I also includes the required runtimes for the partitioning procedures on a SunBlade 1500 machine with 2GB RAM.

3.2) Hardware implementation Results

In this section, we present the hardware overhead results incurred for *MBARC* and compare it against a traditional FPGA design scaled to the nano regime [12]. For hardware estimation, we consider two memory configurations with 4 and 8 read ports allowing parallel evaluation of 4 and 8 different partitions, each with 12 inputs and outputs.

i) Area Estimation: The area for the controller module after synthesis using Synopsys Design Compiler at 70nm technology node was obtained as 18917μm^2 and 61893μm^2 for 4 and 8 memory ports, respectively. Since each partition has 12 inputs/outputs, the number of I/O ports and hence the number of interface logic blocks (sense amplifiers and level shifters) required for 4 and 8 port designs are 48 and 96 respectively. An estimate of the area for the level shifter and sense amplifier at 70nm technology as well as the area for nano crossbar with a 70nm nanowire pitch was taken from [12]. The total area was estimated as, *Total Area = Controller Area + Area for Sense Amps + Area for Level Shifters + Area for each crossbar junction*Total Memory Requirement (as given for PLP-aware partitioning in Table I).*

As suggested in [12], the area for the NanoFPGA implementation can be estimated by taking into consideration the area overhead for i) the configurable logic blocks (CLB) and the configurable interconnect blocks (CIB) (implemented using nano crossbars) and ii) CMOS interface logic. Table II shows the number of CLBs and CIBs required for implementing the benchmark circuits. Each CLB is realized using an 8 input 1 output

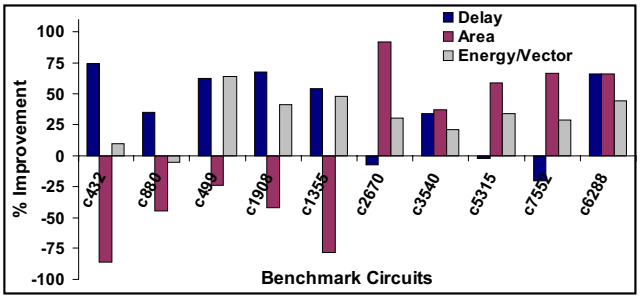

Figure 4: Percentage improvement in design overhead for 8 ports compared to FPGA-based implementation.

Table III: Design overhead for thread level parallelism

Benchmark Ckts	MBARC (12X12X4)				NanoFPGA		
	# of threads	Delay (ns)	Area (μm^2)	Energy /Vector (pJ)	Delay (ns)	Area (μm^2)	Energy /Vector (pJ)
C5315	7	6.5	149615	320.8	7.35	170640	276.9
S13207	8	10.4	178329	752.6	17.9	265678	432.6
S15850	8	9.1	182421	741.0	13.1	389274	630.6

LUT and each CIB is realized using an 8 input 8 output LUT. Thus, the total area for the NanoFPGA can be estimated as: *Total area = (Crossbar area for 8 input LUT + sense amp + level shifter area)*(no of CLB used) + (crossbar area for CIB + sense amp + level shifter area)*(no of CIB used)*. Table II presents the design overhead results for the different benchmarks for both *MBARC* and the NanoFPGA implementations.

ii) Delay Estimation: We estimate the cycle time for each computation and the *Total execution time* is estimated as: *one cycle time* number of cycles required for the benchmark (considering the PLP-aware partitioning).* The time required for each cycle can be estimated as: *memory access and read time + controller delay.* We have used the memory access and read time of 490ps according to the value reported in [12]. It can be noted that the total memory access time is dominated by the interface logic delay [12]. Therefore, we assume that the total delay for the crossbar memory access remains almost constant even for larger crossbar array. The time required for the controller operation at 70nm technology node is obtained to be 156.8ps using Synopsys Design Compiler. Thus the total cycle time is estimated to be 650ps for 4-port configuration. For an 8-port configuration, a rise in the controller delay (due to more complex multiplexers in address generation logic) increases the cycle time to 680ps. The total delay for NanoFPGA can be estimated as: *Delay for nanocrossbar*(# of CIB in critical path) + Delay for nanocrossbar*(# of CLB in critical path).* The total number of configurable logic blocks and programmable block interconnects in the critical path is obtained from Quartus v7.0 by Altera.

iii) Energy/Vector Estimation: Since for each input vector, the proposed framework involves multi-cycle operation, the total energy/vector for *MBARC* can be estimated as: *{(Controller power/Cycle time) + Energy/cycle for all sense amps and level shifters + Energy/cycle for memory access}*(No of cycles required to evaluate the vector).* The power contributed by controller circuit is estimated with Design Compiler. Estimates for energy/cycle for the memory and interface logic were obtained from [12]. The same parameters can be calculated for NanoFPGA as: *(Energy/Cycle for CIB)*No of CIB + (Energy/Cycle for CLB)*No of CLB + Energy/Cycle for all sense amps and level shifters associated with CLB and CIB.* The energy/vector results for the ISCAS'85 benchmark circuits are presented in Table II.

Figure 5: *Procedure:* *Threading*

INPUT: i) Hypergraph representation of the circuit,
ii) Maximum number of Primary Inputs (PI) per thread (P)
OUTPUT: Independent threads of execution

1. *Find the Logic Cone (LC) for each Primary Output (PO);*
2. *Order the LCs in the ascending order of PI counts;*
3. *IF, for a primary output, the number of PIs is more than P, then consider the LC for the output as a single thread;*
4. *ELSE, group the primary outputs with maximum correlation in terms of LC in a single thread such that the number of PIs for the group is less than P;*
5. *For the threads which have overlapping logic cones, duplicate the shared logic in both threads.*

Figure 4 compares the percentage improvement in area, power and delay for the different benchmark circuits for both *MBARC* and NanoFPGA based design. From the results presented, for different benchmarks we note that on an average *MBARC* allows **60%** and **5%** savings in area for 4 and 8 port configurations, respectively compared to a NanoFPGA design. Comparing the percentage improvement in delays; we observe that for a given vector the proposed framework allows **6.6%** and **36.3%** average savings in the total evaluation time. Finally, in terms of energy/vector, *MBARC* requires **36.5%** and **31.6%** less energy/vector on an average compared to NanoFPGA.

IV. DESIGN CONSIDERATIONS

In this section, we will discuss some important design issues associated with *MBARC*.

4.1) Scalability

Since an MCB evaluates logic blocks in time-multiplexed manner (unlike spatial computing), theoretically applications of arbitrary complexity can be mapped to a single MCB if we allow sufficient memory to store the partition responses and enough evaluation cycles. However, in order to improve performance (number of cycles required per vector), the proposed *MBARC* framework can be extended to exploit parallelism among partitions. Further, multiple MCBs can also be connected in pipelined fashion to improve the throughput in multi-vector scenarios.

i) Exploiting Thread Level Parallelism: An investigation in standard circuits reveals that each output of the circuit is not dependent on all the inputs. In other words, the logic for most of the applications is bit-sliced, so that the computations for the different output bits may be performed in parallel to one another. A partitioning algorithm that considers such thread level parallelism in the circuit behavior will therefore be able to reduce the number of partitions in the critical path for each output and thus reduce the total evaluation time. Such a benefit will be more visible for sequential circuits where i) a set of flip flops share the same input cone and ii) the intersection of the input cone among different such sets is quite small so that the logic that accounts for the intersection can be duplicated. A software routine, given in Figure 5 was written that would analyze the circuit to be implemented and

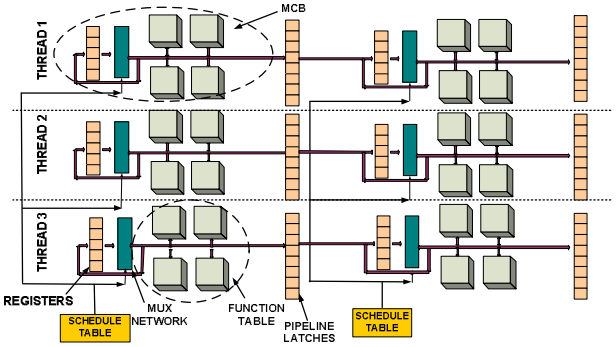

Figure 6: Architectural framework for realization of thread level parallelism and pipelining.

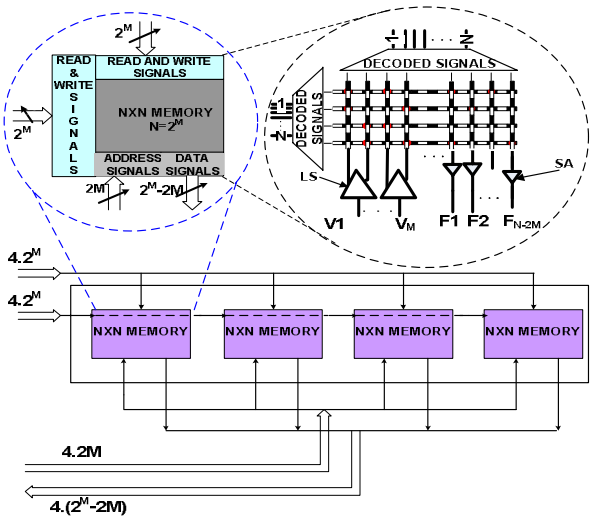

Figure 7: Organization of a large memory module from 4 individual NXN memory blocks.

identify the parallel threads in the design prior to the partitioning procedure so that the PLP or memory-aware partitioning algorithms can operate on these individual threads and generate the revised set of partitions. The hardware implementation of parallel thread evaluation is envisioned in Figure 6, where the partitions to be evaluated is scheduled in parallel MCBs operating independent of each other. The concept was validated for c5315 (ISCAS'85), s13207 and s15850 (ISCAS'89) benchmark circuits for a 4 port memory configuration. Table III lists the design overhead and the performance improvement for parallel thread execution. From the results we note that thread level parallel evaluation on the proposed computational framework allows a **27.9%** improvement in the total execution time and **32.7%** improvement in area at the cost of **25.3%** increase in the total energy/vector. The improvement in performance is particularly noteworthy for c5315 which requires 7.48 ns even for an 8 port memory configuration.

ii) Pipelining: Since the proposed framework inherently supports multi-cycle evaluation, each MCB can be pipelined with another to allow pipelined design for improved throughput. The concept is illustrated in Figure 6. Thus the proposed framework allows the following different granularities of computation: a) A design can be evaluated in a single MCB b) A design can be evaluated in a single pipelined thread or c) A design may be decomposed into several parallel threads that may or may not involve pipelining. Hence, by allowing both time and space multiplexing of its resources, the proposed *hybrid* reconfigurable platform provides significant improvement in the evaluation time for both irregular control as well as regular datapath functions [17].

4.2) Memory Array Implementations with Nanodevices

For the proposed computational framework, it is essential that we have a large memory array to store the LUTs for partitions. A general feature of the emerging nanodevices (such as SET, QCA, CAEN etc.) is that their typical dense and periodic structures are amenable to large memory design. However, since these nanodevices are inherently susceptible to high defect rates and parameter variations, a monolithic large defect-free memory block design may be challenging for these devices. Besides, a large memory block typically requires large access time. Thus, it is desirable to distribute the partitions into smaller memory modules.

In particular, let us consider the case of large memory design with nanoscale crossbar using molecular switches. Figure 7 illustrates the organization of a larger memory array from several smaller nano-crossbars.

Figure 7 realizes a memory of $4N$ words (with word-size = N) from four $N\times N$ memory modules., where $N=2^M$. This implies that the LUT that is realized using this $N\times N$ crossbar can accommodate a total of N minterms corresponding to M ($M=log_2N$) number of inputs. Since both a signal and its complement are required to access the LUT values [12], $2M$ number of vertical lines will be used as inputs to the LUT (Figure 7). Therefore, the maximum number of outputs of a partition that the LUT can support is $N-2M$ or $N-2log_2N$. Note that the level shifters (LS) and the sense amplifiers (SA) are required for interfacing with the crossbar.

V. CONCLUSIONS

We have presented *MBARC*, a reconfigurable memory-based computing model for emerging nanoscale devices, which are amenable for memory design. The proposed model provides dynamic reconfigurability and minimizes the requirement for programmable interconnects as well as interfacing hardware. Though the fundamental computational block for *MBARC* (referred as MCB) evaluates a logic function in a time-multiplexed fashion, the framework can be easily scaled to exploit parallel execution of multiple partitions and threads (for higher throughput) using multiple memory banks or multiple MCBs. Our investigation shows that compared to purely spatial computing framework such as FPGA, the proposed reconfigurable model can render significant saving in area, performance and power. Finally, unlike the previous approaches, the proposed model can potentially make the CMOS-nano integration easier by separating the CMOS interface logic from the memory blocks realized with nanodevices.

REFERENCES

[1] ITRS 2005: Process Integration, Devices and Structures, Available online at: *http://www.itrs.net/Links/2005ITRS/PIDS2005.pdf*.
[2] M. L. Bushnell and V. D. Agrawal, "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits", *Springer*, 2000.
[3] L. Carloni et al., "Theory of latency-insensitive design", *IEEE TCAD*, 2001.
[4] M. Tehranipoor, "Defect tolerance for molecular electronics-based nanofabrics using built-in self-test procedure", *DFT,* 2005.
[5] A. Dehon et al., "Seven strategies for tolerating highly defective fabrication", *IEEE Design & Test of Computers*, 2005, pp: 306-315.
[6] M. Mishra and S.C. Goldstein, "Defect Tolerance at the End of the Roadmap", *ITC,* 2003, pp: 1201-1211.
[7] S.C. Goldstein et al., "NanoFabrics: Spatial Computing Using Molecular Electronics", *ISCA,* 2001.
[8] R. F. Service, "Molecules get wired", *Science*, vol. 294, 2001.
[9] Yong Chen et al., "Nanoscale molecular-switch crossbar circuits", *Nanotechnology* 14, pp. 462-468, 2003.
[10] C. P. Collier et al., "Electronically configurable molecular-based logic gates", *Science*, vol. 285, pp 391-394, 1999.
[11] A. Dehon et al., **"**Hybrid CMOS/nanoelectronic digital circuits: devices, architectures, and design automation", *ICCAD*, 2005.
[12] M. M. Ziegler and M. R. Stan, **"**CMOS/Nano Co-Design for Crossbar-Based Molecular Electronic System", *IEEE Trans. on Nanotech.* 2003.
[13] M. M. Ziegler and M. R. Stan, "Design and Analysis of crossbar circuits for molecular nanoelectronics", *IEEE Nano*, pp. 323-327, 2002.
[14] P. Farm et al., "Nanoeda: architecture and design methodology for nano-scale electronic systems", *Swedish SoC Conf*., 2003.
[15] J. G. Brown et al., "CAEN-BIST: testing the nanofabric", *Proceedings of ITC*, 2004, pp: 462-471.
[16] M. B. Tahoori, "A mapping algorithm for defect-tolerance of reconfigurable nano-architectures", *ICCAD* 2005, pp: 668-672.
[17] S. Hauck, "The roles of FPGAs in reprogrammable systems", *Proceedings of the IEEE*, 1998, pp. 615– 638.
[18] G. Karypis et al., "Multilevel hypergraph partitioning: applications in VLSI domain", *IEEE TVLSI*, 1999, pp: 69-79.
[19] M. Ottavi et al., "Design of a QCA Memory with Parallel Read/Serial Write", *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2005.
[20] http://cadlab.cs.ucla.edu/~xfpga/software/raspsyn.htm.0324
[21] G. Snider et al., "CMOS-like logic in defective, nanoscale crossbars", *Nanotechnology* 15, pp. 881-891, 2004.