A Scalable Memory-Based Reconfigurable Computing Framework for Nanoscale Crossbar

Somnath Paul, Student Member, IEEE and Swarup Bhunia, Senior Member, IEEE

Abstract—Nanoscale molecular electronic devices amenable to bottom-up self-assembly into crossbar structure have emerged as a promising candidate for future electronic systems. To address some of the design challenges in molecular crossbar, we propose “MBARC”, where memory, instead of switch based logic functions, is used as the computing element. MBARC leverages on the fact that regular and periodic structures of molecular crossbar are attractive for dense memory design. The main idea in MBARC is to partition a logic circuit, store 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 (PI); 2) it minimizes the number of CMOS interfacing elements which are required for level restoration and cascading logic blocks; 3) it can achieve higher defect tolerance through efficient use of redundancy. 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. Effectiveness of the framework is also studied for two large sequential circuits, namely 2-D discrete cosine transform and 8-tap FIR filter.

Index Terms—Nanoscale Crossbar, Reconfigurable architecture, Field Programmable Gate Array, 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) [2], carbon nanotube field effect transistor (CNTFET) [3], semiconductor nano-wires, quantum-dot cellular automata (QCA) [4] and chemically assembled electronic nanocomputers (CAEN) [5]. Although most of these emerging nanodevices are still in their infancy, they hold tremendous potential in terms of integration density (in the order of $10^{11}$ devices/cm$^2$ [21]), low power operation and higher switching speed. Molecular device is one such promising alternative that has drawn significant attention of the researchers in recent years [11]. These nanoscale devices comprise of a molecular monolayer of rotaxanes sandwiched between metal nanowires [12]. Researchers have achieved experimental success in their efforts to realize crossbar structures using these nanoscale circuits either by self-assembly process or by nano-imprinting method [12 - 13]. Recently, researchers have reported successful fabrication of 160 Kb of molecular memory patterned at 11$^{11}$ bits per sq. cm [22]. Such experimental success has been complemented with development of device models [23], circuit [6, 15], architecture [5, 14], and CAD tools [16] to support computation using these molecular crossbars. Substantial research has also been done to develop efficient testing [17] and application mapping procedures [18] that can tolerate high defect rate in these self-assembled structures.

Somnath Paul and Swarup Bhunia are with the Dept. of EECS, Case Western Reserve University, Cleveland, OH, USA. 
email: {sxp190, ssh21}@case.edu
Copyright (c) 2009 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

Rotaxane molecules sandwiched between the Ti/Pt nanowires at each molecular 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 [12]. Thus, each crossbar junction can be considered as an 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) [6].

These molecular electronic systems, however, present several design challenges [14]. They are as follows. a) The bistable rotaxane molecules do not provide signal restoration and need to be interfaced with signal restoring circuits before they can be cascaded. It has been proposed in [5 - 6] 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 $\mu$m. Therefore, it is extremely important to choose a crossbar interface architecture judiciously that preserves the high device density offered by the crossbar circuits [6]. b) Some junctions which become permanently irreversible during fabrication are considered defective. Design methodologies attempting to use the nanoscale crossbar should take into account the high defect rate in such devices.

As outlined in [21], in order to address the above challenges, any scalable molecular electronic system should have the following desirable characteristics:

- **Crossbars computing units**: Meshes formed by overlapping wires do not demand precise alignment of end-to-end connections between nanowires, which would otherwise be impossible to achieve in the molecular self-assembly process.
- **High defect tolerance**: A molecular electronic system should be tolerant to high defect densities arising from the random self-assembly process.
- **Sparse interaction between microscale and nanoscale logic**: Due to substantial difference in size that exists between the micro-scale and the nanoscale worlds, a desirable framework should minimize the CMOS/Nano interaction as much as possible.

In this paper, we propose a novel computational framework referred as “Memory Based Architecture for Reconfigurable Computing” (MBARC). The main idea is to decompose a logic circuit into a set of partitions, implement the partitions as LUTs 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 mapping of the partitions to memory are performed during the application mapping process. The localized and separate CMOS interfacing logic in MBARC, potentially facilitates CMOS-nano hybridization process. Moreover, the wide array of existing techniques to test, diagnose and achieve defect tolerance in memory [29] can be extended to the proposed framework.

In particular, the paper makes the following major contributions:

1) **It proposes a scalable reconfigurable memory-based computing**
model for molecular nanodevices. Compared to existing FPGA-like reconfigurable framework, MBARC can achieve considerable improvement in area, performance and energy per vector. The proposed computing framework minimizes requirements for PI and CMOS interfacing hardware.

2) It presents a complete design flow with efficient partitioning and scheduling algorithms for mapping an application to the proposed computational framework.

3) It provides circuit level implementation details for realizing larger memory array using smaller nanoscale crossbar circuits.

4) It also presents an comparative analysis of defect tolerance between nanoFPGA and MBARC.

5) It analyzes the effectiveness of MBARC with respect to technological scalability and reliability of operation. These aspects are compared with nanoFPGA.

The rest of the paper is organized as follows. Section II provides a background on the existing architectures for molecular devices and presents a motivation for memory based computation. Section III provides an overview of the proposed memory based computing framework. Section IV discusses the simulation setup and results for a set of benchmark circuits. It also highlights the improvement in technological scalability and defect tolerance achieved through the proposed reconfigurable framework. Section V explains the design considerations for the proposed reconfigurable platform including circuit level approaches for large memory array design using molecular crossbar. We conclude in Section VI.

II. BACKGROUND AND MOTIVATION

A. Existing architectures for molecular devices

A number of spatial reconfigurable computing frameworks similar to conventional FPGA have been proposed for molecular devices [5, 6, 14]. These frameworks essentially consist of two types of elements: i) nanoBlock which comprises of nanoscale molecular crossbars configured as LUT or PLA and serves to hold the functionality of the mapped application and ii) switchBlock which also consists of nanoscale crossbar and serves as the reconfigurable interconnect. In [5], it was proposed that the signal restoration in such a spatial computing framework would be achieved by using Negative Differential Resistor (NDR) based latches. A close study of these latches [21, 24] however reveals the following difficulties: i) A higher energy-delay product compared to a conventional CMOS latch. ii) NDR latch is a very structured circuit and hence difficult to incorporate using the current nanofabrication techniques. iii) Finally, it has poor drive strength and low noise margins.

In order to avoid these strict requirements of NDR latch for nanoscale operations, authors [6, 21, 25, 26] have advocated using molecular diodes for logic and routing, while the task of signal restoration have been left to conventional CMOS logic. However, the spatial computing model of nanoFPGA is only beneficial if the size of the individual crossbar is very large such that it offsets the area and power requirement of the signal restoration circuitry. Such as CMOS/Nano co-design approach is referred to NanoFPGA.

Another variant of CMOS/Nano reconfigurable framework known as “CMOL” was proposed in [26]. A CMOL FPGA essentially employs diode resistor logic to build fundamental logic gates, which are then routed using nanoscale crossbars. However, it demands a CMOS inverter at each circuit node for signal restoration. CMOL and its variant FPNI [37] can reliably integrate nanoscale molecular devices with CMOS gates without requiring alignment accuracy. A cell-based CMOL FPGA design as proposed in [27] however restricts the CMOL architecture to only molecular diodes with rectifying property, which are more difficult to fabricate compared to molecular devices with non-linear I-V characteristics [36]. A recent study conducted in [38] suggests that instead of molecular diodes, devices with non-linear I-V characteristics are more amenable for realizing scalable crossbar memories. We have therefore selected nanoFPGA as the baseline architecture for comparing the proposed computing model. The primary reason behind such a choice is that nanoFPGA model also employs crossbar memories to realize LUTs and is independent of the underlying device characteristic.

Fig. 1. a) Overall memory based computation scheme. A multi input/output logic function to be evaluated is partitioned and the partitions are stored in a memory array, 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.

B. Motivation

As discussed above, majority of the existing approaches face serious challenges such as: i) requirement to hybridize with CMOS, ii) reliability and yield and iii) implementation difficulty of cascaded irregular logic structures. The above challenges suggest that transforming molecular crossbar to nanoelectronics requires a rethinking of the computational framework. An alternative to the purely spatial computing framework such as FPGA, is a time-multiplexed Memory Based Computing model that uses dense 2-D memory array to map large multi-input multi-output LUTs [9, 28]. 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 [27]. Since time-multiplexed reconfigurable architectures improve the overhead for the programmable interconnects [9], it is worthwhile to investigate the potential of 2-D molecular memories as the primary computing element in the above reconfigurable architecture.
III. MEMORY BASED COMPUTING METHODOLOGY

A. MBARC: The computing model

In the proposed computational framework, illustrated in Fig. 1(a), the partitioning of the target application into smaller multi input, multi 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 Fig. 1(a)) 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.

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 blocks. The number of inputs and outputs to each block is dictated by the design constraints such as memory requirement and delay. Each partition is represented in terms of its output space, i.e. output response for all possible input combinations. Thus, an X-input Y-output partition requires $2^X \times Y$ bits for storing the partition response. Before evaluation of a function, the functional behavior (i.e. 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 or configuration phase.

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. As illustrated in Fig. 1(a), the intermediate partition outputs are stored in a CMOS register file. 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, the controller and the intermediate registers is hereafter referred to as Memory-based Computational Block or MCB. As shown in Fig. 1(a), in an evaluation cycle, the controller communicates with the memory array, providing inputs to and receiving outputs from a partition that is being evaluated. The address for the mapping of a particular partition is provided by the schedule table. Fig. 1(b) explains major steps in the proposed memory based computation flow.

The generic memory based computing model described above applies well to dense nanoscale memory technologies. Since nanoscale crossbar based molecular systems are amenable to dense memory design, function table used in the MBARC model can be built using molecular crossbars. The fabrication of the function table is expected to follow the same guidelines for realizing a molecular embedded memory as illustrated in [12, 22, 31]. In order to obtain a fair comparison with the NanoFPGA model [6], we assume that Level Shifters (LS) and Sense Amplifiers (SA) would be required for interfacing the CMOS controller with the nanoscale memory arrays inside the function table. Due to its smaller size, the schedule table is realized using conventional CMOS-based Static Random Access Memory (SRAM).

B. Circuit Partitioning

Partitioning of the input circuit satisfies one or both of the following objectives: 1) reduce total memory requirement for storing the partitions in the memory in the form of LUT and 2) minimize the evaluation time. Thus the problem of partitioning a circuit can be formulated as an optimization problem considering evaluation time as optimization objective and memory requirement as a constraint. We have developed a heuristic-based solution that ensures no cyclic dependency among the partitions. Conventional hypergraph partitioning techniques [10] which are widely used in VLSI design, typically target minimizing the cut-edges between partitions. They do not ensure topological order and do not target minimization of evaluation time or minimization of memory requirement for representing the partitions in the form of LUTs.

The pseudo code for the proposed heuristic is given in Algorithm 1. 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.

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. To minimize the impact on evaluation time, we break only those partitions which have little PLP. The controller along with the function table is implemented as a two-stage pipeline. Partition inputs, which form the effective memory address, are selected from the register file in the first stage. The second stage of the pipeline consists of memory read operation followed by the register file write. Fig. 2(b) shows the timing diagram for a single cycle of MCB operation.
C. Scheduling the Partitions

Since the dependency and the connectivity among the different partitions are predefined during the partitioning 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. Fig. 2(a) 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 Fig. 2(a), 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 select the inputs of the partitions from the intermediate register bank. The counter is used to select the schedule table outputs in each clock cycle.

IV. SIMULATION SETUP AND RESULTS

To check the effectiveness of the proposed computational framework for molecular crossbar devices, we implemented the partitioning and scheduling algorithms in “C” programming language and performed simulations with a set of ISCAS benchmark circuits. Each of the benchmark circuits was first synthesized and technology-mapped using ‘RASP’ (a FPGA/CPLD Technology mapping and Synthesis Package developed at UCLA [20]). 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 \((2^N + 2N + 2^N + 2M)\) data points are required in nano-crossbar. The rationale is, \(2^N\) \& 2N data points are required for implementing the decoder and \(2^N+2M\) 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 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. 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). In order to observe the effect of parallel memory accesses on the proposed framework, results were obtained for two configurations with 4 and 8 memory read ports, respectively. Design overhead for MBARC was estimated at 70nm technology node. A behavioral description of the CMOS controller module was written in Verilog HDL and synthesized using Synopsys Design Compiler. Area, delay and energy per computation for the nano crossbar based FPGA implementation are obtained from [6]. The number of LUT and Programmable Interconnect Blocks required for a NanoFPGA implementation was estimated by mapping the benchmark circuits to the Stratix III FPGA platform using Altera Quartus Version 7.0 [7] mapping tool.

Algorithm 1 PLP Aware Partitioning

1: Input: Circuit Netlist, partition size \((M \times N)\)
2: Output: Set of partitions
3: Create hypergraph \((G)\)
4: Sort vertices topologically
5: while vertex \(v\) is not traversed do
6: Create a new partition \(P\)
7: Include \(v\) in \(P\)
8: for vertex \(u\) in the fanout cone of \(v\) do
9: if \(u\) satisfies topological order & Size \((M \times N)\) limit & PLP then
10: Include \(u\) in \(P\)
11: end if
12: end for
13: Backtrack to include topologically related vertices
14: Complete partition \(P\)
15: Mark vertices in \(P\) as traversed
16: end while
17: Anneal partitions to reduce partition count

A. 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 under a moderate workload.

B. Hardware Implementation Results

1) Area Estimation: The area for the controller module after synthesis using Synopsys Design Compiler at 70nm technology node was obtained as \(18917\mu m^2\) and \(61893\mu m^2\) for 4 and 8 memory ports, 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 [6]. The total area was estimated as,
### TABLE II

**Comparison of Design overheads for MBARC and NanoFPGA architectures**

<table>
<thead>
<tr>
<th>Ckts</th>
<th>MBARC (12 x 12 x 4)</th>
<th>NanoFPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total Area (µm²)</td>
<td>Delay (ns)</td>
</tr>
<tr>
<td>C432</td>
<td>21168</td>
<td>4.55</td>
</tr>
<tr>
<td>C499</td>
<td>21168</td>
<td>3.25</td>
</tr>
<tr>
<td>C800</td>
<td>21733</td>
<td>4.35</td>
</tr>
<tr>
<td>C1355</td>
<td>21168</td>
<td>3.25</td>
</tr>
<tr>
<td>C1998</td>
<td>21092</td>
<td>3.25</td>
</tr>
<tr>
<td>C2670</td>
<td>23760</td>
<td>6.5</td>
</tr>
<tr>
<td>C3540</td>
<td>23625</td>
<td>11.05</td>
</tr>
<tr>
<td>C3540</td>
<td>23625</td>
<td>11.05</td>
</tr>
<tr>
<td>C6288</td>
<td>23568</td>
<td>12.35</td>
</tr>
<tr>
<td>C7552</td>
<td>27078</td>
<td>14.95</td>
</tr>
</tbody>
</table>

Fig. 3. a) Comparison of design parameters between 4 and 8-port configurations of MCB. b) Percentage improvement in design overhead for 8 ports compared to NanoFPGA based implementation.

**Total Area = Controller Area + Area for Sense Amps + Area for Level Shifters + Area for each crossbar junction \times Total Memory Requirement**

The total memory requirement for PLP-aware partitioning is provided in Table II. As suggested in [6], 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 LUT and each CIB is realized using an 8-input 8-output LUT. Thus, the total area for the NanoFPGA can be estimated as:

\[
\text{Total area} = (\text{Crossbar area for } 8\text{-input LUT } + \text{sense amp } + \text{level shifter area}) \times (\# \text{ of CLB used}) + (\text{crossbar area for } CIB + \text{sense amp } + \text{level shifter area}) \times (\# \text{ of CIB used}).
\]

Table II presents the design overhead results for the different benchmarks for a 4-port MBARC and the NanoFPGA implementation.

2) **Delay Estimation:** We estimate the cycle time for each computation and the total execution time is estimated as: cycle time \times # of cycles required for a 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 490 ps according to the value reported in [6]. The time required for the controller operation at 70nm technology node is obtained to be 156.8 ps using Synopsys Design Compiler. Thus the total cycle time is estimated to be 650 ps for 4-port configuration. For an 8-port configuration, a rise in the controller delay increases the cycle time to 680 ps. The total delay for NanoFPGA can be estimated as:

\[
\text{Delay for nanocrossbar} \times (\# \text{ of CIB in critical path}) + \text{Delay for nanocrossbar} \times (\# \text{ of CLB in critical path}).
\]

The total number of configurable logic blocks and programmable block interconnects in the critical path is obtained from Quartus v7.0 tool by Altera [7].

3) **Energy/Vector Estimation:** Since for each input vector, the proposed framework involves multi-cycle evaluation, the total energy/-vector for MBARC can be estimated as:

\[
(\text{Controller power } \times \text{Cycle time}) + (\text{Energy/cycle for all sense amps and level shifters } + \text{Energy/cycle for memory access}) \times (\# \text{ of cycles required to evaluate the vector}).
\]

The power contributed by controller circuit is estimated with Design Compiler. Estimates for energy/cycle for the memory and interface logic were obtained from [6]. The same parameters can be calculated for NanoFPGA as:

\[
(\text{Energy/for CIB}) \times (\# \text{ of CIB}) + (\text{Energy/Cycle for CLB}) \times (\# \text{ of CLB}) + \text{Energy/Cycle for all sense amps and level shifters associated with CLB and CIB}.
\]

The energy/vector results for the ISCAS’85 benchmark circuits are presented in Table II. For a 4-port configuration, we note that on an average MBARC improves the area, delay and energy/vector by 60%, 6.6% and 36.5%, respectively compared to a NanoFPGA framework.

Fig. 3(a) compares the average design overhead for an 8-port configuration normalized with the corresponding values from a 4-port MBARC model. Fig. 3(a) supports the fact that a larger number of ports at each MCB leads to parallel execution of more partitions, thereby improving the total execution time. Fig. 3(b) shows the percentage improvement in design overheads for an 8-port MBARC model. From Fig. 3(b), we observe that on an average the 8-port configuration improves the area, delay and energy/vector by 5%, 36.3% and 31.6%, respectively compared to a NanoFPGA framework. It is important to note here that for 8-port configuration, the MBARC model incurs a larger area overhead for the smaller benchmark circuits. The reason as evident from Fig. 3(a) is the 3X increase in MCB area going from 4-port to the 8-port configuration.
C. Results for larger benchmarks

We have investigated the effectiveness of the MBARC framework for two large benchmark applications namely discrete cosine transform (DCT) and finite impulse response (FIR) filtering. The DCT benchmark operates on 8 input pixels, each pixel being encoded into 8 bits. Each of the 8 outputs from the DCT module are encoded into 16 bits. The 8-tap FIR filter receives 12-bit inputs and the output is 27 bits wide. Both the benchmarks were partitioned into $12 \times 12$ partitions and mapped to MCBs with 8 ports. Fig. 4 shows the two benchmarks mapped to the nanoFPGA and the MBARC frameworks. The illustrations for mapping to the MBARC framework were obtained by interfacing our mapping tool with the VPR toolset [39]. The results for mapping to the nanoFPGA framework were obtained directly from the VPR toolset with a 8-input LUT architecture. As evident from Fig.4, localized memory based computation in MBARC leads to significant reduction in the number of computing elements as well as programmable interconnects. Table III shows that for DCT, MBARC improves the delay, area and energy/vector by 70%, 54% and 5% respectively over a nanoFPGA based implementation. For the FIR benchmark, the delay and area are improved by 43% and 23% respectively.

D. Scalability and Reliability Results

1) Technological Scalability: For a spatial computing framework such as FPGA, typically more than 70% of the area for the mapped design is contributed by the reconfigurable interconnect structures [32]. This not only affects the delay and power, but also the technological scalability of performance for a specific design. MBARC reduces programmable interconnects drastically by using multi-cycle execution within each MCB. This allows ASIC-like scalability across technology generations. As shown in Fig. 5(a), MBARC provides significantly better performance scalability compared to the nanoFPGA model. It is due to the fact that the critical timing in MBARC is dominated by logic delay and memory access time instead of delay through programmable interconnects.

2) Reliability: For nano-crossbar based architectures, defect tolerance is primarily achieved by insertion of redundant rows and columns. Our analysis shows that MBARC offers better reliability compared to nanoFPGA at equal area overhead. Fig.5(b) compares the reliability between MBARC and nanoFPGA for ISCAS benchmark circuit c1908. Due to the spatial nature of the nanoFPGA model, the interfacing elements for these LUTs cannot be shared across other computing elements. As a result, under a given area constraint, the number of redundant rows/columns that may be inserted into the nanoFPGA framework is much less. However, due to localized multi-cycle execution in MBARC, partitions can be re-mapped from one memory block to another without increasing the area for CMOS interfacing elements which are shared across all the memory blocks in a given MCB.

Fig.6 shows possible defect tolerance approaches in case of both MBARC and NanoFPGA. MBARC is more suitable for the insertion of the redundant rows and columns in memory due to the following reasons:

- For lower defect rates, it is desired that the number of redundant rows/columns in the crossbar is less. Since each element of the nanoFPGA has a fixed number of inputs and outputs ($N \times 1$ LUT and $M \times M$ switch-blocks in Fig. 6(a)), the overhead of adding redundant rows/columns in every LUT or switch-block can be very high. However, in the case of the MBARC model, the same partition can span multiple memory blocks (as shown in Fig. 6(b)). This effectively leads to sharing of the redundant rows across memory blocks and increases the number of partitions that can be mapped to the MBARC framework.

- For higher defect rates, the nanoFPGA LUTs must be smaller with a higher number of redundant rows and columns per LUT. But for smaller LUTs in nanoFPGAs, the CMOS overhead per useful bit is much higher due to the contribution from the interfacing elements. On the contrary, in case of MBARC, the memory blocks can be made smaller without increasing the number of interfacing elements.

In order to compare the defect tolerance of the nanoFPGA and

![Fig. 4](image-url)

![Fig. 5](image-url)
TABLE IV

<table>
<thead>
<tr>
<th>Benchmark circuits</th>
<th>MBARC (12 × 12 × 4)</th>
<th>NanoFPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of threads</td>
<td>Delay (ns)</td>
</tr>
<tr>
<td>C5315</td>
<td>7</td>
<td>6.5</td>
</tr>
<tr>
<td>S13207</td>
<td>8</td>
<td>10.4</td>
</tr>
<tr>
<td>S15850</td>
<td>8</td>
<td>9.1</td>
</tr>
</tbody>
</table>

Fig. 6. Defect tolerance schemes for a) nanoFPGA and b) MBARC models.

the MBARC framework, we have considered mapping the c3540 benchmark to both the frameworks. For each computing block of the nanoFPGA which supports partitions up to 8 × 1, the row (r) and column redundancy (c) was varied for different junction failure (p) values. Fig. 7(a) shows that even with r = 8, c = 8 almost all the partitions remain unmapped for randomly inserted defects with p = 0.01. Fig. 7(b) shows that a MBARC framework improves the defect tolerance significantly (∼ 50%) for both low and high defect scenarios. In Fig. 7(b), a 64 × 8 memory block was used for mapping the partitions.

However, it is to be noted that in Fig. 7(a) and (b), that for high defect scenarios (p = 0.01), increasing the number of redundant rows for a given column redundancy bring negligible improvement in the total number of mapped partitions. The reason is increasing the number of redundant elements also increases the total number of defects in the crossbar. Hence, it is important to determine the optimal memory block size which can address defect tolerance over a large range of values for p. This is illustrated in Fig. 7(c) which shows that a block size of 32 × 8 can improve the defect tolerance further by ∼ 5% over a 64 × 8 memory block configuration.

E. Performance over a range of device parameter values

Here we analyze the performance of MBARC over a wide range of area, delay and energy parameter values for the molecular crossbar device and compare it against the nanoFPGA model. From Fig. 8(a), we note that as the area for each crossbar junction decreases, the average area improvement over all the benchmarks decreases. When the crossbar junction occupies 2× the area reported in [6], MBARC model requires 9% less area compared to nanoFPGA. However, as the junction area is scaled to 0.5× the reported value, the MBARC model requires 8.9% more area compared the nanoFPGA framework. From Fig. 8(b), we note that as both the delay and the energy are decreased from 2× to 0.5× the nominal value reported in [6], the average EDP over all the benchmarks improves from 41.57% to 79.71%. Fig. 8(c) studies the impact of device area for a benchmark circuit c7552. As the crossbar junction area is varied from 4× to 0.25× the nominal area, the area improvement for MBARC over nanoFPGA gradually reduces. Fig. 8(d) shows the variation in EDP improvement as the read delay and the read energy for the molecular crossbars are varied independently from 4× to 0.25× the nominal value. From Fig. 8(a-d), we can observe that MBARC provides notable advantage over nanoFPGA in terms of high level design parameters such as delay and power for wide range of crossbar parameter values.
Fig. 8. Variation in a) area improvement with molecular crossbar junction area. b) EDP improvement with both read delay and read energy varying from $2\times$ to $0.5\times$ of the nominal value. c) area improvement for c7552 with crossbar junction area varying from $4\times$ to $0.25\times$ of the nominal value. d) EDP improvement for c7552 as the read delay and read energy for the molecular crossbar are independently varied from $4\times$ to $0.25\times$ of the nominal value.

V. DESIGN CONSIDERATIONS

A. Multi-MCB Communication Architecture

Exploiting Thread Level Parallelism: An investigation in several benchmark circuits reveals that the logic for many applications can be bit-sliced, so that the computations for the different output bits may be performed in parallel to one another. A circuit decomposition 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. Algorithm 2 was written to identify the parallel threads in the design prior to partitioning. The hardware implementation of parallel thread evaluation is illustrated in Fig. 9, 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 (sequential ISCAS’89 benchmark) circuits for a 4-port memory configuration. Table IV lists the design overhead and the performance improvement for parallel thread execution. From the results in Table IV 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.

B. 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 Fig. 9. 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 is capable of providing significant improvement in the evaluation time for both irregular control as well as regular datapath functions [30].

C. Memory Array Implementations with Nanodevices

In Fig. 10, we illustrate the organization of larger memories from smaller crossbar circuits and the procedure of writing to and reading from such large arrays. Let us consider an $N \times N$ memory module as shown in Fig. 10, where $N = 2^M$. As demonstrated in [38], for molecular devices with non-linear current-voltage characteristics, $N \sim 100$ presents an optimal design choice with reliable readout voltage margin. This implies that the LUT that is realized using

Algorithm 2 Threading

1: Input: Hypergraph representation of the circuit, Maximum number of Primary Inputs (PI) per thread (P)
2: Output: Independent threads of execution
3: Find the Logic Cone (LC) for each Primary Output (PO)
4: Order the LCs in ascending order of PI counts
5: If For a PO, the number of PIs is more than P then
6: Consider LC for the output as a single thread
7: Else
8: 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
9: end if
10: For any two threads with overlapping LC do
11: Duplicate the shared logic in both threads
12: End for

Fig. 9. Architectural framework for realization of thread level parallelism and pipelining.

Fig. 10. Large memory array implementation using smaller $N \times N$ molecular crossbar circuits.
this $N \times N$ crossbar can accommodate a total of $N^2$ minterms corresponding to $\log_2 N = M$ number of inputs. Since both a signal and its complement are required to access the LUT values, $2M$ number of vertical lines will be required as inputs to the LUT as shown in Fig. 10. Therefore, the maximum number of outputs that the LUT can support is $N - 2M$ or $N - 2\log_2 N$. The level shifter and the sense amplifiers are required for interfacing with the crossbar.

Fig. 10 shows the basic organization of a large memory array using smaller $N \times N$ crossbar circuits. A larger module is assumed to consist of $4N \times N$ crossbar circuits. The scaling factor of 4 is in keeping with current $F_{CMOS}/F_{nano}$ values as reported by ITRS [1]. The total number of input and output signals to each $4N \times N$ module is $4 \times 2M$ and $4(2^M - 2M)$, respectively. The idea is to have a single copy of the interfacing logic for each bank and use the same for writing to or reading from the different memory modules. This essentially requires a demultiplexer which use either CMOS switches or novel demultiplexer designs [33]. The trade-off involved in these two approaches are as follows:

- Although a conventional MOS switch offers higher performance compared to a molecular switch [34], it occupies larger area owing to the large pitch mismatch between the micro and nanoscale wires.
- The demultiplexer design as proposed in [33] is amenable to fabrication using self assembly. However, it requires more than $\log_2 n$ number of address lines for addressing $n$ output lines.

Due to its scalability and ease of fabrication, we incorporate the latter demultiplexer design to achieve a robust CMOS/Nano interfacing in the MBARC framework.

Fig. 11(a-b) shows the write and read procedures for a larger memory organization. Note that the values on the input signals in Fig. 11(a-b) conform with the boolean values to be applied to the nanoscale crossbar for the write and read operations as reported in [6, 12, 38]. As illustrated in Fig. 11(a), during the read operation, all the $2^M$ horizontal crossbars in a $N \times N$ block are required to be at $V_{high} \sim 0.5V$, while, the $2^M - 2M$ vertical crossbars are required to be at $V_{low} \sim 0V$ [12]. Output from the level shifters drive the remaining $2M$ vertical crossbars while output from $2^M - 2M$ vertical crossbars serve as inputs to the sense amplifiers. During the write operation $V_{high} \sim 3.5V$, while $V_{low} \sim 1.75V$ [12]. Output from the sense amplifiers are don’t cares during write operation.

VI. CONCLUSION

We have presented MBARC, a reconfigurable memory-based computing model for emerging nanoscale molecular crossbar devices, which are amenable to dense memory array design. 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 framework provides significant saving in area and performance due to large reduction in programmable interconnect and interfacing hardware. Unlike the previous approaches, the proposed model can potentially make the CMOS-nano integration easier by separating the CMOS interface logic from the nanoscale memory. We have also presented analysis of technology scalability and reliability of operation under manufacturing defects. It is observed that MBARC compares favorably with nanoFPGA on both aspects.

REFERENCES

[7] [Online]: http://www.altera.com/products/devices/stratix3t3s Index.jsp
A. Gayasen et al., “Exploring Technology Alternatives for Nano-Scale Pro- 
S. Hauck, “The Roles of FPGA's in Reprogrammable Systems”, 
K.K. Likharev and D.B. Strukov, “CMOL: Devices, Circuits, and Ar-
D. Jones and D.M. Lewis, “A time-multiplexed FPGA architecture for 
S.C. Goldstein and D. Rosewater, “Digital Logic Using Molecular Elec-
M. Ziegler et al, “A universal device model for nanoelectronic circuit 
J.E. Green et al, “A 160-kilobit molecular electronic memory patterned 
G.S. Snider and R.S. Williams, “Nano/CMOS architectures using a field-
“RASP”, “A FPGA/CPLD Technology mapping and Synthesis Package” 
G. Csaba and P. Lugli, “Read-Out Design Rules for Molecular Crossbar 

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.