ENFIRE: A Spatio-Temporal Fine-Grained Reconfigurable Hardware

Wenchao Qian, *Student Member, IEEE*, Christopher Babecki, *Student Member, IEEE*, Robert Karam, *Student Member, IEEE*, Somnath Paul, *Member, IEEE*, and Swarup Bhunia, *Senior Member, IEEE*

***Abstract*— Field programmable gate arrays (FPGAs) are well-established as fine-grained reconfigurable computing platforms. However, FPGAs demonstrate poor scalability in advanced technology nodes due to the large negative impact of the elaborate programmable interconnects (PIs). The need for such vast PIs arises from two key factors: 1) fine-grained bit-level data manipulation in the configurable logic blocks and 2) the purely spatial computing model followed in the FPGAs. In this paper, we propose ENFIRE, a novel memory-based spatio-temporal framework designed to provide the flexibility of reconfigurable bit-level information processing while improving scalability and energy efficiency. Dense 2-D memory arrays serve as the main computing elements storing not only the data to be processed but also the functional behavior of the application mapped into lookup tables. Computing elements are spatially distributed, communicating as needed over a hierarchical bus interconnect, while the functions are evaluated temporally inside each computing element. A custom software framework facilitates application mapping to the framework. By leveraging both spatial and temporal computing, ENFIRE significantly reduces the interconnect overhead when compared with FPGA. Simulation results show an improvement of 7.6× in energy, 1.6× in energy efficiency, 1.1× in leakage, and 5.3× in unified energy efficiency, a metric that considers energy and area together, compared with comparable FPGA implementations.**

***Index Terms*— Energy efficiency, field programmable gate arrays (FPGA), fine-grain reconfigurable hardware, memory-based computing (MBC), spatio-temporal computing.**

I. Introduction

THERE is an inherent tradeoff between flexibility and efficiency in a digital system design, as the implementations range from general purpose processors (GPPs) to application specific integrated circuits (ASICs) at opposite ends of the spectrum. Microprocessors have high flexibility, but relatively low performance, whereas the ASICs provide higher performance, lower area, and greater energy efficiency, but as they are highly optimized and application specific, they lack the flexibility of the GPPs. Reconfigurable computing has emerged to bridge the gap between these two extremes. Such frameworks can generally be categorized into fine-grained and coarse-grained platforms. For fine-grained computing with postsilicon reconfigurability, the field programmable gate arrays (FPGAs) have become increasingly attractive, balancing the application mapping flexibility of the GPPs with the higher performance of the ASICs, while lowering costs and enabling more rapid and less risky development compared with the ASICs [1].

For the past decade, improving FPGA area and latency have been among the major challenges in FPGA design. However, for sub90-nm technology nodes, power has become an increasingly important consideration [2]. In order to reduce FPGA power consumption, manufacturers and researchers have investigated the device level techniques, such as low-K dielectric, a dual-V_{th} process, and triple-oxide approach, and the circuit level techniques, such as decreased logic granularity and clock/power gating [3]. For certain technology nodes, these techniques reduce power and energy effectively, but they cannot keep pace with transistor scaling in the future.

Traditionally, FPGA architectures employ a purely spatial computing model. Applications are mapped into a set of multiple-input, single-output lookup tables (LUTs) connected by the programmable interconnects (PIs) [4]. However, this requires a rather elaborate PI network, which becomes a major performance bottleneck and leads to poor power, performance, and scalability across technology nodes, where the PI network alone has been shown to account for an 80% of power consumption and a 40%–80% of critical path delay [5]. Furthermore, because of the dominance of the PI network, FPGA platforms experience poor performance scalability across the technology nodes. As a result, there is a growing need for alternative reconfigurable computing frameworks, which can reduce the PI requirement, improving energy efficiency and technology scalability over the conventional architectures.

In order to improve area and performance, researchers have investigated using an FPGA's embedded memory arrays for computation, when they are not configured as on-chip memory [6], [7]. Time-multiplexed hardware reconfigurable schemes have also been investigated to increase the hardware utilization, and therefore save area and performance [8], [9]. However, when executing a specific application, they are

Manuscript received December 20, 2015; revised March 31, 2016; accepted May 18, 2016. Date of publication June 24, 2016; date of current version December 26, 2016. This work was supported by Semiconductor Research Corporation under Grant 2015-EP-2650.

W. Qian was with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106 USA. He is now with Xilinx Inc., San Jose, CA 95124 USA (e-mail: wxq18@case.edu).

C. Babecki and S. Paul are with Intel Corporation, Hillsboro, OR 97124 USA (e-mail: cmb135@case.edu; sxp190@case.edu).

R. Karam and S. Bhunia are with the University of Florida, Gainesville, FL 32611 USA (e-mail: robert.karam@case.edu; skb21@case.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2016.2578933

still considered as a fully spatial computing model, which is similar to the traditional FPGAs, and hence incur a large PI overhead. Spatio-temporal reconfigurable hardware schemes have also been proposed to improve the utilization of both PI and computing elements [4]. Instead of using an arithmetic logic unit for multicycle execution as done by PipeRench [10] MAHA [11] uses a memory-based computing (MBC) scheme to improve energy efficiency. However, these frameworks operate with a granularity of at least 8 bits, and therefore suffer from poor resource efficiency and low energy efficiency when executing bit-level applications. In addition, the Garp architecture [12] was proposed to embed a reconfigurable FPGA-like architecture within a CPU; however, this is essentially a temporal computing element (CPU) using a spatial fabric (FPGA) to accelerate some computations, so function evaluation is not strictly spatio-temporal. Also related is the Tabula architecture [13], which still performs spatial data processing, though it time-multiplexes the FPGA resources to increase the apparent number of LUTs from the user's perspective.

As an alternative, we propose an ENergy-efficient FIne-grain spatio-temporal REconfigurable ENFIRE framework, which can be described as a configurable array of the memory-based computing blocks (MCBs). Each computing element, referred to as a memory logic block (MLB), is suitable for temporal computing. It is capable of fine-grained bit-level function evaluation using a combination of a register file, which allows bit-level access, with the ability to store multiple LUT responses in the embedded 2-D memory array. A function is evaluated in an MLB by accessing the memory over multiple cycles. Each MLB contains a schedule table, a dense 2-D memory array, and a small controller. The schedule table stores the instructions, and a minimally modified 2-D memory array stores not only the data to be processed but also the function responses of the mapped application.

Execution with an MLB is performed temporally, while the MLBs are distributed spatially and communicate through a hierarchical bus interconnect. Thus, ENFIRE operates in a spatio-temporal manner, which can be optimized to provide high energy efficiency for a given application. This represents a paradigm shift toward more balanced energy-efficient computing that can serve as either a stand-alone FPGA replacement or as an on-die integrated solution within a CPU leading to increased flexibility, dynamic reprogrammability, and efficient use of the die area for a variety of workloads [14], [15], including fine-grained and energy-efficient FPGA-like logic functions. In addition, we have developed a custom application mapping tool to efficiently map applications into the ENFIRE framework. A set of random logic benchmarks of varying complexities has been successfully mapped using this tool and functionally verified for correctness.

In particular, the paper makes the following novel contributions.

1) It proposes ENFIRE, a novel MBC framework, which uses a 2-D memory array hybridized with CMOS controlling logic. Unlike previous work in MBC, ENFIRE enables energy-efficient mapping of FPGA-like fine-grained logic that can be used stand-alone as an FPGA replacement, or embedded within a GPP for energy-efficient acceleration of fine-grained workloads. Many applications, including analytics, stream processing, and information encoding/decoding can largely benefit from efficient fine-grained data processing, which is not available in the conventional coarse-grain reconfigurable architectures [4].
2) It provides the design details of the proposed architecture including the μ-architecture of the MLBs. It also describes in detail the hierarchal interconnect model between the MLBs.
3) It presents a custom-designed complete software application mapping flow, describing the major steps of the software framework, and also presents several examples of the application mapping.
4) It describes the modeling process of the hardware components, shows the simulation setup, and presents the simulation results. It then demonstrates that the ENFIRE framework can achieve considerable improvement in energy efficiency compared with an FPGA implementation at the cost of increased latency.

The rest of the paper is organized as follows: Section II provides background on MBC and the motivation for this paper; Section III describes the overall hardware architecture of ENFIRE; Section IV presents the software architecture and the algorithms used in application mapping; Section V presents the ENFIRE application mapping flow and hardware model, as well as the FPGA mapping setup; Section VI presents the mapping results; finally, we conclude in Section VII with future directions for this paper.

II. Background and Motivation

In this section, we provide an overview of the earlier works aimed at improving the energy efficiency of the FPGAs and an overview of the MBC framework. We also describe the motivation for developing the ENFIRE framework.

A. Overview of Memory-Based Computing

Improving energy efficiency in the FPGAs has been heavily researched in recent years. In one approach, different resource organizations, such as linear or systolic arrays, were chosen for specific applications given the performance and resource limitations [16]. Improving resource selection was also investigated to reduce interconnect multiplexer requirement, thereby reducing power consumption [17]. Finally, supply voltage programmability has been investigated to reduce interconnect power [18].

MBC also provides a solution to reduce PI overhead [11]. Fig. 1(a) shows the MBC application mapping operation: a target data flow graph (DFG) is decomposed into smaller normalized nodes; those sharing many connections are fused into partitions, realized as the multi-input, multi-output logic blocks. Next, the fused partitions are mapped as the LUTs to the embedded memory array of one or more MLBs, and finally scheduled for evaluation over multiple cycles [11]. During application mapping, information, such as address, scheduling, and connectivity among the partitions, is stored in

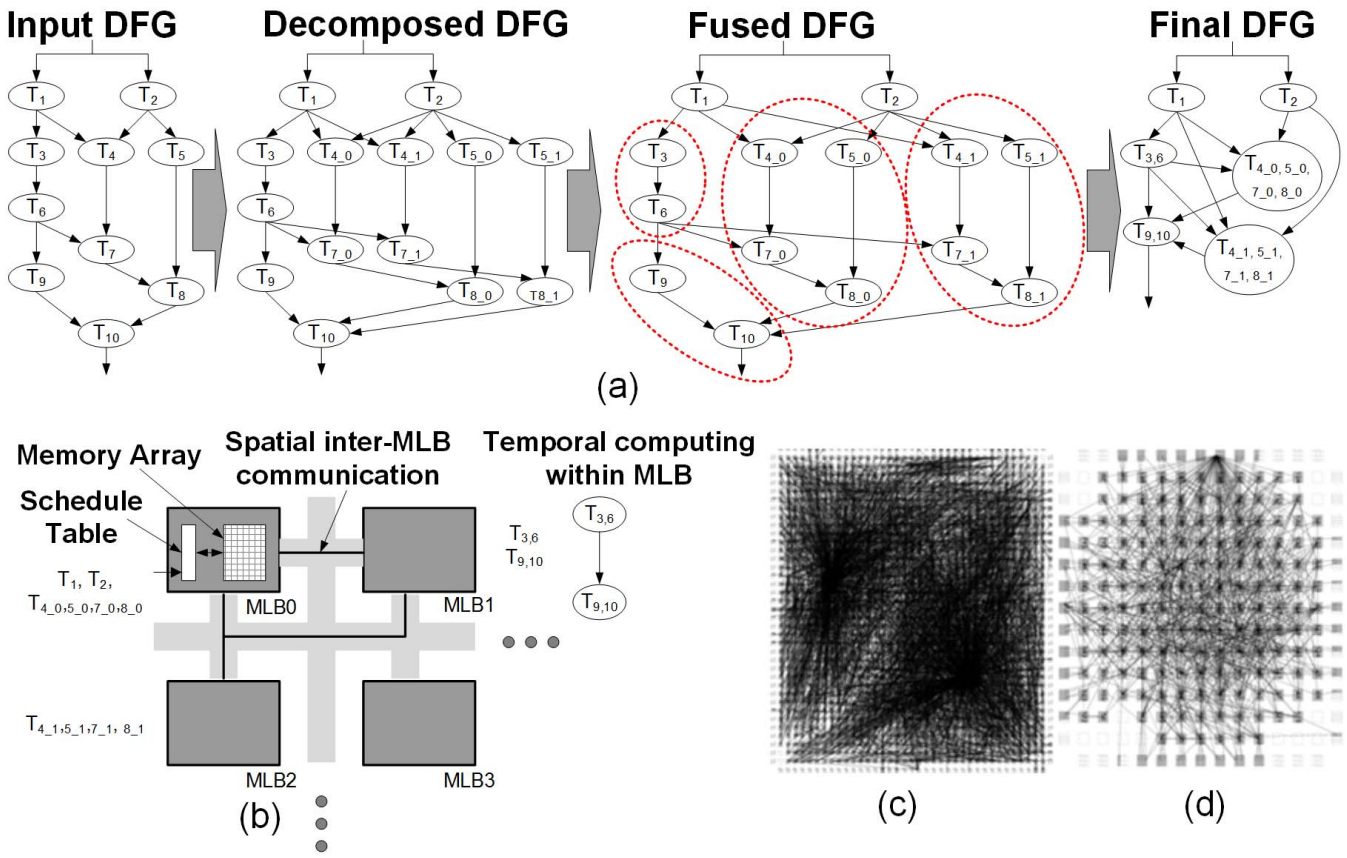

Fig. 1. (a) Decomposition and fusion of input DFG in MBC. (b) MLB structure and data communication in computation. (c) and (d) Improvement in number of computing elements and PI requirement (estimated with VPR [20]) for ISCAS-89 sequential benchmark s38417 when mapped to the 65-nm CMOS FPGA framework and 65-nm MBC [21].

a register bank [schedule table in Fig. 1(b)] that behaves as a microcode controller in each computing block. The embedded memory array, in which the logic partitions are mapped, is referred to as the function table. A bank of flip-flops stores the intermediate partition outputs, to be used in the following evaluation cycles. Together, they form the core of the computational building block. Multiple MCBs are connected through a configurable interconnect framework similar to that in the conventional FPGAs; however, the larger partition size and the temporal execution of partitions inside a single processing element (PE) greatly reduces the PI requirement [Fig. 1(c) and (d)]. In this manner, the proposed framework can achieve significant improvements in energy efficiency over a conventional FPGA framework [19].

B. Motivation

Traditional FPGA frameworks use a fully spatial computing model, enabling high flexibility at the expense of increased area, power consumption, and scalability. Unlike logic circuits, which scale well at new technology nodes, the FPGA PI network does not. Therefore, a reconfigurable framework that can implement the fine-grained functions while simultaneously reducing the interconnect requirement can be viewed as a potential FPGA replacement for the applications, where energy efficiency is crucial.

The proposed MBC architecture is a good candidate for these applications because of its low PI requirement (Section II-A). Energy consumption from data movement through PIs can be minimized with local computation by preferentially mapping frequently communicating functions as multiple LUTs in the same MLB. Moreover, MBC computation relies on the memory accesses; optimizing memory access energy and latency can have a significant positive impact on the framework. As a result, an energy-efficient hardware architecture and an optimized mapping software codesign approach are required.

In particular, the proposed ENFIRE architecture differs from the FPGAs in the following ways.

1) FPGA architecture is fully spatial, while ENFIRE is spatio-temporal. This serves to minimize the PI requirement by using temporal execution within each MLB.
2) FPGAs map applications into small 1-D LUTs, but ENFIRE uses the dense 2-D memory arrays to hold not only the data but also multi-input, multi-output LUTs for computation.
3) FPGAs use a fully spatial PI structure, but communication between the MLBs in ENFIRE uses a time-multiplexed hierarchal bus, greatly reducing interconnect complexity.

In addition to the random logic benchmarks used in this paper, we note several other applications, which can benefit

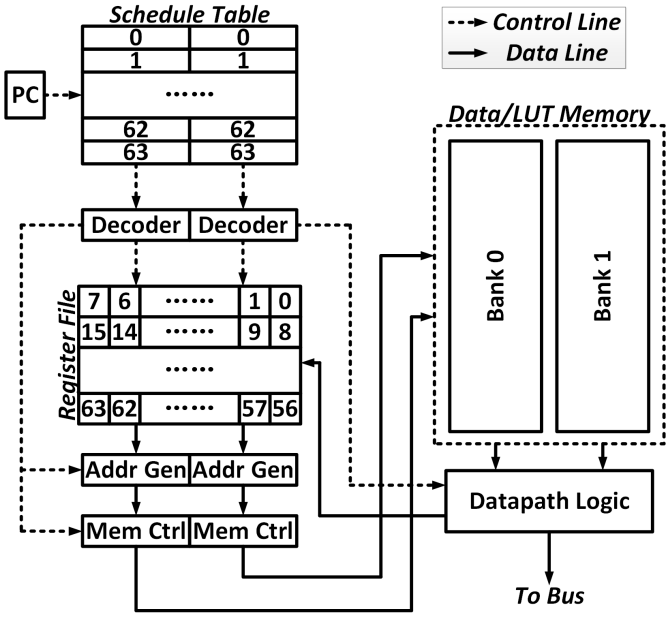

Fig. 2. Block diagram of an MLB.

from an accelerated bit-level access, including various stream ciphers, the Monte Carlo methods [22], pseudorandom number generators such as Mersenne Twister [23], and applications using bitmaps for a compressed data representation.

III. Hardware Architecture

The block diagram of a single ENFIRE PE is shown in Fig. 2. Each PE is referred to as an MLB and operates independently. Each contains its own schedule (instruction) table and data memory array, which holds LUT responses in addition to the data for processing. The MLBs are connected with a two-level hierarchical bus interconnect. The first level, referred to as a cluster, contains four MLBs, while the second, called a tile, contains four clusters. In the remainder of this section, we describe in detail the μ-architecture of a single MLB along with the interconnect structure and the communication patterns between the MLBs.

A. MLB Structure

Fig. 2 shows an MLB block diagram. Each MLB consists of the following components.

1) *Program Counter* tracks the current instruction being executed.
2) *Schedule Table* memory array, which holds the instructions for the given application.
3) *Decoder* responsible for decoding the instructions fetched from the schedule table.
4) *Register File* holds the intermediate results from the application during execution.
5) *Address Generation and Memory Controller* generates the memory access request and corresponding memory address for LUT operations.
6) *Data Memory* large 2-D memory array, which holds the LUTs, as well as the data being processed.
7) *Datapath Logic* controlled by the decoded instruction and determines the output destination.

The MLB is more analogous to the complex logic block (CLB) structure in an FPGA [24] than to a standard processor. The primary difference between an MLB and a CLB is that the MLB stores multiple LUTs, which can be dynamically selected at runtime, as opposed to a single, fixed configuration. This allows for a spatio-temporal computing model that enables more efficient resource reuse than a CLB.

Since the random logic functions tend to have a high level of data parallelism, the MLBs have two separate execution engines to improve throughput. Since these applications also do not typically have a high level of branching, the parallel operation can be statically scheduled in a Very Long Instruction Word (VLIW)-2 manner or two instructions per cycle. To achieve this, each schedule table entry holds two independent MLB instructions, which are decoded and executed in parallel in two separate datapaths. Two memory banks in the data memory are accessed separately with each execution engine to perform the LUT operations in a given cycle.

Each LUT is a function of eight inputs and can have an output width of 1-, 2-, 4-, or 8-bits. The choices of such input and output counts are made after analyzing the application mapping (using the described software mapping flow), instruction encoding complexity, register file size, and memory size. Larger input and output counts yield a lower number of operations overall, which is good for execution latency, but it results in a higher instruction encoding complexity, because each instruction will need to specify more bit locations. It also increases the bits written into the register file at a cycle as well as the size of each LUT. We found the 8-bit input and 1-, 2-, 4-, and 8-bit output widths yield an optimal tradeoff among the considered factors. The inputs come from the register file, and are designated in the instruction as eight unique 6-bit addresses. The instruction also encodes which LUT is to be used in the operation. The proposed MLB μ-architecture studied in this paper allows for eight different LUTs of each size (8×1, 8×2, 8×4, and 8×8) for a total of 32 possible LUTs per MLB. The LUT/data memory is designed to support the accesses of these different widths. To reduce the complexity of the address generation logic, each LUT is packed into separate column(s) of the data memory. The banks are also sized, so that the number of rows is equal to the number of entries required for each LUT ($2^8 = 256$ rows). This allows the input to the LUT to be directly mapped to the row address sent to the memory during the read request. Selection of the desired LUT response can then be accomplished by selecting the appropriate bank and the desired column(s) of the memory to read.

Our design uses the 2-kB memory banks (256 rows $\times$ 64 bits/row). Rather than read an entire row to perform a small memory access, wordline segmentation is used, which ensures that only the minimum number of required cells are energized during the read. AND gates are inserted in the path of the wordline to ensure that the SELECT signal is received only by the desired cells during a read operation. This scheme, shown in Fig. 3, allows for four of each sized LUT per memory bank. Because of a limitation in instruction encoding, there is a

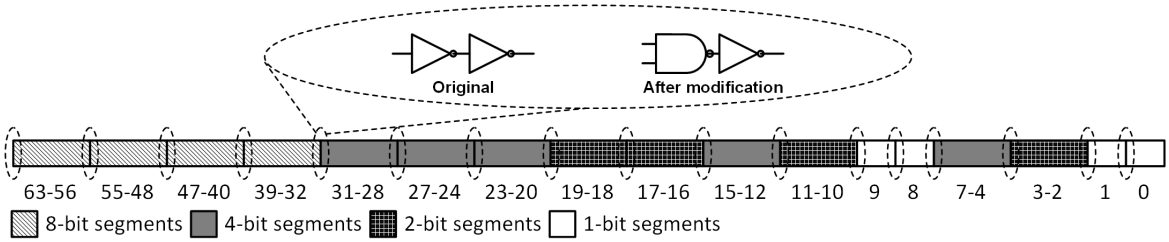

Fig. 3. Wordline segmentation of a data row in the memory bank.

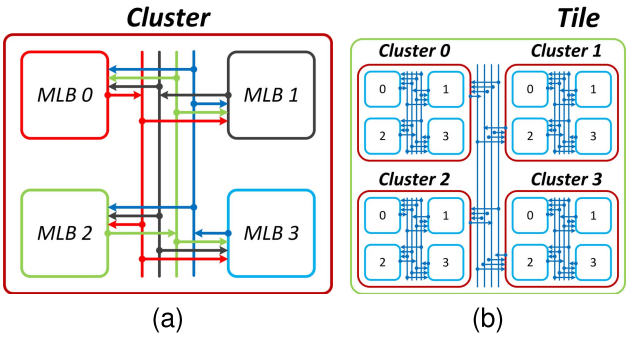

Fig. 4. Hierarchical bus-based interconnect structure of (a) single cluster of four MLBs and (b) single tile of four clusters.

fifth 4-bit wide segment that cannot be used as an LUT, but can be used for general data storage if needed.

Though the register file has 16 unique read ports (eight input bits per instruction, and two instructions per cycle), there are only two write ports (8-bit wide), given the increased complexity of a write port. Therefore, after the LUT response has been read, it must be aligned to the appropriate bits in the register file. This enables a more streamlined instruction encoding, requiring only one write address instead of eight. Writing to each bit within the 8-bit port is individually controlled by an enable signal, so not all the eight bits need to be written in the case of 8×1, 8×2, or 8×4 LUT operations.

B. Interconnect Structure

The statically scheduled random logic functions targeted by ENFIRE eliminate the need for a packet-switched interconnect scheme and the associated power and area overhead for routers. Routing information can be efficiently encoded as a part of the instructions and can be decoded at runtime to enable the inter-MLB communication as needed.

Most random logic functions exhibit strong spatial locality of data. To exploit this, ENFIRE utilizes a sparse hierarchical bus interconnect to enable low-overhead communication for separate MLBs. Within each cluster, an 8-bit bus fully connects all the MLBs [Fig. 4(a)]. This yields an available bandwidth of 10.26 GB/s at the maximum operation frequency of 1.3 GHz (Section V-A). Between the clusters, there is a similar 16-bit bus, 4-bits of which are reserved for each MLB in the source cluster [Fig. 4(b)]. When an MLB places data on its intercluster bus, the data are broadcast to all other clusters in the tile.

Communications are initiated either implicitly as a part of an LUT operation or explicitly through an additional MOVE instruction. Explicit communication can be conducted between any MLBs in a tile and requires the transferring MLB to initiate the request with a MOVE instruction. Once the data are available on the bus, any other MLB may read the data on the following cycle by issuing its own MOVE operation.

LUT operations can also output part, or all, of the LUT response to the bus, in addition to storing the result to the register file, by setting a flag in the instruction. If set, up to four of the bits output from the LUT will be automatically written to the intracluster bus. Data may also be read from the intracluster bus implicitly using virtual register ports, which connect the upper 3×8 bits of the register file with the 3×8 bits of the intracluster bus coming from the other MLBs in the cluster. When a read request is issued to one of these registers, the data from the bus are used instead of the current register contents. This type of communication is only available at the intracluster level.

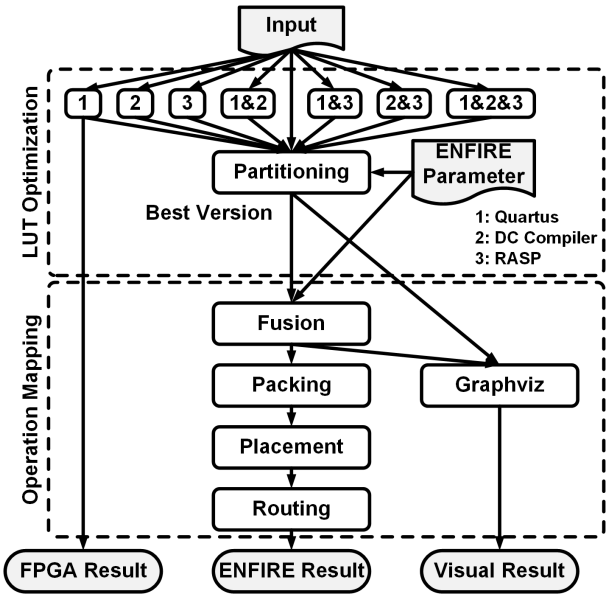

Fig. 5. Overview of ENFIRE software architecture.

IV. Software Architecture

An effective and efficient software mapping flow must be designed to enable the users to map applications into the corresponding hardware framework. We have developed a comprehensive software flow capable of mapping applications to the proposed ENFIRE framework. Fig. 5 shows an overview of the mapping flow, including LUT optimization and

operation mapping. The tool itself is developed in C and has been verified using a set of applications of varying complexity. In the verification process, any intermediate mapping results can be converted to Verilog for functional simulation. The tool has also been interfaced with Graphviz [26] for visually analyzing and debugging the outputs from each intermediate step.

A. LUT Optimization

Using input in the form of a Berkeley logic interchange format (BLIF) [27] file, the tool aims to map applications into 8×1, 8×2, 8×4, and 8×8 LUTs (Section III). Several programs have been used to optimize the LUT response, including Quartus [28], Synopsys Design Compiler [29], Rasp [30], and combinations thereof. These programs map the application into $K \times 1$ LUTs, where K is a number less than 8. Outputs from individual programs and combinations of these programs, along with the original input, are passed to the partitioning process. During this stage, the tool attempts to distribute the input bits on multiple $K \times 1$ LUTs and cluster them into larger 8×1, 8×2, 8×4, and 8×8 LUTs.

A DFG in the form of a directed acyclic graph can be used to represent the set of $K \times 1$ LUTs, where each LUT is a vertex in the DFG. The problem of partitioning the DFG into multi-input, multi-output representation can be formulated as an optimization problem considering evaluation time as the optimization objective and memory requirement as a constraint. The partitioning algorithm starts with creating a hypergraph from the vertices, 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), the 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 the PLP (for improving performance), we refer to 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 the 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.

After partitioning, the DFG that optimally reduces the number of LUTs is selected and to be used in the following operation mapping flow.

B. Operation Mapping

The inputs to the mapping software are the DFG description along with the ENFIRE configuration description, including the number of MLBs, the memory size, and number of registers per MLB. The tool also reports the number of cycles, number of operations by type, and the number of MLBs required for the input application. Latency and energy estimates for executing an application can be calculated based on the mapping results and considering the performance and energy models for each component of the hardware (Section V-A). The essential steps of the operation mapping flow are as follows.

1) *Fusion:* Some operations can be opportunistically combined to minimize the total number of operations needing evaluation as well as the total execution energy and latency. In this step, the partitions are levelized, and the vertices are shuffled among the partitions of the same level as well as across the levels (while maintaining topological dependence). This allows the vertices with shared inputs from some partitions to be subsumed inside another partition, thus, resulting in a reduced number of partitions.
2) *Packing:* Postfusion operations are mapped to the MLBs considering hardware resource availability and the nature of the operations. Maximum fanout free cone (MFFC) and maximum fanout free subgraph (MFFS) sharing [7] heuristics are used. Starting from a new MLB, the partitions are packed into the MLB as a cluster of partitions by using MFFS method under the constraints of memory size, I/O size, register size, and the number of LUTs allowed in each MLB. When the hardware resource is not sufficient for including more partitions, a new MLB will be employed using the MFFC method.
3) *Placement:* Partition clusters are assigned to the MLBs in the memory array. A recursive bisection-based heuristic that minimizes the PI usage is used to place the MLBs in a hierarchical fashion. Clusters are divided into the two groups in a way that minimizes the communication between them. Each group is then similarly divided into subgroups until it reaches the last hierarchical level, and all the subgroups are assigned to the hierarchal MLBs.
4) *Routing:* Communication between the MLBs follows the interconnect structure described in Section III-B. Interconnect information is stored in the instruction and statically scheduled. Communication between the LUTs must consider operation space, bus width, total memory size, and MLB register count. Routing is done in a bottom-up hierarchical manner, where local communications are routed first, followed by routing between the MLBs, then within higher hierarchical levels.

Several routing examples have been presented in Fig. 6. Communication within the clusters is presented on the left, while communication between the clusters is shown on the right. In the first case, the source and destination nodes are in the same MLB, and so the LUT output must be stored locally until the destination node has executed. In the second case, the source and destination nodes are in the same cluster, but different MLBs, and they are scheduled in two subsequent cycles. The intraMLB bus will be used, so that the source LUT output can be used as an input in a different MLB.

1-MOVE communication, including As Late As Possible (ALAP) and As Soon As Possible (ASAP), and bypass communication schemes occur when the destination is in another MLB in a future cycle. ALAP is given priority over ASAP, as it enables the sender to batch several bit transfers in a single MOVE operation. 1-MOVE bypass incurs greater communication overhead, and so it has lower priority than

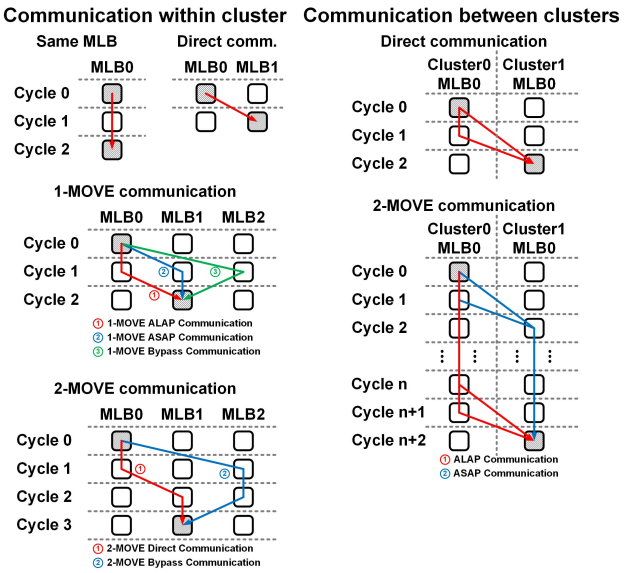

Fig. 6. Communication examples for operation mapping.

both ALAP and ASAP. 2-MOVE communication, including 2-MOVE direct and 2-MOVE bypass, involves two extra MOVE operations leading to higher power consumption and a lower priority than 1-MOVE. The same priority order is employed for intercluster communication. Here, an extra cycle is needed to hold the data on the bus as described in Section III-B. In case none of these schemes work for a particular communication, an additional cycle be inserted between the source and destination cycle. To minimize the total number of cycles, an iterative procedure is used to identify the maximally weighted communication in a given routing round, and the additional cycle is added there. Thus, only one additional cycle will be added per round until every communication has been properly routed.

V. Modeling Approach

Following the software mapping flow, the ENFIRE specifications and mapping result are available, including the number of operations by different LUT types, the number of MOVE operations, and the total number of cycles. These values are used to estimate the energy, latency, and area requirements for each application based on the ENFIRE hardware model. ISCAS and MCNC benchmarks [25] are considered in this paper, as they are representative of typical random logic workloads for this architecture. In this section, we introduce the hardware modeling approach, present the method used to evaluate the mapping results, and finally the method for compared with the FPGA mapping.

A. ENFIRE Hardware Modeling

A register transfer level (RTL) model of the logic elements was created to model the critical path delay, power consumption, leakage energy, and area of an MLB. The schedule table and LUT/data memory were modeled and simulated as latch and SRAM arrays using HSPICE [32]. The HSPICE models

TABLE I
MLB Key Parameters

Parameter	Value
Registers	64 x 1 b
Schedule Table	64 x 128 b
LUT/data Memory	4 kB
MLB Area	30,000 μm^2
Max Clock Frequency	1.3 GHz

TABLE II
MLB Critical Path Delay Breakdown by Unit

Component Delays	Delay (ps)
MLB Controller	50
Schedule Table	110
Decoder	70
Register Read	20
Address Generation	40
Memory Controller	140
Memory Access	250
Datapath	100
Register Write	30
Critical Path Delay	760
Proposed Cycle Time	**780**

Communication Delays	Delay (ps)
Intra-Cluster	12
Inter-Cluster	24

are 32-nm predictive technology models [33]. Both the low-power and high-performance models are included, and the designs are optimized to minimize delay and dynamic power consumption by selecting the optimal transistor models for each component. For example, the memory cells use the low-power model, while mux circuits use the high-performance model for fast access. We assume a worst case transistor temperature of 100 °C, which is the maximum valid value in these models. Other components were coded in synthesizable SystemVerilog and mapped to a 32-nm technology with a fast nMOS, fast pMOS (FF corner), and high temperature (125 °C) library from Synopsys [34] using DesignCompiler [29]. A 1 V power supply is used in the 32-nm modeling process. The simulation temperature is chosen as the worst case operating condition for the most conservative estimate of ENFIRE's power and performance.

Results presented in this section consider the MLB configuration outlined in Table I. Table I also includes the maximum allowable clock frequency for this configuration and required die area.

Each functional unit of the MLB was synthesized separately with delay constraints placed along the overall dataflow path. While this approach yields an inferior mapping result compared with a completely flattened netlist, it allows the energy use of each functional unit to be characterized individually. The critical path delay and $f_{\max}$ were obtained by summing the individual component delays along the critical path (Table II).

TABLE III
MLB ENERGY BREAKDOWN FOR DIFFERENT OPERATIONS

Operation	Contr. Logic (fJ)	Sched.-Tab. & Dec. (fJ)	Reg. File (fJ)	Exec. (fJ)	Total Energy (fJ)
8x1 LUT	2.65	14.92	1.48	37.64	56.69
8x2 LUT				75.77	94.82
8x4 LUT				147.1	166.2
8x8 LUT				287.8	306.9
Intra-cluster Move (4b)	0.50			47.85	64.75
Intra-cluster Move (8b)				95.71	112.6
Inter-cluster Move (4b)				95.71	112.6
Inter-cluster Move (8b)				191.4	208.3

An additional 20-ps timing slack was included to account for potential clock jitter or *RC* wire routing delay, bringing the cycle time to 780 ps (1.3 GHz). Communication delays were computed using an *RC* wireloading model based on the fabrication characteristics of the Synopsys 32 nm (FO4 load). Since these delays are so small compared with the LUT/data memory access time, they can be masked by the schedule table fetch of the previous cycle and are not in the critical path. For ENFIRE, the critical path delay is very different from FPGA, where it is primarily defined by a routing delay [35].

The energy for each component was also obtained from synthesis, assuming a 12.5% switching activity factor. Energy values for schedule table and LUT/data memory accesses were obtained from HSPICE. A read-skewed asymmetric SRAM design is also considered, which results in an additional 40% reduction in cell access energy [21]. Table III sums the energies of the components by the type of MLB operation to compute the total energy per operation.

The estimated area and leakage for an MLB were also taken from the synthesis and HSPCIE results. The total area is approximately 30 000 μm^2, 46% of which is the 4-kB LUT/data SRAM array. 17% of the area comes from the schedule table array, 13% comes from register file, and the remaining 24% is the MLB control and datapath logic. The total static leakage power per MLB is about 321 μW.

In addition to the component-by-component synthesis of the RTL model, a complete RTL model for one MLB was created and simulated using Synopsys VCS [36] to ensure the functional correctness of the model. A test framework using SystemVerilog was created to apply test stimuli. Results for a small test case were manually verified, ensuring that all the configuration data were written to the MLB correctly and that the LUTs produced the correct output responses.

B. FPGA Modeling

To facilitate a fair comparison with FPGA, the original BLIF file of each application was first translated to Verilog, and then directly compiled by Quartus II [28] onto an Altera Stratix V (5SEEBF45I2L, 0.85 V) device.

1) *Latency (T):* The minimum cycle time allowed for a successful mapping with fast silicon high temperature (fast 850 mV 100C) model.

TABLE IV
ENFIRE MAPPING RESULTS, INCLUDING THE NUMBER OF CYCLES, MLBS, OPERATIONS BY TYPE, AND LUT MEMORY REQUIREMENTS

App.	8x1 LUTs	8x2 LUTs	8x4 LUTs	8x8 LUTs	Total LUTs	MOVE Op.	Total Cycle	LUT Mem. (kB)
c432	10	3	8	1	22	5	11	1.75
c880	27	9	7	0	43	2	6	2.28
c1355	12	21	1	0	34	4	5	1.81
c1908	1	4	15	10	30	13	13	4.66
c2670	24	16	13	14	67	10	6	6.88
c3540	99	33	5	4	141	48	14	6.78
c5315	81	42	18	11	152	46	10	10.16
c6288	0	3	85	50	138	42	43	23.31
c7552	96	26	31	21	174	59	11	13.75
alu4	40	8	3	5	56	15	10	3.38
apex2	75	56	24	18	173	106	22	13.34
apex4	0	2	8	3	13	4	3	1.88
des	170	88	6	2	266	27	12	12.06
e64	0	0	0	10	10	0	10	2.50
ex5p	0	0	0	8	8	0	1	2.00
misex3	134	5	7	17	163	57	10	9.63
pdc	82	69	42	30	223	154	25	19.63
seq	270	14	5	29	318	139	21	17.19
spla	83	88	54	37	262	166	26	24.09

2) *Power (P):* A default toggle rate of 12.5% for both FPGA (PowerPlay [37]) and ENFIRE. In PowerPlay's report, three components (combinational cell, clock enable block, and register cell) are calculated into total power.
3) *Energy (E):* Total power multiplied by latency.
4) *Area (A):* We consider logic array block (LAB) area [38] and scale the results to a 32-nm process. The reported area estimates for the LABs include routing area, so that is not accounted for separately. This number is then multiplied by the number of utilized LABs reported by Quartus to compute the total effective area.
5) *Leakage (L):* The core static power dissipation reported by PowerPlay multiplied by the percent of total core area used. The leakage energy is then calculated as leakage power $\times$ latency.

VI. RESULTS

Table IV shows the ENFIRE mapping results for the ISCAS and MCNC benchmarks, as generated by the mapping tool described in Section IV. The energy, latency, area, and leakage estimates are based on these results. In general, small applications with less complexity will require fewer operations, fewer MLBs, and fewer extra MOVEs to finish the task. For example, c432 is a small application, requiring 22 LUTs across two MLBs, and therefore has fewer extra MOVEs and requires fewer cycles than larger applications. Conversely, large applications are expected to have higher complexity, and hence require more operations, more MLBs, more extra MOVEs, and more cycles. For example, seq has 318 LUTs, requiring 16 MLBs. The des application requires many operations, but only 27 extra MOVEs and 12 cycles, since most operations can be processed in parallel. Conversely, c6288 requires only five MLBs, but requires many cycles, since most operations are sequential.

Figs. 7–9 show the comparison results between ENFIRE and FPGA. Although energy consumption is our

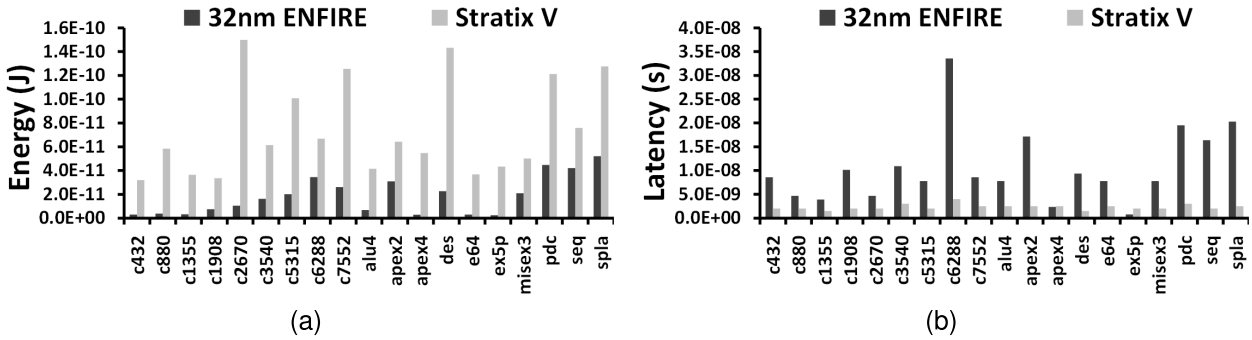

Fig. 7. Comparison between the ENFIRE and FPGA framework. (a) Energy. (b) Latency.

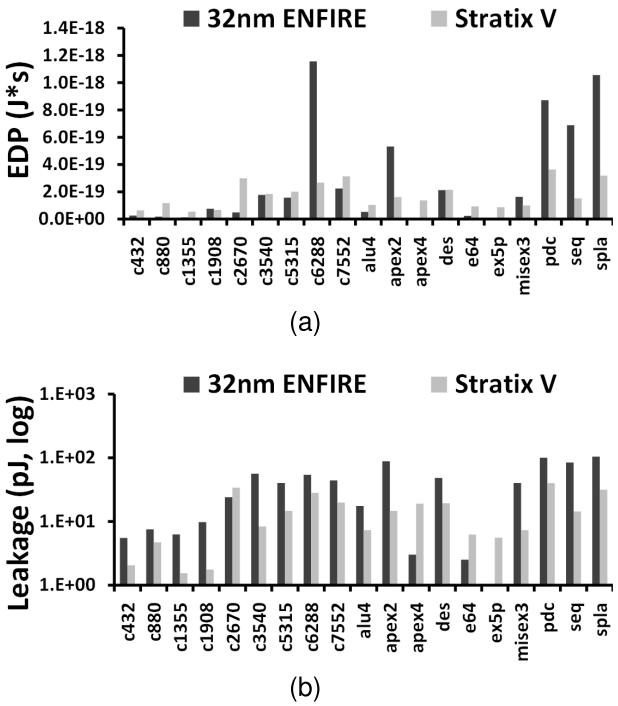

Fig. 8. Comparison between the ENFIRE and FPGA framework for (a) EDP and (b) Leakage.

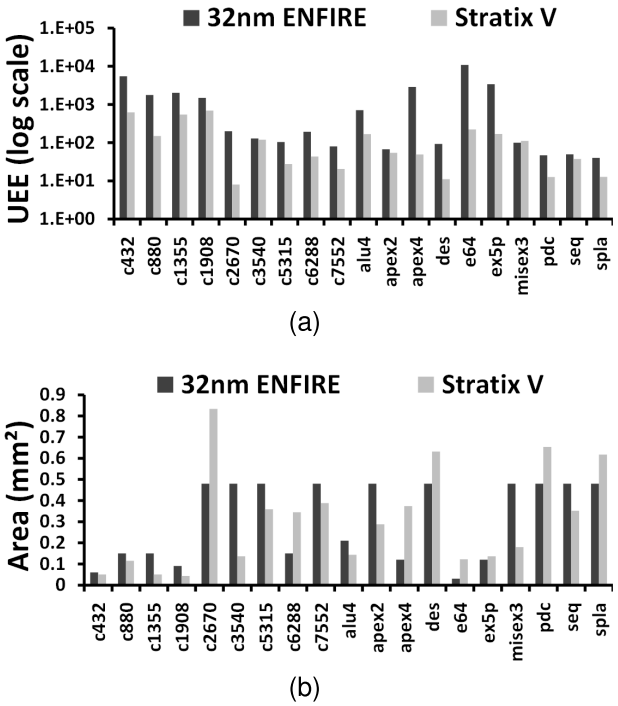

Fig. 9. Comparison between ENFIRE and FPGA framework for (a) UEE (higher is better) and (b) area.

primary concern, we also consider latency, area, and leakage, as well as energy-delay product (EDP) and a unified energy efficiency (UEE) metric, calculated as operation per energy per area. Since the same work load is performed in both ENFIRE and FPGA, the operation component simplifies to 1, leaving UEE as $1/(E * A)$.

Geometric means are used to get the EDP and UEE average over all applications. These two metrics are based on the product of other metrics, since the product may generate a bigger range for comparison. With geometric means, the ranges being averaged are normalized. One application will not dominate the average value, and the changes in all applications have the same effect on the mean value.

A. Energy and Latency Comparison

Comparison of energy between ENFIRE and FPGA is presented in Fig. 7(a) using the methods described in Sections V-A and V-B. ENFIRE energy is expected to be lower due to the spatio-temporal computing model, which reduces the PI requirement, consuming an average 7.6× less energy than FPGA. Similarly, the latency comparison is presented in Fig. 7(b). ENFIRE latency is expected to be higher than FPGA, again due to the computing model. The fusion stage of the software flow helps improve execution latency, since multiple fused functions can be computed in a single cycle. On average, FPGA latency is 2.4× better than ENFIRE.

B. EDP and Leakage Comparison

Comparison of EDP between ENFIRE and FPGA is presented in Fig. 8(a). ENFIRE improves on energy, but suffers a decline on latency compared with FPGA. Despite this, the energy improvement is sufficient, such that ENFIRE generally demonstrates EDP improvement over FPGA. One exception is in the sequential applications like c6288, pdc, seq, and spla, where the ENFIRE results are worse than the FPGA due to the latency. Note that the EDP of some applications, especially those that are more sequential in nature, is dominated by latency rather than energy. Applications in FPGA are

mapped as purely combinational logic, whereas in ENFIRE, every operation requires one full cycle whether the operation is simple or complex, and the same execution overhead applies in both cases. Because of the optimized mapping procedure, this is generally favorable to ENFIRE. Since EDP is a product relation of energy and latency, the geometric mean is taken, resulting in an average improvement of 1.6× over FPGA.

In addition, Fig. 8(b) shows the leakage energy comparison computed using the methods described in Sections V-A and V-B, respectively. Both latency and area contribute to the leakage; thus, in the cases where ENFIRE has lower latency and/or lower area, we observe an improvement in leakage energy over FPGA. Examples include apex4 and ex5p, which have better leakage due to the latency improvement, and c2670, apex4, and e64, which consume fewer resources, and thus less area compared with FPGA. On average, ENFIRE has a 1.1× improvement in leakage energy over FPGA.

C. Area and UEE Comparison

The UEE comparison is presented in Fig. 9(a) using the values generated from the ENFIRE area model and FPGA area estimation (Sections V-A and V-B). Fig. 9(b) shows the area results comparison. On average, the ENFIRE requires 1.2× less area than the FPGA. For most applications, ENFIRE demonstrates an average improvement of 5.3× in UEE, losing only by a small margin on three applications, despite the fact that only the FPGA LAB area was considered in the computation, whereas the entire MLB area was considered, even if only a small portion the MLB was used, but was nevertheless included in the computation. Per-application memory utilization is shown in Table IV. In addition, we note that while the FPGA area utilization is purely for computation, the MLB area includes memory, which is used for both computation (as an LUT) and data storage. From this, it follows that the effective computation area for ENFIRE will be less than that shown in Fig. 9. Finally, we note the VLIW capabilities of the ENFIRE datapath, which may not always be fully utilized in heavily sequential applications, but can be leveraged in applications amenable to parallelization.

D. Resource Comparison

Table V shows the hardware resource comparison. The ENFIRE contains fewer MLBs than the FPGA adaptive logic modules (ALMs). This is due to two reasons: first, ENFIRE has four types of LUTs (Section III), and each LUT is capable of storing a larger number of function outputs; second, each MLB can have up to 32 LUTs, while the ALM can contain at most two adaptive LUTs. As a spatio-temporal framework, the ENFIRE also contains fewer interconnects than FPGA. Spatial computing in ENFIRE is realized using communication buses in a hierarchal interconnect, thus mitigating the PI bottleneck in FPGA.

E. Scalability Analysis

As previously mentioned, scalability at the future technology nodes is a major research area for the FPGAs, and it has been shown that the PI architecture reduces the FPGA scalability. Therefore, we have investigated the scalability of the ENFIRE framework at the 22-nm node, where we expect the energy-efficiency improvement to increase compared with FPGA. The 22-nm model for ENFIRE is calculated by scaling with an industrial scaling factor [39]. The FPGA results are generated in Quartus II using a 20 nm Arria 10 FPGA (model number 10AX115U3F45I2SGES). Table VI shows the ENFIRE and FPGA scalability comparison results between 32- and 22-nm technology. ENFIRE has an 11.8× improvement in energy, a 2.4× improvement in EDP, and a 6.6× improvement in UEE, which by comparison are greater than the improvements at the 32-nm node.

TABLE V
RESOURCE COMPARISON BETWEEN ENFIRE AND FPGA MAPPING

App.	ENFIRE		FPGA	
	# MLBs	# PIs	# ALMs	# PIs
c432	2	8	40	500
c880	5	32	61	966
c1355	5	32	36	1341
c1908	3	8	45	669
c2670	16	48	126	2351
c3540	16	48	167	2282
c5315	16	48	170	4347
c6288	5	32	367	2699
c7552	16	48	210	5725
alu4	7	32	110	830
apex2	16	48	328	3379
apex4	4	8	373	3714
des	16	48	292	11036
e64	1	8	59	1571
ex5p	4	8	145	2359
misex3	16	48	240	1893
pdc	16	48	752	10173
seq	16	48	400	4519
spla	16	48	700	7656

TABLE VI
SCALABILITY COMPARISON BETWEEN 32- AND 22-nm IMPLEMENTATIONS OF ENFIRE AND FPGA

Metric	32nm Improvement	22nm Improvement
Energy	+7.6X	+11.8X
Latency	-2.4X [1]	-2.3X
EDP	+1.6X	+2.4X
Area	+1.2X	+1.0X
Leakage	+1.1X	+2.2X
UEE	+5.3X	+6.6X

VII. CONCLUSION

In this paper, we have presented ENFIRE, a novel reconfigurable framework suitable for fine-grained bit-level manipulation. A corresponding software mapping flow has been developed to effectively map diverse applications into this framework. To the best of our knowledge, this is the first example of a fine-grained reconfigurable spatio-temporal computing framework capable of FPGA-style application mapping. Our analysis with a set of random logic benchmarks show significant improvement in energy efficiency compared with a commercial FPGA at the same process node. ENFIRE uses high-density 2-D SRAM to store not only data but also multi-

input, multi-output LUTs. Since the design of the nanoscale memory array and its density are minimally affected, the proposed architecture can easily exploit the benefits of emerging memory technologies. The application mapping tool identifies the right balance between spatial and temporal computing to optimize performance. It also enables a user to effectively tradeoff performance with energy or resource utilization. ENFIRE's sparse bus interconnect also has a much lower impact on overall energy and scalability than the FPGA PI architecture. Comparison results with FPGA demonstrate considerable average improvement in energy (7.6×), EDP (1.6×), and UEE (5.3×). Future work will include investigations on mapping complex algorithmic tasks, such as applications in the digital signal processing and machine learning domains, which are amenable to spatio-temporal computing. Evaluation of ENFIRE with emerging memory technologies along with its silicon validation through test chip fabrication is also potential research directions.

REFERENCES

[1] P. H. W. Leong, "Recent trends in FPGA architectures and applications," in *Proc. IEEE Int. Symp. Electron. Design, Test Appl.*, Jan. 2008, pp. 137–141.

[2] D. Elléouet, Y. Savary, and N. Julien, "An FPGA power aware design flow," in *Proc. Int. Conf. Integr. Circuit Syst. Design, Power Timing Modeling, Optim. Simulation*, 2006, pp. 415–424.

[3] N. Banerjee, K. Roy, H. Mahmoodi, and S. Bhunia, "Low power synthesis of dynamic logic circuits using fine-grained clock gating," in *Proc. Design Autom. Test Eur.*, Mar. 2006, pp. 1–6.

[4] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[5] A. Rahman, S. Das, A. P. Chandrakasan, and R. Reif, "Wiring requirement and three-dimensional integration technology for field programmable gate arrays," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 1, pp. 44–54, Feb. 2003.

[6] S. J. E. Wilton, "SMAP: Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," in *Proc. Int. Symp. FPGAs*, 1998, pp. 171–178.

[7] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proc. Int. Symp. Field Program. Gate Arrays*, 1998, pp. 179–188.

[8] A. Dehon, "DPGA utilization and application," in *Proc. Int. Symp. FPGAs*, 1996, pp. 115–121.

[9] D. Jones and D. M. Lewis, "A time-multiplexed FPGA architecture for logic emulation," in *Proc. IEEE Custom Integr. Circuits Conf.*, 1995.

[10] S. C. Goldstein *et al.*, "PipeRench: A co/processor for streaming multimedia acceleration," in *Proc. Int. Symp. Comput. Archit.*, 1999, pp. 28–39.

[11] S. Paul, A. Krishna, W. Qian, R. Karam, and S. Bhunia, "MAHA: An energy-efficient malleable hardware accelerator for data-intensive applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1005–1016, Jun. 2014.

[12] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable coprocessor," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 1997, pp. 12–21.

[13] "Tabula, going beyond the FPGA with spacetime," *Field Program. Logic*, 2012. [Online]. Available: http://www.fpl2012.org/Presentations/Keynote_Steve_Teig.pdf

[14] A. Agarwal *et al.*, "A 320 mV-to-1.2 V on-die fine-grained reconfigurable fabric for DSP/media accelerators in 32nm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2010, pp. 328–329.

[15] M. Borgatti *et al.*, "A 1 GOPS reconfigurable signal processing IC with embedded FPGA and 3-port 1.2 GB/s flash memory subsystem," in *ISSCC Dig. Tech. Papers*, Feb. 2003, pp. 450–477.

[16] V. K. Prasanna, "Energy-efficient computations on FPGAs," *J. Supercomput.*, vol. 32, no. 2, pp. 139–162, May 2005.

[17] D. Chen, J. Cong, and Y. Fan, "Low-power high-level synthesis for FPGA architectures," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2003, pp. 134–139.

[18] Y. Hu, Y. Lin, L. He, and T. Tuan, "Physical synthesis for FPGA interconnect power reduction by dual-Vdd budgeting and retiming," *ACM Trans. Design Autom. Electron. Syst.*, vol. 13, no. 2, 2008, Art. no. 30.

[19] S. Paul and S. Bhunia, "A scalable memory-based reconfigurable computing framework for nanoscale crossbar," *IEEE Trans. Nanotechnol.*, vol. 11, no. 3, pp. 451–462, May 2012.

[20] *VPR and T-VPack 5.0.2 Full CAD Flow for Heterogeneous FPGAs*, accessed on Jun. 2017. [Online]. Available: http://www.eecg.utoronto.ca/vpr/

[21] S. Paul, S. Chatterjee, S. Mukhopadhyay, and S. Bhunia, "Energy-efficient reconfigurable computing using a circuit-architecture-software co-design approach," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 1, no. 3, pp. 369–380, Sep. 2011.

[22] J. Hammersley, *Monte Carlo Methods*. London, U.K.: Chapman & Hall, 2013.

[23] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

[24] *FPGA Architecture for the Challenge*, accessed on Jun. 2017. [Online]. Available: http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html

[25] *The Benchmark Archives at CBL*, accessed on Jun. 2017. [Online]. Available: http://www.cbl.ncsu.edu/benchmarks/Benchmarks-upto-1996.html

[26] *Graphviz—Graph Visualization Software*, accessed on Jun. 2017. [Online]. Available: http://www.graphviz.org/

[27] *Berkeley Logic Interchange Format*, accessed on Jun. 2017. [Online]. Available: https://www.ece.cmu.edu/~ee760/760docs/blif.pdf

[28] *Quartus II Subscription Edition Software*, accessed on Jun. 2017. [Online]. Available: http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html

[29] *Synopsys DC Ultra*, accessed on Jun. 2017. [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DCUltra/Pages/default.aspx

[30] *RASP—FPGA/CPLD Technology Mapping and Synthesis Package*, accessed on Jun. 2017. [Online]. Available: http://cadlab.cs.ucla.edu/software_release/rasp/htdocs/

[31] W. Qian, R. Karam, and S. Bhunia, "Trade-off between energy and quality of service through dynamic operand truncation and fusion," in *Proc. Great Lakes Symp. VLSI*, 2014, pp. 79–80.

[32] *HSPICE*, accessed on Jun. 2017. [Online]. Available: https://www.synopsys.com/tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx

[33] *Predictive Technology Model—Introduction*, accessed on Jun. 2017. [Online]. Available: http://ptm.asu.edu/

[34] *Synopsys 32/28 nm Generic Library for Teaching IC Design*. [Online]. Available: http://www.synopsys.com/Community/UniversityProgram/Pages/32-28nm-generic-library.aspx

[35] C. Schäfer, M. Stojilović, and L. Saranovac, "Analysis of impact of FPGA routing architecture parameters on area and delay," in *Proc. Telecommun. Forum (TELFOR)*, Nov. 2011, pp. 924–927.

[36] *VCS*, accessed on Jun. 2017. [Online]. Available: http://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/VCS.aspx

[37] *PowerPlay Early Power Estimators (EPE) and Power Analyze*, accessed on Jun. 2017. [Online]. Available: https://www.altera.com/support/support-resources/devices/power/pow-powerplay.html

[38] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture," in *Proc. 19th ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2011, pp. 5–14.

[39] W. Huang, K. Rajamani, M. R. Stan, and K. Skadron, "Scaling with design constraints: Predicting the future of big chips," *IEEE Micro*, vol. 31, no. 4, pp. 16–29, Jul./Aug. 2011.

Wenchao Qian (S'14) received the B.S. degree in automation from the Beijing University of Chemical Technology, Beijing, China, in 2009, and the Ph.D. degree in computer engineering from Case Western Reserve University, Cleveland, OH, USA, in 2016.

He is currently a Senior Software Engineer with Xilinx Inc., San Jose, CA, USA. His current research interests include reconfigurable computing, hardware architecture, software EDA tools, low-power circuit, neuromorphic computing, and field programmable gate arrays.

Christopher Babecki (S'11) received the B.S. degree in electrical and computer engineering and the M.S. degree in electrical engineering from Case Western Reserve University, Cleveland, OH, USA, in 2014 and 2015, respectively.

He is currently a Component Design Engineer with Intel Corporation, Hillsboro, OR, USA. His current research interests include hardware accelerators, reconfigurable computing, high-speed memory, and DFx circuit design.

Robert Karam (S'12) received the B.S.E. and M.S. degrees in computer engineering from Case Western Reserve University, Cleveland, OH, USA, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree in computer engineering with the University of Florida, Gainesville, FL, USA.

His current research interests include reconfigurable and domain-specific accelerator architectures, biomedical signal processing, neuromorphic computing, and hardware security.

Somnath Paul (M'11) received the Ph.D. degree in computer engineering from Case Western Reserve University, Cleveland, OH, USA, in 2011.

He is currently a Research Scientist with Intel Labs, Intel Corporation, Hillsboro, OR, USA. His current research interests include hardware–software co-design for improving energy efficiency, yield, and reliability in nanoscale technologies.

Dr. Paul was a recipient of the 2011 Outstanding Dissertation Award from the European Design and Automation Association and the 2012 Best Paper Award at the International Conference on VLSI Design.

Swarup Bhunia (SM'09) received the B.E. degree (Hons.) from Jadavpur University, Kolkata, India, the M.Tech. degree from IIT Kharagpur, Kharagpur, India, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA.

He was appointed as the T. and A. Schroeder Associate Professor of Electrical Engineering and Computer Science with Case Western Reserve University, Cleveland, OH, USA. He is currently a Professor with the University of Florida, Gainesville, FL, USA. He has over ten years of research and development experience with over 150 publications in peer-reviewed journals and premier conferences. His current research interests include hardware security and trust, adaptive nanocomputing, and novel test methodologies.

Dr. Bhunia received the IBM Faculty Award in 2013, the National Science Foundation Career Development Award in 2011, the Semiconductor Research Corporation Inventor Recognition Award in 2009, and the SRC Technical Excellence Award in 2005, and several best paper awards/nominations. He has been serving as an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, the IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, the *ACM Journal of Emerging Technologies*, and the *Journal of Low Power Electronics*, and served as a Guest Editor of the *IEEE Design and Test of Computers* in 2010 and 2013 and the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS in 2014. He served as the Co-Program Chair of the IEEE IMS3TW 2011, the IEEE NANOARCH 2013, the IEEE VDAT 2014, and the IEEE HOST 2015, and on the program committee of many IEEE/ACM conferences.