Meltrix: A RRAM-based Polymorphic Architecture Enhanced by Function Synthesis

Boyu Long[1, 2], Libo Shen[1, 2], Xiaoyu Zhang[1, 2], Yinhe Han[1], Xian-He Sun[3], Xiaoming Chen[1, *]
[1]*Institute of Computing Technology, Chinese Academy of Sciences*
[2]*University of Chinese Academy of Sciences*
[3]*Illinois Institute of Technology*
[*]*Corresponding Author*

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323544

***Abstract*—Field-programmable gate arrays (FPGAs) are popular for computational intensive applications and hardware accelerators recently. But they face limitations in memory capacity and its growth, resulting in excessive time spent on data access. The fixed capacity of embedded memory blocks also leads inflexibility and resource waste. Moreover, logic blocks in FPGAs which are insufficient for large-scale applications and fixed memory block positions both lead to high routing overhead. To address these issues, we propose a software-hardware co-designed polymorphic architecture called Meltrix. The hardware architecture, which uses RRAM arrays as the fundamental block, creates a unified fabric that can be reconfigured into logic, storage, and interconnection modes. We achieve multiple times of logic capacity compared with FPGAs' logic blocks and multi-level interconnections inside the tiles, which are used to solve the routing overhead problem in FPGAs. Moreover, the global routing complexity is further reduced by the proposed function synthesis framework, which isolates logic and memory components, synthesizes and maps them to configured tiles of Meltrix. Experiments show that, when comparing with commercial FPGAs and state-out-of-art Liquid-Silicon, Meltrix achieves 1.89-3.14× performance improvement and 2.08-4.17× power reduction in both logic-intensive and memory-intensive applications.**

***Index Terms*—Processing in memory, polymorphic architecture, resistive random-access memory (RRAM), software-hardware co-design, function synthesis**

I. Introduction

Field-programmable gate arrays (FPGAs) have gained popularity in various computational intensive applications owing to their flexibility, high performance, and low power consumption [1], [2]. Moreover, FPGAs are a viable option for implementing hardware accelerators, owing to their highly programmable capability [3]–[5]. Recently FPGAs have undergone significant advancements in the computing capability and hardware resources. For example, compared with the previous generation Virtex 7 [6], the Virtex UltraScale [7] has increased the number of configurable logic blocks (CLBs) by 3.56× and the storage capacity by 30.9%. Even though, FPGAs are still facing challenges when dealing with modern applications, especially data-intensive applications.

This work was supported by National Key R&D Program of China (No. 2018YFA0701500), by National Natural Science Foundation of China (Nos. 62122076, 61834006, and 62025404), by Key Research Program of Frontier Sciences, CAS (No. ZDBS-LY-JSC012), by Strategic Priority Research Program of CAS (No. XDB44000000), and by Innovative Project of Institute of Computing Technology, CAS (No. E261040).

A development trend of FPGAs is that, the memory cell growth has lagged behind that of logic cells [6]–[8]. This limitation in the decreased ratio of logic to memory, results in FPGAs spending too much time on data access and movement with external memory when deploying data-intensive applications such as large-scale neural networks or recommendation systems [9], [10]. Clearly, excessive main memory access has a catastrophic impact on the performance of these applications. Additionally, the fixed capacity of each embedded memory block (EMB) results in a lack of flexibility. In most cases, every storage module mapped to the device inevitably wastes EMB resources, unless the target capacity of each storage module perfectly matches the EMB capacity.

Furthermore, the granularity of the basic logic units in FPGAs, such as Xilinx's CLB which consists of four 6-input look-up tables (LUTs), and Intel's adaptive logic module (ALM) which consists of one 8-input LUT, has become insufficient for the implementation of various large-scale applications [11]. Thus the rapidly increasing demand for logic modules and their increasing complexity is putting significant pressure on routing within FPGAs. Besides, the fixed positioning of EMBs makes routing near storage modules more challenging, and this in turn results in increased power consumption. These two factors combined lead to high routing overhead in FPGAs.

Resistive random-access memory (RRAM) devices, which possess non-volatile variable resistance, can be used as both switch devices and non-volatile storage units [12], [13]. The device-level multi-function feature of RRAM devices provides a potential solution to the problems in FPGAs. Based on this feature, Liquid-Silicon (L-Si) [14] employed reconfigurable RRAM computational arrays to enable storage mode for all tiles to reduce memory granularity and increase flexibility, partially solving the first problem faced by FPGAs as mentioned above. However, due to the lack of relevant peripheral circuits, a storage module requires three times of the logic resources of its storage area, rendering this solution ineffective. Additionally, L-Si replaced the LUT implementation method in FPGA logic with the sum-of-product (SOP) method, attempting to reduce the overall depth of the logic network and mitigate the routing overhead in FPGAs by using multi-input (∼30 inputs) SOPs. However, this method overlooked the issue of exponential signal replication during SOP implementation, resulting in a large number of signal copies and no decrease

in routing overhead despite the reduced depth and interconnection complexity of the logic network.

In this work, we propose a software-hardware co-designed architecture called Meltrix to address the issues in FPGAs and the shortcomings of previous work. For the hardware architecture, we utilize RRAM arrays as the basic structure and implement a homogeneous fabric that can flexibly reconfigure into logic, storage, and interconnection modes with peripheral circuits. By melting the gaps among the three operation modes and fusing them into a matrix-like fabric, Meltrix is a polymorphic architecture that can efficiently support various functions by a unified architecture. In logic mode, Meltrix supports several times of the LUT capacity and multi-level interconnections in the tiles comparing that of FPGAs. With the addition of avoiding signal duplication, we effectively solve the routing overhead problem present in both FPGAs and L-Si. In storage mode, Meltrix contains an input decoding framework in the tiles with low latency and power consumption. Thus we break the limited storage capacity of FPGAs and achieve a higher flexibility compared with L-Si. In interconnection mode, RRAM devices serve as switches, enabling inputs to be arbitrarily remapped to outputs.

On the software side, to fully exploit the potential of the hardware architecture of Meltrix, we propose a function synthesis framework which takes hardware description languages, such as Verilog and VHDL, as input, and generates function and logic mapping results with minimal routing overhead as output. The high-level description is first compiled during which functions are extracted. The isolated logic and storage components are synthesized independently. The synthesized modules are further optimized through an innovative Two-stage Top-down Minimum Communication Clustering method (TTMCC) and Fully-Interconnected Basic Units (FIBUs) for resource mapping and placement. Together with the multi-level interconnections in tiles and the localized connectivity of the arrays, the complexity of global routing is maximally reduced.

In summary, we make the following contributions.

- We propose a novel integrated polymorphic architecture named Meltrix, featuring coarse-grained logic mode with arbitrary internal interconnectivity, reconfigurable storage mode with minimal logic resource usage, and the fully connected interconnection mode for input and output.
- We introduce a function synthesis framework corresponding to the architecture as mentioned earlier, which isolates logic and memory components, and optimally maps them to the resources of Meltrix. The logic components are mapped through the first-time proposal of the TTMCC method and FIBUs.
- We implement the hardware architecture using 45nm technology and experiment with the following architectures using the work flow as discussed above:
  - 45nm SRAM FPGA,
  - 18nm SRAM FPGA,
  - 45nm RRAM L-Si.

  Compared with the 45nm commercial SRAM FPGA, the delay and power consumption are reduced by 3.14× and 3.13×, respectively. Compared with the 18nm SRAM FPGA, the power consumption is reduced by 4.17× with the same delay. Compared with the state-of-the-art L-Si (45nm), Meltrix achieves 1.89× performance improvement and 2.08× energy reduction.

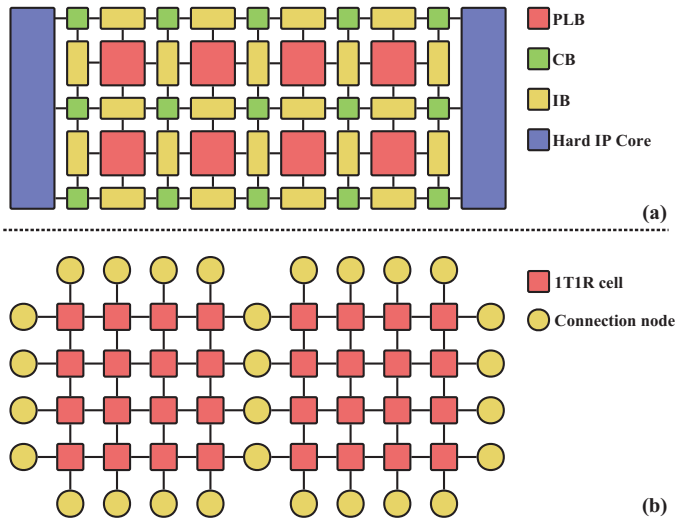

Fig. 1. (a) FPGA architecture. (b) Liquid-Silicon architecture.

II. Background

A. FPGA Architecture

The classic architecture of FPGAs is illustrated in Fig. 1(a), consisting of programmable logic blocks (PLBs), interconnection blocks (IBs), switching blocks (SBs), and hard IP cores. The PLBs are used for implementing arbitrary logic operations, while hard IP cores perform specific tasks beyond general logic, such as EMBs, digital signal processing (DSP), and high-speed serial interfaces. The IBs and SBs form the interconnection network, connecting the logic units, hard IP cores, and IO ports.

This conventional FPGA architecture has the following issues. 1) EMBs, as a hard IP core, are scarce in number and fixed in position compared with logic units, thus limiting the storage flexibility and capacity to under 100Mb [7]. This limitation results in excessive communication with external storage in data-intensive applications, significantly impacting performance. 2) Facing the increasing scale of applications, the fine-grained design of the PLBs introduces undue complexity to global routing, compounded by the fixed position of the EMBs mentioned in 1), resulting in high delay and power consumption, namely, routing overhead in FPGAs.

B. RRAM Device

As one of the mainstream novel non-volatile memory devices, RRAM exhibits excellent scalability (<10nm) [15], full compatibility with CMOS, fast read/write speed (<10ns for wirte) [16], and low operating voltage. By applying proper voltage to the terminals of a RRAM, its resistance can be altered, and its resistive state can be measured without changing

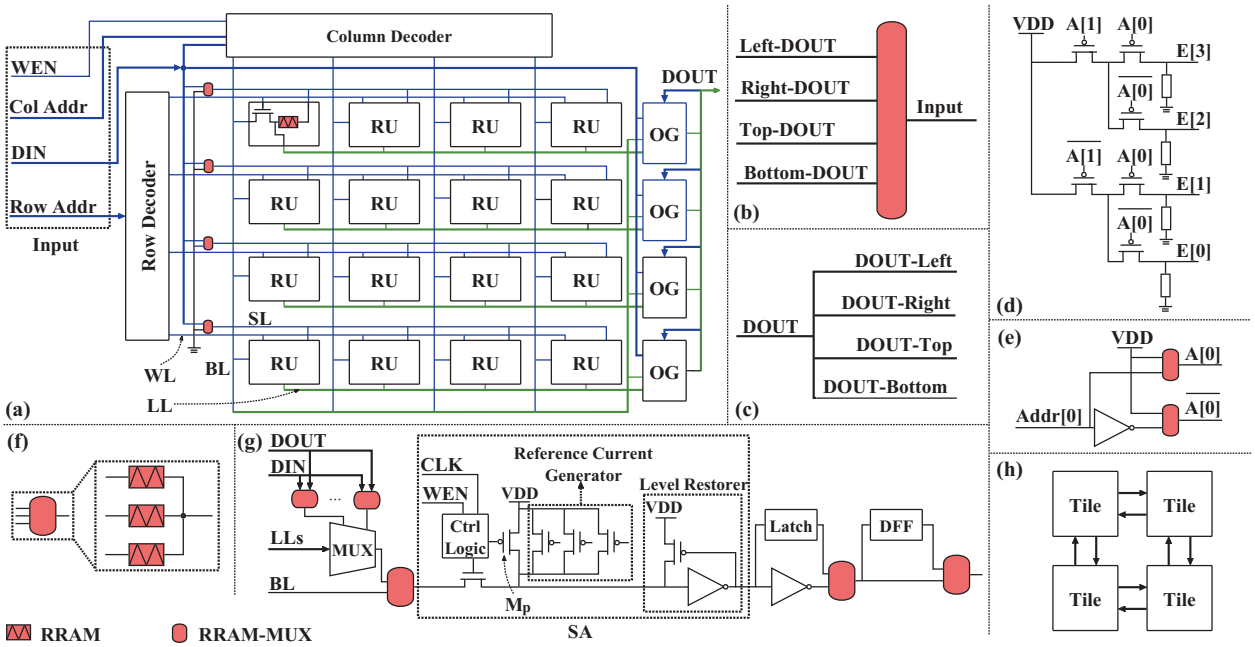

Fig. 2. (a) Tile architecture of Meltrix. (b) Input structure. (c) Output structure. (d) An example of a 2-input one-hot decoder. (e) An example of a 1-bit address selector. (f) RRAM-MUX. (g) Output generator (OG). (h) Overall architecture.

the resistance by applying a weak voltage. This straightforward read/write method enables us to easily construct high-density and low-power crossbars.

More importantly, RRAM devices can act as both storage components and switching units, making them an optimal solution to create reconfigurable architectures without employing volatile and area-consuming SRAMs, thus not only improving performance and energy efficiency, but also greatly enhancing the flexibility in the architecture for functional polymorphism. Through functional polymorphism, we break the boundaries of different functions and use the same resources for all functions in FPGAs, thus resolving the problems mentioned above.

C. Related Work

Several works to apply RRAMs in FPGAs have been proposed recently. In these works RRAMs have been utilized as an improvement component for FPGAs [17]–[19], such as replacing CBs and SBs or using RRAMs to construct high-density EMBs in hard IP cores. However, these approaches only address some aspects of the problems and cannot solve the capacity limitation and routing overhead problems that exist in FPGAs due to architectural reasons.

Another interesting study is conducted by L-Si [14], which attempted to solve the above issues by constructing crossbars composed of RRAMs, as shown in Fig. 1(b). However, the issues still remain because of the simple peripheral circuits. For example, when completing the storage mode, the three adjacent tiles of a memory tile need to be configured as relevant address decoding logic, which leads to a great waste of resources. The use of SOPs instead of LUTs in the logic aspect increases the routing complexity due to a large amount of signal duplication, but L-Si has no internal routing. So all routing is directly reflected in the interconnection modules, which cases other form of routing overhead.

III. Meltrix Hardware Architecture

A. Overview

The Meltrix architecture is a densely-packed 2D planar array composed of identical tiles, where each tile is an identical structure composed of a 2D RRAM crossbar array and peripheral circuits. Fig. 2(a) shows the tile architecture. Each tile is directly connected to the surrounding four tiles through configurable interconnection wires (Fig. 2(h)). Each tile can be flexibly configured into three modes: **logic mode**, **interconnection mode**, and **storage mode**. This feature allows us to adjust the storage ratio based on the workload and implement distributed storage modules. The interconnection mode and local connectivity replaces the global routing modules of FPGAs, thereby improving the flexibility of routing configuration and increasing resource utilization efficiency. The logic mode is based on LUTs. Each row of a tile corresponds to a LUT.

Fig. 2(a) shows the construction of a tile (the blue wires are inputs and the green wires are outputs), which includes a 2D array composed of RRAM Units (RUs), output generators (OGs), row and column address decoders (see Section III-D), and RRAM-MUXs (RMUXs, as shown in Fig. 2(f)). Fig. 2(b) and Fig. 2(c) show the IO connections between tiles. Each tile accepts outputs of adjacent tiles in four directions and selectively connects to the required inputs including DIN, WEN, row address, and column address. At the same time, the output of each tile drives the output wires in all four directions.

As the fundamental part of a tile, an RU is a 1T1R storage cell. A word line (WL) of the crossbar array controls the

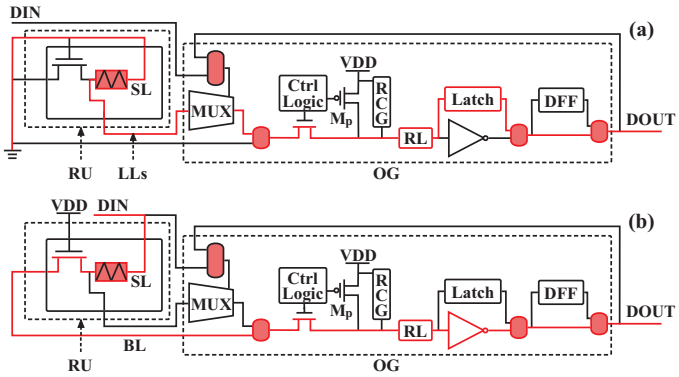

Fig. 3. (a) Data path in **logic** mode. (b) Data path in **interconnection** mode.

transistor status of each row of RUs. In addition to controlling the writing of a column of RUs, the bit line (BL) is also directly connected to the corresponding row's OG (the BL in the m^{th} column connects to the OG in the m^{th} row). The sense line (SL) can selectively connect the corresponding DIN or ground. The LUT line (LL) drawn from the connection between the access transistor and RRAM is directly connected to the corresponding row OG.

The mode switching of the tiles is achieved by forming different data paths by configuring RMUXs. RMUXs are implemented by RRAMs, whose resistances are set during the initialization (i.e., burning) phase, at the same stage of setting the storage units of LUTs, without modification during runtime. It should be noted that all RRAMs in this work adopt binary settings, i.e., using high resistive state (HRS) and low resistive state (LRS).

Fig. 2(g) illustrates that an OG comprises an input-MUX, a sense amplifier (SA), a latch branch, and a DFF branch. The input-MUX employs a MUX and RMUXs to select the required input from BL and LLs. The SA with a reference current generator (RCG) controlled by different modes uses a level restorer (LR) to ensure accurate signal amplification. The latch and DFF branches correspond to output holding and sequential-combinational selection branches, respectively. In the following contents we will introduce the three operation modes of the tiles.

B. Logic Mode

In logic mode, the RUs and the OG in the same row construct a LUT, in which the RRAMs in the RUs store the truth table. Thus a tile with $N \times N$ RUs can form up to N $\log_2 N$-input LUTs. The data path of logic mode is shown in Fig. 3(a). WLs and SLs are disabled. The LUTs work in a dynamic logic-like way. In the precharge phase (CLK is low), the "Ctrl Logic" in OG precharges the SA's input for the next clock cycle by using the transistor M_p. At the same time, the latch in the OG maintains the output. Then in the evaluation phase (CLK is high), the SA's input is connected to the selected RRAM by MUX. The resistance of the selected RRAM determines the SA's output. The control signals of OG's MUX are jointly screened from DIN and DOUT, ensuring that LUTs in the same tile can be cascaded arbitrarily. If a combinational circuit is formed, then the DFF

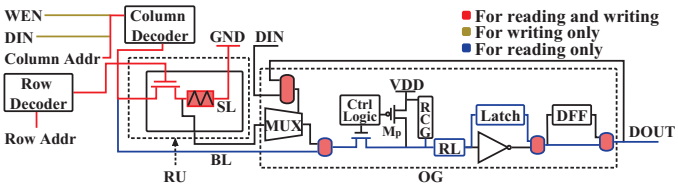

Fig. 4. Data path in **storage** mode.

is bypassed and the output of the latch is the final output. On the other hand, a sequential circuit can be built by using the DFF.

C. Interconnection Mode

The tiles working in interconnection mode are very similar to SBs in FPGAs. It accepts inputs from four directions and connects them to the corresponding outputs. All enabled outputs have a surjective relationship with inputs, meaning that each output has a corresponding input, but each input may drive multiple outputs. In the data path of Fig. 3(b), all WLs are set to high to turn on all access transistors of the RUs. A DIN passes through SL and is connected to BL through a RRAM in LRS. The level restorer in the OG will select BL as input and directly connect to DOUT through an inverter. More generally, for an interconnection tile, if DIN[p] is connected to DOUT[q], the RRAM in row p and column q will be set to LRS, and the remaining RRAMs in column q will be set to HRS. It should be noted that each column can have at most one RRAM in LRS, but each row can have multiple RRAMs in LRS. The whole process is a combinational logic, so long signals can be transmitted through multiple interconnected tiles.

D. Storage Mode

A complete storage module consists of a storage mode tile and a logic mode tile. Due to the fixed width of tile outputs, when the word size (defined by applications) is less than the row length, there will be redundant outputs from the storage mode tile. For example, consider a case where the row length is 4bit and the word size is 2bit. If the RRAM resistance values in a row are "0110", reading the low-order word will output "0010", and reading the high-order word will output "0100" when we only utilize a storage mode tile. So the logic tile implements necessary operations to eliminate redundant outputs. If the word size equals to the row length, only a storage tile is needed to form a storage module. Fig. 4 shows the details of the data path in storage mode.

Row and column decoders are activated in the storage mode. The row decoder is a one-hot decoder which takes row address as input and selects the corresponding row. An example of a 2-input one-hot decoder is shown in Fig. 2(d). In the column decoder, besides the one-hot decoder, an address selector is also needed to mask the unwanted bits when the word size is less than the row length. Fig. 2(e) shows an example of a 1-bit address selector.

1) Write Operation: The row and column addresses are input into the row and column decoders, respectively, to select the corresponding RUs to be written. SLs are grounded. When

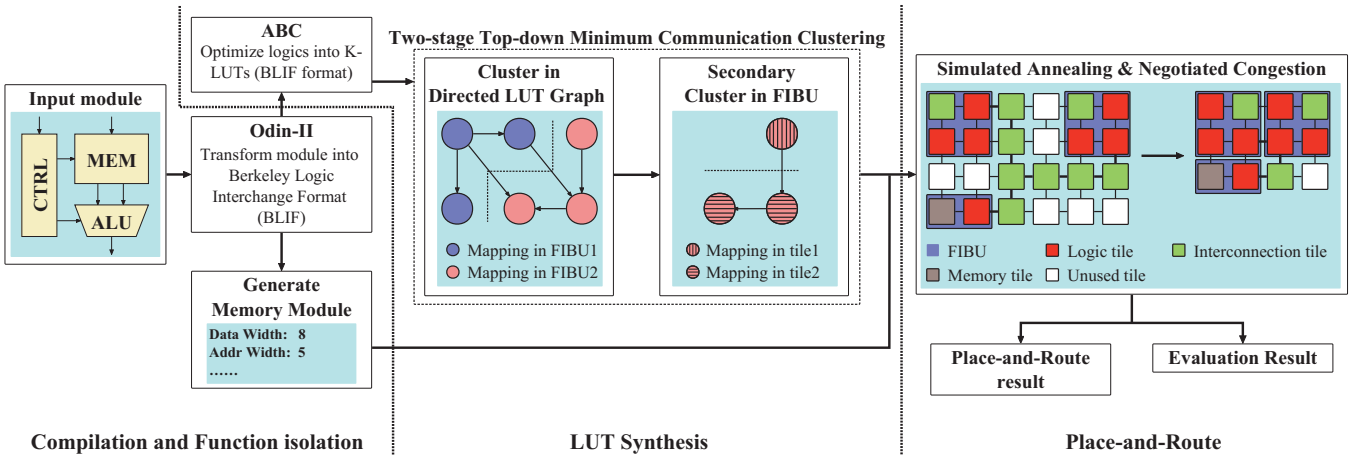

Fig. 5. Function synthesis framework of Meltrix.

WEN is set to high, the column decoder generates proper BL voltages according to DIN, thereby completing the write operation for the selected RRAMs.

2) Read Operation: The process of the read operation is similar to the logic mode operation. First, in the precharge phase, the transistor M_p sets the input of SA to high and the latch maintains the output. In the evaluation phase, while M_p being disconnected, the resistance of the RRAM connected to BL determines whether the output is pulled to ground or V_{DD}. It is important to note that if some columns are not be selected by the column address, the corresponding BLs will be set to high by the column decoder and the outputs of these columns will be 0.

IV. Meltrix Function Synthesis

By melting three different operation modes into a unified tile fabric, applications can fully leverage the potential of the Meltrix architecture to achieve high performance. When mapping applications to the hardware architecture of Meltrix, an important problem must be solved: how many tiles every mode needs and where they should be placed. In this section we introduce a function synthesis framework to automatically map applications to the proposed polymorphic architecture. In the function synthesis process, this framework aims to enhance the global routing efficiency at the multi-tile mapping and interconnection levels.

Fig. 5 illustrates the function synthesis framework, which has three main steps. The function synthesis framework takes hardware description languages (Verilog HDL and VHDL) as input. The first step is **compilation and function isolation**. The input modules are first converted into logic netlists by employing ODIN-II [20], which is a part of the open-source academic FPGA computer-aided-design (CAD) tool VTR [21], aiming at converting HDL designs into flattened logic netlists. The storage format is the Berkeley Logic Interchange Format (BLIF) [22] which is particularly suitable for representing combinational and sequential components such as LUTs and latches. The framework then identifies and separates memory modules within the logic netlist, leaving behind a pure logic network composed of LUTs and latches.

The second step deals with **LUT synthesis**. The logic network undergoes optimization to K-input LUTs (K-LUTs) using the open-source tool ABC [22], which is a logic synthesis and formal verification tool utilized for logic circuits within hardware designs. The optimized logic network is then transformed into a directed graph in which nodes represent LUTs and edges represent dependencies between LUTs. The graph utilizes the proposed FIBUs (refer to Section IV-A) and the TTMCC algorithm (refer to Section IV-B) to generate an initial layout in conjunction with auto-generated memory FIBUs.

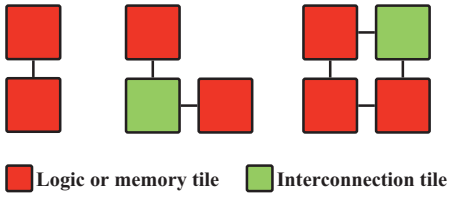

Fig. 6. Three types of FIBUs which contain 2, 3 and 4 tiles, respectively.

The last step is the **place-and-route** phase. It determines where each tile should be placed and the interconnections between them. The simulate annealing (SA) method [23] is used to explore the design space. The SA method randomly makes perturbations in each iteration, which includes FIBUs' movement, rotation and swap, to the current layout. The perturbed layout is routed using a negotiated congestion routing algorithm [24] that minimizes the routing congestion through negotiation and recorded methods. Following the SA method, the framework generates a final mapping with minimized routing cost, along with detailed network evaluation results.

A. Fully-Interconnected Basic Unit

A Fully-Interconnected Basic Unit (FIBU) is a set of tiles, which meets the following condition (Fig. 6 shows three examples of different FIBUs):

- Any two tiles in a FIBU has a path which can be 1) formed by using interconnection tiles in this FIBU, or 2) a direct connection.

The purpose of proposing the concept of FIBU is to reduce the global routing complexity. The internal connection relationship within a FIBU can be completely described by interconnection tiles in itself or direct connections. Therefore, by introducing FIBUs, the global routing can be divided into two parts: interconnections between FIBUs and interconnections within FIBUs, where the latter can be trivially determined so that the global routing complexity is greatly reduced. Therefore, in the routing process, we only optimize the inter-FIBU interconnections.

B. Two-stage Top-down Minimum Communication Clustering Algorithm

The proposed TTMCC algorithm is a top-down clustering approach that utilizes METIS [25] to enable minimum communication cuts twice for the input graphs. As shown in Fig. 5, prior to using TTMCC, a directed graph is generated where LUTs serve as nodes, and their dependencies are represented as edges. The pseudo code of TTMCC is shown in Algorithm 1. Initially, TTMCC partitions the complete graph into multiple clusters optimized for FIBUs by using METIS [25] (line 1), which is followed by some initialization works (lines 2-3). The first while-loop verifies the validity of the clustering results (lines 6-12), that is, if the number of inputs and outputs or the node number in the clustering results surpasses the upper limit of FIBUs/tiles, the number of clusters will be incremented, and the clustering process restarts (line 13) until all clusters pass the validity test (lines 4 and 14). Subsequently, TTMCC further divides these FIBU-optimized clusters into secondary clusters that can be mapped onto tiles (lines 18-19) and perform a validity check on the second clustering results similar to the first clustering (lines 16, 21-24 and 26-27).

Comparing with the greedy bottom-up seed-based clustering employed in VTR [21], the rationale behind Meltrix utilization of TTMCC lies in the characteristics of FIBUs and tiles, namely, the internal connections after clustering are irrelevant. Ref. [21] shows that achieving high logic utilization has an adverse effect to global routability [26], [27] in FPGAs. So in the former approach, both the number of nodes and the connection density within the clusters are equally crucial, whereas our algorithm focuses primarily on the interconnections among clusters.

Algorithm 1: Two-stage Top-down Minimum Communication Clustering

Input: Directed LUT graph G
Output: List of FIBUs L_{FIBU}, and list of tiles L_{tiles}

1 $FC = MinComPartGraph_{Metis}(G)$;
2 $clusters_{illegal} = \varnothing$;
3 $L_{FIBU}.clear$;
4 **while** FC *is not empty* **do**
5 $clusters_{illegal}.clear$;
6 **foreach** *cluster in FC* **do**
7 New $FIBU$;
8 **if** $Mapping(cluster, FIBU).success$ **then**
9 Push $FIBU$ into L_{FIBU};
10 **else**
11 Push $cluster$ into $clusters_{illegal}$;
12 Delete $cluster$ from FC;
13 $Result = SplitIllegalClusters(clusters_{illegal})$;
14 Connect FC with $Result$;
15 $L_{tiles}.clear$;
16 **while** FC *is not empty* **do**
17 $clusters_{illegal}.clear$;
18 **foreach** *cluster in FC* **do**
19 $SC = MinComPartGraph_{Metis}(cluster)$;
20 New L_{tile};
21 **if** $Mapping(SC, L_{tile}).success$ **then**
22 Push L_{tile} into L_{tiles};
23 **else**
24 Push $cluster$ in $clusters_{illegal}$
25 Delete $cluster$ from FC
26 $Result = SplitIllegalClusters(clusters_{illegal})$;
27 Connect FC with $Result$;

TABLE I
BENCHMARKS AND TEST SAMPLES USED IN EVALUATION.

Benchmark	Sample	Description
ISCAS'89	s38417	Large scale sequential circuit
	s38584	
MCNC	dsip	Encryption
	bigkey	
	dalu	Dedicated ALU
	mm30a	Minmax circuit
	mult32a	Multiplier
	mult32b	
VTR	sha	Sha-160 hash
	ch_intrinsics	Memory controller
	mkPktMerge	Memory merge

V. EVALUATION

This section presents a comprehensive performance evaluation of Meltrix. We employ the ISCAS'89 and MCNC benchmarks [28], along with the built-in benchmarks of VTR. Table I lists the test samples and their purposes. We select the two largest sequential circuit samples from ISCAS'89. In MCNC, our focus is on real applications, encompassing two encryption algorithms and four distinct computing units. In VTR's benchmarks, we evaluate Meltrix's performance in storage by using a memory controller and a memory merge process. Both of the memory samples have varying storage capacities. Moreover, a hash algorithm in VTR's benchmarks with complex routing is utilized to test the logic performance of Meltrix.

In the evaluation, three different baselines are selected for comparison. First, we select the 45nm Xilinx Spartan 6 series FPGA [8] to showcase the comprehensive performance advantages of Meltrix with the identical technology. Additionally, we opt for a 16nm Xilinx Virtex Ultrascale+ series FPGA [7], further substantiating the significant competitiveness of Meltrix despite utilizing outdated manufacturing processes. To ensure that Meltrix's performance advantages primarily stem from its superior architecture rather than the device itself, we

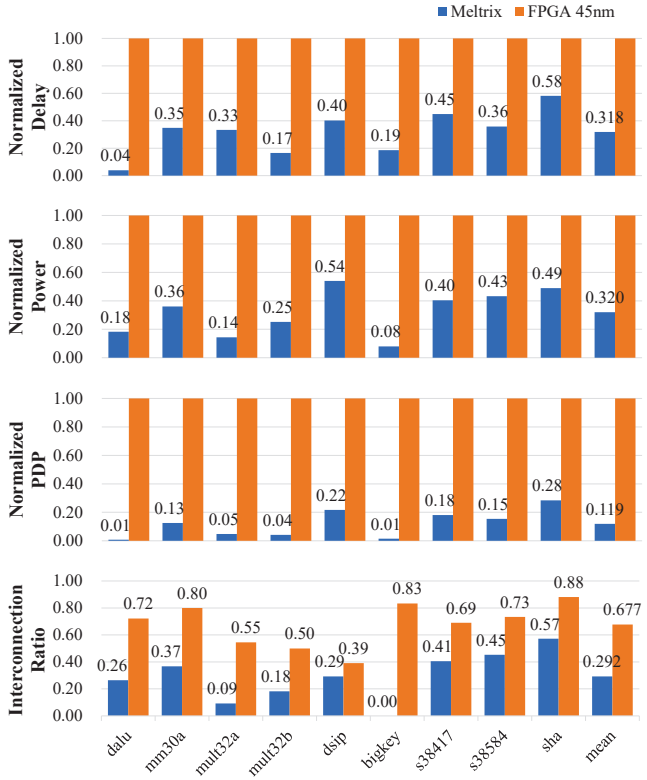

Fig. 7. Evaluation compared with 45nm FPGA.

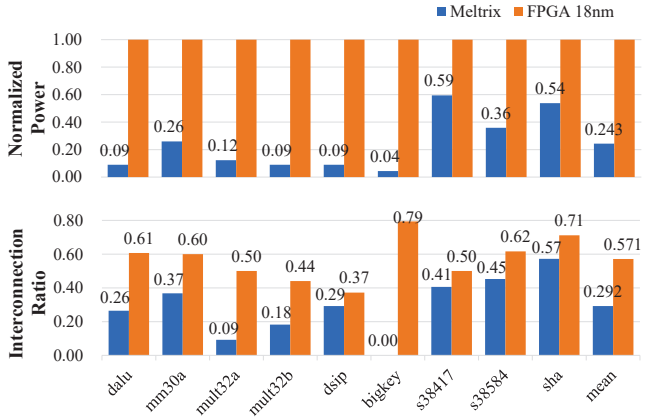

Fig. 8. Evaluation compared with 18nm FPGA.

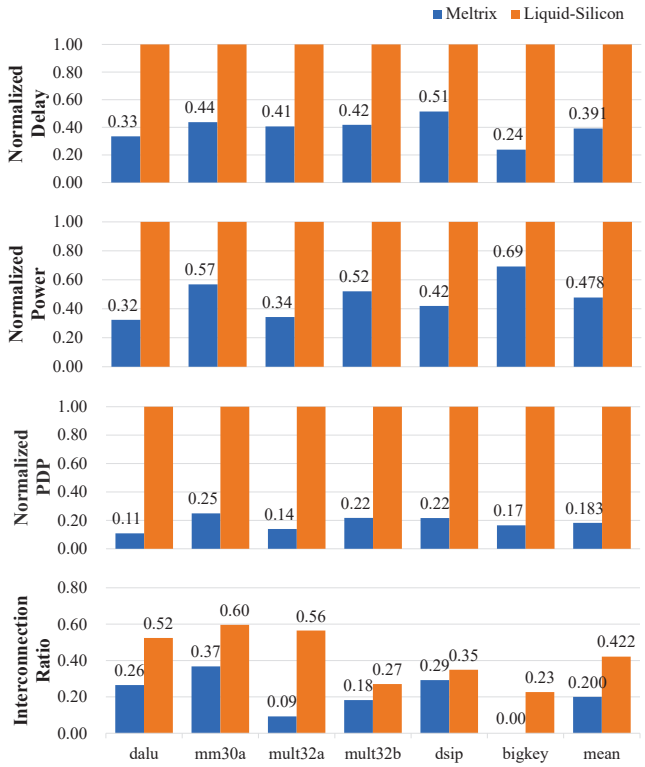

Fig. 9. Evaluation compared with 45nm Liquid-Silicon.

also include the RRAM-based reconfigurable architecture L-Si (45nm technology) as the third baseline. This approach guarantees that Meltrix's performance benefits extend beyond emerging non-volatile device technologies.

To obtain the most accurate delay and power consumption results when testing Spartan 6 and Virtex Ultrascale+, we employ Xilinx's commercial CAD tool Vivado [29]. We use HSPICE [30] to measure the parameters of each mode of a tile for Meltrix and L-Si. Additionally, the synthesis and performance estimation for Meltrix and L-Si rely on the function synthesis (see Section IV) which is written in C++ and **typically takes anywhere from a few seconds to approximately ten minutes**. The characteristics of RRAM devices used in Meltrix and L-Si are 1) $R_{HRS}/R_{LRS} = 2M\Omega/4.82k\Omega$, 2) $\pm 1V@10ns$ pulse voltage for set/reset [16], [31].

A. Logic Evaluation

The experiment employs all pure logic circuits from Table I. The experimental results of Meltrix will be individually compared with the three baselines mentioned above. Furthermore, we will provide detailed explanations for the observed outcomes.

In Fig. 7, we use a 45nm Spartan 6 series FPGA as the baseline, considering delay, power consumption, power-delay product (PDP), and the interconnection power ratio (ratio of the interconnection power to the total power) as the evaluation criteria. It is important to note that the test results indicate a trade-off between delay and power consumption. To ensure standardization, all test samples in the Meltrix versus 45nm FPGA comparison are synthesized using the "optimal latency" mode. In this experiment, Meltrix achieves an average delay reduction of 68.2% compared with the baseline, demonstrating that Meltrix's coarse-grained logic tiles and optimized function synthesis alleviate the complexity of global interconnections, thereby reducing delay of critical paths. Moreover, Meltrix exhibits a 68% reduction in power consumption, leading to an 88.1% decrease in PDP compared with the baseline. The interconnection power ratio comparison reveals a significant decrease from 67.7% in the baseline to 29.2% in Meltrix, indicating the effective solution of the routing overhead issue encountered in FPGAs. Notably, the **bigkey** test sample exhibits a interconnection power ratio of 0, signifying that all interconnections in this sample are accomplished through direct connections of the logic tiles. The results comprehensively demonstrate the considerable advantages of Meltrix over

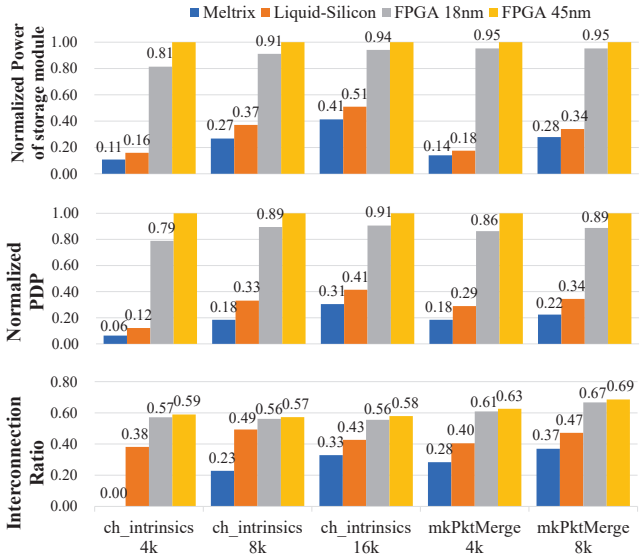

Fig. 10. Evaluation compared with 45nm Liquid-Silicon.

FPGAs of the same technology in terms of latency and power consumption.

To further corroborate the results above, we replicate the experiment using an 18nm Virtex UltraScale+ series FPGA as the baseline. Since the FPGA at 18nm technology allows for significantly higher clock frequencies compared with 45nm technology, the delay for all test samples is based on Meltrix, enabling a comparison of power consumption and interconnection power ratio under the same delay. The findings presented in Fig. 8 substantiate a 75.7% reduction in power consumption and a 51.1% decrease in interconnection power ratio, thereby affirming the conclusions drawn from the previous experiment. This outcome further signifies the superior performance of Meltrix compared with state-of-the-art FPGAs.

The advanced RRAM-based reconfigurable architecture L-Si will be utilized as a baseline to demonstrate that Meltrix's capabilities are not only derived from emerging device technology (RRAM). As depicted in Fig. 9, employing a synthesis with the "optimal latency" strategy, Meltrix exhibits reductions of 60.9%, 52.2%, and 81.7% in latency, power consumption, and PDP, respectively, highlighting substantial architectural advantages. Furthermore, Meltrix achieves a 52.6% decrease in interconnection power ratio compared with L-Si, indicating that, our function synthesis effectively resolves the problem of the routing overhead in FPGAs and L-Si.

B. Storage Evaluation

This section presents a performance test conducted on the memory modules of Meltrix, comparing the EMBs of two different FPGAs' and L-Si' memory modules mentioned previously. For this experiment, we choose a memory controller and a memory merge procedure with various capacities of single-port RAMs for testing under the same latency. It is important to note that the EMB unit capacity of Xilinx series FPGAs is 18kb, while the storage mode of Meltrix and L-Si has a single tile capacity of 4kb.

Fig. 10 illustrates the power consumption of the single-port RAM modules in the memory test samples, the total PDP and the interconnection ratio. Clearly, there is a gradual increase in the power consumption of Meltrix as the capacity rises, indicating a linear relationship. As the test capacity does not exceed the unit capacity of EMBs, the power consumption utilized by the FPGA remains basically unchanged. However, even in the 16kb test nearing the EMB unit capacity, Meltrix maintains a power advantage of over 56.3% compared with FPGA. When comparing with L-Si, Meltrix exhibits power reductions ranging from 17.6% to 31.3% across all test samples. Moreover, Meltrix has at least 65.9% and 41.1% reductions in PDP and interconnection ratio, respectively, compared with FPGA, which presents a superior performance cased by high energy efficiency and flexibility of the storage modules. Similar to **bigkey** in Section V-A, the interconnection power ratio in **ch_intrinsics 4k** is also 0, indicating that Meltrix has a exceptional ability in reducing global routing.

Based on the experiment above, two distinct advantages of Meltrix over FPGAs in storage are observed. First, Meltrix offers a larger dynamic range of capacity, as all tiles of Meltrix can be configured in storage mode while FPGAs' capacity is fixed. Additionally, as stated in Ref. [32], a flexible storage layout based on computational requirements consumes less energy compared with a fixed-positioned high-capacity storage layout. Meltrix's storage modules can be arranged in any position, whereas FPGAs lack this capability. Moreover, Meltrix's storage module granularity is smaller, resulting in lower capacity loss when facing varying RAM capacities compared with FPGAs. These advantages effectively address the issues of fixed position and limited capacity present in FPGAs' EMBs.

Furthermore, compared with L-Si, Meltrix not only has a power advantage in all tests, but the number of logic tiles used in each storage module is only 33% of L-Si. This observation suggests that, under similar circumstances, Meltrix requires fewer resources and consumes less power.

VI. Conclusion

By utilizing the storage/switch reconfigurability of RRAM devices, multi-functional crossbar arrays and polymorphic architectures with function reconfigurability can be created. In this paper, we propose a RRAM-based polymorphic architecture Meltrix and the corresponding function synthesis framework. Our evaluations have shown that the hardware architecture of Meltrix combined with the function synthesis process can effectively solve the problems of high routing overhead, storage capacity limitations, and fixed positions of EMBs in conventional FPGAs. This demonstrates the enormous application prospects of Meltrix and also indicates that it is an effective supplement to reconfigurable structures.

References

[1] A. Sahoo and E. S. Ebbini, "Real-time fpga implementation of a second order volterra filter for ultrasound nonlinear imaging," in *2022 IEEE*

International Symposium on Circuits and Systems (ISCAS), 2022, pp. 895–899.

[2] U. Lee, H. K. Kim, Y. J. Lim, and M. H. Sunwoo, "Resource-efficient fpga implementation of advanced encryption standard," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 1165–1169.

[3] Y. Ling, Y. Yan, K. Huang, and G. Chen, "Flowacc: Real-time high-accuracy dnn-based optical flow accelerator in fpga," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 112–115.

[4] M. Zhu, J. Luo, W. Mao, and Z. Wang, "An efficient fpga-based accelerator for deep forest," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 3334–3338.

[5] Y. Shi, Y. Sun, J. Jiang, G. He, Q. Wang, and N. Jing, "Fast fpga-based emulation for reram-enabled deep neural network accelerator," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[6] *7 Series FPGAs Data Sheet: Overview*, Xilinx, 2020. [Online]. Available: https://docs.xilinx.com/v/u/en-US/

[7] *UltraScale Architecture and Product Data Sheet: Overview*, Xilinx, 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview

[8] *Spartan-6 Family Overview*, Xilinx, 2011. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds160

[9] N. Brown and D. Dolman, "It's all about data movement: Optimising fpga data access to boost performance," in *2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2019, pp. 1–10.

[10] Y. Wang, J.-J. Lee, Y. Ding, and P. Li, "A scalable fpga engine for parallel acceleration of singular value decomposition," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 370–376.

[11] A. D. Gunter and S. Wilton, "Towards a machine learning approach to predicting the difficulty of fpga routing problems," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023, p. 231.

[12] D. J. Wouters, L. Zhang, A. Fantini, R. Degraeve, L. Goux, Y. Y. Chen, B. Govoreanu, G. S. Kar, G. V. Groeseneken, and M. Jurczak, "Analysis of complementary rram switching," *IEEE Electron Device Letters*, vol. 33, no. 8, pp. 1186–1188, 2012.

[13] Y. He, Y. Huang, J. Yue, W. Sun, L. Zhang, and Y. Liu, "C-rram: A fully input parallel charge-domain rram-based computing-in-memory design with high tolerance for rram variations," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 3279–3283.

[14] Y. Zha, E. Nowak, and J. Li, "Liquid silicon: A nonvolatile fully programmable processing-in-memory processor with monolithically integrated reram," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 908–919, 2020.

[15] B. Govoreanu, A. Ajaykumar, H. Lipowicz, Y.-Y. Chen, J.-C. Liu, R. Degraeve, L. Zhang, S. Clima, L. Goux, I. Radu, A. Fantini, N. Raghavan, G.-S. Kar, W. Kim, A. Redolfi, D. Wouters, L. Altimime, and M. Jurczak, "Performance and reliability of ultra-thin hfo2-based rram (uto-rram)," in *2013 5th IEEE International Memory Workshop*, 2013, pp. 48–51.

[16] A. Grossi, E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. El Hajjam, R. Crochemore, J. Nodin, P. Olivo, and L. Perniola, "Fundamental variability limits of filament-based rram," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 4.7.1–4.7.4.

[17] X. Tang, E. Giacomin, P. Cadareanu, G. Gore, and P.-E. Gaillardon, "A rram-based fpga for energy-efficient edge computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 144–a–144–f.

[18] N. C. Dao and D. Koch, "Memristor-based pass gate for fpga programmable routing switch," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[19] J. P. Cardoso de Lima, M. Brandalero, and L. Carro, "Endurance-aware rram-based reconfigurable architecture using tcam arrays," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 40–46.

[20] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin ii - an open-source verilog hdl synthesis tool for cad research," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 149–156.

[21] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, and V. Betz, "Vtr 8: High-performance cad and customizable fpga architecture modelling," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, no. 2, pp. 1–55, 2020.

[22] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Proceedings of the 22nd International Conference on Computer Aided Verification*, 2010, p. 24–40.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[24] Y. Zha and J. Li, "Revisiting pathfinder routing algorithm," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2022, p. 24–34.

[25] *METIS*, Karypis Lab, 2023. [Online]. Available: https://github.com/KarypisLab/METIS

[26] A. DeHon, "Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% lut utilization)," in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, 1999, p. 69–78.

[27] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven titan: Enabling large benchmarks and exploring the gap between academic and commercial cad," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 2, mar 2015.

[28] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.

[29] *Vivado ML Edition*, Xilinx, 2023. [Online]. Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html

[30] *PrimeSim HSPICE*, Synopsys, 2023. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-hspice.html

[31] B. Hajri, H. Aziza, M. M. Mansour, and A. Chehab, "Rram device models: A comparative analysis with experimental validation," *IEEE Access*, vol. 7, pp. 168 963–168 980, 2019.

[32] A. M. DeHon, "Location, location, location: The role of spatial locality in asymptotic energy minimization," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2013, p. 137–146.