A Field Programmable Process-In-Memory Architecture Based on RRAM Technology

EEE 280 Advanced Computer Architecture

Author: Adeel Ghumman, Nitin Nath, Neha Gour

May 04, 2023

Abstract:

Traditional reprogrammable architectures like FPGA face limitations in terms of on-chip memory as well as limited memory resources which make it difficult to handle data-intensive applications. Specifically, the indicators under evaluation are power, access delays, and power delay products. Developing nonvolatile devices like Resistive-Random Access Memory (RRAM) can operate in a logic-in-memory way, that is they may function as both nonvolatile memory and a switch as a result they can be more flexible and capable of handling more data. Our proposal is an FPGA logic-in-memory architecture based on RRAM technology. It is composed of a 2D tile array, and the tile can be transformed into three fundamental modes which are logic, memory, and interconnects. The architecture overview will make an illustration to show the hardware changes or enhancements as well as any necessary software.

Table of Contents

List of Figures

1. Introduction:

Advancements in Technology drive the use of edge computing for compute-intensive tasks, which requires heterogeneous hardware systems to be more capable in data processing. Reconfigurable architecture has been prominent in the edge computing system due to its versatility in hardware implementation. Field-Programmable Gate Arrays (FPGAs) are the standard representative of reconfigurable architecture which have significantly expanded the number of logic cells available for use in edge computing. However, memory resources have become severely constrained in data-intensive applications as it is placed in a specific location in block random-access memory (RAM) which limits the flexibility of memory access [1]. The data-intensive operations utilize almost all of their processing time on handling the data and I/O which leads to an excessive amount of memory access. FPGAs insufficient storage resources are going to restrict their ability to handle data-intensive tasks in modern days [1].

FPGA platform includes Block RAM, DSP slices, PCI Express support, and programmable fabric [2]. Programmable interconnects are used for connecting the CLBs to I/Os and block RAM. But, more than 80% of the area, delay, and power consumption can be attributed to the routing components [3]. This Routing cost may become more problematic due to the local memory architecture of FPGAs, which is based on Static Random-Access Memory (SRAM), made up of six transistors, volatile, and have large power consumption [4]. Additionally, SRAMs lose configuration data when the power is turned off. As a result, after turning on, SRAM-based FPGAs require an initialization to transfer config data from the external nonvolatile memory (NVM) to internal SRAMs. Moreover, one of the major problems with SRAM-based FPGAs is their high standby leakage power.

To address the limitation of a local memory architecture in conventional FPGAs introduced Non-Volatile Memory (NVMs) technology specifically Resistive Random-Access memory (RRAM) technology because of higher integration density, low power consumption, high switching speed, small cell size, high resistance ratio, and low switching voltage (< 1V) as a potential substitute of SRAM configuration [5]. Various studies reported that RRAM has recently been widely used to replace the SRAM cells either in LUTs or routing devices in FPGA design to achieve high-performance and low-power FPGAs [1].

We proposed an FPGA memory architecture based on RRAM technology in this work. We utilize the two-dimensional crossbar array of identical tiles where every tile can be configured into one of three modes: Logic mode, Memory mode, and Interconnect mode. Additionally, we brief about the new placement and route algorithm which not only exploits the characteristics of logic-in-Memory architecture but also reduces the PDP as a result [1].

2. Field Programmable Gate Array (FPGA):

In this section, we will discuss the architecture of a traditional Field-Programmable Gate Array (FPGA) and related concepts. An FPGA consists of three basic elements which are the configurable logic block (CLB) which is also called Logic Array Block (LAB), the programmable interconnection, and the programmable I/O blocks or elements. All these three elements are programmable and constitute the programmable fabric of an FPGA. Figure 2.1 shows the basic fabric of an FPGA. The fourth element, Block RAM, which is not programmable serves the memory needs of the CLB and I/O blocks and is embedded as a fixed memory block. [6]

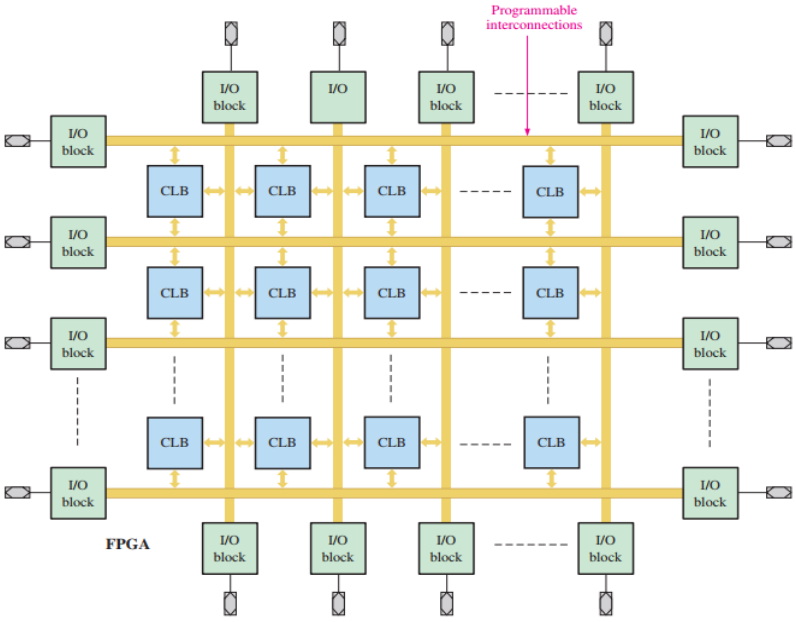

Figure 2.1: Basic Fabric of an FPGA

FPGAs could come in two different types considering their programmability. The first type of FPGA uses SRAMs to implement their configurable logic blocks to lose their configuration data, i.e., volatile when power is turned off but reprogrammable. The second type uses anti-fuse process technology to implement the logic blocks and interconnections. The second type is only one-time

programmable and is only used for batch implementation and is not very common. SRAM-based FPGAs use an embedded nonvolatile configuration memory embedded on the chip to store the configuration data and each time on powering on the data is loaded onto the fabric. Figure 2.2 shows the FPGA using an embedded non-volatile memory for configuration data. [7]

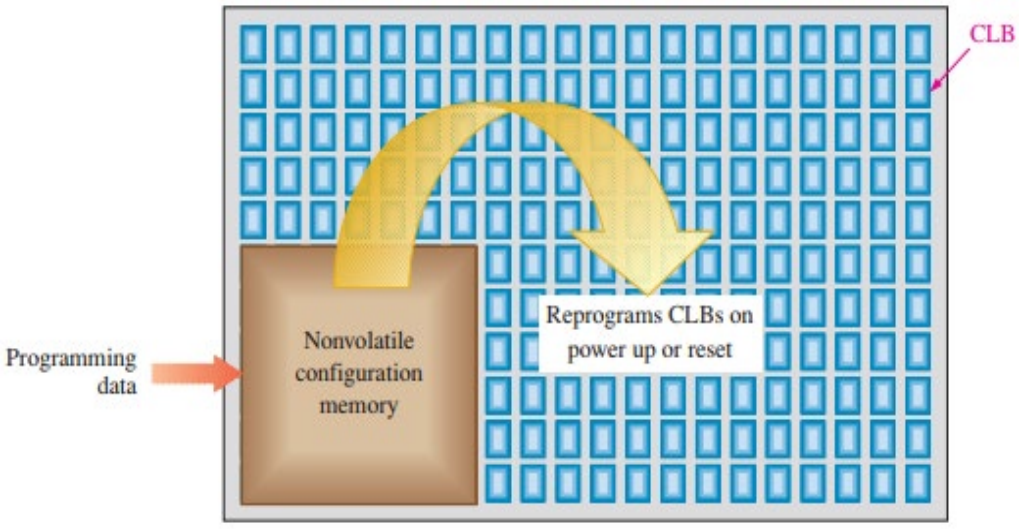

Figure 2.2: SRAM-based FPGA with non-volatile configuration memory

In addition to the basic FPGA fabric and Block RAM, many different dedicated physical hardware blocks are included in an FPGA. These blocks, called embedded functions, mostly consist of DSP blocks, multiplier blocks, PLL blocks, and SerDes transceivers. These blocks are implemented to enhance performance. It is possible to implement these embedded function blocks using logic blocks as well. [7]

The CLBs could also be used as memory blocks when some logic blocks are not used for the logic and the memory need exceeds the capacity of Block RAM. This is called Distributed RAM. Figure 2.3 shows a block diagram of an FPGA along with distributed RAM blocks and DSP blocks.

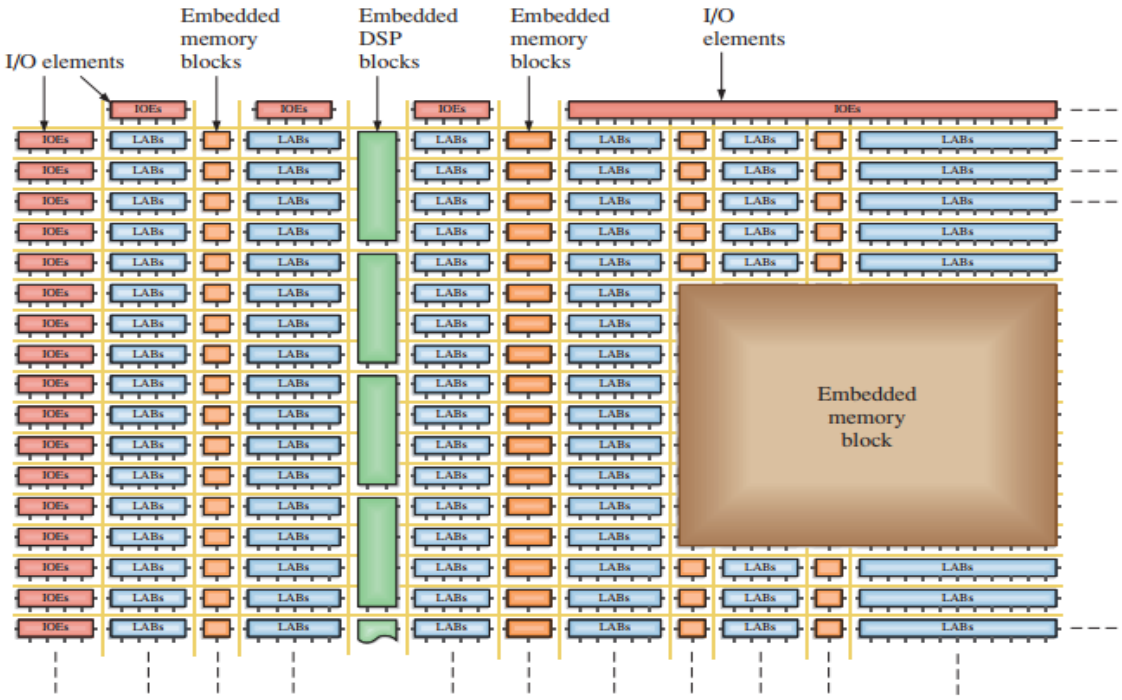

Figure 2.3: Block diagram of FPGA with memory block

Different vendors could implement CLBs or LABs in a different fashion. We will use a basic configuration for illustrative purposes. A CLB consists of lookup tables (LUTs) and associated logic. This associated logic with very limited programmability mostly consists of flip flops, full adders, and multiplexers. LUTs are at the heart of FPGA programmability. These are implemented using SRAMs and provide combinatorial logic. Figure 2.4 shows a generic CLB along with LUT and the associated logic. The figure also shows the combinatorial logic being implemented by 3-input LUT. [8]

Using the very limited programmability of the associated logic, a CLB could be used in different modes such as normal mode, arithmetic mode, and extended LUT mode. [6]

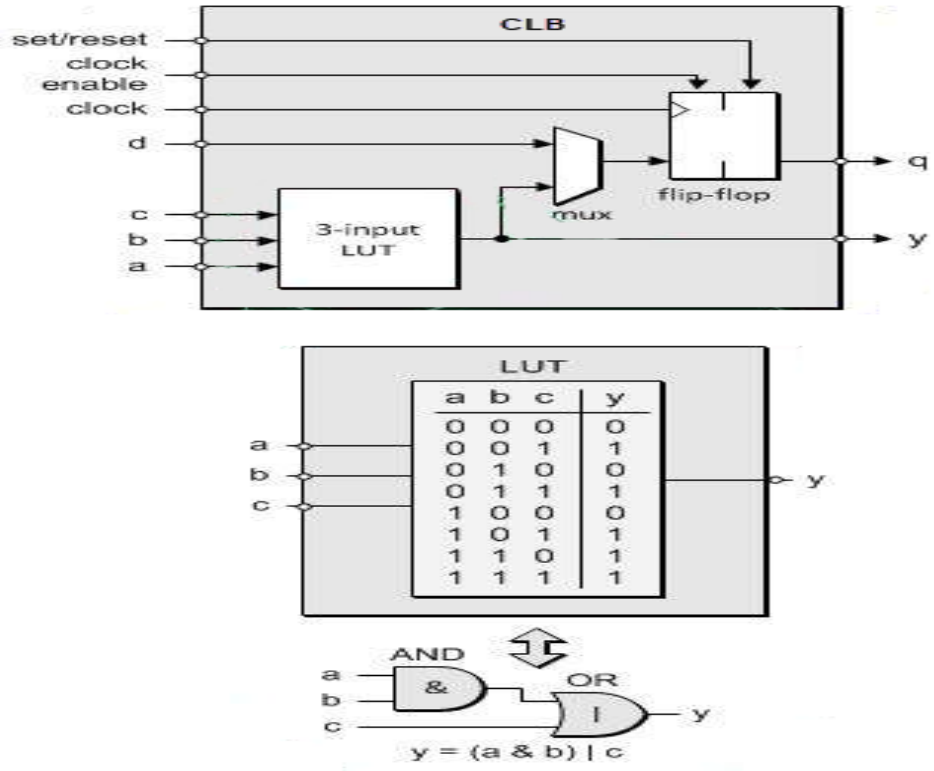

Figure 2.4: Generic CLB

The programmable interconnections are used to interconnect CLBs with I/O blocks and other available blocks. These interconnections are arranged in rows and columns as shown in Figure 1. A very basic implementation of interconnect matric could consist of multiplexers and pass transistors controlled via the bit steam in configuration data. The interconnections could offer different routing based on user-defined path optimization parameters and timing constraints. [9] The programmable I/O blocks at the edges of the FPGA allow selective access to the external environment for input, output, or bidirectional purposes. The I/O blocks can handle diverse signal standards, such as LVCMOS, HSTL, and LVDS, among others, and offer both single-ended and differential options. [9]

Selectable drive strengths are also available in many I/O blocks, while tristate control pins allow for the configuration of pins as inputs, outputs, or both. Additionally, I/O blocks may have pullup

or pulldown options. Moreover, I/O pins feature adjustable input and output delay circuits, with some even having built-in serializers and de-serializers for high-speed signaling. Different I/O blocks could be provided with different clock frequencies using PLL blocks. [8]

The fabric of an FPGA is like a blank slate that can be used by an end user to program it for any logic. As was mentioned earlier, FPGAs come with embedded extra hardware blocks to enhance performance. The hardware block which has been put there by the manufacturer is called hardcore. One benefit of using the hard-core approach is the potential for reducing the FPGA's available capacity usage, resulting in a smaller chip footprint and decreased development time for the user. However, the drawback of the hard-core method is that the specifications are pre-set during manufacturing, and the end user cannot modify any of the specifications. Hard cores provide functions such as DSP units, standard I/O interfaces, and even microprocessors. [7]

Many of the hardcore logic blocks are also provided as third-party IP blocks which could be implemented by the end user with custom specifications. These IP blocks are called soft cores and their implementation consumes the fabric of an FPGA. Softcore libraries are extensively used since the user can customize the design according to the requirements. The most used third-party logical components (soft cores) are CPUs (e.g., MicroBlaze), and external interface controllers (e.g., ethernet, USB, I2C/SPI, HDMI, memory controller, video processing. [7]

3. Resistive Random-Access Memory (RRAM) Technology:

In this section, we will discuss the makeup of RRAM technology and its potential benefits. RRAM is the short form of Resistive Random-Access Memory. [1] We know from this course and other related coursework that RAM (Random Access Memory) can either be a volatile type of memory where data stored in this type of memory is lost when the device using this memory is powered off or it can be non-volatile where data stored in the memory is retained after power from the memory is removed. Comparatively, we also know that SRAM and DRAM are both volatile types of memory. Resistive Random-Access Memory is a non-volatile type of RAM with advantages in faster IOs, lower cost, lower power consumption, and scalability. [12] Memory technologies such as SRAM, DRAM, and Flash are charge storage types of memories. [13] Each of these memories is designed to store charge in some form. One of the key hurdles of these types of charge-based memories is the inability to scale down the size of these memories in nanometers, mainly due to their substrate-level properties. [13] It is due to this reason that scalability becomes an issue with the storage of charge-based memories. Consequently, once a bottleneck is observed in mainstream memories, there is reason and motivation to pursue other emerging memories such as RRAM. RRAM consists of resistance-switching memory cells and this resistance-switching occurrence makes it viable for binary on or off states. [13]

What is RRAM composed of? The RRAM cell has an insulating layer that sits between two Metal electrodes. *Figure 3.1* [13] The choice of electrode material varies. There are five categories of electrodes based on their composition, elementary substance electrodes, silicon-based electrodes, alloy electrodes, oxide electrodes, and nitrate-based electrodes. Some of the commonly used elementary substance electrodes are Aluminum, Titanium, and Silver whereas silicon-based electrodes are p-type and n-type Silicon. Alloy electrodes that are utilized for

stabilizing the resistive switching of RRAM cells are Copper Titanium, Copper Tellurium, and Platinum Aluminum. [13] How does the RRAM cell function? An application of a voltage through the RRAM cell allows the cell to transition from a high resistive state (HRS) or OFF to a low resistive state (LRS) ON. [13]

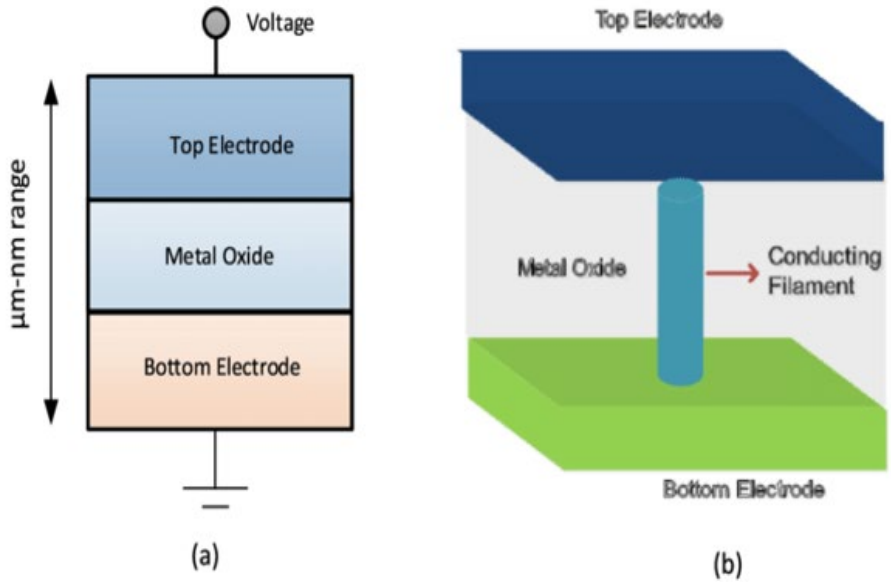

Figure 3.1. RRAM Technology

This is what is described as resistance switching mode. Where the logical 1 and logical 0 can be distinguished at a cell level. When this RRAM cell is prepared it is initially in the high resistive state (HRS) and to switch it to the low resistive state (LRS), a voltage pulse is applied which forms a conductive path in the switching layer and the RRAM cell transitions to LRS. [13] Once the RRAM cell is in LRS it needs to be able to switch back to HRS mode. The transition from LRS to HRS is accomplished by applying a voltage pulse or RESET voltage. [13] This completes the switching cycle from HRS to LRS then back to HRS. It is because of this property RRAM is considered a different type of RAM memory than SRAM or DRAM. Next, we investigate how the data is read and how the RRAM memory is non-volatile. To read the data

from the RRAM cell a small read voltage is applied to determine whether the RRAM cell is in HRS or LRS; the applied read voltage does not alter the RRAM cell state and additionally since the HRS and LRS retain their states after the removal of the applied voltage, the memory is therefore non-volatile. [13] Since RRAM can switch very fast it allows for faster IO speeds.

4. Proposed Architecture Overview:

In this section, we will discuss the utilization of RRAM technology in FPGAs which are used to build flexible modules. These modules behave as building blocks of FPGA such as logic, interconnection, and memory that are merged into a unified array forming the core of ME stands for Merge Everything. These arrays can be configured to logic mode, interconnection mode, or memory mode where it can process computation and data-intensive tasks. [14] When compared with SRAM-based FPGAs this architecture provides greater flexibility than the SRAM-based architectures.

The ME architecture is an array with integrated identical tiles. This tile is the foundation of the ME architecture. It contains N to 1 LUT multiplexers, N sense amplifiers, some remoldable multiplexers as well as other components. The unified crossbar array forms a N x N memory matrix which can also be used as an N-to-N routing switch. The tile modes of operation are determined by the moldable MUXes which are also implemented by RRAMs. Just like when the FPGAs are programmed the remoldation process of ME is done prior to any execution. The modes of operation do not change during runtime and the RRAM operates in binary ON or OFF.

How do the three modes of operation work? In the Logic mode, the LUTs are used as reconfigurable logic blocks. There are access transistors, word lines, and sense lines that are

disabled. There are bit lines that are grounded and the output is fed back to the input of the MUXes of the LUTs which allows the tile to support up to N levels of cascaded logic. [14]

In interconnection mode, in this mode one RRAM is ON in each row and in each column. This forms at most N routing paths. The input signal DIN goes through the Bit line, a T1R cell, a sense line as well as a sense amplifier. In the sense of amplifiers, the clock is disabled, and the latches are bypassed. The sense amplifiers are used to restore the voltage drop caused by the access transistors (Ats). [14]

In Memory mode, the crossbar array is effectively the memory array. The array accepts DIN which is the input data, RADDR which is the row address, CADDR which is the column address, REN which is read enable, WEN which is write enable, and DOUT which are the outputs of the crossbar array. In its initial state, the BLs (Bit Lines) are in the grounded mode. During a read operation, the output is sensed from the sense lines. The sense amplifiers function the same way as they do in the logic mode where the latch maintains the output during the pre-charge (clk is 0). However, during evaluation (clk = 1) if the RRAM is ON then the system will pull down the input voltage of the sense amplifier and the latch output is high. Consequently, if the RRAM is OFF the latch output stays low. [14]

This is the unified tile architecture of the FPGA that is proposed and clearly, there is a lot of flexibility built in. Specifically, the three modes of operation make this architecture very flexible.

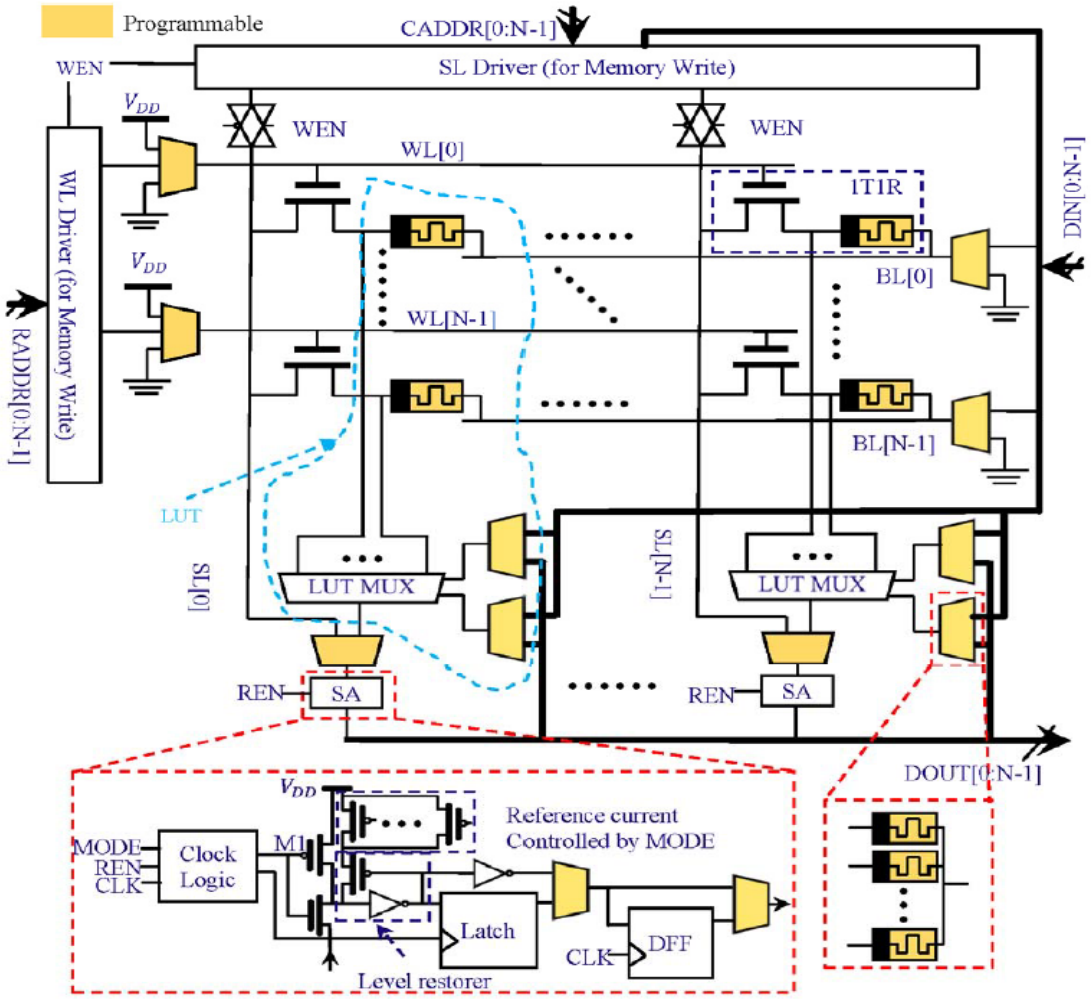

Figure. 4.1 Field programmable Logic-in-Memory Architecture

5. Method:

The logic and interconnect resources in our architecture are programmable, however, the routing resources in the switching and connection blocks are fixed. As a consequence, we'll need to come up with our own version of the place-and-route tool [1].

In this, the design achieves interconnect structure within the tile, in which route and placement can carry out in a similar manner. To develop a new algorithm, we have to keep two things in our

mind. First is cost function value should be directly proportional to the utilization of tiles. Second, the new cost function should ensure that non-critical path logic and interconnect resources are placed in tiles with particular usage to ensure a low total tile utilization [1]. When the tile is fully utilized, the cost function should be set to an extremely high value to ensure that the tile is no longer used. [1]

We create a place-route algorithm that can perform "Adaptive Resource Partitioning" while keeping to the two limitations characterized above. The approach may provide high-quality mapping results by fine-grained partitioning of tiles into logic and interconnecting resources.[1]

In the proposed architecture new algorithm" Adaptive Resource partitioning" works in a way we can perform routing and placement concurrently [1]. In every new iteration, netlist will generate a different routing path as per the given placement. So, we can get the estimated utilization of logic and interconnect tiles. As a consequence, partitioning can be fine-grain controlled so can change as per utilization on a per-tile basis [1].

To achieve the algorithm, we introduced the cost function for a logic primitive I as [1]:

$$\textbf{Cost} = (C - A(i)) * P(i) + C * R(i)$$

Where,

C – length of critical path

A(i)- Length of a longest path that contains the logic primitive i

P(i)- Placement cost of the logic Primitive

R(i)- the cost of routing

Based on the first constraint, we define P(i) as the function of the use of the tile containing the logic primitive I. P(i) is the reciprocal of the number of logic and connection resources used in the current tile when it is less than 100% used. Otherwise, it's a big plus. Finally, we defined R(i) as the average length of routing paths [1].

When the term C- A(i) is not zero, logic primitives are more likely to be placed in tiles with specific utilization on non-critical paths that satisfy the second restriction [1]. Otherwise, it optimizes the delay on the critical path. The total cost is the sum of all logic primitive costs [1].

6. Experiment Results:

We used the Xilinx Virtex-6 FPGA architecture as the foundation of the study and evaluated the proposed architecture at the same technology node. The new placement-route tool VTR is utilized for logic primitive placement and routing [1]. All the required numbers for Virtex-6 such as power delay have been taken from the given datasheet.

In this proposed architecture the overall power consumption may be determined by multiplying the total number of used tiles by the power of each tile in benchmark evaluations on our design due to the homogeneous nature [1].

6.1 Power:

Power of the proposed architecture in Figure 7(a) is compared with Virtex-6 and observed the reduction by 1.9x on average due to the reconfiguration of logic and interconnects. By using the place and route algorithm we can define the resource allocation which reduces the power consumption during the transmission of data [1].

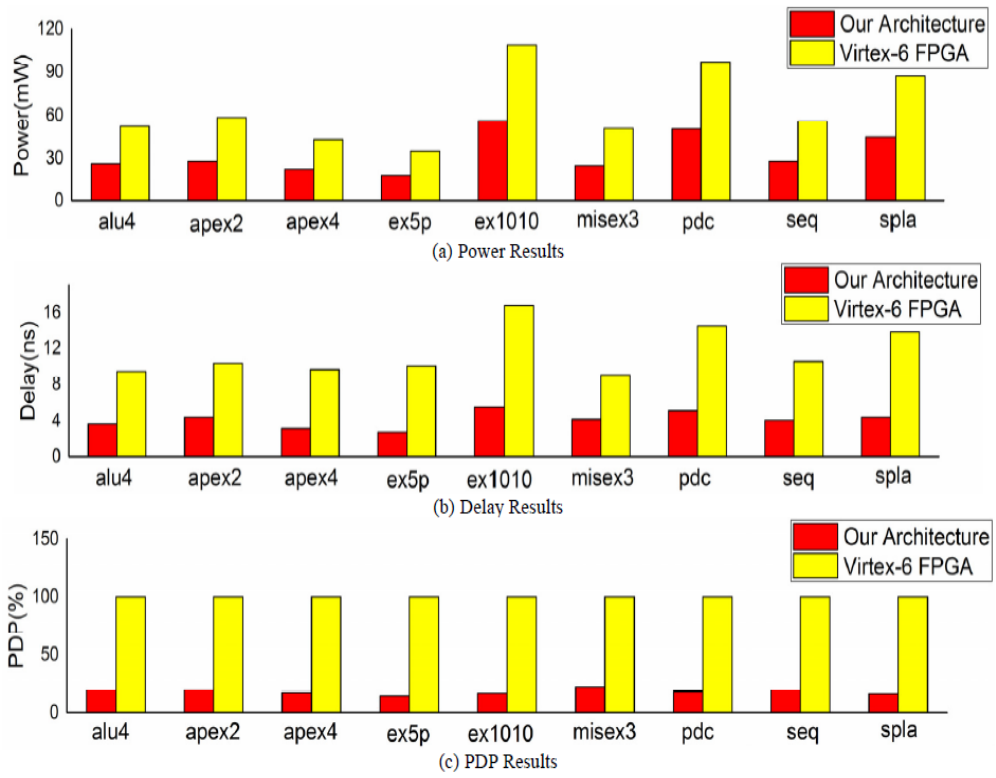

Figure: 6.1: Power, Delay and PDP products

6.2 Delay:

The flexibility of the proposed architecture provides a better use of hardware resources. Additionally, during the annealing process, logic and interconnect keeps on the same tile which also impacts the reduction in the length of wire, hence the reduction in delay as shown in figure 7(b). [1]

6.3 Power Delay Product:

Due to the flexibility in the logic and interconnect and adaptive resource algorithm proposed architecture compare to benchmark architecture not only reduces hardware resources which results in a reduction in power consumption but also reduces the delay because of the Routing length. And as per Figure 7(c) PDP results have improved in new architecture by 5.6x. [1]

7. Conclusion:

In this work, we studied RRAM-based FPGA memory architecture and discuss the new “Adaptive resource partitioning” place and route algorithm. Moreover, compare the proposed architecture with Xilinx FPGA Virtex-6 and concluded as per the experiment results that reduction in power consumption and delay improves the significant performance in the RRAM based logic-In-Memory Architecture, which is an efficient solution for data-compute applications such as Weather forecasting.

8. References:

[1] Y. Liang, L. Yin, and N. Xu, "A Field Programmable Process-In-Memory Architecture Based on RRAM Technology," 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE), Harbin, China, 2020, pp. 2323-2326

[2] SDAccel Development Environment Help, Understanding FPGA Architecture

[3] J. Cong and Bingjun Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," 2011 IEEE/ACM International Symposium on Nanoscale Architectures, San Diego, CA, USA, 2011, pp. 1-8, doi: 10.1109/NANOARCH.2011.5941476.

[4] Ian Kuon, Russell Tessier and Jonathan Rose, "FPGA Architecture: Survey and Challenges", 2008 Foundations and Trends® in Electronic Design Automation: Vol. 2: No. 2, pp 135-253. doi: http://dx.doi.org/10.1561/1000000005

[5] Abinaya. S, Gayathri. J, Giridharan. S, Swaminathan. M. "Nonvolatile field-programmable gate array with high-reliability and high-intensity using1D2R RRAM array", 2018, International Journal of Advance Research, Ideas, and Innovations in Technology

[6] Field-Programmable Gate Array, Wikipedia

[7] Designing with Xilinx® FPGAs: Using Vivado 1st ed. 2017, Sanjay Churiwala https://link.springer.com/book/10.1007/978-3-319-42438-5#bibliographic-information

[8] Park, Hyunbin, and S. Kim. "Hardware accelerator systems for artificial intelligence and machine learning." 2021 *Advances in Computers*. Vol. 122. Elsevier, 2021. 51-95.

[9] Introduction to Reconfigurable Supercomputing, Marco Lanzarote, Stephen Bique, Robert Rosenberg, 2010 https://citations.springernature.com/book?doi=10.1007/978-3-031-01726-1

[10] The MCU guy's introduction to FPGAs: Hardware, https://www.embedded.com/the-mcu-guys-introduction-to-fpgas-the-hardware/

[11] Digital Design, 5th ed, Morris Mano

[12] A. Sai Kumar, N. Neelima, B. Seetharamulu, N. Suresh, S. Siva Priyanka, "A Novel RRAm-based FPGA architecture with Improved performance and Optimization Parameters," 2022 IEEE 19th India Council International Conference (INDICON), Kochi, India, 2022, pp. 1-5, doi: 10.1109/INDICON56171.2022.10040133

[13] Zahoor, F., A. Zulkifli, T.Z. & Khanday, F.A. Resistive Random-Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (mlc) Storage, Modeling, and Applications. Nanoscale Res Lett 15, 90 (2020). https://doi.org/10.1186/s11671-020-03299-9

[14] X. Chen, L. Yin, B. Liu, and Y. Han, "Merging Everything (ME): A Unified FPGA Architecture Based on Logic-in-Memory Techniques," 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, pp. 1-2.

Bibliography:

[1] Jim Ledin, “Architecting High-Performance Embedded System” Packt Publishing, February 2021,

Figures:

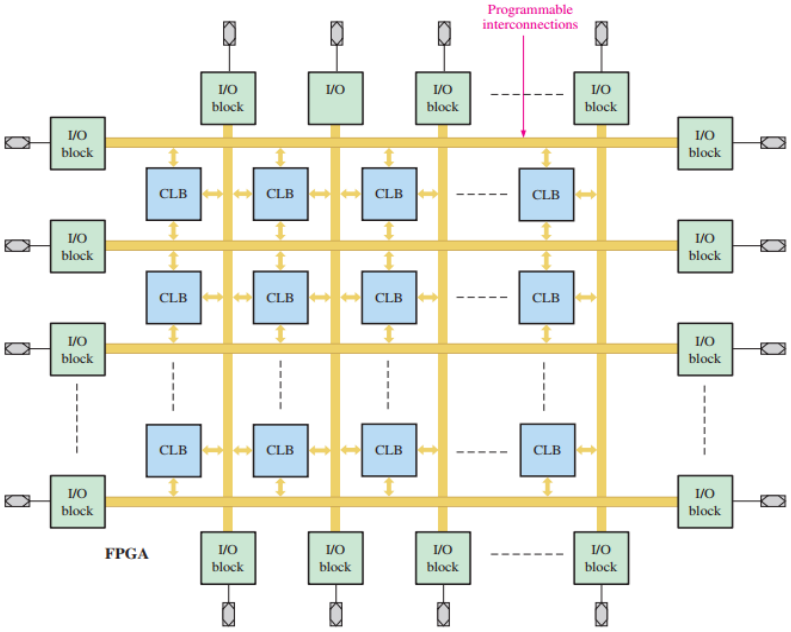

Figure 2.1: Basic Fabric of an FPGA [11]

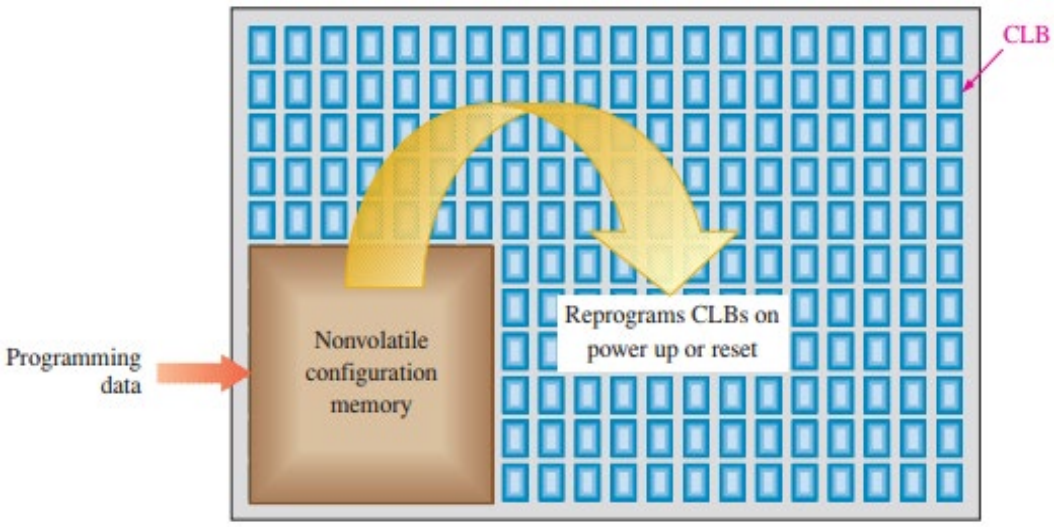

Figure 2.2: SRAM-based FPGA with non-volatile configuration memory [11]

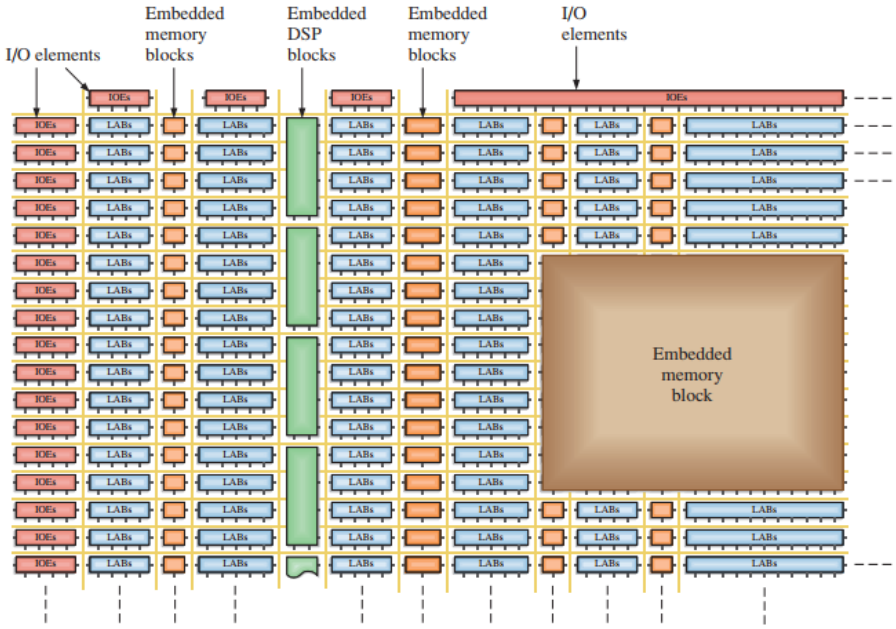

Figure 2.3: Block diagram of FPGA with memory block [11]

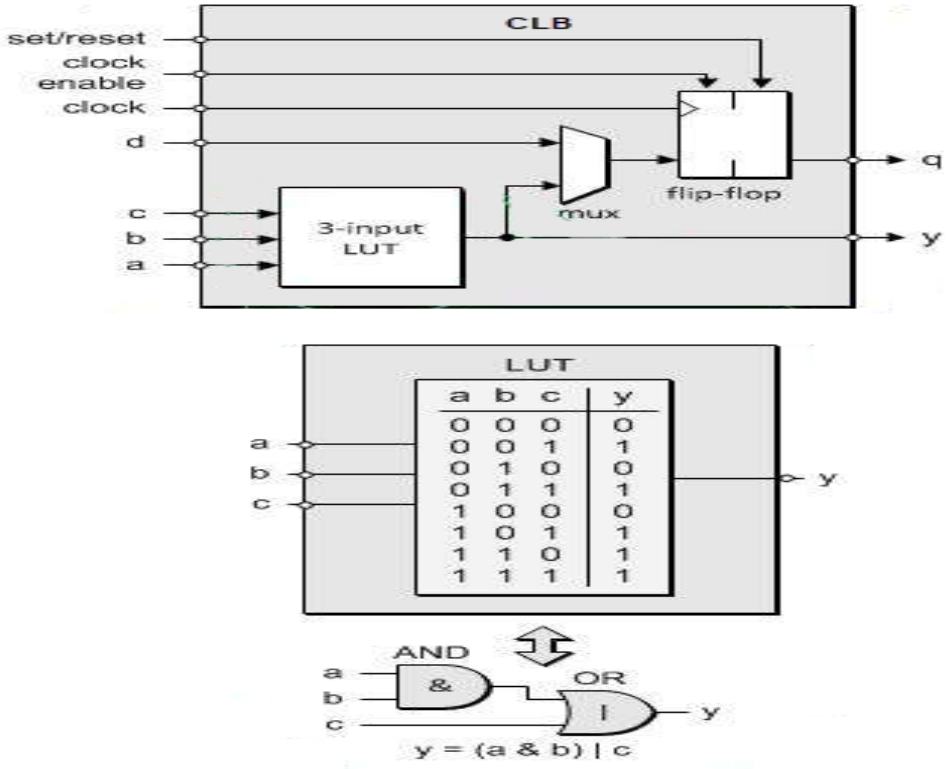

Figure 2.4: Generic CLB [10]

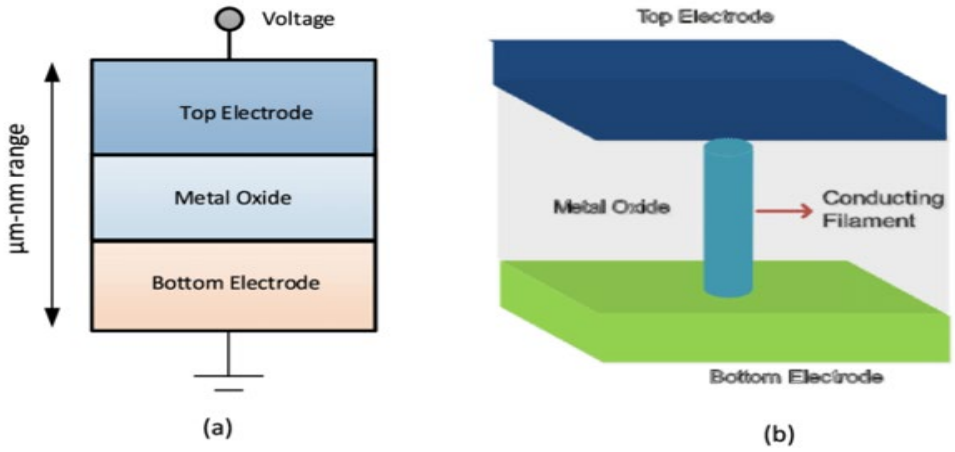

Figure 3.1. RRAM Technology [13]

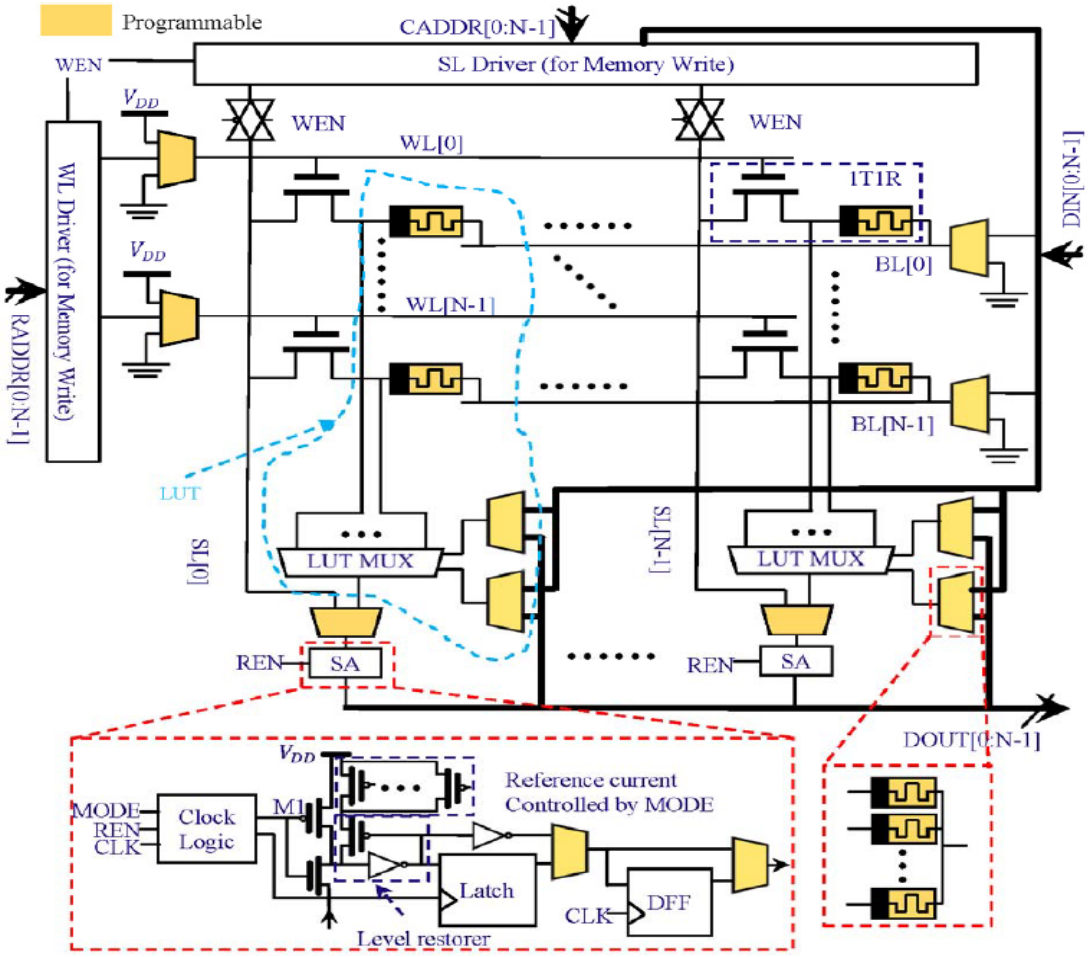

Figure. 4.1 Field programmable Logic-in-Memory Architecture [1]

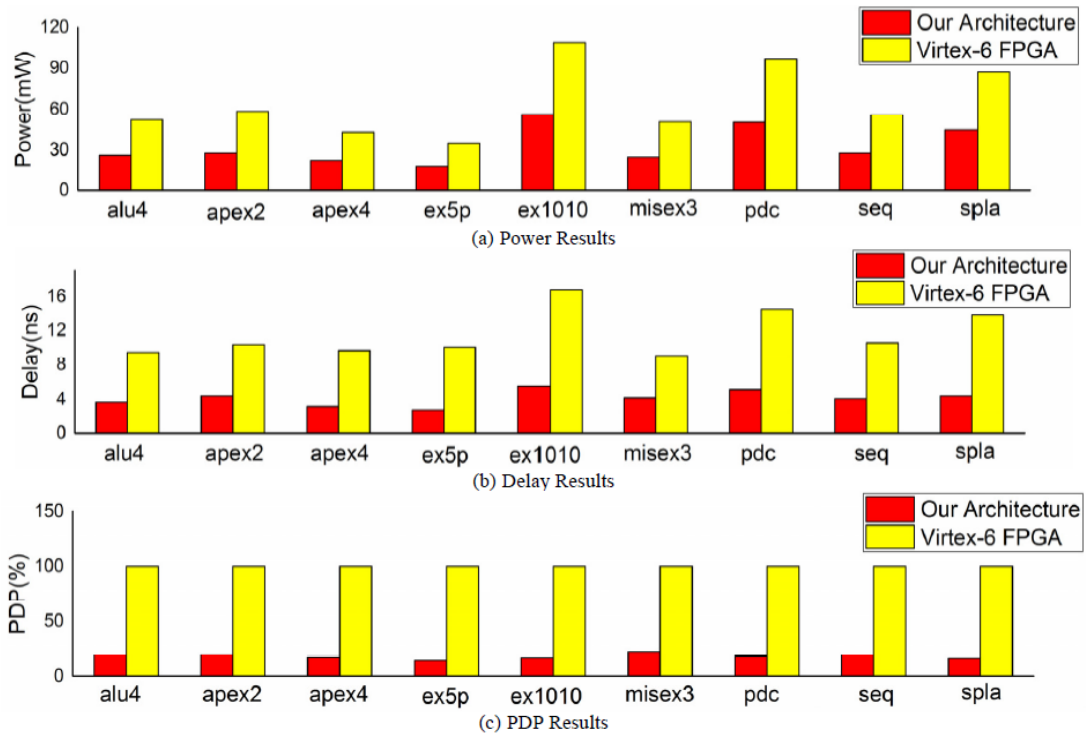

Figure: 6.1: Power, Delay and PDP products[1]