

## Chapter 1

# PROJECT OVERVIEW

### 1.1 Introduction

The proposed idea for our Main Project for the final year (2015-2016) is Systolic Processor Architecture using FPGA. A systolic array is a homogeneous network of tightly coupled Data Processing Units (DPUs) called cells or nodes. Each node or DPU independently computes a partial result as a function of the data received from its upstream neighbors, stores the result within itself and passes it downstream.

They are sometimes classified as Multiple Instruction Single Data (MISD) architectures.

The parallel input data flows through a network of hard-wired processor nodes, resembling the human brain which combine, process, merge or sort the input data into a derived result. Because the wave-like propagation of data through a systolic array resembles the pulse of the human circulatory system, the name systolic was coined from medical terminology. The name is derived from Systole (medicine) as an analogy to the regular pumping of blood by the heart. A major benefit of systolic arrays is that all operand data and partial results are stored within (passing through) the processor array. There is no need to access external buses, main memory or internal caches during each operation as is the case with Von Neumann or Harvard sequential machines.

Systolic arrays are therefore extremely good at artificial intelligence, image processing, pattern recognition, computer vision and other tasks which animal brains do so particularly well.

## **1.2 Objectives**

The main objective of our Project is to :-

- Study on Systolic Architectures.
- To design and Implement an n-bit Processor using Systolic Array Architecture.

## **1.3 System Specifications**

- Virtex 4 – FPGA
- Power Supply 5V, 2A
- Serial Communication

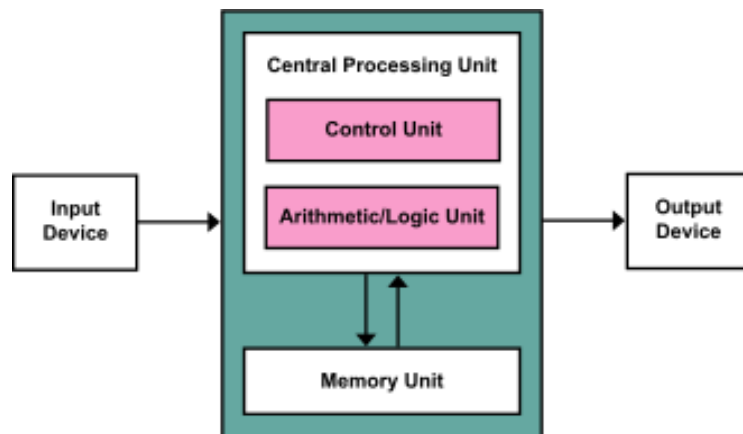
## Chapter 2

### LITERATURE REVIEW

This section consists of a detailed study of Systolic Processors and its comparison with the existing processing architectures like Harvard and Von Neumann.

#### 2.1 Comparison with Von Neumann Architecture

This design architecture consists of a processing unit containing an arithmetic logic unit and processor registers, a control unit containing an instruction register and program counter, a memory to store both data and instructions, external mass storage, and input and output mechanisms. The meaning has evolved to be any stored-program computer in which an instruction fetch and a data operation cannot occur at the same time because they share a common bus.



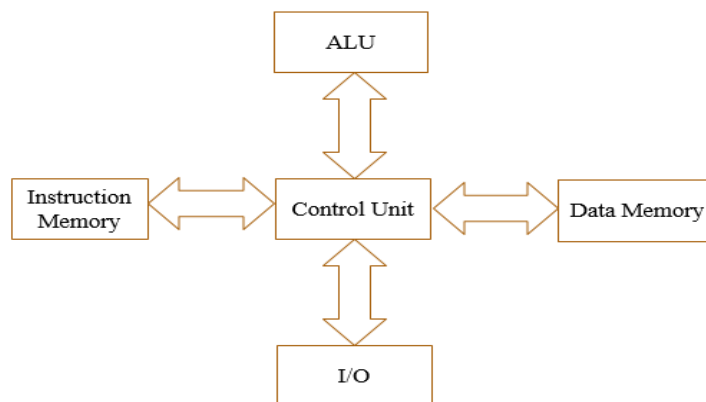
*Figure 2.1 :- Von Neumann Architecture*

### 2.1.1 Limitations and Overcoming Methods

The main limitation of the von Neumann architecture is known as the "von Neumann bottleneck". This is due to the fact that all instructions and all data must pass through the same shared common multiplexed bus to get in or out of the processor, sooner or later things have to wait for other things to get access to this multiplexed bus and the processor gets starved for instructions and/or data. The result is the processor is unable to maintain its designed performance but waits idle instead of doing work. A major benefit of systolic arrays is that all operand data and partial results are stored within (passing through) the processor array. There is no need to access external buses, main memory or internal caches during each operation as is the case with Von Neumann or sequential machines.

## 2.2 Comparisons with Harvard Architecture

The Harvard architecture is a computer architecture with physically separate storage and signal pathways for instructions and data. These machines had data storage entirely contained within the central processing unit, and provided no access to the instruction storage as data. Programs needed to be loaded by an operator; the processor could not initialize itself.



**Figure 2.2 :- Harvard Architecture**

### **2.2.1 Limitations and Overcoming Methods**

Free data memory cant be used for instruction and vice- versa. Production of a computer with two buses is more expensive and needs more time. Moreover this implementation requires more number of pins and it is not commonly used. It is possible to access program memory and data memory simultaneously. Typically, code (or program) memory is read-only and data memory is read-write. Therefore, it is impossible for program contents to be modified by the program itself. All these can be overcome using Systolic Architecture.

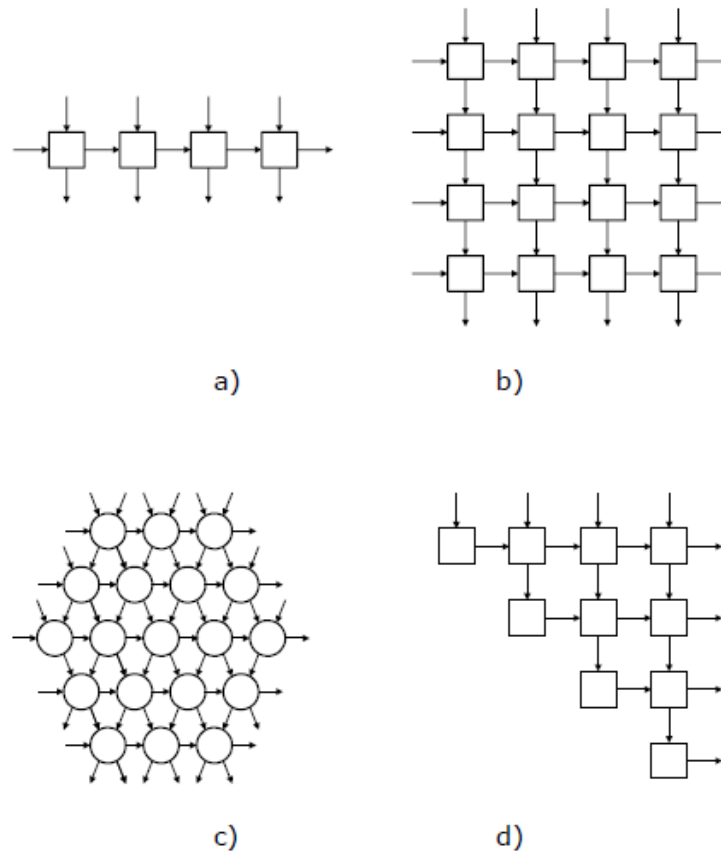
## **2.3 Theory**

The term systolic array was introduced in the computer science by H. T. Kung at the end of 70's. A systolic array typically consists of a large number of similar processing elements interconnected in an array. The interconnections are local meaning that each processing element can communicate only with a limited number of neighboring processing elements. Data move at a constant velocity through the systolic array passing from one processing element to the next processing element. Each of the processing elements performs computations, thus contributing to the overall processing needed to be done by the array. Systolic arrays are synchronous systems. A global clock synchronizes the exchange of data between directly communicating processing elements. Data can be exchanged only at the ticks of the clock. Between two consecutive clock ticks, each processing element carries out computation on the data which it has received upon the last tick and produces data which is sent to neighbouring processing elements at the next clock tick. The processing element can also hold data stored in the local memory of the processing element.

The systolic arrays are usually represented as array of processing elements and array of interconnections connecting the PEs with some particular pattern. In other words, systolic array is a collection of PEs that are arranged in a particular pattern. The pattern and the PEs are defined with the implemented algorithm.

The PEs are composed of combinatorial part and/or memory. The combinatorial part is responsible for arithmetic operations required by the systolic algorithm. By memory

(registers) we denote delay elements within the PE that hold data and thus control data flow into and out of the PE. In general, no large memory is associated with PEs.



**Figure 2.3 :- Examples of Systolic Array**

a) **Linear systolic array** :- Processing elements are arranged in one-dimension. The interconnections between the processing elements are nearest neighbour only. Linear systolic arrays differ relative to the number of data flows and their relative velocities.

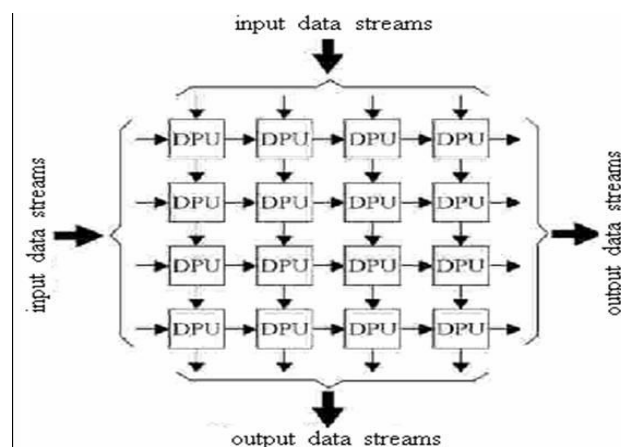
b) **Orthogonal systolic array** :- Processing elements are arranged in a two-dimensional grid. Each processing element is interconnected to its nearest neighbours to the north, east, south and west. Again, the systolic arrays differ relative to the number and direction of data flows and the number of delay elements arranged in them.

c) **Hexagonal Systolic array** :- Processing elements are arranged in a two-dimensional grid. The processing elements are connected with its nearest neighbours where interconnections have hexagonal symmetry.

d) **Triangular Systolic array** :- It refers to two-dimensional systolic array where processing elements are arranged in a triangular form. This topology is mostly used in different algorithms from linear algebra. In particular it is used in Gaussian elimination and other decomposition algorithms.

Systolic systems consists of an array of Processing Elements. Each elements are called cells, and each cell is connected to a small number of nearest neighbours in a mesh like topology. Each cell performs a sequence of operations on data that flows between them. Generally the operations will be the same in each cell. It performs an operation or small number of operations on a data item and then passes it to its neighbor. Systolic arrays compute in “lock-step” with each cell (processor) undertaking alternate compute/communicate phases. A Systolic array is a computing network possessing the following features:

- **Synchrony:-** means that the data is rhythmically computed (Timed by a global clock) and passed through the network.
- **Modularity:-** means that the array(Finite/Infinite) consists of modular processing units.
- **Regularity:-** means that the modular processing units are interconnected with homogeneously.
- **Spatial Locality:-** means that the cells has a local communication interconnection.
- **Temporal Locality:-** means that the cells transmits the signals from from one cell to other which require at least one unit time delay.
- **Pipelinability:-** means that the array can achieve a high speed



**Figure 2.4 :- DPU general structure**

A systolic array is composed of matrix-like rows of data processing units called cells. Data processing units (DPUs) are similar to central processing units (CPUs), (except for the usual lack of a program counter, since operation is transport-triggered, i.e., by the arrival of a data object). Each cell shares the information with its neighbors immediately after processing. The systolic array is often rectangular where data flows across the array between neighbor DPU's, often with different data flowing in different directions. The data streams entering and leaving the ports of the array are generated by auto-sequencing memory units, ASM's. Each ASM includes a data counter. In embedded systems a data stream may also be input from and/or output to an external source. Systolic arrays are arrays of DPUs which are connected to a small number of nearest neighbor DPUs in a mesh-like topology. DPUs perform a sequence of operations on data that flows between them. Because the traditional systolic array synthesis methods have been practiced by algebraic algorithms, only uniform arrays with only linear pipes can be obtained, so that the architectures are the same in all DPUs. The consequence is, that only applications with regular data dependencies can be implemented on classical systolic arrays. Like SIMD machines, clocked systolic arrays compute in "lock-step" with each processor undertaking alternate compute | communicate phases.

A systolic array typically consists of a large monolithic network of primitive computing nodes which can be hardwired or software configured for a specific application. The nodes are usually fixed and identical, while the interconnect is programmable. The more general wavefront processors, by contrast, employ sophisticated and individually programmable nodes which may or may not be monolithic, depending on the array size and design parameters. The other distinction is that systolic arrays rely on synchronous data transfers, while wavefront tend to work asynchronously.

Unlike the more common Von Neumann architecture, where program execution follows a script of instructions stored in common memory, addressed and sequenced under the control of the CPU's program counter (PC), the individual nodes within a systolic array are triggered by the arrival of new data and always process the data in exactly the same way. The actual processing within each node may be hard wired or block microcoded, in which case the common node personality can be block programmable.

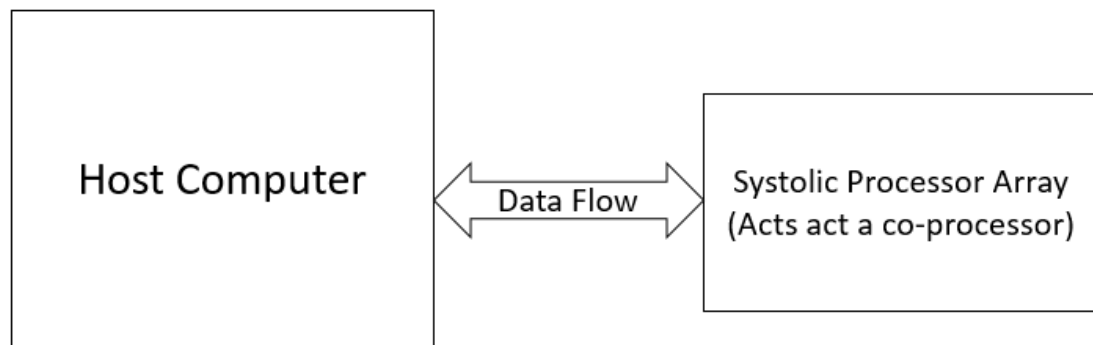


The systolic array paradigm with data-streams driven by data counters, is the counterpart of the Von Neumann architecture with instruction-stream driven by a program counter. Because a systolic array usually sends and receives multiple data streams, and multiple data counters are needed to generate these data streams, it supports data parallelism. The actual nodes can be simple and hardwired or consist of more sophisticated units using micro code, which may be block programmable.

## Chapter 3

### SYSTEM OVERVIEW

The entire system is divided into 3 parts namely Host computer, Communication or Data Flow and the Systolic processor (FPGA).



*Figure 3.1 :- System Overview*

### 3.1 Host Computer

The main system which interacts with the user. The host computer passes on the computationally taxing tasks to the co-processor which is designed to take on such tasks with minimum delay through extensive parallelism and pipelining.

The host computer communicates the input values given by the user to the co-processor through a suitable communication protocol.

Once the task is executed the resultant data is transferred back to the host computer and is given to the user as the output.

## 3.2 Communication

Serial communication protocol is employed here for the purpose of communication between the host computer and the co-processor.

Serial communication is done at 9600 baud rate with 1 stop bit per byte and no parity bits.

At the input side of the co-processor is an input data receive module which takes the serial input converts it into parallel outputs appends the required amount of zeroes into the data using an indigenously made zero append algorithm which interleaves the required amount of zeroes in between the data for individual DPUs.

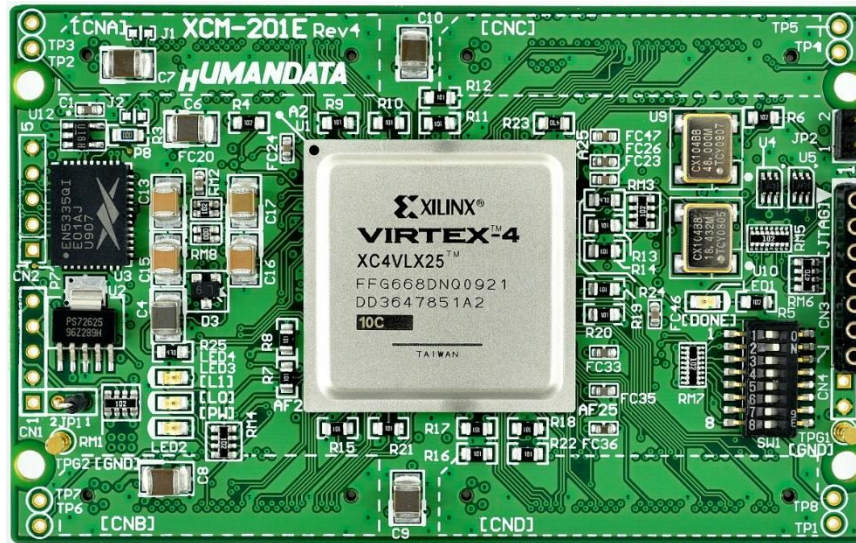
Similarly, at the output side is an output data module which sends the output of each DPU back to the host computer serially using the same specifications for the protocol.

## 3.3 FPGA

Virtex-4 is the FPGA used. They are considered to be legacy device. Virtex is the flagship family of FPGA products developed by Xilinx. Virtex FPGAs are typically programmed in hardware description languages such as VHDL or Verilog, using the Xilinx ISE or Vivado Design Suite computer software.

The Virtex series of FPGAs are based on Configurable Logic Blocks (CLBs), where each CLB is equivalent to multiple ASIC gates. Each CLB is composed of multiple slices, that differ in construction between Virtex families.

Virtex FPGAs include an I/O Block for controlling input/output pins on the Virtex chip, that support a variety of 11signaling standards. All pins default to “input” mode (high impedance). I/O pins are grouped into I/O Banks, where each Bank can support a different voltage. In addition to configurable FPGA logic, Virtex FPGAs include fixed-function hardware for multipliers, memories, microprocessor cores,etc.



**Figure 3.2 :- Virtex 4 FPGA**

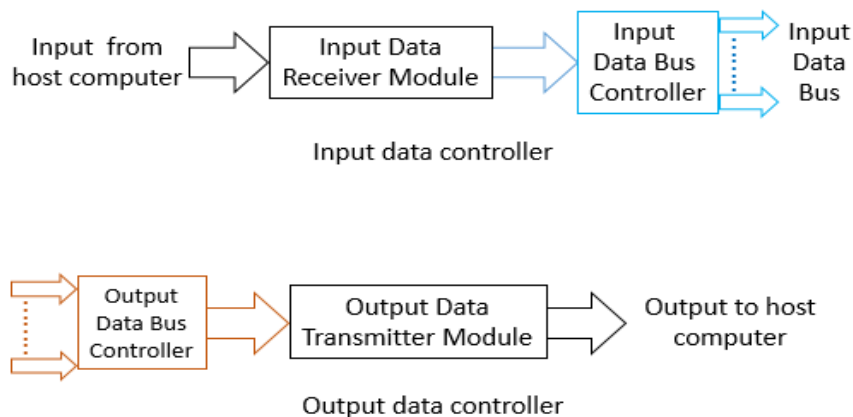
## Chapter 4

### SYSTEM DESIGN

This section deals with the complete information regarding the data flow, functional block diagrams, working of the Systolic array and the hardware description of the logic used.

#### 4.1 Functional Block Diagram

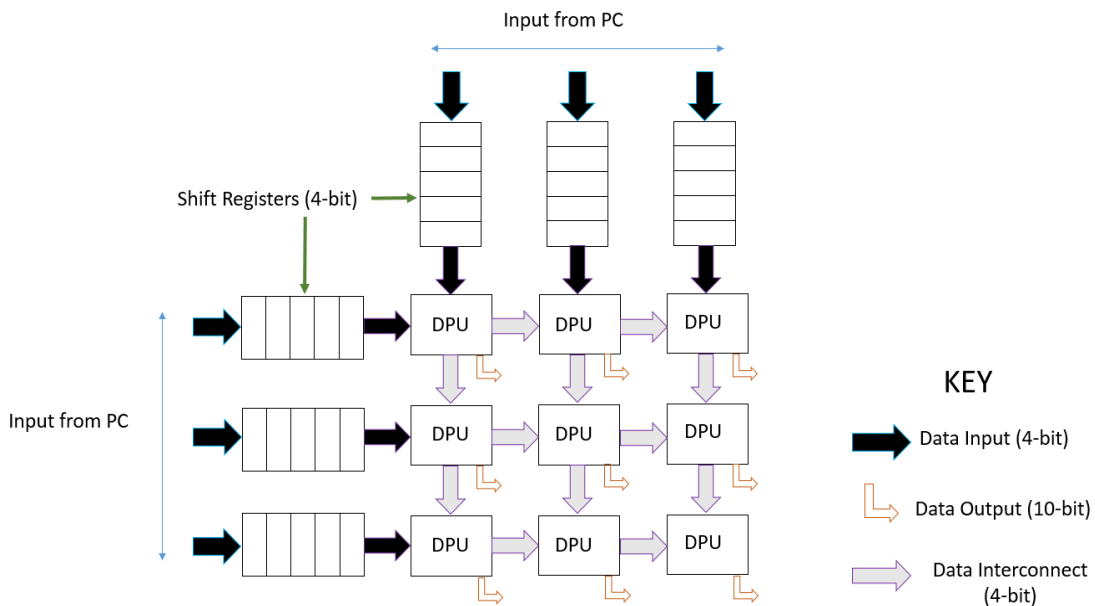
The input matrices are stored inside an Host Computer, these are inputted by the User. The Input Data receiver module along with the Data Bus Controller decides how the stream of input data goes into the processor. After processing the outputs from each individual DPU is taken and transmitted back to the Host Computer using the Output Data bus controller and the transmitter module.



**Figure 4.1 :- Functional Block Diagram**

## 4.2 DPU Block Diagram

The Input matrix are preloaded into a 5\*4-bit register Cache Memory. After each clock cycle is completed the input is then shifted into the DPU for the next computation. Each DPU has two 4-bit inputs and two 4-bit outputs that are passed along to the neighboring DPU's.



**Figure 4.2 :- DPU Block Diagram**

## 4.3 Working

We have taken a 3\*3 Matrix Multiplication to show the working of the Systolic Processor.

Each of the Inputs are of 4-bits, 15 being the maximum value of an input.

Let the sample matrix be

$$\text{Input 1} = \begin{pmatrix} A1 & A2 & A3 \\ C1 & C2 & C3 \\ E1 & E2 & E3 \end{pmatrix}$$

$$\text{Input 2} = \begin{pmatrix} B1 & D1 & F1 \\ B2 & D2 & F2 \\ B3 & D3 & F3 \end{pmatrix}$$

$$\text{Output Matrix} = \begin{pmatrix} \text{DPU1} & \text{DPU2} & \text{DPU3} \\ \text{DPU4} & \text{DPU5} & \text{DPU6} \\ \text{DPU7} & \text{DPU8} & \text{DPU9} \end{pmatrix}$$

The systolic processor consists of 9 DPU's arranged in a 3\*3 fashion, where each DPU computes each of the product as shown above.

$$\text{DPU1} = A1 * B1 + A2 * B2 + A3 * B3$$

$$\text{DPU2} = A1 * D1 + A2 * D2 + A3 * D3$$

$$\text{DPU3} = A1 * F1 + A2 * F2 + A3 * F3$$

$$\text{DPU4} = C1 * B1 + C2 * B2 + C3 * B3$$

$$\text{DPU5} = C1 * D1 + C2 * D2 + C3 * D3$$

$$\text{DPU6} = C1 * F1 + C2 * F2 + C3 * F3$$

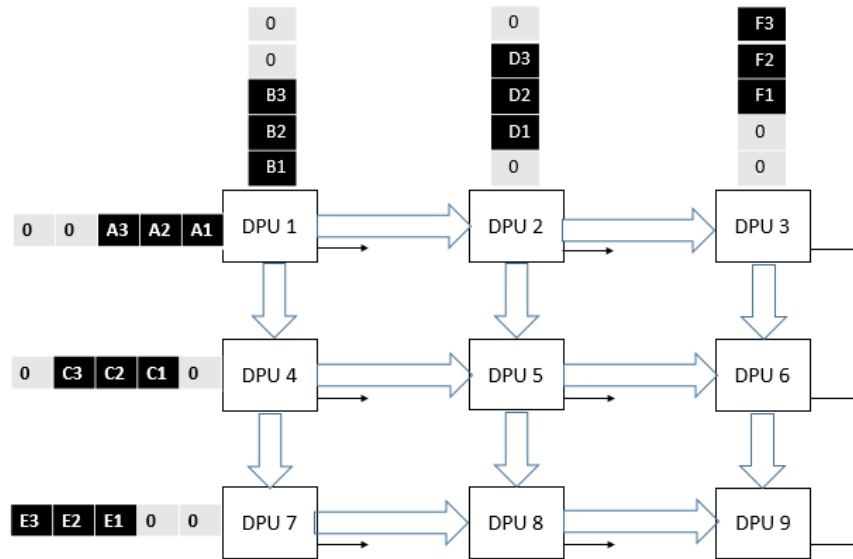
$$\text{DPU7} = E1 * B1 + E2 * B2 + E3 * B3$$

$$\text{DPU8} = E1 * D1 + E2 * D2 + E3 * D3$$

$$\text{DPU9} = E1 * F1 + E2 * F2 + E3 * F3$$

Each of the multiplication and addition takes one clock cycle each. Effectively each DPU can finish its computation within 3 cycles after it receives its input.

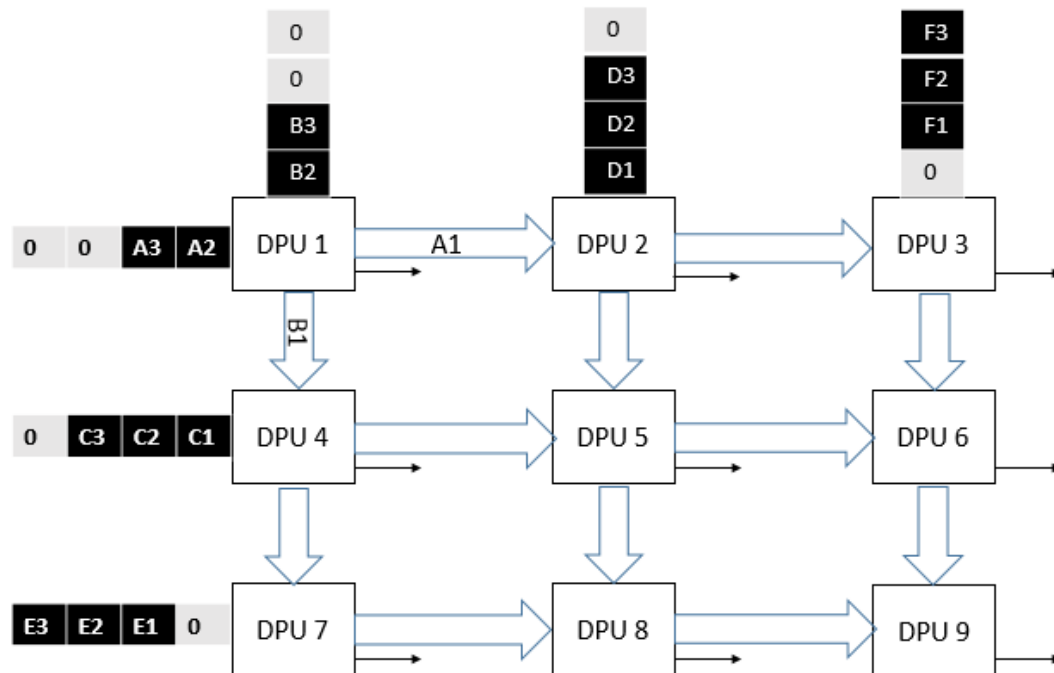
Each of the inputs are pre-fetched into the Register Cache memory as shown. The detailed working is shown.



**Figure 4.3 :- Data Pre-fetching**

### Clock 1

DPU1 receives the inputs and begins its 3-cycle computation. All other DPU's remain idle and after computation the inputs are passed to DPU2 and DPU4. All values in the register cache memory are shifted.

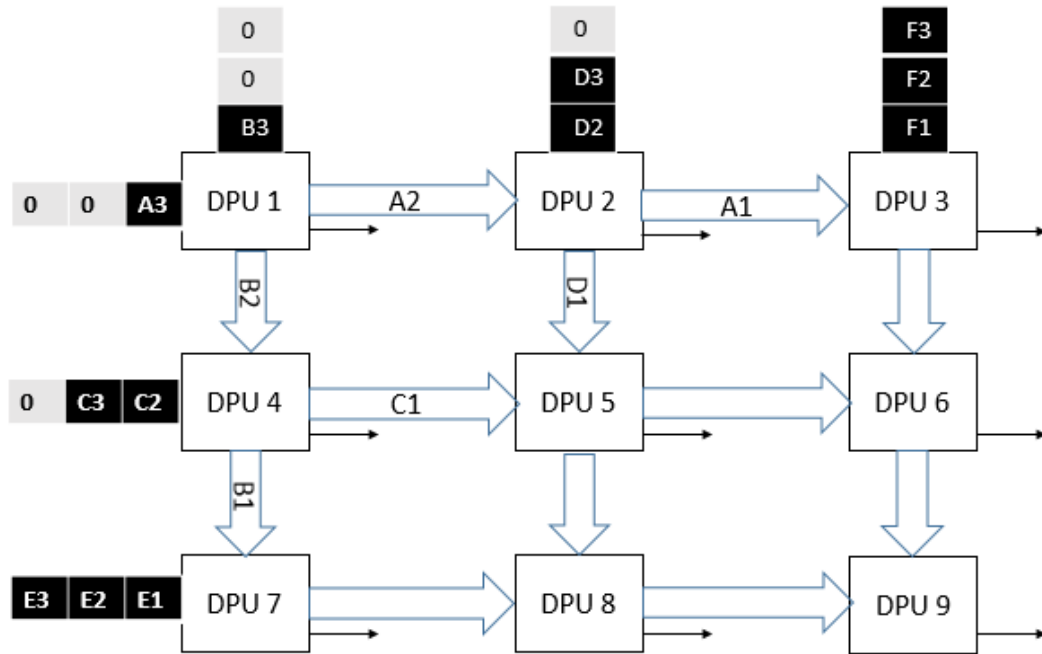


**Figure 4.4 :- Working cycle 1**



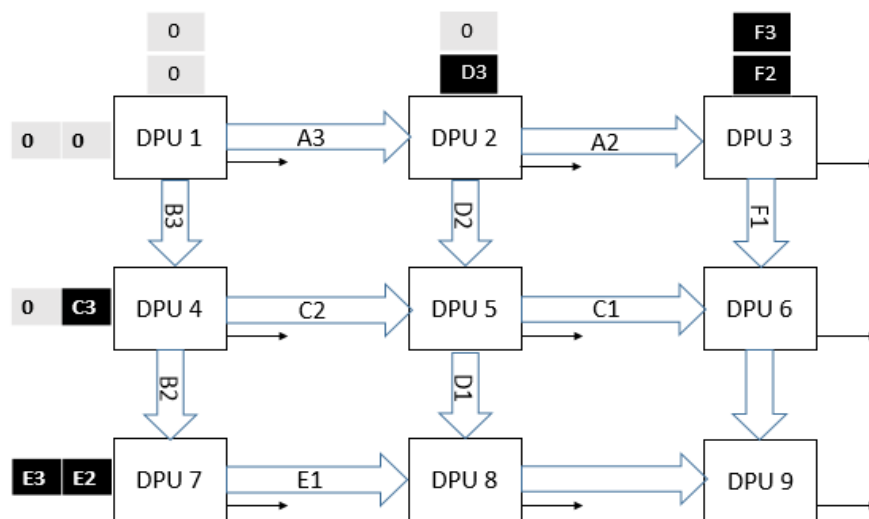
## Clock 2

The DPUs 2 and 4 receive the rippled inputs and begin its computation.

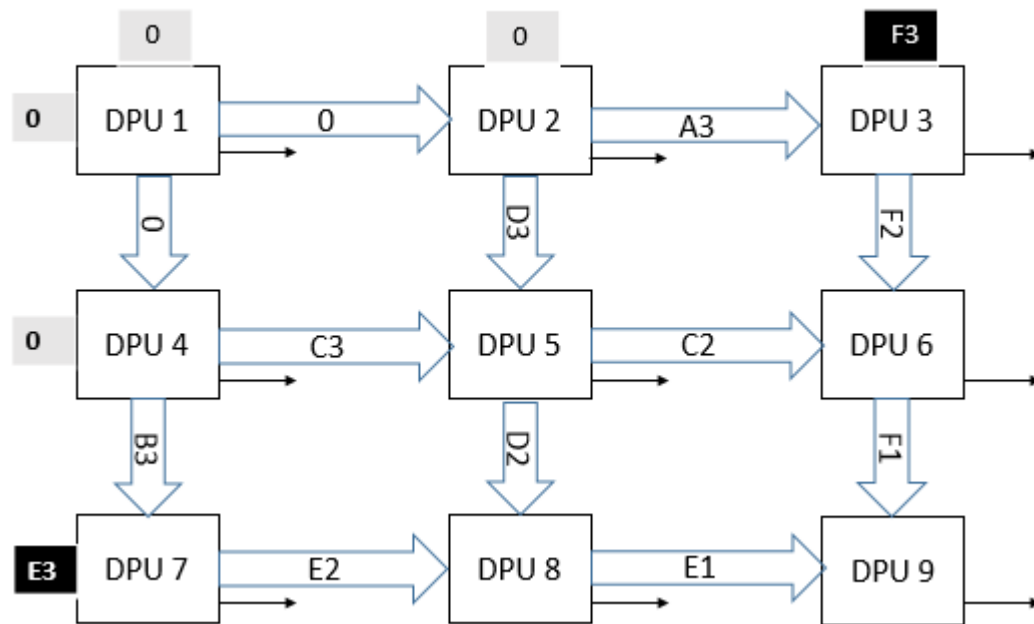
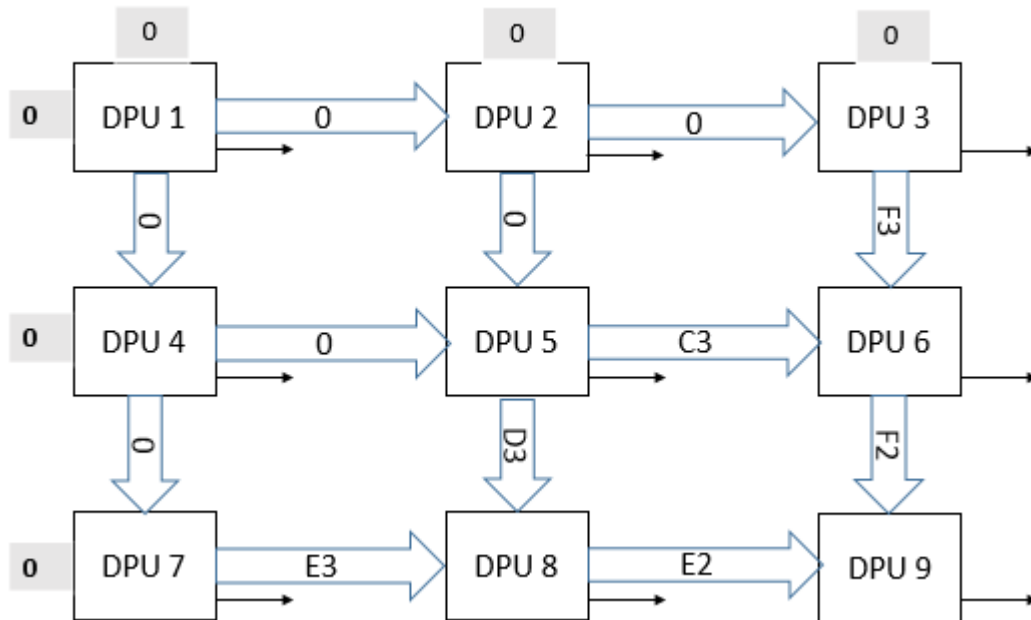


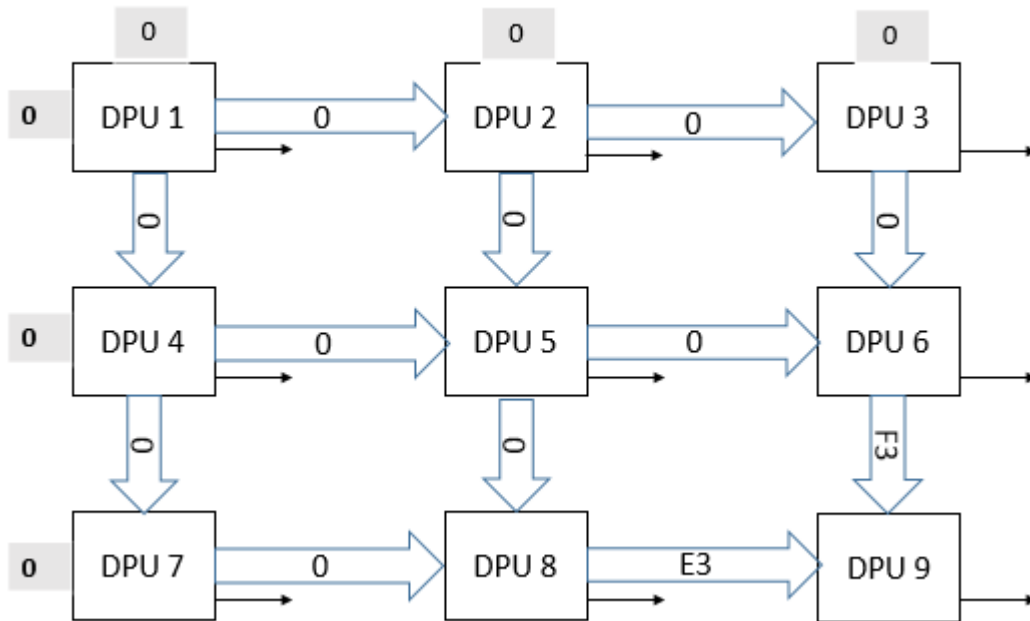
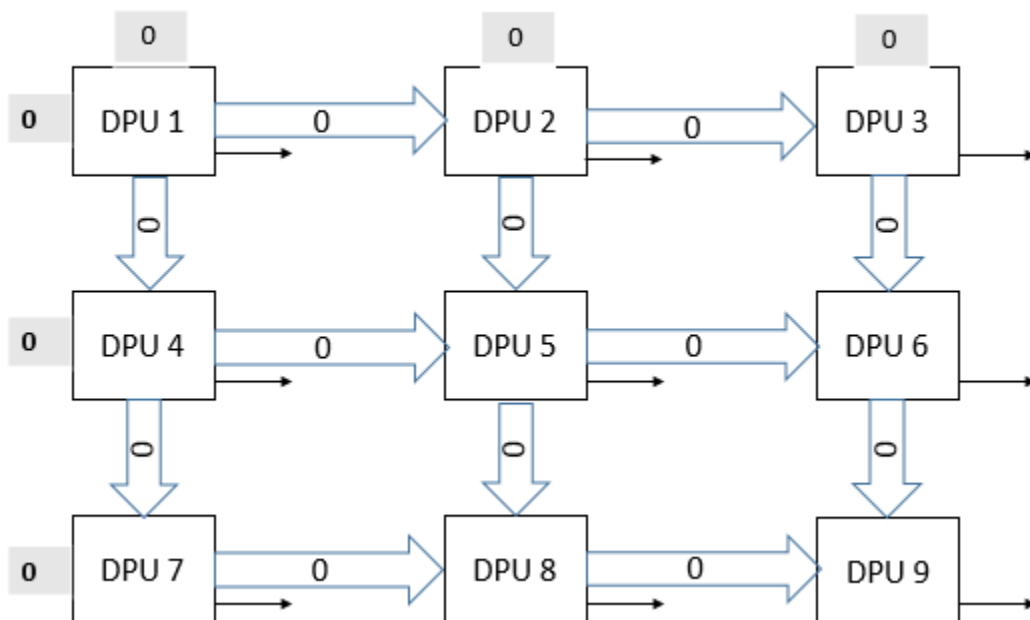
*Figure 4.5 :- Working cycle 2*

## Clock 3



*Figure 4.6 :- Working cycle 3*

**Clock 4***Figure 4.7 :- Working cycle 4***Clock 5***Figure 4.8 :- Working Cycle 5*

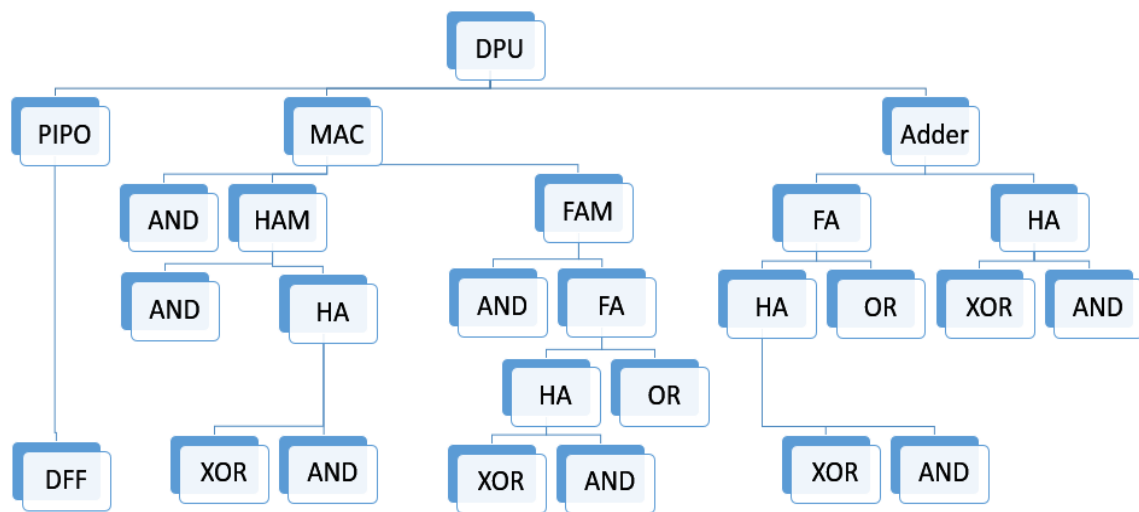
**Clock 6***Figure 4.9 :- Working Cycle 6***Clock 7***Figure 4.10 :- Working cycle 7*

The Systolic Architecture is divided into 3 design components namely Data Processing Unit (DPU), Interconnects and Register Cache.

## 4.4 Data Processing Unit Design

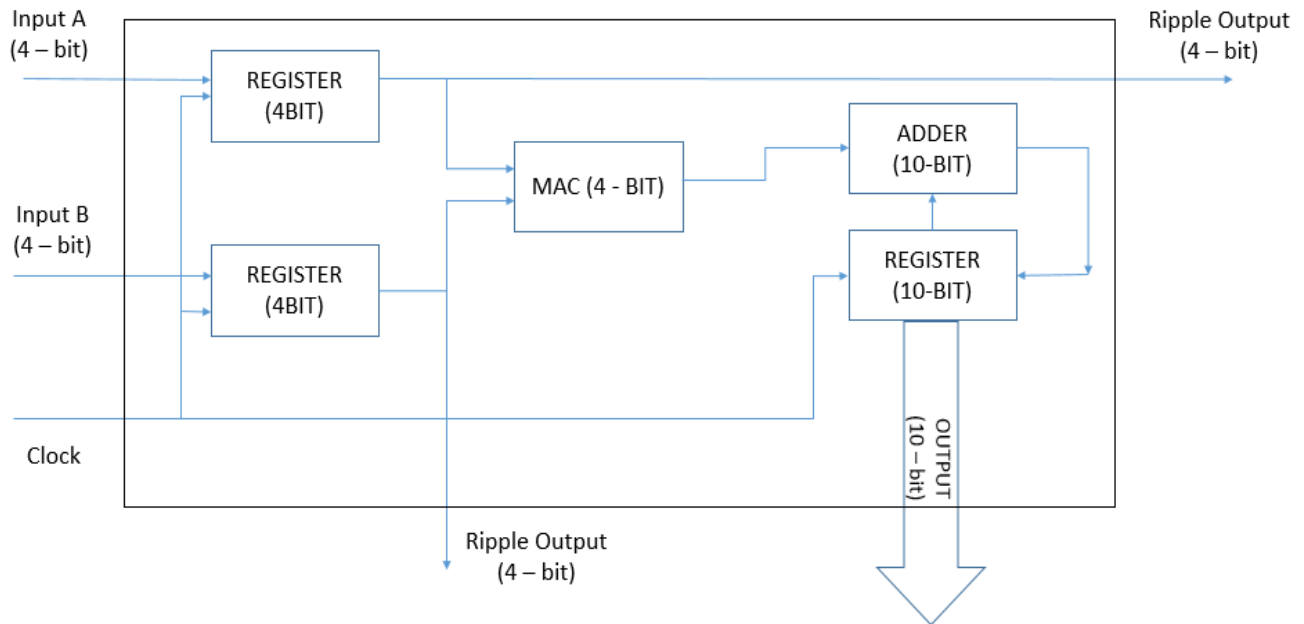
Each DPU computes a partial product of the matrix multiplication. It consists of a Parallel In-Parallel Out (PIPO) Shift Register, Carry Save Multiplier and a Full Adder.

The Tree structure shows how the DPU is made.



**Figure 4.11 :- DPU Tree structure**

All the DPU's are of the same design, which is the trademark of Systolic Processors. The input registers are of 4-bits, used for storing inputs. The multiplier produces an output of 8-bits. After multiplication a 10-bit Full Adder is used to add values and accumulates it into a 10-bit register. After 3 cycles the final 10-bit is taken from this register.

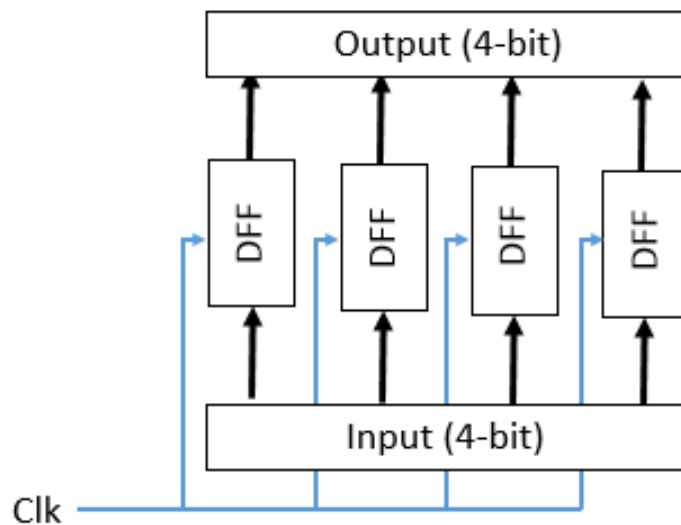


**Figure 4.12 :- DPU architecture**

#### 4.4.1 Parallel In-Parallel Out Shift Register

PIPO consists of a 4 positive edge triggered D-Flip Flops with asynchronous clear signal.

It is used for storing the input 4-bit from the host computer.



**Figure 4.13 :- PIPO 4-bit**

### 4.4.2 D- Flip Flop

It has a Data input along with a Clock input. The 2 outputs are named as Q and its complement. The D-FF used is a positive edge triggered Flipflop.

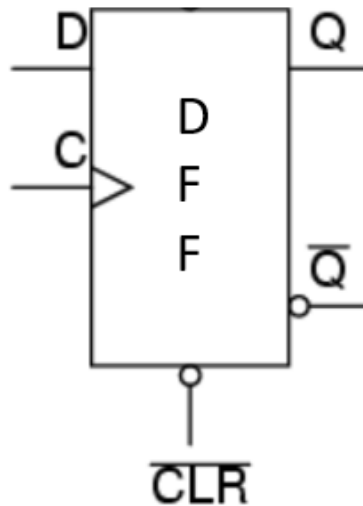


Figure 4.14 :- D- Flip Flop

### 4.4.3 Carry Save Multiplier

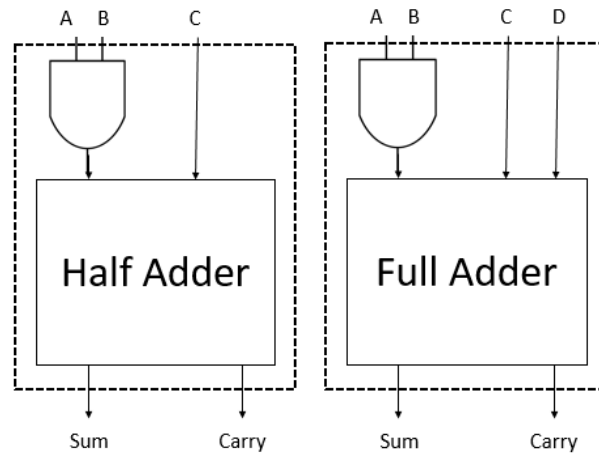
It consists of Half Adders, Full Adders and AND gates. Two 4-bit numbers A and B are multiplied together to get the result, the operation is shown below.

$$\begin{array}{r}
 \begin{array}{cccc}
 A_3 & A_2 & A_1 & A_0 \\
 B_3 & B_2 & B_1 & B_0
 \end{array} \times \\
 \hline
 \begin{array}{cccc}
 A_3 \cdot B_0 & A_2 \cdot B_0 & A_1 \cdot B_0 & A_0 \cdot B_0 \\
 A_3 \cdot B_1 & A_2 \cdot B_1 & A_1 \cdot B_1 & A_0 \cdot B_1 \\
 A_3 \cdot B_2 & A_2 \cdot B_2 & A_1 \cdot B_2 & A_0 \cdot B_2 \\
 A_3 \cdot B_3 & A_2 \cdot B_3 & A_1 \cdot B_3 & A_0 \cdot B_3
 \end{array} \\
 \hline
 \begin{array}{ccccccc}
 P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
 \end{array}
 \end{array}$$

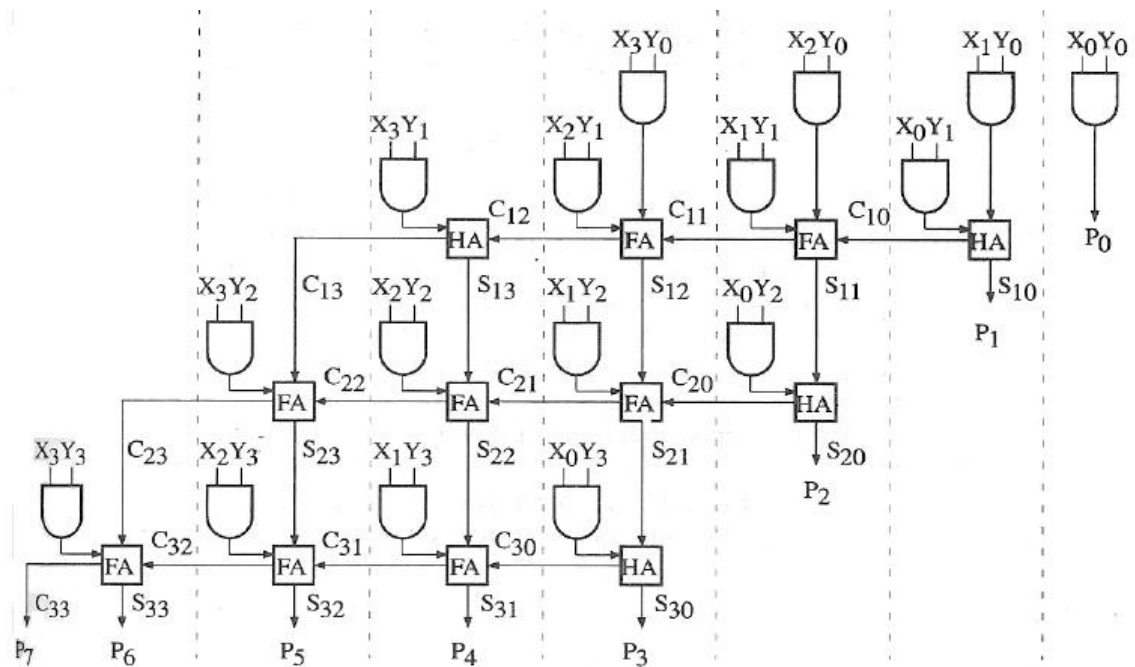
Figure 4.15 :- Multiplication operation

Two 4-bit numbers are multiplied and the maximum output that can be obtained is an 8-bit number, hence we have used an 8-bit multiplier.

For sake of easy coding we have combined the AND gate and Half Adders together into a Modified Half Adder.



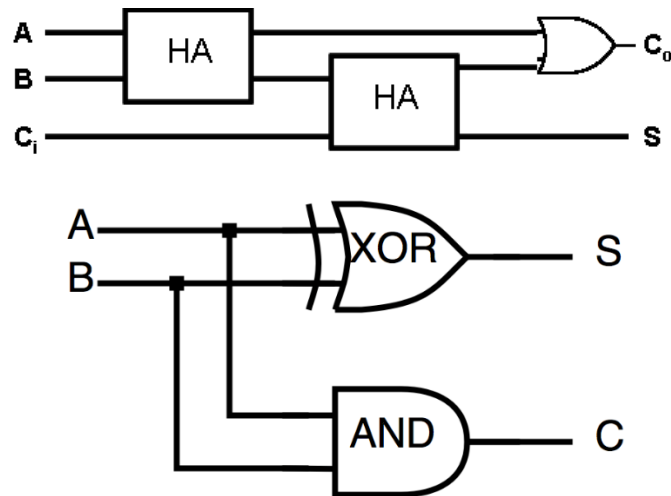
**Figure 4.16 :- Half and Full Adder**



**Figure 4.17 :- Carry Save Multiplier architecture**

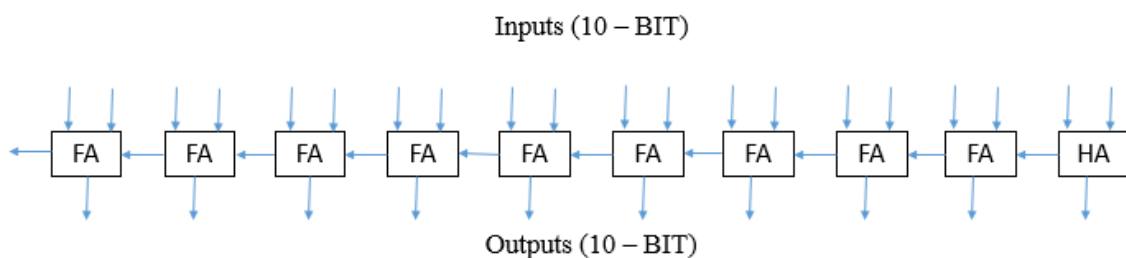
#### 4.4.4 Full Adder

The inputs to the full adder are from the Multiplier and from the 10-bit Shift Register. A,B and C are the inputs and the outputs are Carry and Sum for a Full Adder. For an Half Adder the Sum is the XOR of the inputs and Carry is the AND of the inputs



*Figure 4.18 :- Full Adder*

Final output of the Full Adder is of 10-bits and is stored in the 10-bit PIPO shift register.

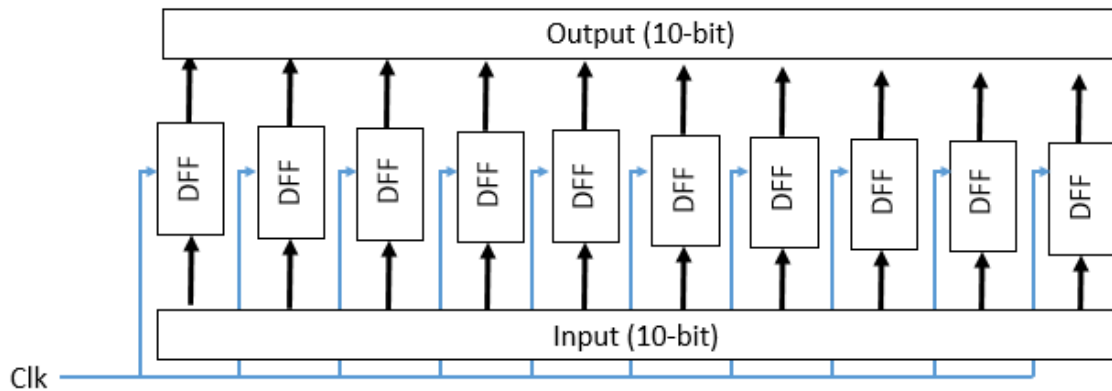


*Figure 4.19 :- Full Adder Architecture*



### 4.4.5 10-bit PIPO Shift Register

It consists of 10 positive edge triggered D-Flip Flops with asynchronous clear signal.



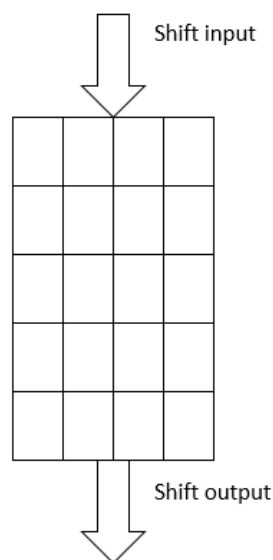
**Figure 4.20 :- PIPO 10-bit**

### 4.5 Interconnects

The interconnect consists of a 4 – bit wide bus that facilitates the rippling of input data in each rising edge of the clock. There are three outputs per DPU, two of which are interconnects for the data rippling between the DPUs.

### 4.6 Register Cache

It consists of five 4-bit registers that are used to store the input data to be given as input to the DPU's.



**Figure 4.21 :- Register Cache**

## **Chapter 5**

# **SOFTWARE DESCRIPTION**

This section aims to give a general idea regarding the various software used in simulating the Systolic Processor.

### **5.1 VHDL**

VHDL (VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming language.

VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a testbench.

VHDL has constructs to handle the parallelism inherent in hardware designs. VHDL is strongly typed and is not case sensitive. VHDL has file input and output capabilities, and can be used as a general-purpose language for text processing, but files are more commonly used by a simulation testbench for stimulus or verification data. One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE, Altera Quartus, Synopsys Synplify or Mentor Graphics HDL Designer) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate testbench. To generate an appropriate testbench for a particular circuit or VHDL code, the inputs have to be defined correctly.

The key advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires).

Another benefit is that VHDL allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time. A VHDL project is multipurpose. Being created once, a calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned (capacity parameters, memory size, element base, block composition and interconnection structure). A VHDL project is portable. Being created for one element base, a computing device project can be ported on another element base, for example VLSI with various technologies.

## **5.2 Xilinx ISE**

Xilinx ISE (Integrated Synthesis Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize (compile) their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors. The Xilinx ISE is primarily used for circuit synthesis and design, while ISIM or the ModelSim logic simulator is used for system-level testing. The primary user interface of the ISE is the Project Navigator, which includes the design hierarchy (Sources), a source code editor (Workplace), an output console (Transcript), and a processes tree (Processes).

The Design hierarchy consists of design files (modules), whose dependencies are interpreted by the ISE and displayed as a tree structure. System-level testing may be performed with ISIM or the ModelSim logic simulator, and such test programs must also be written in HDL languages. Test bench programs may include simulated input signal waveforms, or monitors which observe and verify the outputs of the device under test.<sup>7</sup>

ModelSim or ISIM may be used to perform the following types of simulations:-

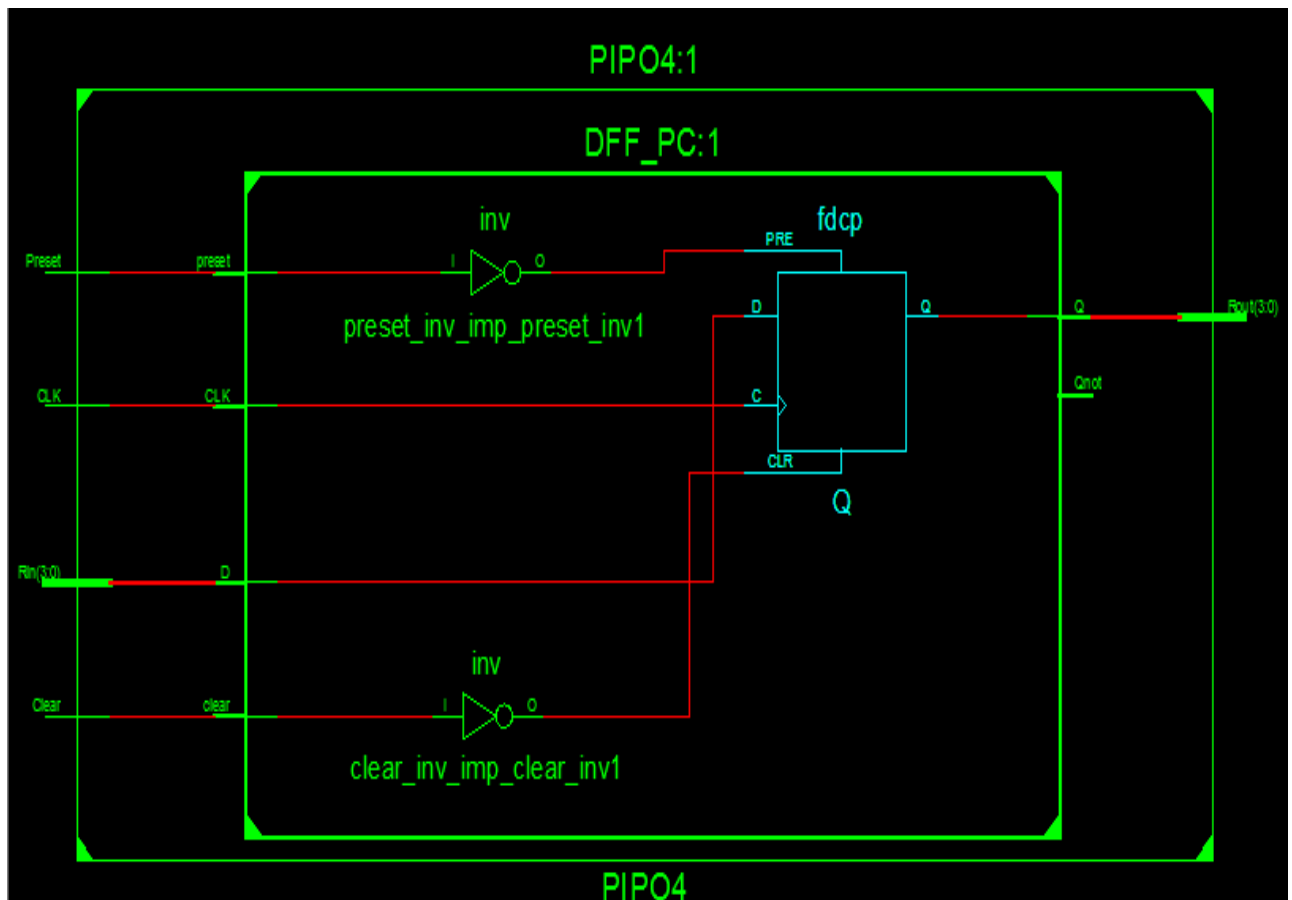
- Logical verification, to ensure the module produces expected results
- Behavioural verification, to verify logical and timing issues
- Post-place & route simulation, to verify behaviour after placement of the module within the reconfigurable logic of the FPGA

## Chapter 6

### IMPLEMENTATION ANALYSIS AND RESULTS

#### 6.1 PIPO 4-bit

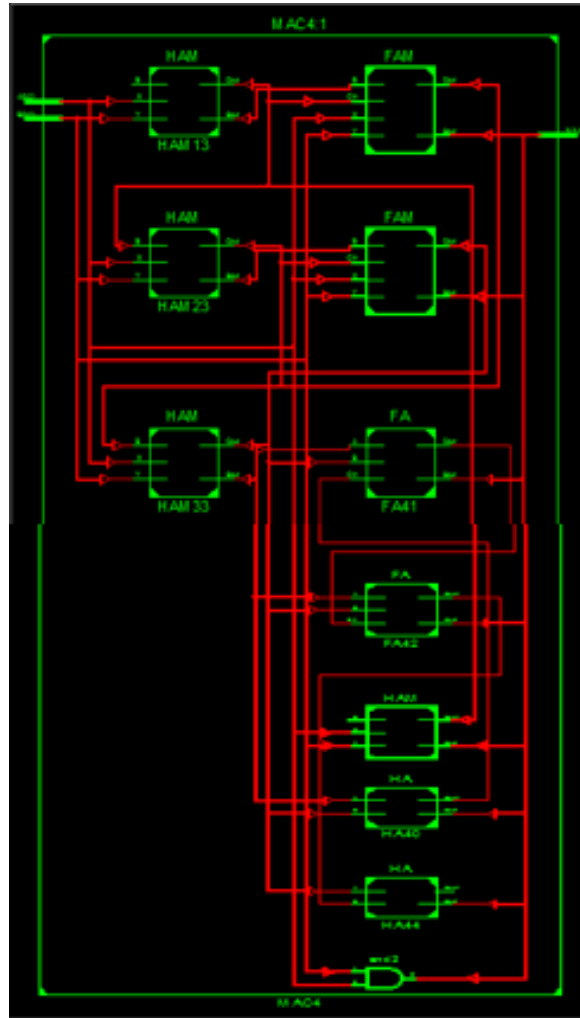
The 4 bit parallel in parallel out register is basically a D-flip flop with preset, clear, clock and the data bus. The below given diagram shows the RTL schematic for the same.



*Figure 6.1 :- RTL of PIPO 4-bit*

## 6.2 MAC

The multiplier accumulator is used for multiplying 4-bits. The below given diagram shows the RTL schematic for the same.



*Figure 6.2 :-RTL of MAC*

## 6.3 Full Adder

The full adder is of 10 bits. It takes the input from MAC and the 10 bit PIPO register. It consists of AND gate, Half Adder and XOR gates.

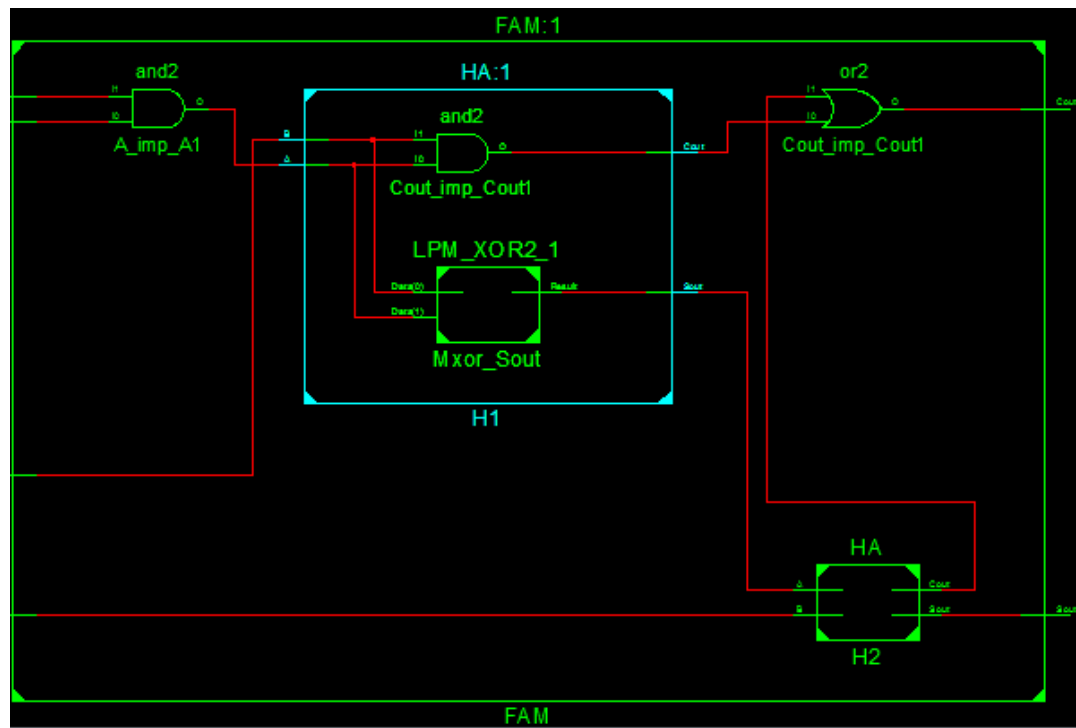


Figure 6.3 :- RTL of Full Adder

## 6.4 D-Flip Flop

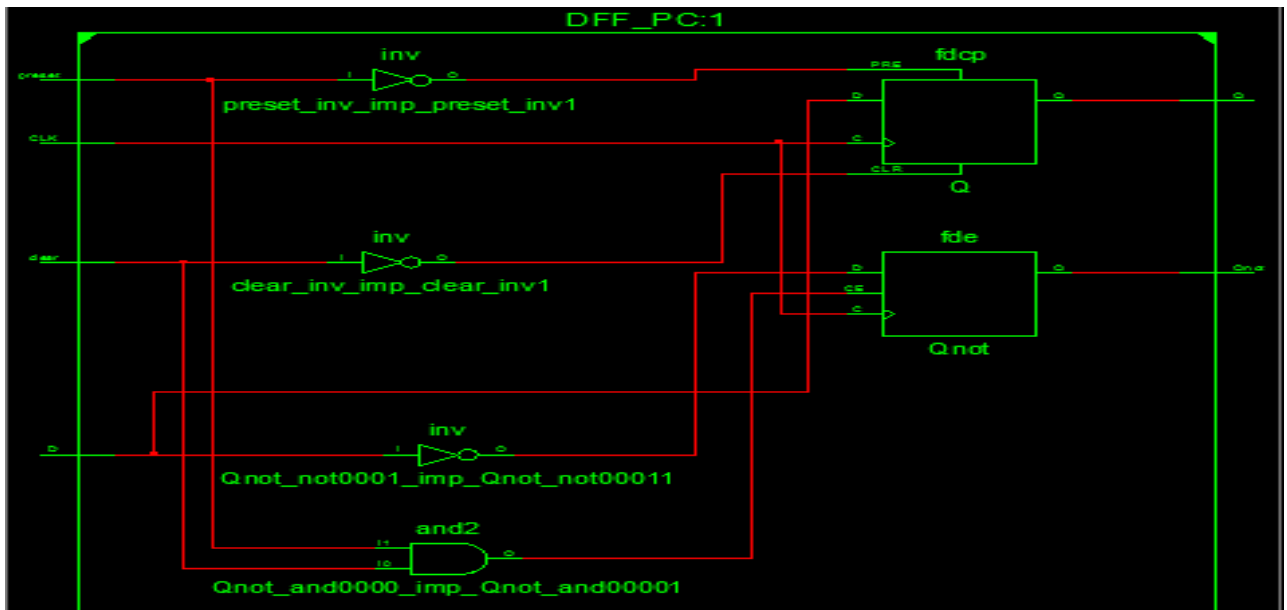
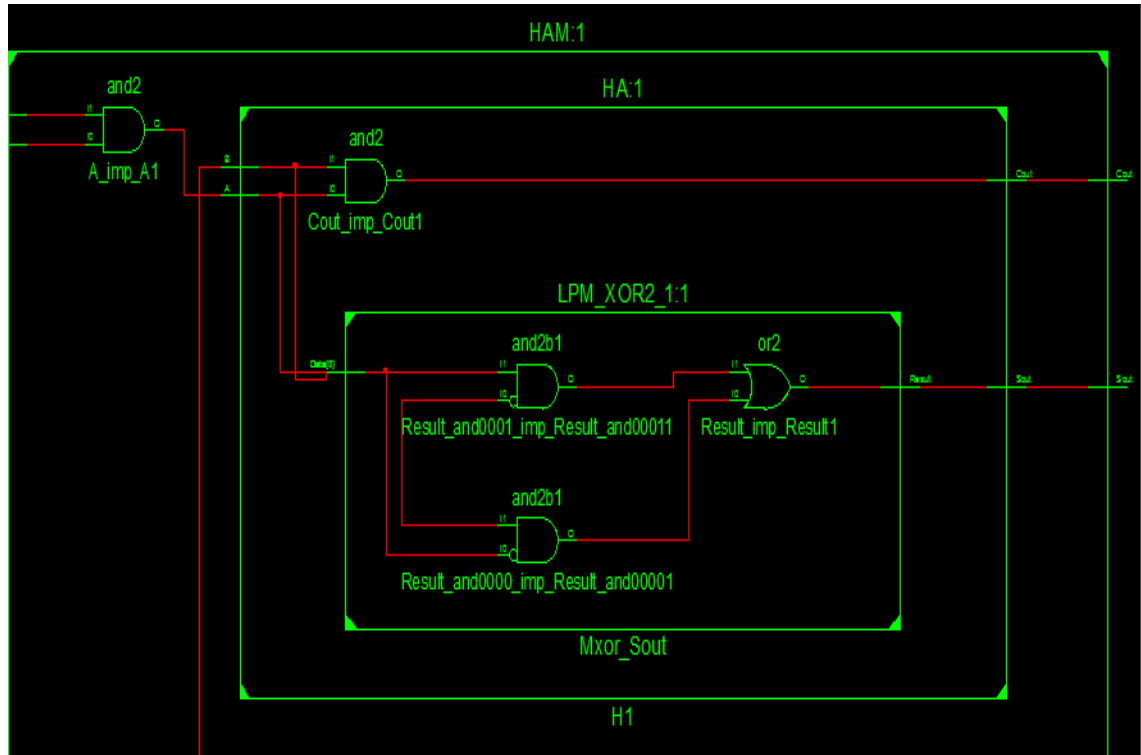


Figure 6.4 :- RTL of D- Flip Flop

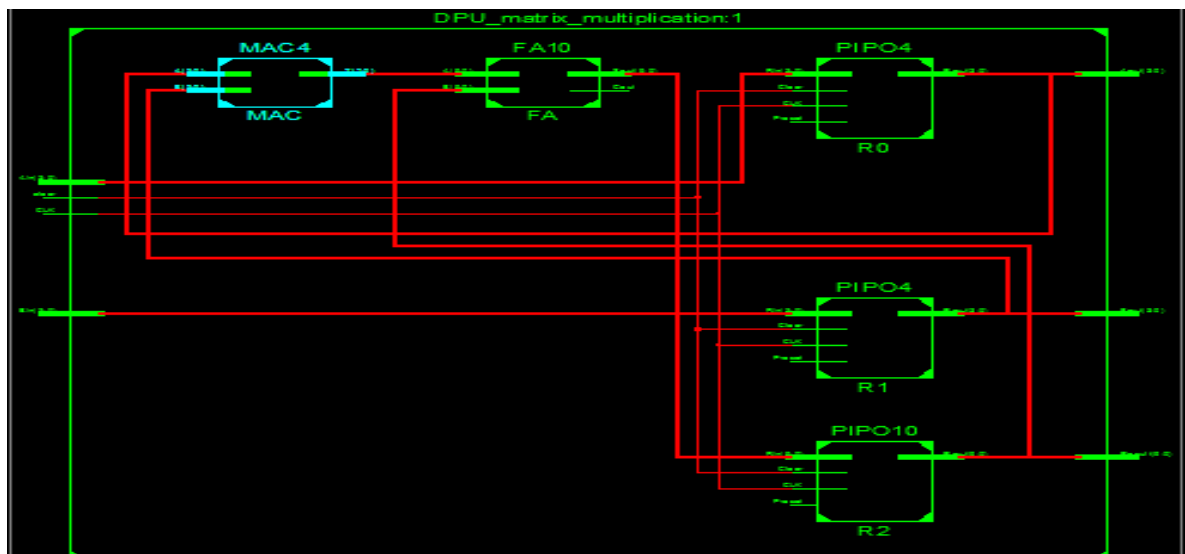
## 6.5 Half Adder Modified

For the sake of easy programming AND gate and Half Adder is combined together to form Half Adder Modified. The RTL schematic is shown below.



*Figure 6.5 :- RTL of Half Adder Modified*

## 6.6 DPU



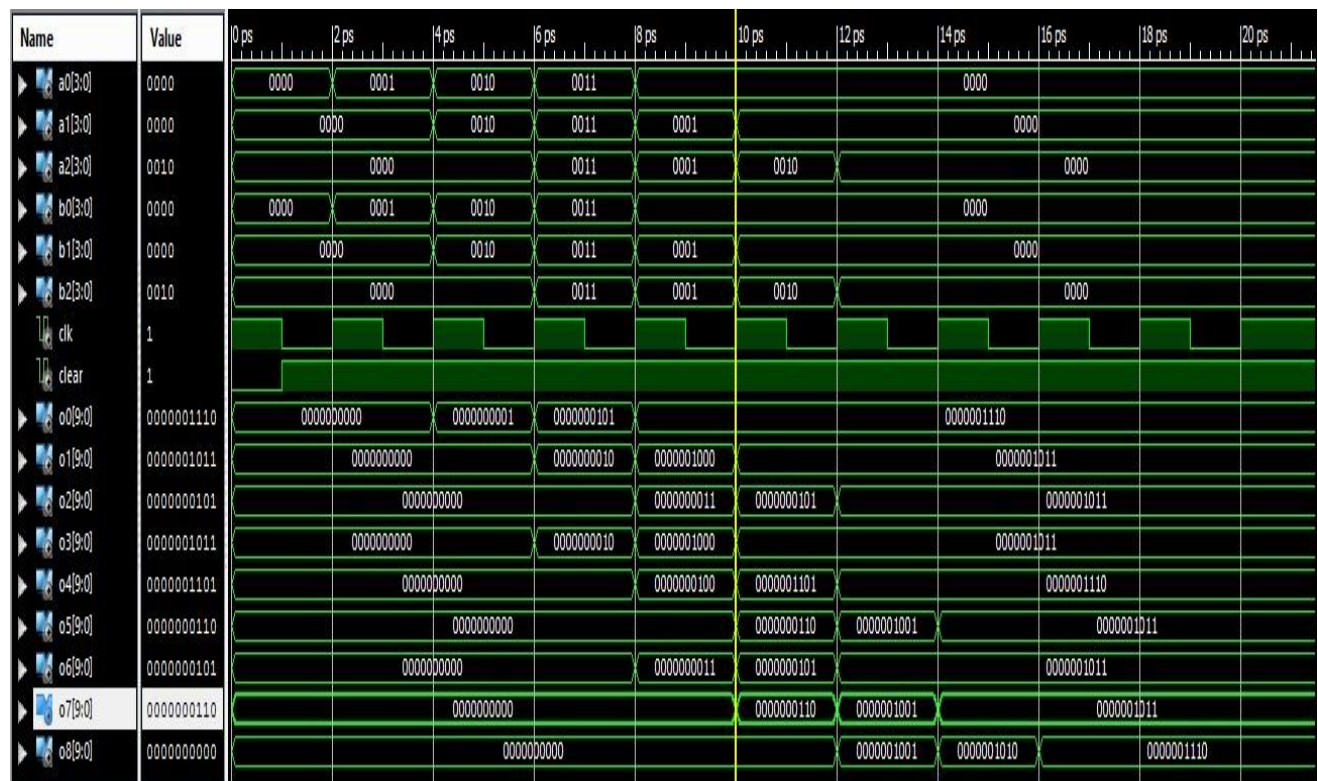
*Figure 6.6 :- RTL of DPU architecture*



Data Processing unit consists of the following two 4-bit registers that acts as a buffer for the input data, MAC unit that performs the multiplication operation, a 10-bit Full Adder and a 10-bit output feedback register in which the final output is stored.


## 6.7 Analysis

The final simulated output of a 3\*3 matrix multiplication is shown below. The inputs a0, a1, a2, b0, b1, b2 are of 4-bits. The outputs o0, o1, o2, o3, o4, o5, o6, o7, o8 are of 10-bits. The inputs are given as below and the output obtained of each DPU is plotted along with a global clock.



**Figure 6.7 :- Simulation Analysis**

The above Design Utilization Estimation summary shows the Device utilization of Xilinx Virtex-4 FPGA for implementing a Systolic Processor architecture for 3\*3 Matrix Multiplication.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	162	10,944	1%	
Number of 4 input LUTs	450	10,944	4%	
Number of occupied Slices	327	5,472	5%	
Number of Slices containing only related logic	327	327	100%	
Number of Slices containing unrelated logic	0	327	0%	
Total Number of 4 input LUTs	450	10,944	4%	
Number of bonded <a href="#">IOBs</a>	116	240	48%	
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Average Fanout of Non-Clock Nets	3.45			

**Figure 6.7 :- Device Utilization Analysis**

## **Chapter 8**

# **CONCLUSION**

As technology grows there is need for parallel growth in processing power. One of the efficient ways to increase the processing speed is to incorporate parallelism. Systolic processor array aims at improving the processing efficiency by incorporating highly extended parallelism and pipelining at reduced power expenditure especially for applications that has intensive use of multiplication and addition operations. Systolic architecture was studied thoroughly and its effectiveness was demonstrated by using matrix multiplication as an example. For a  $3 \times 3$  matrix multiplication using current processor architecture we require 45 clock cycles to obtain the result, whereas in systolic processors we require only 7 clock cycles to obtain the result.

A 4-bit Systolic Processor was successfully designed and implemented. The processor consists of 9 Data processing units, all interconnected to its neighboring DPU's. Each individual DPU which processes parts of the input (partial product) consists of two 4-bit PIPO registers, a 10-bit Full Adder, a 4-bit Multiplier and a 10-bit register for storing the final output.

## **Chapter 9**

### **FUTURE SCOPE**

Systolic processors is an untouched subject with immense possibilities. Generally the data flow of the DPU's is towards its neighboring DPU's. Higher efficiencies and processing speeds can be achieved if the DPU's can communicate Data with any of the other DPU's, making it a Wave front Array processor. These processors can be effectively used in Supercomputers and other similar devices.

Systolic processors are highly application specific, if at all they can be made into a general purpose processor they can effectively replace all the existing processors including i3, i5, i7, etc; which makes it a really big thing.

A system of re-configurable templates to run various applications can be developed and implemented. A system of Systolic array architecture with a flexible data path can be used to develop a universal platform of reconfigure hardware along with sufficient memory and registers and ports for input data and output data.