CHAPTER-5
IMPLEMENTATION OF AGENT BASED NOC ARCHITECTURE

5.1 INTRODUCTION

Before the invention of field programmable gate arrays (FPGA), the systems were fabricated by a fixed hardware for a fixed purpose of work. That hardware has limited flexibility by adding software, one of the examples for this fixed hardware was microprocessor. The upgrade in hardware logic is not guaranteed, if a system with this hardware needs to perform an extra work there is a need to change whole hardware setup, which is unreliable.

A FPGA is IC designed to be programmed or configured by end user or by designer after the manufacturing. Hence these ICs are called as “Field programmable”. The hardware description language (HDL) is used generally to configure FPGA. This HDL is very similar to application specific IC (ASIC), where circuit diagrams are main for analyzing the configuration, but these are very rare. To perform complex digital computations the FPGA with large number of logic gates and RAM are needed and these are called as contemporary FPGAs.

The FPGA design performs the operation with very fast I/O directional data buses. Hence, it is very complex to verify the timing of the valid data within the setup and hold times. To meet these time constraints, the FPGA is employed with floor planning. This floor planning forms resource allocation in the FPGA. All the logical operations an ASIC can perform can also be implemented by the FPGA. When compared to ASIC, the added future of the FPGA is that it can have an updated functionality after the manufacturing of a complete system, it is possible to have a re-configuration of certain portion of systems design with a low cost for the designer than ASIC. The FPGA has many advantages than ASIC in all the aspects like designing, programming and applications.
The FPGA is made up of programmable logic components which are called as logic blocks. It also consists of changeable interconnections to allow blocks wired together to made inter-wired connections between the logic gates to form different configurations. Logic blocks are configured to perform complex logical functions with the changeable interconnections basically from simple logic like AND and XOR. These FPGAs mostly contain memory elements, which are formed by simple flip-flops or blocks of memory.

Some of FPGAs are designed to have analog operations in addition to the digital operations. Programmable slew rate and drive strength on each output pin were the main features provided for analog operations by FPGA. This FPGA provides convenient to the engineers to set slow rates on lightly loaded pins. It also provides faster rates on heavily loaded pins on high speed channels, otherwise the system will run too slow.

In addition to the above features, the FPGA has an additional feature for analog feature like differential comparators on input pins. These comparators are used to connect differential signaling channels. Some of the mixed signal FPGAs have on-chip analog to digital and digital to analog to allow analog and digital operations allowing operate on-chip. Some of the FPGAs are having blur line between analog and digital ICs. In FPGA, digital ones and zeros are carried on internal programmable interconnect fabric, whereas in FPAA analog, the signals are carried out on its internal programmable on interconnect fabric.

The following advantages can be seen with the use of FPGAs:

1. Reprogrammable logic, it also meant reusability.
2. Lower non-recurring engineering cost.
3. It has a major application in prototyping.
4. Marketing of FPGAs need less time.
5. These are used for testing of electronic devices.
6. For the design of small size of products FPGAs are most suitable.
7. Researchers and industry persons can easily understand digital design concepts with the help of FPGAs.

5.2 Fundamental Concepts

Fundamentally, the FPGAs do not implement logic using gates. This is because the FPGAs are general purpose and using gates would create flexibility issues for implementing various types of logic. Therefore, FPGA uses the multiplexers and look-up tables for implementing logic as described in the next few paragraphs.

If the truth table is carefully analyzed, it can be noted that it is more than selection logic. Thus, it can be expressed by using a multiplexer as shown in the Figure 5.1. Figure 5.1 shows the implementation of the XOR gate. The output combinations are hardwired to the inputs of the 4:1 multiplexer and the select lines act as the input variables. Depending on the input combination, the output will be selected from one of many available inputs as per the truth table.

It can be concluded that just by changing the input values of the multiplexer, any of the several functions can be implemented which are possible with two variables such as AND and OR.

![Figure 5.1: Logic implemented by using a multiplexer](image)
In general, to implement any truth table, $2^n$ to 1 multiplexer should be chosen. However, it is not practical to have a multiplexer with variable inputs on silicon. Therefore, higher multiplexers are constructed by using the basic 2 to 1 multiplexer. In FPGAs, 2:1 multiplexer is one of the basic building blocks.

One more issue with multiplexers is that the inputs are considered hardwired to certain values. But in order to make it programmable, some sort of memory element can be used. In FPGAs, $2^n$X1 memories are used to keep the values. Such memories are called LUTs and these are look-up tables. The output values of the truth table are written into the LUTs during programming and the selection process is done through the multiplexers.

In a given FPGA family, the LUT sizes are fixed to maintain uniformity. When the EDA tools synthesize the design and create a configuration file, one of the information it possesses is the values that must go into the LUTs and how the interconnections should happen are shown in Figure 4.15 and 4.16.

![Selection logic](image)

Figure 5.2: Truth Table of Full Adder
Higher variable logic is achieved by replicating the LUT-MUX structure as shown in the Figure 5.4. As it can be noted, the structure remains the same and only the values are stored into the LUTs to change the functionality. This is the basis for programmability of hardware in FPGAs.

Hither to discussed is only the combinational logic that is the main functional part of any digital circuit. To implement sequential circuits, the FPGAs can also support the sequential elements called the master sequential elements.
Thus, LUTs, multiplexers and sequential elements form the most basic building blocks of FPGAs. They are arranged with other important elements which support logic building elements in the form of slices (Xilinx terminology). The slices are organized as configurable logic blocks. The switching matrix, clock management circuitry, IO blocks along with some major and minor on-chip IP cores create the components of the FPGA. This can be seen in the Figure 5.5 that shows the internals of a Xilinx CLB.

Figure 5.5: Xilinx CLB illustrated
5.3 Components in FPGA

The Xilinx FPGA architecture is basically formed with five fundamental programmable functional elements, viz. CLBs, I/O Blocks, Block RAM, Multiplier and DCM.

Configurable Logic Blocks (CLBs)

It will replace the operation of flip-flops or latches. These are designed to contain flexible look-up tables for the logic and storage operations like flip-flops. These CLBs perform variety of logical operations and are also used for the storage of data.

Input/output Blocks (IOBs)

These blocks are used to control the data flow between Input/Output pins and logic part that is internally available in the device. These IOBs support 3-stage operation with bi-directional data flow. In additions to this, these blocks support several varieties of signal standards. It also includes high performance differential standards. This additional feature is provided by double data rate registers.

Block RAM

This block RAM provides data storage in the form of 18-Kbit in dual-port blocks.

Multiplier or DSP48A Blocks

These blocks calculate the product by accepting two 18-bit binary numbers. These DSR48A blocks are available in extended family version of Spartan-3A. These processors contain pre-adder of 18-bit and 48-bit accumulator.

Digital Clock Manager (DCM)

The blocks and clocking network and the clock manager provide digital solutions for distributing. In addition, it provides dividing, multiplying, distributing and phase shifting of clock signals. The Figure 5.6 shows the clocking network.
5.4 FPGA Design Flow

Xilinx ISE (Integrated Software Environment) is product equipment delivered by Xilinx for synthesis and analysis of HDL plans, empowering the engineer to incorporate ("aggregate") their outlines, perform timing analysis, analyze RTL graphs, reproduce a configuration's response to various stimuli, and arrange the target device with the developer. The Figure 5.7 demonstrates the FPGA plan stream.
Implementation of the NOC design will follow the steps as shown in the flow diagram in the Figure 5.7. This is in accordance with the FPGA design flow specified by Xilinx. All the analysis is done by using the Xilinx tools.

![Figure 5.7: Xilinx FPGA design flow](image)

### i. Design Creation

During outline creation, make an ISE task and after that, make or add source records to that project. The SE1 projects can contain numerous sorts of source documents and plan modules, including HDL, EDIF net list, schematic, licensed innovation (IP), embedded processor and Digital Signal Processing modules.
ii. Synthesis

In the synthesis process, the engine compiles the configuration and change HDL source into particular outline net list. The utilization of Xilinx synthesis technology is supported by ISE programming. This programming can also be used for the combination of outside devices and simple precision programming.

iii. Constraints Entry

Utilizing plan imperatives, it is more possible to determine timing, position and other configuration necessities. The ISE programming gives editors to encourage limitations passage for timing requirements and also I/O pin and format imperatives.

iv. Implementation

Once the synthesis is completed, to made changes over logical design into physical document, run the outline execution. That can be downloaded to choose the target devices. The following accomplished steps are needed to run the plan execution after blend/synthesis:

- Translate combines the incoming net lists and constraints into a Xilinx design file.
- Map fits the design into the available resources on the target device and places the design.
- For the timing constraints mention Place, Route – places.
- Generate Programming File that in turn generates the bit stream file that can be downloaded to the device.

By using the Project Navigator Design Goals and Strategies, change the process properties to control the implementation and optimization of the design

v. Implementation Analysis

After execution, investigate proposed outline for execution against requirements, device, resources utilization, timing execution, and power utilization. It is possible see
results in static report records and by taking a gander at real gadget execution in graphical format apparatuses, for example, the Plan Ahead programming and FPGA Editor. One can intuitively examine timing and power results utilizing the Timing Analyzer and XPower Analyzer instruments. Furthermore, one can perform in-framework investigating utilizing the Chip Scope Pro-instrument.

vi. Implementation Improvement

The formulation of changes to design constraints, design sources and process properties are done based on the analysis of design results. In addition to these operations, it returns synthesis implementation to achieve design enclosure.

vii. Device Configuration and Programming

The configuration of the device is after the creation of the programmable file. During configuration, generate configuration files and programming files are downloaded to a Xilinx device from a host computer.

The configuration data is written to the FPGA, after applying power, by using any of eight different modes mentioned below:

- Master Serial from a Xilinx Platform Flash PROM
- Serial Peripheral Interface (SPI) from an industry standard SPI serial flash
- Internal SPI flash memory (Spartan-3AN devices)
- Byte Peripheral Interface (BPI) Up from an industry standard x8 or x8/x16 parallel NOR flash
- Slave Serial, typically downloaded from a processor
- Slave Parallel, typically downloaded from a processor
- Boundary Scan (JTAG), typically downloaded from a processor or system tester
- Multi-Boot configuration
IO Standards supported in FPGAs

FPGAs support the following single-ended standards:

- 3.3V low-voltage TTL (LVTTL)
- Low-voltage CMOS (LVCMOS) at 3.3V, 2.5V, 1.8V,
- 1.5V, or 1.2V
- 3.3V PCI at 33 MHz or 66 MHz
- HSTL I, II, and III at 1.5V and 1.8V, commonly used in-memory applications
- SSTL I and II at 1.8V, 2.5V, and 3.3V, commonly used for memory applications

FPGAs in the Extended Spartan-3A family support the following differential standards:

- LVDS, mini-LVDS, RSDS, and PPDS I/O at 2.5V or 3.3V
- Bus LVDS I/O at 2.5V
- TMDS I/O at 3.3V
- Differential HSTL and SSTL I/O
- LVPECL inputs at 2.5V or 3.3V

5.5 Overview of Xilinx FPGA Families

Currently Xilinx offers the following line of FPGAs

Virtex Series

These series of FPGAs have the integrated features like FIFO, ECC logic, DSP, Ethernet MAC controllers, PCI-Express controllers and high speed transceivers. In addition to these features, the series of FPGA logic has vertex series which include embedded fixed function hardware for commonly used functions like multipliers, memories and microprocessor cores. The above capabilities are used in wired and wireless equipment, medical equipment and defense systems.

Kintex Series

It is the first Xilinx mid-range of FPGA family. These series have the Virtex-6 family performance at low cost and power consumption. The kintex families FPGAs have
high bit transfer 12.5 GB/S and 6.5 GB/S low cost optimized FPGA. This family of FPGA provides memory, serial connectivity and logic performance for the applications like 10G high volume optical wired communication. It also provides balanced signal processing performance, low power consumption and low cost support to long term evolution wireless networks.

**Artix**

This design of FPGA is based on unified vertex series architecture. Mainly artix-7 family has 50 per cent low power consumption and 35 per cent low cost when compared with Spartan-6 family. Out of them, the Artix-7 FPGA has got an address sensitive, high-volume markets which are served by ASSPs and ASICs with low cost. These Artix family FPGAs are designed to meet the applications of low power portable ultrasound equipment, digital camera lenses control and military communication. Later on, due to lack of high bandwidth transceivers in Spartan-7, the Artix-7’s position is preferred as it has transceiver optimization.

**Spartan**

The Spartan series are mainly designed for high volume applications with low power and low cost in 2009. These types of FPGAs are mostly used in displays, wireless routers and set-top boxes etc. These Spartan-6 family FPGAs are fabricated with 45-nanoeter, 9-metal layers dual-oxide process technology. These FPGA applications are extended to flat-panel display and video surveillance applications due to cost.

**Zynq**

Zynq-7000 family group is used for high-end embedded applications like automated driver assistance automation, next-generation wireless etc. The Zynq-7000 is integrated with ARM Cortex-A9 MP Core-processor design based 28 nm systems. This architecture is different from the other families of programmable and embedded processors. Due to this, in the Zynq family, the technology is extended from an FPGA-centric platform to a processor-centric model. For the engineers, zynq-7000 appears as a
standard because of ARM processor based on SOC. This FPGA has immediate booting while starting and can be operated on different operating systems without the knowledge of programmable logic. Later, this family is extended to DSP integration with a release of Zynq-7100. This enables the programmable systems to have added advantages in military and wireless broadcast applications.

5.6 Implementation of Agent Based NOC on FPGA

Following the simulation of the agent based network on chip, further in order to realize the design on the FPGA platform, the design direly needs to be synthesized and implemented. Synthesis is the process of converting the behavioral code into the gate level net list. This is an essential step towards implementing the design on FPGA. The following snapshot summarizes the settings for synthesizing in the project.

<table>
<thead>
<tr>
<th>Synthesis Options Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>---- Source Parameters</td>
</tr>
<tr>
<td>Input File Name            : &quot;processing_element.prj&quot;</td>
</tr>
<tr>
<td>Input Format               : mixed</td>
</tr>
<tr>
<td>Ignore Synthesis Constraint File: NO</td>
</tr>
<tr>
<td>---- Target Parameters</td>
</tr>
<tr>
<td>Output File Name           : &quot;processing_element&quot;</td>
</tr>
<tr>
<td>Output Format              : EDIF netlist</td>
</tr>
<tr>
<td>Target Device              : xc3s5000-5-fg900</td>
</tr>
<tr>
<td>---- Source Options</td>
</tr>
<tr>
<td>Top Module Name            : processing_element</td>
</tr>
<tr>
<td>Automatic FSM Extraction   : YES</td>
</tr>
<tr>
<td>FSM Encoding Algorithm     : Auto</td>
</tr>
<tr>
<td>Safe Implementation       : No</td>
</tr>
<tr>
<td>FSM Style                  : LUT</td>
</tr>
<tr>
<td>RAM Extraction             : Yes</td>
</tr>
<tr>
<td>RAM Style                  : Auto</td>
</tr>
<tr>
<td>ROM Extraction             : Yes</td>
</tr>
<tr>
<td>Mux Style                  : Auto</td>
</tr>
</tbody>
</table>
Decoder Extraction : YES
Priority Encoder Extraction : Yes
Shift Register Extraction : YES
Logical Shifter Extraction : YES
XOR Collapsing : YES
ROM Style : Auto
Mux Extraction : Yes
Resource Sharing : YES
Asynchronous To Synchronous : NO
Multiplier Style : Auto
Automatic Register Balancing : No

---- Target Options
Add IO Buffers : YES
Global Maximum Fan-out : 500
Add Generic Clock Buffer (BUFG) : 8
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Use Clock Enable : Yes

Use Synchronous Reset : Yes
Pack IO Registers into IOBs : Auto
Equivalent register Removal : YES

---- General Options
Optimization Goal : Speed
Optimization Effort : 1
Keep Hierarchy : No
Net list Hierarchy : As Optimized
RTL Output : Yes
Global Optimization : All Clock Nets
Read Cores : YES
Write Timing Constraints : NO
Cross Clock Analysis : NO
Hierarchy Separator : /
Bus Delimiter : <>
Case Specifier: Maintain
Slice Utilization Ratio: 100
BRAM Utilization Ratio: 100
Verilog 2001: YES
Auto BRAM Packing: NO
Slice Utilization Ratio Delta: 5

These settings are common across all the readings which are taken between various families of the Xilinx FPGA used for the comparative studies. Further, certain implementation settings are kept default. Once the place and route are completed successfully, the bit file generation should be done. This configuration file is transported into the FPGA device through JTAG. The experimental setup is discussed in 5.7.

5.7 Experimental Setup

The experimental setup includes the following equipment:

1. Computer
2. FPGA Board
3. USB – Serial cable

It also needs a serial terminal to send and receive data. The Figure 5.8 shows the schematic setup. The Figure 5.9 shows the actual setup with FPGA board.

![FPGA setup for real time packet sending](image-url)
The packets should be framed in advance before sending. It should also be analyzed once they arrive as well. It is because of this reason, a simple script can be written to actually produce and analyze packets using MATLAB or Python that give an access to the serial ports of the computer.

Figure 5.9: Actual setup with FPGA
5.8 PERFORMANCE ANALYSIS

First, the performance metrics of the PE and 3*3 Mesh Network on various FPGA families is presented separately. The PE that is designed can be used as a standalone IP as well. Because of this reason, a study was made on the area and speed performance of the PE by using 4 input LUT devices and 6 input LUT devices. This is shown in Tables 5.1 and 5.2.

Table 5.1: PE implementation on a few FPGA families

<table>
<thead>
<tr>
<th>FPGA Family</th>
<th>Area (slices)</th>
<th>FFS LUTs</th>
<th>Speed (MHz)</th>
<th>Memory (%)</th>
<th>IOs</th>
<th>Area %</th>
<th>Delay (ns)</th>
<th>Power(mW) (5000 packets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spartan 3</td>
<td>365</td>
<td>356</td>
<td>625</td>
<td>122</td>
<td>9</td>
<td>679</td>
<td>1.10</td>
<td>6.85</td>
</tr>
<tr>
<td>Spartan 3A</td>
<td>360</td>
<td>362</td>
<td>616</td>
<td>134</td>
<td>10</td>
<td>679</td>
<td>3.16</td>
<td>7.43</td>
</tr>
<tr>
<td>Spartan 6</td>
<td>306</td>
<td>298</td>
<td>298</td>
<td>191</td>
<td>3</td>
<td>679</td>
<td>0.24</td>
<td>5.234</td>
</tr>
<tr>
<td>Virtex 4</td>
<td>368</td>
<td>361</td>
<td>627</td>
<td>226</td>
<td>2</td>
<td>679</td>
<td>0.40</td>
<td>4.421</td>
</tr>
<tr>
<td>Virtex 5</td>
<td>360</td>
<td>437</td>
<td>437</td>
<td>276</td>
<td>1</td>
<td>679</td>
<td>1.00</td>
<td>3.613</td>
</tr>
<tr>
<td>Virtex 6</td>
<td>305</td>
<td>343</td>
<td>343</td>
<td>350</td>
<td>0.50</td>
<td>679</td>
<td>0.04</td>
<td>2.662</td>
</tr>
<tr>
<td>Virtex 7</td>
<td>305</td>
<td>343</td>
<td>343</td>
<td>375</td>
<td>0.40</td>
<td>679</td>
<td>0.02</td>
<td>1.523</td>
</tr>
</tbody>
</table>

Table 5.2: 3x3 PE Network on various FPGA families

<table>
<thead>
<tr>
<th>FPGA Family</th>
<th>Area (slices)</th>
<th>FFS LUTs</th>
<th>Speed (MHz)</th>
<th>Memory (%)</th>
<th>IOs</th>
<th>Area %</th>
<th>Delay (ns)</th>
<th>Throughput (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spartan 3</td>
<td>2049</td>
<td>2684</td>
<td>2846</td>
<td>105.662</td>
<td>81</td>
<td>159</td>
<td>6</td>
<td>9.464</td>
</tr>
<tr>
<td>Spartan 3A</td>
<td>2066</td>
<td>2676</td>
<td>2832</td>
<td>121.198</td>
<td>92</td>
<td>159</td>
<td>18</td>
<td>8.251</td>
</tr>
<tr>
<td>Spartan 6</td>
<td>2398</td>
<td>2781</td>
<td>2398</td>
<td>200.698</td>
<td>27</td>
<td>159</td>
<td>3</td>
<td>4.983</td>
</tr>
<tr>
<td>Virtex 4</td>
<td>2176</td>
<td>2677</td>
<td>3177</td>
<td>211.372</td>
<td>18</td>
<td>159</td>
<td>2</td>
<td>4.731</td>
</tr>
<tr>
<td>Virtex 5</td>
<td>2487</td>
<td>2710</td>
<td>2487</td>
<td>286.64</td>
<td>9</td>
<td>159</td>
<td>1</td>
<td>3.489</td>
</tr>
<tr>
<td>Virtex 6</td>
<td>2443</td>
<td>2746</td>
<td>2443</td>
<td>350.152</td>
<td>4.5</td>
<td>159</td>
<td>0.30</td>
<td>2.856</td>
</tr>
<tr>
<td>Virtex 7</td>
<td>2443</td>
<td>2746</td>
<td>2443</td>
<td>375.672</td>
<td>3.6</td>
<td>159</td>
<td>0.10</td>
<td>2.662</td>
</tr>
</tbody>
</table>
NOVELTY OF THE WORK

In this work, we propose a new architecture based on elements called agents which curtail the need of separate routers. Following this design approach, it is easy to develop and build network on chips and synthesize them for practical applications such as implementing on an FPGA. In this methodology, the core logic forms the first layer and the agent forms the peripheral layer surrounding the core logic. The communications between the core and the agent happens through the input and output agents. The buffers make sure that packets are not lost during the process.

The highlights of the projects are the following:

Organization of core and routing logic:

The core resides within the agent. The agent takes care of bringing the packets in and taking the packets out. Agents are aware of the status of the neighboring elements directly. They can manage unprocessed packet very efficiently.

Communication protocol:

Transfer to the adjacent processing element is done in the ration one transaction per packet which is the most efficient in the class of network on chip systems done so far as mentioned in state of the art.

Multi-dimensional routing:

This design features inter processing element routing along with global routing. This avoids the need for multiple hops for transfer of packets. This would avoid unnecessary transfer of packets to the adjacent elements which would claim some of their processing time and power. Thus making it more efficient in terms of processing delay and power.

Packet structure:

The packet structure is modified to contain a lot of processing related information making it easy for the host systems to keep track of the packet status.
Efficient packet distribution in the network:

Packet injection takes place uniformly such that all the processing elements are uniformly loaded.

Smart packet control:

The agents are well aware of the processing elements processing status, thus it keeps track of the packets in a smart way. Three scenarios can be explained in this context:

1. When the packet is taken in and if it’s not processed since the core was busier than expected, it sends the packet to another processing element
2. If the no processing elements accept the packet, it keeps it in the buffer until it can be routed.
3. In case, within the allotted time if the packet is not accepted anywhere, it is routed to the host output controller, where the host reads the packet and takes further decisions.

5.9 CONCLUSION

In this chapter Agent based NOC architecture is implemented on various FPGA families and performance is evaluated. It is proved that the proposed Agent based NOC architecture getting efficient values on various FPGAs, shown in Table 5.1 and Table 5.2. It should also be noticed that most proposals are simulation based and thus there is a great need to analyze many of the problems or needs should be researched through implementation strategies: We have proved that proposed Architecture can be implemented on various FPGA families.