2023

Rapid Prototyping and Functional Verification of Power Efficient AI Processor on FPGA

Vivek Liladhar Ladhe
University of Windsor

RAPID PROTOTYPING AND FUNCTIONAL VERIFICATION OF POWER EFFICIENT AI PROCESSOR ON FPGA

by

Vivek Liladhar Ladhe

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfilment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

Rapid Prototyping and Functional Verification of Power Efficient AI Processor on FPGAs

by
Vivek Liladhar Ladhe

APPROVED BY:

I. Ahmad
School of Computer Science

E. Abdel-Raheem
Department of Electrical and Computer Engineering

M. Khalid, Advisor
Department of Electrical and Computer Engineering

January 23, 2023

Declaration of originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Prototyping a design on a Field Programmable Gate Array (FPGA) involves different stages such as developing a design, performing synthesis, handling placement and routing and finally generating the programming bit file for the FPGA. After successful completion of the above stages, it is important to functionally verify the design. This thesis addresses the challenges involved in rapid prototyping and functional verification of a low power AI processor provided by the industry partner. This research also addresses the methodology used in generating programming bit file and testing the design. Traditional method of testing a design using RTL level testbench utilises more time and relies on functioning of other components associated with the design. This thesis incorporated a new technique of testing the design using software programs focusing on verification of the functionality of a particular module without depending on the other. This methodology reduced the time for functionality verification for part of the design from approximately 1 month to about 2 weeks. Finally, using the methodology mentioned above, the design was synthesized for two FPGA kits, along with analysing the power consumption of the design. The results show the low power nature of the design as it does not use any external memory resulting in faster Arithmetic Logic Unit (ALU) operations thereby saving time to access the data.

Dedication

I would like to dedicate this thesis to my mom for her incredible love and support. Because I believe that she is the real backbone of our family, this is to appreciate her selfless hard work and efforts towards the family. Furthermore, I dedicate it to my father to raise me like a son and give me the wings to fly. To my grandfather, for always trusting me and supporting me in my hard times, without his encouragement, nothing would have been easy. And to my entire family for their unconditional affection towards me.

Acknowledgements

Firstly, I would like to express my deepest gratitude to my supervisor Dr. Mohammed Khalid for giving me the opportunity to conduct my Master's thesis research under his supervision. I am very grateful for his patience, his kind support and assistance during the project. His encouragement helped me to the surpass difficulties that I have encountered during my research and study. I am grateful and fortunate to have him as a mentor and supervisor.

I would like to thank Aarish Technologies and MITACS for providing me an opportunity to witness industrial experience. I am very thankful to Mr. Pavel Sinha and Aarish Technologies for providing the design of AI Processor and all the help required to complete this thesis work.

I would like to thank Dr. Esam Abdel-Raheem and Dr. Imran Ahmad for taking time from their busy schedule to be part of my thesis committee and for providing insightful suggestions to improve my research.

And finally, I am dedicating my research to my parents, for their kind support in all steps of my life and especially their continued support during this project.

Contents

List of Tables

List of Figures

List of Abbreviations

CAD Computer-Aided Design

RTL Register Transfer Language

CNN Convolutional Neural Network

AI Artificial Intelligence

DNN Deep Neural Network

GPU Graphics Processing Unit

FC Fully Connected

CONV Convolution

LRN Local Response Normalization

MAC Multiply-Accumulate

PE Processing Engine/Element

EDA Electronic Design Automation

RF Register File

WS Weight Stationary

ADC Analog-to-Digital Converter

DAC Digital-to-Analog Converter

RAM Random Access Memory

IC Integrate Circuit

MLA Machine Learning Accelerator

MIPI Mobile Industry Processor Interface

PFA Performance Functional Array

PFU Performance Functional Unit

CC Core Compute

ISA Instruction Set Architecture

WNS Worst Negative Slack

TNS Total Negative Slack

DEF Data Exchange Format

FPGA Field Programmable Gate Array

SoC System on Chip

ASIC Application Specific Integrated Circuit

PFU Programmable Functional Unit

UART Universal Asynchronous Receiver-Transmitter

I2C Inter-Integrated Circuit

SPI Serial Peripheral Interface

GPIO General Purpose Input/Output

SRAM Static Random Access Memory

DM Debug module

CSI Camera Serial Interface

FIFO First-In First-Out

HLS High Level Synthesis

TCL Transaction Control Languages

DSP Digital Signal Processing

CPU Central Processing Unit

CLB Configurable Logic Block

LE Logic Element

ALM Adaptive Logic Module

LUT Lookup Table

FA Full Adder

FF Flip-Flop

HDL Hardware Description Language

ALU Arithmetic Logic Unit

Chapter 1

Introduction

1.1 Motivation

In 1965, Golden Moore predicted that the number of transistors in a single integrated circuit will double after every eighteen months. Since last five decades this statement has been proved correct [4]. In recent years, scaling down of transistor size has become extremely difficult and in parallel to it the demand for high performance and power efficient microprocessors is increasing. The main reason being applications that are emerging in the fields of mobile computing, machine learning, data mining and computer graphics. Simply adding more computational devices and memory into a processor may not be an ideal solution of increasing throughput. Current research efforts are focusing on using specialized hardware accelerators for speeding up computationally intensive tasks.

Artificial Intelligence is one of the fastest growing areas today. It is already applied in fields such as data mining, computer vision, robotics, stock trading, image and video, speech and language, medical, etc. However, most of the machine learning algorithms are computationally intensive, running such algorithms on GPUs resulted in high performance, but at the same time power consumption of those GPUs and efforts to design cooling systems for handling heat dissipation become a challenge.

A new AI processor was designed by our industry partner Aarish Technologies for low power applications in edge computing. This thesis deals with rapid prototyping and functional verification of this power efficient AI processor on FPGA. Prototyping is not a push-button process. Detailed consideration and care are required at various phases of prototyping. The following points perfectly describe various highlights of using FPGAs for prototyping [3].

1. High performance and accuracy - Only FPGA based prototyping provides both speed and accuracy necessary to properly test many aspects of the design. For example, a team can aim to validate some of the embedded software codes and see how it runs at-speed on real hardware

2. Real-time dataflow - Part of the reason that verifying a design is hard because its state depends on many variables like previous state, sequence of inputs, possible feedback of the outputs. Running the design at real-time speed connected to the rest of the system allows to see immediate effect on all of them.

3. Software Validation - Using FPGAs we can achieve speeds up to real time and yet still be modelling at a level of full RTL cycle accuracy. This enables the same prototype to be used not only for accurate models required by low-level software validation, but also high-speed models required by high-level application developers. The complete software stack can be modeled on the single FPGA Prototype.

4. Interfacing benefit - Different modules can be interfaced with FPGA which helps verifying the correct processing at each level to observe that correct output data is generated.

5. Prototyping usage out of labs - One true unique aspect of FPGA based prototyping for validating a design is its ability to work standalone. This is because FPGAs can be configured from FLASH EEPROM cards or other self contained

medium without supervision from a host PC. The prototype therefore can run standalone and be used for testing design in different situations provided by other modelling techniques such as emulation which completely relies on host intervention.

1.2 Thesis Objectives

The goals of this thesis were: to prototype the Aarish AI processor on the FPGA, perform functional verification of the design and to evaluate the power consumption of the implemented design to verify power efficiency. The research goals were achieved in the following phases.

1. Manual way of designing RTL required modifications to the design on daily basis hence, to generate the build of the modified RTL and to perform functional verification of the design frequently we integrated Jenkins pipeline for automating the complete cycle.

2. Different FPGA boards were considered to synthesize the design, also due to the memory limitations it was not possible to prototype the complete design, hence a scaled downed design was synthesized on VCU118 and Arty A7 FPGA kit.

3. Using Jenkins programming bit file was generated along with synthesizing and implementing design on the FPGA kit.

4. Different software test programs were developed for verifying the functionality of different modules involved in the design.

5. After running these software test programs simulation was checked to ensure accurate functionality of the design under test.

6. Finally, after synthesizing the design power analysis was performed to evaluate the power efficiency of the design using Xilinx vivado.

1.3 Contributions

This work explores the rapid prototyping and functional verification of AI processor on the FPGAs. This AI processor provided by our industry partner is highly scalable and aims at low power consumption. Due to its low power capability the design was synthesized on the FPGAs.

Prototyping a design involves several steps hence while designing using register transfer language (RTL) it is time consuming to modify RTL, perform placement and route, synthesise the design and at last generate a programming bit file regularly. Integrating Jenkins with the design helped to complete all these tasks at a much faster pace. Usage of TCL scripting and Shell scripting for automating jobs has aided the flow with increased time efficiency.

Testing the design is a crucial part while developing using the RTL approach. Traditional method for testing a design is by using RTL-level testbench, but this process is extremely time consuming. In this thesis software test programs using C language were developed to verify the functionality of each module used in the design. Usage of such test programs not only saved time but also provided us with the flexibility in testing as it didn't rely on the status of other modules. Adapting this approach saved around 50 percent of the time as compared to traditional testbench based methodology.

1.4 Thesis Outline

The remainder of this thesis is structured as follows:

Chapter 2 provides a review of background information about the Field Programmable Gate Arrays (FPGAs) and different hardware design methodologies available to develop a design.

Chapter 3 provides an overview on artificial intelligence along with highlighting importance of convolutional neural network as the prototyped design is closely related to it. An overview of low power AI processor provided by our industry partner is provided in detail, highlighting different components involved in the design. Later, complete working of the processor and functionality of different modules available in the design are described briefly. Different hardware platforms and CAD tools that were used in prototyping and verifying the design are described. Brief summary on available resources is provided for two different FPGA kits that were used in this research. Later, all the steps associated with generating the programmable bit file required to program FPGA boards are explained with the help of flow diagram. Finally, functional verification of the design using different methodologies is explained with the help of table.

In chapter 4 test setup and test approaches are illustrated and later functionality, performance and hardware realization results of the low power Aarish AI processor are discussed in brief. The design was synthesized for FPGA kits mentioned above and complete summary on utilization of the resources by individual modules are highlighted with the help of tables and pie charts. Finally, this chapter displays power consumption results obtained for both FPGA kits used in this thesis.

Finally, in chapter 5, we have concluded with a summary of this thesis and provided suggestions for further research and future work.

Chapter 2

Background and Related Work

2.1 Motivation for Rapid Prototyping using FPGAs

Field Programmable Gate Arrays (FPGAs) are semiconductor devices which are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. Fig 2.1 highlight CLBs, Programmable Interconnects and Input/Output Blocks as mentioned above. The main advantage of using FPGAs is they can be re-programmed to any application or design after manufacturing. This is what makes an FPGA standout in comparison to Application Specific Integrated Circuits (ASICs) which are manufactured only for a particular design or task [2].

Normally a logic block comprises of few logic cells (each cell is also termed as adaptive logic module (ALM), a logic element (LE), slice, etc.). A typical cell comprises of a 4-input lookup table (LUT), a multiplexer and a D-type flip-flop (DFF) as shown in Fig 2.2.

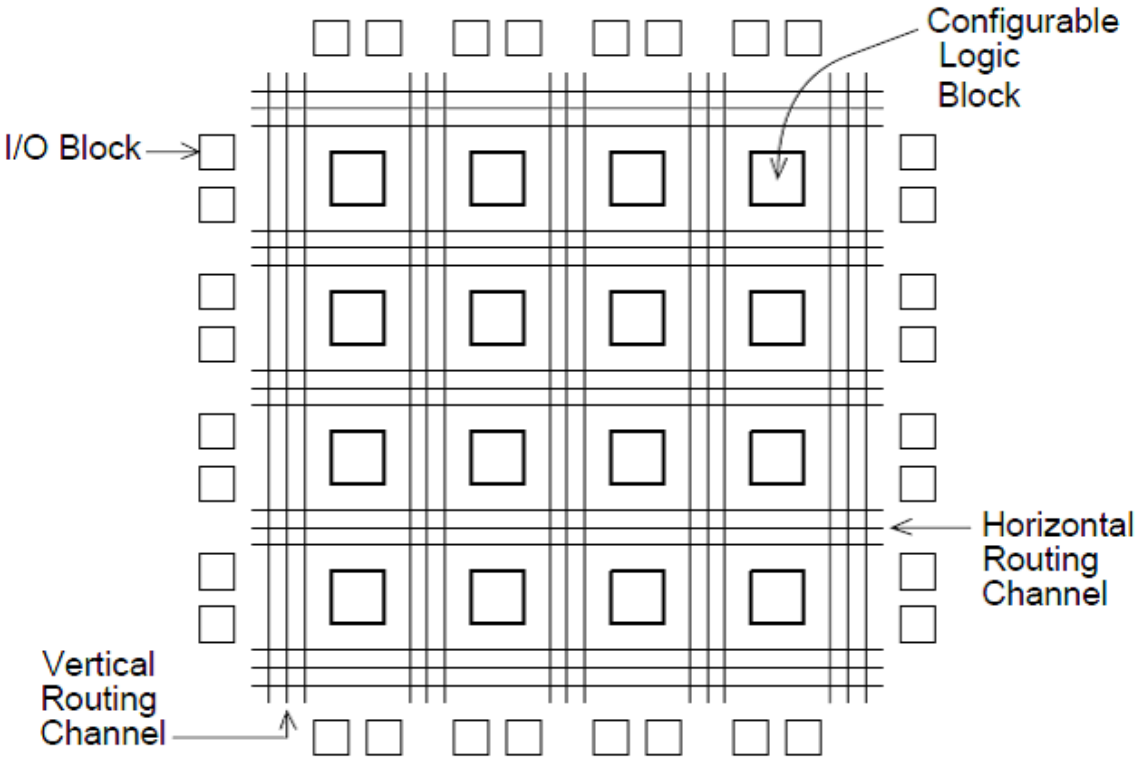

Figure 2.1: Layout of FPGA [1]

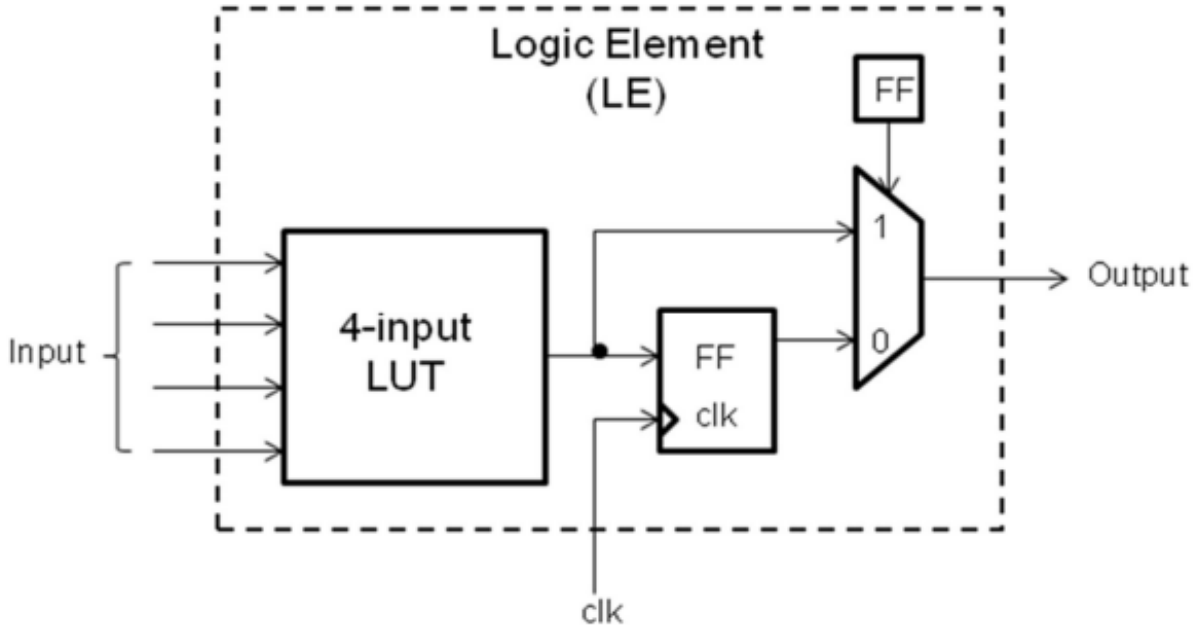

Figure 2.2: Structure of FPGA [6]

2.2 Rapid Prototyping Hardware design methodologies

Hardware design is done wither to execute a specific task (single purpose) or variety of task (general purpose). Example of general-purpose IC (Integrated Circuit) is CPU in a computer/laptop as it can execute multiple tasks in any given frame of time. One such example of an IC dedicated to a fixed task is an IC used in image compression and decompression. Designing hardware involves three main steps:

1. Analysis - This is the beginning where we define the algorithm of the system.

2. RTL generation and verification - Here coding and verifying reference model of the system and coding and verifying RTL model from the reference model takes place.

3. Physical Target - After successful completion of the above step the design is either fabricated on an ASIC or programmed on to an FPGA.

In order to perform the step 2 there are two ways -

1. RTL (Register Transfer Level) - Here a hardware description language (HDL) such as Verilog, system Verilog and VHDL is used to configure an FPGA at RTL level. This is often referred to Manual flow of developing a design.

2. HLS (High Level Synthesis) - Here the FPGA is programmed using High Level Synthesis, this is an automated way of generation an RTL.

The Fig 2.3 shows complete design flow. Both hardware design techniques have their own advantages and disadvantages. HLS technique is believed to be a better option in most of the designing applications because of its speed. Also using HLS brings better reach among engineers because one can write the design in C, C++ or system C and perform high level synthesis to generate the RTL without knowing VHDL or Verilog languages. Hence HLS automates generation of RTL and perform floor planning, placement and routing in optimised way. On the other hand, the manual way of design is a slow process of generating the RTL because unlike HLS, engineers must manually code their complete design using VHDL, Verilog and system Verilog languages, perform synthesis and generate RTL which is a very complex and at the same time extremely time-consuming task. But the main advantage of manual way of generating the RTL lies in the ability to program the design for any corner condition which the HLS might not consider. This technique gives the programmer flexibility to generate the design at the minute level of detailing which in turn increases the performance and robustness of the design at the cost of some excess time. Hence to summarise despite HLS appearing to be a better option for generating the reference model and synthesis Aarish AI processor is developed using the manual way of designing the RTL. HLS is still not believed to be ready for practical applications to that extent for large scaled applications, reason being HLS flow doesn't work with

templates. The area, speed and power consumption results provided by HLS may not be acceptable for custom IC designs such as Aarish AI processor. RTL level design, although time consuming and expensive allows for greater optimization of design metrics such as area, power and speed consumption.

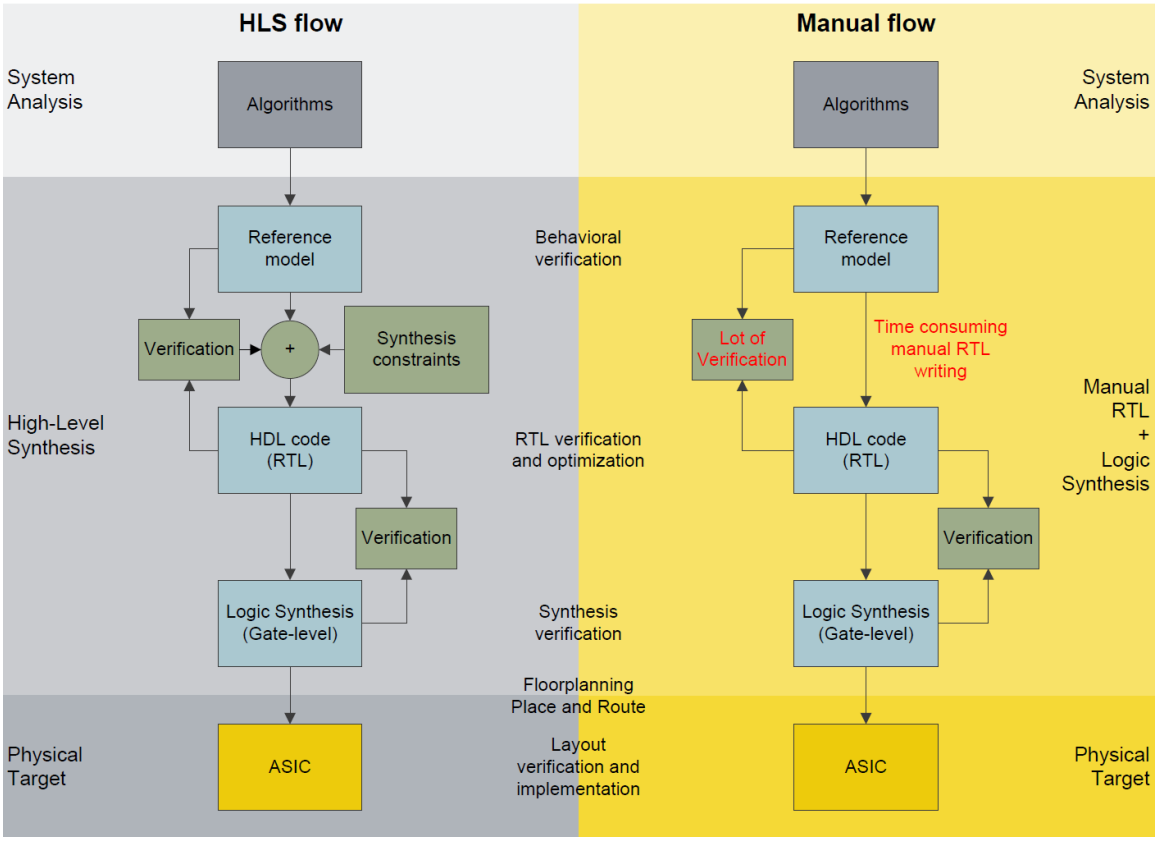

Figure 2.3: Hardware Design Techniques [7]

2.3 Related Work

ASIC designs continue to increase in size, complexity, and cost; at the same time aggressive competition has made electronics markets extremely sensitive to time-to-market pressures. A typical ASIC design cycle is in the order of 9-18 months, while the window of opportunity for the introduction of a product using this device can be as little as 2 to 4 months [**?**]. Failing to have a product available within this time period may result in significant reduced revenue. These factors have dramatically

increased the pressure for ASIC designs to be "right-first-time" with no re-spins. In turn, this has driven the demand for fast, efficient, and cost-effective verification at both the chip and system levels using FPGAs. Several research works have been done in recent years for implementing rapid prototyping of different processors on FPGA and performing functional verification of the design. In this section some of the research work related to FPGA prototyping and functional verification is reviewed.

In [**?**], a faster system level prototyping approach was used for IP using Xilinx Embedded Development Kit (EDK) flow approach. In addition to prototyping this paper used pre-verified standard designs along with design under test to validate IP. Proposed method utilized Xilinx EDK to build system with microblaze processor on FPGA, and Software Development Kit (SDK) to develop application on microblaze which that could control the stimulus to IP.

In [**?**] lower-level Hardware Description Language (HDL) implementation of edge detection algorithm was realized on FPGA and compared with its implementation in CPU. Implementation was performed on Xilinx Spartan 3 FPGA, which is inexpensive and not powerful FPGA and developed several micro processing units in parallel inside FPGA to enhance its processing speed. Using FPGA hardware implementation, this research achieved 4.4 times faster performance in edge detection compared to software implementation in CPU.

In [**?**] findings from multiple studies on functional verification trends are addressed along with their impact into the state of FPGA market in terms of both design and verification trends. This paper highlights various techniques used for functional verification of a design and compared FPGA project results in terms of verification effectiveness. This paper also described difficulties such as non-trivial bug escapes, different types of flaws resulting in non-trivial bug escapes that were encountered due to inefficient functional verification techniques.

In [**?**] P4FPGA, a new tool for developing and evaluating data plane applications is presented. P4FPGA is an open-source compiler and runtime. The compiler extended

the P4.org reference compiler with a custom backend that generated the FPGA code. P4FPGA supported different architecture configurations, depending on the needs of application. This paper concluded that code generated by P4FPGA ran at line-rate at all packet size with latencies comparable to commercial ASICS thereby providing developers to rapidly prototype and deploy new applications.

In [25] a modified Non-Local Means (NLM) filter for speech enhancement was prototyped on FPGA. The hardware architecture of the proposed method was designed and verified by implementing it on Zynq-7000 FGPA using Xilinx system generator. This research concluded with better performance results after successful completion of prototyping modified design on the FPGA.

Chapter 3

Rapid Prototyping and Functional Verification of AI Processor

In this chapter we provide an overview of artificial intelligence (AI) and convolutional neural networks (CNNs) as the processor that we have used has an unique method of training CNNs. Later we provide a detailed overview of this low power AI processor provided by our industry partner.

3.1 Overview of Artificial Intelligence

Fig 3.4 shows the relationship of neural networks with the whole of artificial intelligence. The insight of designing an effective machine learning algorithm is clear. Instead of hundreds of trial-and-error approaches of developing a different, custom application to solve each and every distinct problem in a domain, the single machine learning algorithm just needs to learn via the approach called training, to take care of every new problem occurring. The way artificial intelligence works is associated with that of brain as it is often referred to brain-inspired computation. Any artificial intelligence is associated with millions of neurons, the representation of one such neuron is show in Fig 3.5. The neuron takes in a signal entering it via the dendrites later

performs computation on it and produces a signal on the axon. These incoming and outgoing signals are referred as activations. The axon of one neuron branches out and is connected to dendrites of many other neurons. The connection between a branch of axon and a dendrite is called a synapse. The important highlight of synapse is that it can scale the signal crossing it as shown in the fig 3.5. This scaling factor is referred to as a weight.

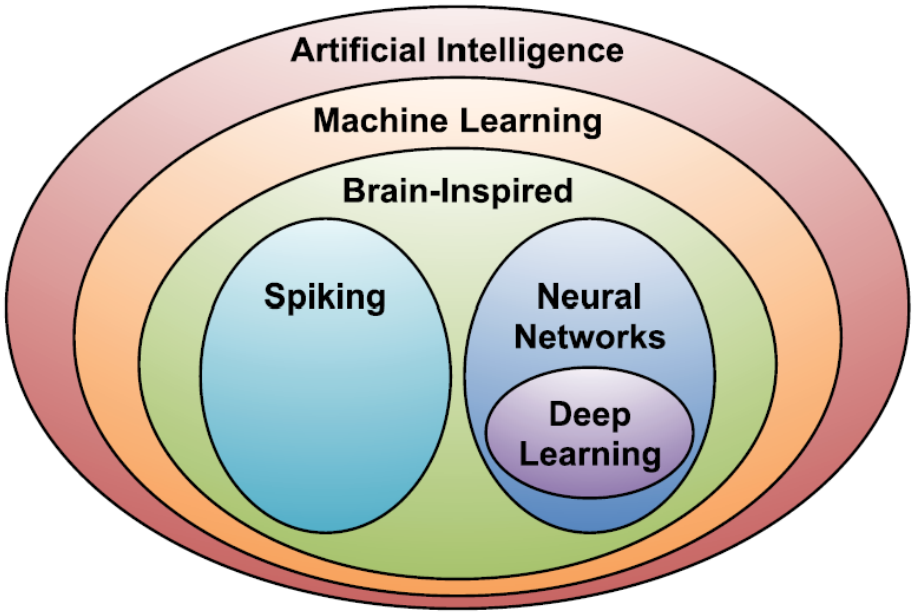

Figure 3.4: Context of Artificial Intelligence [5]

3.2 Overview of Convolutional Neural Network

Fig 3.6 shows multiple layers present in the Convolutional Neural Network. In such networks each layer generates a successively higher layer of abstraction of the input data, called a feature map (fmap), which preserves essential yet unique information. Modern CNNs can achieve superior performance by employing a very deep hierarchy of layers. Each of the CONV layers shown in the Fig 3.6 is composed of high dimensional convolutions. In this computation the input activations of a layer are structured as a set of 2-D input feature maps (ifmaps), each of which is called a

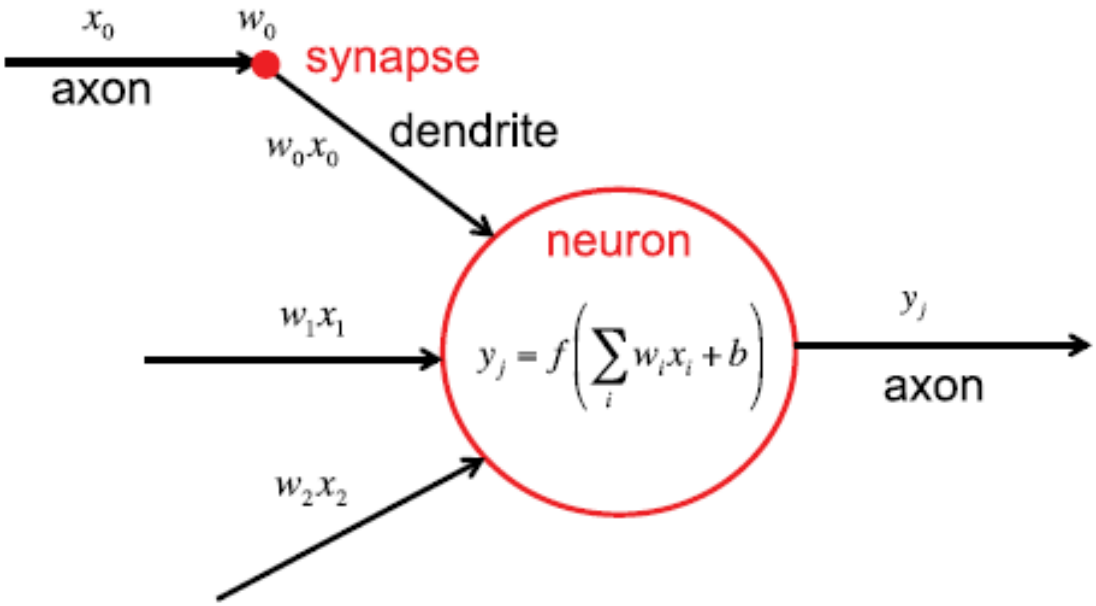

Figure 3.5: Connections to neuron [5]

channel. Each channel is convolved with a distinct 2-D filter from the stack of the filters, one for each channel; this stack of 2-D filters is often referred to as a single 3-D filter. Finally multiple input feature maps may be processed together as a batch to potentially improve the reuse of the filter weights. In general, from five to more than a thousand CONV layers are commonly used in recent CNN models. In addition to CONV (convolutional) and FC (fully connected) layers, various optional layers are present such as the nonlinearity, pooling and normalization.

3.3 Overview of low power AI processor

An important feature of this low power AI processor provided by our industry partner, which is a CNN processor, is in the method used for training CNNs. This low power AI processor comprises of core compute circuit elements and each element is designed to perform certain CNN function. Each of these circuit elements comprises of memory which holds the weight used to perform different CNN functions. The

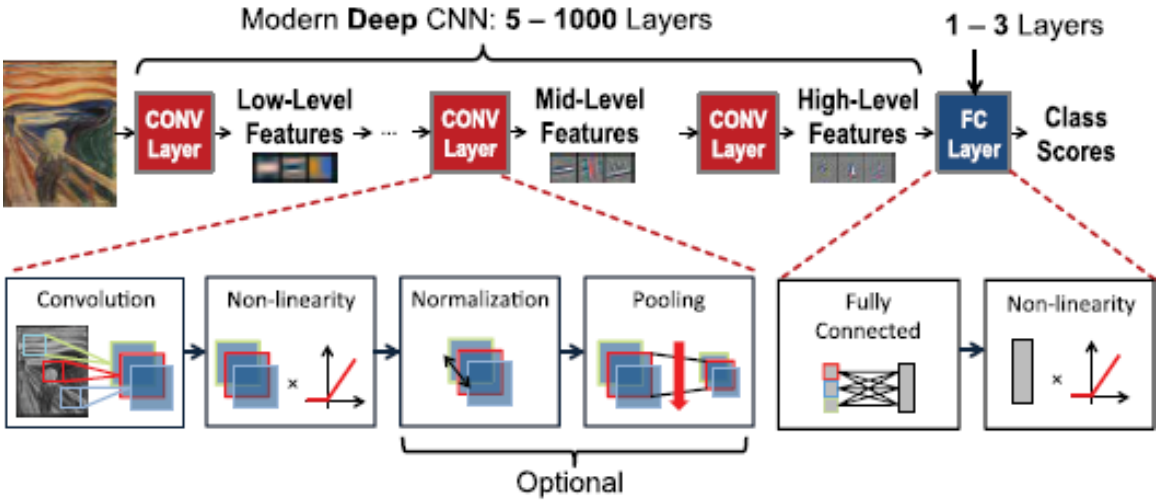

Figure 3.6: Convolutional Neural Network [5]

architecture of CNN processor where the most of the computation takes place is as shown in the Fig 3.7 which is called Programmable Functional Array or CNN processor. It is comprised of four Programmable Functional Units (PFUs). Each PFU as shown in the Fig 3.8 is comprised of 4 core compute elements. These elements consist of Arithmetic Logic Units (ALUs) which have their own memory and control logic and can communicate with each other. Also, there is an active memory buffer also referred as intelligent memory buffer which stores results of computations performed. Each of the core compute circuit elements are accessible through few read and write ports of the memory buffer. PFU further includes an input data interface and an output data interface. Input data received via the input data interface and output data sent via output data interface could directly interface with a read and write port, respectively, within the intelligent memory buffer. This allows other PFU units to communicate with each other on a point-to-point basis via the read and write ports based on a transmitter and receiver configuration. A read port and a write port can also be used to serialize and de-serialize data to be communicated over the serial to parallel interface such as an SPI, with the other PFUs on a different

chip. The SPI 308 in Fig 3.7 can provide a low power implementation of a communication channel between two PFUs across the chip boundary. The PFU displayed is implemented using a single chip, data sent via the parallel interface within the PFU chip can be serialized and transmitted over a PCB and then parallelized once received at the destination chip (could be a second PFU). The serial link could be any kind of serial link, from a simple SPI to a more complicated clock embedded link. The PFU also includes an interface with an external memory buffer outside the PFU for the core compute elements to access a larger pool of memory. In a typical CNN, only a few layers need PFU configured with only enough weight memory to store an average number of weights for processing/computing a CNN layer. Whenever a core compute element needs to access a larger amount of weight memory, it can fetch from the external larger pool of memory. However, the memory bandwidth for the external memory may be sufficient to support two core compute elements without any back pressure. Any larger number of core compute circuit element accessing the larger pool of weight memory may result in reduced throughput. When a particular convolution operation does not fit in single core compute element due to weight memory constraint, a convolutional transformation can also be utilised to split the convolution across multiple core compute elements. This mechanism allows regular PFUs to be restricted to a relatively low amount of weight memory, across multiple core compute elements using convolution transformations. Generally, all the traditional architectures use load/store architecture whereas the configurable CNN processor uses dataflow architecture that makes the processor highly efficient. Fig 3.9 shows simplified PFU with an active memory buffer and core circuit elements. This processor has many use cases such as:

1. Image Recognition
2. Object Detection
3. Lip Movement to Speech

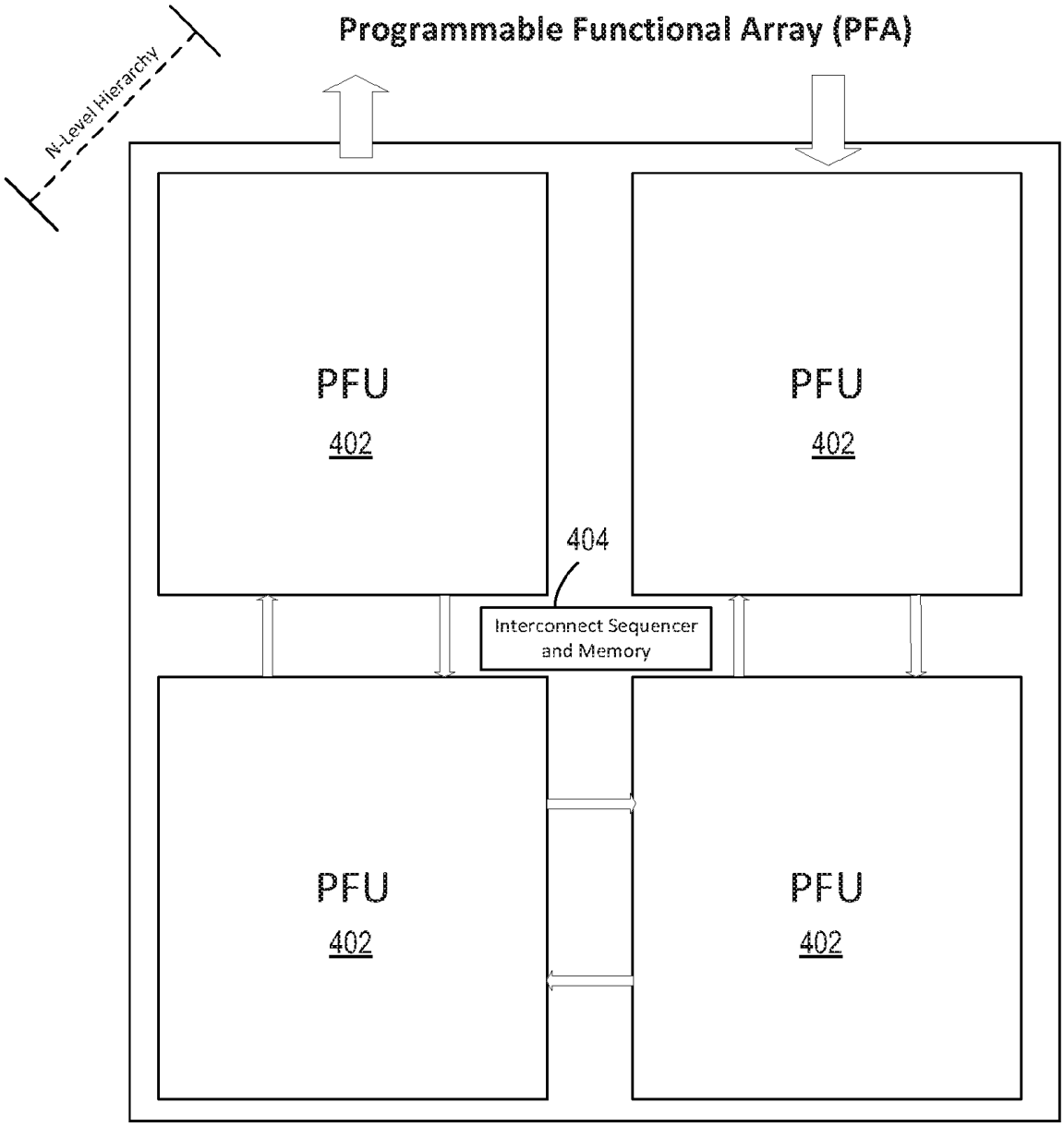

Figure 3.7: Representation of Programmable Functional Array (PFA) [8]

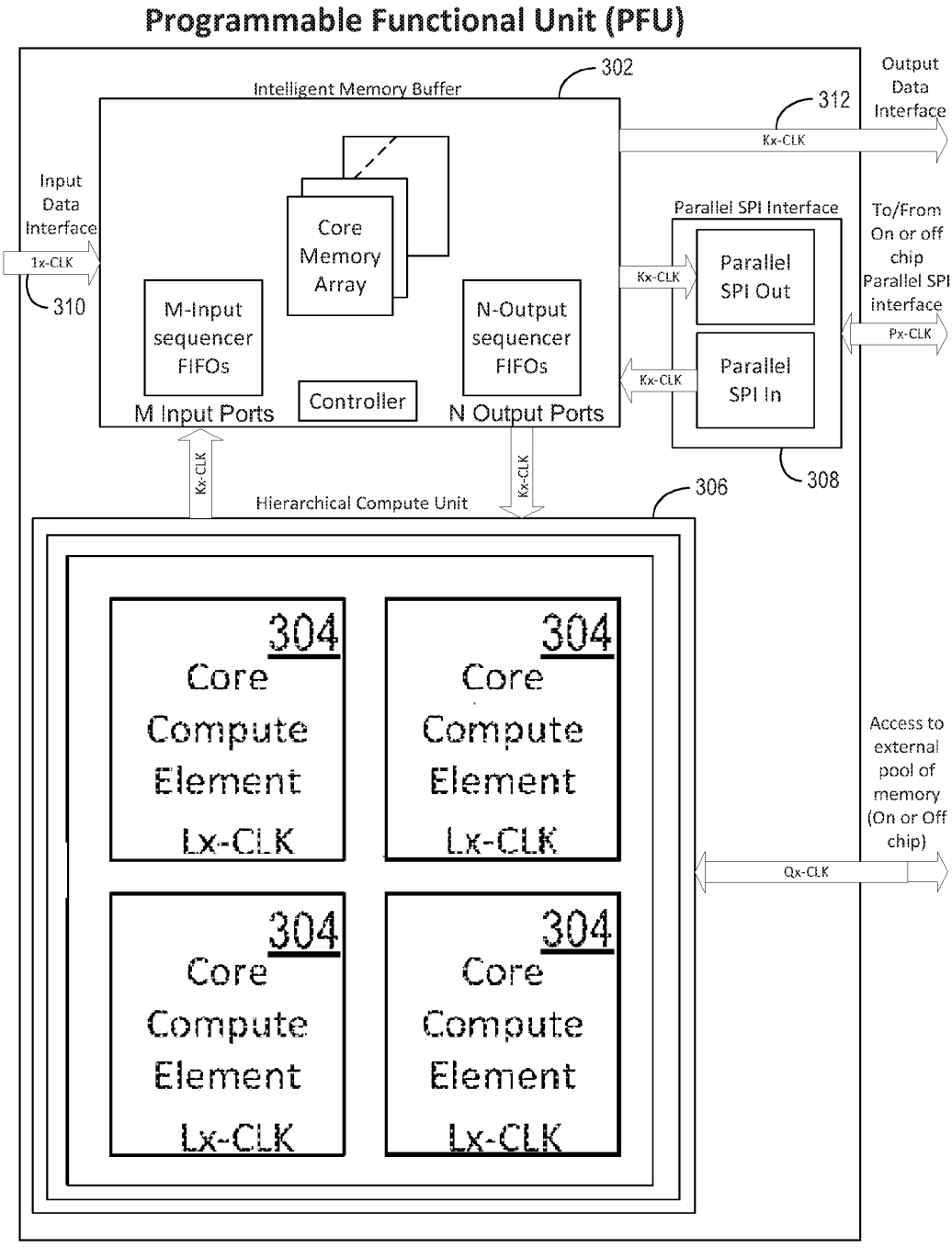

Figure 3.8: Representation of Programmable Functional Unit (PFU) [8]

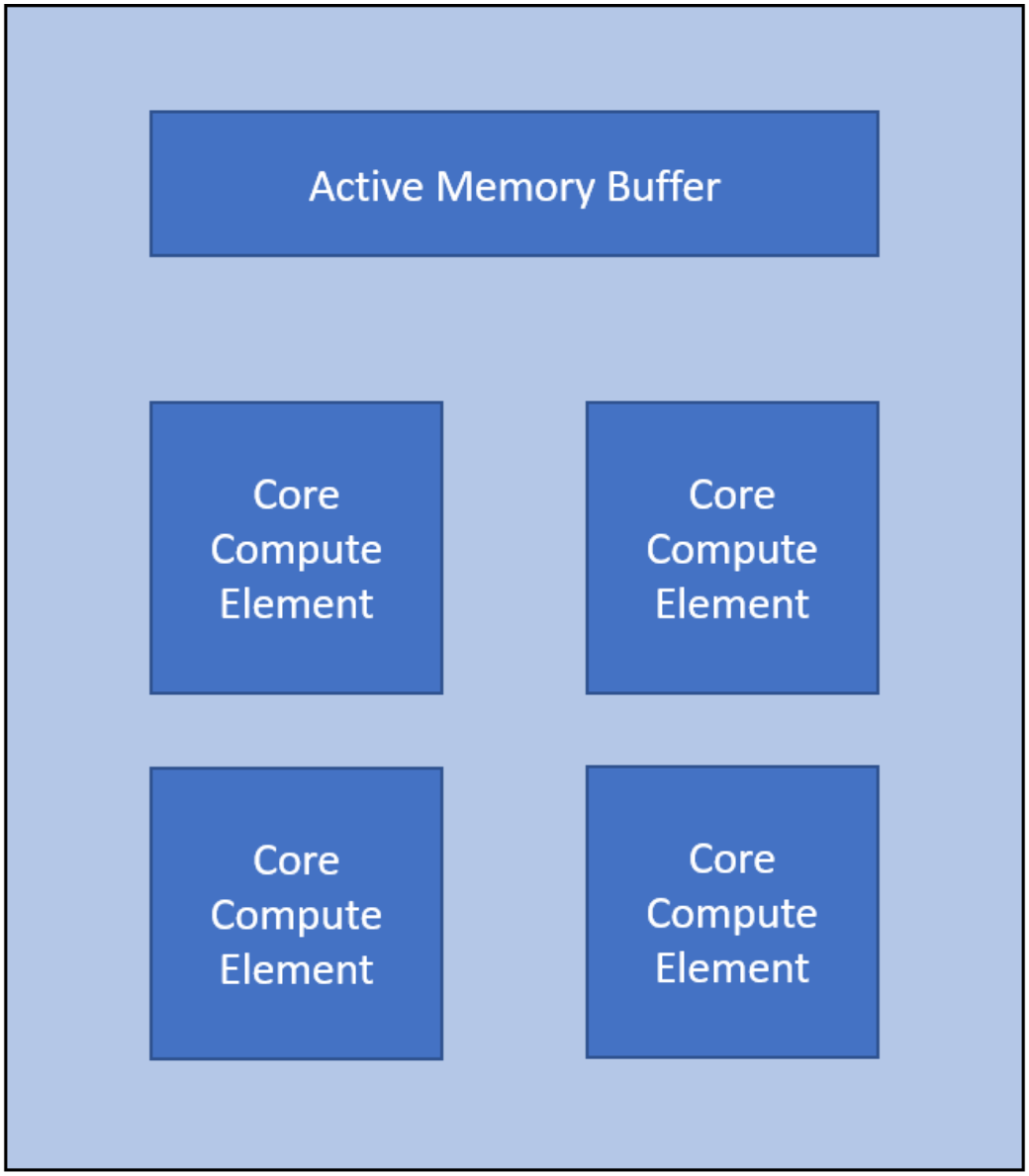

Figure 3.9: Simplified PFU [8]

3.4 Operation of low power AI processor

Fig 3.10 illustrates the complete CNN architecture. This architecture includes a configurable PFA/CNN sub processor, a micro controller (RISC-V processor as shown) the mobile industry processor interface (MIPI) subsystem and standard input output devices such as universal asynchronous receiver-transmitter (UART), general purpose input output (GPIO), serial peripheral interface (SPI) and inter-integrated circuit (I2C). All the memory in the system (both the CNN sub-processor and the RISC-V microcontroller) can be memory mapped and can be accessible by different masters driving the internal memory bus. The programming of the PFA/CNN sub-processor can be done in one of the two different modes available.

1. by the internal microprocessor configuring the CNN processor from the SPI-FLASH

2. by the application processor (shown in Fig 3.11) likely coupled to MIPI sink through standard IO interfaces such as SPI, I2C and UART that are master devices on the memory bus. In the second mode, the SPI-FLASH is eliminated.

In the first mode, the RISC-V internal processor is responsible for different housekeeping functions and can also be used for computation when required. It keeps the track of the state of the CNN sub-processor for interfacing with the outside world. The RISC-V internal processor can also handle any kind of exception that may occur in the system at runtime in a flexible way. In one aspect, the RISC-V processor can be an instruction set architecture controller covered by an open-source license, making the processor easy to adopt. The RISC-V processor is optional. In a case without the RISC-V processor, the application processor could configure the configurable CNN sub-processor via SPI/UART/I2C bus. In such case, the CNN sub-processor output could be read by the application processor, or the CNN sub-processor could embed the output onto MIPI frames and then send those to the application processor. Though, having RISC-V processor makes the system flexible as it can program sub-processor

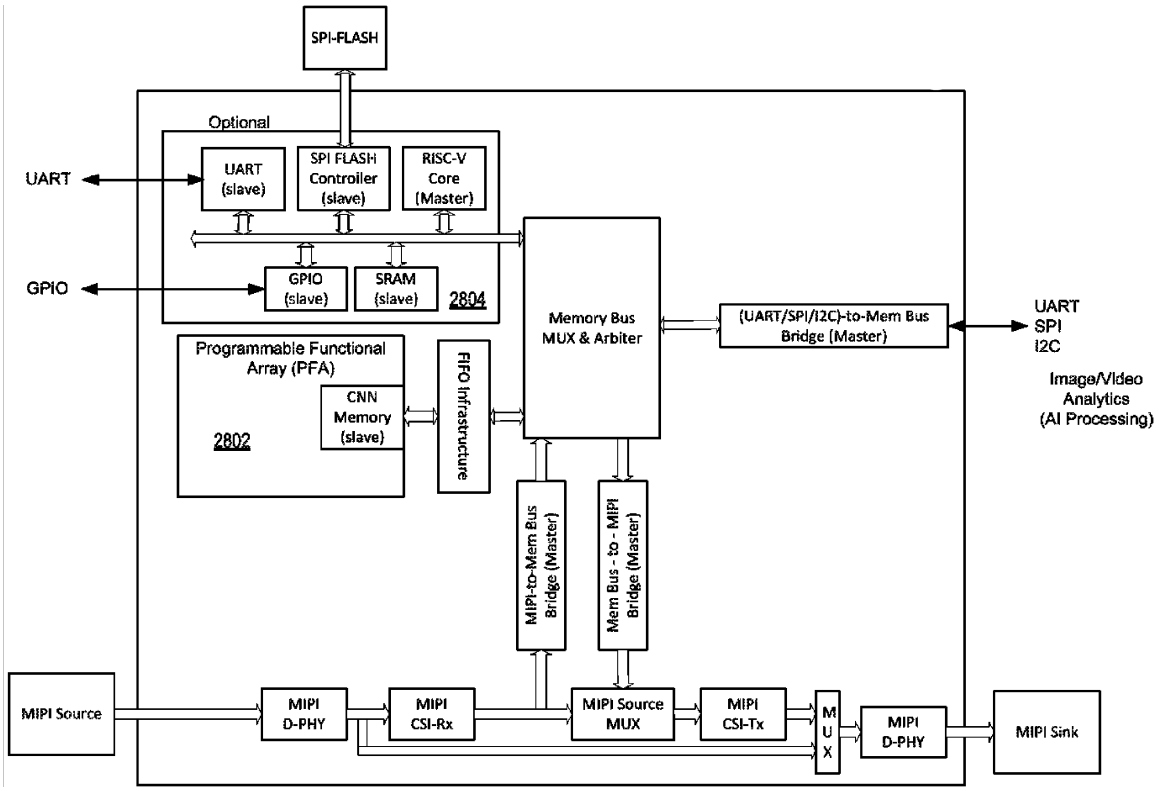

Figure 3.10: Architecture of CNN Processor [8]

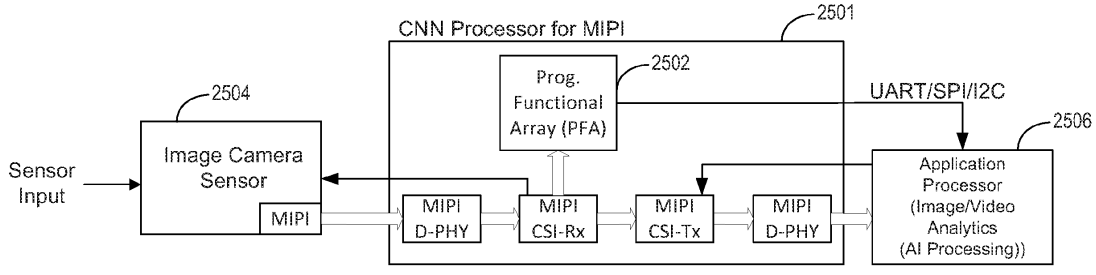

Figure 3.11: Architecture of CNN Processor [8]

at power on directly from the SPI-FLASH without waiting on application processor to do programming.

The MIPI system shown in Fig 3.10 is a technical specification for the mobile ecosystem, particularly smart phones but including other mobile industries. It has different components in the complete subsystem. First MIPI source inputs the image sensor data for processing at CNN sub-processor. This image sensor data is sent to CNN sub processor through MIPI CSI Rx and MIPI D-PHY. MIPI CSI Rx is camera serial interface is a specification of the Mobile Industry Processor Interface alliance which defines an interface between camera and host processor. MIPI D-PHY is physical layer between camera and application processor. This disclosure describes systems that can process the data in general and generate analytics. One of the example applications is to process image/video data. Analytics could be in the form of tasks such as object detection/recognition from a scene, image enhancement form low lighting conditions or any form of intelligent tasks that are intended to be computed either on a frame basis or on a group of frames defined as a video sequence. Recognition of video sequence could include temporal dependencies such as action recognition etc.

So far, we described a CNN processor configured for MIPI that inputs image sensor data processes it at programmable functional array (after receiving it via MIPI D-PHY and MIPI CSI-Rx). These processed analytics of the PFA/CNN sub-processor could be provided in two forms.

1. In one form, the output analytics can be defined by a few data-words such as classification result.

2. In the other form, the processed output can be defined in significant amount of data.

These computed analytics can be sent to application processor in two different ways.

1. In the first the analytics is sent over standard communication bus such as SPI/UART/I2C to application processor.

2. In the second the computed data is directly embedded on MIPI CSI Tx to the application processor and it successfully extracts the analytics form the MIPI-CSI bus. This data is sent in form of MIPI frames.

Although the overall power of the system can be minimised by eliminating the need of an application processor to transfer data specifically. This further reduces the overall system latency. In sleep mode, the default dataflow path is from the input to the output of the MIPI-CSI bus bypassing the PFA/CNN sub-processor. This ensures that power for the CNN processor is consumed or is primarily consumed, only when the PFA/CNN sub-processor is used to compute. In one aspect the MIPI bus can be implemented using a flexible cable. In such case the CNN processor can be disposed serially along the bus, and along the cable. In one aspect the CNN sub-processor can be implemented using any of the PFAs described herein. In one aspect the CNN sub-processor can be implemented using a traditional instruction set architecture processor [e.g., load/store processor].

3.5 Hardware platform and CAD tools used for rapid prototyping

Prototyping a design on a FPGA is an important step in verifying the functionality of design, hence we have prototyped the design on Artix-7 and Virtex VCU118 FPGA kits to test individual modules. To verify the functionality we have developed an application based on C that can be used to communicate with the FPGA board. Using this application, we can point to different registers and compare their present value with the expected value. [11]

3.5.1 Overview of Arty 7 Board

Arty is a ready to use development platform designed around the Artix-7 FPGA from Xilinx. Unlike other Single Board Computers, Arty isn't bound to a single set of processing peripherals. It is a communication powerhouse chock-full of UARTs, SPIs, I2Cs and an Ethernet MAC, and next it has dozen 32-bit timers. This board has following features.

1. It has 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)
2. 1800 Kbits of RAM
3. 90 DSP slices
4. 16MB Quad-SPI Flash
5. Programmable over JTAG and Quad-SPI Flash
6. USB-UART Bridge

Outline for Arty 7 FPGA board is presented in fig 3.12 and the pin callouts for the board as highlighted in Fig 3.12 are displayed in Table 3.1.

3.5.2 Overview of Xilinx VCU118 kit

VCU118 is a general-purpose evaluation board for rapid-prototyping based on the Zynq Ultrascale + XCVU9P-FLGA2104-21-E MPSoC (multiprocessor system-on-chip). High speed DDR4 SODIMM and component memory interfaces, FMC expansion ports, multi-gigabit per second serial transceivers, a variety of peripheral interfaces, and FPGA logic for user customized designs makes it a flexible prototyping platform. The block diagram for VCU118 board is as shown in Fig 3.13

Each FPGA board is capable of programming certain size of design. Resources such as logic cells, CLB flip-flops, RAM, DSP slices together defines the capacity of

Callout	Description
1	FPGA programming DONE LED
2	Shared USB JTAG / UART port
3	Ethernet connector
4	Power select jumper (Ext. supply / USB)
5	Power jack (for optional ext.supply)
6	Power good LED
7	User LEDs
8	User slide switches
9	User pushbuttons
10	Arduino/chipKIT shield connectors
11	SPI header (Arduino/chipKIT compatible)
12	chipKIT processor reset jumper
13	FPGA programming mode (JTAG/FLASH)
14	chipKIT processor reset
15	Pmod headers
16	FPGA programming reset button
17	SPI FLASH
18	Artix FPGA
19	Micron DDR3 memory
20	Analog devices ADP 5052 power supply

Table 3.1: Pin Callout in Xilinx ARTY 7 Board [9]

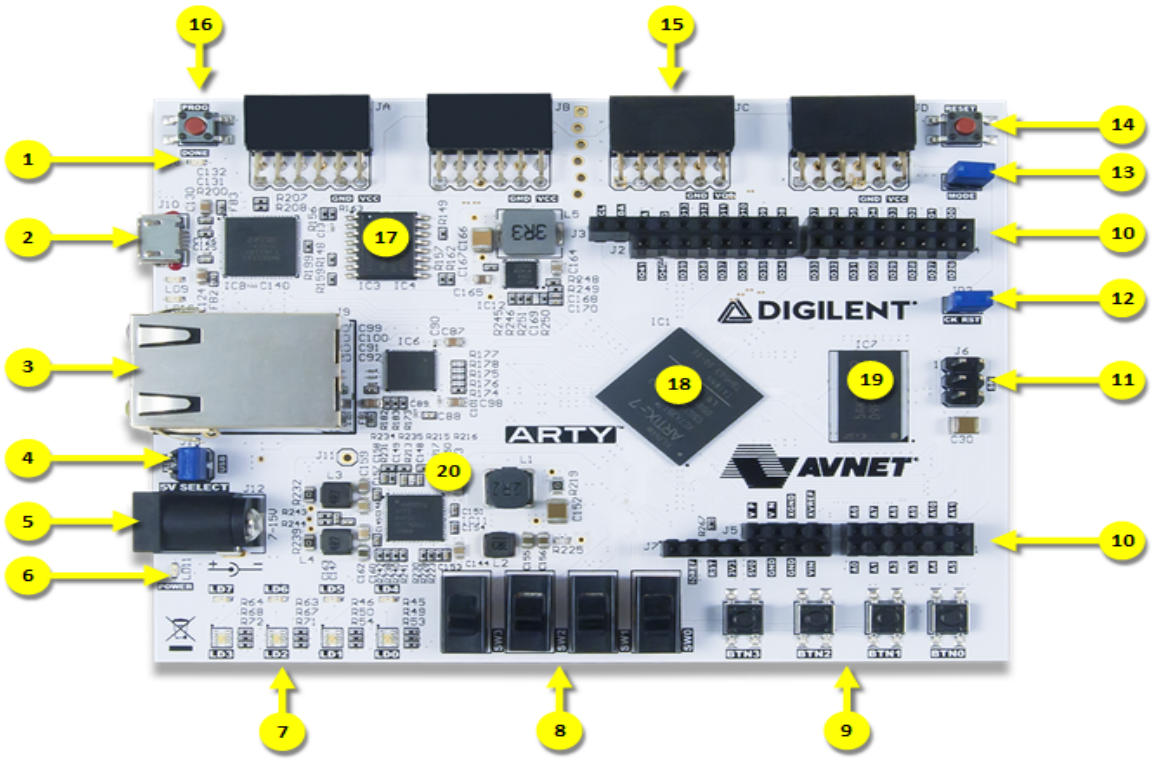

Figure 3.12: Arty 7 Board Layout [9]

any FPGA board. The Table 3.2 presented below highlights the resource count of Xilinx VCU118 FPGA board.

3.5.3 CAD tools used in FPGA Prototyping

CAD tools stands for computer-aided design tools for FPGAs, and they cannot be underestimated. FPGAs have large blocks of RAM, processors or analog-to-digital converters and all these features are not useful enough if they are not utilised to their full potential. In order to enhance capabality of a FPGA CAD tools are used, they can take a description of a digital circuit as input, along with constraints (e.g., on speed performance, are of power), and automatically map the circuit into the hardware blocks and perform the routing. These CAD tools optimize the speed performance, area and power consumption of the circuit implemented on FPGAs. Working with an FPGA involves use of CAD tools, and these tools are truly the "face" of vendor seen by engineers and designers in the field. CAD tools provide us with an environment

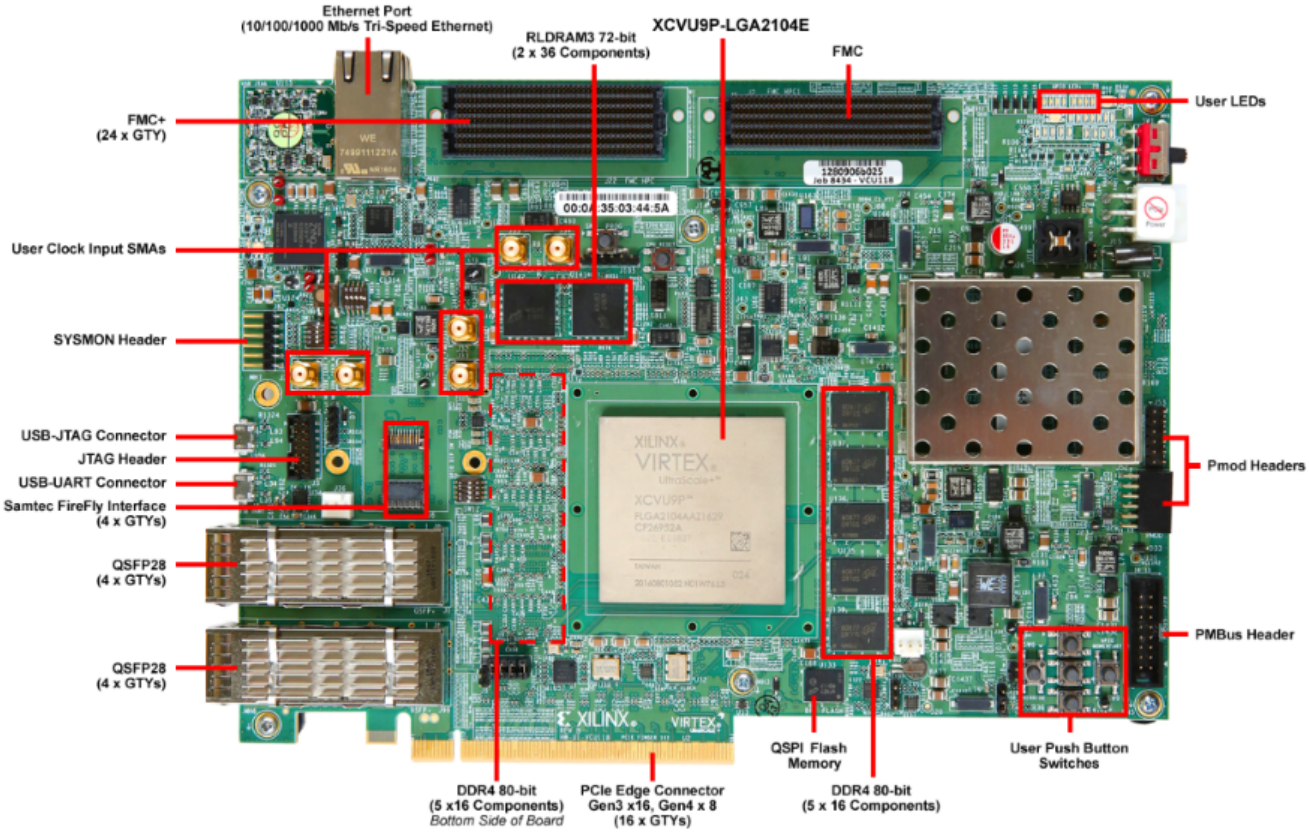

Figure 3.13: VCU118 Evaluation Board [10]

Feature	**Resource Count**
CLB LUTs	1182240
CLB Registers	2364480
CARRY8	147780
F7 Muxes	591120
F8 Muxes	295560
CLB	147780
LUT as logic	1182240
LUT as Memory	591840
Block RAM Tile	2160
DSPs	6840
Bonded IOB	832
HPIOB-M	384
HPIOB-S	384
HPIOB-SNGL	64
HPIOBDIFFINBUF	720
BITSLICE-RX-TX	1560
GLOBAL CLOCK BUFFERs	1800
MMCM	30

Table 3.2: Resources available in Xilinx VCU118 board [10]

which is closely relatable with the actual FPGA hardware wherein we can run the design, test its functionality. We can also check the waveforms which provides us the information about different registers and their state at a given time. Currently there are two major vendors in the market who predominantly work on developing such CAD tools. Vivado by Xilinx and Quartus Prime by Intel are the two well known CAD tools used for building designs that are programmed on FPGAs. These CAD tools supply a complete tool floe from RTL-to-bits, often free of charge to their customers and to universities. Alternative third-party tools are also available for the initial stages of the flow, such as the popular Synopsys (Synplicity) and Magma Design Automation tools and are known to produce excellent results. Historically the use of FPGA vendor tools has been mandatory for the back end of the flow, beginning with packing, however that may be changing as Synplicity now offers a flow encompassing packing, placement and physical synthesis. The functioning of these CAD tools is described with the help of CAD flow that is shown in Fig 3.14. Though is not shown in the Fig 3.14 simulation, testing and verification of the design can be done at any stage. In practice, many customers simulate their initial design specification (RTL Verilog or VHDL), and then do not simulate again. Rather designers leverage FPGA reconfigurability to accelerate their verification. After routing a bitstream is generated for the design, the FPGA is programmed, and verification is done in the lab using the actual hardware. Such a verification flow is impossible for custom IC technologies, yet it is feasible for programmable logic where designs can be modified, and devices reconfigured following the discovery of the design flaws. This diagram describes various steps involved in the CAD flow which are described below one by one.

1. The first step involves building a design specification using Very High-Speed Integrated Circuit(VHSIC) Hardware Description Language (.VHDL), Verilog (.v) or system Verilog (.sv) at the register-transfer level (RTL). While not entirely independent of the target FPGA, the early steps of the flow tend to be

more generic, while the later steps are specifically targeted to the specific hardware available.

2. The second step involves including constraints in the design. These are the .xdc files which contains complete information of the pin constraints for the design with respect to the device targeted.

3. The RTL synthesis step parses the input and transforms the VHDL/Verilog into a block-level circuit description, usually consisting of large blocks such as multipliers, adders, multiplexers, state machines, RAMs and chunks of generic Boolean logic. Logic synthesis later optimizes the circuit at the level of Boolean equations.

4. Next step is technology mapping, here the circuit is mapped from a generic form into an equivalent circuit composed of basic logic elements available on the target device, e.g., LUTs, registers and multiplexers. In this is also undergoes packing, which is also referred to as clustering, in which elements of the technology mapped circuit are packed into the logic blocks.

5. Next comes the place and route, in placement stage it is decided where each logic block should be located on the two-dimensional FPGA and these decisions are made with respect to increase the overall efficiency by taking in consideration factors such as speed, size etc. Routing forms the desired connections between the placed logic blocks. Finally, the bitstream is generated for programming the FPGA device.

Also the diagram shows a block named analysis, all the steps in CAD flow have an impact on the circuit speed and power therefore the tools must have access to such analysis data. Exact analysis of timing and power is impossible before routing is complete, so estimates are used at the earlier phases of the flow.

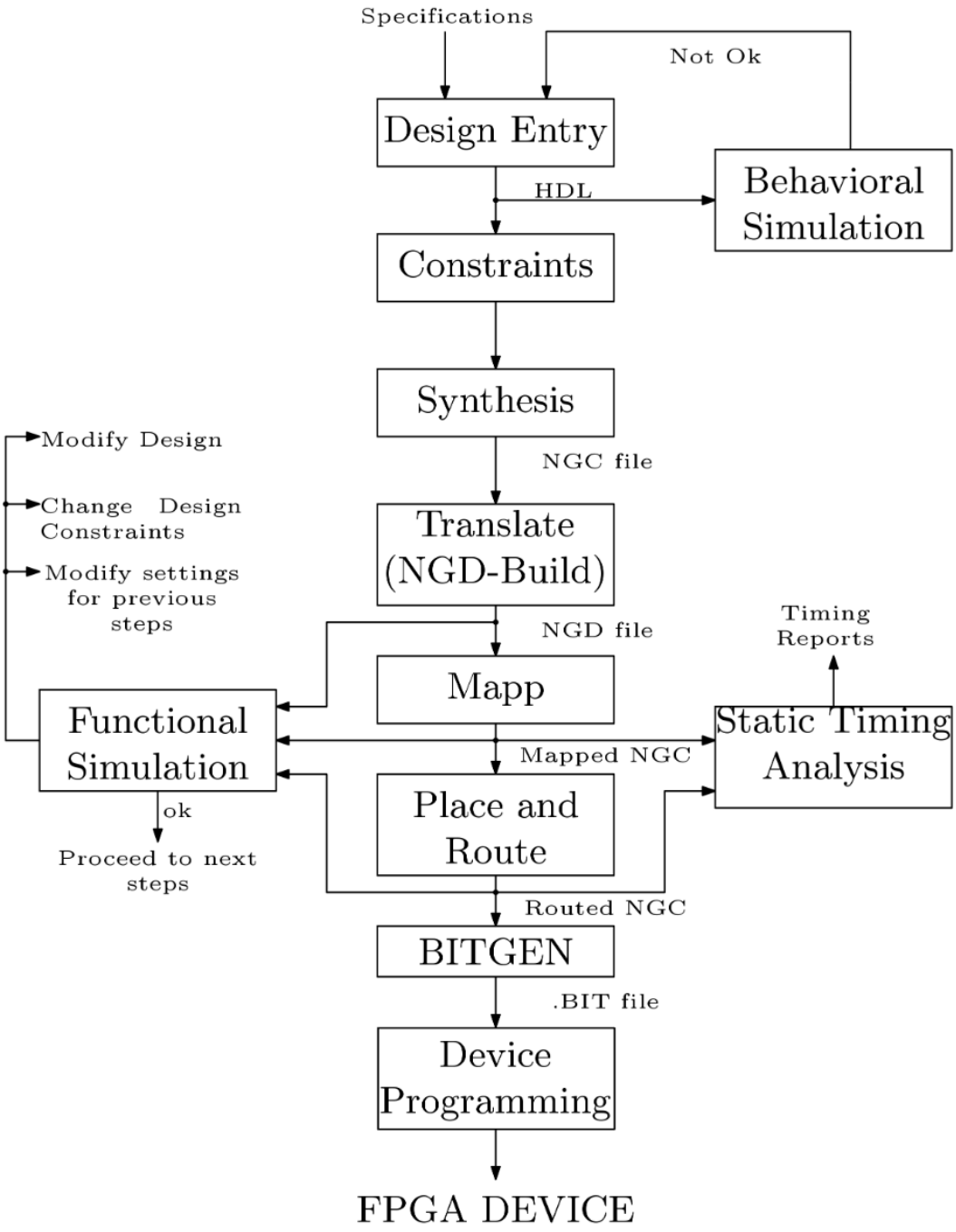

Figure 3.14: CAD flow

3.5.4 FPGA Prototyping sequence

Till now we have discussed different techniques used to design hardware in section 2.2 along with differnt CAD tools used as highlighted in section 3.5.3. This thesis prototyped low power AI design using Xilinx Vivado Lab Edition which is a compact and standalone product targeted for use in the lab environments. It provides aid for programming and logic/serial IO debug of all Vivado supported devices. Two important aspects of this design were hardware and software. First to prototype the design on hardware it was important to program the design on target FPGA board and to do so a programming (*.bit file*) file was required. Fig 3.14 highlights flow diagram that was followed for generating *.bit file*. To generate this file we have used Transaction Control Language (TCL) script which carried all the steps presented in the Fig 3.14. The hardware design was written in Verilog and compiled using Vivado. This TCL script instructed Vivado framework and performed the three main stages like synthesis, implementation and bitstream generation. This generated *.bit file* only contained hardware design; it was missing with software information related to the design. In early stages of development and debug we want to include the software in *.bit file* so that the system starts running even if the external interfaces of the design are not working. Software aspect of the design infers to the information related to functional verification of the design which is discussed in the next section.

3.6 Functional Verification of the design

To improve efficiency of FPGA prototyping, verification engineer must have a well planned and carefully thought-out verification plan. The verification plan should clearly state which parts of the design will be subjected to code or functional coverage and which parts will be exercised on FPGA. For those areas where FPGA is the focus, a detailed description of how to accomplish verification is required. It may seem like conservative approach to ignore the FPGA prototype when verifying the

design but when you are trying to reduce time to market it makes sense to approach verification holistically. It is better to apply effort in untested areas than to duplicate it verifying the same things in different ways. The traditional directed test verification should concentrate on the aspects of the design that cannot be accurately prototyped on FPGA. Almost all the design-for-test (DFT) areas and memories available in FPGAs fall in this category. The interface to these memories should be checked by traditional methods. With the increasing size and complexity of FPGA devices, there is a need for more efficient verification methods. Timing simulation can be the most revealing verification method; however, it is often one of the most difficult and time consuming for many designs. Timing simulations that traditionally were measured in days or weeks, can now for some projects be measured in few hours sometimes minutes, using standard desktop computers requiring high-powered 64-bit servers. This cuts into the time-to-market and cost-of-implementation advantages of using FPGAs in the first place. One of the biggest challenges that FPGA design and verification engineers face today is time and resource constraints. With FPGAs growing in speed, density and complexity, there is a lot of taxation not only on manpower but also on computer processors and available memory to complete a full timing verification. A research proved that there is an increase in requirement of verification engineers, latest study reported that in an FPGA market, requirement of design engineers increased by 4 percent while the requirement for test engineers increased by 10 percent. This increase in the requirement of test engineers highlights the importance of verifying a design to prevent re-spins thereby decreasing time-to-market and many more problems. Furthermore, there is an escalating challenge for the design and verification engineer (which many times can be the same person) to get proper testing of today's FPGA designs in shorter timeframes with an increased confidence of first-pass success [12]. The traditional FPGA verification methods are:

1. Functional simulation -: Functional simulation is a very important part of the

verification process, but it should not be the only part. When performing functional simulation, only functional capabilities of the RTL design are tested. It does not include timing information, nor does it take into consideration changes occurred to the original design due to implementation and optimization.

2. Static Timing Analysis / Formal Verification -: Most engineers see this as the only technique needed to verify that the design fulfills timing. There are lot of drawbacks in using this as the only timing analysis methodology. Static analysis cannot find any of the problems that can be seen while running a design dynamically. This analysis will only be able to show if the design can meet setup and hold requirements and generally is only as good as the timing constraints applied. In a real system, dynamic factors can cause timing violations on the FPGA. One example of this can be provided as Block Ram collisions. With the introduction of Dual Port Block Rams in FPGA devices, care should be taken not to read and write to the same location at the same time, as this would result in incorrect data being read back. Static analysis tools will never be able to find this problem. Similarly, if there are misconstrued time specifications, static timing analysis cannot find this problem.

3. In-System Testing -: Virtually every engineer relies on this method as the ultimate test. If the design works on the board and passes the test suites, then it is ready to be released. This is definitely a very good test, but it may not catch all the problems right away. At times the design needs to be running for quite some time before corner-case issues will manifest. Issues like timing violations may not manifest themselves in the same way in all chips. Usually by that time the design is in the end-customer's hands. This means high costs, downtime and frustration to try to figure out the problem. In order to get proper in-system testing completed, all the hardware hurdles will need to be realized such as problems with SSO, cross talk and other board related issues. If there are

external interfaces that needs to be connected prior to commencement of the in-system testing, this test will increase the time to market of the product [13].

So till now we have discussed need for functional verification and listed different FPGA verification methods [17]. Here we have described use of software based verification technique to perform functional verification of low power AI processor. Fig 3.10 indicates different modules that are involved in the design. Hence it becomes very important to verify that all these components are functioning accurately. To ensure that we verify the design completely, we developed different tests that allowed us to check correctness of each module. Fig 3.15 describes an example of a test case developed to verify functionality of uart module present in the design. These tests are software programs developed using C. Using software based verification methodology it becomes possible to access simulation of the test performed. Fig 3.16 describes simulation waveform for the test. In this example the UART module is configured with the default values to transmit random data and at the receiver side it is checked if the data received is the same as data transmitted. Fig 3.16 contains two signals - .rdata and .wdata. Similarly, the purple area in the waveform highlights modules involved in the test, by selecting a module all the signals defined in it can be monitored. The left section in the figure presents all the modules available in the design, by selecting a module we can check status of any desired signal or register. By looking at the waveform, it is concluded that data sent and data received are correct. Also, these waveform helps to understand number of clock cycles required to perform any test. All this information is considered while developing test cases to increase the efficiency of functional verification.

Table 3.3 shown below highlights different tests developed to ensure the functional verification [14] [15] [16]. This table only describes few tests but in order to perform the verification of each module associated with the design there are many different tests developed to ensure the functionality check covering all the corner cases in the

smoke_test.c

```c
#include "uart.h"
#include "../socc/printf.h"
#include "device.h"
#include "../socc.h"
#include "../socc/socc_test_status.h"

static const uint8_t kSendData[] = "Smoke test!";

void main(void) {
  uart_configure((uart_config_t){
      .baudrate = UART_BAUDRATE,
      .clk_freq_hz = CLOCK_CPU_FREQ_HZ,
      .parity_enable = false,
      .parity = kDifUartParityEven,
      .uart_option = 0,
    });

  uart_sys_loopback_set(true,0);
  uart_fifo_reset(kDifUartFifoResetAll,0);

  // Send all bytes in `kSendData`, and check that they are received via
  // the loopback mechanism.
  for (int i = 0; i < sizeof(kSendData); ++i) {
    uart_byte_send_polled(kSendData[i],0);
    CHECK((uart_byte_receive_polled(0) == kSendData[i]),"uart_smoke_test : UART data received did not match");
  }
  TEST_PASSED("uart_smoke_test");
}
```

Figure 3.15: Test for UART module

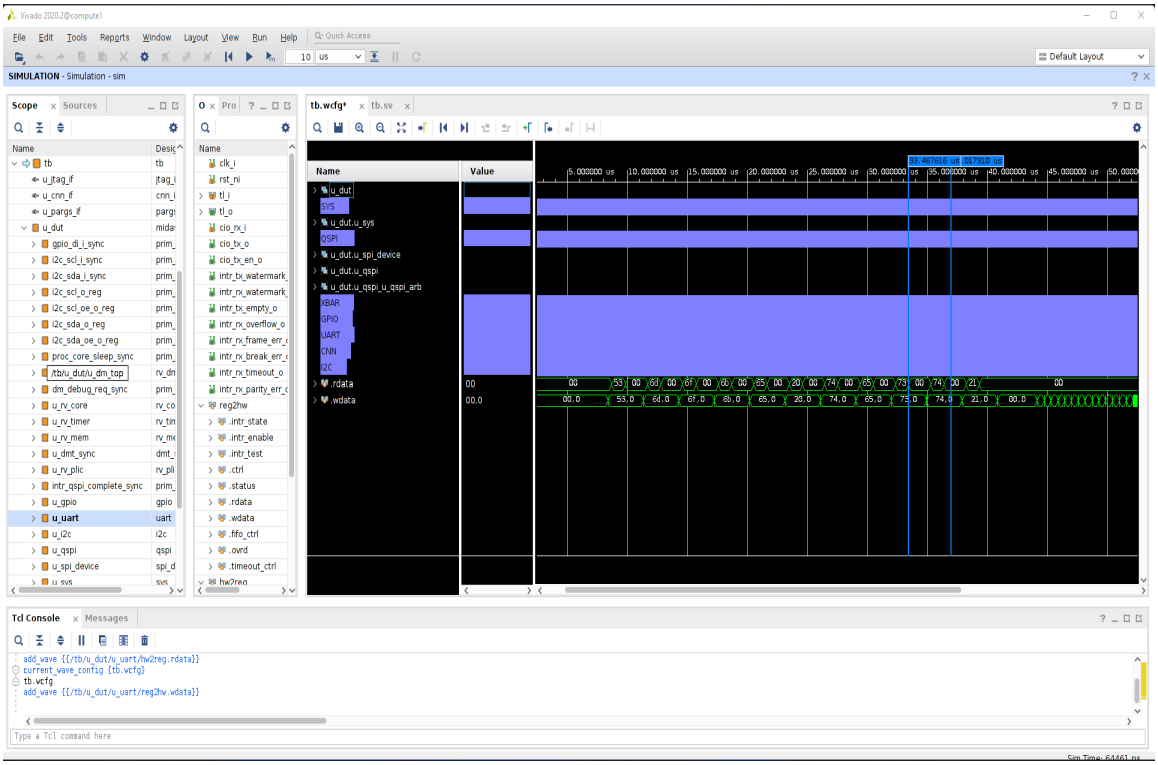

Figure 3.16: Waveform for UART smoketest

design.

Test Name	**Description**
Smoke Test	This test ensures the simple transaction of the device, for example in UART it will check if the device send the data and successfully receive the data.
Interrupt Test	This test checks if the respective interrupt is triggered while execution of a task, for example in qspi it will check kTopMidasPlicIrqIdQspiComplete interrupt id when a transaction is initiated.
Loopback Test	This test check if a device can transmit random data to another device and then again receive the same data from the device where the data was sent intially to ensure loopback mechanism.
Illegal access Test	This test checks if a certain operation is performed in a given boundary or not. For example an error is detected in an event of accessing a register outside the valid address domain.
Random reset Test	This test checks if can operate properly if a sudden reset command is applied to it.

Table 3.3: Test cases for functional verification

Chapter 4

Experimental Results

In this chapter, we present the results generated by synthesizing the design for VCU118 [10] and Arty7 [9] FPGA kits. Resource utilization and performance reports for individual modules are compared and described in detail. This chapter also presents all the test programs that were developed for functional verification of the design. Finally, it describes the power consumption reports in detail and compares the values obtained for both the FPGA kits.

4.1 Test Setup and Functional Verification

The design provided by the industry partner was functionally verified by running all the test cases described in Table 4.4. The traditional RTL-level testbench methodology is time consuming and involves complex architecture. Hence, the modern technique of testing a design was used by developing different test programs based on the software language C. These test programs were easy to design and saved around 40 percent of time compared to using the traditional method of testing a design using testbench. These test programs provided a "black box" interface, thereby focusing on a particular module of the design under test. This enabled us to test a module immediately after designing it, increasing time efficiency. The RTL-level test bench,

on the other hand, is dependent on the status of other modules associated with the design. The tests in Table 4.4 were written with the help of OpenTitan documentation. [16] [14] [15].

The tests highlighted in Table 4.4 were executed by the RISC-V microcontroller. These tests had flags indicating "Test Passed" or "Test Failed" after completion of each test. Verifying the design using software programs allowed us to verify the correctness of the test with the help of simulation. Fig 4.17 describes simulation result of a random test. All these tests were cross-verified with the help of simulation by pointing to any register value and comparing it with the expected value. These software programs can be tested on actual FPGA as well. There are two ways to run the program. First is *.hex* and the second is *.img*. *.hex* creates a hex file as the output of the operation while the *.img* creates an image of the test and runs the program through bootloader by loading the application. Aarish team have developed a software

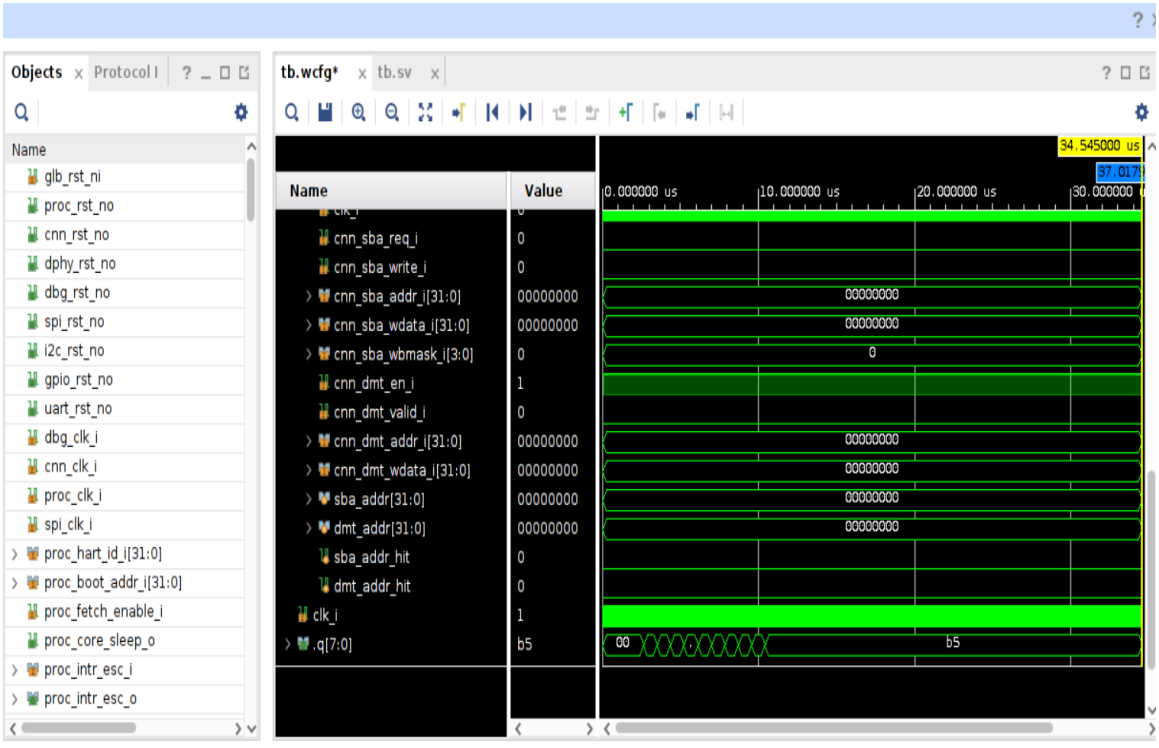

Figure 4.17: Simulation Waveform

application based on C called as "UART-APK" with an aim to interact with FPGA

Test	**Description**
Loop back test	This test checks if few bytes transmitted and received are the same
CSR (control and status registers) test	This test is to program CSR through write operation and check the FIFOs
Random reset test	This test checks if the module performs certain operation correctly after performing reset
FIFO test	This test perform checks on the length of FIFOs by sending data within and over the constrained limit
Delay test	This test checks if the operation is performed correctly after adding delays to it
Interrupt test	This test checks if an interrupt is triggered by writing random values to interrupt enable CSRs and verifying through interrupt state CSRs
Out of bound test	This test verifies if the design triggers an error/warning at times when a address outside the range is read or written to
Illegal Access test	This test alerts if a read only register is written any value or a write only register is read
Stress test	This test checks if the CSR operations are performed accurately when performed one after the other in continuous sequence with random reset

Table 4.4: Functional Verification Tests [14] [15] [16]

when running the design on it. This application can point to any register value, status of interrupt at a given time and cross check it's present value with the expected one. Hence, this application provided us the ability to check the functionality of any module present in the design at any random clock cycle. The working of this application is by sending data from host to FPGA in the form of packets. This packet comprised of about six components, each signifying specific importance with respect to design. By sending such packets it was possible to capture any register value, timer status, interrupt status, bus status etc. Example of one such packet is shown in the Fig 4.18

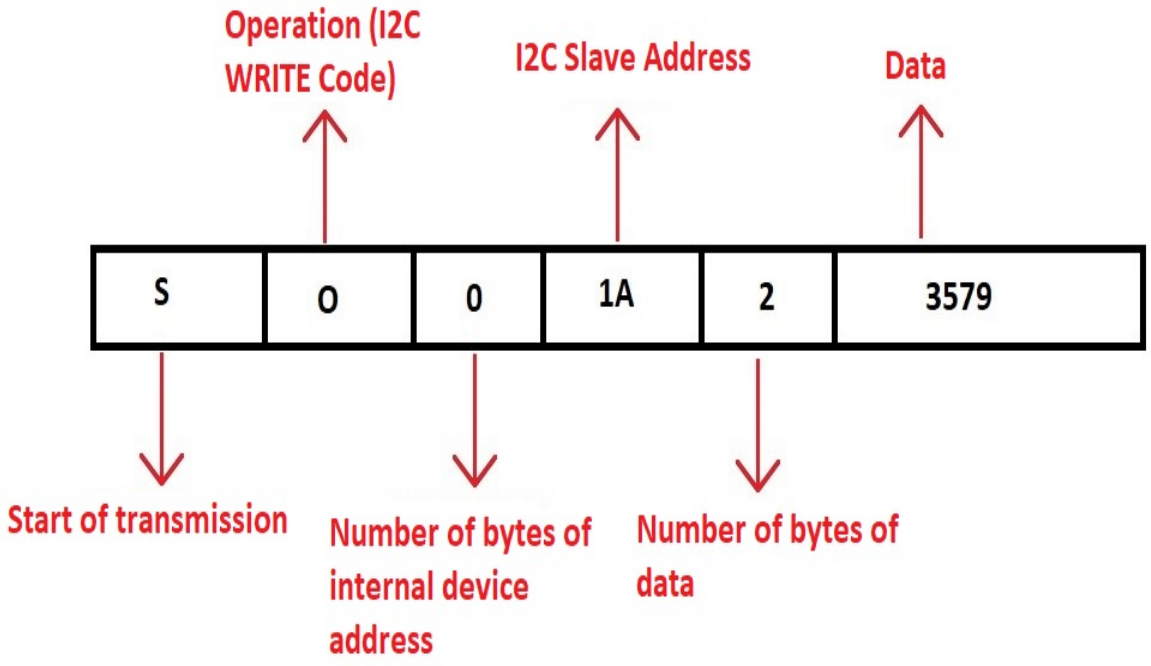

Figure 4.18: Packet

The operation byte in the packet decides which module to trigger for instance Fig 4.18 highlights I2C write operation, there are few other modes as well which are presented in Table 4.5

Instruction Code	Operation
R	Simple Read Code
W	Simple Write Code
Q	QSPI Write Code
I	I2C Read Code
O	I2C Write Code

Table 4.5: Packet operation codes

4.2 Jenkins Setup

Jenkins is a self-contained, open-source automation server which was used to automate tasks related to building, testing and delivering or deploying a job or software. As an extensible automation server, Jenkins can be used as a Simple Continuous Integration server or turned into the Continuous Delivery hub for any project [18]. Jenkins is a self-contained Java-based program, ready to run out-of-box, with packages for Windows, Mac OS X and other Unix-like operating systems. Jenkins can be easily set up and configured via its web interface, which includes on-the-fly error checks and built-in help. With hundreds of plugins in the Update Center, Jenkins integrates with practically every tool in the continuous integration and continuous delivery toolchain. Jenkins can be extended via its plugin architecture, providing nearly infinite possibilities for what Jenkins can do. Jenkins can easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster. In this thesis Jenkins was integrated to build and test the design at regular intervals. Using Jenkins, we designed a pipeline that helped us to build RTL and test it with the help of an automated script using Shell scripting and TCL scripting. These automated scripts instructed Jenkins to run all the tests that are mentioned above and to evaluate functionality of the design. After successful completion of functioanlity verification the script would instruct Jenkins for building *.bit*

file for the design with the help of TCL script. In case of any errors that may occur in the pipeline we designed a automated script that could extract the error messages and alert the development team which helped to debug the issue with reduction in time. Setup for the Jenkins was created in such a way that whenever there was a change in the RTL or if any new test case was added to the flow Jenkins pipeline was triggered, and completed the operation as mentioned above. Traditionally to perform all the steps illustrated above it would take 4-5 hours to complete, but with the inclusion of Jenkins this time went down to 2.5 hours thereby reducing time needed by about 50 percent.

4.3 FPGA Resource Utilisation

As highlighted in section 4.2 a *.bit file* was generated after completing placement, routing, synthesis and implementation of the design which provided us the reports on resource utilization. This process is performed for both the FPGA boards and the results were displayed as follows. In this section setup for implementing the functional verification and generating bit stream is illustrated in detail. Resources used by individual modules are compared and presented with the help of pie charts. Finally, detailed description is provided for power utilization on both the boards and difference between static and dynamic power consumed by the boards is explained.

Table 4.6 highlights total resources available and used in VCU118 FPGA Kit, while Table 4.7 shows similar data for Arty A7 kit. Among all the resources utilised, the important ones are presented in detail. Fig 4.19, Fig 4.20 and Fig 4.21 highlights CLB, LUT as logic, F7 Muxes, F8 Muxes, CLB LUTs and CLB Registers utilised in different modules of the design like RISC-V, QSPI, UART, GPIO, I2C e.t.c. implemented on VCU118 design. Similarly Fig 4.22, Fig 4.23 and Fig 4.24 represents similar results obtained for Arty A7 board. These charts clearly specify that most of the resources used are within the elements like CLB LUTs, CLB Registers, F7 Muxes,

Resource Name	**Resources Utilized**	**Total Resources Available**
CLB LUTs	17293 (1.46 percent)	1182240
CLB Registers	9988 (0.42 percent)	2364480
CARRY8	212 (0.14 percent)	147780
F7 Muxes	681 (0.12 percent)	591120
F8 Muxes	184 (0.06 percent)	295560
CLB	3511 (2.4 percent)	147780
LUT as logic	16965 (1.43 percent)	1182240
LUT as Memory	328 (0.06 percent)	591840
Block RAM Tile	20 (0.92 percent)	2160
DSPs	1 (0.01 percent)	6840
Bonded IOB	31 (3.7 percent)	832
HPIOBM	13 (3.4 percent)	384
HPIOBS	15 (4 percent)	384
HPIOBSNGL	3 (4.7 percent)	64
HPIOBDIFFINBUF	1 (0.14 percent)	720
BITSLICERXTX	10 (0.6 percent)	1560
GLOBAL CLOCK BUFFERs	6 (0.33 percent)	1800
MMCM	1 (3.33 percent)	30

Table 4.6: VCU118 Resource Utilization

F8 Muxes, LUT as logic and CLBs. CLB stands for configuration logic block and it is the basic repeating logic block on an FPGA. There are hundreds of CLBs available within a FPGA connected via routing resources, the purpose of these CLBs is to implement combinational and sequential logic. VCU118 board had total of 147780 CLBs out of which 3511 were utilised by the design. Three essential CLB components are Flip-Flops, Look-up-Tables and Multiplexers. Flip flop is a single bit storage element available in FPGA. LUT which is called as Look-up Table is considered as the heart of FPGA. It contains all the logically possible outputs of the design. LUT is a mux-based architecture in which the inputs are the possible outputs based on the correct selection on the multiplexer select lines. Multiplexers are basically a circuit which does the transition between different outputs based on select line inputs. If the LUTs uses Flip-Flops in the logic, then it is described "LUT as Memory", otherwise it is described "LUT as Logic". The next important resources utilized were the F7 Muxes and F8 Muxes. VCU118 board has about total of 591120 F7 Muxes while Arty A7 has about 31700 out of which only 680 were utilised in the design. Similarly, VCU118 board had about total of 295560 F8 Muxes while the Arty A7 board had about 15850 F8 Muxes out of which only 180 were utilised. The purpose of F7 Mux is to combine the output of two 6-input LUTs to create a 7-input function, inputs to F7 Muxes are the output of LUTs immediately adjacent to them. Purpose of F8 Muxes is to combine the output of two F7 Muxes to create a 8-input function.

Different report and compilation result files are generated after hardware compilation including the synthesis report, RTL description of the design, placement and routing, programming bit file and etc. The testing setup also provided us with the diagram indicating implementation of the design on FPGA. This chip view diagram allowed us to analyse resource utilization of design in more detail and visualize the implementation of the FPGA design. The chip view displays actual placement of the modules used in the design on the FPGA board, thereby enabling us to access

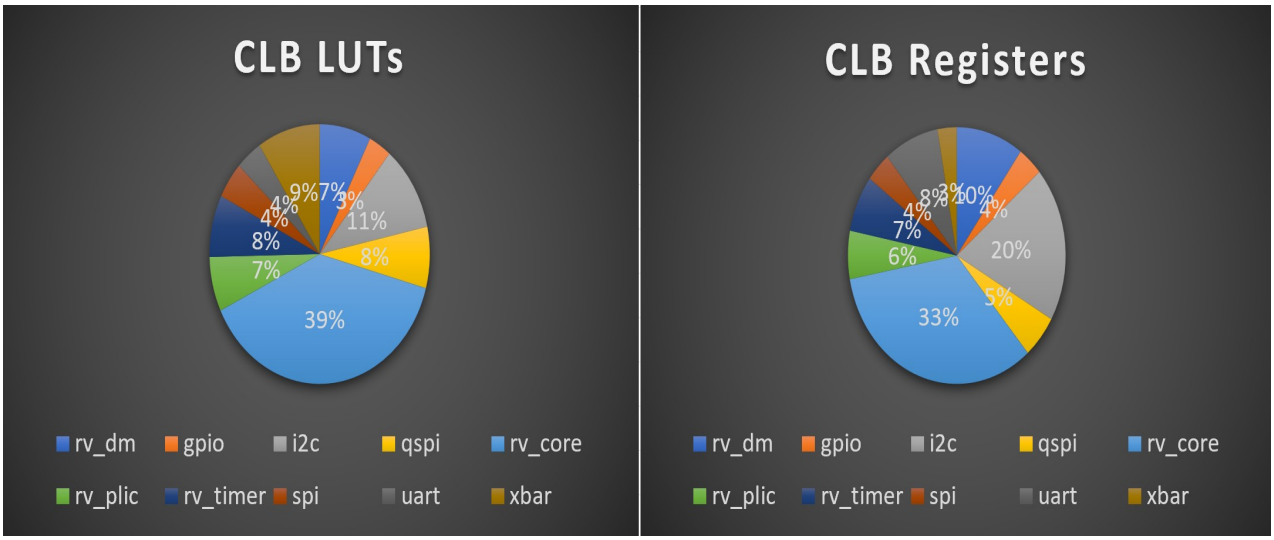

Figure 4.19: VCU118 Resource Utilization on individual modules

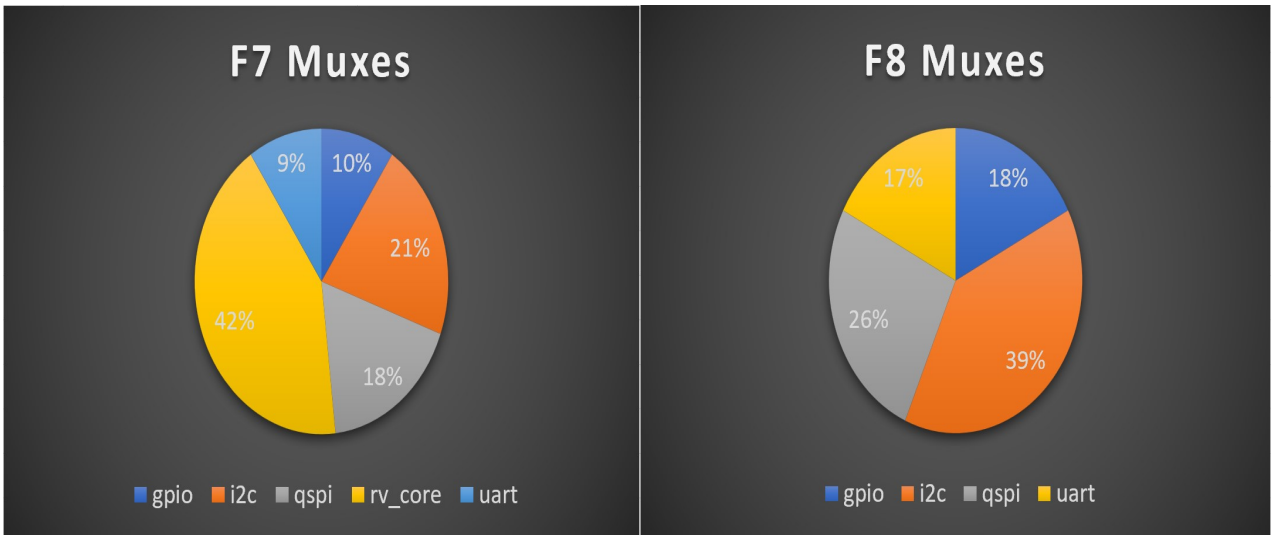

Figure 4.20: VCU118 Resource Utilization on individual modules

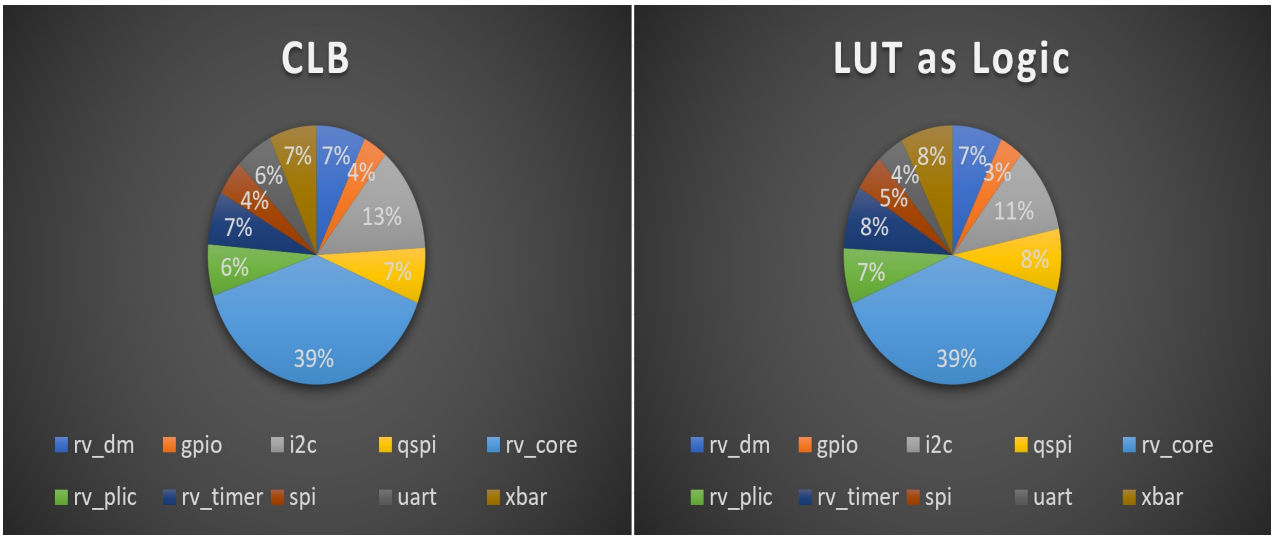

Figure 4.21: VCU118 Resource Utilization on individual modules

Resource Name	**Resources Utilized**	**Total Resources Available**
Slice LUTs	18341 (29 percent)	63400
Slice Registers	10536 (8 percent)	126800
F7 Muxes	689 (2 percent)	31700
F8 Muxes	184 (1 percent)	15850
Slice	6360 (40 percent)	15850
LUT as Logic	17989 (28 percent)	63400
LUT as Memory	352 (2 percent)	19000
Block Ram Tile	20 (15 percent)	135
DSPs	1 (0.4 percent)	240
Bonded IOB	123 (59 percent)	210
ILOGIC	8 (4 percent)	210
OLOGIC	5 (2 percent)	210
BUFGCTRL	7 (22 percent)	32
MMCME2ADV	1 (17 percent)	6

Table 4.7: Arty A7 Resource Utilization

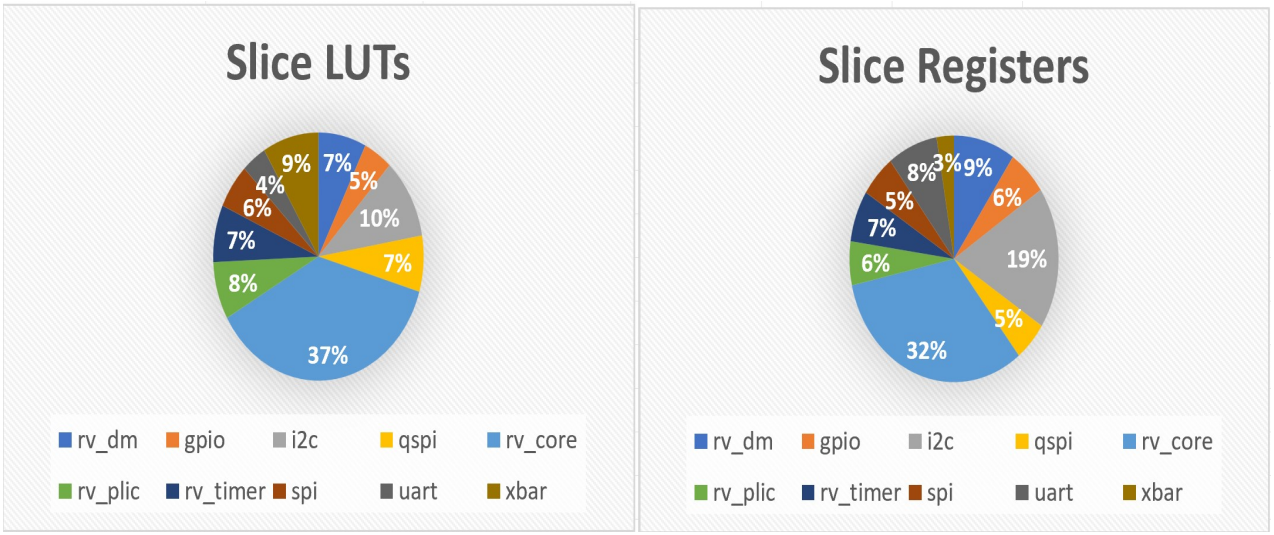

Figure 4.22: Arty A7 Resource Utilization on individual modules

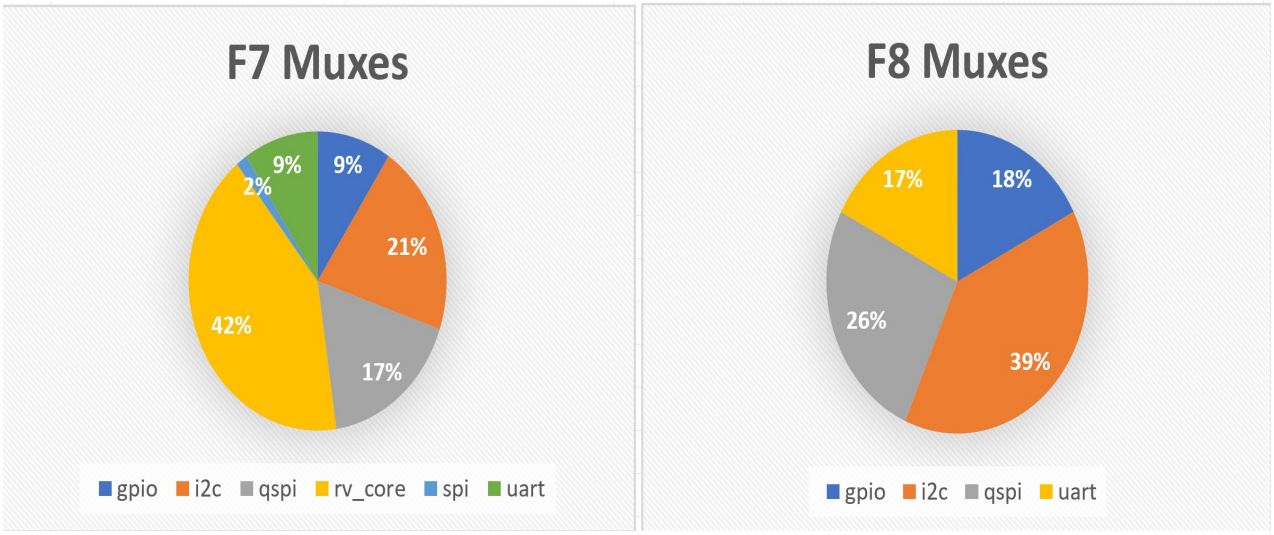

Figure 4.23: Arty A7 Resource Utilization on individual modules

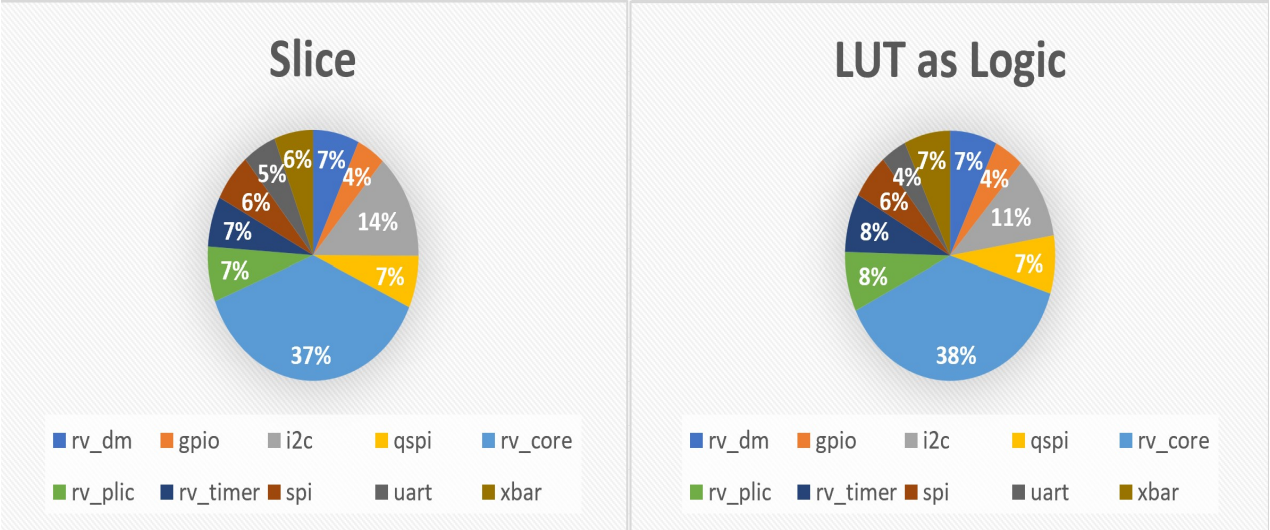

Figure 4.24: Arty A7 Resource Utilization on individual modules

detailed information of the implementation at the lowest level component in the design. Fig 4.26 shows the chip view of the design implemented on VCU118 board. By zooming-in the chip view, the device architecture and configuration of FPGA can be analysed, including Adaptive Logic Modules (ALM), Phase-Lock Loops (PLL), DSP blocks, RAM blocks and Memory Cells. The area displayed in blue represents the components available in design along with its routing and placement. While the remaining black section in the rectangular box is the part of FPGA that was not utilized. All these components are routed specifically to form the desired logic.

4.4 Power Analysis

Designing for low power in today's high-speed FPGA designs is more important than ever. Knowing the final design's power usage early in the design process is necessary for making power supply design and power budgeting decisions. Vivado power analysis tool was used in this thesis to perform highly accurate estimates of power usage. The two main features in understanding power are Static power and Dynamic power. Dynamic Power depends on capacitance, voltage and switching activity of the resources utilized in FPGA i.e., models the power consumed during charging and

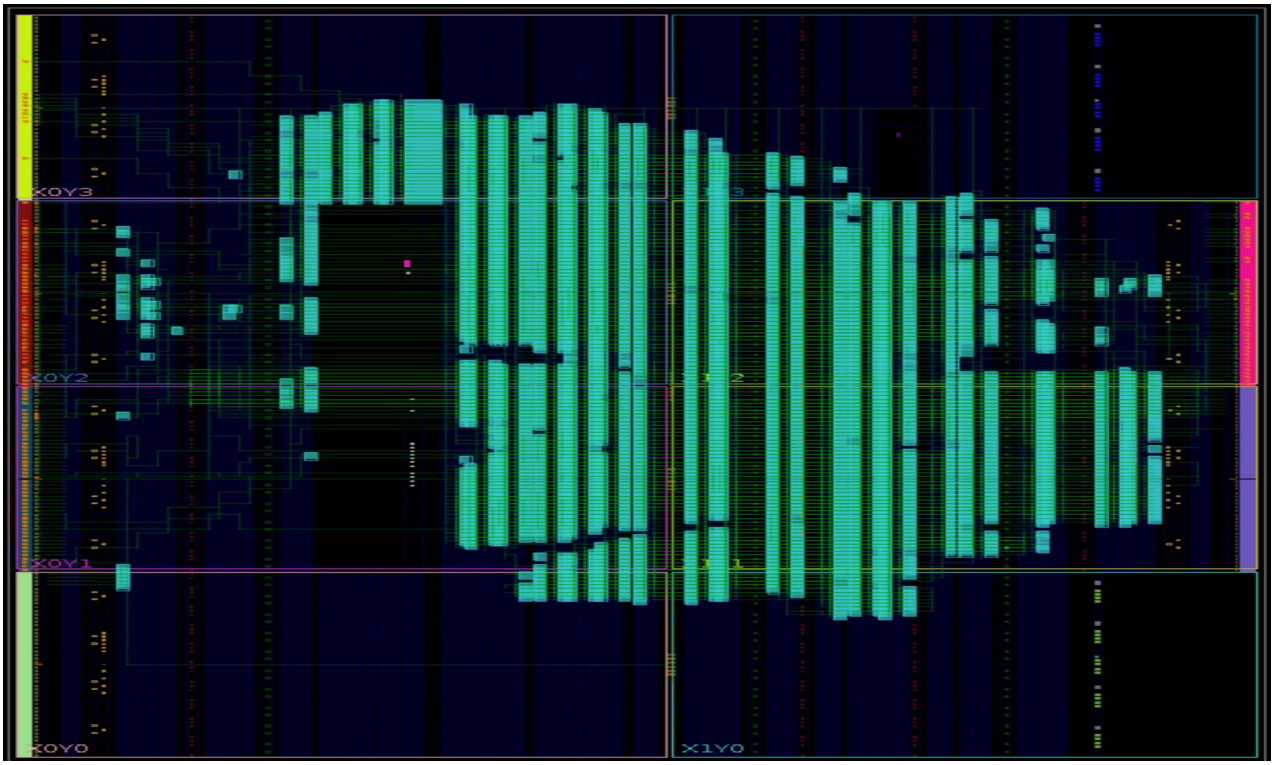

Figure 4.25: Arty A7 Chip View

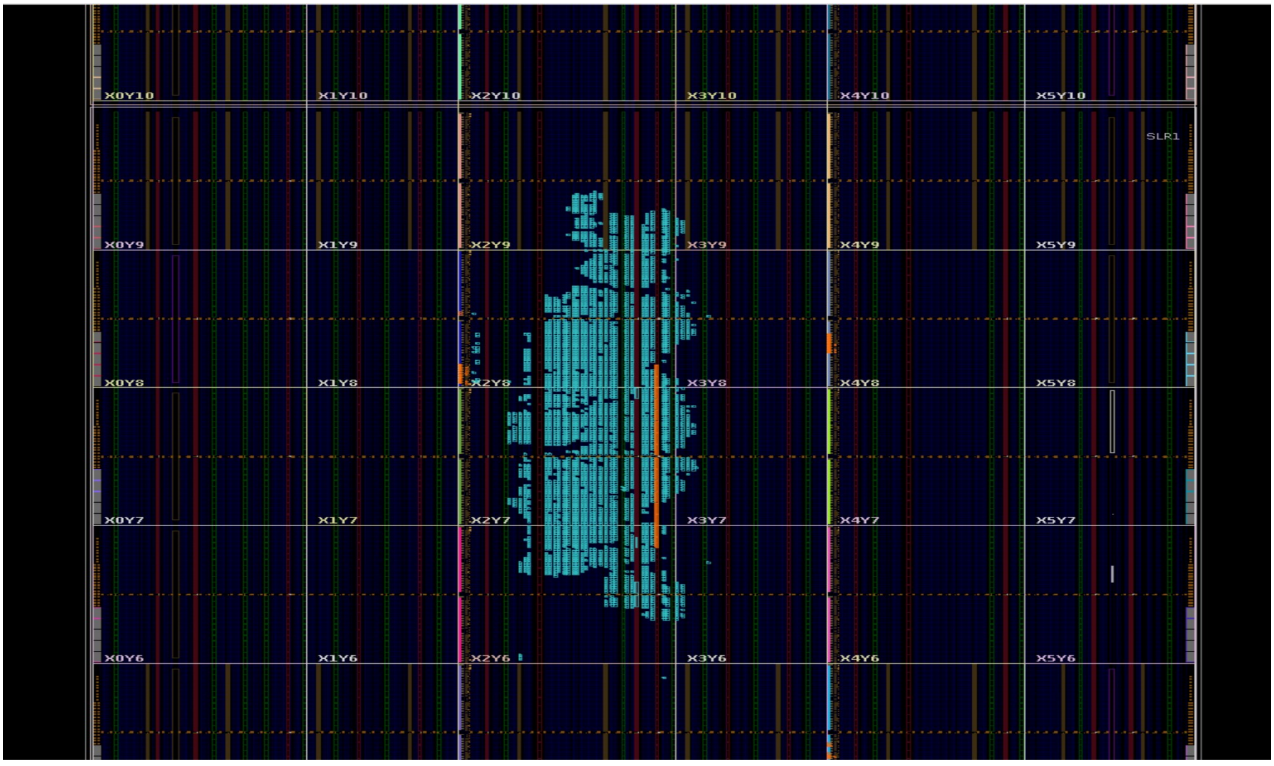

Figure 4.26: VCU118 Chip View

discharging the capacitive load associated with the resource. Hence more the utilization of the resources more would be the dynamic power consumption. Dynamic Power consumption also occurs as a result of short circuit created when an output transitions from logic 0 to 1 or vice versa. Hence to summarise this if switching is more in the design it is expected to consume more dynamic power. Static Power is consumed when there is no circuit activity for example power consumed by D Flip-flop when neither clock nor D input are active [19], [20], [21]. Figures 4.27 and 4.28 refers to power analysis results obtained for VCU118 and Arty A7 boards respectively. VCU118 board provided about 2.585 W of total power consumption, out of which Dynamic power was 0.113 W and static power was 2.471 W. Here the Dynamic Power is less as compared to static power, as described above the Dynamic power is related to resources used in comparison to the total resources available and in VCU118 FPGA board the resources are less utilised as compared to total resources available. Static power of such large boards is expected to consume about 10 W of power, hence consuming just 2.471W shows that the design is power efficient. Ideally Dynamic power is expected to be in the range of 50-60 percent of the total power. Fig 4.28 describes this behaviour as the design on Arty a7 board consumes 0.121 W of Dynamic power and 0.098 W of Static Power. These figures also represent the power consumed by individual components of an FPGA. Clocks, Signals, Logic, BRAM, DSP, MMCM and I/O are the ones highlighted. DSP stands for digital signal processing and operate on instructions and code and can include branching operations and high-level decision trees. Using DSPs it is possible not only to perform individual mathematical operations but also to accommodate entire signal processing algorithms. MMCM stands for Mixed-Mode Clock Manager. The MMCM primitive is used to generate multiple clocks with defined phase and frequency relationships to a given input clock. BRAM stands for block random access memory; they are used for storing large amount of data inside the FPGA.

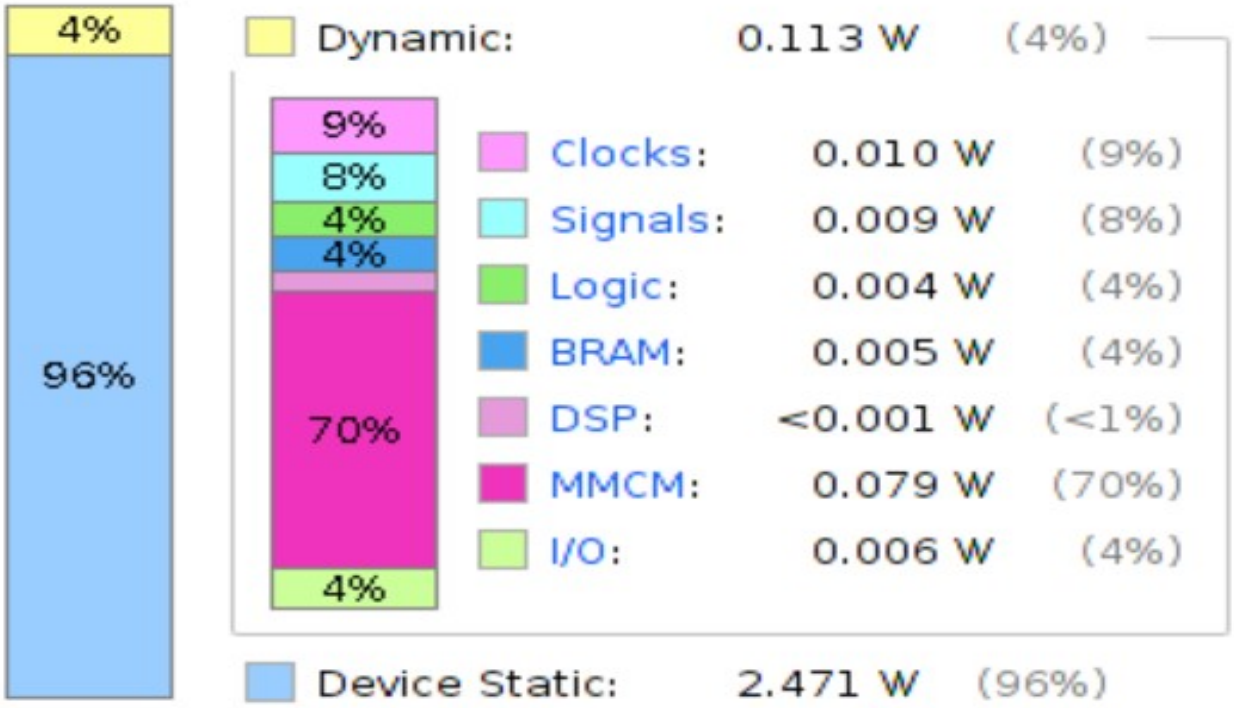

Figure 4.27: Power Analysis of VCU118

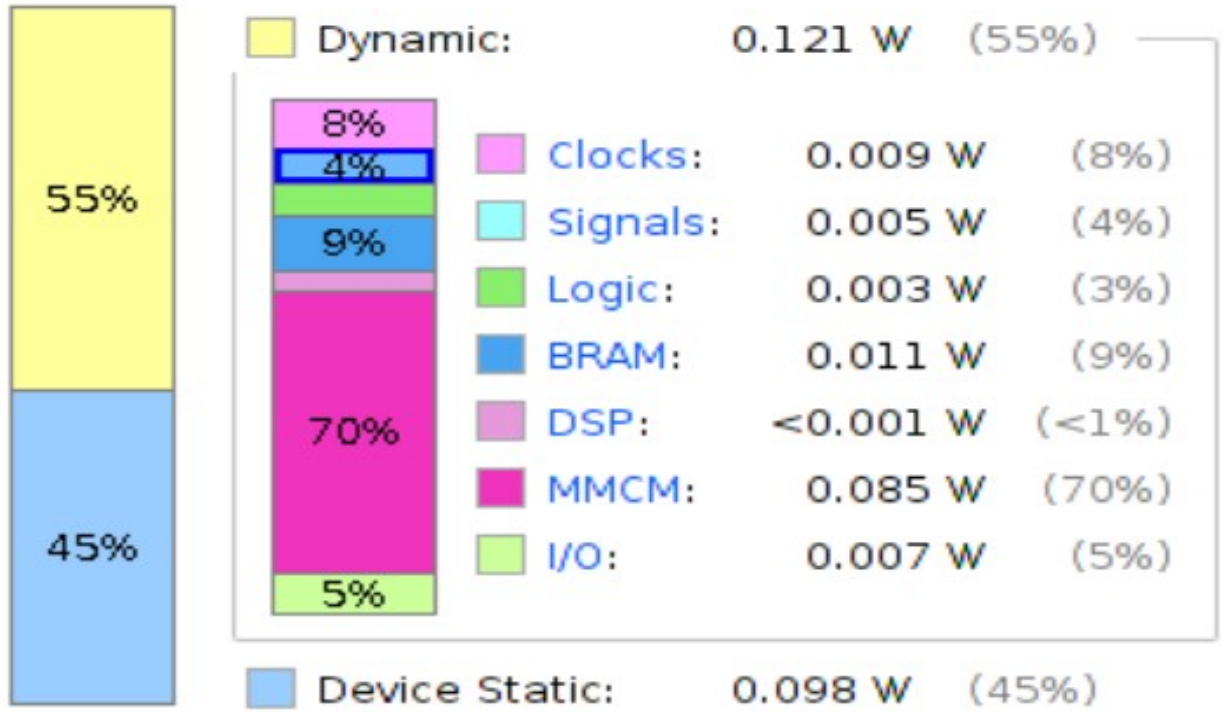

Figure 4.28: Power Analysis of Arty A7

Chapter 5

Conclusion and Future work

This thesis addresses prototyping and functional verification of a low power AI processor on FPGA. The FPGA kits targeted were VCU118 and Arty A7. The power consumption of the AI processor for these two FPGA kits was also evaluated and compared. This thesis also discussed different methodologies for hardware design such as register transfer language (RTL) and High-Level Synthesis (HLS). For efficient processor design, RTL based methodlogy is still preferred as it provides better quality of results, i.e. area, speed and power consumption.

Performing functional verification of a complex processor design can especially be difficult and time consuming if it is carried out using traditional method of using RTL-Level testbench. It requires accurate functioning of all the modules in the design. It relies on the status of other components in the design which may not be related to it. Software test programs developed in this thesis enabled functional verification of the design without relying on the status of other modules thereby allowing a module to be tested immediately after its development. This is because these software test programs operate based on their driver (software program in which all the signals and functions are defined). The software test programs are developed using C language which made it faster to design and test thereby increasing time efficiency. These software programs displayed simulation results which helped to understand the

design under test in much more detail.

To prototype a design on FPGA using RTL method of hardware design requires completion of few stages at frequent intervals. This slows the process and involves repetitive work. This thesis integrated Jenkins and automated the process of generating programming bit file and performing functional verification of the design for every modification in the design. Integrating Jenkins supported fetching errors in the pipeline and that resulted in increased time efficiency. The experimental results shows reduction in time needed by about 40 percent. Shell scripting and TCL scripting were used to run the Jenkins pipeline which provided with an ability to add triggers and automation for designing efficient pipeline. Using Jenkins, the design was synthesized for VCU118 and Arty 7 FPGA kits.

High power dissipation is one of the major disadvantages of FPGAs. A main part of the power consumed is caused by glitches. The design provided by industry partner did not use external memory. This resulted in achieving low power values of about 0.22W and 2.5W for Arty7 and VCU118 FPGA kits respectively. The experimental power results proved that the design provided by our industry partner was power efficient. Preventing use of external memory also resulted in faster arithmetic logic unit (ALU) operations as the time to access data got reduced.

This thesis can be further extended by implementing the design on FPGA and performing functional verification of the design with the help of actual FPGA. Additional test cases can be added in the functionality testing. The test programs defined in the thesis are developed on C, moving these programs to software language like Python would help in saving more time. The scope of prototyping such processor on FPGA could be using it in small scale applications of Image recognition, Lip movement to speech conversion and Object detection.

Bibliography

[1] "Internal Structure of an FPGA," https://digitalsystemdesign.in/wp-content/uploads/2019/05/fpga.png?is-pending-load=1, Last Accessed Sep 7,2022.

[2] "Xilinx Documentation," https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html, Last Accessed Sep 7,2022.

[3] By Doug Amos, René Richter, Synopsys and Austin Lesea, Xilinx,"What can FPGA-based prototyping do for you?," https://www.techdesignforums.com/practice/technique/what-can-fpga-based-prototyping-do-for-you/, Last Accessed Sep 8,2022. 2012.

[4] Moore, Gordon E.,"Cramming more components onto integrated circuits, Reprinted from Electronics," *IEEE Solid-State Circuits Society Newsletter*, volume 38, number 8, pp.114 ff, 2006.

[5] Sze, Vivienne and Chen, Yu-Hsin and Yang, Tien-Ju and Emer, Joel S.,"Efficient Processing of Deep Neural Networks: A Tutorial and Survey" *Proceedings of the IEEE*, vol. 105, no. 12, p. 2295–2329, 2017.

[6] "A Simple Logic Block of FPGA, Lec Notes - Dr. Khalid", 2022.

[7] Joonas J rviluoma,"Rapid prototyping from algorithm to FPGA prototype," 2015

[8] Pavel Sinha,"Configurable Processor for implementing convolutional neural networks," https://patentscope.wipo.int/search/en/detail.jsf?docId=IN355932989 2022

[9] Xilinx, "Arty Board User Guide," https://reference.digilentinc.com/arty:refmanual 2022 Last Accessed Sep 22, 2022.

[10] Xilinx, "Virtex Board User Guide," https://www.xilinx.com/support/documents/boards_and_kits/vcu118/ug1224-vcu118-eval-bd.pdf 2022 Last Accessed Nov 14, 2022.

[11] Jason H. Anderson and Tomasz S. Czajkowski, "Computer-Aided Design for FPGAs: Overview and Recent Research Trends," 2014

[12] Ron Landry, AMI Semiconductor, Inc., "FPGA Prototyping as a Verification Methodology,"

[13] Premduth Vidyanandan, Xilinx Inc., "Verification Techniques For FPGA Designs," 2008 https://www.electronicdesign.com/technologies/fpgas/article/21787381/verification-techniques-for-fpga-designs Last Accessed Sep 24, 2022.

[14] "OpenTitan documentation, lowRISC contributors," https://docs.opentitan.org/hw/ip/uart/doc/dv/ Last Accessed Sep 24, 2022.

[15] "OpenTitan documentation, lowRISC contributors," https://docs.opentitan.org/hw/ip/spi_device/doc/ Last Accessed Sep 24, 2022.

[16] "OpenTitan documentation, lowRISC contributors," https://docs.opentitan.org/hw/ip/i2c/doc/ Last Accessed Sep 24, 2022.

[17] "System Verilog reference verification methodlogy," https://www.eetimes.com/systemverilog-reference-verification-methodology-rtl/ Last Accessed Nov 25, 2022.

[18] "Jenkins," https://www.jenkins.io/ Last Accessed Nov 25, 2022.

[19] "FPGA Dynamic Power," https://1library.net/article/fpga-dynamic-power-fpga-power-consumption.q78x4dvz Last Accessed Nov 25, 2022.

[20] "Xilinx," https://docs.xilinx.com/r/en-US/ug949-vivado-design-methodology/Dynamic-Power Last Accessed Nov 25, 2022.

[21] "Xilinx," https://docs.xilinx.com/r/en-US/ug949-vivado-design-methodology/Static-Power Last Accessed Nov 25, 2022.

[22] "Xilinx," https://www.xilinx.com/developer/products/vivado.html Last Accessed Nov 25, 2022.

[23] Bellizia, Davide and Hoffmann, Clément and Kamel, Dina and Liu, Hanlin and Meaux, Pierrick and Standaert, François-Xavier and Yu, Yu,"Learning Parity with Physical Noise: Imperfections, Reductions and FPGA Prototype" *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 10.46586/tches.v2021.i3.390-417, 2021.

[24] Gschwind, M. and Salapura, V. and Maurer, D.,"FPGA prototyping of a RISC processor core for embedded applications" *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 2, number 2, 2001.

[25] Vumanthala, Sagar and Kalagadda, Bikshalu,"Modified Nonlocal Means Filtering for Speech Enhancement and Its FPGA Prototype" *Circuits, Systems, and Signal Processing*, volume 40, 10.1007/s00034-021-01750-5, 2021.

[26] Dueck, Stuart,"A power evaluation framework for FPGA applications and CAD experimentation" `https://open.library.ubc.ca/collections/ubctheses/24/items/1.0073893` *Electronic Theses and Dissertations (ETDs) 2008+*, 2013.

[27] Albert Reuther and Peter Michaleas and Michael Jones and Vijay Gadepally and Siddharth Samsi and Jeremy Kepner,"Survey of Machine Learning Accelerators" `https://arxiv.org/abs/2009.00993` *CoRR*, abs/2009.00993, 2022.

[28] Foster, Harry D.,"2018 FPGA Functional Verification Trends" *Circuits, Systems, and Signal Processing*, 10.1109/MTV.2018.00018 2018.

[29] Li Du and Yuan Du,"Hardware Accelerator Design for Machine Learning" = https://doi.org/10.5772/intechopen.72845 *IntechOpen*, 10.5772/intechopen.72845, 2017.

[30] Luthra, Siddhant and Khalid, Mohammed A.S. and Moin Oninda, Mohammad Abdul,"FPGA-Based Evaluation and Implementation of an Automotive RADAR Signal Processing System using High-Level Synthesis" *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 10.1109/CCECE47787.2020.9255725 2020.

Vita Auctoris

NAME:	Vivek Liladhar Ladhe
PLACE OF BIRTH:	Dombivli, Maharashtra, India
YEAR OF BIRTH:	1997
EDUCATION:	Ramrao Adik Institute of Technology Nerul, Maharashtra, India 2015-2019, Bachelor of Engineering Electronics and Telecommunication Engineering University of Windsor, Windsor ON, Canada 2021-2022, Master of Applied Science Electrical and Computer Engineering