Media Engineering and Technology Faculty
German University in Cairo

DSP Runtime Emulator on FPGA

Bachelor Thesis

Author: Ahmad Hassan Sayed Ahmad Taha
Supervisors: Assoc. Prof. Hassan Soubra
Submission Date: 12 June, 2022

Media Engineering and Technology Faculty
German University in Cairo

DSP Runtime Emulator on FPGA

Bachelor Thesis

Author:	Ahmad Hassan Sayed Ahmad Taha
Supervisors:	Assoc. Prof. Hassan Soubra
Submission Date:	12 June, 2022

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowledgment has been made in the text to all other material used

Ahmad Hassan Sayed Ahmad Taha
12 June, 2022

Acknowledgments

I would like to start by thanking my family for always supporting me, as well as providing me with the opportunities that I would say are primary reasons for who I am today. I would also like to thank my friends whom I consider to be my second family, I am really grateful to all of you. I also wish to express my sincere gratitude to Dr. Hassan Soubra for his unconditional support. Thank you for the constant guidance and advice.

Abstract

A digital signal processor (DSP) is a high-performance, high-power microprocessor chip.A soft-core processor is one that is synthesized the programmable logic resources of a Field Programmable Gate Array (FPGA) and whose design and behaviour are completely described using a Hardware Description Language (HDL).VHDL is used to design and implement of an 8-bit fixed point Digital Signal Processor Core.To achieve high speed and throughput, the design makes use of pipelining concepts. The design's modules are compatible with a Cyclone IV (DE2-115) FPGA .The performance of these DSP operations is then evaluated by ensuring proper execution of functions and benchmarked against the TMS320DM64x+ DSP by Texas Instruments. Most of the time, the instructions in the emulator are as quick as, if not faster than, the TMS320DM64x+.

Contents

List of Figures

Chapter 1

Introduction

- A digital signal processor (DSP) is a high-performance, high-power microprocessor chip. DSP, as shown in figure[1.1], starts taking analog real-world signals such as voice, audio, video, temperature, pressure,etc. DSP is designed to execute mathematical operations like add, subtract, multiply, and divide very quickly.The real-world signal is then converted by converters such as an Analog-to-Digital converter into the digital format of 1's and 0's. The DSP takes over by acquiring and processing the digital data. It then sends the digital data back into the actual world. It accomplishes this in one of two ways:-
  - digital format
  - analog format
    - Using a Digital-to-Analog converter

  DSPs must have a very high throughput since they work in real time constraints.

Figure 1.1: DSP

- DSPs use the modified Harvard architecture to benefit from program memory and data memory separation. To increase throughput, It permits the contents of the instruction memory to be accessed as data memory.

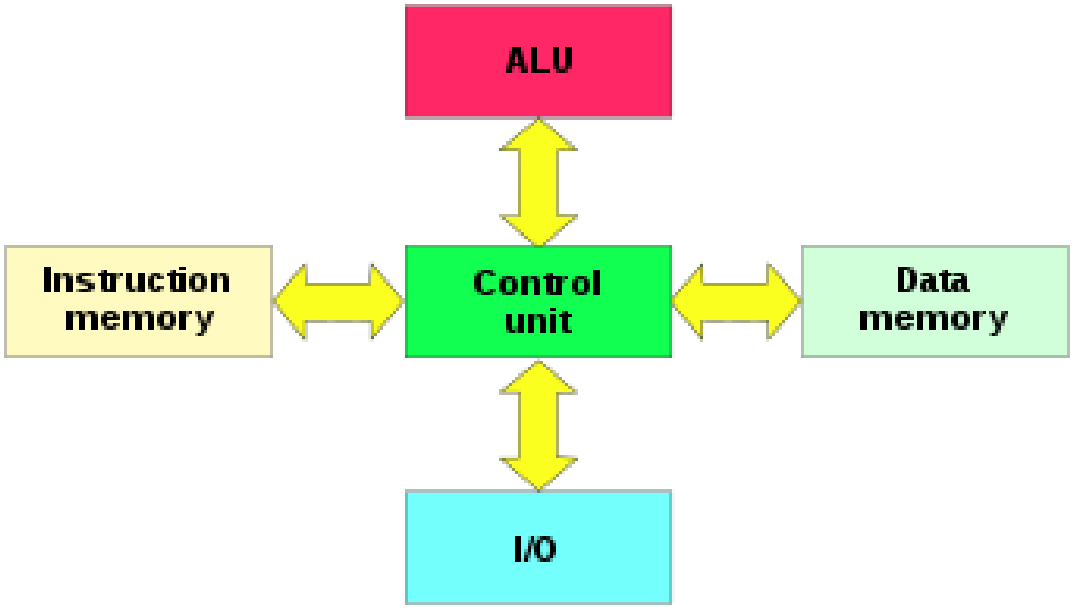

Figure 1.2: SHARC

- A soft-core processor is one that is synthesized the programmable logic resources of a Field Programmable Gate Array (FPGA) and whose design and behaviour are completely described using a Hardware Description Language (HDL). The FPGA is one of the finest technologies for implementing a DSP. FPGAs offer the ability to be reconfigurable inside a system, which may be extremely useful in applications that require several trial versions throughout development while still maintaining a realistic time to market.

- VHDL is used to implement DSP emulator. Each of the 10 modules has its own code. A top-module connects all of these sub-modules. The performance of various DSP operations is benchmarked against TMS320DM64x+ DSP by Texas Instruments.

Chapter 2

Literature review

2.1 Designing

2.1.1 Modeling and Implementation of DSP FPGA Solution

Recent advancements in high-level mathematical modelling tools' simulation capabilities have opened up fascinating new design flow possibilities. Support for bit true modelling at the system level allows designers to develop floating and fixed point models in the same environment and make scaling and rounding decisions early in the design process. FPGA companies have also increased their commercial IP offerings to include higher-level DSP functionality.Together, these technologies enable a novel data channel design procedure that incorporates system-level design iterations.DSP FPGA design used to necessitate the collaboration of a DSP engineer and a hardware engineer proficient with HDL or schematic-based design. We will demonstrate a novel way for obtaining an HDL netlist for a data channel directly from a system level tool in this workshop. The procedures involve creating an ideal mathematical model, investigating implementation consequences, creating a test bench, and creating a hardware netlist. The design is then implemented using traditional HDL design approaches. Examples of digital filter realisations, Discrete Wavelet Transforms (DWT), and digital communications applications are used to demonstrate these ideas.The goal of this study is to create a design environment and flow that is familiar and easy to use for DSP system architects while also targeting FPGA hardware directly. They aim to:

- Provide high-level DSP system modelling, including simple operators like delay,addition, and multiplication, as well as FIR(Finite Impulse Response) and IIR(Infinite Impulse Response) filters, FFTs(Fast Fourier Transform), and Discrete Wave

- Ensure that the same model from original theoretical design through coefficient quantification and data word length selection.

- Automatically generate an FPGA implementation of the datapath from the system model without the need for device-specific information.

- Support in the creation of control logic for FPGA implementation.

- Create test benches automatically for logic simulation on the final FPGA implementation.

[8]

2.1.2 The State-of-art FPGA Design

DSPs with programmable platforms have typically been used to implement digital signal processing tasks.However, as the demands of many computationally complex applications exceed the processing capabilities of DSPs, FPGAs have become increasingly popular. Furthermore, research is continuing to build and deploy high-level design tools that can assist reduce the time it takes to construct signal processing solutions utilising FPGAs. The process for creating the model for a valuable communication technology,namely MIMO, utilising Xilinx System Generator (XSG) and AccelDSP (for supporting XSG) is shown in this work.Its method demonstrates the particular advantages that such tools provide, such as reusability and quicker design durations due to fewer bugs. It also examines the relative merits of FPGAs over DSPs as DSP implementation platforms. It examines current FPGA usage in DSP applications such as MIMO and digital electronics. The merits and limitations of Xilinx System Generator as a design tool for FPGA are highlighted, since choosing an efficient design tool is a critical decision. Matlab and Simulink already have a big user base, and System Generator's improved performance has made it the favoured option of many DSP experts today.

- System Generator
  - An Xilinx DSP design tool that allows FPGA designers to utilize the Mathworks Simulink model-based design environment. It's a system-level modelling tool that uses a Xilinx-specific Blockset to record designs in the DSP-friendly Simulink modelling environment. It has a number of capabilities, including System Resource Estimation to fully use FPGA resources, Hardware Co-Simulation, and accelerated simulation using hardware in the loop co-simulation,all of which result in a significant boost in simulation speed. It also includes a DSP FPGA system integration platform that brings together the RTL, Simulink, MATLAB, and C/C++ components of a DSP system in a single simulation and implementation environment.

- MIMO
  - In the domains of its applications, multiple input multiple output antenna designs offers substantial performance and reliability advantages. MIMO, on the other hand, introduces new degrees of DSP complexity, performance, and changing standards, all of which need the use of a platform FPGA. The MIMO receiver has significantly higher processing needs than the SISO system. The usage of FPGA is required since the complexity and processing needs are well above the capability of ordinary DSP processors. Because MIMO is a relatively new technique, the amount of processing power required is expected to grow dramatically and To get faster data rates, it is necessary to use complicated algorithms. Furthermore, such calculations may necessitate the use of many CPUs.

[5]

2.1.3 Designing a Custom DSP Circuit Using VHDL

Designers need more powerful computer design methodologies throughout the design and development process as digital signal processing systems expand in size and complexity. Traditional computer-aided engineering tools, which are mostly used in the development phase, are inadequate for designing algorithms and architectures. Furthermore, they frequently lack tools for exchanging designs and incorporating new technology. VHDL solves these difficulties by providing a wide variety of modelling options for DSP designers to examine algorithms, architecture, and technologies.The IEEE standard language for describing electrical circuits is VHDL (Very High-speed Description Language). The majority of ASIC (application specific integrated circuit) designers use VHDL, and most major CAD suppliers have included VHDL tools in their product lines.' Simulators, debuggers, graphical interfaces,and synthesisers are among these tools.The use of VHDL in the design and construction of a custom DSP complex-to-magnitude circuit is described in this paper. Using an approximation approach, this circuit calculates the magnitude of a complex number. A bit-serial architecture with a set of basic operators for DSP design is the goal implementation. Using the real number type, It studies the method and architecture at the word level and apply the conclusions to the bit level. Denyer and Renshaw's case study on the design of a 16-point FFT (fast Fourier transform) computer provided us with this example. [3]

2.2 Modification

2.2.1 HUB-Floating-Point for improving FPGA implementations of DSP Applications

New digital signal processing applications' growing complexity necessitates the usage of floating-point values in their hardware implementations.A HUB-FP number is an FP number with an Implicit Least Significant Bit (ILSB) of one in its mantissa (or significand). It has the same amount of explicit bits and precision as a conventional format, but the same bit-vector represents a value biassed half Unit-in-the-LastPlace.The advantages of employing HUB formats to implement various floating-point applications on FPGAs are investigated. These new floating-point formats allow the rounding logic on floating-point arithmetic units to be effectively eliminated. To begin, it shows that on DSP application implementations, HUB and standard formats deliver equal SNR(signal to noise ratio). The paper then goes into depth on the benefits of utilising HUB numbers to create floating-point adders and multipliers on FPGAs.In majority of the scenarios investigated, the HUB strategy decreases resource consumption and boosts the speed of these FP units while maintaining statistically equal accuracy.However, HUB multipliers take considerably more resources than the standard technique for some particular sizes. [2]

2.2.2 Design and Implementation of an Embedded FPGA Floating Point DSP Block

It discusses the architecture and implementation of an FPGA DSP Block that can efficiently handle both fixed and single precision (SP) floating point (FP) arithmetic, from both the perspective of target applications and circuit design. DSP blocks that enable simple multiply-add-based fixed-point arithmetic cores are embedded in most modern FPGAs. Due to the routing and soft logic limits on the devices, larger systems cannot be successfully implemented, resulting in severe space, performance, and power consumption penalties compared to ASIC implementations. It examines previously suggested embedded FP implementations and demonstrates why they are unsuitable for use in a production FPGA.It compares these to the solution in which the SP FP multiplier is overlaid on fixed point constructs, the SP FP Adder/Subtracter is integrated as a separate unit, and the multiplier and adder can be combined in a way that is both arithmetically useful and efficient in terms of FPGA routing density and congestion. A innovative approach of smoothly merging any number of DSP Blocks in a low latency framework will also be demonstrated. It will be demonstrated that with this innovative technique, a low-cost, low-power, and high-density FP platform can be implemented on current production 20nm FPGAs. It also describes a future DSP block upgrade that will enable subnormal numbers.

[4]

2.3 Implementation

2.3.1 Implementation of Real Time Image Processing System with FPGA and DSP

Real-time image processing is in great demand for a variety of applications such as security systems,remote sensing, industrial processes, and multimedia, all of which require high performance.This study proposes image processing systems based on that need, employing a heterogeneous platform named TMS320DM642. An FPGA chip and a DSP processor are included in the platform. Image sampling and display are handled by the FPGA chip, while crucial image processing is handled by the DSP processor. The suggested hardware design and its operating principle are discussed first, followed by several major concerns relating to the external memory interface.Finally, an image edge detection technique is described to evaluate the operation of the suggested system.Edge detection is the process of finding intensity transitions in an image. Many image processing systems, such as object recognition, object tracking, segmentation, and so on, have employed edge detection. As a result, edge detection is an essential subprocess in many image processing systems. Many edge detection approaches have been proposed, including Sobel, Prewitt, Roberts, and Canny.Sobel edge detection is used to show the functionality of the proposed real-time image processing employing DSP and FPGA. The horizontal mask and vertical mask of the Sobel edge detector are used to locate intensity transitions in the horizontal and vertical directions. Sobel masks come in 3X3 and 5X5 sizes. The created system can capture liveframes from a camera, show images on a VGA monitor or NTSC/PAL TV, and perform image processing activities such as colour model conversion, pixel-based operation, and so on. It is also demonstrated that the created system can fulfil the real-time performance requirements. [7]

2.3.2 Design and Implementation Of a SHARC Digital Signal Processor Core In Verilog HDL

They present the design and implementation of an 8-bit fixed point DSP core in verilog HDL. In order to achieve high performance and throughput, the design takes advantage of pipelining and parallelism principles. The design's modules are compatible with an XILINX,XC4O1OXL FPGA with 13OK gates and a clock frequency of 32.31 MHz. This DSP design achieves a balance between a high-performance processor core and high-performance buses, Progrom Memory (PM) and Data Memory (DM) . Every instruction in the core can be completed in a single cycle. To keep the execution rate up, the buses and instruction cache offer unrestricted data flow to the core. The architecture presented is very modular, which makes it appropriate for VLSl implementation.It has a high throughput rate of one sample per cycle and a high data input rate of one sample per cycle. Control circuits are built as counter-based logic with a pause function to stop all activities in the functions of a smooth data flow. The design is verified using Verilog simulation based on register transfer level descriptions.

[6]

2.3.3 Hardware Implementation of DSP Filter on FPGAs

The hardware implementation of digital signal processing filters using FPGAs (Field programmable gate arrays) is described in this study. A digital filter is a numerical algorithm that converts a given series of numbers into a second sequence with better qualities. In the frequency domain, a digital filter is better understood. The time domain impulse response is simply convexed with the sampled signal in the filter implementation. A filter is constructed with a frequency domain impulse response that is as near to the intended ideal response as possible given the implementation restrictions. The frequency domain impulse response is subsequently translated into the filter coefficients' time domain impulse response. In certain situations such as linear phase, extremely high stop band attenuation, and very low pass band ripple, digital filters are more compact and power efficient than analogue filters. The response of the filter must be programmable or adaptive, the filter must control phase and extremely low shape factors. For low-rate applications,digital signal processing (DSP) algorithms are often implemented using general purpose programmable DSP devices. For high-performance applications, special purpose fixed function DSP chipsets and application specific integrated circuits (ASICs) are utilised. Xilinx's technological advancements in FPGAs in recent years have opened up new avenues for engineers to create numerous applications on FPGAs. The FPGAs claim the ASIC's high explicitness while avoiding its high development cost and inability to tolerate design changes after manufacturing. FPGAs, which are highly adaptable and design-flexible,allow maximum device use by conserving board space and system power, which are benefits not accessible with many stand-alone DSP processors.

[1]

Chapter 3

Methodology

3.1 Architecture Outline

SHARC DSP is based on a 32-bit super Harvard architecture that includes a unique memory architecture comprised of two large on-chip ,it relaxes the strict separation between instruction and data while still letting the CPU concurrently access two memory buses It includes dual-poned SRAM blocks coupled with a sophisticated I/O processor, which gives SHARC the bandwidth for sustained high speed computations.

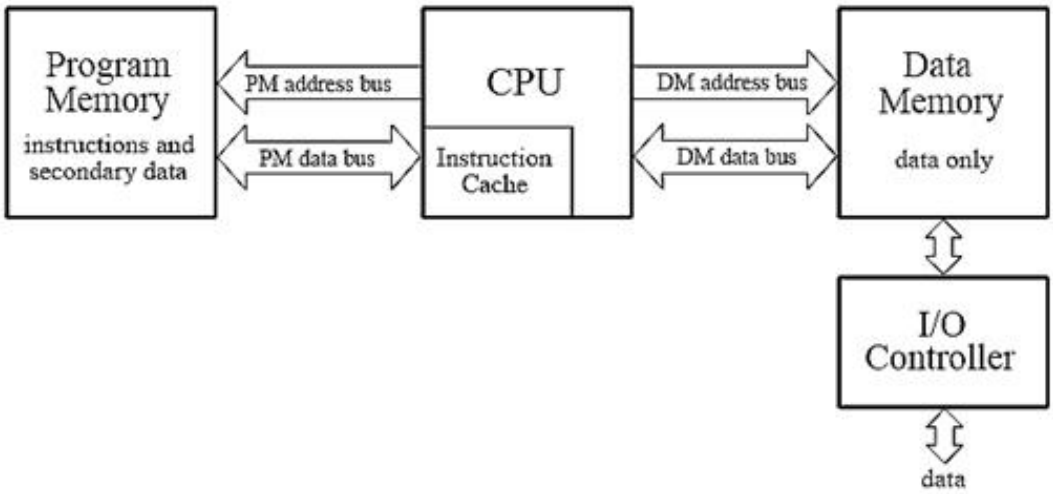

Figure 3.1: SHARC

3.1.1 Execution Modules

- Arithmetic Logic Unit (ALU)
- Barrel Shifter
- Multiply and Accumulate Unit (MAC)

3.1.2 Controller Modules

- Program Counter
- Instruction Decoder
- Control Unit

3.1.3 Memory Modules

- Data Memory
- Coefficient Memory
- Program Memory
- Register File

Chapter 4

Implementation

4.1 Architecture Details

4.1.1 Execution Modules

- Arithmetic Logic Unit (ALU)
  - Arithmetic logic unit is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. It has three bit inputs that are utilised to determine which operation to perform. The ALU decodes the selected lines into eight separate operations. Following the completion of an operation, the ALU sends the result to a destination accumulator.

- Barrel Shifter
  - Barrel shifter circulates data bits in a synchronous manner. The Barrel Shifter is used for Scaling operations such as:
    * Performing a logical shift of the accumulator value
    * Performing arithmetic shift of the accumulator value

- Multiply and Accumulate Unit (MAC)
  - The Multiplier and Adder unit is capable of multiplying and accumulating (MAC).

$$Q = \sum_{n=0}^{Count-1} (+1)\ Multiplicand(n) * Multiplier(n)$$

Figure 4.1: MAC Equation

* Q is the primary data output. Multiplicand and Multiplier are multiplied together. The product added from the current result. The count value in the Equation is set to a fixed value by a parameter.
* The MAC uses Array Multiplier for multiplication and a high-speed hardware Implemented adder for addition

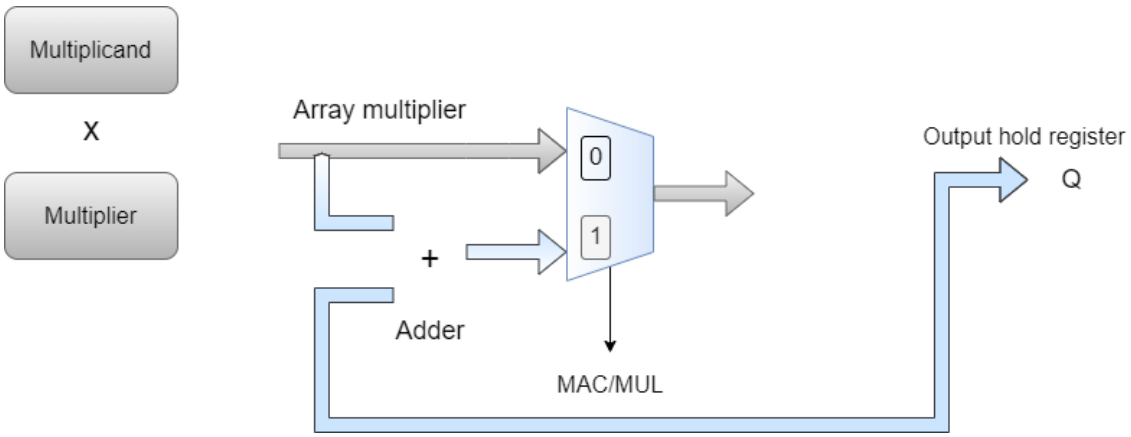

Figure 4.2: MAC

* As in figure[4.1], Mac architecture based on an Array multiplier, a Adder, a Mux and Output hold register

– Multiplier

* The array multiplier is becoming more affordable and popular. The initial stage in a multiplication process is the generation of partial products. These partial products must be combined together to provide the multiplication final product. The ultimate result is created by combining all partial components. As a result, the multiplication process consists of creating partial products and combining them together.

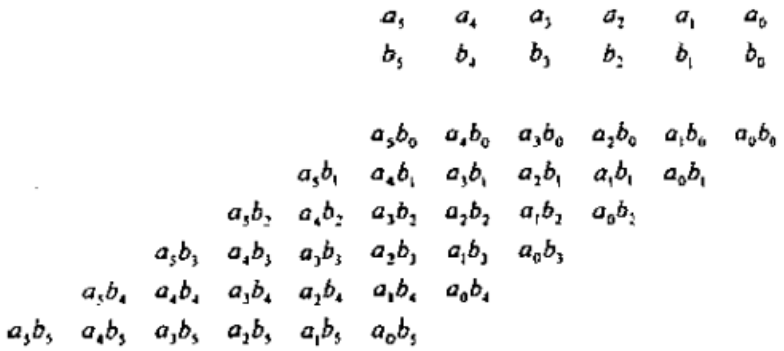

Figure 4.3: Partial multiplication

– Adder

* Many additional mathematical processes, such as multiplication, depend on adders.A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits .

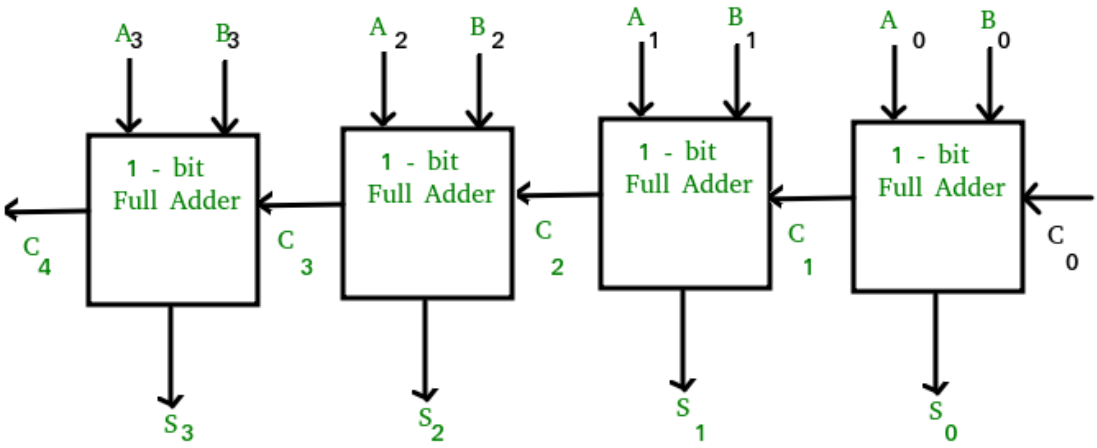

Figure 4.4: Carry-lookahead adder

4.1.2 Controller Modules

- Instruction Decoder
  - The 11-bits instruction is decoded by the instruction decoder. It also creates memory control signals. The following information is included in the instructions.
    * Opcode
    * Data Memory Address
    * Register File Address
    * Coefficient Memory address
  - The instruction has several formats according to below figure ,The first 5-Bits(MSB) of any instruction is the opcode, For the Data Memory the address is the remaining 6-Bits ,For the Coefficient Memory the address is 4-Bits(LSB) and for the Register File ,the file return two values the remaining 6-bits divided to two equal halves ,each half is 3-Bits.

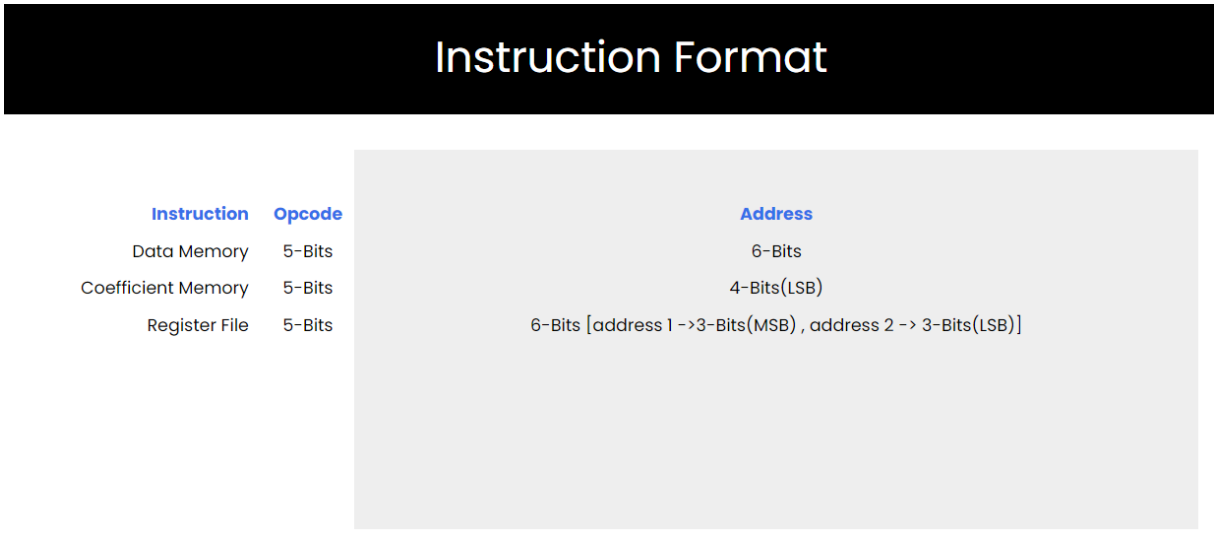

Figure 4.5: Instruction Formats

- Control Unit
  - The control unit decodes the 5-bit opcode further.This module separates the 5-bits of opcode and assigns them to the appropriate functional unit's control register.It has seven control register that linked to the memory units and the register file.These control register control the read , write and branch operations. It also has three control registers. These Control registers are linked to the ALU , MAC and Barrel Shifter units' Function select pins.The following information is contained in this opcode:
    * Defines the function to be carried out.
    * Select a specific functional unit for that operation
    * Specify the source of data to that functional unit.

Control Unit

Instruction	DMRead	DMWrite	CMRead	RFWrite	RFRead	Branch	ALUS	MACS	BSS
Load from DM	1	0	0	0	0	0	0	0	0
Store in DM	0	1	0	0	0	0	0	0	0
Load from CM	0	0	1	0	0	0	0	0	0
Load from RF	0	0	0	0	1	0	0	0	0
Store in RF	0	0	0	1	0	0	0	0	0
ALU	0	0	0	0	0	0	1	0	0
ALU(op = 111)	0	0	0	0	0	1	1	0	0
MAC	0	0	0	0	0	0	0	1	0
BS	0	0	0	0	0	0	0	0	1

Figure 4.6: Control Unit registers

  - The Data Memory has two control register used to load and store, As DMRead used for loading and DMWrite used for storing.
  - The Coefficient Memory has one control register used to load , AS CMRead used for loading.
  - The Register File has two control registers used to load and store , AS RFRead used for loading and RFWrite used for storing.
  - The Branch is a control register that used for branch operation.
  - The ALUS is a control register that select ALU unit for the operation.
  - The MACS is a control register that select MAC unit for the operation.
  - The BSS is a control register that select Barrel Shifter unit for the operation.

- Program Counter

Figure 4.7: Program Counter

  - The Program Counter is a counter whose output is linked to the Program Memory. It creates a count value that is used to address the Program Memory, where the instructions are stored. It has the ability to choose up to 64 memory locations.As in figure[4.3] The Program Counter typically carries the address of the next instruction[1] from the presently running instruction by program counter bus.

4.1.3 Memory Modules

- Data Memory
  - Data Memory is used to store just data from external sources or the output registers of functional units. It can hold 64 words of 8 bits each. Because the address bus is 6-bit width, it can easily handle 64 memory locations .As in figure [4.3] The memory is an array constructed with a distinct read and write control bus[1] that is linked to the control unit. It is also constructed with a data bus[2] that is linked with the register file to save the value temporary for any operation and constructed with an address bus[3] that is linked with the instruction decoder for sending the address of the loaded or stored data .

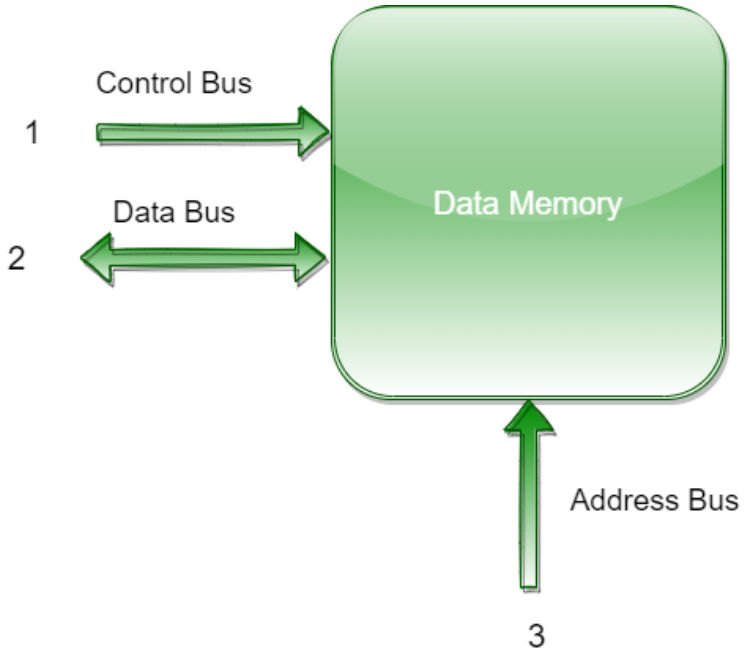

Figure 4.8: Data Memory

- Coefficient Memory

  – Only Filter coefficients are stored in Coefficient Memory. Because the address bus width is four bits, it can easily access 16 memory locations. As in figure[4.4],The memory is an array constructed with a single address bus[1] that is linked to the instruction decoder that sends the address of the loaded data , a read control bus[3] that is linked to the control unit that allow read operation from the memory and a data bus[2] that is linked to the register file to save the value temporary for any operation . Prior to simulation, the memory is configured with a Memory initialization file that provides the required Coefficients for a certain operation.

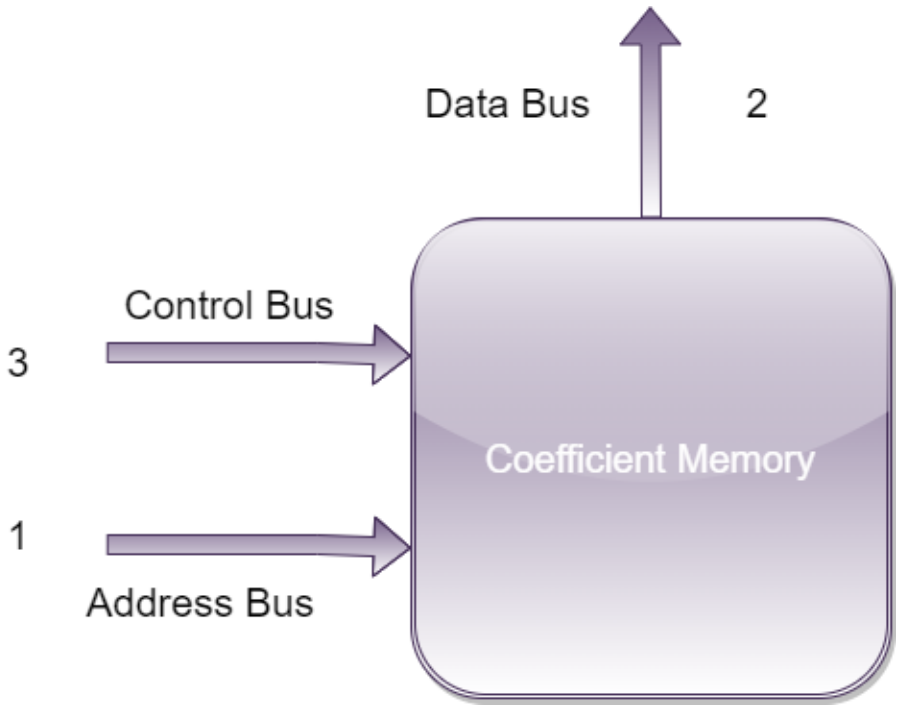

Figure 4.9: Coefficient Memory

- Program Memory
  - Only instructions given as input by the user during simulation are stored in Program Memory. It can hold 64 words of 11 bits each. Because the address bus is 6-bit width, it can easily handle 64 memory locations.As in figure[4.5], The memory is an array constructed with a program counter bus[1] that contain the program counter value that act as the address of the next instruction and a data bus[2] linked to the instruction decoder that holds the next instruction . To begin, the load instructions must be run in order to load the data from the Data Memory to the register file.

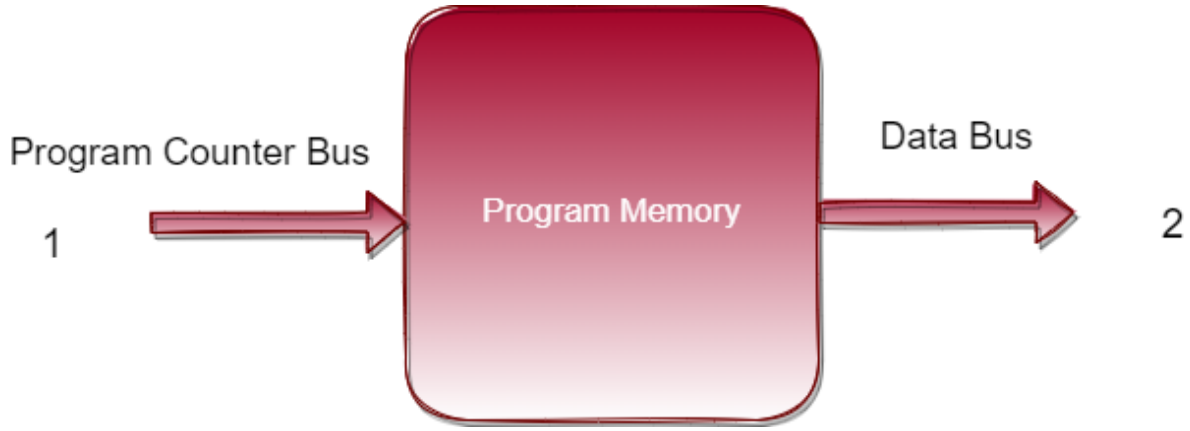

Figure 4.10: Program Memory

  - As in figure[4.6], The instructions has guidelines to be performed . It all depends on a 5-Bits opcode that is distributed to source and function . The source is 2-Bits(MSB) and the function is 3-Bits(LSB) . The source is distributed to four sets :-
    - 11-set :- Used for memory and register file access and it depends on five function :-
      - 110-function :- Used to load from data memory

- 000-function :- Used to store in data memory
- 111-function :- Used to load from coefficient memory
- 101-function :- Used to load from register file
- 001-function :- Used to store in register file

* 00-set :- Used for ALU access and branching and it depends on seven functions :-
  - 010-function :- Addition
  - 001-function :- Subtraction
  - 011-function :- Division
  - 100-function :- Logical shift left
  - 101-function :- Logical shift right
  - 110-function :- Multiplication
  - 111-function :- used for branching and compare

* 10-set :- Used for MAC access

* 01-set :- Used for Barrel shifter access

OPCODE

Instruction	Source	Func
Load from DM	11	110
Store in DM	11	000
Load from CM	11	111
Load from RF	11	101
Store in RF	11	001
ALU(ADD)	00	010
ALU(SUB)	00	001
ALU(DIV)	00	011
ALU(LSL)	00	100
ALU(LSR)	00	101
ALU(MUL)	00	110
ALU(COMPARE)	00	111
MAC	10	-
BS	01	-

Figure 4.11: OPCODE

- Register File

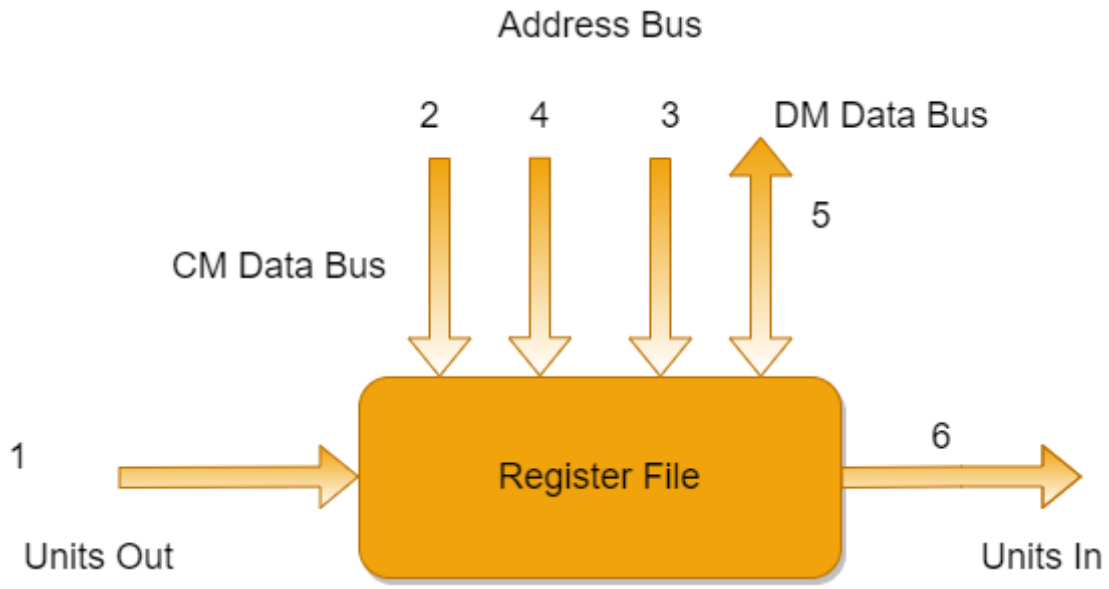

Figure 4.12: Register File

– Register file is used just to temporary store data loaded from memories or the output values from functional units . Register file is an array that can hold 8 words of 8 bits each. Because as in figure[4.7], it is constructed with two address buses[3,4] of 3-bit width linked with the instruction decoder . It can easily handle 8 memory locations . It is constructed with a single coefficient memory data bus[2] linked with the coefficient memory that have the data loaded from the memory. It is also constructed with a data memory data bus[5] that is linked with the Data Memory to load or store data and constructed with a two functional units buses[1,6] that is linked with the ALU , MAC and barrel shifter for sending the data to the functional units [6] or storing the results [1].

4.2 DSP program flow

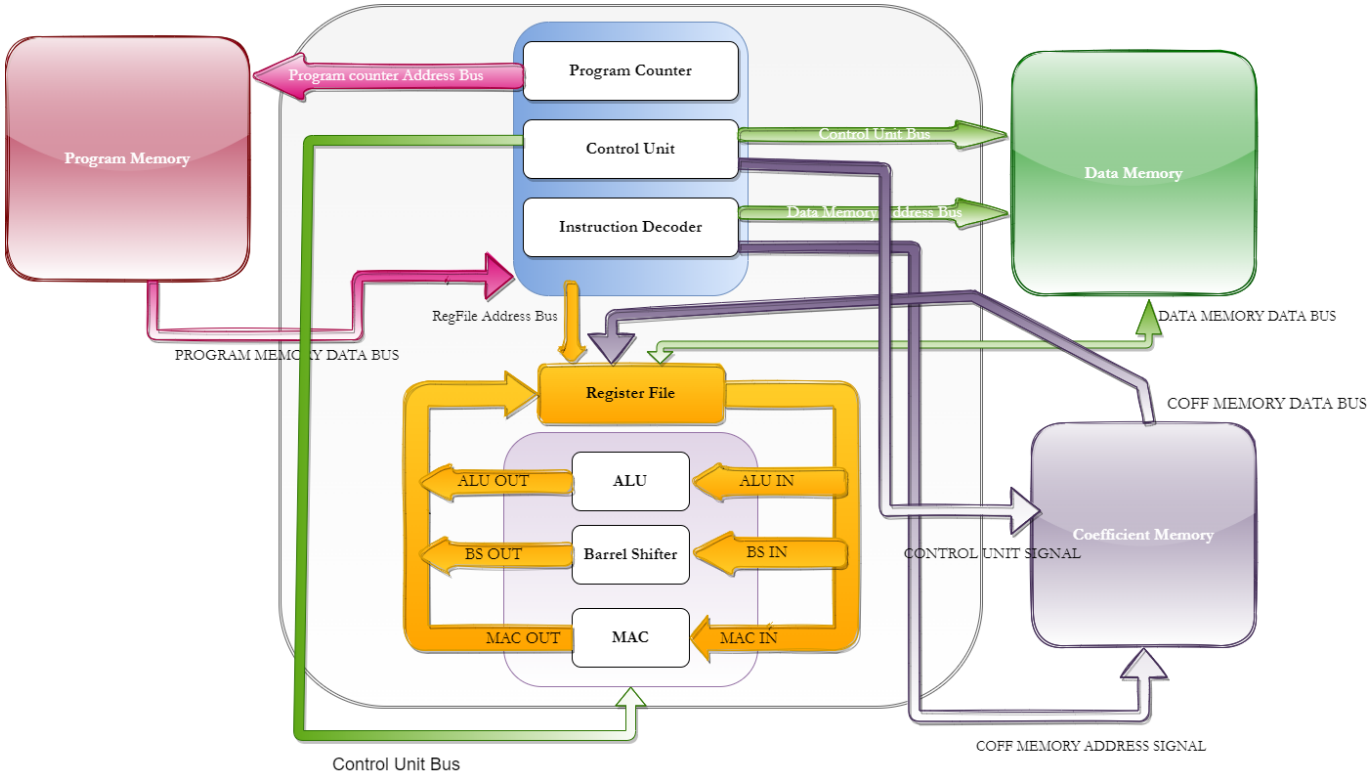

Figure 4.13: DSP Diagram

- The emulator starts from the program counter that has the value of the next instruction . The value delivered by to the program counter address bus then the instruction is loaded from the program memory . The instruction delivered to the instruction decoder by program memory data bus . The instruction is decoded according to the instruction format[figure [4.1]] . The instruction decoder sends the opcode to the control unit . It starts to divide the opcode to source and function as [figure[4.6]] . It determine the unit that will be used by assign the control registers of each unit as [figure[4.2]] by checking the source and function as[figure[4.6]] . Each unit has it's own control registers as :-

- Data Memory (source = 11 , function = 110)
  - DMRead = 1
  - DMWrite = 0
  - CMRead = 0
  - RFRead = 0
  - RFWrite = 0
  - Branch = 0
  - ALUS = 0
  - MACS = 0
  - BSS = 0
  - Address = 6-Bits

  The emulator enters the data memory to load data from the 6-Bits address

- Data Memory (source = 11 , function = 000)
  - DMRead = 0
  - DMWrite = 1
  - CMRead = 0
  - RFRead = 0
  - RFWrite = 0
  - Branch = 0
  - ALUS = 0
  - MACS = 0
  - BSS = 0
  - Address = 6-Bits

  The emulator enters the data memory to store data in the 6-Bits address

- Coefficient Memory
  - DMRead = 0
  - DMWrite = 0
  - CMRead = 1
  - RFRead = 0
  - RFWrite = 0
  - Branch = 0
  - ALUS = 0
  - MACS = 0
  - BSS = 0
  - Address = 4-Bits

  The emulator enters the coefficient memory to load data from the 4-Bits address

- Register File (source = 11 , function = 101)
  - DMRead = 0
  - DMWrite = 0
  - CMRead = 0
  - RFRead = 1
  - RFWrite = 0
  - Branch = 0
  - ALUS = 0
  - MACS = 0
  - BSS = 0
  - Address1 = 3-Bits
  - Address2 = 3-Bits

The emulator enters the register file to load 2 values with address1 and address2

- Register File (source = 11 , function = 001)
  - DMRead = 0
  - DMWrite = 0
  - CMRead = 0
  - RFRead = 0
  - RFWrite = 1
  - Branch = 0
  - ALUS = 0
  - MACS = 0
  - BSS = 0
  - Address1 = 3-Bits
  - Address2 = 3-Bits

The emulator enters the register file to store data with address1

- ALU
  - DMRead = 0
  - DMWrite = 0
  - CMRead = 0
  - RFRead = 0
  - RFWrite = 0
  - Branch = 0
  - ALUS = 1
  - MACS = 0
  - BSS = 0

The emulator enters the ALU to make an arithmetical operation .

- ALU (function = 111)
  - DMRead = 0
  - DMWrite = 0
  - CMRead = 0
  - RFRead = 0
  - RFWrite = 0
  - Branch = 1
  - ALUS = 1
  - MACS = 0
  - BSS = 0

The emulator enters the ALU for checking the condition of the branching

- MAC
  - DMRead = 0
  - DMWrite = 0
  - CMRead = 0
  - RFRead = 0
  - RFWrite = 0
  - Branch = 0
  - ALUS = 0
  - MACS = 1
  - BSS = 0

The emulator enters MAC to make multiplication or accumulation .

- Barrel Shifter
  - DMRead = 0
  - DMWrite = 0
  - CMRead = 0
  - RFRead = 0
  - RFWrite = 0
  - Branch = 0
  - ALUS = 0
  - MACS = 0
  - BSS = 1

  The emulator enters the barrel shifter for making shifting operations

- For all Memory loading and functional operations the values return back to the input register to use it later .

- The emulator till now is a single cycle program .

4.3 Finite State Machine

- Finite State Machine is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The Finite State Machine can change from one state to another in response to some inputs. The change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition.
  - State is a description of the status of a system that is waiting to execute a transition
  - Transition is a set of actions to be executed when a condition is fulfilled or when an event is received.

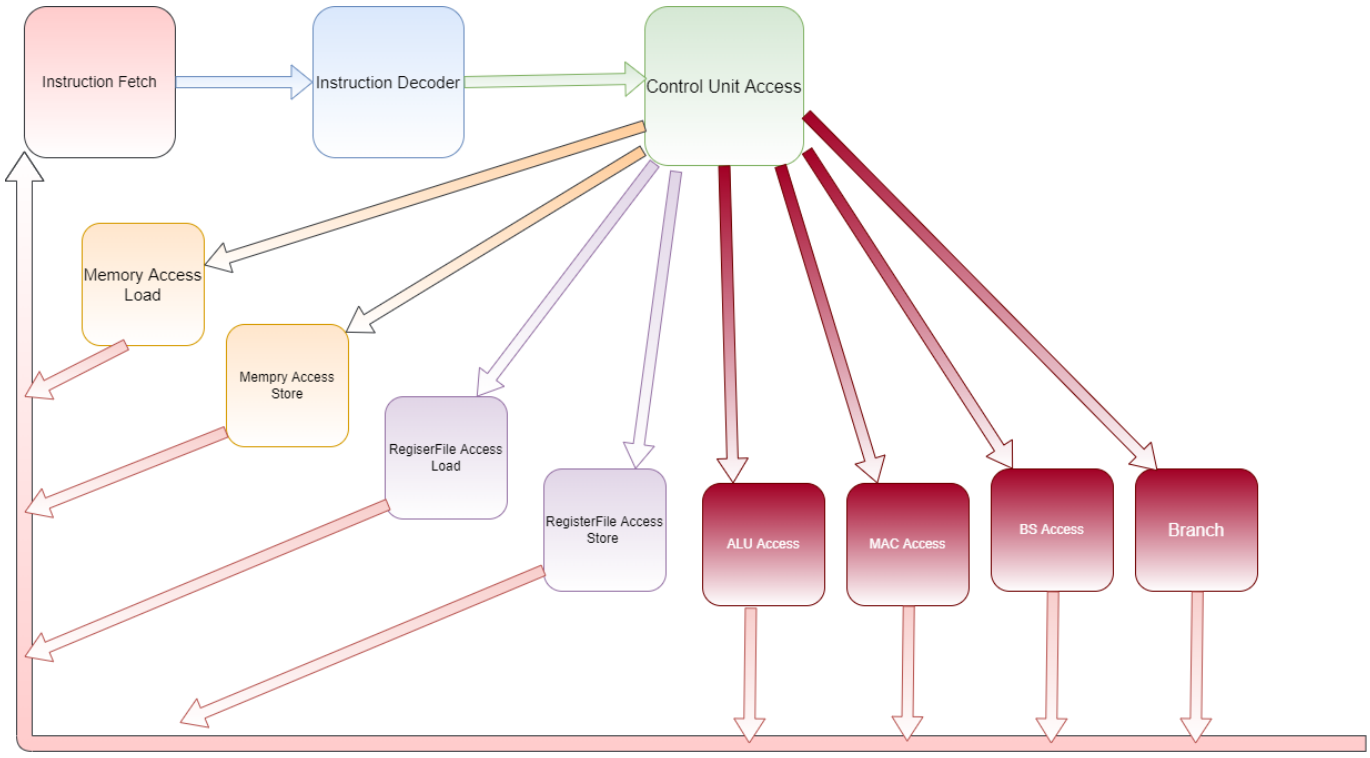

Figure 4.14: Finite State Machine

- As in figure[4.9], The emulator has 11 state for the finite state machine , Each state has a next state .
  - Instruction Fetch state
    - In this state, The emulator fetch the instruction from the program memory with the program counter value.
  - Instruction Decode state
    - In this state, The emulator decode the instruction and return the opcode and the address of data or coefficient memory or register file if the instruction need it.
  - Control Unit Access state
    - In this state , The emulator starts to distribute the opcode to source and function and according to them the return the 9 register signals.
  - Memory Access Load state
    - In this state , The emulator starts to load data from data memory or coefficient memory.
  - Memory Access Store state
    - In this state , The emulator starts to store data in data memory.
  - RegisterFile Access Load state
    - In this state , The emulator starts to load data from register file.
  - RegisterFile Access Store state
    - In this state , The emulator starts to store data in register file.
  - ALU Access state
    - In this state , The emulator starts to execute an operation in the ALU unit.
  - MAC Access state
    - In this state , The emulator starts to execute an operation in the MAC unit.
  - BS Access state
    - In this state , The emulator starts to execute an operation in the barrel shifter unit.

  - Branch state
    - In this state , The emulator starts to execute branch operation .

- Next States
  - Instruction Fetch
    - Next State :- Instruction Decode
  - Instruction Decode
    - Next State :- Control Unit Access
  - Control Unit Access
    - Next State :- Memory Access Load
    - Next State :- Memory Access Store
    - Next State :- RegisterFile Access Load
    - Next State :- RegisterFile Access Store
    - Next State :- ALU Access
    - Next State :- MAC Access
    - Next State :- BS Access
    - Next State :- Branch
  - Memory Access Load
    - Next State :- Instruction Fetch
  - Memory Access Store
    - Next State :- Instruction Fetch
  - RegisterFile Access Load
    - Next State :- Instruction Fetch
  - RegisterFile Access Store
    - Next State :- Instruction Fetch
  - ALU Access
    - Next State :- Instruction Fetch
  - Mac Access
    - Next State :- Instruction Fetch

  - BS Access
    * Next State :- Instruction Fetch
  - Branch
    * Next State :- Instruction Fetch

- Control Unit Access next state guideline
  - The control Unit Access depends on the source and the function as in figure[4.6]:-
    * Memory Access Load
      · 11-source and 110-function
    * Memory Access Store
      · 11-source and 000-function
    * RegisterFile Access Load
      · 11-source and 101-function
    * RegisterFile Access Store
      · 11-source and 001-function
    * ALU Access
      · 00-source
    * MAC Access
      · 10-source
    * BS Access
      · 01-source
    * Branch
      · 00-source and 111-function

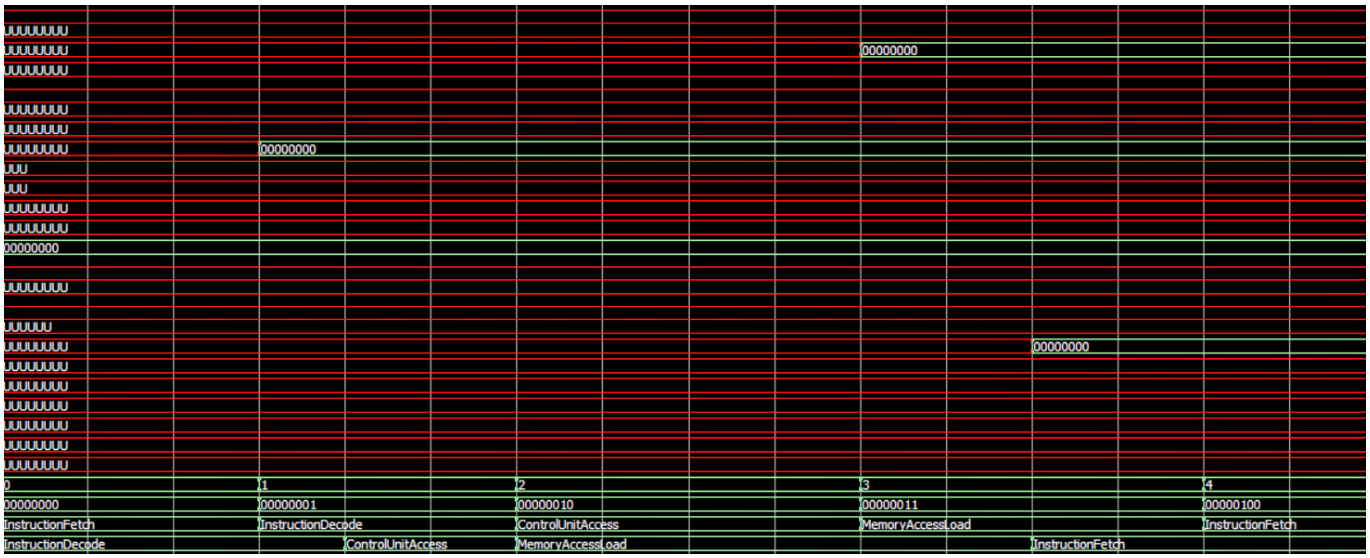

Figure 4.15: Finite State Machine Example

- In figure[4.10], This is an example by executing a load instruction from data memory. The execution begins by instruction fetch and the next state is instruction decode. It enters instruction decode with the opcode and the address of the value that will be loaded and the next state is control unit access. The control access unit starts to detect the next state which is memory access load, Then it enters the memory access load and starts to load the value and detect the next state which is instruction fetch.

4.4 Pipeline

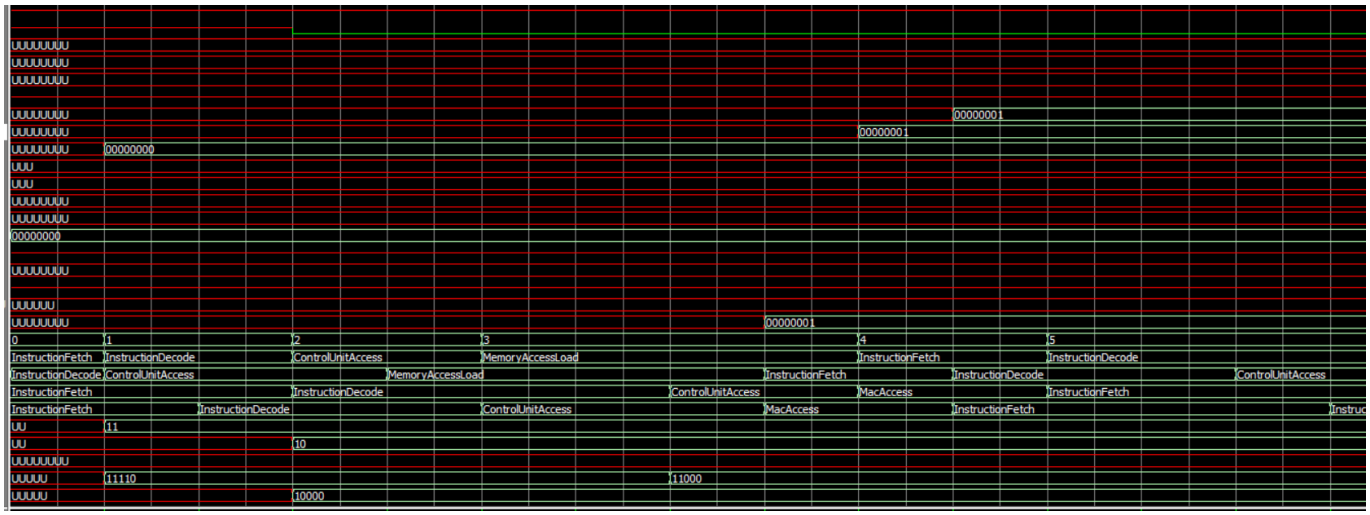

Figure 4.16: Pipeline

- The emulator is pipelined by using two finite state machine. The program counter is incremented once the instruction is fetched and the second finite state machine starts executing.

- As in figure[4.11], The first finite state machine starts fetching the instruction. The second finite state machine holds until the program counter incremented and starts executing. The first finite state machine starts fetching another instruction while the second finite state machine is executing.

- The first instruction takes four cycles to execute and thereafter the subsequent instructions take only one or two cycle to execute according to the functionality of the instruction. Therefore, the throughput has been increased. This initial delay of four cycles is called the latency of the architecture.

4.5 Hardware Implementation

- The hardware implementation based on taking input signal from real life instead of writing values and store it in the data memory. This emulator taking sound signal from real life by an analog sound sensor. The emulator convert the signal from analog to 8-Bits digital value by using analog to digital converter. The digital value considered as an input to the FPGA. The FPGA controls the analog sound sensor by a switch. The analog sound sensor keep taking different signals if the switch is closed. Once the switch is opened, The FPGA lock up over a random value and print it on the FPGA's leds. The FPGA takes the input values and start to store them in the data memory. The FPGA controls the emulator with another switch to be pretty sure that the real values is recorded and stored in the data memory. To start using the emulator, Open the switch that control the emulator. Load instruction from the data memory must be executed first to load the real values from data memory.

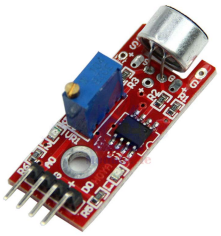

Figure 4.17: Analog Sound Sensor

Figure 4.18: Analog-Digital converter

4.5.1 Components

Analog Sound Sensor

- The emulator uses two analog sound sensors. The analog sound sensor has 4 pins. They are connected with a breadboard.

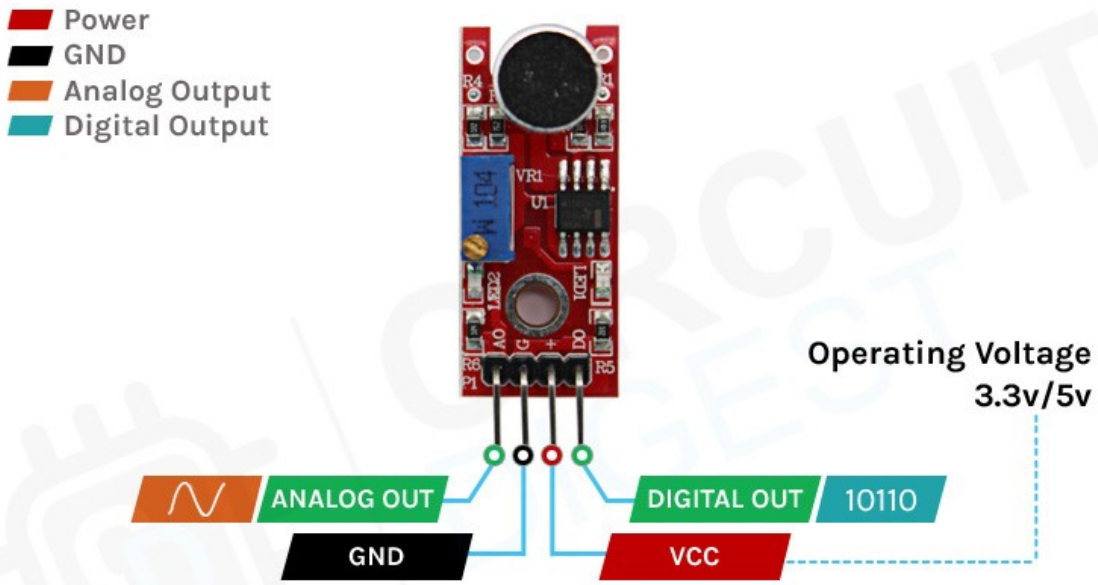

Figure 4.19: Analog Sound Sensor Pins

- VCC
  - It is connected to the VCC of the breadboard
- GND
  - It is connected to the GND of the breadboard
- ANALOG OUT
  - It is connected to the analog-digital converter input signal
- DIGITAL OUT
  - Left opened

Analog-Digital Converter

- ADC0804LCN is an IC(integrated circuit) which converts the input analog voltage to its equivalent digital output. It is a stand alone analog to digital converter. Analog-Digital Converter were used with intelligent hardware/microcontroller etc because microcontrollers didn't have build in analog-digital converter. Now almost every microcontroller has a built in ADC.

- The emulator uses two analog-digital converter. Each converter has 20 pins.

Figure 4.20: Analog-Digital converter pins

- CS (chip select)
  - Connect this pin to the ground of the breadboard to active the analog-digital converter

- RD (read)
  - Connect this pin to the ground of the breadboard. This will bring data from internal registers to the output pins after conversion.

- WR (write)
  - Connect this pin to pin INTR pin[5] of the breadboard. This start conversion of data(analog to digital)
- CLKIN (Clock)
  - Connect external clock to this pin.Connect this pin to a 100pf capacitor.
- Interrupt (INTR)
  - This pin automatically goes low when conversion is done by ADC0804 or when digital equivalent of analog input is ready. This pin is connected with the WR.
- Vin(+)
  - Connect the analog output pin of the analog sound sensor to this pin.
- Vin(-)
  - Connect this pin to the ground of the breadboard.
- GND
  - This is analog ground. Ground this pin.
- Vref/2
  - Left open
- GND
  - This is digital ground.Ground this pin.
- DB7 - DB0
  - The digital output that is converted.
- CLK R
  - Used with clock IN pin with 10k resistor when internal clock source is used.
- VCC
  - Connect this pin to the VCC of the breadboard.

Capacitor

- Since the IC comes with an internal clock we do not need many components to make it work. However to make the internal clock to work we have to use a RC circuit. The IC should be powered by +5V and the both ground pins should be tied to circuit ground. To design the RC circuit simply use a resistor of value 10k and capacitor of 100pf and connect them to CLK R and CLK IN pins.

Figure 4.21: Capacitor

- Connect the longest pin in the CLK IN and CLK R and the another pin in the ground.

4.5.2 Connection with FPGA

- The outputs of the two analog-digital converters are connected as an input to the FPGA. They are connected to Expansion Header JP5.

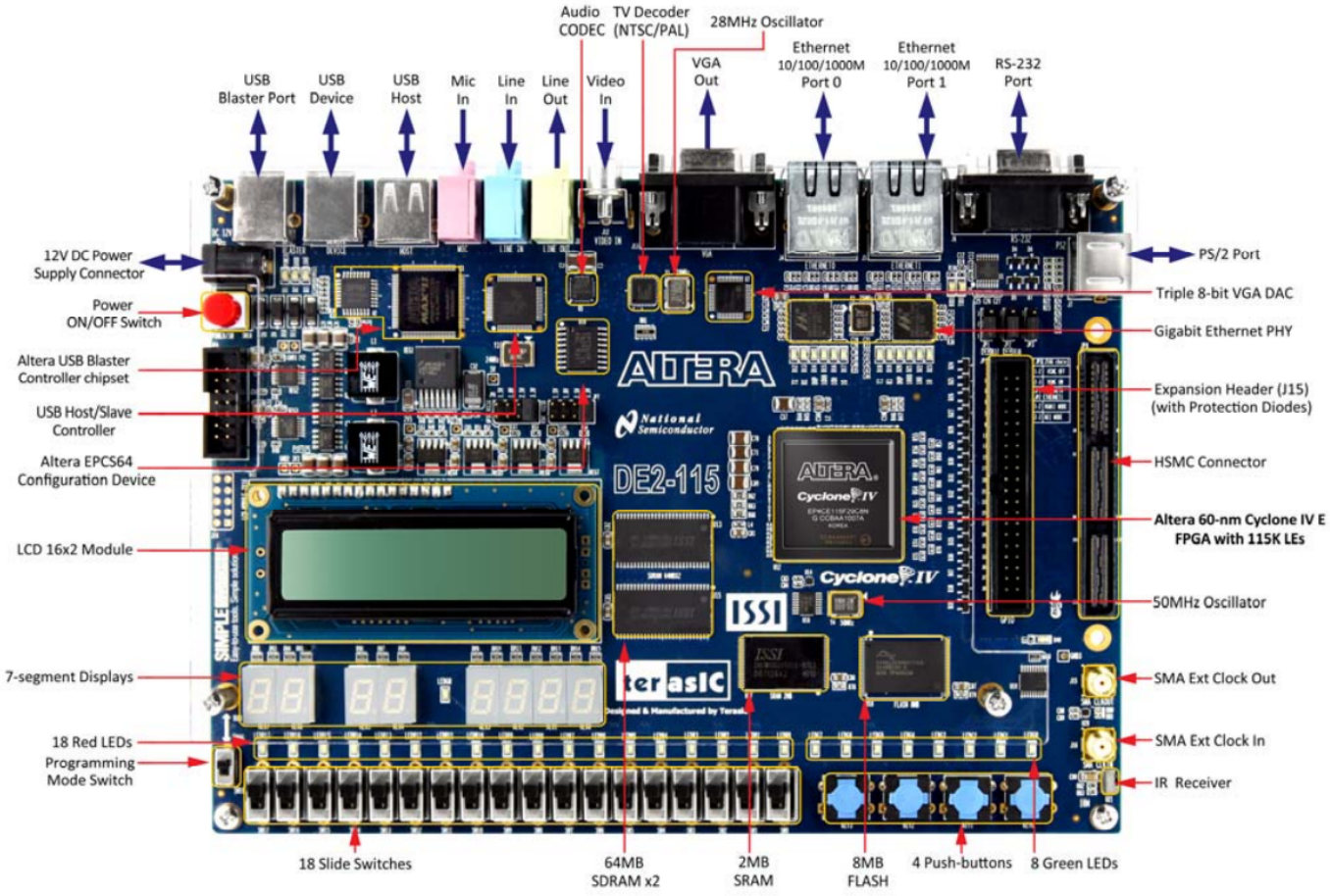

Figure 4.22: FPGA

- The first input is connected to
  - DB7 connected to GPIO[35]
  - DB6 connected to GPIO[33]
  - DB5 connected to GPIO[31]
  - DB4 connected to GPIO[29]
  - DB3 connected to GPIO[27]
  - DB2 connected to GPIO[25]
  - DB1 connected to GPIO[23]
  - DB0 connected to GPIO[21]
- The second input is connected to
  - DB7 connected to GPIO[15]
  - DB4 connected to GPIO[13]
  - DB5 connected to GPIO[11]
  - DB4 connected to GPIO[9]
  - DB3 connected to GPIO[7]
  - DB2 connected to GPIO[5]
  - DB1 connected to GPIO[3]
  - DB0 connected to GPIO[1]
- The first switch that control the selected value from the analog sound sensor is connected to slide switch[0].
- The second switch that control the emulator is connected to slide switch[1].

4.5.3 Example

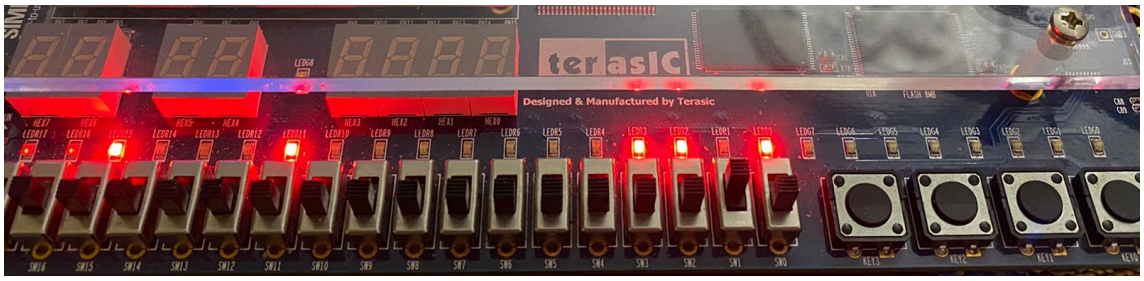

Figure 4.23: Temporary values1

- In figure[4.18], The FPGA temporary stored two values from the analog sound sensor by opening the switch.

- The values then stored in the data memory

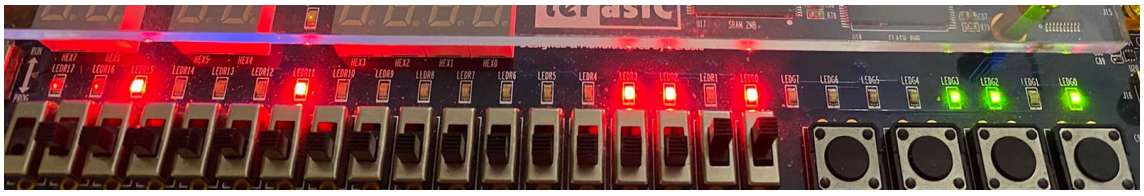

Figure 4.24: Load Instruction run1

- In figure[4.19],The emulator test the real values by executing a load instruction to load the first value from the memory. By opening the second switch, The emulator start to execute and return the value in the green leds.

- In figure[4.20], The FPGA temporary stored two values from the analog sound sensor by opening the switch.

Figure 4.25: Temporary values2

- In figure[4.21],The emulator test the real values by executing a load instruction to load the second value from the memory. By opening the second switch, The emulator start to execute and return the value in the green leds.

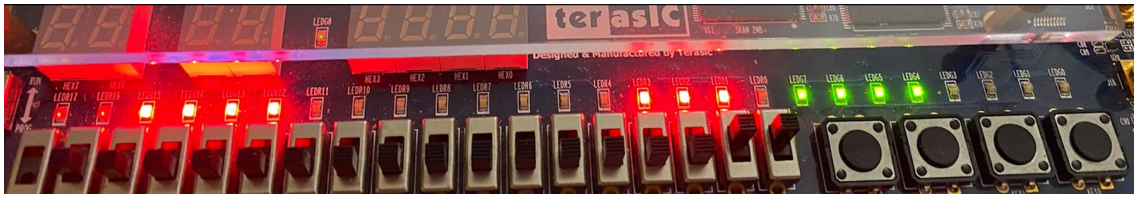

Figure 4.26: Load Instruction run2

4.6 Integration of components

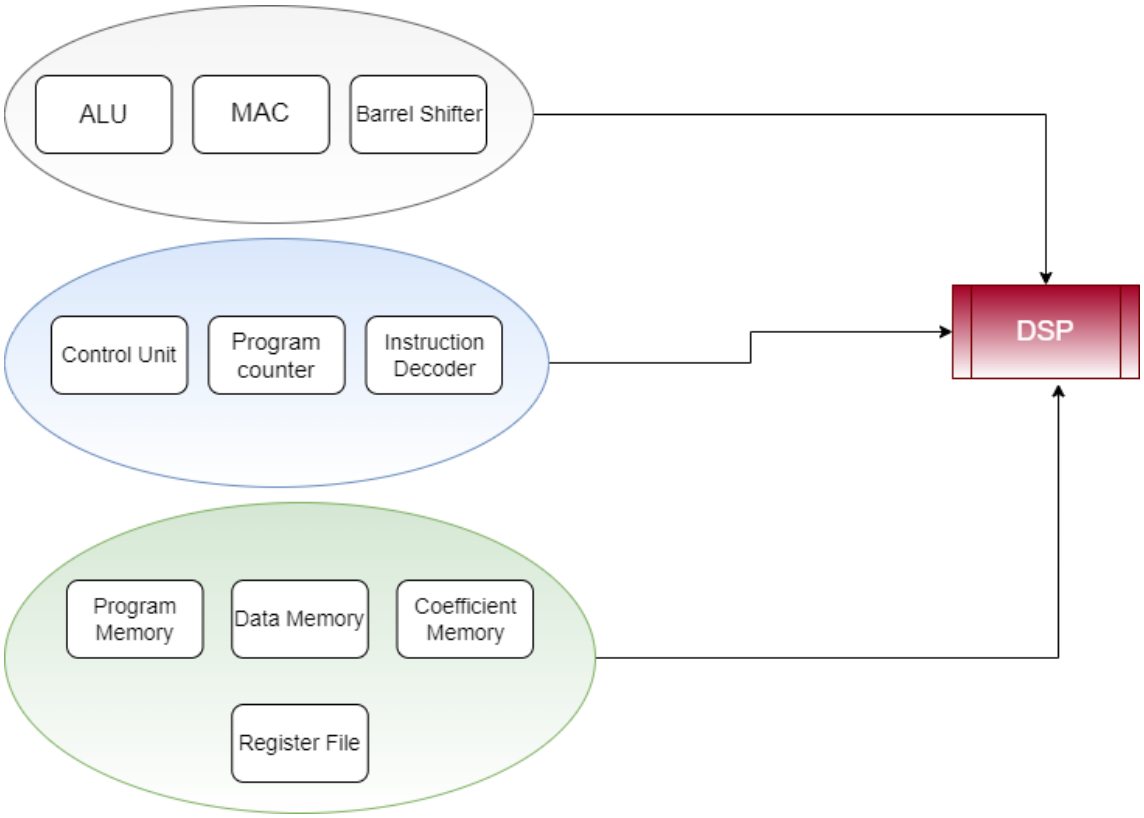

Figure 4.27: Integration of components

- To achieve the design of the Digital Signal Process, the modules were implemented according to the plan sets

- Set 1
  - ALU
  - MAC
  - Barrel Shifter

- Set 2
  - Control Unit
  - Program Counter
  - Instruction Decoder
- Set 3
  - Program Memory
  - Data Memory
  - Coefficient Memory
  - Register File
- The individual modules were combined to form the Digital Signal Processor core.

Chapter 5

Testing and Results

- The performance of these emulator operations is benchmarked against the TMS320DM64x+ DSP by Texas Instruments

5.1 OVERVIEWS OF TMS320DM64X+

5.1.1 TMS320DM64x+

TMS320DM64x+ is a fixed-point DSP developed by Texas Instruments, it uses the Very-Long-Instruction-Word(VLIW) architecture.The chip's special architecture coupled with special instructions work to optimise performance and increase throughput. TMS320DM64x+ chips are programmable through software which allows them to cater for a wide range of tasks.

5.2 Fast Fourier Transform

- Instructions for Fast Fourier Transform
  - Load value1 from data memory
  - Store value1 in the register file
  - Load value2 from coefficient memory
  - Store Value2 in the register file
  - Load the two values from the register file
  - multiply the two values using MAC

- Store the result of the MAC in the data memory
- Store the result of the MAC in the register file
- Load value3 from the data memory
- store value3 in the register file
- Load the result of the MAC and value3 from the register file
- Subtract the result of the MAC from value3
- Store the result of subtraction in the data memory at the address of value2
- Load the result of the MAC and value3 from register file
- Add the result of MAC with value3
- Store the result of the addition in the data memory at address of value1

- The W1 and W2 are the twiddle factors and are stored in the Coefficient Memory prior to the processing. For the convenience of understanding, we have supposed the values of these as the 1 and 2 respectively and are stored in the Coefficient Memory.The algorithm uses four butterflies and each requires 16 instructions and so 4 butterflies needs 64 instructions to be computed.

- As in figure[5.1], The emulator execute the Fast Fourier Transform in 84 cycles only . While the TMS320DM64x+ executes in 88 cycles.

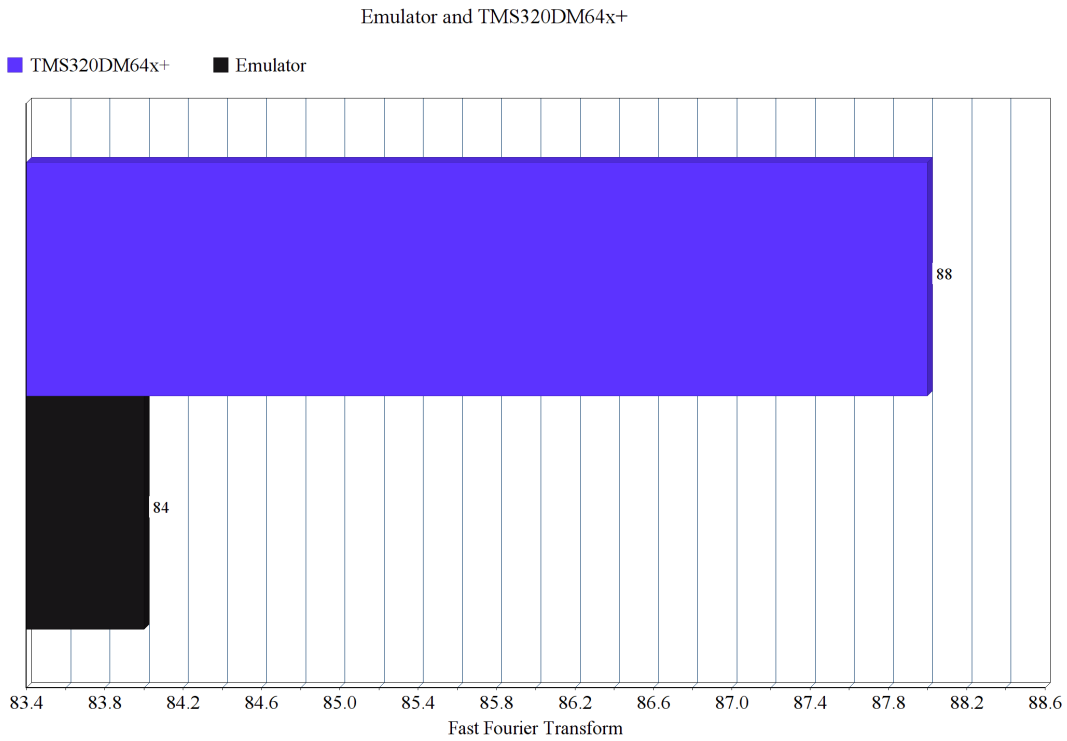

Figure 5.1: Fast Fourier Transform

5.3 Instruction Level Comparison

5.3.1 Arithmetic Instructions

Emulator without pipelining

- The emulator executes addition operation in four cycles, executes the subtraction operation in four cycles and executes the multiplication operation in four cycles.

- The TMS320DM64x+ executes addition operation in one cycle ,executes the subtraction operation in one cycle and executes the multiplication operation in thirteen cycles.

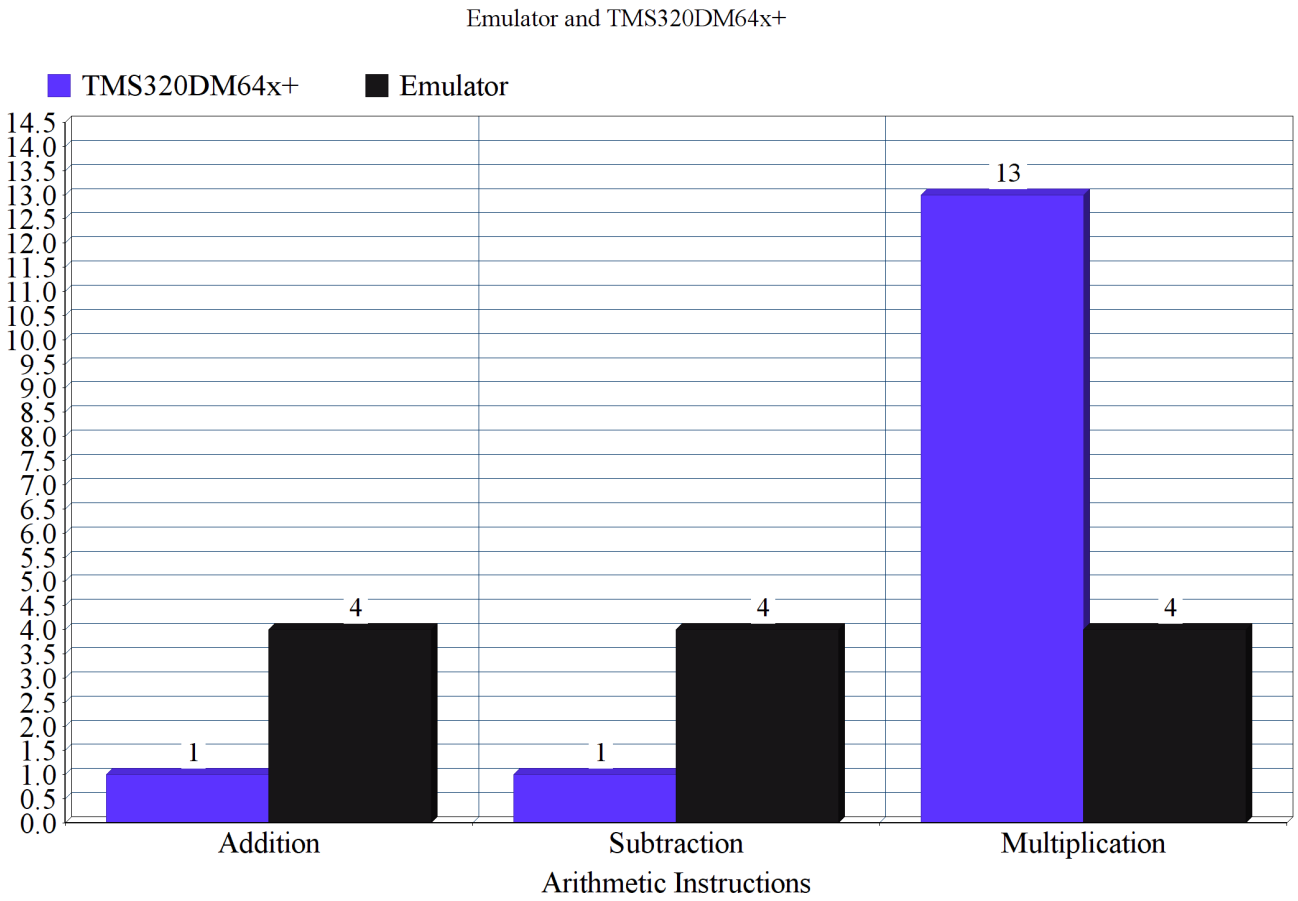

Figure 5.2: Arithmetic Instructions

Pipelined Emulator

- The emulator executes addition operation in two cycles, executes the subtraction operation in two cycles and executes the multiplication operation in one cycle.

- The TMS320DM64x+ executes addition operation in one cycle ,executes the subtraction operation in one cycle and executes the multiplication operation in thirteen cycles.

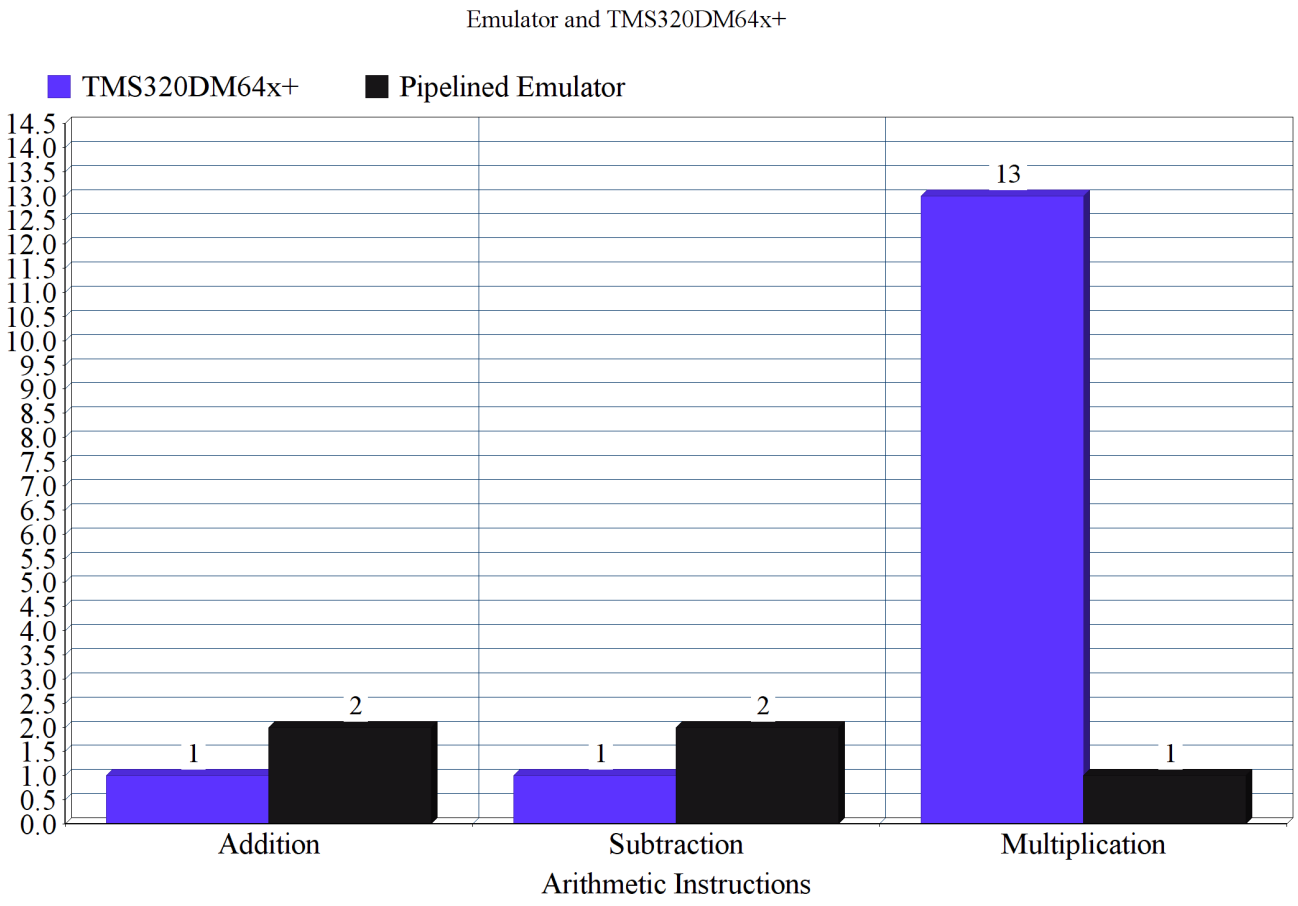

Figure 5.3: Arithmetic Instructions

5.3.2 Loading and Storing Instructions

Emulator without pipelining

- The emulator executes load operation in four cycles and executes the store operation in four cycles.

- The TMS320DM64x+ executes load operation in five cycles and executes the store operation in one cycle.

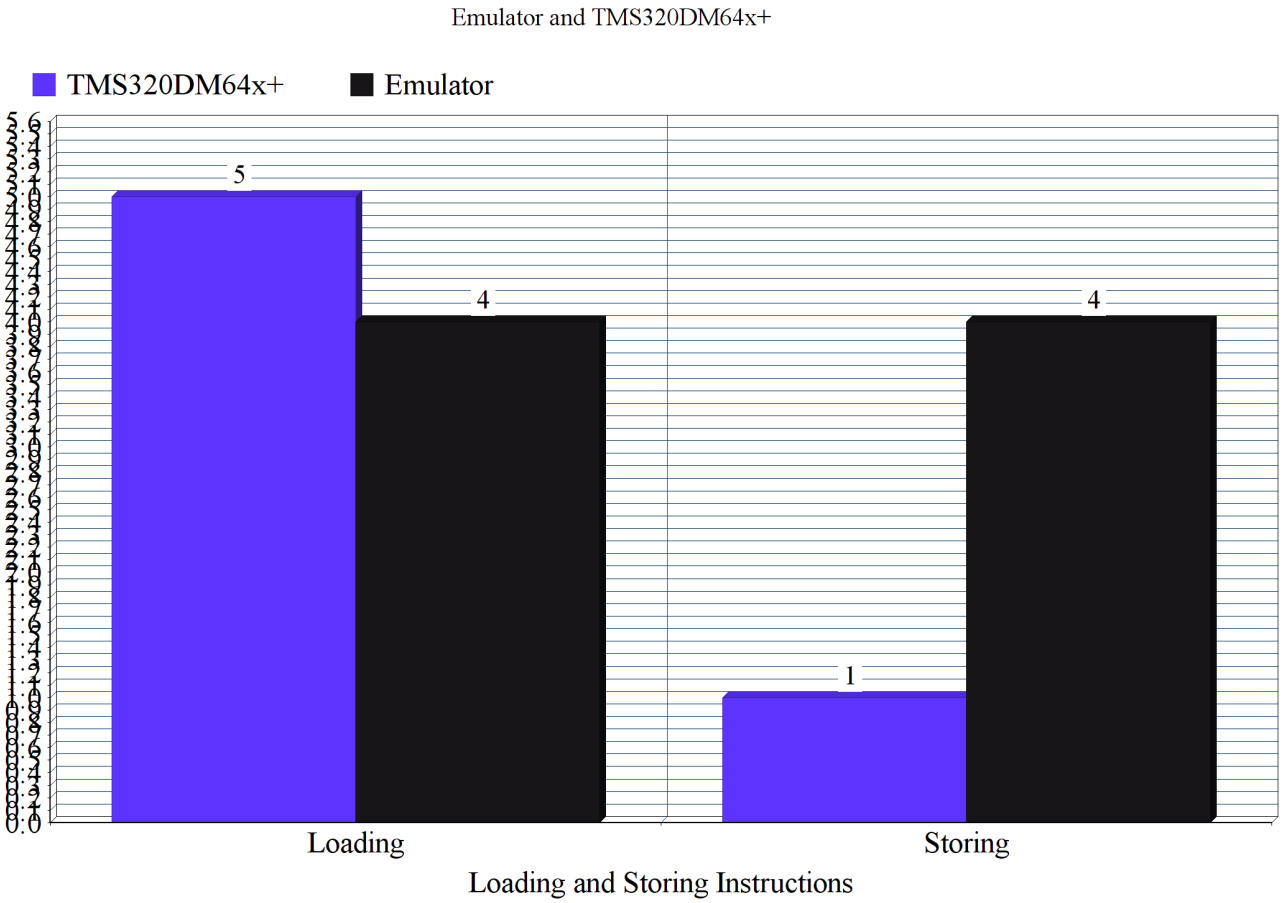

Figure 5.4: Loading and Storing Instructions

Pipelined Emulator

- The emulator executes load operation in one cycle and executes the store operation in one cycle.

- The TMS320DM64x+ executes load operation in five cycles and executes the store operation in one cycle.

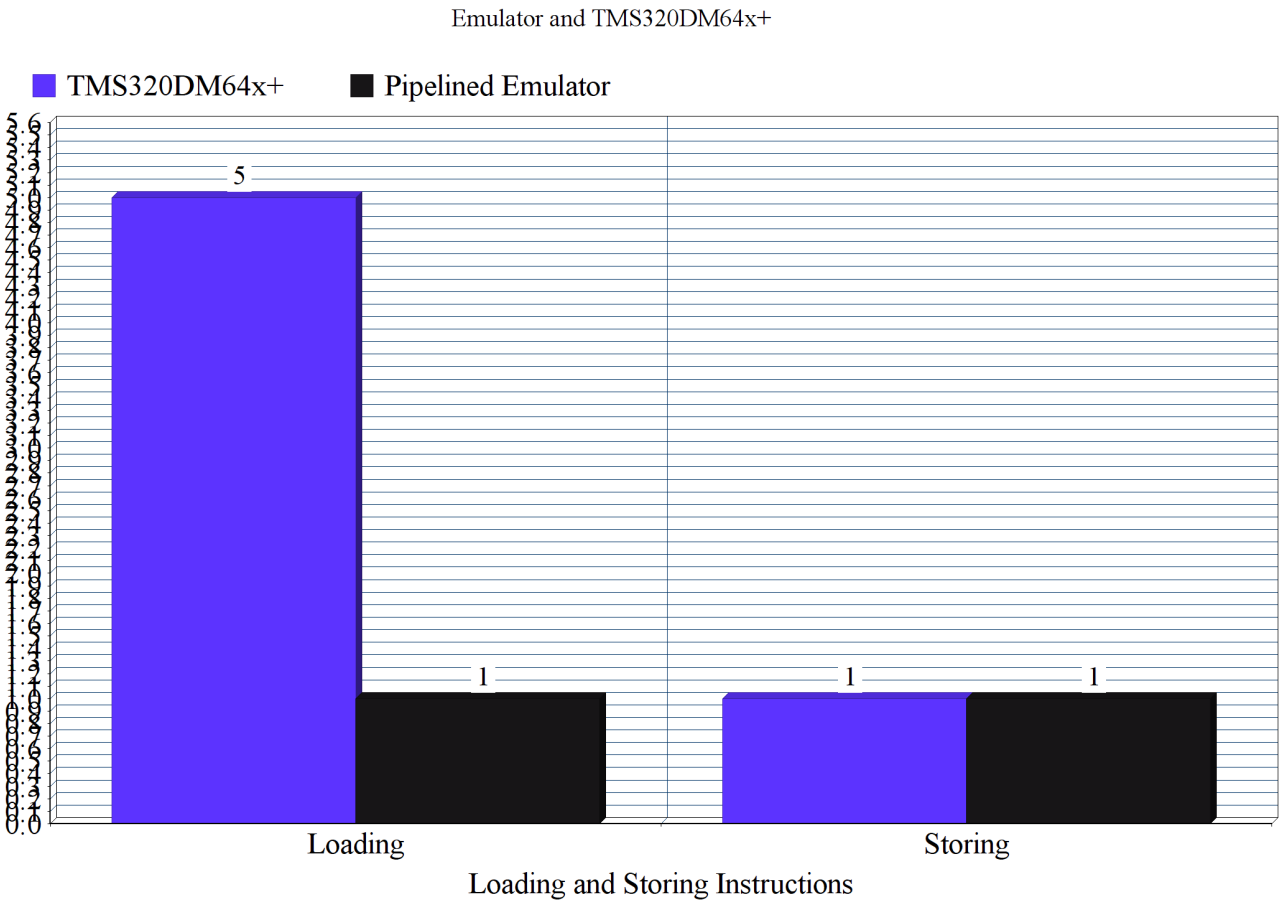

Figure 5.5: Loading and Storing Instructions

5.3.3 Branching Instructions

Emulator without pipelining

- The emulator executes branch operation in five cycles.
- The TMS320DM64x+ executes load operation in twelve cycles.

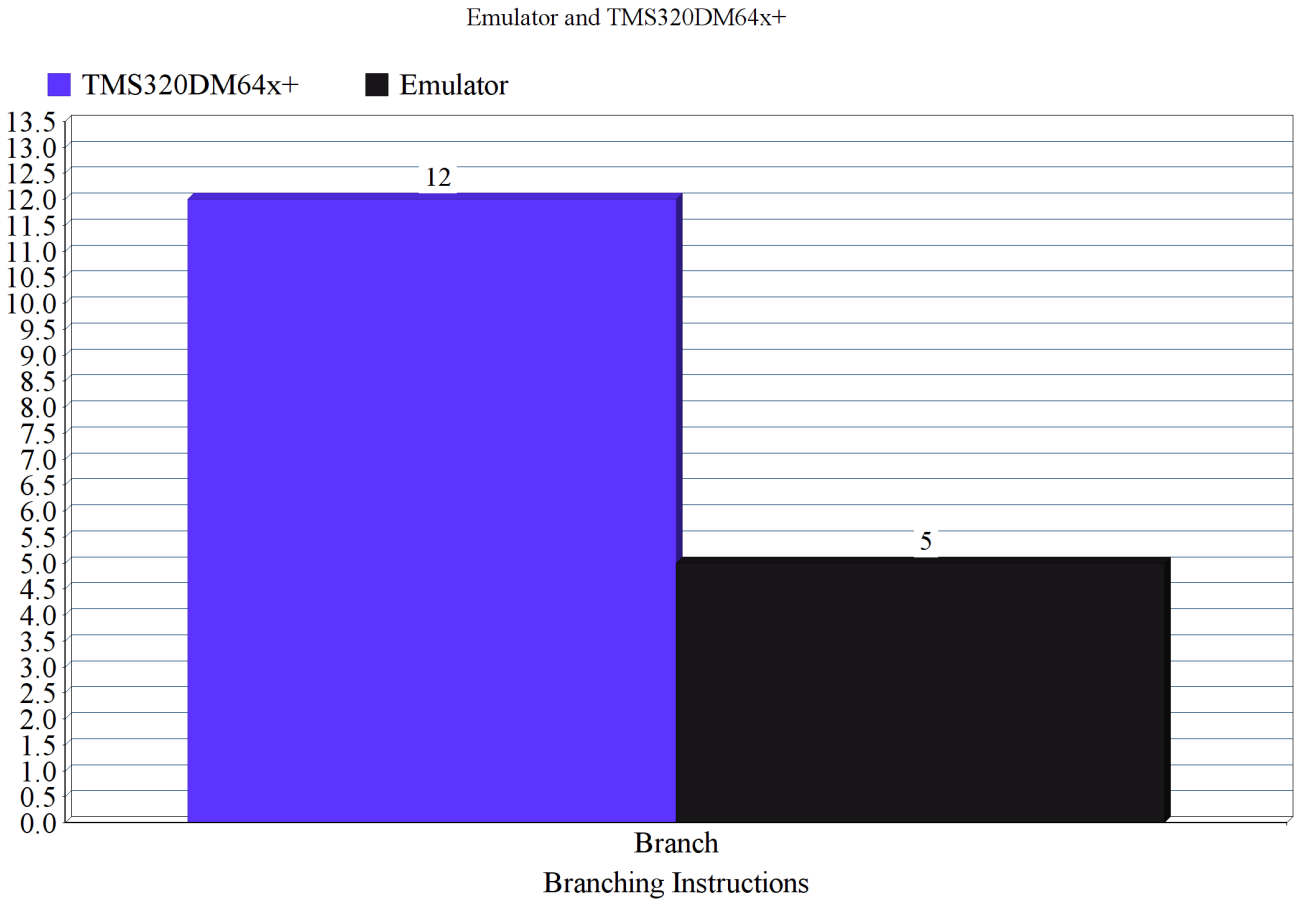

Figure 5.6: Branching Instructions

Pipelined Emulator

- The emulator executes branch operation in two cycles.
- The TMS320DM64x+ executes load operation in twelve cycles.

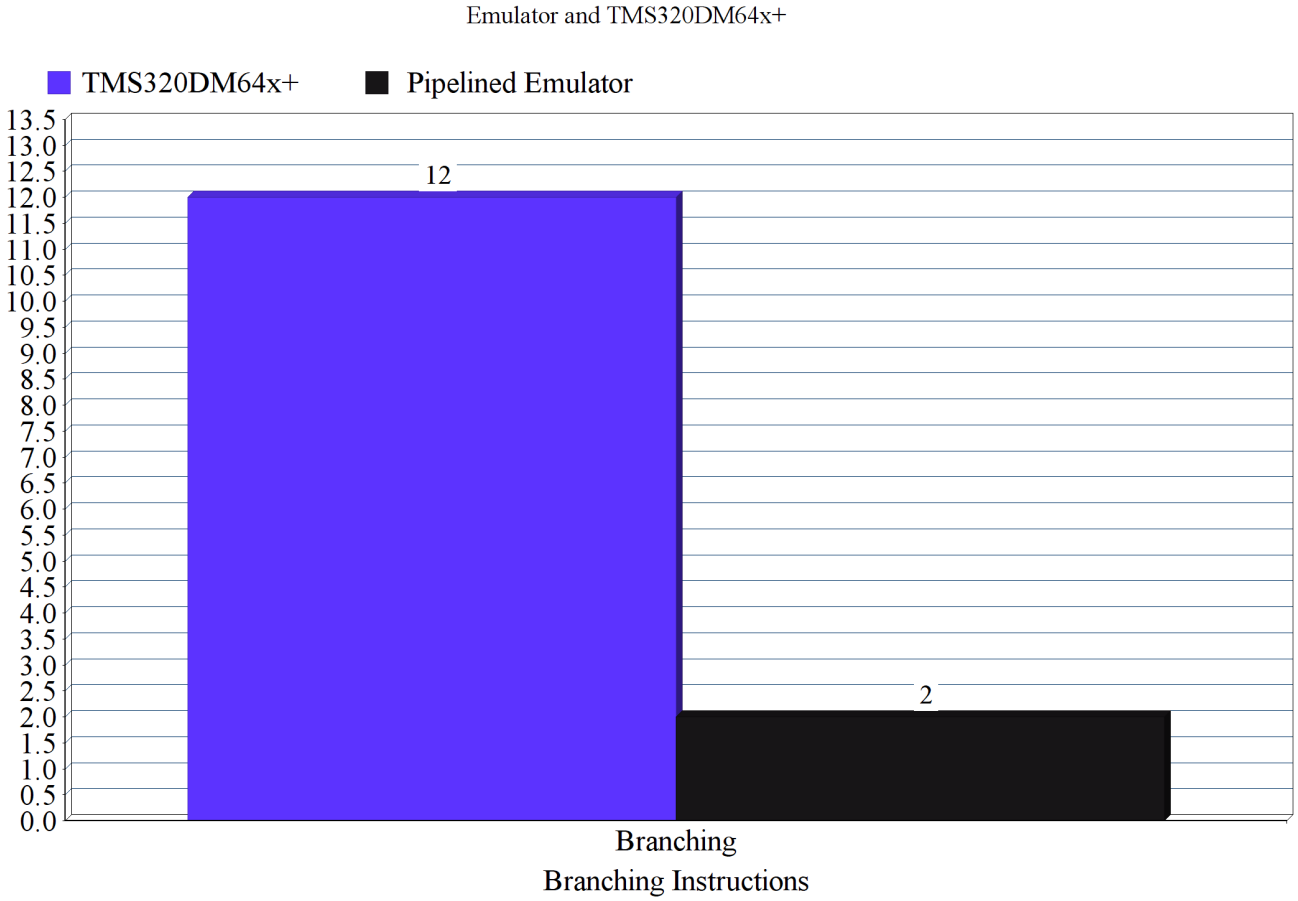

Figure 5.7: Branching Instructions

Chapter 6

Conclusion

- DSPs are a solution to the urgent demand for high-speed and intensive processing technologies that are both inexpensive and simple to use. The objective was to suggest a feasible method for implementing a DSP runtime emulator on an FPGA. Fast Fourier Transform was one of the DSP operations utilised as a test case. The findings of this investigation demonstrate that the FPGA can execute digital signal processing algorithms. Most of the time, the instructions in the emulator are as quick as, if not faster than, the TMS320DM64x+.

- The provided architecture is extensively modularized, making it ideal for VLSl implementation. It has a fast throughput rate and a high input data rate of one sample per cycle. The architecture is validated using VHDL simulation based on register transfer level specifications. The complete design is fitted on a Cyclone IV FPGA with a 50 MHz clock frequency.

Bibliography

[1] Apurva Singh Chauhan, A. Mukund Lal, Varun Maheshwari, and D. Bhagwan Das. Article: Hardware implementation of dsp filter on fpgas. *International Journal of Computer Applications*, 62(16):34–37, January 2013.

[2] Javier Hormigo and Julio Villalba. Hub floating point for improving fpga implementations of dsp applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(3):319–323, 2017.

[3] K.A. Kumar and B. Petrasko. Designing a custom dsp circuit using vhdl. *IEEE Micro*, 10(5):46–53, 1990.

[4] Martin Langhammer and Bogdan Pasca. Design and implementation of an embedded fpga floating point dsp block. In *2015 IEEE 22nd Symposium on Computer Arithmetic*, pages 26–33, 2015.

[5] Sparsh Mittal, Saket Gupta, and Sudeb Dasgupta. System generator: The state-of-art fpga design tool for dsp applications. 03 2008.

[6] N. Mozaffar and N.Z. Azeemi. Design and implementation of a sharc digital signal processor core in verilog hdl. In *7th International Multi Topic Conference, 2003. INMIC 2003.*, pages 247–252, 2003.

[7] M V Ganeswara Rao, P Rajesh Kumar, and A Mallikarjuna Prasad. Implementation of real time image processing system with fpga and dsp. In *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)*, pages 1–4, 2016.

[8] Robert D. Turney, Chris Dick, David B. Parlour, James Hwang, and Robert D. Turney. Modeling and implementation of dsp fpga solutions. 2000.