Design and FPGA Implementation of a Digital Signal Processor

A Thesis submitted in partial fulfillment of the requirements for the Degree of

Bachelor of Technology

In Electronics and Instrumentation Engineering

by

Arifa Parveen

Roll No.109EI0333

Under the supervision of

Dr. Kamala Kanta Mahapatra

Professor

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

Session 2012-2013

Design and FPGA Implementation of a Digital Signal Processor

A Thesis submitted in partial fulfillment of the requirements for the Degree of

Bachelor of Technology

In Electronics and Instrumentation Engineering

by

Arifa Parveen

Roll No.109EI0333

Under the supervision of

Dr. Kamala Kanta Mahapatra

Professor

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

Session 2012-2013

National Institute of Technology, Rourkela

C E R T I F I C A T E

This is to certify that the thesis entitled, '**Design and FPGA Implementation of A Digital Signal Processor'** submitted by Arifa Parveen in partial fulfillment of the requirements for the award of **Bachelor of Technology Degree** in **Electronics and Instrumentation Engineering** at the **National Institute of Technology, Rourkela** is an bonafide piece of work carried out by her under my supervision. To the best of my knowledge the matter embodied in the Thesis has not been submitted by her to any other University/Institute for the award of any Degree/Diploma.

Date

Prof. Kamala Kanta Mahapatra

Dept. of Electronics and Communication Engg.,

National Institute of Technology, Rourkela

ACKNOWLEDGEMENT

The project in itself is an acknowledgement of the inspiration, guidance and the technical assistance contributed to it by many people. It would not have been possible without the help received from them.

First and foremost, I would like to convey my sincere gratitude and deepest regards to my guide **Dr. K K Mahapatra, Professor, Department of Electronics and Communication Engineering, NIT Rourkela**, who has been the continuous driving force behind this work. I thank him wholeheartedly for giving me the opportunity to work under him by trusting my credentials and capabilities, and helping me to explore my potential to the fullest.

I am thankful to **Prof. S. Meher**, Head of the Department, Electronics and Communication Engineering, for permitting me to use the facilities available in the department to carry out the project successfully.

I am thankful to **Prof. Ayas Kant Swain** for allowing me access to the VLSI lab library whenever asked for and **Mr. Jagannath Prasad Mohanty**, PG student in the Department of Electronics and Communication Engineering, NIT Rourkela, for his generous help and continuous encouragement in various ways towards the completion of this project.

Last but not the least I would like to thank all my friends for their support. I am thankful to my classmates for all the thoughtful and mind stimulating discussions we had, prompting me to direct my thoughts beyond the obvious.

Arifa Parveen

ABSTRACT

The project aims at designing a Digital Signal Processor with 32-bit ISA (Instruction Set Architecture) using Verilog HDL and the implementation of its components in FPGA (Field Programmable Gate Array). The processor is demonstrated using uniform 32- bit length instruction set containing instructions that are categorized into three formats, referred to as Register, Immediate and Jump type instructions. The project gives detailed description of design and simulation of the individual modules like the MAC, control module, arithmetic and logic unit, memory units, register file, program counter, data registers, muxes, ALU control, sign extender and the main module instantiating all formerly mentioned modules. For demonstration purposes, the processor is instructed to find the convolution of two input sequences, thus making use of all three instruction formats. After simulation, schematics generation and timing analysis is carried out in Xilinx ISE simulator. The individual modules are implemented and tested in Spartan 3E family XC3S500E FPGA board.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Motivation

For a long time now the field of Digital Signal Processing has been dominated by Microprocessors. This is mainly because they furnish designers with the advantages of single cycle multiply-accumulate instruction as well as special addressing modes.[1] Digital Signal Processors (DSPs) are microprocessors with the following characteristics:

a) Real-time digital signal processing capabilities. Typically, DSPs have to process data in real time, i.e., the correctness of the operation depends heavily on the time when the data processing is completed[2].

b) High throughput. DSPs can sustain processing of high-speed streaming data, like audio and multimedia data processing[2].

c) Deterministic operations. The execution time of DSP programs can be foreseen accurately, therefore guaranteeing a repeatable, desired performance[2]-[4]

d) Re-programmability by software. Different system behaviours might be obtained by recoding the algorithm executed by the DSP instead of by hardware modifications[2].

DSPs appeared on the market in the early 80s. Over the last 15 years they have been the key enabling technology for a large number of electronics products in fields such as communication systems, automotive, instrumentation and military[3]. Table 1 provides an overview of some of these fields and their corresponding typical DSP applications.

Table 1: A short selection of DSP fields of use and specific applications

Field		Application
Communication	Broadband	Video conferencing / phone
		Voice / multimedia over IP
		Digital media gateways (VOD)
	Wireless	Satellite phone
		Base station
Consumer	Security	Biometrics
		Video surveillance
	Entertainment	Digital still /video camera
		Digital radio
		Portable media player / entertainment console
	Toys	Interactive toys
		Video game console
Industrial and entertainment	Medical	MRI
		Ultrasound
		X-ray
	Point of sale	Scanner
		Vending machine
	Industrial	Factory automation
		Industrial / machine / motor control
		Vision system
Military and aerospace		Guidance (radar, sonar)
		Avionics
		Digital radio
		Smart munitions, target detection

Figure 1 shows a real-life DSP application, namely the use of a Texas Instruments (TI) DSP in a MP3 voice recorder–player. The DSP implements the audio and encode functions[3]. In addition, there are tasks carried out like file management, controlling the user interface, and post-processing algorithms such as equalization and bass management[5].

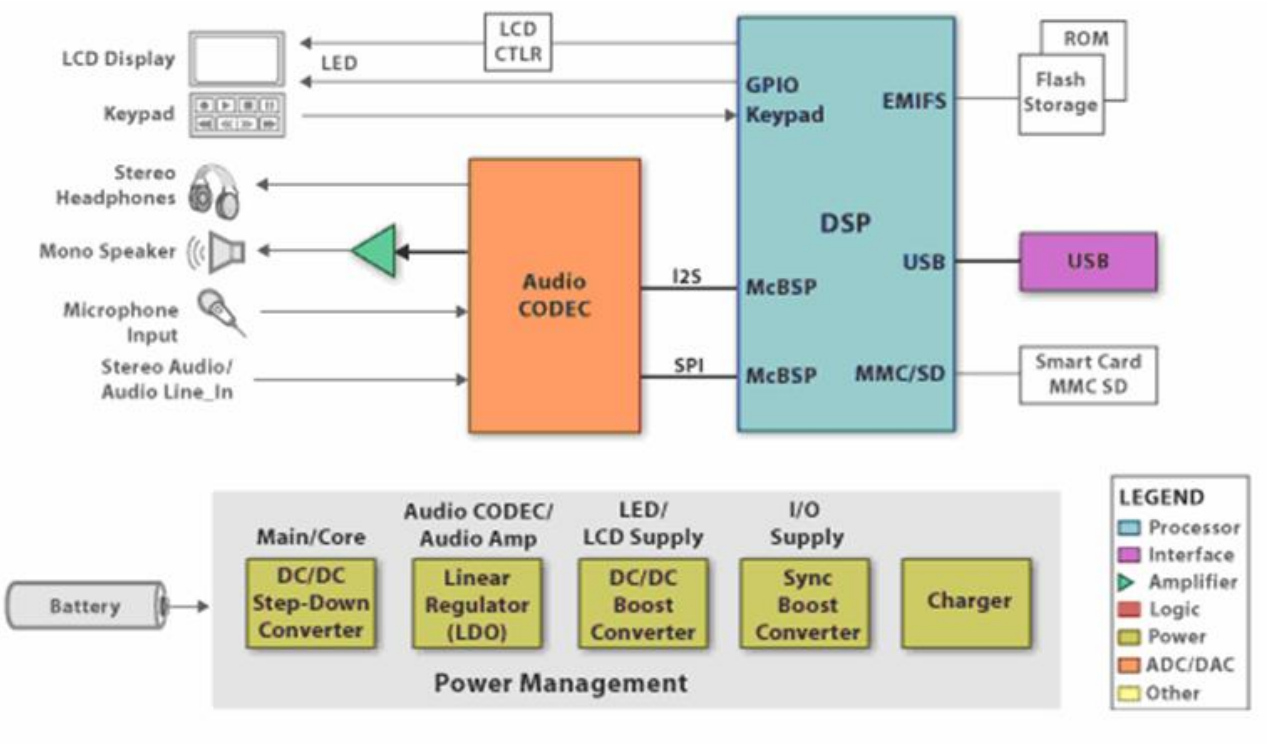


Fig. 1: Use of Texas Instruments DSP in a MP3 player/recorder system. [Courtesy of Texas Instruments from www.ti.com.[6]]

FPGA provides the hardware environment in which dedicated processors can be tested for their functionality. They perform various high-speed operations that cannot be realized by a simple microprocessor. The primary advantage that FPGA offers is On-site programmability[7]. Thus, it forms the ideal platform to implement and test the functionality of a dedicated processor designed using an HDL.

1.2 Problem Statement

The primary objective of this project is to design 32-bit Digital Signal Processor Using Verilog, implement this design on a FPGA, verify and test for its functionality, and analyze its performance.

1.3 Organization of Thesis

The Thesis has been divided into five chapters including this one.

Chapter 1 introduces the project and the motivation behind it.

Chapter 2 deals with literature review of the essentials of the project i.e. Signal Processing, DSPs and Field Programmable Gate Arrays.

The third chapter presents the different algorithms and architectures available during the design of the processor.

Chapter 4 presents the results and related discussions.

Conclusions and future scopes are proposed in Chapter 5.

Chapter 2

LITERATURE REVIEW

2.1 Signal Processing

Signal Processing is the art and science of modifying acquired time-series data[5] for the purposes of analysis or enhancement. A digital signal is a piece of information in binary form. Digital Signal Processing techniques improve signal quality or extract important information by removing unwanted parts of the signal. The various dimensions of digital signal processing are discussed now.

2.1.1 Fourier transforms

it is an extremely powerful mathematical tool[8] that allows us to view our signals in a different domain, inside which several difficult problems become very simple to analyze.

The Fourier transform can be viewed as an extension of the above Fourier series to nonperiodic functions. For totality and for clarity, the Fourier transform is discussed here. If $x(t)$ is a continuous, integrable signal, its Fourier transform, $X(f)$ is given by

$$X(f)=\int_{\mathbb{R}} x(t)e^{-\jmath 2\pi ft}\,dt, \forall f\in\mathbb{R}$$

and the inverse transform is given by

$$x(t)=\int_{\mathbb{R}} X(f)e^{\jmath 2\pi ft}\,df, \forall t\in\mathbb{R}$$

a Fourier transform of a signal tells you what frequencies are present in your signal and in what proportions. The magnitude square of the Fourier transform, $|X(f)|2$ instantly tells us how much power the signal $x(t)$ has at a particular frequency f. Convolutions in the time domain are equivalent to multiplications in the frequency domain. For discrete signals, with the development of efficient FFT algorithms, it is faster to implement a convolution operation in the frequency domain than in the time domain. By being able to split signals into their constituent frequencies, one can easily reject certain frequencies selectively by nullifying their contributions.

2.1.2 Power spectra analysis

"Power Spectra" answer the question "which frequencies contain the signal´s power?"[9] It is in the form of a distribution of power values as a function of frequency, where "power" is considered to be the average of the signal². In frequency domain, this is the square of FFT´s magnitude.

Power spectra can be estimated for the entire signal at once (a "periodogram") or periodograms of segments of the time signal can be averaged together to form the "power spectral density".

2.1.3 Convolution and correlation

You can use convolution to compute the response of a linear system to an input signal. This linear system is defined by its impulse response. The output signal response is convolution of the input signal and the impulse response. Digital filtering is accomplished[10] by determining a linear system´s impulse response that when convolved with the signal accomplishes the desired result (low-pass or high-pass filter).

The correlation algorithm is very similar mathematically to convolution, although, it is used for different purposes. It is usually used to identify the time delay at which two signals "line up", or are "most similar"[10].

2.1.4 Digital Filters

Digital filters are a natural tool when data is already digitized. Reasons for digital filtering the data include:

- Elimination of unwanted signal components ("noise")[11]
- Enhancing of required signal components[11]
- Detecting the presence of desired signals[11]
- Simulation of linear systems (compute the output signal given the input signal and the system´s "transfer function")[11]

Digital filters are generally of two types: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters.

2.2 Digital Signal Processors

2.2.1 Introduction to DSP

DSP is a programmable chip and is capable of carrying out millions of operations per second[12]. Typical DSP applications are audio and video signal processing, image processing and telecommunications devices. DSP technology is the basis of many devices including mobile

phones, personal computers, recorders, CD players, hard disc controllers and modems. Given below is a block diagram of the signal processing sequence.

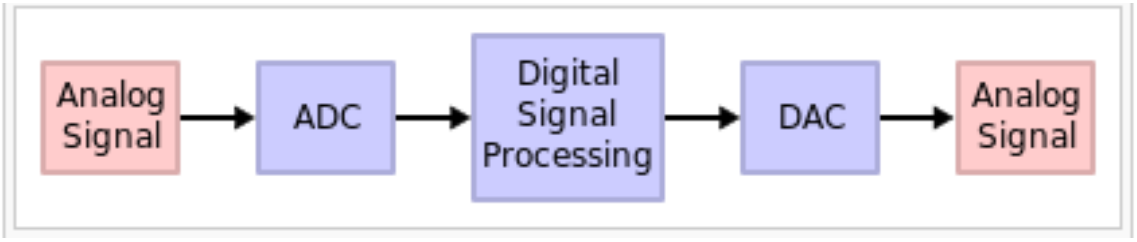


Fig 2: Block Diagram of Signal Processing Sequence

The digital signal processor can be programmed to perform a variety of signal processing, such as filtering, spectrum estimation[13], and other DSP algorithms. Depending on the speed and computational requirements of the application, the digital signal processor may be realized by a general purpose computer, minicomputer, special purpose DSP chip, or any other digital hardware dedicated to performing a particular signal processing task. . A typical digital signal processing system is shown below.

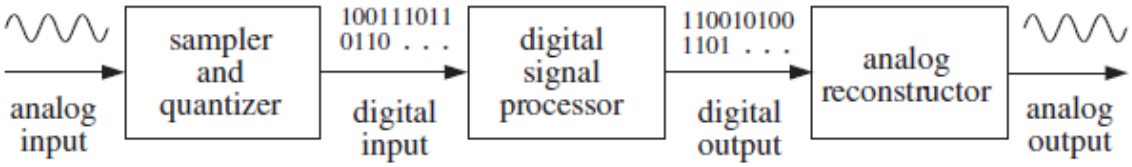


Fig 3: A typical Digital Signal Processing Sequence

DSPs appeared on the market in the early 1980s. Since then, they have undergone an intense evolution in terms of hardware features, integration, and software development tools. DSPs are now a mature technology. This section gives an overview of the evolution of the DSP over their 25-year life span; specialized terms such as ‘Harvard architecture’, ‘pipelining’, ‘instruction set’

or 'JTAG'[14] are used. In the late 1970s there were many chips aimed at digital signal processing; however, they are not considered to be digital signal processing owing to either their limited programmability or their lack of hardware features such as hardware multipliers. The first marketed chip to qualify as a programmable DSP was NEC's MPD7720, in 1981: it had a hardware multiplier and adopted the Harvard architecture. Another early DSP was the TMS320C10, marketed by TI in 1982. From a market evolution viewpoint, we can divide the two and a half decades of DSP life span into two phases: a development phase, which lasted until the early 1990s, and a consolidation phase, lasting until now. Figure 4 gives an overview of the evolution of DSP features together with the first year of marketing for some DSP families.

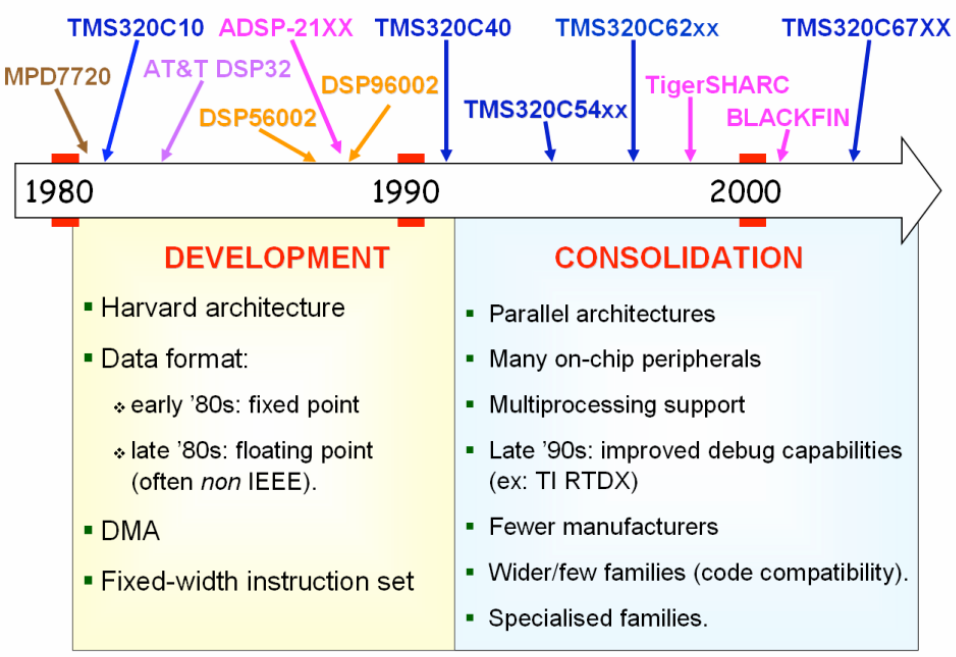


Fig. 4: Evolution of DSP features from their early days until now. The first year of marketing is indicated at the top for some DSP families.

2.2.2 DSP current scenery

The number of DSP vendors is currently somewhat limited: Analog Devices (ADI), Freescale (formerly Motorola), Texas Instruments (TI), Renesas, Microchip and VeriSilicon are the basic players. Amongst them, the biggest share of the market is taken by only three vendors, namely ADI, TI and Freescale[15]. In the accelerator sector one can find mostly ADI and TI DSPs, hence most of the examples in this document will be focused on them.

Table 2: Main ADI and TI DSP families, together with their typical use and performance

Manufacturer	Family	Typical use and performance
TI	TMS320C2x	Digital signal controllers
	TMS320C5x	Power efficient
	TMS320C6x	High performance
ADI	SHARC	Medium performance. First ADI family (now three generations)
	TigerSHARC	High performance for multi-processor systems
	Blackfin	High performance and low power

DSP architecture has been shaped by the requirements of predictable and accurate real-time digital signal processing. An example is the Finite Impulse Response (FIR) filter, with the corresponding mathematical equation (1), where *y* is the filter output, *x* is the input data and *a* is a vector of filter coefficients. Depending on the application, there might be just a few filter coefficients or many hundreds or more.

$$y(n) = \sum_{k=0}^{M} a_k \cdot x(n-k) \ .$$

As shown in, the main component of a filter algorithm is the 'multiply and accumulate' operation, typically referred to as MAC[16]. Coefficients data have to be retrieved from the memory and the whole operation must be executed in a predictable and fast way, so as to sustain a high throughput rate. Finally, high accuracy should typically be guaranteed. Table 3 shows a selection of processing requirements together with the main DSP hardware features satisfying them.

Table 3: main Requirements and corresponding hardware implementations for predictable accurate real time digital signal processing

Processing requirements	Hardware implementations satisfying the requirement
3.2 Fast data access	• High-bandwidth memory architectures • Specialized addressing modes • Direct Memory Access (DMA)
3.3 Fast computation	• MAC-centred • Pipelining • Parallel architectures (VLIW, SIMD)
3.4 Numerical fidelity	• Wide accumulator registers, guard bits, etc.
3.5 Fast execution control	• Hardware-assisted, zero-overhead loops, shadow registers, etc.

Traditional general-purpose microprocessors are based upon the Von Neumann architecture, shown in Fig. 5(a). This consists of a single block of memory, containing both data and program instructions, and of a single bus (called data bus) to transfer data and instructions from/to the CPU. The disadvantage of this architecture is that only one memory access per instruction cycle[17] is possible, thus constituting a bottleneck in the algorithm execution. DSPs are typically based upon the Harvard architecture, shown in Fig. 5(b), or upon modified versions of it, such as

the Super-Harvard architecture shown in Fig. 5(c). In the Harvard architecture there are separate memories for data and program instructions, and two separate buses connect them to the DSP core. This allows fetching program instructions and data at the same time, thus providing better performance at the price of an increased hardware complexity and cost.

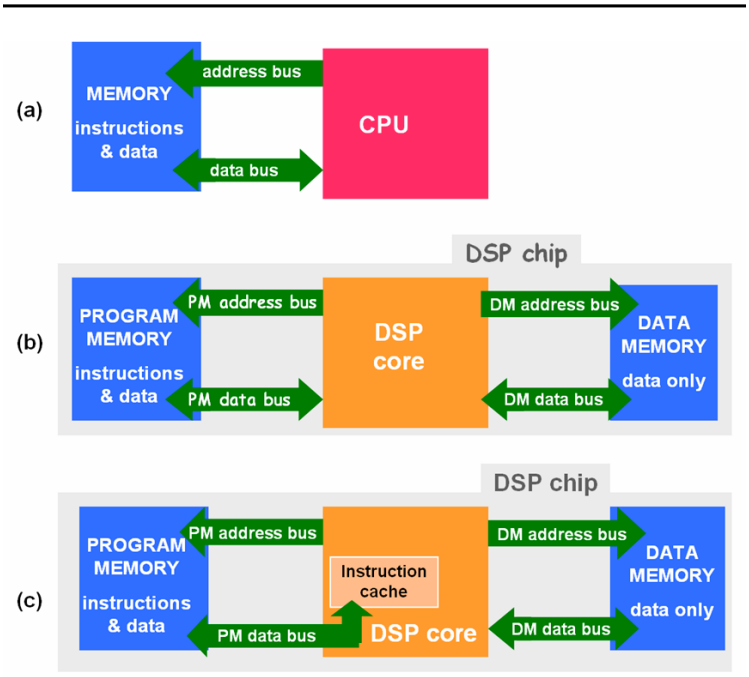


Fig. 5: (a) Von Neumann architecture, typical of traditional general-purpose microprocessors.b) Harvard and (c) Super-Harvard architectures, typical of DSPs.

2.2.3 Assembly language

The assembly language is very close to the hardware, as it explicitly works with registers and it requires a detailed knowledge of the inner DSP architecture. To write assembly code typically takes longer than to write high-level languages; additionally, it is often more difficult to understand other people's assembly programs than to understand programs written in high-level languages. The assembly grammar/style and the available instruction set/peripherals depend not only on the DSP manufacture, but also on the DSP family and on the targeted DSP. As a

consequence, it might be difficult or even impossible to port assembly programs from one DSP to another. For instance, for DSPs belonging to the TI C6xxx family there is about an 85% assembly code compatibility[18], i.e., when going from a C62x to a C64x DSP there are no issues but if moving from a C64x to a C62x one might have to introduce some changes in the code owing to the different instruction set.

2.3 FPGA

FPGA or Field Programmable Gate Arrays can be programmed or configured by the user or designer after manufacturing and during implementation. Hence they are otherwise known as On-Site programmable. Unlike a Programmable Array Logic (PAL) or other programmable device, their structure is similar to that of a gate-array or an ASIC. Thus, they are used to rapidly prototype ASICs, or as a substitute for places where an ASIC will eventually be used [19]. This is done when it is important to get the design to the market first. Later on, when the ASIC is produced in bulk to reduce the NRE cost, it can replace the FPGA. The programming of the FPGA is done using a logic circuit diagram or a source code using a Hardware Description Language (HDL) to specify how the chip should work. FPGAs have programmable logic components called ‚logic blocks', and a hierarchy or reconfigurable interconnects which facilitate the ‚wiring' of the blocks together. The programmable logic blocks are referred to as configurable logic blocks and reconfigurable interconnects are referred to as switch boxes. CLBs can be programmed to perform complex combinational functions, or simple logic gates. In most FPGAs the logic blocks also include memory elements, which can be as simple as flip-flops, or as complex as complete blocks of memory.

2.3.1 FPGA Architecture

FPGA architecture depends on its vendor, but they are usually variation of that shown in the figure. The architecture comprises Configurable Logic Blocks, Configurable Input/Output blocks and Programmable Interconnects. It also houses a clock circuitry to drive the clock signals to each logic block. Additional logic resources like ALUs, Decoders and memory may be available. The number of CLBs and I/Os required can easily be determined from the design but the number of routing tracks is different even within the designs employing the same amount of logic.

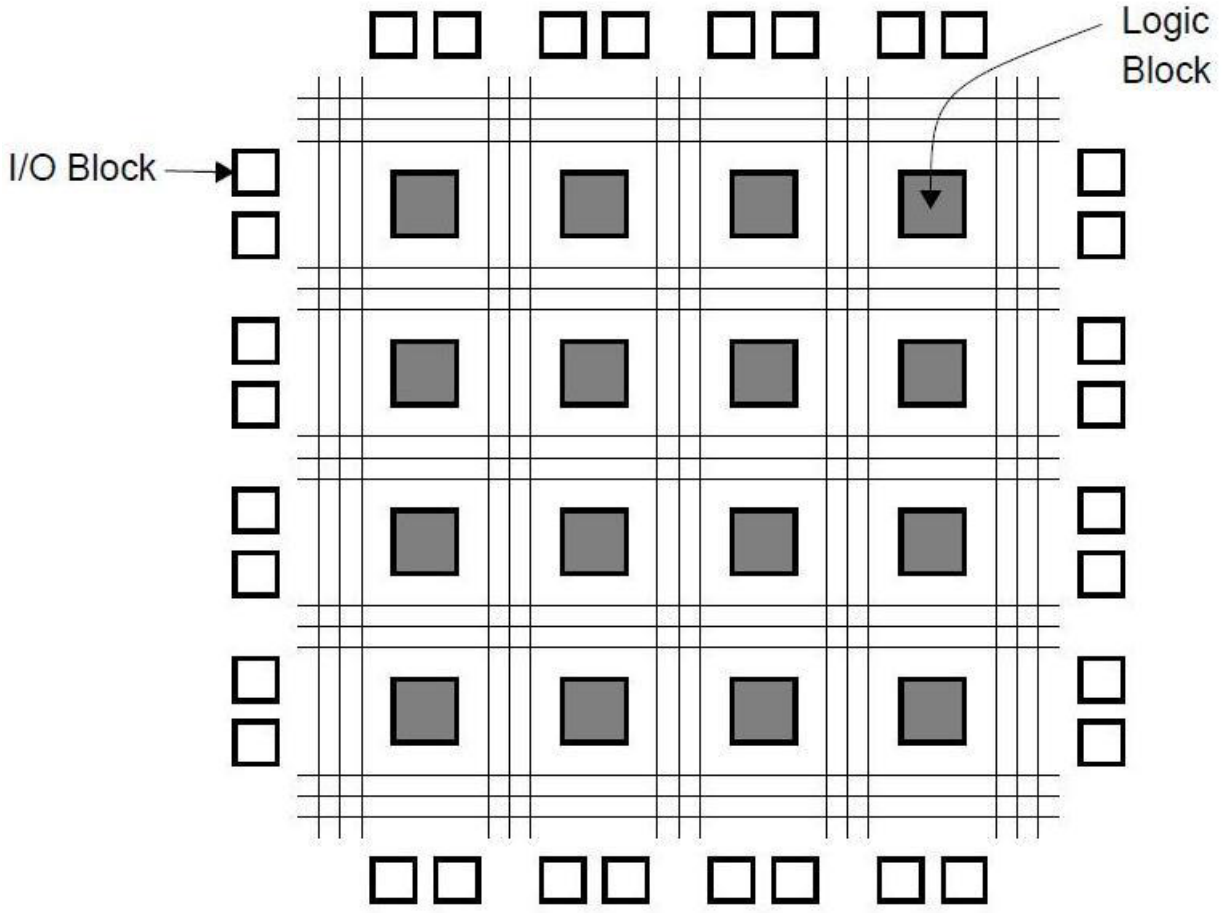


Fig. 6: FPGA Architecture

1. Configurable Logic Blocks

They contain the logic for the FPGA. CLBs contain RAM for creating arbitrary combinatorial logic functions. It also has flip-flops for clocked storage elements, and multiplexers that route the logic within the block to/from external resources.

2. Configurable I/O Blocks

Configurable I/O block is used to route signal towards and away from the chip. It comprises input buffer, output buffer with three states and open collector output controls. Pull-up and Pull-down resistors may also be present at the output. The output polarity is programmable for active high or active low output.

3. Programmable Interconnects

FPGA interconnect is similar to that of a gate array ASIC and different from a CPLD. There are long lines that interconnect critical CLBs located physically far from each other without introducing much delay. They also serve as buses within the chip. Short lines that interconnect CLBs present close to each other are also present. Switch matrices that connect these long and short lines in a specific way are also present. Programmable Switches connect CLBs to interconnect lines and interconnect lines to each other and the switch matrix. Three-state buffers connect multiple CLBs to a long line creating a bus. Specially designed long lines called Global Clock lines are present that provide low impedance and fast propagation times.

4. Clock circuitry

Special I/O blocks having special high-drive clock buffers, called clock drivers, are distributed throughout the chip. The buffers are connected to clock I/P pads. They drive the clock signals

onto the Global Clock liens described above. The clock lines have been designed for fast propagation time and less skew time.

2.3.2 FPGA Design Flow

The flow for the design using FPGA outlines the whole process of device design, and guarantees that none of the steps is overlooked. Thus, it ensures that we have the best chance of getting back a working prototype that will correctly function in the final system to be designed.

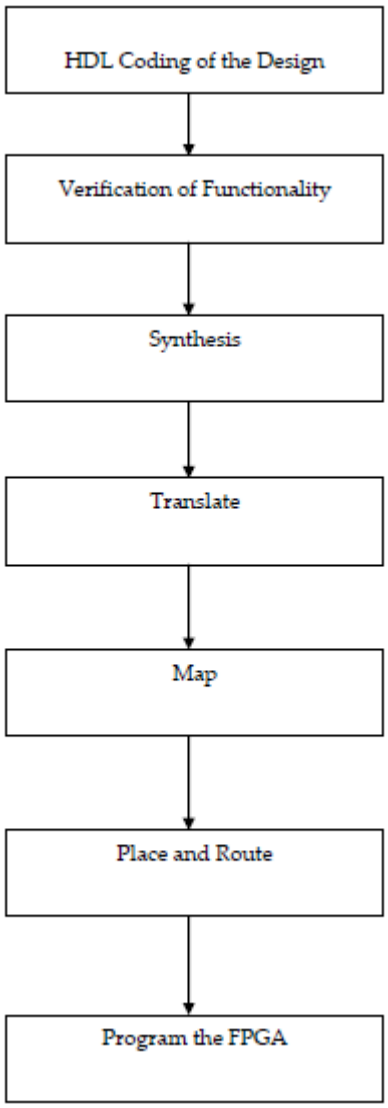


Fig. 7: FPGA Design Flow

2.3.3 Behavioral Simulation[20]

After HDL designing, the code is simulated and its functionality is verified using simulation software, e.g. Xilinx ISE or ModelSim simulator. The code is simulated and the output is tested for the various inputs. If the output values are consistent with the expected values then we proceed further else necessary corrections are made in the code. This is what is known as Behavioral Simulation. Simulation is a continuous process. Small sections of the design should be simulated and verified for functionality before assembling them into a large design. After several iterations of design and simulation the correct functionality is achieved. Once the design and simulation is done then another design review by some other people is done so that nothing is missed and no improper assumption made as far as the output functionality is concerned.

2.3.4 Synthesis of Design

Post the behavioral simulation the design is synthesized. During simulation following takes place:

(i) HDL Compilation

The Xilinx ISE tool compiles all the sub-modules of the main module. If any problem takes place then the syntax of the code must be checked.

(ii) HDL synthesis

Hardware components like Multiplexers, Adders, Subtractors, Counters, Registers, Latches, Comparators, XORs, Tri-State buffers, Decoders are synthesized from the HDL code.

2.3.5 Design Implementation[20]

(i) Translation

The translate process is used to merge all of the input net-lists and the design constraints. It outputs a Xilinx NGD (Native Information and Generic Database) file. The logical design reduced to Xilinx device primitive cells is described by this .ngd file. Here, User Constraints are

defined by assigning the ports in the design to physical elements (e.g. pins, switches, buttons, etc) for the target device as well as specifying timing requirements. This information is stored in a UCF file which can be created using PACE or Constraint Editor.

(ii) Mapping

After the translation process is complete the logical design described in the .ngd file to the components or primitives (Slices/CLBs) present on the .ncd file is mapped onto the target FPGA design. The whole circuit is divided into smaller blocks so that they can be appropriately fit into the FPGA blocks. The mapping is done onto the CLBs and IOBs in accordance with the logic.

(iii) Placing and Routing

After the mapping process the PAR program is used to place the sub-blocks from the map process onto the logic blocks as per the constraints and then connect these blocks. Trade-off between all the constraints is taken into account during the placement and routing process. Place process places the sub-blocks according to logic but does not provide them the physical routing. On running the Route process physical connections between the sub-blocks are made using the switch-matrices.

(iv) Bit file generation

Bit-stream is used to describe the collection of binary data used to program the reconfigurable logic device. The 'Generate Programming File' process is run after the FPGA design has been completely routed. It runs BitGen, the Xilinx bit-stream generation program, to produce a .bit or .isc file for Xilinx device configuration. Using this file the device is configured for the intended design using the JTAG boundary scan method. The working is then verified for different inputs.

(v) Testing

System testing is necessary to ensure that all parts of the system correctly work together after the prototype is mapped onto the system. If the system doesn't work then the problem can be fixed by making some changes in the system or the software. The problems are documented so that on the next revision or production of the chip they are fixed. When the ICs are produced it is necessary to have some sort of burnt-in self-test mechanism such that the system gets tested regularly over a long period of time [22].

2.3.6 Advantages of FPGA [21]

FPGAs have become very popular in the recent years owing to the following advantages that they offer:

Fast prototyping and turn-around time- Prototyping is the defined as the building of an actual circuit to a theoretical design to verify for its working, and to provide a physical platform for debugging the core if it doesn't. Turnaround is the total time between expired between the submission of a process and its completion. On FPGAs interconnects are already present and the designer only needs to fuse these programmable interconnects to get the desired output logic. This reduces the time taken as compared to ASICs or full-custom design.

NRE cost is zero- Non-Recurring Engineering refers to the one-time cost of researching, developing, designing and testing a new product. Since FPGAs are reprogrammable and they can be used without any loss of quality every time, the NRE cost is not present. This significantly reduces the initial cost of manufacturing the ICs since the program can be implemented and tested on FPGAs free of cost.

High-Speed- Since FPGA technology is primarily based on referring to the look-up tables the time taken to execute is much less compared to ASIC technology.

Low cost- FPGA is quite affordable and hence is very designer-friendly. Also the power requirement is much less as the architecture of FPGAs is based upon LUTs.

2.3.7 FPGA Specifications

The FPGA used in this project has the following specifications:

Vendor: Xilinx

Family: Spartan 3E

Family: XC3S500E

Package: FG320

Speed grade: -5

Synthesis Tool: VHDL

Simulator: Xilinx ISE 10.1

Chapter 3

DESIGN AND ARCHITECTURE

3.1 Introduction and Specifications

The Processor design started with determining the number of bits in the instruction set. In this case, it was decided to be 32 bit. The specifications are

- 32 bit instruction set.
- 32 bit registers and data memory.
- 8 bit instruction memory.
- 32 bit address and data bus.
- Huge number of multiplications and additions are usually required, therefore a separate MAC (multiplier accumulator) unit is needed.

A datapath was designed and required individual functional units (Multiplier Accumulator, Program counter, Program and Data memory, Register File etc.) were built. Depending on the signal controlling the different units in the datapath, an FSM was made and a controller was designed. Finally All the Modules were linked together and the Processor was simulated. All the

modules were built using Verilog HDL in Xilinx ISE Design tool. Figure 8 shows broadly the steps in designing the processor.

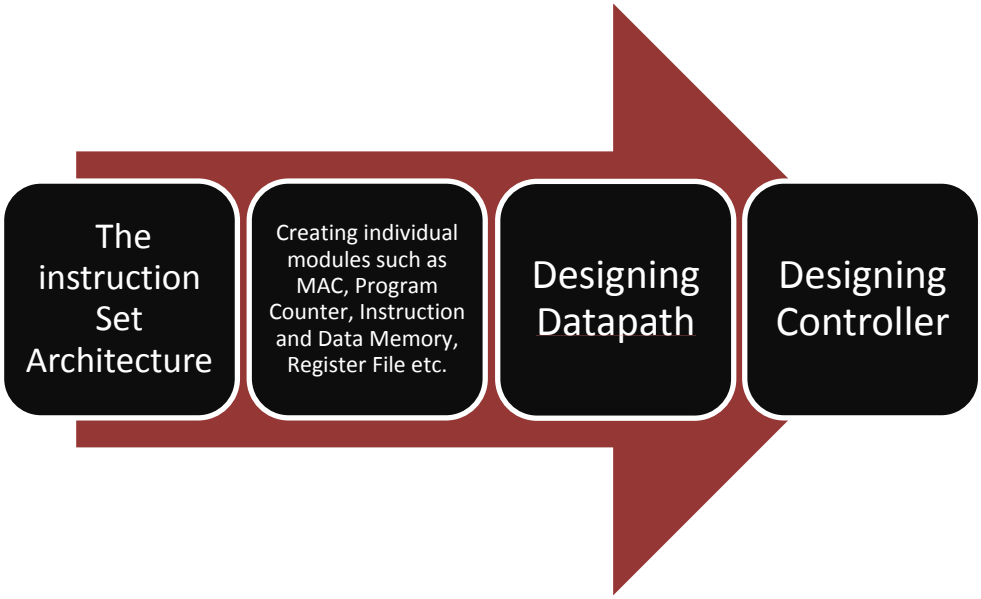


Figure 8: steps in designing the processor

3.2 The Instruction Set Architecture

A a uniform length of the instruction set is always more simpler to implement. For our convenience, in our design, all instructions are confined to 32 bits with the opcode being present in the 31-26 bits. But the rest of the bits will vary in meaning depending on the type of instruction.

The instruction types can be broadly categorized into three groups.

- Register/Memory Addressing
- Arithmetic
- Jump

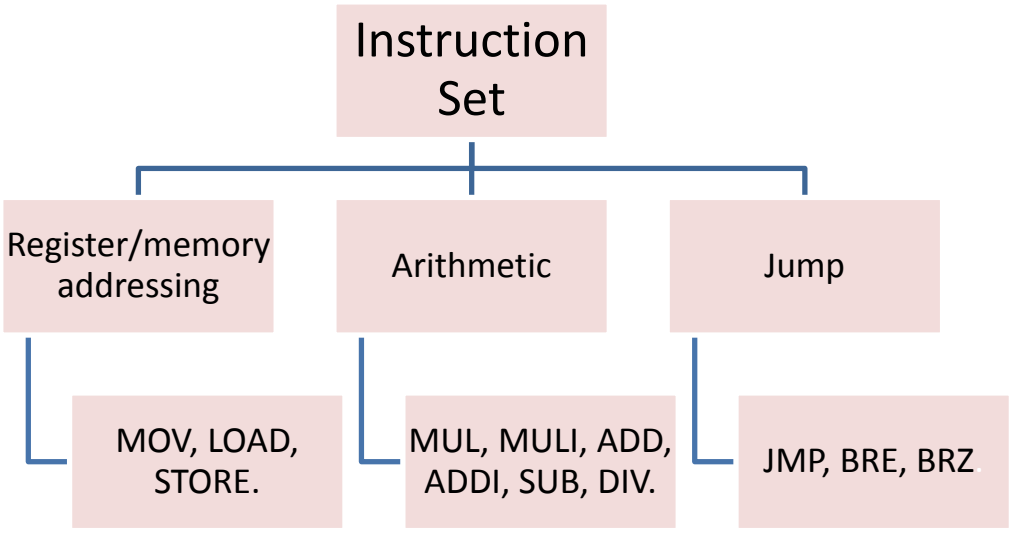


Figure 9: Instruction Set Architecture

In a general instruction, contents of read registers rs and rt (source registers) are added and the sum is stored in destination write register rd.

Opcode (6)	rs (5)	rt (5)	rd (5)		Funct (6)

However, In case of immediate data the instruction format will be

Opcode (6)	rs (5)	rt (5)	Immediate (16)

And, In case of branching Instructions, the format will look like

Opcode (6)	Jump (26)

3.3 The Data Path

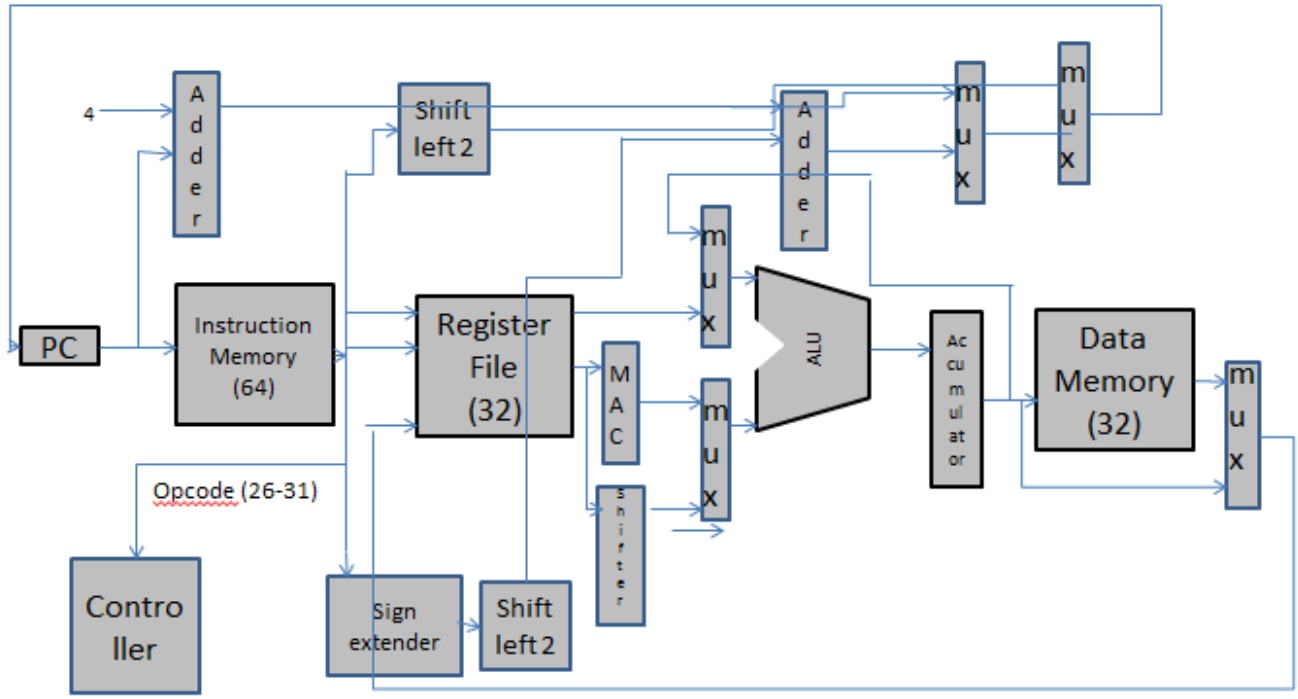


Figure 10: The Data path

In the data path, the execution starts with instruction fetch from the address pointed by the program counter in the instruction memory. The instruction memory sends 32-bit instruction to the register file, of which bits 21-25 are given to read register one, 16-20 to read register 2 and 11-15 to write register, in case of register type instructions. In case of immediate type bits 16-20 are fed to the write register as well. This selection is done with the help of a 2:1 multiplexer. From the register file two 32 bit data are generated by accessing the address in data memory that

are fed to the ALU and the MAC and shifter through muxes as shown in the figure where the required operations are performed and the output is stored in the data memory. In this processor, a MAC with a carry look ahead multiplier is used for faster computations. The 8 bit program counter is incremented by 4 address locations for the execution of the next instruction.

In case of immediate instructions, bits 0-15 are extended to 32 bits with the help of a sign extender. The ALU selects this with the help of a 2:1 mux as shown in the figure. If the instruction is of LW or SW type, then the address location generated by the ALU is accessed in the memory and required operation is performed. For branching, If the inputs are equal then the signal zero is set. This and branch signal generated from the control unit together forms the selection bit for the mux that that decides the address location to be sent to the program counter. The address is generated by adding the 32 bits that are generated by shifting the output of sign extender by two bits to the current address location pointed by the program counter.

In case of Jump instruction, the bits 0-25 are passed through a sign extender before shifting left by two, and concatenated with four bits from the PC to give the 32 bit address, that is again fed to the PC with the help of a mux.

All the control and selection signals are generated in the control unit.

3.4 Control Unit

The control signals are generated from the instruction opcode bits 31-26. Fig 11 shows the Control FSM designed to make the combinational circuit making the control unit.

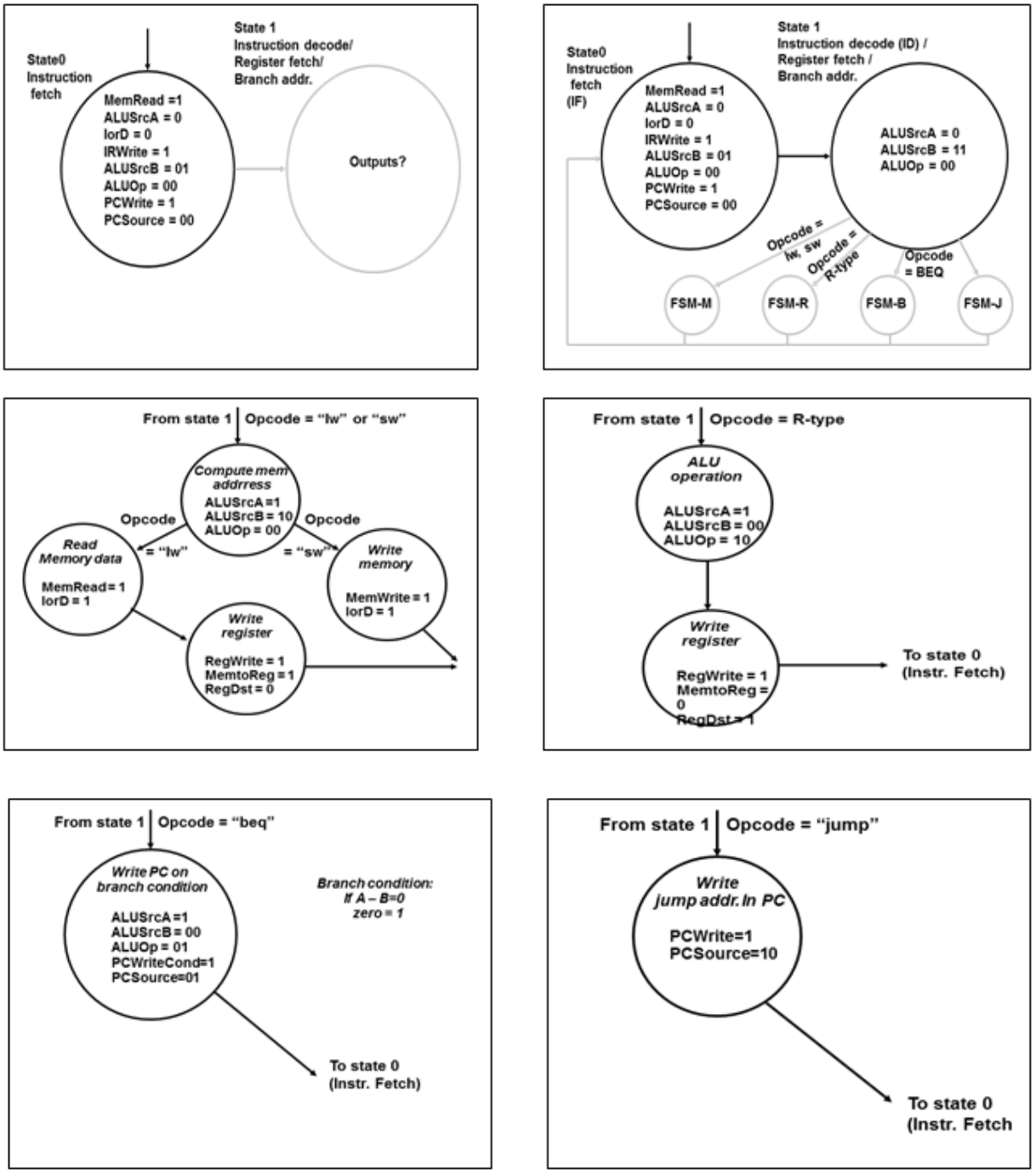


Fig 11: FSM for controller design

Thus, For different types of instructions, we designate different states in the FSM with different states of the control signals. This Control FSM takes multiple clock cycles to execute an FSM. It can be simplified further into the given Combinational circuit below, that can work for instructions that are required to be executed within single clock cycle.

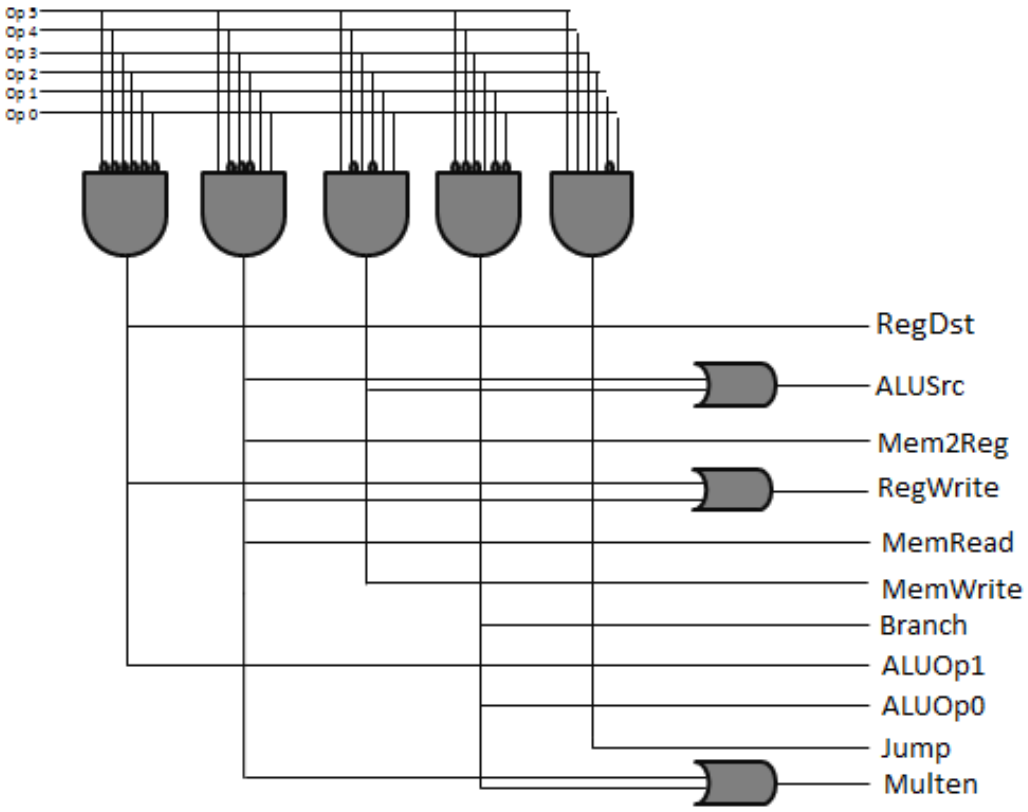


Fig 12: Combinational circuit for the controller

The controller thus designed provided the datapath with different signals to operate the muxes and various functional units. The RegDst decides the register addresses that are fed to the register file. The ALUSrc decides whether the data goes to the ALU directly from the register or from the accumulator, whereas Multen enables the MAC operation. The signal Mem2reg sends data from the data memory to the register through a mux. Regwrite, MemRead and MemWrite enables reading from and writing into the registers and memory correspondingly. Branch and Jump

operations enable Unconditional and Conditional jump respectively. ALUOp1 and ALUOp0 decides what arithmetic operations to be carried out by the ALU.

After programming all the individual modules and controller unit using Verilog HDL, final linking was done according to the data path designed earlier and simulated using Xilinx ISE tool. A simple case of Convolution was taken up for this purpose as explained in the next chapter and the results were verified.

3.5 List Of Instructions

Table 4: List of Instructions

MOV	Moves Data From Register to Register
MOVI	Moves Immediate data to register
LOAD	Loads data from memory to register
STORE	Stores data value from register to memory
ADD	Addition
ADDI	Addition with immediate data
SUB	Subtraction
SUBI	Subtraction with immediate data
MUL	Multiplication
MULI	Multiplication with immediate data
DIV	Division
MACC	Multiply and accumulate
JMP	Jump
JMPE	Jump if equal
JMPC	Jump if carry

Chapter 4

RESULTS AND SIMULATIONS

Fig 13 shows the input and output signals to the control unit. The block take sin the 6-bit opcode as input and generates the control signals as shown in the figure.

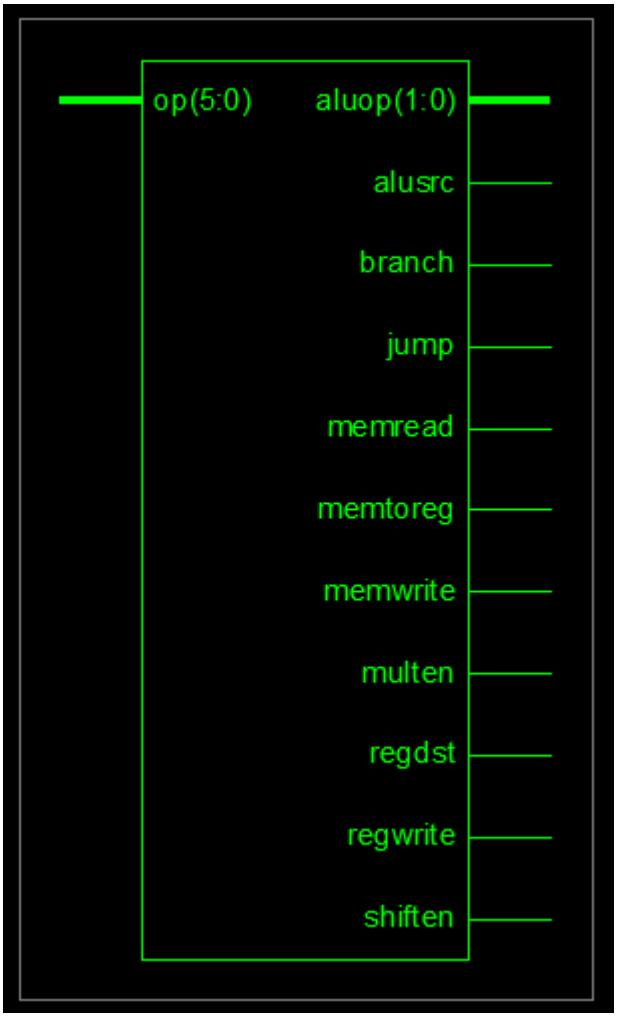


Fig 13: RTL schematic of the control unit

In Figure below, the RTL schematic of the processor is shown. The clock input is taken by the DSP block and 25 outputs are shown in the simulation results, including the input and output registers and all the control signals.

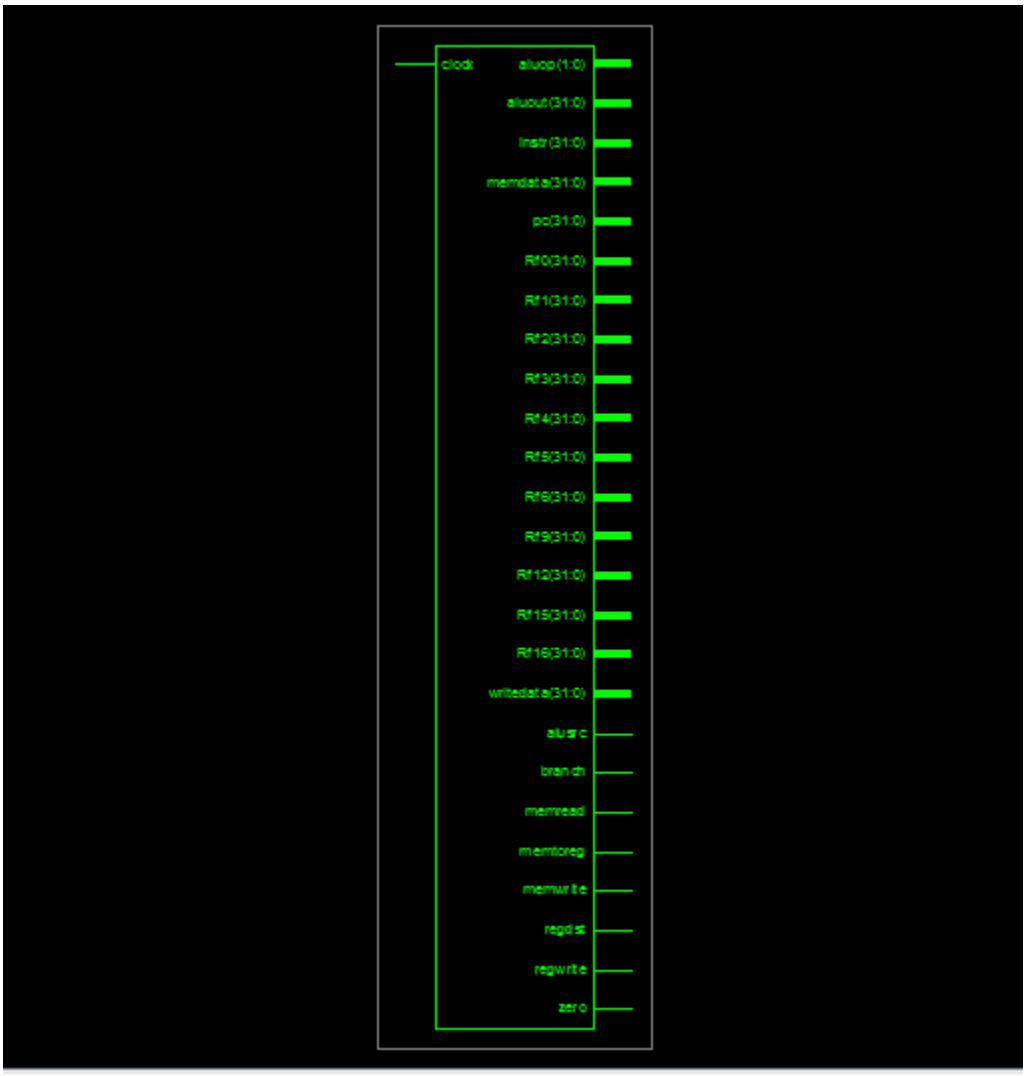


Fig 14: RTL schematic of the DSP

The image is magnified to give the following schematic figure. All the input and output signals are shown explicitly here.

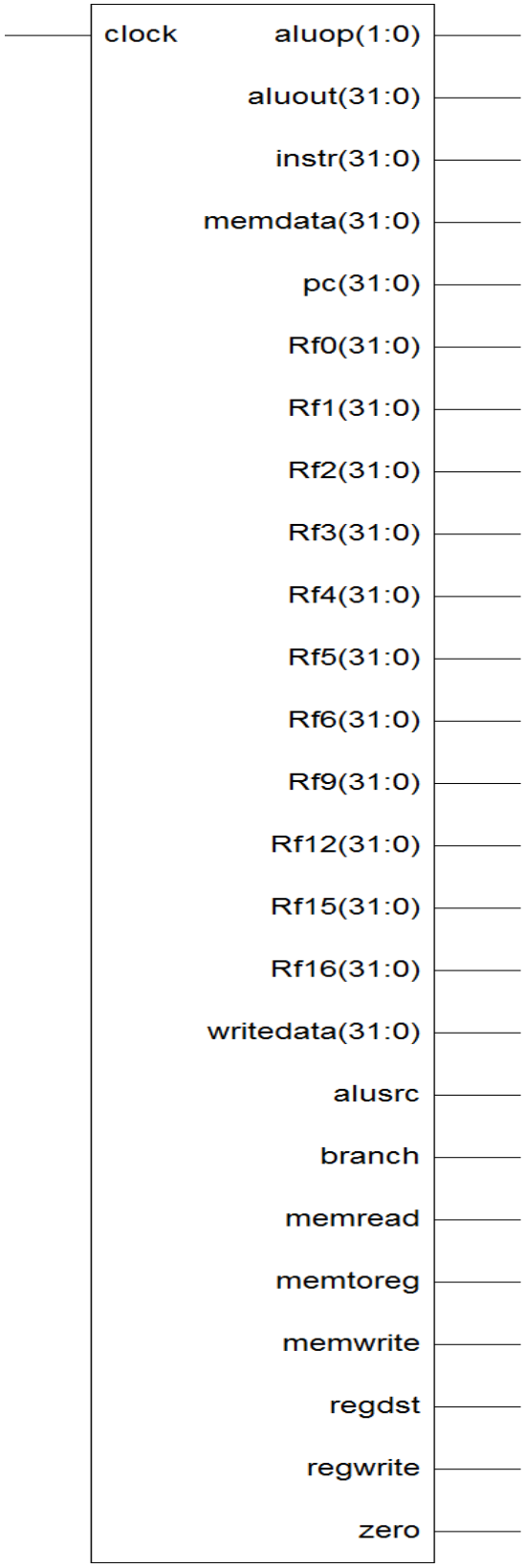


Fig 15: A magnified RTL schematic of the DSP block

For the purpose of simulation, a case of convolution was taken up.

For the input sequences were {1,2,3} and {4,5,6} the expected output is {4,13,28,27,18}. The first sequence of inputs was fed to registers {Rf0, Rf1, Rf2} and the second sequence was fed to {Rf3, Rf4, Rf5}. Registers {Rf6, Rf9, Rf12, Rf15, Rf16} were designated to be the output registers.

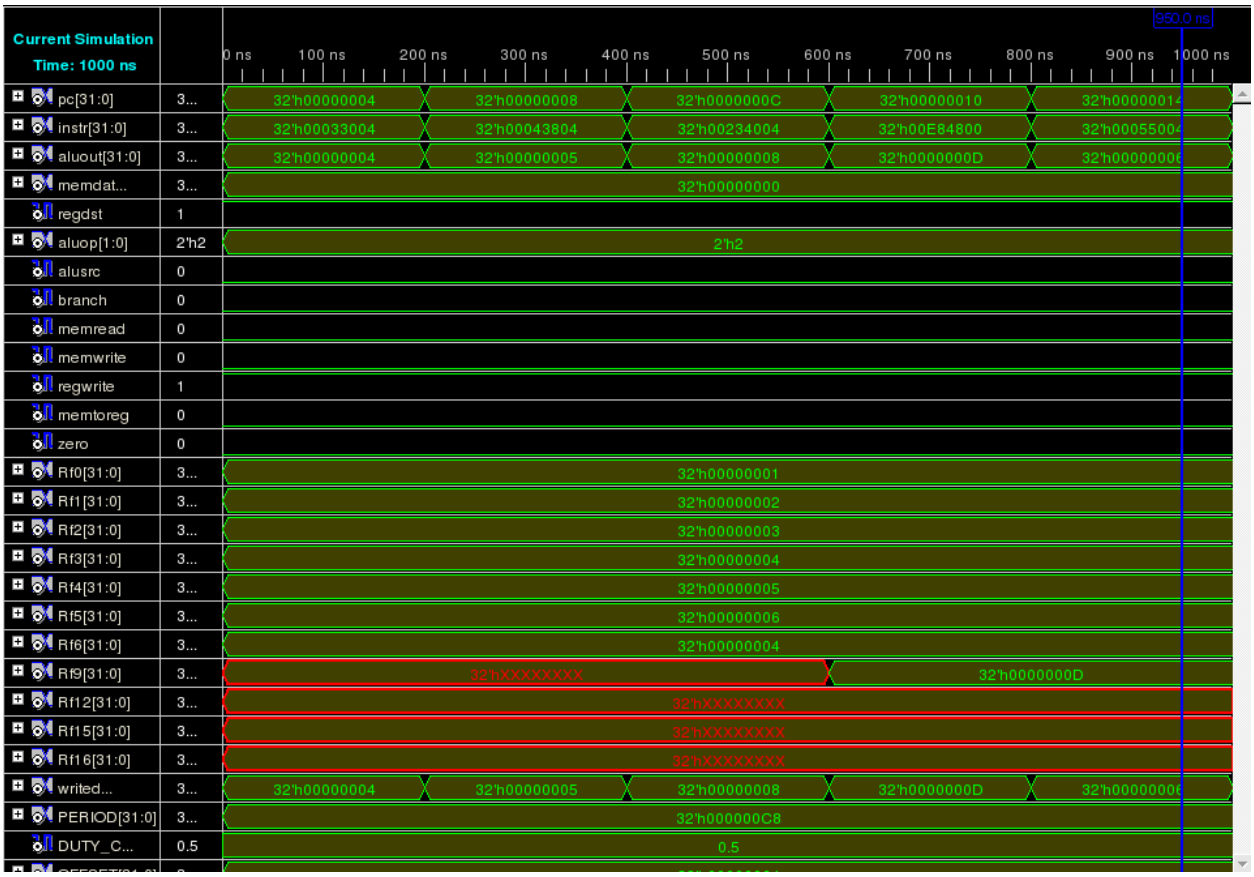


Fig 16 (a): Simulation result after the clock is initiated

As shown in fig 16(a), after execution of the first set of multiplication, result 4 is outputted to Rf6. After a certain period of time, Register Rf9 is outputted 13, or 0D H (Hexadecimal). Similarly, all the values were obtained and the final result is shown in figure 16(b).

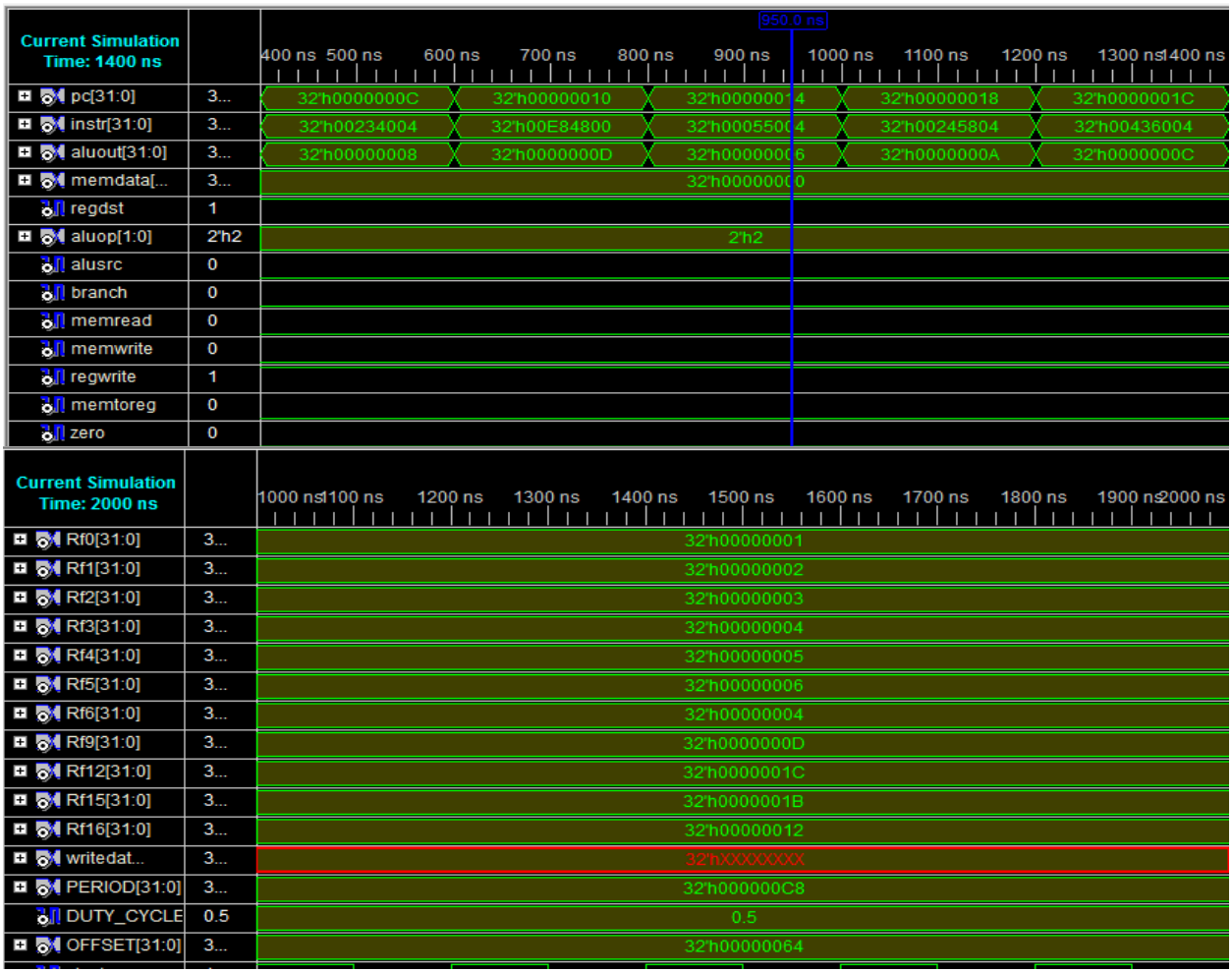


Fig 16(b) : simulation result after all the output values are stored

The final result is shown in table 5. This is found to be consistent with calculated result of the convolution.

Table 5: Convolution result

Rf6	04 (04H)
Rf9	13(0DH)
Rf12	28(1CH)
Rf15	27(1BH)
Rf16	18(12H)

The final synthesis report for the above simulation has been provided below:

```
=========================================================================
*                            Final Report                               *
=========================================================================



Final Results
RTL Top Level Output File Name     : sc.ngr
Top Level Output File Name         : sc
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 523

Cell Usage :
# BELS                             : 90
#      GND                         : 1
#      INV                         : 1
#      LUT1                        : 29
#      MUXCY                       : 29
#      VCC                         : 1
#      XORCY                       : 29
# FlipFlops/Latches                : 30
#      FD                          : 30
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 522
#      OBUF                        : 522
=========================================================================
```

```
Device utilization summary:
---------------------------

Selected Device : 3s500efg320-4

 Number of Slices:                       16  out of   4656     0%
 Number of Slice Flip Flops:             30  out of   9312     0%
 Number of 4 input LUTs:                 30  out of   9312     0%
 Number of IOs:                         523
 Number of bonded IOBs:                 523  out of    232   225% (*)
 Number of GCLKs:                         1  out of     24     4%
```

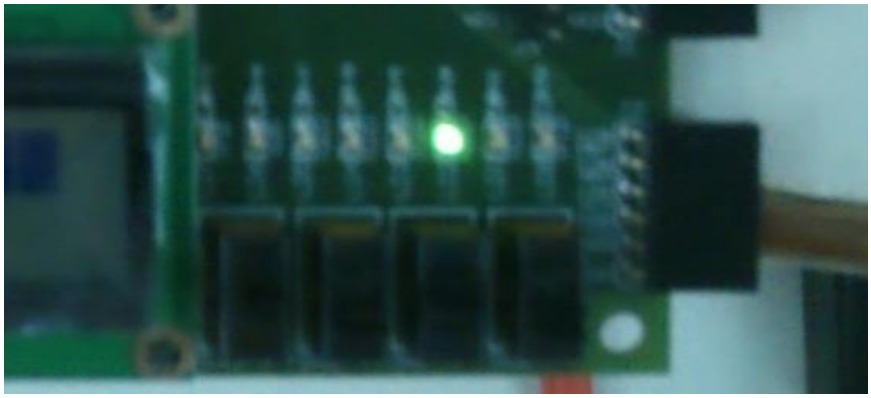
```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 5.086ns (Maximum Frequency: 196.618MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 4.310ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Fig 17: Synthesis Report

FPGA Implementation

For implementing the above processor in FPGA, SPARTAN 3E board was used. However, the board has certain limitations. The Number of input ports provided is restricted to 4, each of 1 bit. There are 8 output LEDs provided, each representing a bit. Therefore to implement the processor, a compromise was made in the number of input and their bit size. For the purpose of convolution, two arrays of size two were taken with the data size being 1 bit. The program was dumped into the board after implementing the design and Routing.

In the first case, The arrays were {1,0} and {1,0}. The output was found to be {1,0,0} which is consistent with expected values.



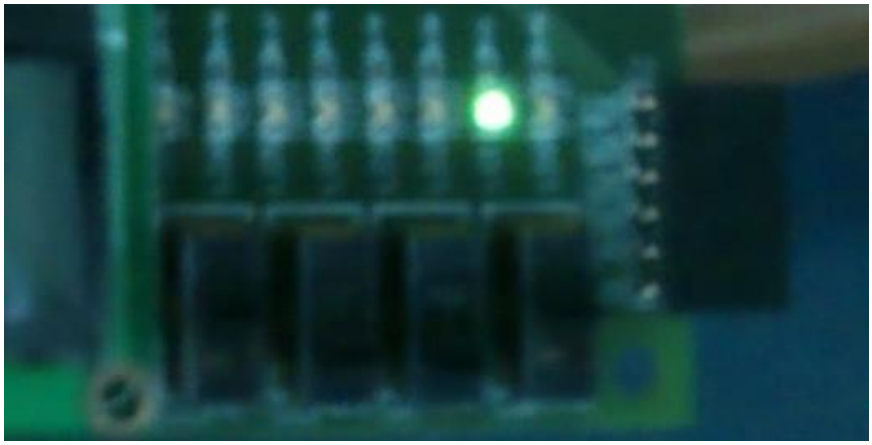


Fig 18(b): FPGA result for sequence {1,0};{0,1}

In the second case, The arrays were {1,0} and {0,1}. The output was found to be {0,1,0} which is consistent with expected values. Thus, The DSP was found to work successfully with the sets of input data provided.

Chapter 5

CONCLUSIONS AND FUTURE WORK

The DSP was designed using Xilinx ISE tool in Verilog HDL and it was found to work successfully with given inputs. From the synthesis report it was seen that the maximum output required time after clock is 4.310ns. The clock frequency was 196.618MHz.

In this design, a total of 32 registers are provided which can be modified to use as temporary registers, accumulators etc. All the jump operations were found to perform correctly, as were the arithmetic operations. Implementing this processor on a platform like FPGA gives us a powerful mechanism of implementing complex computations on a platform that provides a lot of resources and flexibility at a relatively lesser cost.

Finally the DSP was implemented on a Spartan 3E FPGA kit. The output values were found to be consistent with the actual values. The device utilization summary showed that minimum resources were consumed.

Future Scope

Although this project primarily deals with the design of a fixed-point processor The 32 bit Instruction Set allows enough flexibility to build a floating point DSP based on the current one. Also more instructions can be added in the existing structure itself, to customize it according to user requirements. The processor uses a typical carry look ahead multiplier but the speed can be improved by using any fast MAC or ALU that occupies lesser space.

References

[1] B Venkataramani M bhaskar, "Digital Signal Processors architecture programming and application"

[2] Digital control applications with TI TMS 320 processors, Texas Instruments

[3] Subra Ganesan,"Digital Signal Processing Design Using TMS 320C5X Processor"

[4] M.E. Angoletta, "Digital signal processor fundamentals and system design," CERN, Geneva, Switzerland

[5] Steven W. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing"

[6] J. G. Proakis and D. G. Manolakis , "Digital Signal Processing: Principles, Algorithms, and Applications"

[7] E.A. Lee, Programmable DSP Architectures: Part II, IEEE ASSP Mag., January 1989, pp. 4-14.

[8] TMS320C621x/C671x DSP Two-Level Internal Memory Reference Guide, Texas Instruments Literature Number SPRU609A, November 2003.

[9] TMS320C620x/C670x DSP Program and Data Memory Controller/Direct Memory Access

[10] (DMA) Controller - Reference Guide, Texas Instruments Literature Number SPRU234, July 2003.

[11] Extended-Precision Fixed-Point Arithmetic On The Blackfin Processor Platform, Analog Devices Engineer-to-Engineer Note EE-186, May 2003.

[12] TMS320C6000 DSP Inter-Integrated Circuit (I2C) Module – Reference Guide, Texas Instruments Literature Number SPRU581A, October 2003.

[13] TMS320C6000 Peripherals – Reference Guide, Texas Instruments Literature Number SPRU109D, February 2001.

[14] D. Dahnoun, Bootloader, Texas Instruments University Program, Chapter 9, 2004.

[15] D. Dart, DSP/BIOS Technical Overview, Texas Instruments Application Report SPRA780, August 2001.

[16] TMS320C6000 Optimizing Compiler – User's Guide, Texas Instruments Literature Number SPRU187L, May 2004.

[17] TMS320C6000 Assembly Language Tools – User's Guide, Texas Instruments Literature Number SPRU186N, April 2004.

[18] Rewind User's Guide, Texas Instruments Literature Number SPRU713A, April 2005.

[19] TMS320C6000 Instruction Set Simulator – Technical Reference, Texas Instruments Literature Number SPRU600F, April 2005.

[20] C. Brokish, Emulation Fundamentals for TIs DSP Solutions, Texas Instruments Application Report SPRA439C, October 2005.

[21] HUTCHINGS, B. L. AND NELSON, B. E., 2001. Gigaop DSP on FPGA. International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

[22] Chapman K. 1996. Constant Coefficient Multipliers for the XC4000E. Xilinx Application Note, www.xilinx.com.