ECEN438

Advanced Computer Architecture Project

Modified Single Cycle Architecture with DSP Unit

Team Members:

Mohaned Asker – 211000845

Mohamed Alaa – 211001955

Marcelino Emad – 211001330

Sara Abdelnasser – 211001619

Hana Nasef – 211000128

Table of Contents

1. Abstract

This project extends a textbook single-cycle MIPS-like CPU by integrating a dedicated DSP unit to implement a 3rd-order moving-average FIR filter. The modified architecture introduces a new DSP instruction (opcode 000001) and a custom R-type instruction format (replacing the shamt field with a 5-bit dsp field) to control the filter unit. Thirty-two external 32-bit input samples (e.g. a sine wave) are preloaded into the register file for processing. In simulation, the DSP unit correctly computes the moving-average output, and standard MIPS instructions (loads, stores, arithmetic) operate normally. Waveform analysis confirms that the filter output matches the expected averaged waveform and that register and memory updates follow the correct MIPS semantics. These results demonstrate that the design meets the project goals of correct DSP extension and MIPS compatibility.

2. Introduction:

In this project, we designed a MIPS single-cycle architecture using VIVADO software and System Verilog. This architecture supports R, I, and J type instructions such as add, OR, load, store, beq, and jump. The basic components include a Program Counter (PC), Instruction Memory, Register File (RF), Arithmetic Logic Unit (ALU), Data Memory, adders, multiplexers, a sign extend unit, shift left units, and a control unit. To expand its functionality, we integrated a Digital Signal Processing (DSP) block into the MIPS architecture and introduced a new DSP instruction, modeled as an R-type instruction. This instruction processes data from a register using the DSP block and stores the result back in a register. We used a 3rd-order Finite Impulse Response (FIR) filter designed in MATLAB to generate 6-bit coefficients and 16-bit input samples. The DSP block processes these input samples, and the results are stored in 16 registers. This paper presents the enhanced MIPS architecture with DSP integration, detailing the design and implementation of the DSB instruction. The results demonstrate the architecture's ability to efficiently perform moving average signal processing, extending the capabilities of the traditional MIPS processor.

3. Literature Review

Single-Cycle CPU Architectures. A single-cycle CPU executes every instruction in one clock cycle. Consequently, all instructions take the same amount of time, and the cycle period must accommodate the slowest (longest-latency) instruction. This design simplifies control logic but suffers from inefficiency: hardware units often sit idle for much of the cycle, and overall clock speed is limited by the most complex instruction. In practice, this means that although a single-cycle design is conceptually simple, it typically has a longer cycle time and lower performance than pipelined or multi-cycle designs. Nevertheless, single-cycle processors are widely used in educational settings and small embedded cores because of their simplicity. As described in standard MIPS references, the single-cycle datapath includes separate instruction and data memories, an ALU, registers, and multiplexers; it uses a universal cycle time equal to the longest-path delay (for example, a load instruction's memory access). DSP Unit Integration. Digital Signal Processing (DSP) extensions to general-purpose CPUs add specialized instructions and hardware to accelerate common DSP tasks. For example, the MIPS32 DSP Applications-Specific Extension (ASE) introduces new instructions (mac, dot-product, saturating arithmetic, etc.) and additional registers to optimize signal processing and multimedia operations. These extensions are typically backward-compatible: the original ISA remains valid, and the new instructions use reserved opcodes or funct fields. More generally, DSP architectures often employ Harvard or modified-Harvard memory structures to fetch instructions and data simultaneously, which is especially beneficial for filter and convolution loops. In such designs, during a single multiply-accumulate (MAC) operation, the processor can fetch the next instruction from program memory and a filter coefficient from data memory in the same cycle. This parallelism (dual instruction/data fetch) and the inclusion of multiple MAC units or accumulators allow DSP hardware to achieve high throughput. In our project, the DSP unit is integrated into a standard single-cycle MIPS datapath: a new DSP instruction (opcode 000001) triggers the filter hardware, and the datapath is modified to route operands into this unit. The rest of the CPU (register file, ALU, memory, etc.) remains unchanged for general instructions. FIR Filter Implementations. Finite Impulse Response (FIR) filters produce each output sample as a weighted sum of a finite number of recent input samples. A moving-average filter is a simple FIR filter whose coefficients are all equal (e.g. [1/4,1/4,1/4,1/4] for a 4-tap average). Such filters are

popular for noise reduction because they “operate by averaging a number of points from the input signal to produce each point in the output signal”. In hardware, FIR filters are commonly implemented using multiply-accumulate (MAC) structures. A basic approach is a single-multiplier MACC (multiply-and-accumulate) filter: input samples and coefficients are sequentially multiplied and summed in an accumulator. This sequential MAC trades off hardware cost for throughput (one multiply-add per cycle). Alternatively, parallel or systolic implementations use multiple multipliers and adders to compute all products in one cycle, at higher area cost. For our 3rd-order moving-average filter, the hardware simply accumulates the current and previous samples and divides by the number of taps. For example, if “3rd-order” implies four samples, the output is

$$(x[n] + x[n-1] + x[n-2] + x[n-3]) / 4$$

This accumulator-and-shift structure fits naturally into a CPU datapath with a dedicated DSP unit. In summary, the project’s DSP filter follows established principles: it is a fixed-coefficient FIR realized via MAC operations, triggered by a special instruction, and is compatible with the CPU’s control flow and data paths.

4. Methodology:

The implementation of the modified single cycle architecture with a dsp unit is divided into two phases. Phase one consists of implementing the basic single cycle architecture without any modifications and the single cycle architecture that our design will be based on is MIPS. Meanwhile, phase two consists of modifying the implementation from phase one to accommodate a DSP unit which will support a moving average FIR filter operation.

4.1. Implementation of the MIPS Single Cycle Architecture.

To implement the MIPS single cycle architecture, we will further divide the process into 2 phases, as shown in figure 1. Phase one will consist of implementing the actual MIPS processor which will contain the control unit and the data path. Phase two will consist of implementing the external memory, instruction and data memories, and connecting it to the mips processor.

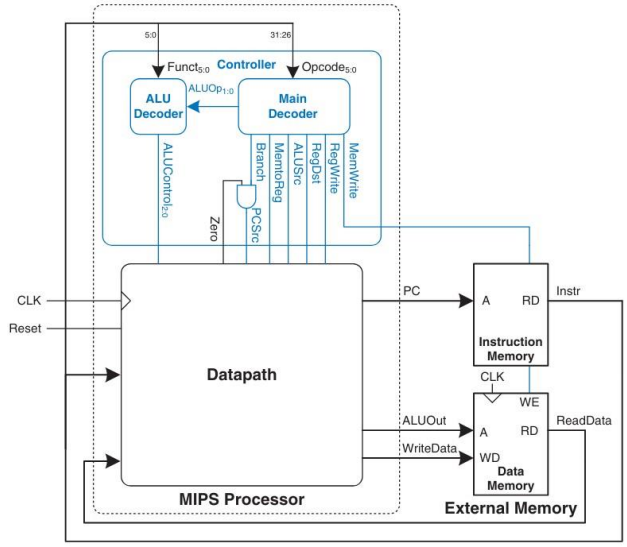

Figure 1: MIPS Top View

4.1.1. MIPS:

The MIPS single cycle processor consists of a control unit and a data path, as shown in figure 2. The control unit is responsible for providing the necessary control signals to be used by the data path and memory to execute the needed instructions. The data path is the pathway that the data takes to execute an instruction.

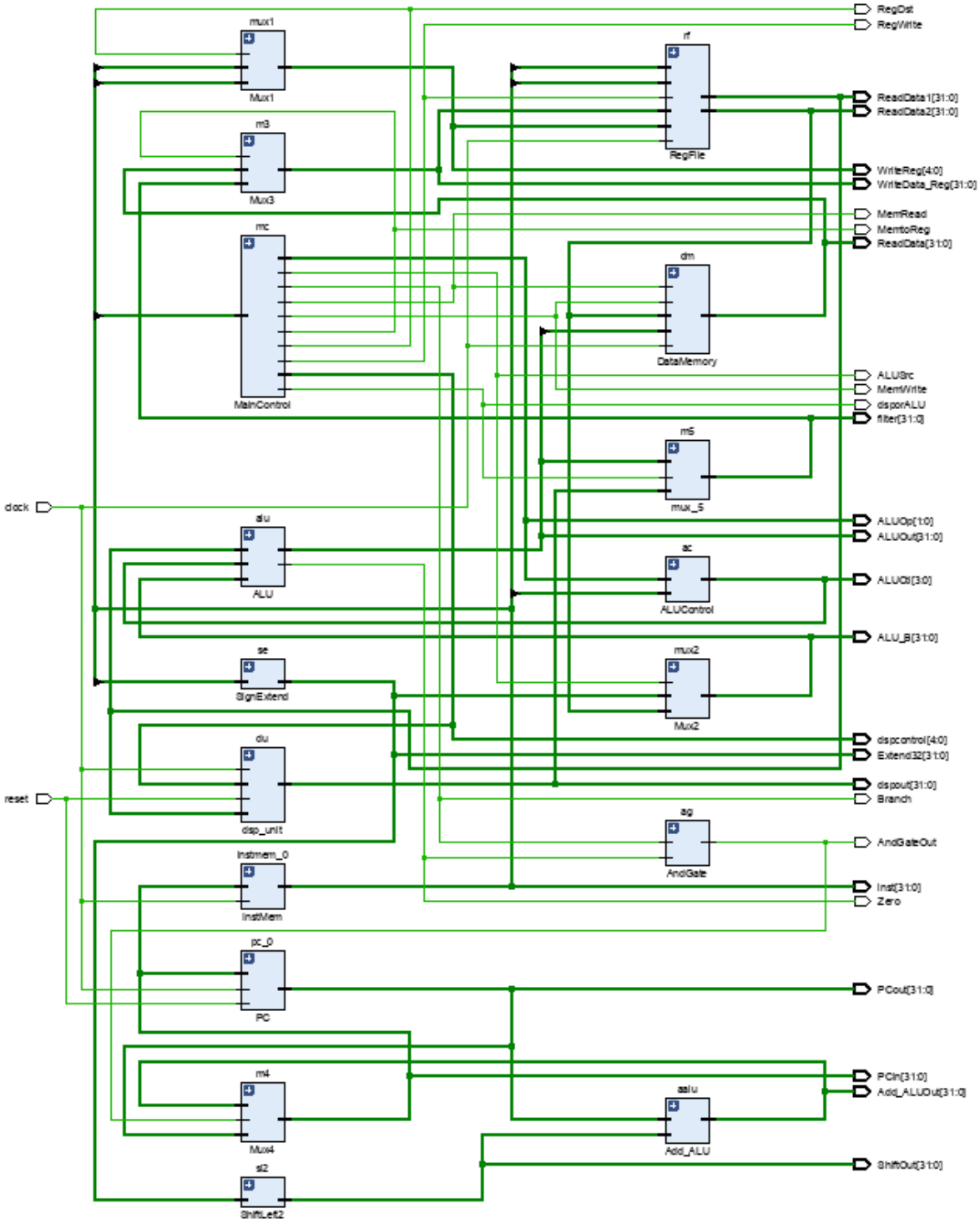

Figure 2: MIPS Processor

4.1.1.1. Datapath:

The data path of a MIPS single cycle processor consists of some basic modules. These modules are the program counter (PC), adder, 2 32-bits input MUX, 2 16-bits input MUX, sign

extend, shift left 2, and an ALU. The data path that we will implement will support r-type, sw, lw, beq, addi, and jump instructions. **A. PC:**

The program counter is a 32-bit register that holds the address of the next instruction in the instruction memory. The PC is increased by 4 bytes after each instruction fetch using an adder, as shown in figure 2.

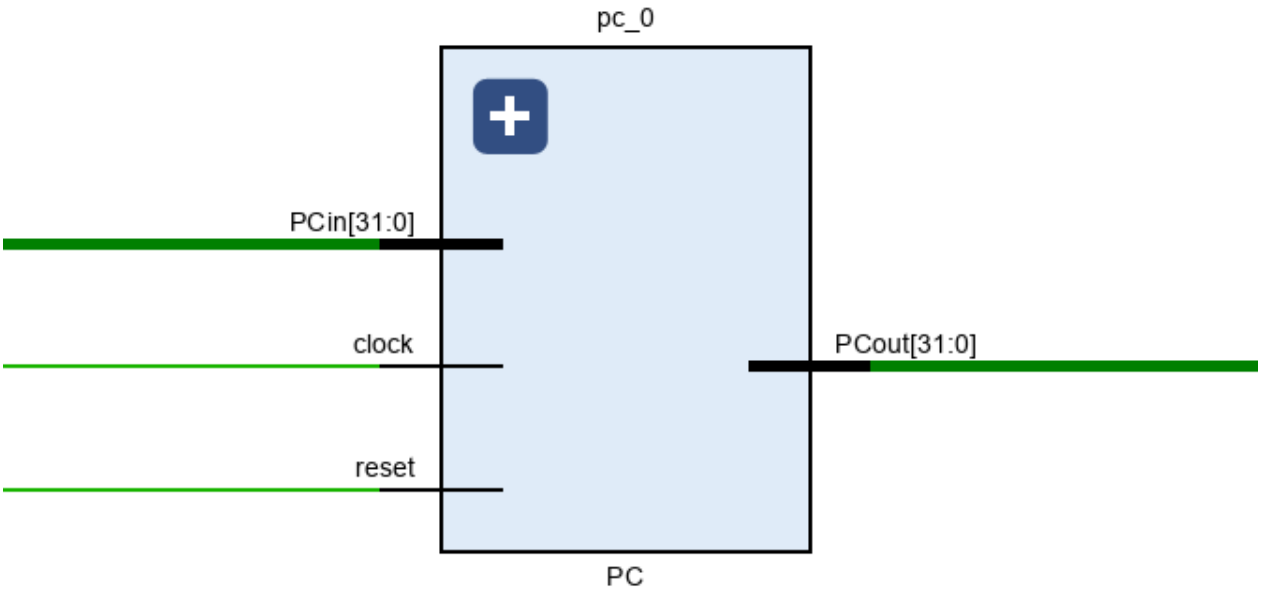

Figure 3: PC RTL Schematic

B. RF:

The RF holds 32 32-bits general purpose register. Each register can be used in an instruction depending on the input addresses and control signals.

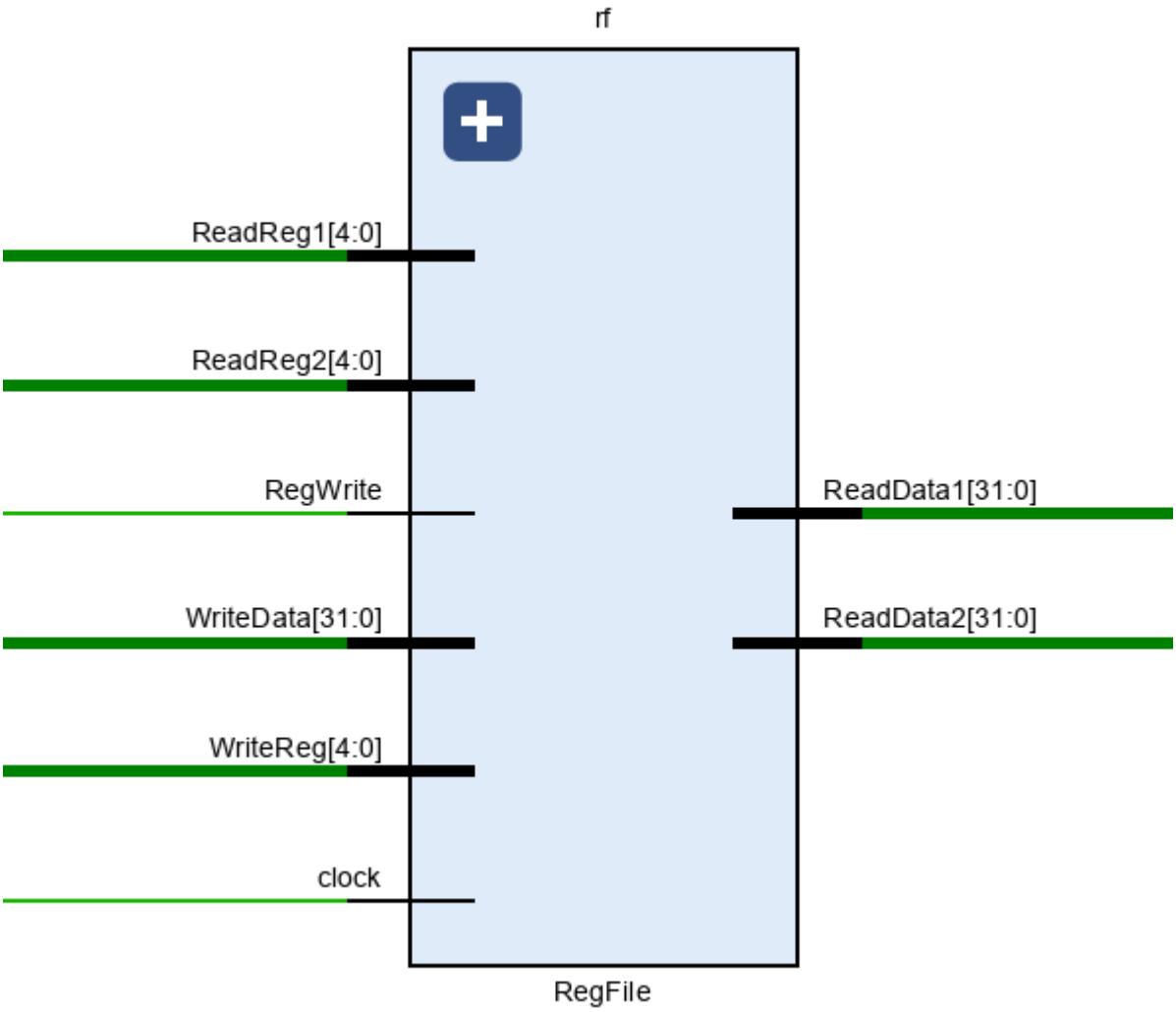

Figure 4: RF RTL Schematic

C. Adder:

The adder is used in the architecture to add 2 32-bits values together. As shown in figure 2, we can see that adders are being used to increase the value of the PC by 4 and to calculate the address of a branch instruction.

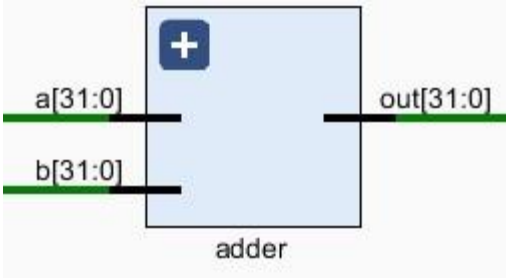

Figure 5: Adder RTL Schematic

D. MUXs:

In the architecture multiplexers are used to select between different data sources. As shown in figure 2, we can see that MUXs are being used to select between the possible PC values, to select the read register 3 (RD3) register address and is used to select the SrcB of the ALU.

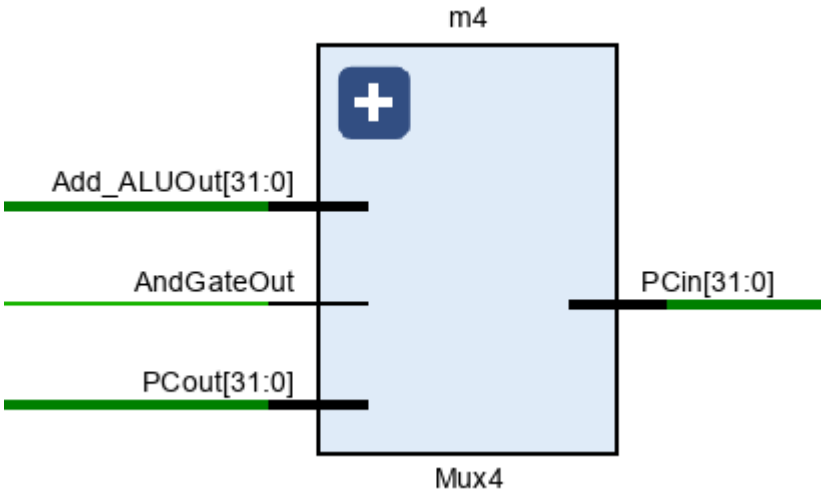

Figure 6: MUXs RTL Schematic

E. Sign Extend:

The sign extend module is used to extend the sign bit of a 16-bits signed value for it become 32-bits. As shown in figure 2, the sign extend module is used to extend a 16-bits immediate value to 32-bits which is used in i-type instructions.

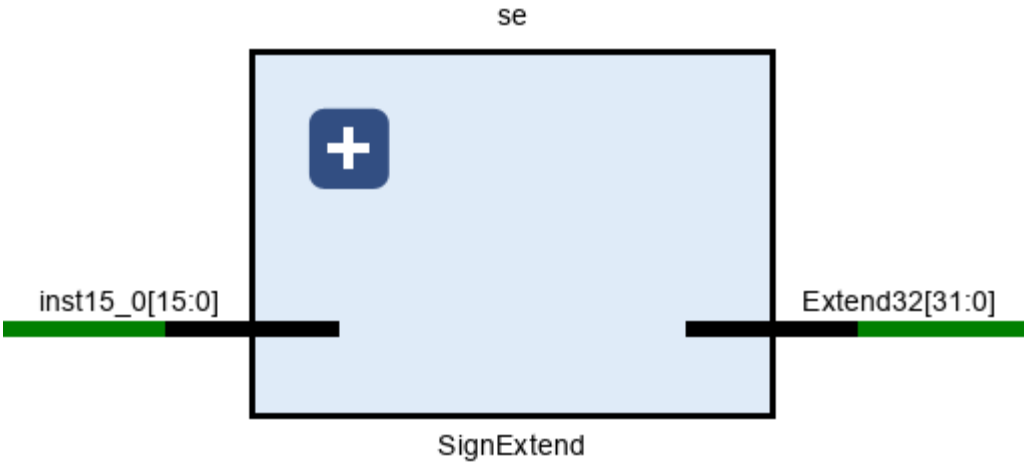

Figure 7: Sign Extend RTL Schematic

F. Shift Left 2:

The shift left 2 module is used to shift a binary value 2 bits to the left and filling zeros on the right. As shown in figure 2, the shift left 2 module is used in the process of finding the branch address.

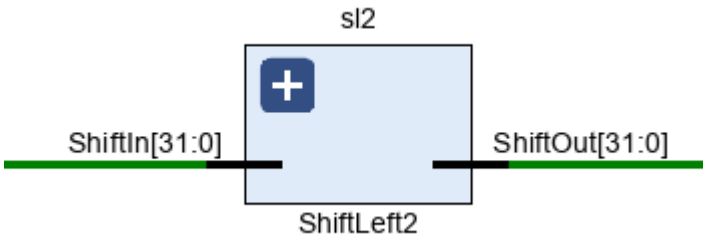

Figure 8: Shift Left 2 RTL Schematic

G. ALU:

The ALU module is a crucial module which is used to perform arithmetic and logical operations on the input data. The ALU in our implementation of the MIPS single cycle architecture can do 5 different operations. The operations are subtraction, addition, and, or, and set less than.

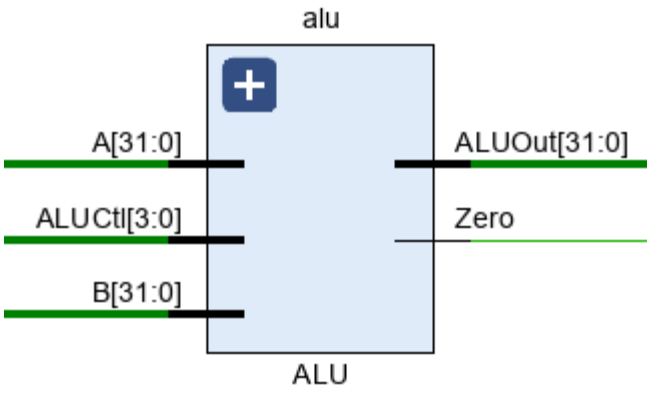

Figure 9: ALU RTL Schematic

4.1.1.2. Control Unit:

A MIPS single cycle architecture's control unit consists of 2 decoders, as shown in figure 3. The first decoder is called the "Main Decoder" which is responsible for generating the control signals for the different modules in the data path except the ALU unit. The ALU decoder is responsible for generating the control signals responsible for controlling what operation will be conducted by the ALU.

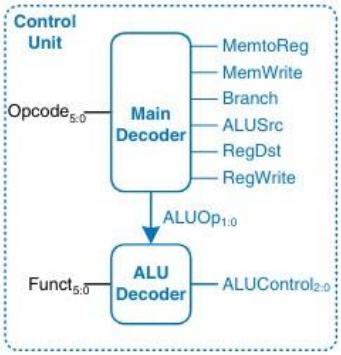

Figure 10: MIPS Control Unit

A. Main Decoder:

The main decoder works by taking the opcode from an instruction and determining the necessary control signals accordingly using a truth table. In table 1 is our main decoder's truth table.

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump
R-type	000000	1	1	0	0	0	0	10	0

lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
addi	001000	1	0	1	0	0	0	00	0
j	000010	0	X	X	X	0	X	XX	1

Table 1: Main Decoder's Truth Table

B. ALU Decoder:

The ALU decoder works by taking the ALUOp generated by the main decoder to determine which operation and instruction type it is using a truth table, in table 2 is our ALU decoder's truth table. After determining the operation and instruction it will send a control signal (ALUControl) to the ALU to tell it which operation to do.

Op	Funct	ALUControl
00	XXXXXX	010 (add)
01	XXXXXX	110 (sub)
1X	100000	010 (add)
1X	100010	110 (sub)
1X	100100	000 (and)
1X	100101	001 (or)
1X	101010	111 (slt)

Table 2: ALU Decoder

C. Control Unit RTL Schematics:

After implementing the main decoder and the ALU decoder and connecting them we can view the RTL schematic of the control unit. In figure 12, we can view the top view of the control unit while figure 13 and 14 we can view the inside of the decoders.

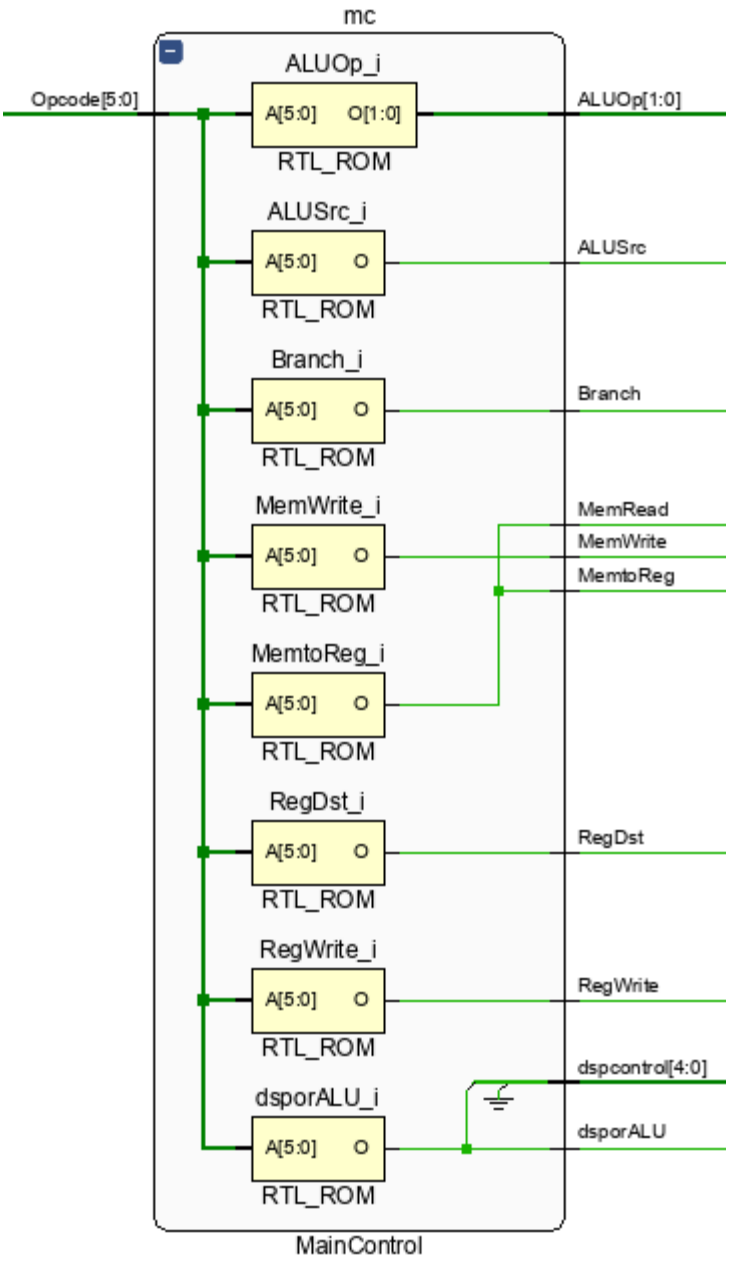

Figure 11: Control Unit Top RTL Schematic

4.1.1.3. MIPS Schematic:

After finishing the implementation of the data path and the control unit we connected them together for the data path to be able to receive the control signals from the control unit.

4.1.2. External Memory:

4.1.2.1. Instruction Memory:

The instruction memory module is responsible for holding the instructions that the processor will execute. The instruction memory takes an input the address of the instruction to be fetched and gives an output of a 32-bit instruction. In our design the instruction memory is 4KB which will have 1024 addressable memory locations.

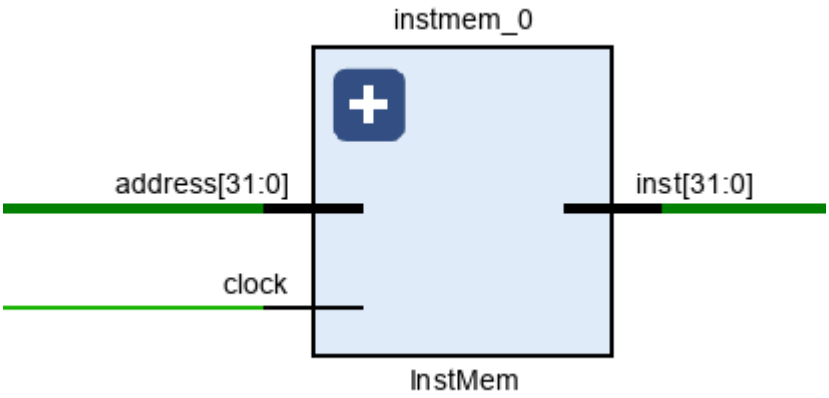

Figure 12: Instruction Memory RTL Schematic

4.1.2.2. Data Memory:

The data memory module is responsible for storing data. Data memory can be accessed using load and store instructions. The data memory takes an input of the address location the instruction wants to read/write from/to and the data to be written and outputs the data to be read. In our design the data memory is 4KB which will have 1024 addressable memory locations similar to the instruction memory.

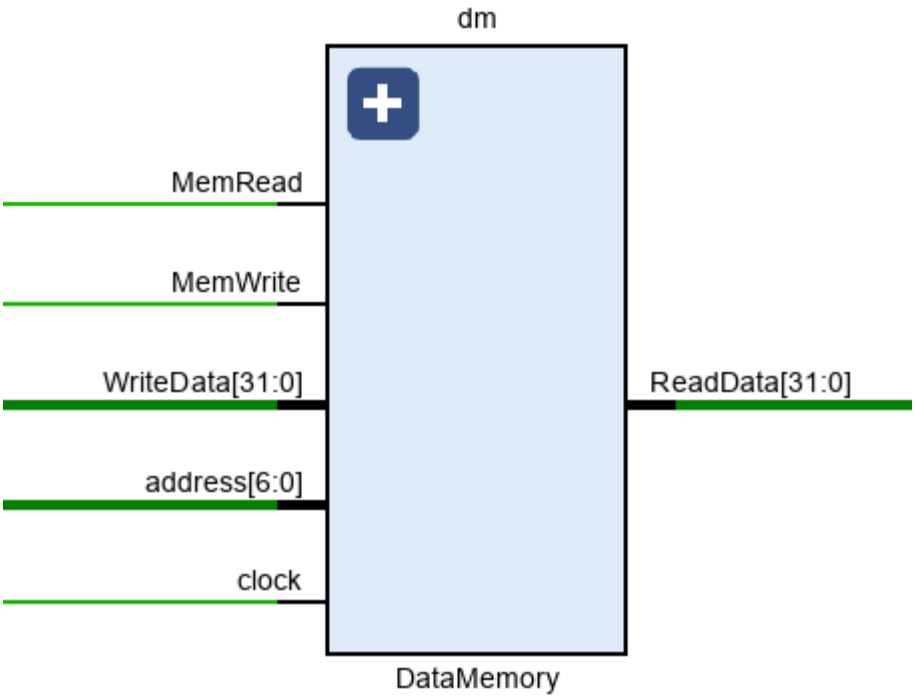

Figure 13: Data Memory RTL Schematic

4.2. Modified Architecture with DSP Unit:

To modify the architecture implemented to include a dsp unit, which will do an moving average fir filter operation, we first had to go through several steps to accommodate and test the unit. Firstly, we had to decide how we will modify the architecture to have the DSP unit alongside a new instruction format to be able to handle it. Secondly, we had to implement the moving average FIR filter inside the DSP. Thirdly, we had to use MATLAB to produce a sample signal with noise to test the DSP unit's moving average FIR filter operation.

4.2.1. Architecture Modifications:

2.2.1.1. Modified R-type instruction format:

To implement a DSP unit inside our implementation of the MIPS single cycle architecture we had to introduce/modify an instruction format.

In our approach we modified the R-type instruction format to be able to handle the new DSP unit with a new field in its format “dsp”. We had chosen this approach because we wanted the DSP unit to load and store the input/output signal samples from registers. Furthermore, to modify the format of the instruction we had to consider that the instruction must be 32-bits and we can’t take any bits from the opcode, rs, rt, and rd to produce the new “dsp” field because if we changed their number of bits they will affect the rest of the architecture drastically. To accomplish this, we replaced the dsp field with the shamt field as our implementation doesn’t support any shifting instructions.

Furthermore, the operation of this instruction is that rs is the input sample to the dsp and rd is the register where the filtered output sample will be stored and the dsp field will be the operation to be done inside the dsp unit, currently as we only have 1 operation the dsp can do it will be “00001” for ease of use.

Here is the before and after the modification of the R-type instruction format:

Old R-type Format:

opcode	rs	rt	rd	shamt	funct
6-bits	5-bits	5-bits	5-bits	5-bits	6-bits

New R-type Format:

opcode	rs	rt	rd	dsp	funct
6-bits	5-bits	5-bits	5-bits	5-bits	6-bits

2.2.1.2. Modified Datapath:

To implement the DSP unit, we had to modify the data path of the implemented architecture. We will place our DSP unit at the execution stage of the data path. The DSP unit will take the input samples from the register file and the output samples will go through a newly added multiplexer which will be controlled using a control signal “DSPorALU”. The multiplexer’s job will be to choose between the output of the DSP or the output of ALU. Figure 22 shows the updated datapath.

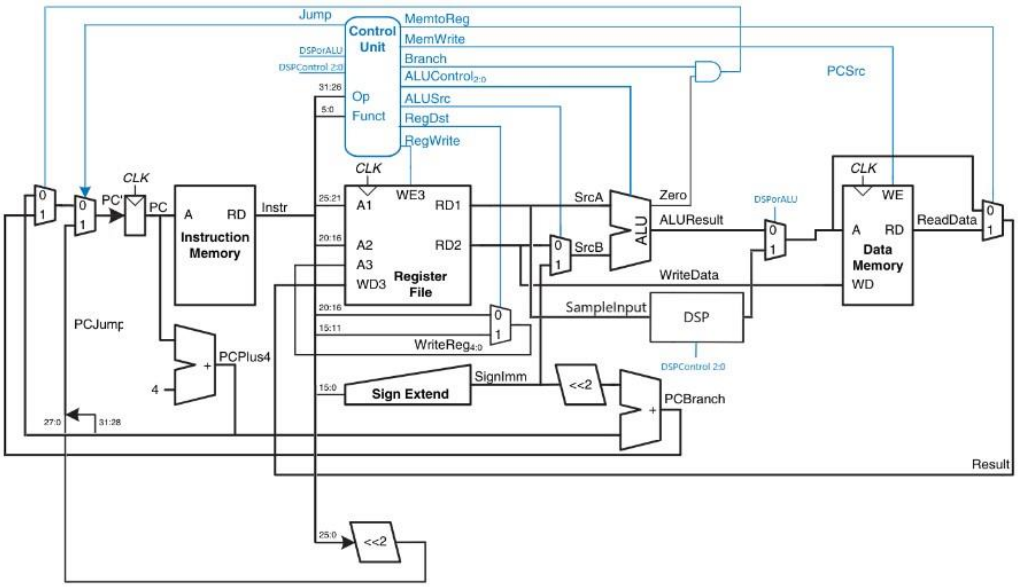

Figure 14: Modified Datapath

2.2.1.3. Modified Control Unit:

To accommodate the new changes to the datapath and the appearance of the new control signals DSPorALU and DSPControl we must modify the control unit.

Firstly, in the main decoder a new instruction will be present and will be called “dsp”. This new instruction will have the opcode “000001” and will have an ALUOp of “11”. Furthermore, a new signal will be added for the “DSPorALU” control signal. The updated Main decoder modified truth table is shown in table 3.

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	DSPorALU	MemWrite	MemtoReg	Op	Jump
R-type	000000	1	1	0	0	0	0	0	10	0
lw	100011	1	0	1	0	0	0	1	00	0
sw	101011	0	X	1	0	0	1	X	00	0
beq	000100	0	X	0	1	0	0	X	01	0
addi	001000	1	0	1	0	0	0	0	00	0
j	000010	0	X	X	X	X	0	X	XX	1
dsp	000001	1	1	X	0	1	0	0	11	0

Table 3: Modified Main Decoder Truth Table

Secondly, in the ALU decoder, or shall we say the secondary decoder as it will handle both ALU and DSP units, we will add new fields “DSP” and “DSPControl” to handle the DSP

instruction which will have the Op "11". The updated secondary decoder modified truth table is shown in table 4.

Op	Funct	ALUControl	DSP	DSPControl
00	XXXXXX	010 (add)	XXXXX	XXX
01	XXXXXX	110 (sub)	XXXXX	XXX
10	100000	010 (add)	XXXXX	XXX
10	100010	110 (sub)	XXXXX	XXX
10	100100	000 (and)	XXXXX	XXX
10	100101	001 (or)	XXXXX	XXX
10	101010	111 (slt)	XXXXX	XXX
11	XXXXXX	XXX	00001	000

Table 4: Modified Secondary Decoder Truth Table

4.2.2. DSP Unit FIR Filter Operation:

After modifying the architecture to accommodate the DSP unit we went ahead into implementing the operation of the DSP unit which is a 3rd order moving average fir filter.

A moving average fir filter is a simple and widely used fir filter. The filter's main function is to smooth an input signal that might have some noise. The filter works by averaging a set of input samples to produce smoothed out output samples. The process of the filter is that for every input sample it will be averaged with the previous three input samples, initially the previous three input samples are 0. This process can be represented mathematically as $y[n] = \frac{1}{N}\sum_{k=0}^{N-1} x[n-k]$ and also by figure 23 where x[n] is the input sample, y[n] is the output sample, and 1/N is the coefficient where N is the size of the averaging window (Order+1). Furthermore, because of its simplicity all of the coefficients have the same value and can be easily calculated using the equation $b = \frac{1}{Order+1}$, since in our case we are using a third order version the coefficients will just be $b = \frac{1}{3+1} = 0.25$.

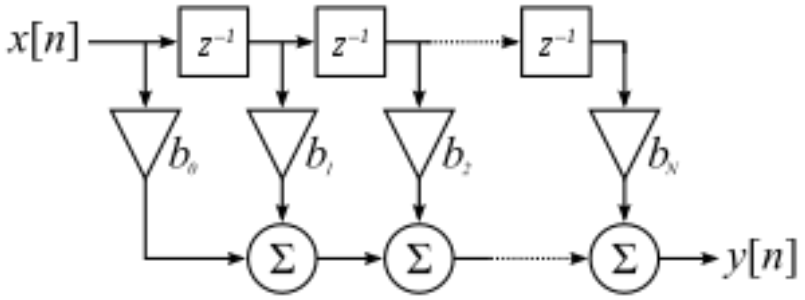

Figure 15: General Diagram for FIR Filters

The RTL schematic of the implemented filter is shown in figure 24.

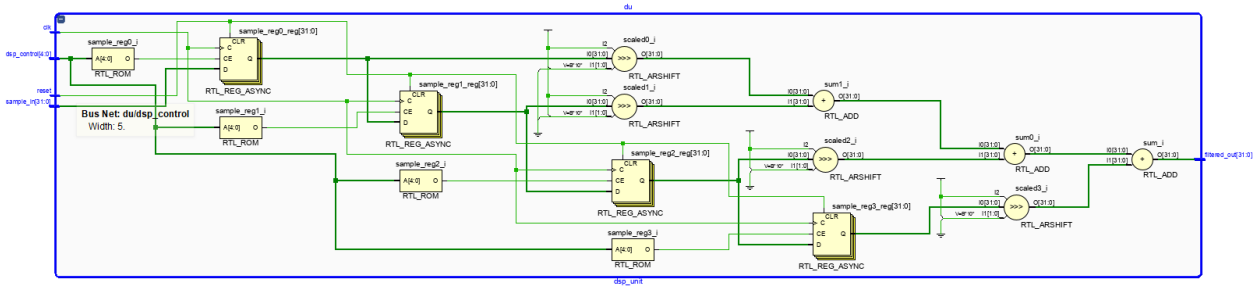

Figure 16: Moving Average FIR Filter RTL Schematic

4.2.3. Testing:

After fully implementing the dsp unit inside our single cycle architecture we can proceed to testing.

Our project successfully developed a testbench code responsible for generating the necessary clock and reset signals for the architecture. We also prepared a set of instructions specifically for both the DSP unit and the MIPS CPU. A crucial part of our setup involved initializing the register file with the DSP samples. The DSP instructions were designed to sequentially fetch and process each sample directly from this initialized register file. This entire setup and execution process was verified as successful.

Code:

20000000 // ADDI $0, $0, 0

8C040000 // LW $4, 0($0) - Load value from memory address 0 into $4

8C050004 // LW $5, 4($0) - Load value from memory address 4 into $5

00853020 // ADD $6, $4, $5 - Add $4 and $5, store in $6

AC060008 // SW $6, 8($0) - Store $6 into memory address 8

00852022 // SUB $7, $4, $5 - Subtract $5 from $4, store in $7

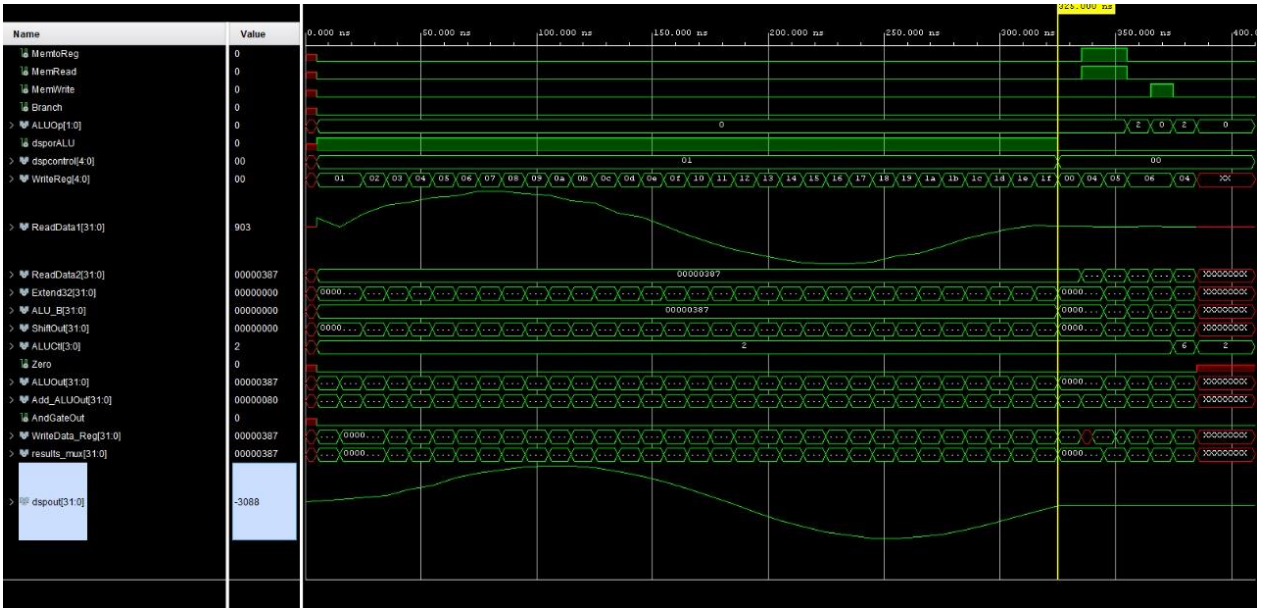

Figure 17: Waveform showing successful the DSP unit

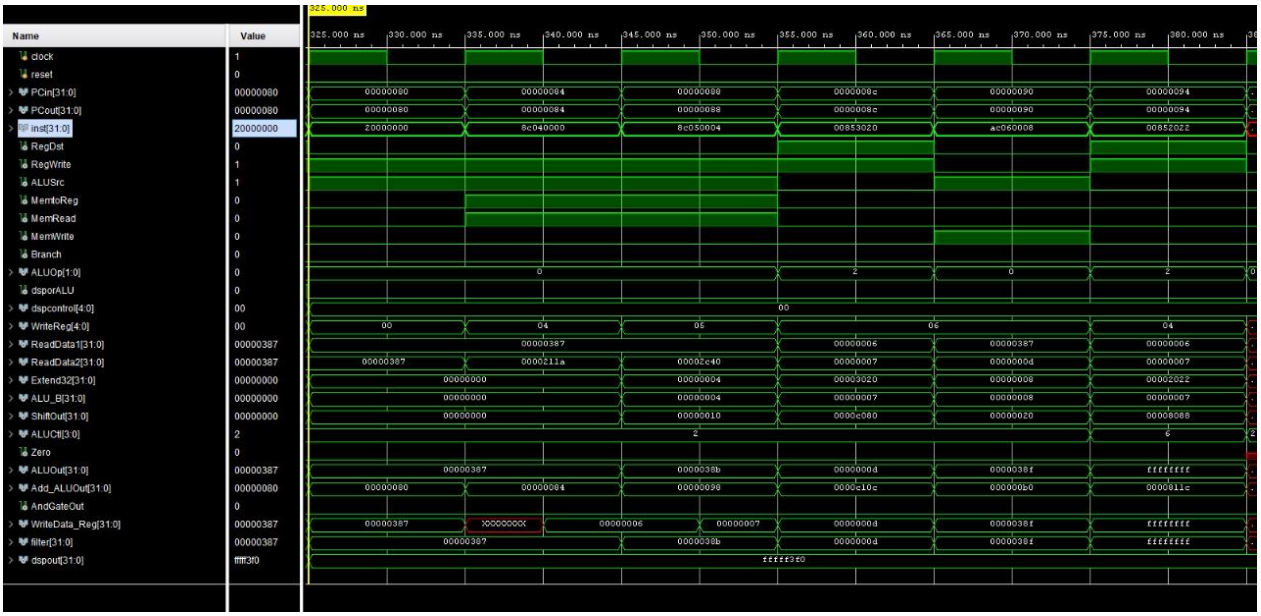

Figure 18: Waveform for multiple R-type I-type Instructions

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.841 ns	Worst Hold Slack (WHS):	0.052 ns	Worst Pulse Width Slack (WPWS):	8.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1014	Total Number of Endpoints:	1014	Total Number of Endpoints:	265

All user specified timing constraints are met.

Figure 19: Synthesis timing report

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.123 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.4°C**
Thermal Margin:	58.6°C (5.0 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

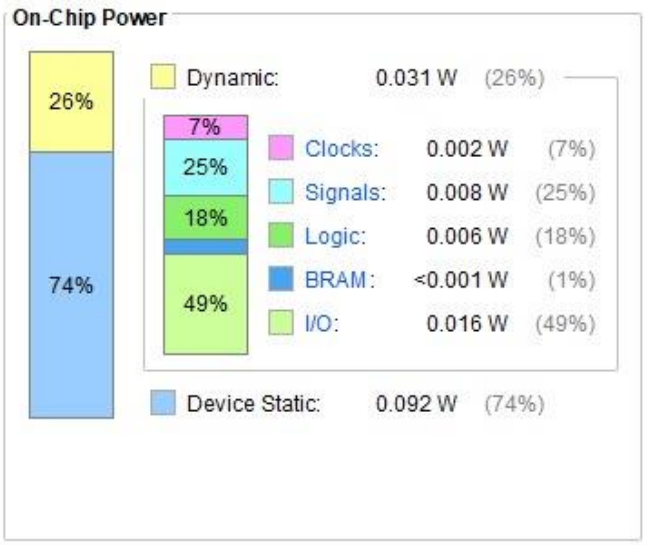

Figure 20: Synthesis power report

5. Conclusion:

This project successfully designed and verified a modified single-cycle CPU that includes a DSP filter unit. Key achievements include the addition of a new DSP instruction (opcode 000001) with an extended R-type format, and the implementation of a 3rd-order moving-average FIR filter in hardware. The DSP unit produces the correct filter output for a 32-sample input dataset, and standard MIPS instructions continue to function correctly. The results confirm that the extension integrates smoothly: the waveform outputs demonstrate the proper averaging behavior and normal CPU signal activity. In summary, the modified architecture meets the project objectives of integrating a DSP unit into a simple CPU while maintaining full MIPS compatibility.

6. References:

- U. Washington CSE378 Lecture Notes (single-cycle MIPS).
- “MIPS® DSP Module for MIPS32 Architecture” (MIPS® Processor Extensions).
- Analog Devices, Mixed-Signal and DSP Design Techniques (Harvard Architecture for DSPs).
- V. Oppenheim & R. W. Schafer, DSP: Moving Average Filters.
- AMD Application Note “Single-Multiplier MACC FIR Filter” (FIR hardware implementation).