PHASE 3 – MIPS Processor

Michael Mollica & Nidhay Patel

ECE 4120

SPRING 2023 04/25/2023

I. Modified Block Diagrams

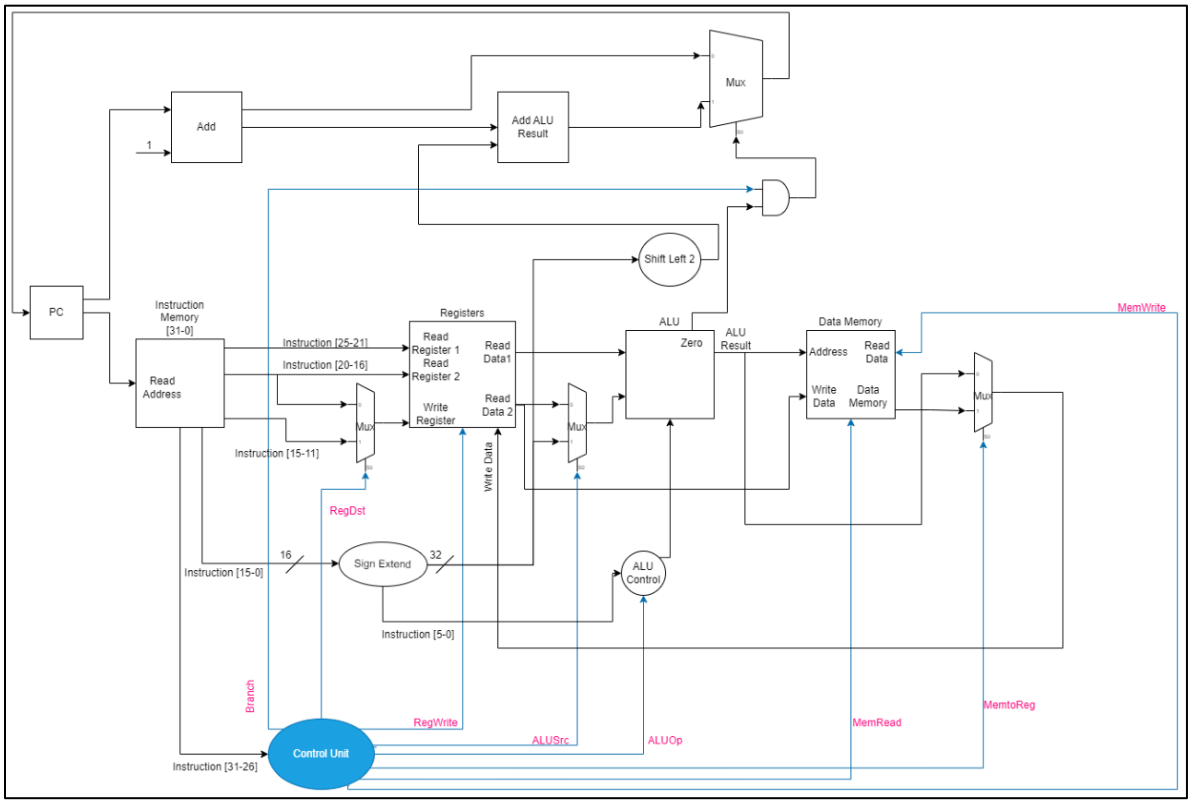

Figure 1: Single Cycle Block Diagram

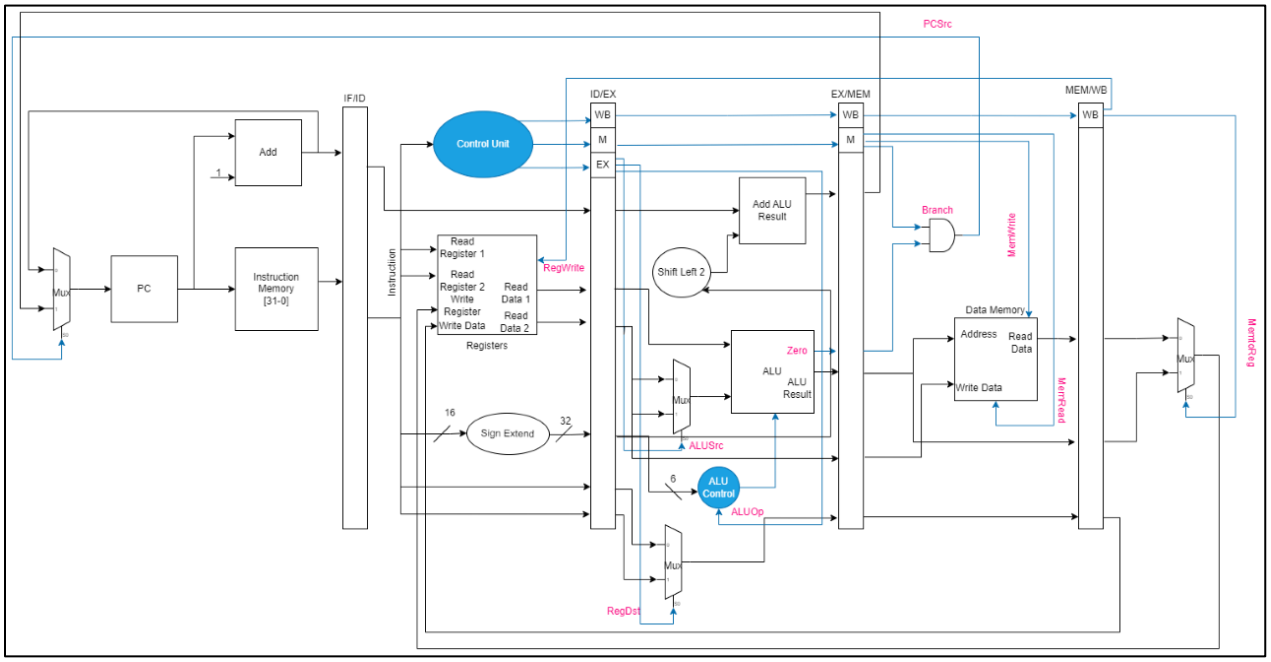

Figure 2: Pipelined Block Diagram

II. VHDL Elaboration

a. Control Unit

The control unit was implemented using a behavioral architecture. This architecture uses a case statement that checks the 6-bit opcode input to determine all output control signals. The control unit checks R-type, load, store, and branch instructions.

b. ALU Control Unit

The ALU control unit was implemented using a behavioral architecture. This architecture uses a series of if-statements and nested if-statements to check the ALUOp signal from the control unit and the function code from the instruction memory. The ALU control unit add, sub, and, or, sll, lw, sw, and beq instructions.

c. Data Memory Unit

The data memory unit was generated using a megafunction and adding a read enable input. Additionally, the output flip-flops were removed to reduce overhead in the system.

d. Branch Adder

The branch adder was implemented using a behavioral architecture. The architecture using the IEEE numeric standard library to perform addition the PC input and left shifted sign extension output. The left shift is performed by concatenating a 0 to the lower 31 bits of the sign extended immediate value from the instruction memory output.

e. Intermediate Registers

Each intermediate register (IF/ID, ID/EX, EX/MEM, MEM/WB) was implemented using behavioral architectures. The architectures use a process to set the input signals to the output signals on the rising edge of the clock signal. Each register has a unique set of inputs and outputs based on the block diagram.

III. DE10-lite Device Selection

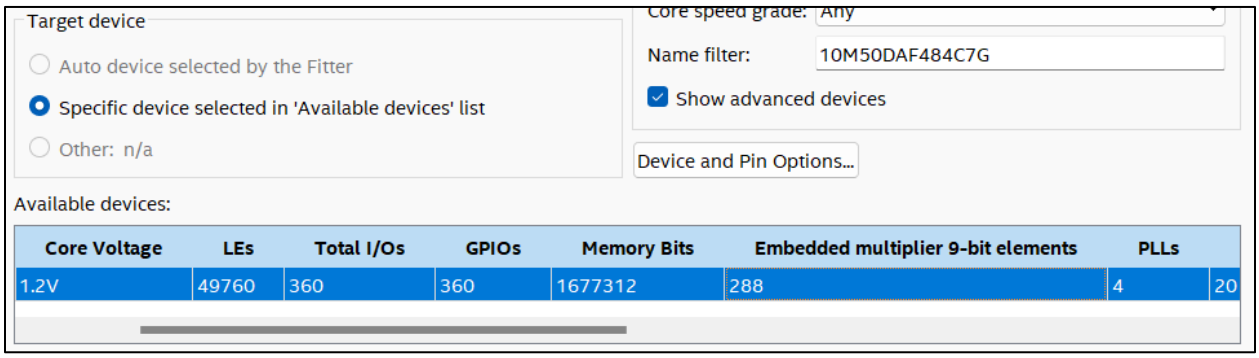

Figure 3: 10M50DAF484C7G Device Selection

IV. Flow Summary, RTL View, & Technology Map View

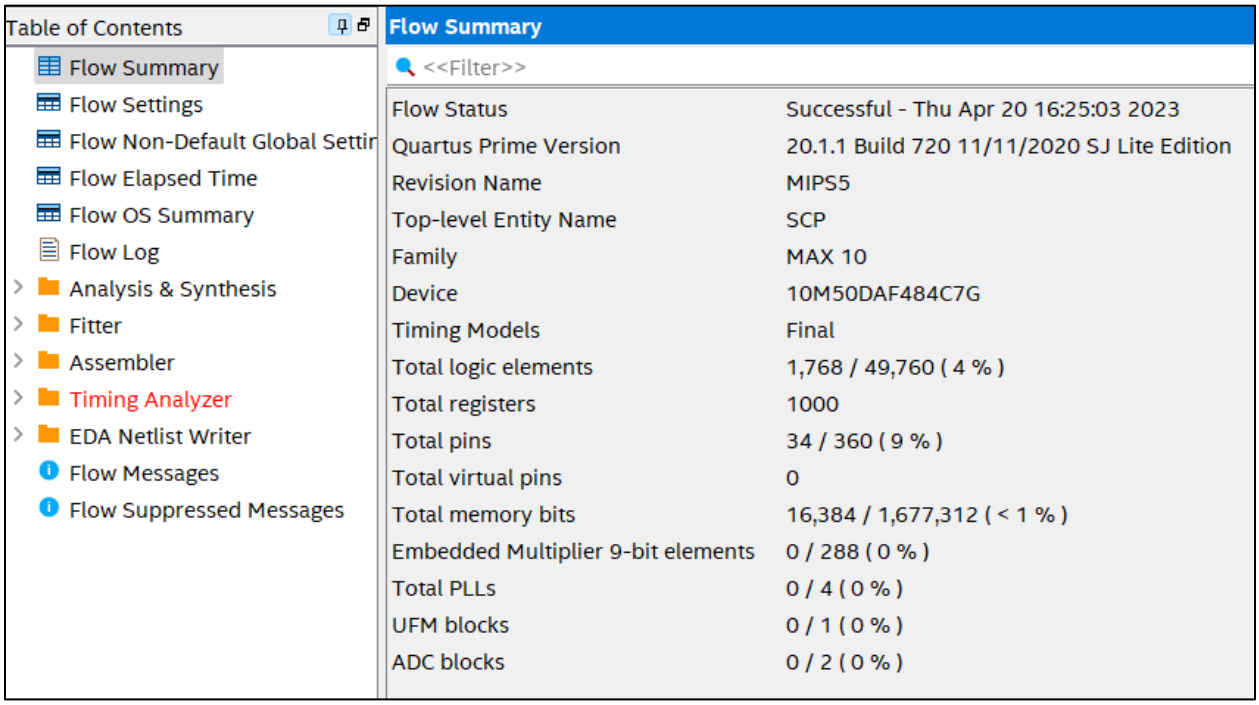

Figure 4: Single Cycle Flow Summary

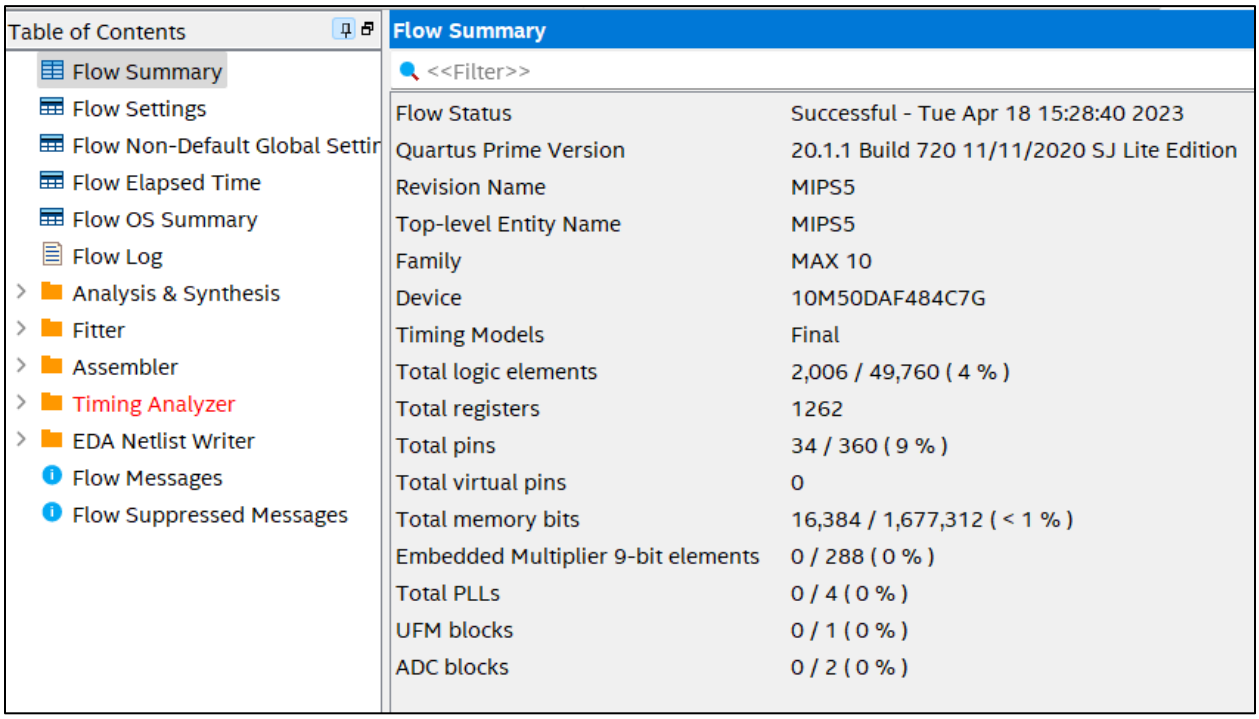

Figure 5: Pipelined Flow Summary

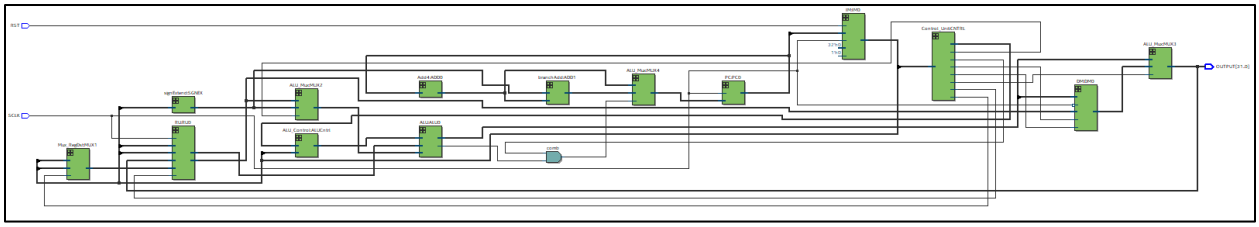

Figure 6: Single Cycle RTL View

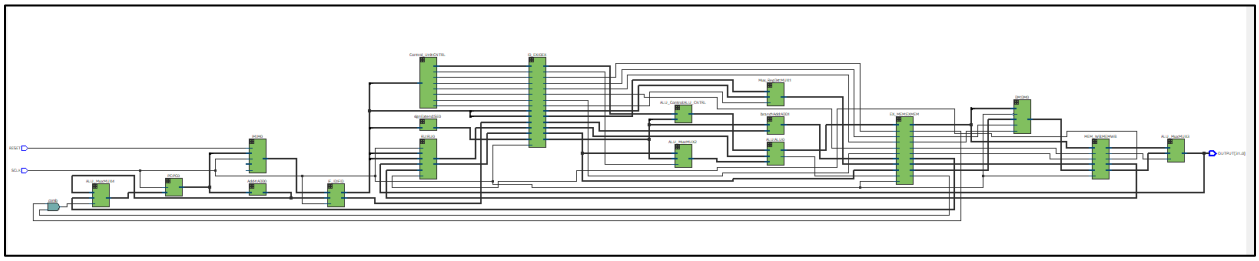

Figure 7: Pipelined RTL Viewer

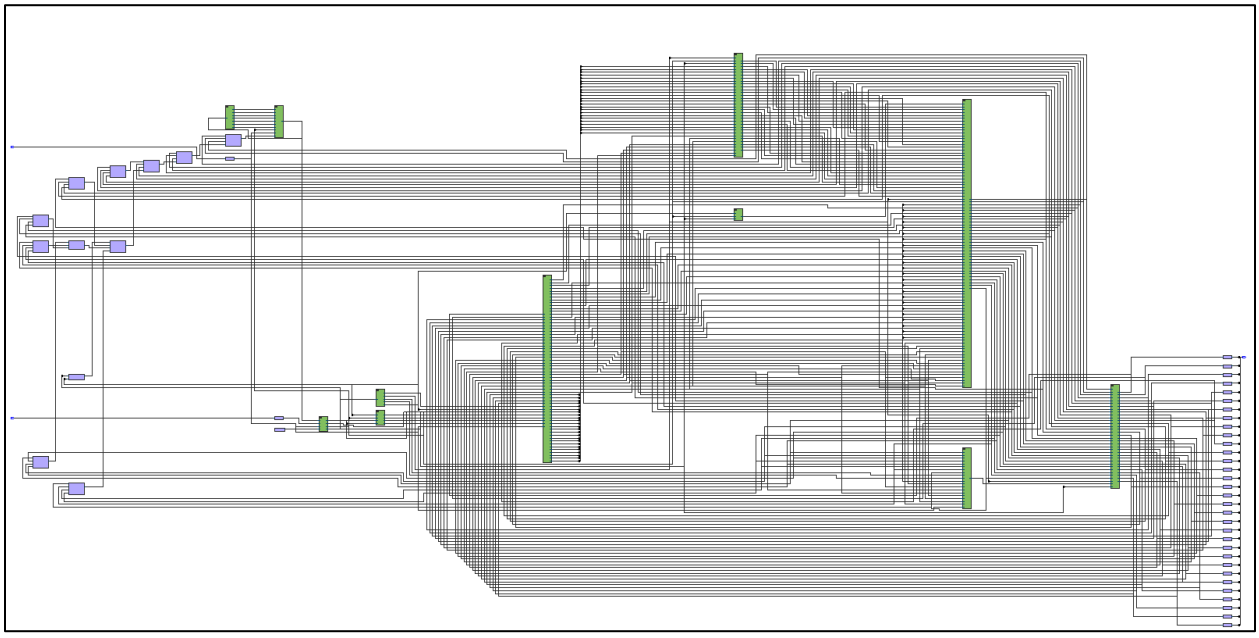

Figure 8: Single Cycle Technology Map View (Post-Mapping)

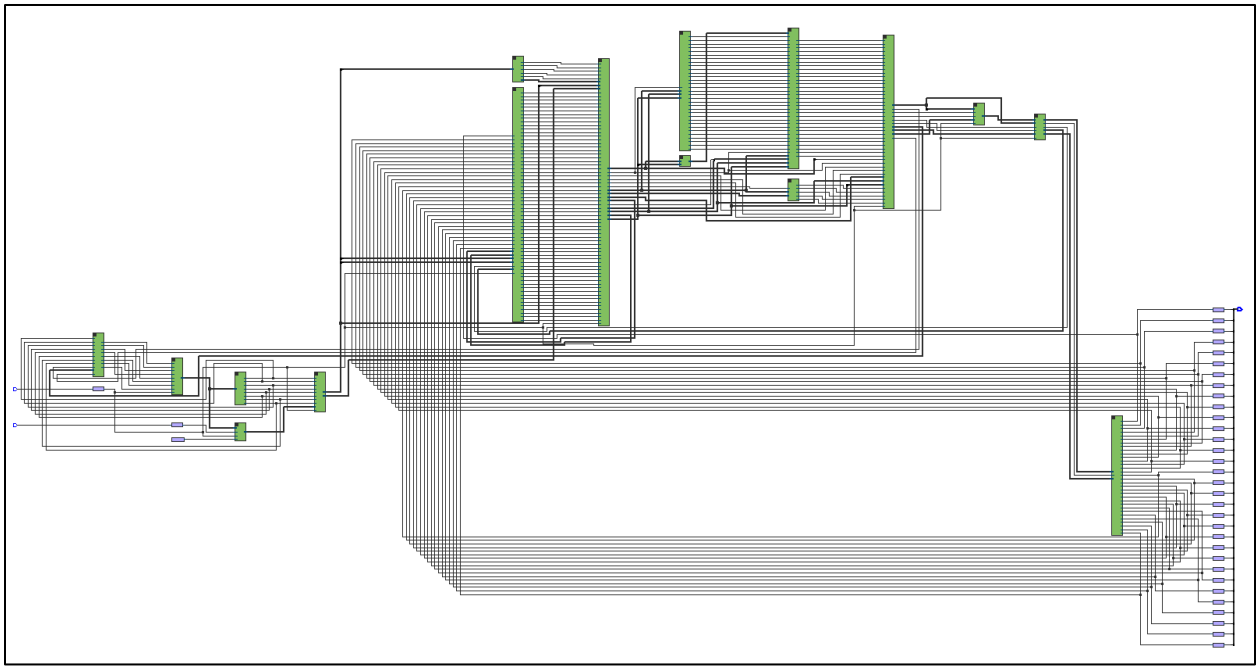

Figure 9: Pipelined Technology Map View (Post-mapping)

V. Testbench Elaboration

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity testbench_scp is
5   end entity;
6
7   architecture test of testbench_scp is
8   -- SCP Component
9   component SCP is
10  port ( SCLK     : in std_logic;
11         RST      : in std_logic;
12         OUTPUT   : buffer std_logic_vector(31 downto 0));
13  end component;
14
15  -- SIGNALS
16  signal clock        : std_logic := '0';
17  signal rst          : std_logic := '0';
18  signal mux3out      : std_logic_vector(31 downto 0);
19
20  -- PORT MAP
21  begin
22     DUT : SCP port map (clock, rst, mux3out);
23
24     clock <= not clock after 100ns;
25
26  end architecture;
```

Figure 10: Single Cycle Testbench Code

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity testbench is
5   end entity;
6
7   architecture test of testbench is
8   -- MIPS Component
9   component MIPS5 is
10  port ( SCLK    : in std_logic;
11         RESET   : in std_logic;
12         OUTPUT  : buffer std_logic_vector(31 downto 0));
13  end component;
14
15  -- SIGNALS
16  signal clock   : std_logic := '0';
17  signal rst     : std_logic := '0';
18  signal mux3out : std_logic_vector(31 downto 0);
19
20  -- PORT MAP
21  begin
22     DUT : MIPS5 port map (clock, rst, mux3out);
23
24     clock <= not clock after 50ns;
25
26  end architecture;
```

Figure 11: Pipelined Testbench Code

The testbench requires only port mapping the top-level entity and creating a clock signal for the entire system. All register values were instantiated using .mif files and verified in ModelSim.

VI. Waveforms

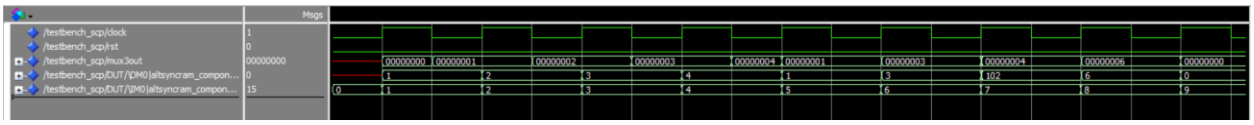

Figure 12: Single Cycle Output Waveform

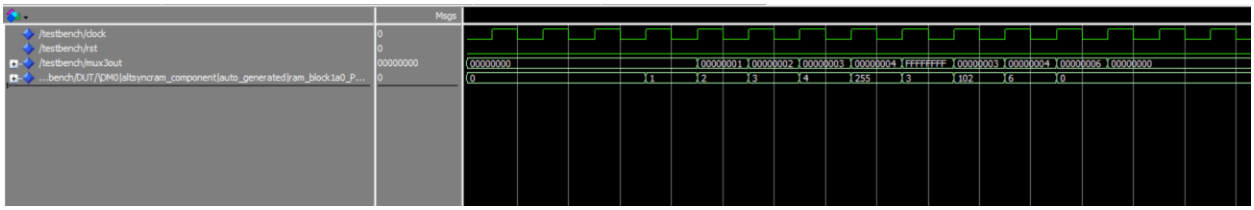

Figure 13: Pipelined Output Waveform

The figures show the output of the writeback mux for the single cycle and pipelined implementation respectively. The single cycle waveform is top to bottom as follows: Clock, Reset, Writeback Mux, Data Memory Address, and Instruction Memory Address. The pipelined waveform is top to bottom as follows: Clock, Reset, Writeback Mux, and Data Memory Address. Both waveforms were loaded with the same instructions and register values. The instruction set and expected outputs can be seen below.

Table 1 – Waveform Results

Instruction	SC Expected Output	SC Actual Output	PL Expected Output	PL Actual Output
lw $t1, 1($t0)	0x00000001	0x00000001	0x00000001	0x00000001
lw $t2, 2($t0)	0x00000002	0x00000002	0x00000002	0x00000002
lw $t3, 3($t0)	0x00000003	0x00000003	0x00000003	0x00000003
lw $t4, 4($t0)	0x00000004	0x00000004	0x00000004	0x00000004
beq $t1, $t2, Equal	0xFFFFFFFF	0x00000001	0xFFFFFFFF	0xFFFFFFFF
add $t1, $t1, $t2	0x00000003	0x00000003	0x00000003	0x00000003
sw $t3, 100($t2)	0x00000004	0x00000004	0x00000004	0x00000004
or $t1, $t4, $t2	0x00000006	0x00000006	0x00000006	0x00000006

As seen by the results in the table above, there is only a single error that was detected in out single cycle implementation during the branch equal instruction. This error could not be resolved by the deadline of the project; however, it is not detrimental to the system function as the branch is properly not taken and the input is not being written to memory.

VII. Timing Analysis

Table 2 – Timing Analysis

Value	Single Cycle	Pipelined
Max Frequency	**40.62 MHz**	**122.73 MHz**
Setup Slack Time	**0.190 ns**	**0.852 ns**
Hold Slack Time	**0.174 ns**	**0.432 ns**

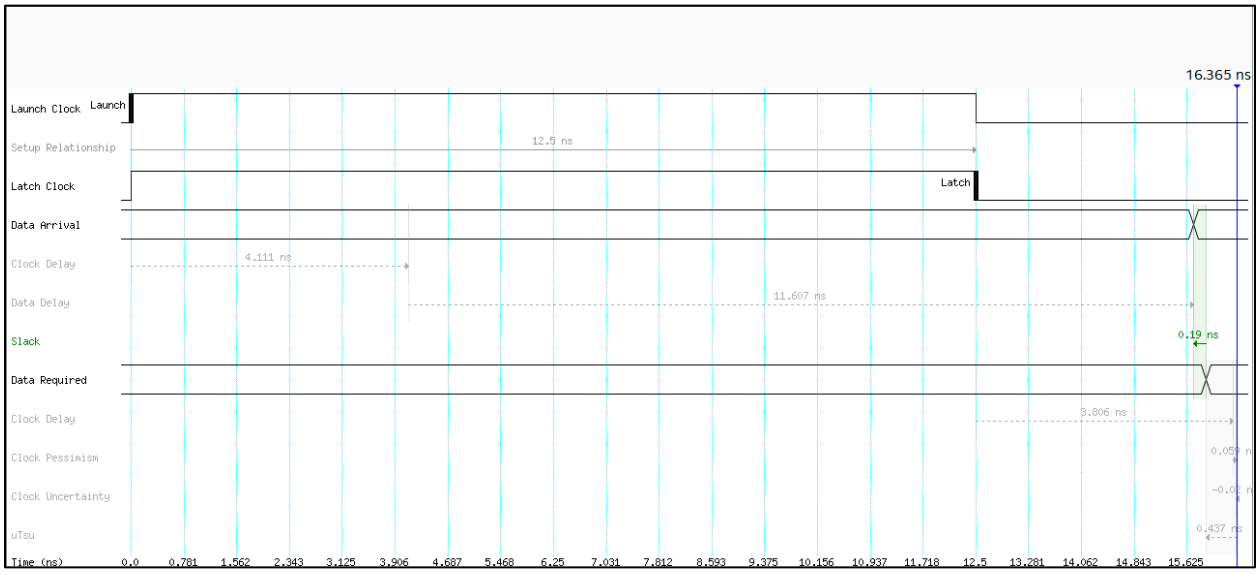

Figure 15: Single Cycle Positive Slack Time using 25 ns clock period

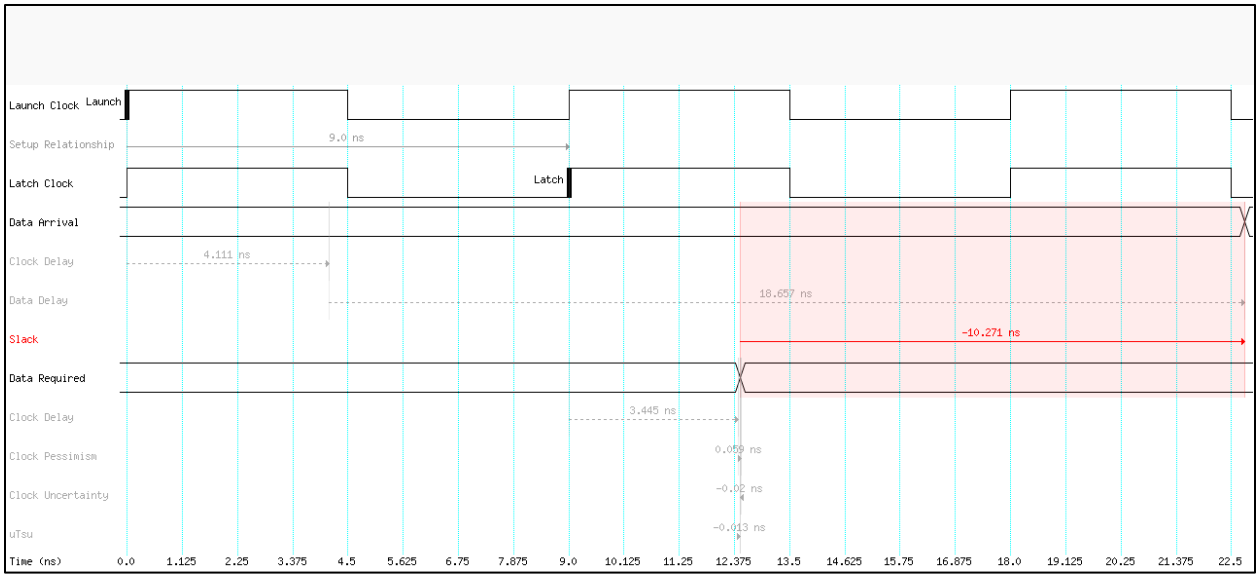

Figure 16: Single Cycle Negative Slack Time using 9 ns clock period.

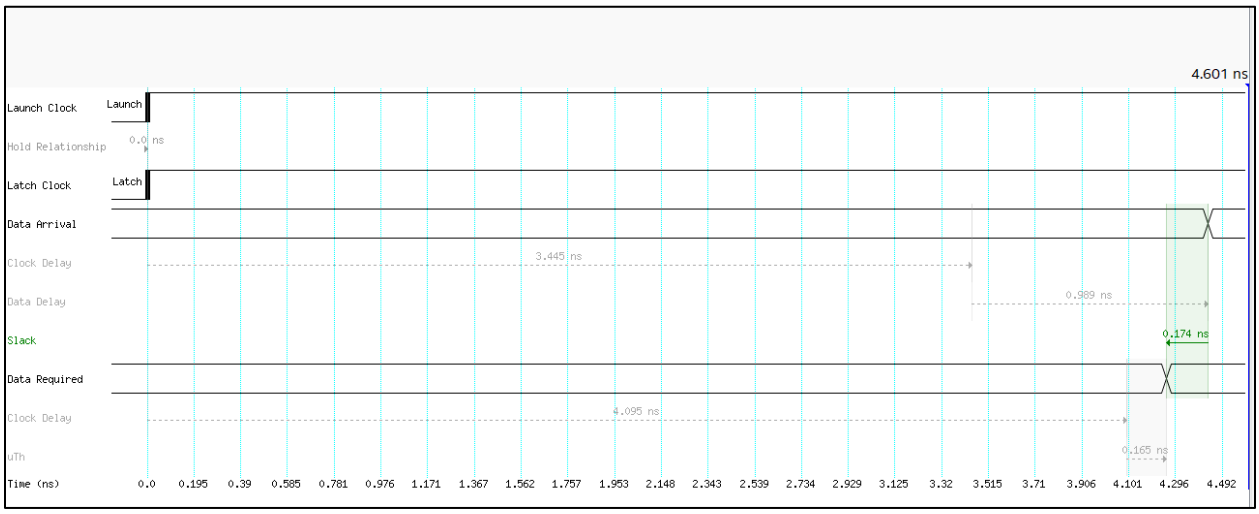

Figure 17: Single Cycle Hold Time

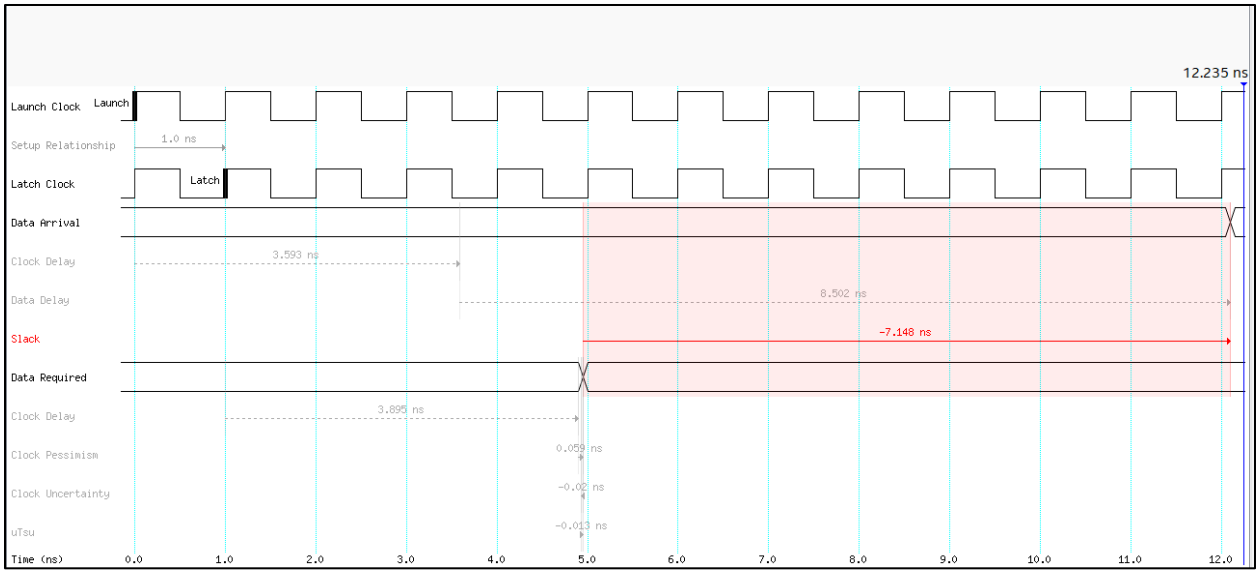

Figure 10: Pipelined Negative Slack Time using 1 ns clock period.

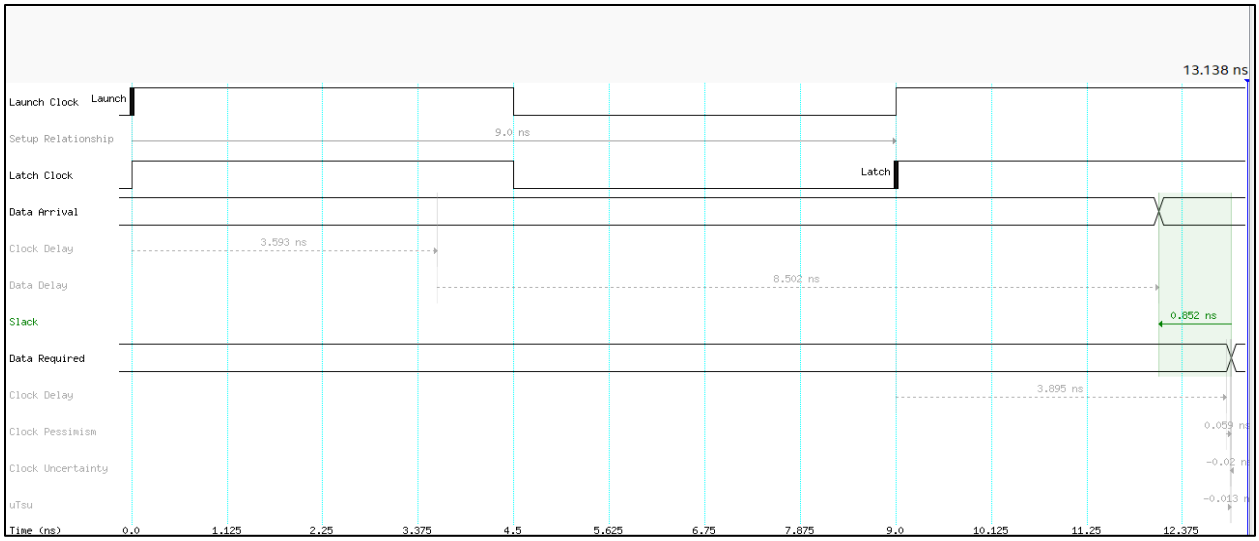

Figure 11: Pipelined Positive Slack Time using 9 ns clock period.

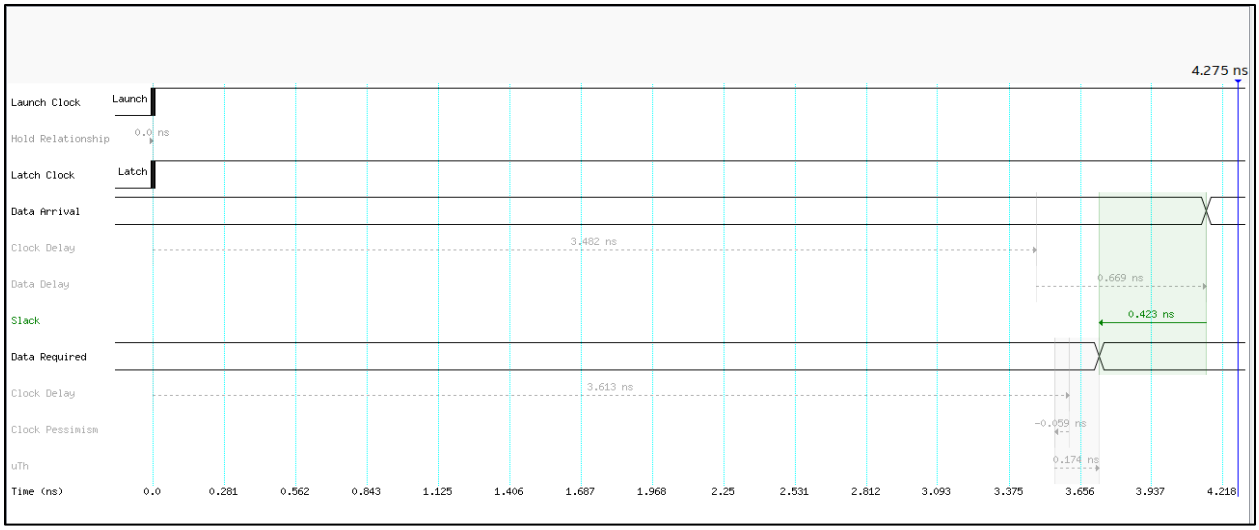

Figure 12: Pipelined Positive Hold Time

VIII. Pipelining Improvements

The pipelined architecture can be clocked over 80 MHz faster than the single cycle implementation of the MIPS processor. However, when using only 8 instructions (lw * 4, beq, add, sw, or), the single cycle can complete the instructions in less clock cycles. The single cycle implementation requires only 8 cycles to complete the instruction set whereas the pipelined implementation requires 12 clock cycles (5 + 8 – 1). Regardless of the single cycle processor taking less clock cycles to complete the instruction set, the overall time of execution is still greater than the pipelined processor.

$$P_{sc} = \#\ of\ cycles * CCT = 8 * 25\ ns = \mathbf{200\ ns}$$

$$P_{pl} = \#\ of\ cycles * CCT = 12 * 9\ ns = \mathbf{108\ ns}$$

IX. Pipelined Hardware Overhead

Since the pipelined processor uses megafunctions for instruction memory and data memory, there is some overhead on the inputs to both components. This is due to the irremovable D flip-flops on the inputs of for addressing, data input, and enables of the components. These flip-flops add additional delay in the system. Additionally, the intermediate registers between each pipeline stage cause more register delay. However, this overhead is necessary to successfully implement the pipelining process. Finally, according to the timing analyzer in Intel® Quartus® Prime, there is a clock skew of 0.361 ns.

X. Conclusion

After observing all the results, the single cycle and pipelined MIPS processor were successfully implemented with the exception of a single error during the branch equal instruction. Along with the code, the output is successfully displayed on the DE10-Lite board, in which further details can be found under the PROJECT_README and BOARD_README. Future work on this project

can include fixing the single cycle branch instruction implementation and adding hardware for forwarding support in the pipelined implementation.