Jacob Johnson
T00237585
Due: 4/1/2021
Dr. Hasan

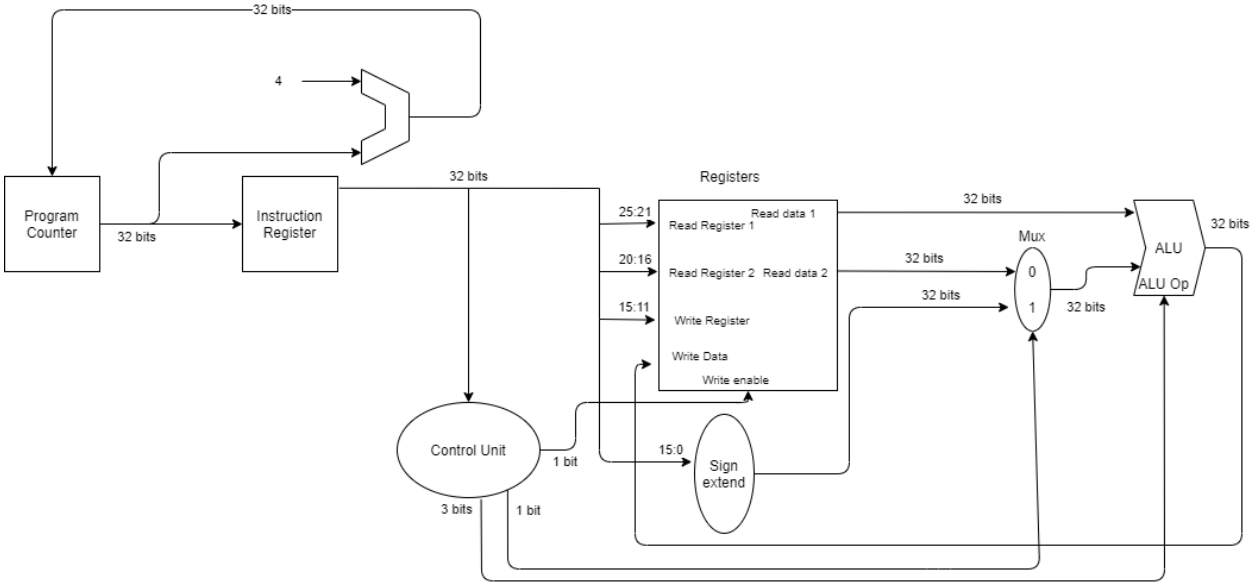
Milestone 2 Report

The Objective:

The goal of this portion of the project is to create logical blocks to recreate a MIPS processor register and arithmetic logic unit (ALU). To properly use these blocks, a signed extension and mux are also created.

The block diagram:

The block diagram in figure 1 depicts the conceptual implementation of what is fabricated for milestone 2. Note that milestone 1 was added to this implementation. This was chosen so that integration may be continual and not all at once.



The VHDL implementation:

As stated earlier, the implementation has 2 major components: the ALU and the registers. The ALU was provided as an appendix to the milestone 2 assignment. This ALU uses a behavioral approach by simply using a switch statement for the control signal. In each case, the basic operation is used on the operands, immediately sending the output without a clock.

The registers also use a behavioral approach, relying on process statements to do most of the logic. The register outputs the data from the register (unclocked). The correct register is indexed from the input std_logic_vector of the read register. This is done twice so that 2 registers

can be used as operands for read data 1 and read data 2. The ALU output is directed back into the registers as write data. This data is written when write enable and the clock are both high. In this way, it acts as a latch more than a register. This means it is not edge driven, but level driven. Combining the read and write unit allows for a simple usage of the registers.

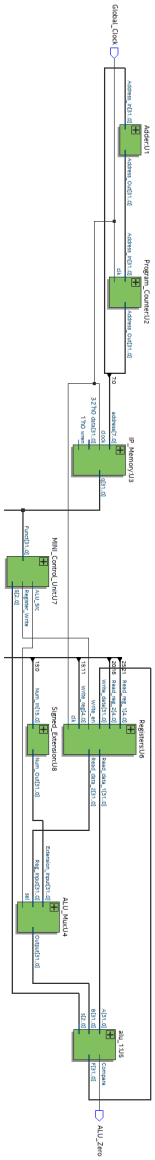
The sign extension operates on a basic if statement. The most significant bit (MSB) of the input signal is tested. If '1' then 16 ones are appended to the beginning of the input signal and driven to the output. If not '1' then the same is done with 16 zeros.

The mux is used to determine which source should go into ALU input 2. The '0' case passes the register output 2 to the ALU. The '1' case passes the output of the signed extension to the ALU input.

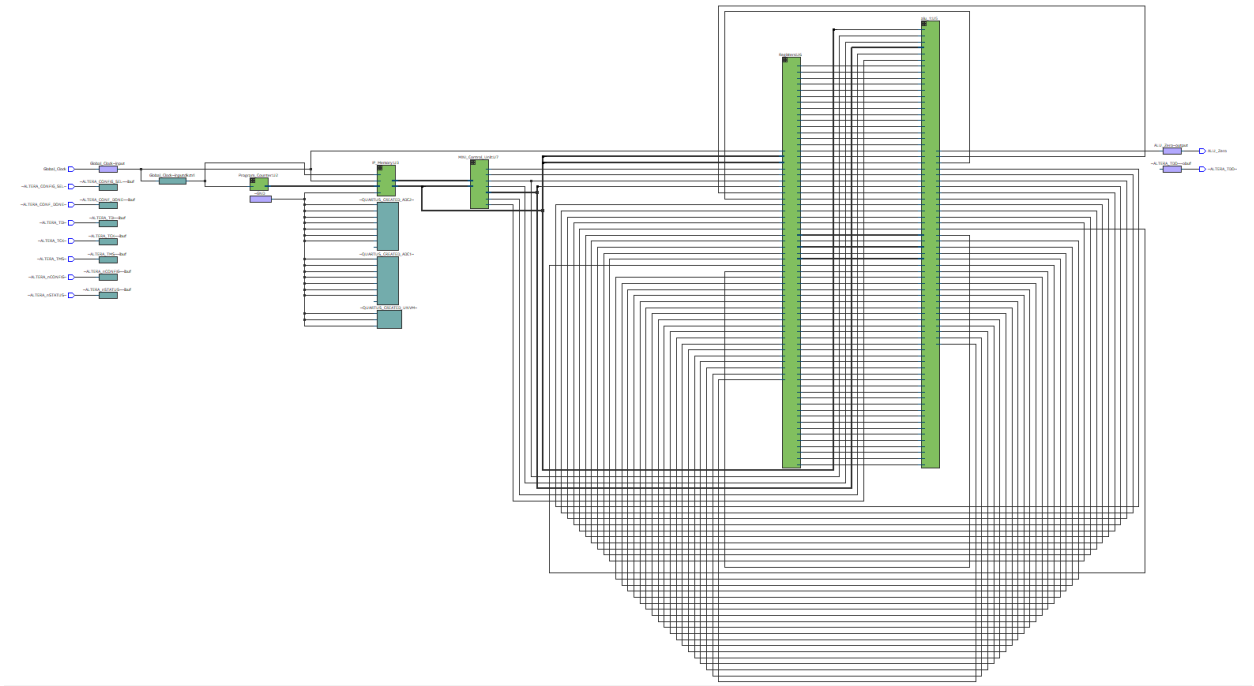
Flow summary:

Flow Status	Successful - Fri Apr 02 10:55:46 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	MIPS_Processor
Top-level Entity Name	Top_Level
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	2,679 / 49,760 (5 %)
Total registers	8
Total pins	2 / 360 (< 1 %)
Total virtual pins	0
Total memory bits	6,912 / 1,677,312 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

RTL View:


Global_Clock
Adder101
Program_Counter102
Memory103
MIPS_Control_Unit107
Register106
Signed_Extension108
ALU_Main104
Alu_1105
ALU_Zero

Technology map:



Elaboration on testbench:

For my implementation, I did not use a testbench. I created a .mif file for instruction memory to execute the instructions. Based on my assembly cheat sheet (attached as an appendix), I created the correct binary equivalent of the required 3 functions. These are add, subtract and logical AND. A testbench would serve the function of providing control signals, but this was not possible if I was not using a testbench. This meant I needed a control unit. This is not the final implementation of the control unit. The control unit receives the instruction memory out and outputs the ALU operation, register write, and select line of the ALU mux. This is done asynchronously and provides the otherwise impossible control signals to the necessary components. The registers are loaded initially by using default parameters in the signal declaration. This bypasses the need for any load or addi instructions.

Waveform elaboration:

The waveforms are located in the back as figure 2 and figure 3.

Figure 1:

This figure depicts the overall timing of the system. The waveform starts with a falling edge (to improve readability) and all values are at their default. The clock rises, giving an output from milestone 1. This instruction is decoded in the registers and fed to the ALU. Simultaneously, the control unit decodes the instruction and outputs control signals to the registers, mux and ALU. The ALU uses the decoded information to provide the proper operation for the output. The output is then fed back to the registers where the data is written back. This is because our operations all have writeback. If an instruction did not have writeback, the write enable line would have '0'.

Figure 2:

The ALU in signals are asserted from the registers with the ALU mux, always asserting '0'. Since none of the operations required signed extension, the signed extend unit is never passed through. If addi instructions were used to load the registers, signed extension would be needed. The writeback signal is generated in the control unit when the rest of the function is decoded, despite the propagation delay through the registers, mux and ALU.

Results:

The waveform generated is correct for the instructions provided. This is verified from the hand computed math being checked against the final register values. When the values are set in the beginning we generate the following:

$t0 = 3
$t1 = 3
$t2 = 4
$s0 = 10
$s1 = 8
$s2 = 0

The values are stored as binary numbers in a std_logic_vector with big endianness. Based on the instructions provided, $t3, $t4 and $t5 store the result of the operations. The final values are stored in std_logic_vectors (represented as integers) as follows:

$t3 = 6
$t4 = 2
$t5 = 2

This stands to be correct based on the parameters of the assignment.

Time analysis:

The FMax provided for the 85C model is 65.71 MHz and the 0C model is 69.31 MHz. The rate at which I calculate the clock to be is 50 MHz. This is less than the maximum frequency but I want to allow for fine tuning at the end of the design. During phase 1, I used a PLL to drive the clock of the system. For this milestone I was able to get rid of the PLL and program the clock directly. This allows for less resource allocation for something that was not necessary.

The slack times are as follows:

85C Model:

Setup slack: 2.391

Hold slack: 0.340

0C Model:

Setup slack: 2.786

Hold slack: 0.307

Notice that all slack is positive in this case. The hold slack is not dependent on the clock speed, and therefore is not considered as long as Quartus provides that the slack is positive and not negative.

The setup slack is a much larger number in our case and is dependent on the clock. Data required time - data arrive time is the basic equation for setup slack. Data arrival time is decided by combinational logic. This parameter should be reduced to the smallest possible but depends on the design of your project and can only be reduced to a certain point. Data required time is much easier to change and can be done so by changing the clock frequency. After calculating the current frequency, I was able to change the value to closely match the FMax found earlier. Since there is positive slack, the frequency is appropriate for the design implemented.

Conclusion:

The goal of this potion of the project is to implement an ALU and registers for use in a MIPS processor. This has been satisfied and works as required. For future work, more logic will need to be implemented for non R-type instructions to function as intended. This would be through a MUX at the entry of the write register index. The control unit would also need to include all cases of operation code and function code. However, since the control unit was not asked for as part of phase 2, the current functionality of this unit works.

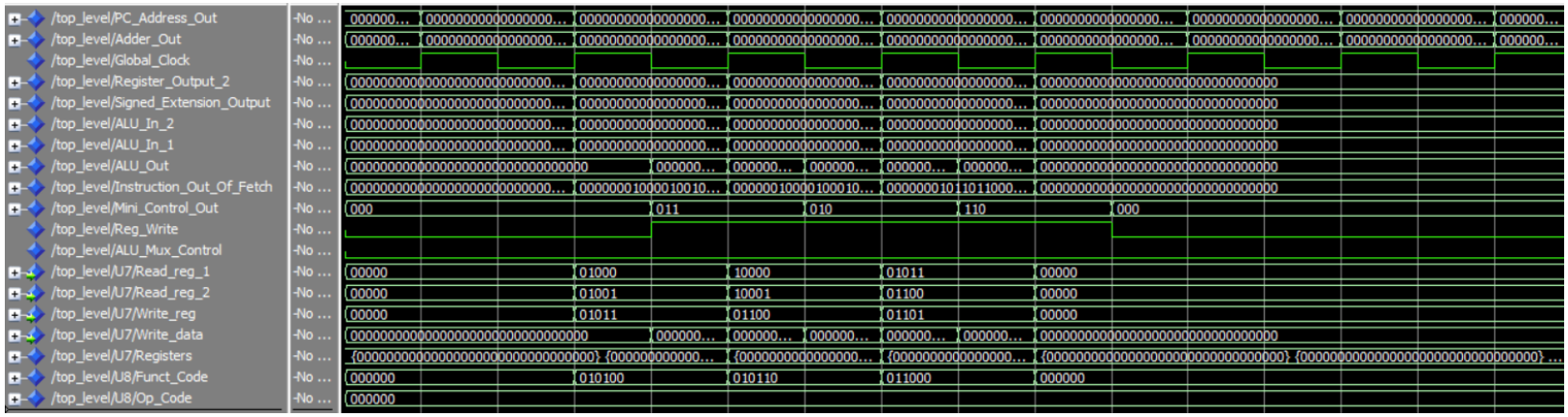


Figure 2

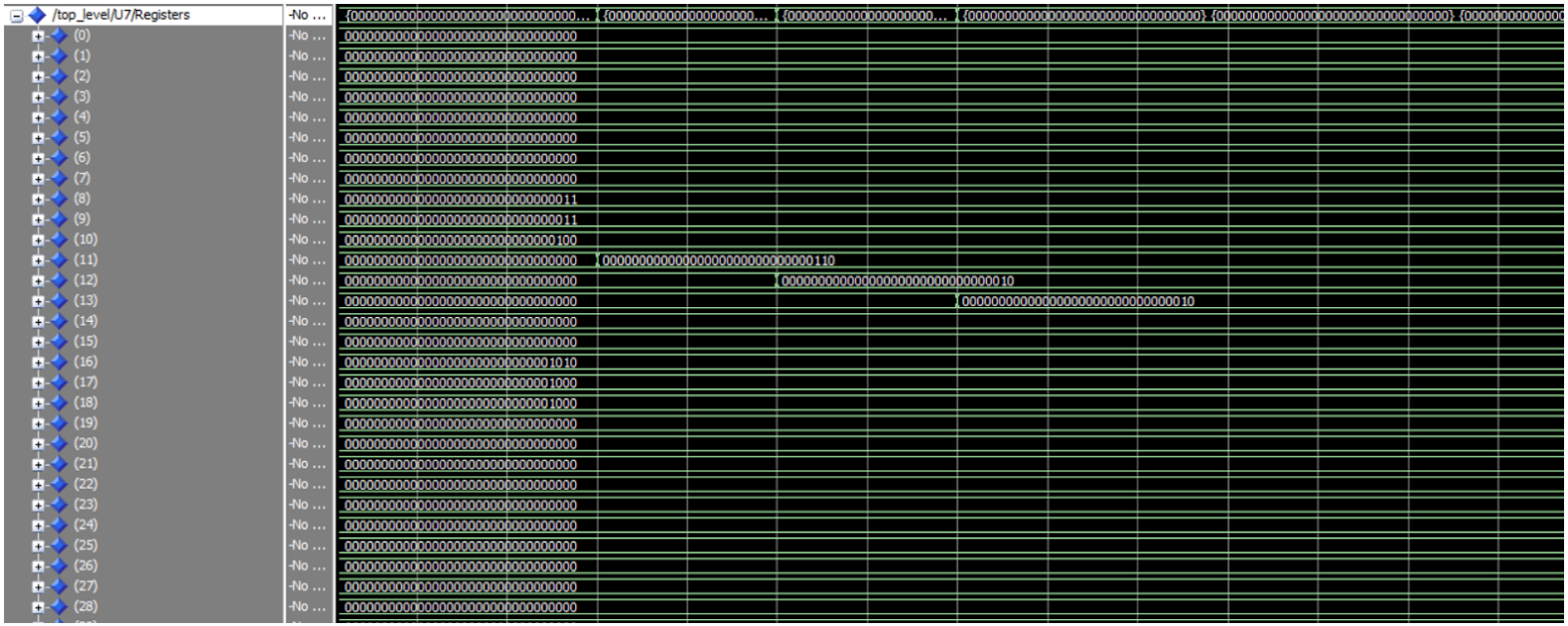


Figure 3