

Qualifying Phase Report - JoSDC’24

|  |  |  |
| --- | --- | --- |
| Team Name | | |
| **The Plumbers** | | |
| Team Members | | |
| # | Name | Email |
| 1 | Saleh Al-Ghobari | [SalehGhobari@outlook.com](mailto:SalehGhobari@outlook.com) |
| 2 | Omar Alhalabi | [o.alhalbi@gju.edu.jo](mailto:o.alhalbi@gju.edu.jo) |
| 3 | Ahmad Fratekh | [Ahmadkhalil.af@gmail.com](mailto:Ahmadkhalil.af@gmail.com) |
| 4 | Ahmad Omar | [a.omar3@gju.edu.jo](mailto:a.omar3@gju.edu.jo) |
| 5 | Omar Kasasbeh | [omar.kasasbeh.02@gmail.com](mailto:omar.kasasbeh.02@gmail.com) |

# Summary

|  |  |
| --- | --- |
| Total number of bugs found | 10 |
| Total number of bugs fixed | 10 |

# Corrected errors.

Table 1 Bug 1 information

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Bits assigned to Rs, Rt, and Rd | | |
| Bug ID : | 1 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/23rd/2024 |
| Assigned to: | The Plumbers | Close date | Sep/25th/2024 |
| Description | An error was found in the “processor.v” file regarding the assignment of the rs, rt, and rd bits. The bit assignment of each of them does not match the intended design, leading to an error and causing a mismatch between the registers where the results should be stored to or loaded from. | | |
| Steps to reproduce | Use the following instructions:  assign rd = instruction[25:21]  assign rs = instruction[20:16]  assign rt = instruction[15:11] | | |
| Expected Behavior | The rs register should consistently act as the source register, while the rd register should serve as the destination register. In R-Type instructions, the rt register functions as a source register, while in I-Type instructions, it is expected to act as a destination register. | | |
| Actual Behavior | The rs register was acting as though it were the rt register, while the rd register was functioning in place of the rs register. Additionally, the rt register was incorrectly behaving as the rd register. | | |
| Solution implemented | assign rd = instruction[15:11]  assign rs = instruction[25:21]  assign rt = instruction[20:16] | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | RF Mux “writeRegister” | | |
| Bug ID : | 2 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/23rd/2024 |
| Assigned to: | The Plumbers | Close date | Sep/25th/2024 |
| Description | An error was found in the “processor.v” file where the mux named “RFMux” provides the input to the register file’s “writeRegister” port. There was a naming mismatch, Particularly, the letter “W” was capitalized. | | |
| Steps to reproduce | Keep the name of the output of the RFMux module set as “WriteRegister”. | | |
| Expected Behavior | The Mux should select a signal, and the selected value should not be of high impedance (Floating value). | | |
| Actual Behavior | The incorrectly named RFMux output will be an implicitly defined floating net which causes the register file to operate incorrectly. | | |
| Solution implemented | Adjust the naming mismatch between the mux output and the register file input.  (WriteRegister -> writeRegister) | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Incorrect order of WB-mux inputs | | |
| Bug ID : | 3 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/26th/2024 |
| Assigned to: | The Plumbers | Close date | Sep/28th /2024 |
| Description | An error was found in the processor.v file, where the WB-mux input order is reversed. This results in incorrect data being written back to the register file. | | |
| Steps to reproduce | Use this module instantiation  mux2x1 #(32) WBMux(.in1(memoryReadData), .in2(ALUResult), .s(MemtoReg), .out(writeData)); | | |
| Expected Behavior | If memtoreg equals 0, the selected input should come from the ALU result. If memtoreg equals 1, the selected input should be the data from memory. | | |
| Actual Behavior | When memtoreg was 1, the Mux incorrectly selected the ALU result. When memtoreg was 0, the Mux incorrectly selected the input from memory. | | |
| Solution implemented | The WB-mux input order was corrected. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Wrong opSel assignment | | |
| Bug ID : | 4 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/27th/2024 |
| Assigned to: | The Plumbers | Close date | Sep/29th/2024 |
| Description | An error was discovered in the ALU.v file where the opSel signals for the AND and ADD operations were assigned incorrectly. This resulted in every addition operation performing a bitwise AND, and every AND operation performing an addition. The correct opSel values are:  ADD: 000  AND: 010 | | |
| Steps to reproduce | Set the opSel of the ADD operation to 010.  Set the opSel of the AND operation to 000. | | |
| Expected Behavior | When the opSel is 000, an ADD operation should be performed.  When the opSel is 010, an AND operation should be performed. | | |
| Actual Behavior | When the opSel is 000, an AND operation is performed.  When the opSel is 010, an ADD operation is performed. | | |
| Solution implemented | Adjust the assignment of opSel for the AND and ADD operations as follows:  AND: 010  ADD: 000 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | SLT operands order | | |
| Bug ID : | 5 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/27th/2024 |
| Assigned to: | The Plumbers | Close date | Sep/28th/2024 |
| Description | An error was detected in the “ALU.v” file, where the operand order in the SLT operation is incorrect, causing it to function improperly | | |
| Steps to reproduce | Use this assignment for the result value inside of the case statement  result = (operand2 < operand1) ? 1 : 0 | | |
| Expected Behavior | The destination register (rd) should be set to 1 if the content of the “operand1” register (rs) is less than the content of the “operand2” register (rt) and 0 otherwise. | | |
| Actual Behavior | The destination register (rd) is set to 1 if the content of the “operand2” register (rt) is less than the content of the “operand1” register (rs) and 0 otherwise. | | |
| Solution implemented | Reverse the order of the comparison operation in the case statement result = (operand1 < operand2) ? 1 : 0 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Missing default value for “result” in “ALU.v” | | |
| Bug ID : | 6 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/27th/2024 |
| Assigned to: | The Plumbers | Close date | Sep/27th/2024 |
| Description | Missing default value for “result” in “ALU.v”  An error was found in the “ALU.v” file where “result” was not assigned a default value, potentially leading to the creation of unintended latches. The issue was resolved by initializing “result” to 32'b0 at the beginning of the always block, ensuring proper initialization before the case statement is executed. | | |
| Steps to reproduce | Get rid of the default case/initialization statement. | | |
| Expected Behavior | The result signal should not infer a latch and should behave as combinational logic, properly representing the outcome of the ALU operations. | | |
| Actual Behavior | If the ALU module receives an invalid opSel input that is not defined in the case statement, the result variable will retain its previous value from the last valid opSel input, leading to latch behavior. | | |
| Solution implemented | Initialized "result" to 32'b0 at the beginning of the always block, preventing the inference of latches. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Incorrect Multiplexer Size in mux2x1.v File | | |
| Bug ID : | 7 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/29th/2024 |
| Assigned to: | The Plumbers | Close date | Sep/29th/2024 |
| Description | The width of the multiplexer input and output in the “mux2x1.v” file is larger than the required 32 bits by 1 due to an incorrect assignment. | | |
| Steps to reproduce | Assign the input and output widths of the multiplexer as [size:0]. | | |
| Expected Behavior | If the input and output widths are correct, the multiplexer should not exhibit a high impedance (Z) signal on the most significant bit. | | |
| Actual Behavior | Due to the incorrect size, the multiplexer input and output display a high impedance signal on the most significant bit. | | |
| Solution implemented | Correct the module's input and output widths to [size-1:0]. This adjustment will ensure that the multiplexer operates as intended. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Control unit “ALUop” for the OR operation | | |
| Bug ID : | 8 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/30th/2024 |
| Assigned to: | The Plumbers | Close date | Oct/1st/2024 |
| Description | An error was found in the “controlUnit.v” file where the OR operation was being assigned a wrong “ALUop” value it was set in decimal instead of binary. Coincidentally, in this specific case it does not affect the functionality of the OR operation since 3’d011 is the decimal value 11 represented in binary as 1011 and since the width is specified to be 3 bits the MSB 1 is truncated consequently leading to the same behavior as if it were assigned the value 3’b011. However, the assignment of 3’d011 is incorrect and inconsistent since it only works in this specific case. | | |
| Steps to reproduce | Use this assignment in the “funct” case statement  \_or\_ : begin  ALUOp = 3'd011  end | | |
| Expected Behavior | The “ALUop” value should be assigned correctly as 3'b011 (binary representation of 11) for the OR operation. This assignment should ensure that the control unit correctly directs the ALU to perform the OR operation. | | |
| Actual Behavior | Due to the incorrect assignment of 3'd011 (decimal representation of 11), the “ALUop” value is initially set incorrectly. However, in this specific case, the truncation of the most significant bit (MSB) during assignment to a 3-bit value results in the same binary representation (3'b011) as if the correct assignment were used. Therefore, the OR operation is executed correctly despite the initial incorrect assignment. | | |
| Solution implemented | Use this assignment in the “funct” case statement  \_or\_ : begin  ALUOp = 3'b011  end | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Control Unit “lw” instruction control signals. | | |
| Bug ID : | 9 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/30th/2024 |
| Assigned to: | The Plumbers | Close date | Oct/1st/2024 |
| Description | An error was found in the “controlUnit.v” file regarding the "lw" instruction. The incorrect control signals were being asserted, leading to unintended behavior. | | |
| Steps to reproduce | Use the following assignments for these “lw” instruction control signals:  RegDst = 1'b1  MemReadEn = 1'b0  MemWriteEn = 1'b1 | | |
| Expected Behavior | The “lw” instruction should load a value from the data memory into a register specified in the instruction and in order to achieve this behavior the following control signals should have these values:  RegDst should be set to 0 to select the destination register from the read register file.  MemReadEn should be set to 1 to enable memory read.  MemWriteEn should be set to 0 as this is not a write operation. | | |
| Actual Behavior | RegDst was incorrectly set to 1, selecting the destination register incorrectly.  MemReadEn was set to 0, disabling memory read.  MemWriteEn was set to 1, causing a write operation instead of a read. | | |
| Solution implemented | Change the incorrect values of the following control signals as specified below:  RegDst = 1'b0 instead of 1’b1  MemReadEn = 1'b1 instead of 1’b0  MemWriteEn = 1'b0 instead of 1’b1 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Bug Title: | Control Unit “beq” instruction control signals. | | |
| Bug ID : | 10 | Bug Type | Logical error |
| Reported by: | The Plumbers | Open Date | Sep/30th/2024 |
| Assigned to: | The Plumbers | Close date | Oct/1st/2024 |
| Description | An error was found in the Control Unit regarding the "beq" instruction. The “ALUSrc” control signal was incorrectly asserted, leading to unintended behavior | | |
| Steps to reproduce | Use the following assignment for the "beq" instruction “ALUSrc” control signal:  ALUSrc = 1'b1 // This was the incorrect value  (All other control signals were correctly set) | | |
| Expected Behavior | The "beq" instruction should compare two registers and branch if they are equal. To achieve this behavior, the “ALUSrc” control signal should be set to 0 to select register input for the ALU.  (All other control signals are correctly set) | | |
| Actual Behavior | “ALUSrc” was incorrectly set to 1, causing the ALU to use an immediate value instead of the second register value for comparison.  (All other control signals were correctly set) | | |
| Solution implemented | The “ALUSrc” signal was changed from 1'b1 to 1'b0 to correctly select the register input for the ALU during the "beq" instruction execution. | | |

# Full Analysis

Table 2 Control Unit Analysis Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | RegDst | Branch | MemReadEn | MemToReg |
| ADD | 1 | 0 | 0 | 0 |
| ADDI | 0 | 0 | 0 | 0 |
| SUB | 1 | 0 | 0 | 0 |
| AND | 1 | 0 | 0 | 0 |
| OR | 1 | 0 | 0 | 0 |
| SLT | 1 | 0 | 0 | 0 |
| LW | 0 | 0 | 1 | 1 |
| SW | X | 0 | 0 | X |
| BEQ | X | 1 | 0 | X |
|  |  |  |  |  |
| Instruction | ALUOp | MemWriteEn | ALUSrc | RegWriteEn |
| ADD | 000 | 0 | 0 | 1 |
| ADDI | 000 | 0 | 1 | 1 |
| SUB | 001 | 0 | 0 | 1 |
| AND | 010 | 0 | 0 | 1 |
| OR | 011 | 0 | 0 | 1 |
| SLT | 100 | 0 | 0 | 1 |
| LW | 000 | 0 | 1 | 1 |
| SW | 000 | 1 | 1 | 0 |
| BEQ | 001 | 0 | 0 | 0 |

# Functional Testing

Table 3 Functional Testing Benchmark 1

|  |  |  |  |
| --- | --- | --- | --- |
| # | Instruction | Hexadecimal (Machine Code) | Result |
| eg | addi $5, $0, 0xff | **0x200500FF** | **$5 = 0x000000FF** |
| 1 | addi $6, $0, 0x55 | **0x20060055** | **$6 = 0x00000055** |
| 2 | sub $7, $5, $6 | **0x00a63822** | **$7 = 0x000000AA** |
| 3 | sw $7, 0x0($0) | **0xac070000** | **dataMem [0] = 0x000000AA** |
| 4 | lw $8, 0x0($20) | **0x8e880000** | **$8 = 0x000000AA** |
| 5 | beq $6, $7, fin | **0x10e60003** | **$6 != $7 (branch is not taken)** |
| 6 | or $9, $6, $7 | **0x00c74825** | **$9 = 0x000000FF** |
| 7 | and $8, $6, $7 | **0x00c74024** | **$8 = 0x00000000** |
| 8 | add $0, $6, $7 | **0x00c70020** | **$0 = 0x000000FF** |
| 9 | fin : slt $10, $0, $5 | **0x0005502a** | **$10 = 0x00000000** |

A screen shot of a computer

Description automatically generated

The simulation above shows the execution of the given test code. As part of our team's debugging process this waveform allows us to analyze how the design handles each instruction.

**Key observations:**

* Immediate addition loads the correct values into $5 and $6.
* ALU operations (subtraction, OR, AND) produce expected results in $7, $9, and $8
* The branch instruction (BEQ) is correctly not taken
* The final SLT instruction sets $10 as expected

This step-by-step view of register values changing over time is crucial for verifying the datapath and control logic. It helps us identify any discrepancies between expected and actual behavior.

# Performance Results

In this section, you are required to configure your corrected code to run in the Quartus tool. Using the Timing Analyzer (as discussed during the training phase), you will need to create a clock and specify timing constraints suitable for your processor. You are free to apply any constraints you deem appropriate.

After completing the analysis, report the performance metrics and required data in the table below. Additionally, provide the necessary commands from the Synopsys Design Constraints (SDC) file.

Table 4 Performance Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Metric | Value | Description |
|  | | Clock Frequency | 34.48 MHz | clock frequency you configured in the Quartus tool. |
|  | | Design Size (LE) | 1894 Logic Elements | Size of design in terms of logic elements |
| model | Slow 85C | Fmax | Fmax = 35.1 MHz  Setup Slack = 0.255 Hold Slack = 0.783 | Fmax: The highest frequency at which the processor can operate reliably.  Setup Time: The time required to set up signals before the clock edge.  Hold Time: The minimum time signals must remain stable after the clock edge. |
| Setup Slack |
| Hold Slack |
| Slow 0C | Fmax | Fmax = 38.18 MHz  Setup Slack = 1.403  Hold Slack = 0.724 |
| Setup Slack |
| Hold Slack |
| Fast 0C | Fmax | Fmax = 84.93 MHz  Setup Slack = 8.613 Hold Slack = 0.307 |
| Setup Slack |
| Hold Slack |

Provide the commands from your **SDC file** that define the clock and timing constraints.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Time Information

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

set\_time\_format -unit ns -decimal\_places 3  
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Create Clock

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

create\_clock -name {clk} -period 29.000 -waveform { 0.000 14.500 } [get\_ports {clk}]

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Set Clock Uncertainty

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

set\_clock\_uncertainty -rise\_from [get\_clocks {clk}] -rise\_to [get\_clocks {clk}] 0.020

set\_clock\_uncertainty -rise\_from [get\_clocks {clk}] -fall\_to [get\_clocks {clk}] 0.020

set\_clock\_uncertainty -fall\_from [get\_clocks {clk}] -rise\_to [get\_clocks {clk}] 0.020

set\_clock\_uncertainty -fall\_from [get\_clocks {clk}] -fall\_to [get\_clocks {clk}] 0.020

# Free Space

1. **Challenges Faced:**

Finding the errors was the most major challenge, since no clear syntax/runtime errors were found, however, the design was filled with logical errors, hence, they were hard to find.

And to demonstrate such an example, in the processor.v file at line 33, the error occurred because the mux output’s first letter was capitalized, WriteRegister instead of writeRegister, as it’s declared and used in the register file instance.

1. **Debugging:**

Firstly, we started by debugging each file on its own manually, successfully being able to extract most of the errors. However, some errors such as the “slt” error were hard to notice, and so, using the test bench provided and modelsim, and after simulating the design, we were able to detect all the errors mentioned in the bug report tables above and fix them.

1. **Testing:**

We conducted both functional simulation and waveform analysis to verify the performance and functionality of each component, using the modelsim tool to ensure everything operated as intended.

1. **More Information:**

To simplify the operation of converting the instructions into binary format, we built an assembler (using java), which is included with the final version of the design, that takes the instructions to be executed and outputs the binary format of each one, additionally, providing the structure of the “.mif “ file.

1. **Unmentioned Bugs :**
2. Although it was not mentioned in the bugs report tables, we noticed a small error in the signextender.v file, where the module name is set differently and doesn’t match the file name. However, this doesn’t affect the design except when trying to test this specific file on its own, by setting it as the top-level entity.
3. In the registerfile.v file, we believe that a slight modification should be made, such that it checks if the register address is 0, to prevent writing to it, as it is used to provide the value “0” by the definition of the Mips32 green sheet.