JoSDC'24

المسابقة الوطنية لتصميم الشرائح الإلكترونية

Jordan National Semiconductors Design Competition

Qualifying Phase Report - JoSDC'24

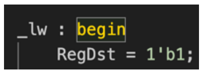
Team Name		
SiliCore		
Team Members		
#	Name	Email
1	Nedal Abu Haltam	nedalhaltam1@gmail.com
2	Qabas Ahmad Alkaissi	qab20210786@std.psut.edu.jo
3	Lojain Murad Hamdan	loj20210576@std.psut.edu.jo
4	Hasan Al-Hasnawi	has20210402@std.psut.edu.jo
5	Alanoud Assad Alsalem	ala20210371@std.psut.edu.jo

1. Summary

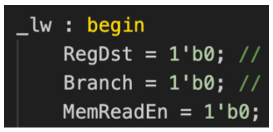
Total number of bugs found	22
Total number of bugs fixed	22

2. Corrected errors.

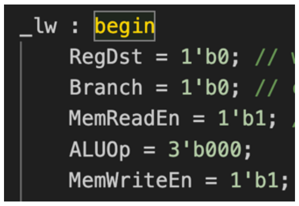
Table 1 Bug 1 information

Bug Title:	Control Unit lw RegDst Signal		
Bug ID:	1	Bug Type	Logic Error
Reported by:	Alanoud Alsalem	Open Date	2/10/2024
Assigned to:	Alanoud Alsalem	Close date	2/10/2024
Description	The lw instruction is an I-type instruction with destination register rt ([20-16] locations of the instruction). Hence, the RegDst signal to the multiplexor producing the output to the write register input of the register file should be 0 rather than 1. A screenshot of the bug is shown below:  `_lw : begin` `RegDst = 1'b1;`		
Steps to reproduce	In the controlUnit module, change the _lw RegDst signal to 1’b0.		

Expected Behavior	The destination register of the lw instruction should be Rt.
Actual Behavior	The destination register of the lw instruction is part of the immediate value in the I-type instruction.
Solution implemented	Change the RegDst signal to 1’b0

Bug Title:	Control Unit lw MemReadEn Signal		
Bug ID:	2	Bug Type	Logic Error
Reported by:	Alanoud Alsalem	Open Date	2/10/2024
Assigned to:	Alanoud Alsalem	Close date	2/10/2024
Description	The lw instruction reads from the data memory and stores the result in the register rt, so the MemReadEn signal should be 1. A screenshot of the bug is shown below:  `_lw : begin` `RegDst = 1'b0; //` `Branch = 1'b0; //` `MemReadEn = 1'b0;`		
Steps to reproduce	In the controlUnit module, change the _lw MemReadEn signal to 1’b1.		
Expected Behavior	The lw instruction should be able to read from the data memory.		

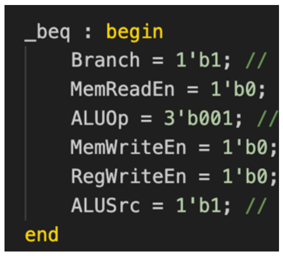
Actual Behavior	The lw instruction is not able to read from the data memory.
Solution implemented	Change the MemReadEn signal to 1’b1.

Bug Title:	Control Unit lw MemWriteEn Signal		
Bug ID:	3	Bug Type	Logic Error
Reported by:	Alanoud Alsalem	Open Date	2/10/2024
Assigned to:	Alanoud Alsalem	Close date	2/10/2024
Description	The lw instruction does not write to the data memory, so the MemWriteEn signal should be 0. A screenshot of the bug is shown below: 		
Steps to reproduce	In the controlUnit module, change the _lw MemWriteEn signal to 1’b0.		
Expected Behavior	The lw instruction should not write to memory.		
Actual Behavior	The lw instruction is able to write to memory.		

```
_lw : begin
    RegDst = 1'b0; //
    Branch = 1'b0; //
    MemReadEn = 1'b1;
    ALUOp = 3'b000;
    MemWriteEn = 1'b1;
```

Solution implemented	Change the MemWriteEn signal to 1’b0.

Bug Title:	Control Unit lw Missing MemToReg Signal		
Bug ID:	4	Bug Type	Logic Error
Reported by:	Alanoud Alsalem	Open Date	2/10/2024
Assigned to:	Alanoud Alsalem	Close date	2/10/2024
Description	The lw instruction writes back a word from memory to the register file. The multiplexor after the data memory should receive a control signal of 1 from the control unit allow for this write back. This signal is missing in the controlUnit module.		
Steps to reproduce	In the controlUnit module, remove the _lw MemToReg signal assignment.		
Expected Behavior	The multiplexor after the data memory should receive a control signal of 1 in the lw instruction.		
Actual Behavior	The multiplexor after the data memory is receiving a random signal during the lw instruction since it is not assigned.		
Solution implemented	Add the MemToReg signal assignment for the _lw case in the controlUnit module.		

Bug Title:	Control Unit beq ALUSrc Signal		
Bug ID:	5	Bug Type	Logic Error
Reported by:	Alanoud Alsalem	Open Date	2/10/2024
Assigned to:	Alanoud Alsalem	Close date	2/10/2024
Description	The beq instruction compares registers rs and rt for equality, so the second operand entering the ALU must be rt. This is enabled by the control signal ALUSrc in the controlUnit module which must be equal to 0 rather than 1. This error is shown in the screenshot below:  `_beq : begin` `Branch = 1'b1; //` `MemReadEn = 1'b0;` `ALUOp = 3'b001; //` `MemWriteEn = 1'b0;` `RegWriteEn = 1'b0;` `ALUSrc = 1'b1; //` `end`		
Steps to reproduce	In the controlUnit module, change the _beq ALUSrc signal to 1’b1.		
Expected Behavior	The ALUSrc signal must be equal to 0 for rt to be the ALU’s second operand.		
Actual Behavior	The ALUSrc signal is equal to 1, making part of the immediate value of the I-type instruction the ALU’s second operand.		
Solution implemented	Change the ALUSrc signal to 1’b0.		

Bug Title:	Blocking assignment in Control Unit		
Bug ID:	6	Bug Type	Functional Error
Reported by:	Alanoud Alsalem	Open Date	2/10/2024
Assigned to:	Alanoud Alsalem	Close date	2/10/2024
Description	In the control unit module, a blocking assignment (=) was first used but it does not reflect real hardware behavior due to the execution in sequence. By using a non-blocking assignment (<=) is used to ensure parallel execution.		
Steps to reproduce	In the always block, use the blocking assignment to assign values to signals.		
Expected Behavior	Parallel execution of the statements emulating real hardware.		
Actual Behavior	Sequential execution of the statements.		
Solution implemented	Change the blocking assignment to non-blocking assignment.		

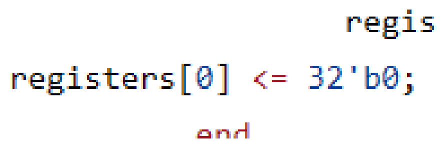
Bug Title:	Signal Assignment for Default funct Case		
Bug ID:	7	Bug Type	Logic Error

Reported by:	Alanoud Alsalem	Open Date	2/10/2024
Assigned to:	Alanoud Alsalem	Close date	2/10/2024
Description	The default case for the funct for the R-type instruction in the controlUnit module should ensure that the RegWriteEn signal is zero to prevent writing random values to the register file in case of an invalid funct value.		
Steps to reproduce	In the controlUnit module, remove RegWriteEn signal assignments in the default case for the funct in the R-type opcode case.		
Expected Behavior	If an invalid funct in an R-type instruction is received by the control unit, it should ensure that random data does not corrupt the register file. Hence, the RegWriteEn signal should be equal to zero.		
Actual Behavior	The RegWriteEn is left unassigned when a default case funct for an R-type instruction is received.		
Solution implemented	Assign the RegWriteEn signal to zero in the default case for the funct for R-type instructions in the controlUnit module.		

Bug Title:	Blocking assignment in always-block		
Bug ID:	8	Bug Type	Functional Error
Reported by:	Qabas Ahmad	Open Date	2/10/2024

Assigned to:	Qabas Ahmad	Close date	2/10/2024
Description	In the register file module, a blocking assignment (=) was first used but it does not reflect real hardware behavior due to the execution in sequence. By using a non-blocking assignment (<=) is used to ensure parallel execution.		
Steps to reproduce	In the always block, use the blocking assignment to assign values to the registers upon reset.		
Expected Behavior	Parallel execution of the statements emulating real hardware.		
Actual Behavior	Sequential execution of the statements.		
Solution implemented	Change the blocking assignment to non-blocking assignment.		

Bug Title:	Value of register 0		
Bug ID:	9	Bug Type	specification violation
Reported by:	Qabas Ahmad	Open Date	2/10/2024
Assigned to:	Qabas Ahmad	Close date	2/10/2024

Description	In the register file module, inside the always-block, there is no condition to check whether the write operation happened to register number 0 which should always contain the value 0.
Steps to reproduce	Assign the value to any register without checking whether or not the destination register is 0.
Expected Behavior	For the following instructions, the instructions accessing register 0 will have the value 0.
Actual Behavior	Register 0 will have the wrong value in it thus resulting in incorrect output result for all successive instructions using register 0.
Solution implemented	Add a statement that affirms the value of register 0 to be 0 as shown in this screenshot :  `regis` `registers[0] <= 32'b0;` `end`

Bug Title:	capitalize the w in writeRegister variable definition		
Bug ID:	10	Bug Type	syntax
Reported by:	Nedal Abu haltam	Open Date	30/9/2024
Assigned to:	Nedal Abu haltam	Close date	30/9/2024

Description	it is a syntax error, referred to in the modules as WriteRegister unlike the definition
Steps to reproduce	just capitalize the w
Expected Behavior	compiling
Actual Behavior	not compiling
Solution implemented	instead of defining it like writeRegister, we define it as WriteRegister

Bug Title:	assigning the rd field incorrectly		
Bug ID:	11	Bug Type	logical error
Reported by:	nedal abu haltam	Open Date	30/9/2024
Assigned to:	nedal abu haltam	Close date	30/9/2024
Description	the rd field of the instruction was not assigned correctly according to the given mips instruction format		

Steps to reproduce	assign the correct field to the rd variable by indexing the instruction correctly
Expected Behavior	the rd variable has the correct index specified in the instruction
Actual Behavior	the rd variable doesn't have the correct index specified in the instruction
Solution implemented	correctly indexing the instruction to get the rd field according to the MIPS format

Bug Title:	assigning the rs field incorrectly		
Bug ID:	12	Bug Type	logical error
Reported by:	nedal abu haltam	Open Date	30/9/2024
Assigned to:	nedal abu haltam	Close date	30/9/2024
Description	the rs field of the instruction was not assigned correctly according to the given mips instruction format		
Steps to reproduce	assign the correct field to the rs variable by indexing the instruction correctly		

Expected Behavior	the rs variable has the correct index specified in the instruction
Actual Behavior	the rs variable doesn’t have the correct index specified in the instruction
Solution implemented	correctly indexing the instruction to get the rs field according to the MIPS format

Bug Title:	assigning the rt field incorrectly		
Bug ID:	13	Bug Type	logical error
Reported by:	nedal abu haltam	Open Date	30/9/2024
Assigned to:	nedal abu haltam	Close date	30/9/2024
Description	the rt field of the instruction was not assigned correctly according to the given mips instruction format		
Steps to reproduce	assign the correct field to the rt variable by indexing the instruction correctly		
Expected Behavior	the rt variable has the correct index specified in the instruction		
Actual Behavior	the rt variable doesn’t have the correct index specified in the instruction		

Solution implemented	correctly indexing the instruction to get the rt field according to the MIPS format

Bug Title:	correcting the inputs to the WBmux		
Bug ID:	14	Bug Type	logic error
Reported by:	neda abu haltam	Open Date	30/9/2024
Assigned to:	nedal abu haltam	Close date	30/9/2024
Description	the inputs of the WBmux must be aligned with the selection line of the mux so it can be chosen correctly when needed		
Steps to reproduce	swap the first input with the second input of the mux		
Expected Behavior	if MemtoReg = 0 then select ALUResult else select memoryReadData		
Actual Behavior	if MemtoReg = 0 then select memoryReadData else select ALUResult		
Solution implemented	swap the first input with the second input of the mux		

Bug Title:	ALUOp Code Parameter Declaration Inconsistent with Guidelines		
Bug ID:	15	Bug Type	Logic Error
Reported by:	Lojain Hamdan	Open Date	2/10/2024
Assigned to:	Lojain Hamdan	Close date	2/10/2024
Description	The ALUOp codes for _ADD and _AND are incorrectly swapped. According to the guidelines, _ADD should be 0 and _AND should be 2, but in the code, _ADD is set to 'b010 (2) and _AND is 'b000 (0).		
Steps to reproduce	1. Check the ALUOp parameter declaration. 2. Observe the binary codes for _ADD and _AND.		
Expected Behavior	_ADD should be 3'b000 (0) and _AND should be 3'b010 (2).		
Actual Behavior	_ADD is 'b010 (2) and _AND is 'b000 (0).		
Solution implemented	Swap _ADD and _AND codes to follow the guidelines. parameter _ADD = 3'b000, _SUB = 3'b001, _AND = 3'b010, _OR = 3'b011, _SLT = 3'b100;		

Bug Title:	Missing Size Specifiers for Binary Literals in ALU Operation Parameters		
Bug ID:	16	Bug Type	Logic Error

Reported by:	Lojain Hamdan	Open Date	2/10/2024
Assigned to:	Lojain Hamdan	Close date	2/10/2024
Description	The binary literals assigned to the ALU operation parameters (_AND, _SUB, _ADD, _OR, _SLT) are missing size specifiers. In Verilog, if the size specifier is omitted, the default width is 1-bit (1'b0), which leads to size mismatch errors since opSel is a multi-bit signal.		
Steps to reproduce	1. Define the ALU parameters using binary literals without size specifiers (e.g., _ADD = 'b000). 2. Check for warnings or errors regarding binary literal sizes mismatches.		
Expected Behavior	The operation codes should be defined with the correct width that matches the opSel signal.		
Actual Behavior	The synthesis tool treats the literals as 1-bit values, which causes size mismatches when assigned to or compared with the opSel signal.		

Solution implemented	Explicit size specifiers were added to each binary literal.
	`parameter   _ADD = 3'b000, _SUB = 3'b001, _AND = 3'b010,`
	`            _OR   = 3'b011, _SLT = 3'b100;`		

Bug Title:	Incorrect Operand Comparison in SLT Operation		
Bug ID:	17	Bug Type	Logic Error
Reported by:	Lojain Hamdan	Open Date	2/10/2024
Assigned to:	Lojain Hamdan	Close date	2/10/2024

Description	The SLT (Set Less Than) operation uses an incorrect operand comparison. The expression (operand2 < operand1) is used, which reverses the intended comparison. The SLT operation should compare operand1 with operand2 to check if operand1 is less than operand2, not the other way around, as specified by MIPS core instruction set manual which specifies the operation in Verilog as: R[rd] = (R[rs] < R[rt]) ? 1 : 0
Steps to reproduce	1. Run the ALU with the opSel set to the SLT operation (_SLT). 2. Provide values where operand1 is smaller than operand2. 3. Observe that result incorrectly shows 0 when operand1 is less than operand2.
Expected Behavior	The SLT operation should set the result to 1 if operand1 < operand2, indicating that the first operand is smaller than the second.
Actual Behavior	The SLT operation incorrectly sets result to 1 if operand2 < operand1, reversing the intended logic.
Solution implemented	The comparison was corrected to (operand1 < operand2) to ensure proper SLT functionality. _SLT: result = (operand1 < operand2) ? 1 : 0;

Bug Title:	Incorrect SLT Comparison for Signed Operands		
Bug ID:	18	Bug Type	Logic Error
Reported by:	Lojain Hamdan	Open Date	2/10/2024
Assigned to:	Lojain Hamdan	Close date	2/10/2024
Description	The SLT (Set Less Than) operation does not correctly handle signed operand comparisons. The expression (operand1 < operand2) is used, which performs an unsigned comparison. However, SLT should handle signed numbers, where negative numbers are correctly treated as smaller than positive ones.		

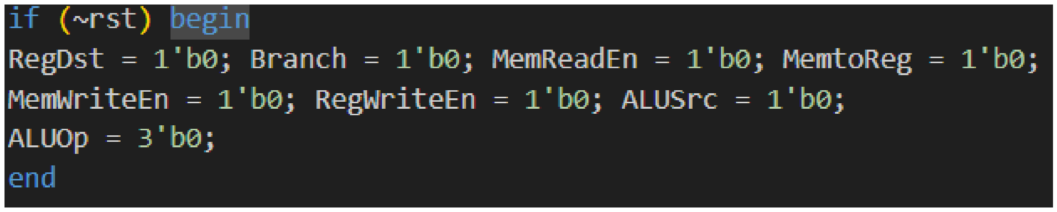
Steps to reproduce	1. Run the ALU with the opSel set to the SLT operation (_SLT). 2. Use signed operands where operand1 is negative and operand2 is positive. 3. Observe that result incorrectly shows 0 instead of 1.
Expected Behavior	The SLT operation should correctly perform a signed comparison, setting result to 1 if operand1 is less than operand2 when considering signed values.
Actual Behavior	The SLT operation uses an unsigned comparison, resulting in incorrect behavior when comparing signed values.
Solution implemented	The SLT operation was updated to use a signed comparison by using the $signed() function. _SLT: result = ($signed(operand1) < $signed(operand2)) ? 1 : 0;

Bug Title:	Missing Default Case in ALU Operation		
Bug ID:	19	Bug Type	Logic Error
Reported by:	Lojain Hamdan	Open Date	2/10/2024
Assigned to:	Lojain Hamdan	Close date	2/10/2024
Description	The ALU module did not have a default case for the opSel input, which could result in an unintended latch or undefined behavior when an invalid opSel is provided.		
Steps to reproduce	1. Provide an invalid opSel value (e.g., 3'b101, 3'b110). 2. Observe the behavior of the result signal. (Notice that the result might hold its previous value or remain undefined, leading to unpredictable behavior.)		
Expected Behavior	The ALU should default to setting the result to zero for invalid opSel values.		
Actual Behavior	The result register could hold a previous value or enter an undefined state.		

Solution implemented	A default case was added to set result to zero when opSel is invalid: default: result = 32'b0;

Bug Title:	Incorrect Zero Flag Comparison with Inconsistent Width		
Bug ID:	20	Bug Type	Logic Error
Reported by:	Lojain Hamdan	Open Date	2/10/2024
Assigned to:	Lojain Hamdan	Close date	2/10/2024
Description	The expression used to assign the zero flag compares result to 'b0, which is a 1-bit value. This results in a size mismatch, as result is a multi-bit signal (based on data_width). This can cause incorrect behavior during synthesis or result in warnings about mismatched widths.		
Steps to reproduce	1. Set up the ALU with any valid operation and ensure the result is being compared to 'b0 (1-bit zero). 2. Observe that the zero flag may not be correctly set due to the size mismatch between result and 'b0. 3. Check for synthesis warnings related to width mismatches.		
Expected Behavior	The zero flag should be correctly set by comparing the entire data_width of result against a data_width-wide zero vector (e.g., 32'b0 if data_width = 32).		
Actual Behavior	The comparison uses a 1-bit zero ('b0), causing potential mismatches, incorrect results, or synthesis warnings.		
Solution implemented	The comparison was corrected to use a zero vector with the same width as result, ensuring consistency in the comparison. zero = (result == 32'b0);		

Bug Title:	resetting the control signal before starting the execution		
Bug ID:	21	Bug Type	logical
Reported by:	nedal abu haltam	Open Date	30/9/2024
Assigned to:	nedal abu haltam	Close date	30/9/2024
Description	the control signal should be reset so we can fetch the first instruction appropriately and then figure the control signals according to the fetched instructions		
Steps to reproduce	add the reset signal to the control unit's port list		
Expected Behavior	the control signal to be all zeros so we can fetch the first instruction (i.e. the nextPC to be zero)		
Actual Behavior	the control signals are random or in the simulation they are x's(undefined)		

Solution implemented	add if condition to reset the signals if the rst signal is low  `if (~rst) begin`
	`RegDst = 1'b0; Branch = 1'b0; MemReadEn = 1'b0; MemtoReg = 1'b0;`
	`MemWriteEn = 1'b0; RegWriteEn = 1'b0; ALUSrc = 1'b0;`
	`ALUOp = 3'b0;`
	`end`		

Bug Title:	Syntax error in ALUop assignment in Control Unit		
Bug ID:	22	Bug Type	Syntax Error
Reported by:	Nedal abu haltam	Open Date	10/10/2024
Assigned to:	Nedal abu haltam	Close date	10/10/2024
Description	The or instruction was assigned a decimal value instead of a binary value.		
Steps to reproduce	Change 3’b011 to 3’d011		
Expected Behavior	The ALUop should be equal to 011 in binary for the or operation		
Actual Behavior	The ALU op is equal to 011 in decimal for the or operation		
Solution implemented	The d is changed to a b in the ALUop assignment		

#supplement pages for Bug report.

3. Full Analysis

In this section, you are required to fully analyze the processor and prepare to debug and trace its execution process. This involves summarizing the key signals and their behavior for all nine instructions supported by the processor. Understanding these signals is critical for troubleshooting and verifying that each instruction functions as expected.

The table below should be filled with an analysis of the key signals for each instruction, including their expected values during execution.

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	0	0	0	1
ADDI	0	0	1	1
SUB	1	0	0	1
AND	2	0	0	1
OR	3	0	0	1
SLT	4	0	0	1
LW	0	0	1	1
SW	0	1	1	0
BEQ	1	0	0	0

4. Functional Testing

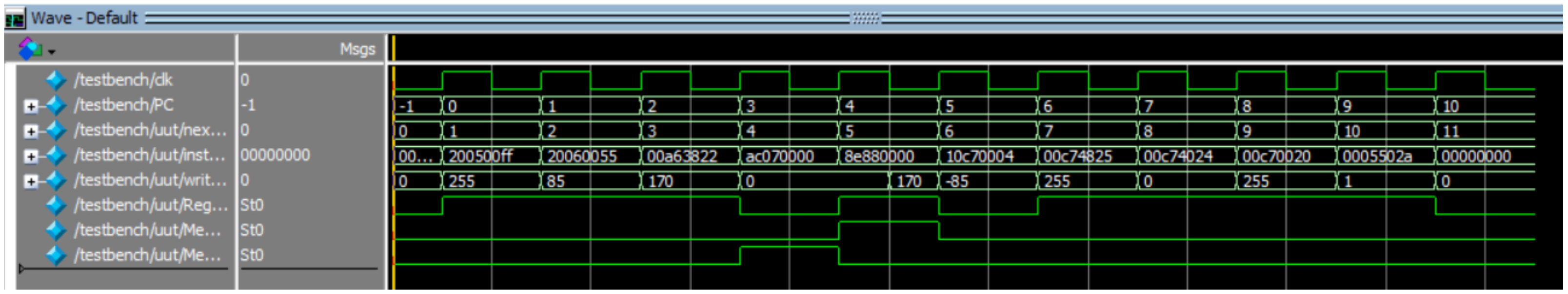
In functional testing part, instructions should be hand-assembled, converted to hexadecimal (machine code), and loaded into the instruction memory. Then make sure to fill out the required fields below and add the corresponding screen shots to show your testing process.

Table 3 Functional Testing Benchmark 1

#	Instruction	Hexadecimal (Machine Code)	Result

e g	addi $5, $0, 0xff	0x200500FF	$5 = 0x000000FF
1	addi $6, $0, 0x55	0x20060055	$6 = 85
2	sub $7, $5, $6	0x00A63822	$7 = 170
3	sw $7, 0x0($0)	0xAC070000	MEM[0] = $7 = 170
4	lw $8, 0x0($20)	0x8E880000	$8 = 170
5	beq $6, $7, fin	0x10C70004	$6 ≠ $7
6	or $9, $6, $7	0x00C74825	$9 = 255
7	and $8, $6, $7	0x00C74024	$8 = 0
8	add $0, $6, $7	0x00C70020	$0 = 0
9	fin : slt $10, $0, $5	0x0005502A	$10 = 1

Waveform:


```
Wave - Default
                          Msgs
/testbench/clk            0
/testbench/PC             -1        -1  0         1         2         3         4         5         6         7         8         9         10
/testbench/uut/nex...     0         0   1         2         3         4         5         6         7         8         9         10        11
/testbench/uut/inst...    00000000  00... 200500ff 20060055 00a63822 ac070000 8e880000 10c70004 00c74825 00c74024 00c70020 0005502a 00000000
/testbench/uut/writ...    0         0   255       85        170       0              170  -85       255       0         255       1         0
/testbench/uut/Reg...     St0
/testbench/uut/Me...      St0
/testbench/uut/Me...      St0
```

#supplement page for simulation results.

5. Performance Results (Optional)

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		clock frequency you configured in the Quartus tool.
		Design Size (LE)		Size of design in terms of logic elements
model	Slow 85C	Fmax		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		
		Setup Slack		
		Hold Slack		
	Fast 0C	Fmax		
		Setup Slack		
		Hold Slack		

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

```
#
```

6. Free Space

Clarification:

we changed the input of the branch adder. instead of adding the PCPlus1 we replaced it with PC so we can calculate the target address from the value of the PC of the fetched instruction and not the one after it.

#supplement page for free space section.