CPE 233 Hardware Assignment 5

Branch Condition Generator and Branch Address Generator

Report by:
Ethan Vosburg (evosburg@calpoly.edu)

February 9, 2024

Table of Contents

1 Project Description

In this project, the branching hardware for the Otter CPU was made. The branch condition generator was created to check the condition given two different source registers. This then resulted in an output of whether or not the values were equal, less than, or less than unsigned. The branch address generator was created to decide where to branch when a branch condition is met. In other words, this unit would modify the program counter given the program counter, j type immediate, b type immediate, i type immediate, and the source register 1. These modules were then formally tested using the SymbiYosys suite and proved to be functional.

2 Structural Design

2.1 Arithmetic Logic Unit Elaborated Design

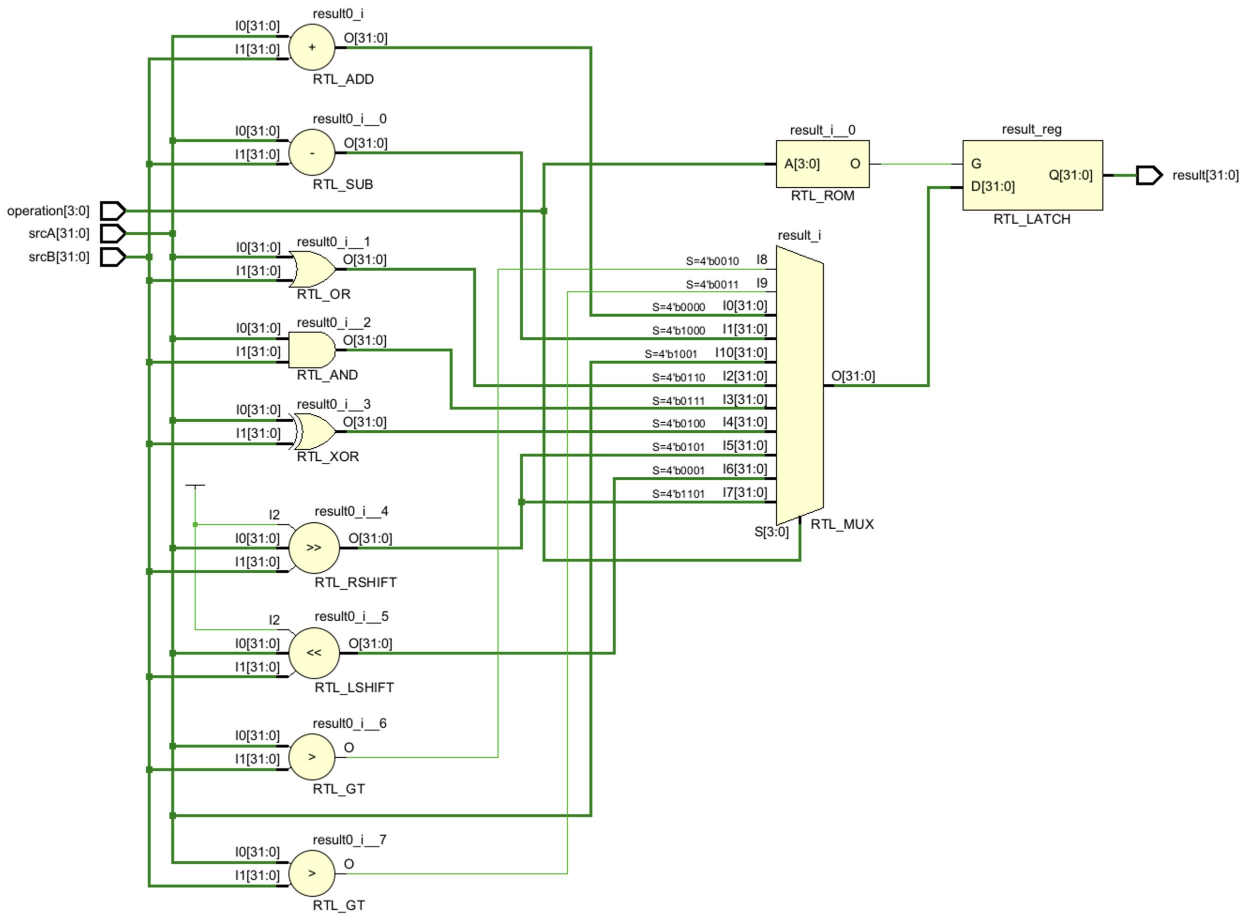


Figure 1: Arithmetic Logic Unit Elaborated Design

2.2 Immediate Generator Elaborated Design

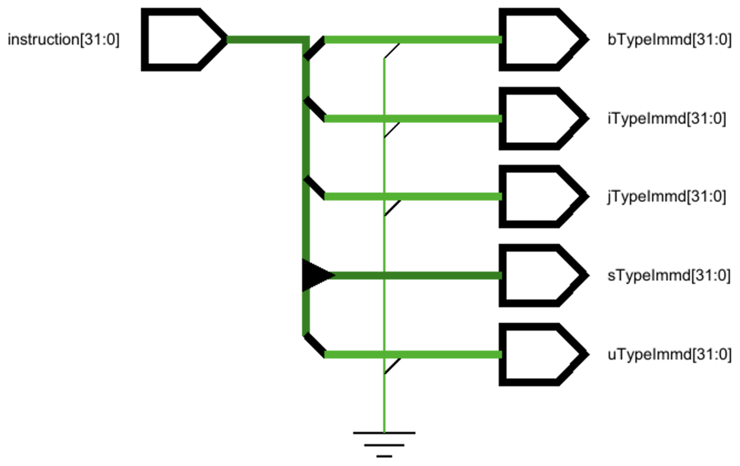
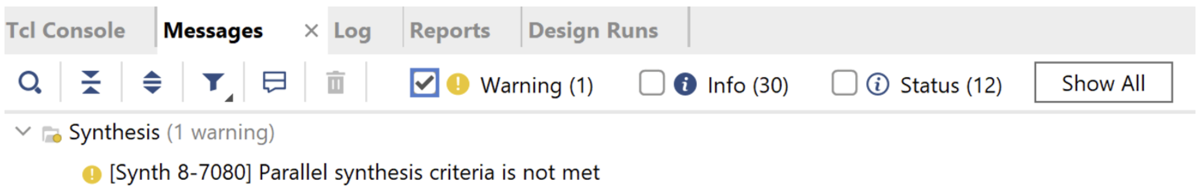


Figure 2: Immediate Generator Elaborated Design

2.3 Otter Memory Module Signal Table

3 Synthesis Warnings

3.1 Arithmetic Logic Unit Synthesis Warnings

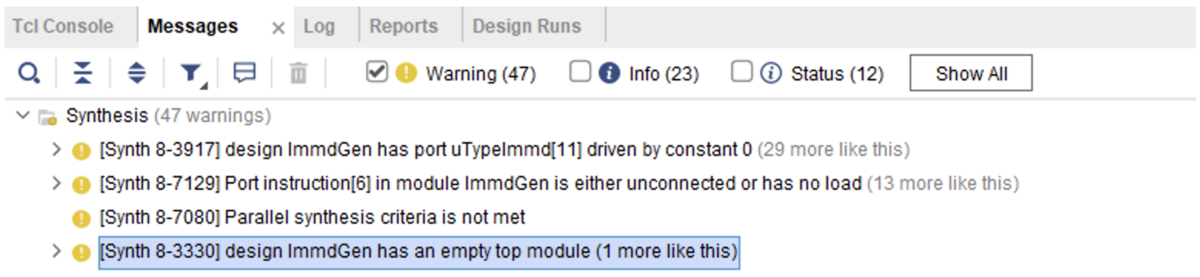


Synthesis (1 warning)

[Synth 8-7080] Parallel synthesis criteria is not met

Figure 3: Arithmetic Logic Unit Synthesis Warnings

3.2 Immediate Generator Synthesis Warnings



```
Synthesis (47 warnings)
  [Synth 8-3917] design ImmdGen has port uTypeImmd[11] driven by constant 0 (29 more like this)
  [Synth 8-7129] Port instruction[6] in module ImmdGen is either unconnected or has no load (13 more like this)
  [Synth 8-7080] Parallel synthesis criteria is not met
  [Synth 8-3330] design ImmdGen has an empty top module (1 more like this)
```

Figure 4: Immediate Generator Synthesis Warnings

The three warning messages that came up in figure **??** are not concerning. Warning `Synth 8-3917` references a constant driving the output of the module. This is expected because we are padding the immediate values with zeros which are constants. Warning `Synth 8-7129` is also not a concern as to maintain the same terminology as the RiscV specification for the immediate values, the first 7 bits of the input to the immediate generator are not used. Warning `Synth 8-3330` is also not a concern since this module is simply parsing an immediate value and is not performing logic. Finally, warning `Synth 7080` is not a concern as we are not performing parallel synthesis.

4 Verification

4.1 Arithmetic Logic Unit Testbench Coverage

The testbench for the ALU was designed to test many of the edge cases of the ALU as well as make sure that every operation was tested in a way where all of the bits in and out were either 1 or 0 for at least one operation. The testbench was able to verify that the ALU worked properly and passed all of the test cases.

4.2 Immediate Generator Testbench Coverage

1. **U Type** For testing the U Type immediate, several test cases were made in RARS to test the maximum and minimum values of the immediate as well as make sure every bit could be a 1 or a 0.
2. **I Type** For testing the I Type immediate, several test cases were made in RARS to test the maximum and minimum values of the immediate as well as make sure every bit could be a 1 or a 0.
3. **S Type** For testing the S Type immediate, several test cases were made in RARS to test the maximum and minimum values of the immediate as well as make sure every bit could be a 1 or a 0.
4. **S Type** For testing the S Type immediately, several values were tested as well as

changing the sign extension of some values to make sure the sign extension was working properly.

5. **B Type** For the B Type immediate, a similar approach was taken as with the S Type where the sign extension was tested as well as several values.

4.3 Arithmetic Logic Unit Testbench Code

Listing 1: System Verilod Testbench Code for Arithmetic Logic Unit

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/02/2024 08:04:37 PM
// Module Name: ALU_TB
// Project Name: Arithmetic Logic Unit
// Target Devices: Basys 3
// Description: This is a test bench for the ALU module
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ALU_TB();
    // Inputs
    logic [31:0] srcA_TB;          // 32-bit input A
    logic [31:0] srcB_TB;          // 32-bit input B
    logic [3:0] operation_TB;      // 5-bit function code

    // Outputs
    logic [31:0] result_TB;         // 32-bit result

    // Testing Array
    const int testArraySize = 99;
    logic [31:0] testArray [0:98];

    // Instantiate the Unit Under Test (UUT)
    ALU uut (
        .srcA(srcA_TB),
        .srcB(srcB_TB),
        .operation(operation_TB),
        .result(result_TB)
    );

    initial begin
    // Read in mem file with test cases
    string filename = "aluVerification.mem";
    $readmemh(filename, testArray);

        // Iterate through test cases
        for (int i = 0; i < testArraySize; i +=3) begin
            // Set test parameters iterating in chunks of 3
            operation_TB = testArray[i][3:0];   // 4-bit function code is first line
            srcA_TB = testArray[i+1];           // srcA is second line
            srcB_TB = testArray[i+2];           // srcB is third line
            #10 // Allow signals to propagate

            // Display values for debugging and verification
            // Debugging scheme 1
            // $display("srcA_TB: %h", srcA_TB);
            // $display("srcB_TB: %h", srcB_TB);
            // $display("operation_TB: %h", operation_TB);
            // $display("result_TB: %h", result_TB);

            // Debugging scheme 2
            $display("srcA: %h | srcB: %h | operation: %h | result: %h", srcA_TB, srcB_TB,
                operation_TB, result_TB);

            // Debugging scheme 3
            // $display("%h", result_TB);

        end

    $finish;
    end
endmodule
```

Listing 2: System Verilog Arithmetic Logic Unit Test Case File

```
00000000
A50F96C3
5AF0693C
00000000
84105F21
7B105FDE
00000000
FFFFFFFF
00000001
00000008
00000000
00000001
00000008
AA806355
550162AA
00000008
550162AA
AA806355
00000007
A55A00FF
5A5AFFFF
00000007
C3C3F966
FF669F5A
00000006
9A9AC300
65A3CC0F
00000006
C3C3F966
FF669F5A
00000004
AA5500FF
5AA50FF0
00000004
A5A56C6C
FF00C6FF
00000005
805A6CF3
00000010
00000005
705A6CF3
00000005
00000005
805A6CF3
00000000
00000005
805A6CF3
00000100
00000001
805A6CF3
00000010
00000001
805A6CF3
00000005
00000001
805A6CF3
00000100
0000000d
805A6CF3
00000010
0000000d
705A6CF3
00000005
0000000d
805A6CF3
00000000
0000000d
805A6CF3
00000100
```

```
00000002
7FFFFFFF
80000000
00000002
80000000
00000001
00000002
00000000
00000000
00000002
55555555
55555555
00000003
7FFFFFFF
80000000
00000003
80000000
00000001
00000003
00000000
00000000
00000003
55AA55AA
55AA55AA
00000009
01234567
76543210
00000009
FEDCBA98
89ABCDEF
```

4.4 Immediate Generator Testbench Code

Listing 3: System Verilog Immediate Generator Testbench

```
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/02/2024 05:45:57 PM
// Module Name: ImmdGen_TB
// Project Name: Immediate Generator Test Bench
// Target Devices: Basys 3
// Description: This is the test bench for the immediate generator module
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ImmdGen_TB();
    // Inputs
    logic [31:0] instruction_TB; // 32-bit instruction

    // Outputs
    logic [31:0] uTypeImmd_TB;  // U type immediate
    logic [31:0] iTypeImmd_TB;  // I type immediate
    logic [31:0] sTypeImmd_TB;  // S type immediate
    logic [31:0] jTypeImmd_TB;  // J type immediate
    logic [31:0] bTypeImmd_TB;  // B type immediate

    // Testing Logic
    logic pass = 1; // 1 = pass, 0 = fail
    string testType; // u, i, s, j, b

    // Testing Array
    const int testArraySize = 20; // Set to number of test cases
    logic [31:0] testArray [0:19];



    // Instantiate the Unit Under Test (UUT)
    ImmdGen uut (
        .instruction(instruction_TB),
        .uTypeImmd(uTypeImmd_TB),
        .iTypeImmd(iTypeImmd_TB),
        .sTypeImmd(sTypeImmd_TB),
        .jTypeImmd(jTypeImmd_TB),
        .bTypeImmd(bTypeImmd_TB)
    );

    initial begin
        // Read in mem file with test cases
        $readmemb("ImmdGenVerification.mem", testArray);

        // Iterate through test cases
        for (int i = 0; i < testArraySize; i++) begin
            case(i)
               0: begin
                    $display ("\nU Type Immediate\n");
                    testType = "u";
                end
                4: begin
                    $display ("\nI Type Immediate\n");
                    testType = "i";
                end
                8: begin
```

```
                    $display ("\nS Type Immediate\n");
                    testType = "s";
                end
                12: begin
                    $display ("\nJ Type Immediate\n");
                    testType = "j";
                end
                16: begin
                    $display ("\nB Type Immediate\n");
                    testType = "b";
                end
               default;
            endcase

            // Display values for debugging and verification
            $display ("Instruction: %h", testArray[i]);
            instruction_TB = testArray[i];
            #10;
            case (testType)
                "u": $display ("uTypeImmd: %h", uTypeImmd_TB);
                "i": $display ("iTypeImmd: %h", iTypeImmd_TB);
                "s": $display ("sTypeImmd: %h", sTypeImmd_TB);
                "j": $display ("jTypeImmd: %h", jTypeImmd_TB);
                "b": $display ("bTypeImmd: %h", bTypeImmd_TB);
            endcase
        end
        $finish;
    end
endmodule
```

Listing 4: System Verilog Immediate Generator Testbench

```
00000000000000000000001010110111
10101010101010101010001100110111
01010101010101010101001110110111
11111111111111111111111000110111
00000000000000111111001010010011
00101010101000101111001100010011
01010101010100110111001110010011
01111111111111100111111000010011
00000000010100101000000000100011
00101010010100101000010100100011
01010100010100101000101010100011
01111110010100101000111110100011
11111101000111111111000001101111
01111011101111111111000001101111
11111110100111111111000001101111
11111111010111111111000001101111
11111100011100111101000011100011
01111100011100111101011011100011
11111100011100111101110011100011
11111110011100111101001011100011
```

4.5 Arithmetic Logic Unit Testbench Output

Running this testbench produced the following output in the TCL code console:

Table 2: Flow Chart 1 Test Cases

ALU FUN	ALU_SEL	A	B	Output
ADD	0000	0xA50F96C3	0x5AF0693C	0xffffffff
	0000	0x84105F21	0x7B105FDE	0xff20beff
	0000	0xFFFFFFFF	0x00000001	0x00000000
SUB	1000	0x00000000	0x00000001	0xffffffff
	1000	0xAA806355	0x550162AA	0x557f00ab
	1000	0x550162AA	0xAA806355	0xaa80ff55
AND	0111	0xA55A00FF	0x5A5AFFFF	0x005a00ff
	0111	0xC3C3F966	0xFF669F5A	0xc3429942
OR	0110	0x9A9AC300	0x65A3CC0F	0xffbbcf0f
	0110	0xC3C3F966	0xFF669F5A	0xffe7ff7e
XOR	0100	0xAA5500FF	0x5AA50FF0	0xf0f00f0f
	0100	0xA5A56C6C	0xFF00C6FF	0x5aa5aa93
SRL	0101	0x805A6CF3	0x00000010	0x0000805a
	0101	0x705A6CF3	0x00000005	0x0382d367
	0101	0x805A6CF3	0x00000000	0x805a6cf3
	0101	0x805A6CF3	0x00000100	0x00000000
SLL	0001	0x805A6CF3	0x00000010	0x6cf30000
	0001	0x805A6CF3	0x00000005	0x0b4d9e60
	0001	0x805A6CF3	0x00000100	0x00000000
SRA	1101	0x805A6CF3	0x00000010	0x0000805a
	1101	0x705A6CF3	0x00000005	0x0382d367
	1101	0x805A6CF3	0x00000000	0x805a6cf3
	1101	0x805A6CF3	0x00000100	0x00000000
SLT	0010	0x7FFFFFFF	0x80000000	0x00000000
	0010	0x80000000	0x00000001	0x00000001
	0010	0x00000000	0x00000000	0x00000000
	0010	0x55555555	0x55555555	0x00000000
SLTU	0011	0x7FFFFFFF	0x80000000	0x00000001
	0011	0x80000000	0x00000001	0x00000000
	0011	0x00000000	0x00000000	0x00000000
	0011	0x55AA55AA	0x55AA55AA	0x00000000
LUI COPY	1001	0x01234567	0x76543210	0x01234567
	1001	0xFEDCBA98	0x89ABCDEF	0xfedcba98

4.6 Immediate Generator Testbench Output

Listing 5: Verilog Code for Arithmetic Logic Unit

```
U Type Immediate

Instruction: 000002b7
uTypeImmd: 00000000
Instruction: aaaaa337
uTypeImmd: aaaaa000
Instruction: 555553b7
uTypeImmd: 55555000
Instruction: fffffe37
uTypeImmd: fffff000

I Type Immediate

Instruction: 0003f293
iTypeImmd: 00000000
Instruction: 2aa2f313
iTypeImmd: 000002aa
Instruction: 55537393
iTypeImmd: 00000555
Instruction: 7ffe7e13
iTypeImmd: 000007ff

S Type Immediate

Instruction: 00528023
sTypeImmd: 00000000
Instruction: 2a528523
sTypeImmd: 000002aa
Instruction: 54528aa3
sTypeImmd: 00000555
Instruction: 7e528fa3
sTypeImmd: 000007ff

J Type Immediate

Instruction: fd1ff06f
jTypeImmd: ffffffd0
Instruction: 7ddff06f
jTypeImmd: 000fffdc
Instruction: fe9ff06f
jTypeImmd: ffffffe8
Instruction: ff5ff06f
jTypeImmd: fffffff4

B Type Immediate

Instruction: fc73d0e3
bTypeImmd: ffffffc0
Instruction: 7c73d6e3
bTypeImmd: 00000fcc
Instruction: fc73dce3
bTypeImmd: ffffffd8
Instruction: fe73d2e3
bTypeImmd: ffffffe4
```

4.7 Simulation Results

Running this testbench produced the following simulation results:

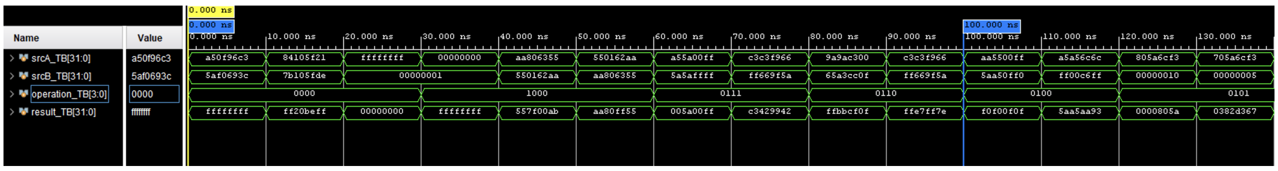


Figure 5: Arithmetic Logic Unit Simulation 0ns - 100ns

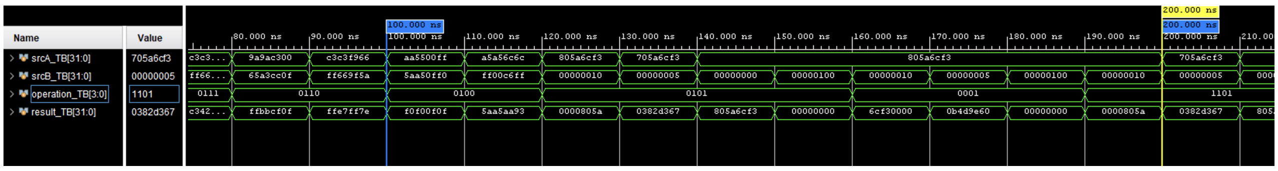


Figure 6: Arithmetic Logic Unit Simulation 100ns - 200ns

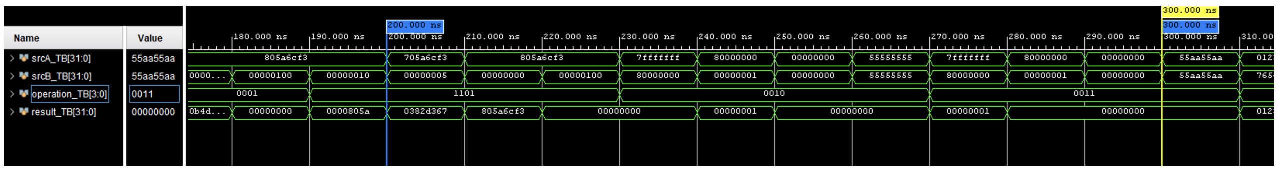


Figure 7: Arithmetic Logic Unit Simulation 200ns - 300ns

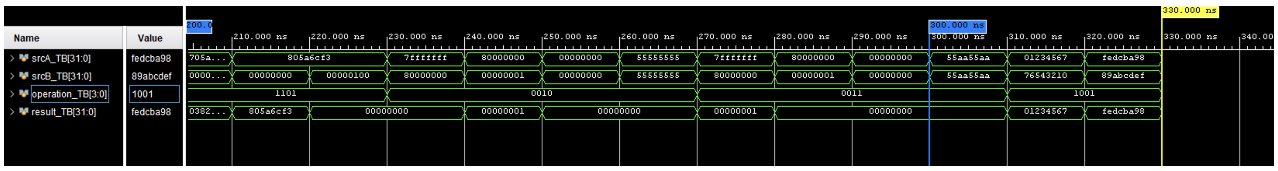


Figure 8: Aritmetic Logic Unit Simulation 300ns - 330ns

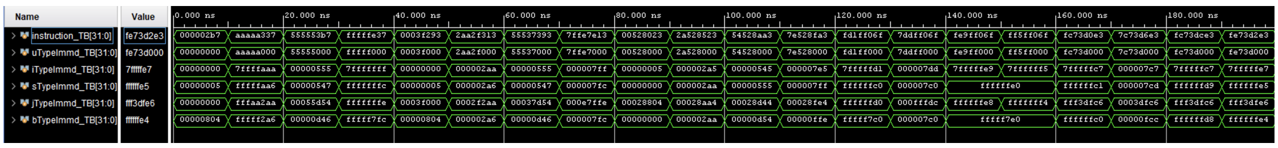


Figure 9: Immediate Generator Simulation 0ns - 200ns

5 Source Code

5.1 Arithmetic Logic Unit

Listing 6: Verilog Code for Arithmetic Logic Unit

```verilog
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal POly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/02/2024 11:47:05 AM
// Module Name: ImmdGen
// Project Name: Arithmetic Logic Unit
// Target Devices: Basys 3
// Description: This module is used to perform arithmetic on two inputs
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////

module ALU(
    // Inputs
    input [31:0] srcA,          // 32-bit input A
    input [31:0] srcB,          // 32-bit input B
    input [3:0] operation,      // 5-bit function code

    // Outputs
    output logic [31:0] result  // 32-bit result
    );

always_comb begin
    case (operation)
        4'b0000: result = srcA + srcB;                        // ADD: Add
        4'b1000: result = srcA - srcB;                        // SUB: Subtract
        4'b0110: result = srcA | srcB;                        // OR: or the two inputs
        4'b0111: result = srcA & srcB;                        // AND: and the two inputs
        4'b0100: result = srcA ^ srcB;                        // XOR: xor the two inputs
        4'b0101: result = srcA >> srcB;                       // SRL: logical shift left
        4'b0001: result = srcA << srcB;                       // SLL: logical shift right
        4'b1101: result = srcA >>> srcB;                      // SRA: shift right arithmetic
        4'b0010: result = srcA > srcB;                        // SLT: set less than
        4'b0011: result = $signed(srcA) > $signed(srcB);     // SLTU: set less than or equal
        4'b1001: result = srcA;                               // LUI-COPY: copy srcA to result
        default: result = 32'b0;                              // default case should not be reached
    endcase
end

endmodule
```

5.2 Immediate Generator

Listing 7: Verilog Code for Immediate Generator

```verilog
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/02/2024 11:47:05 AM
// Module Name: ImmdGen
// Project Name: Arithmetic Logic Unit and Immediate Generator
// Target Devices: Basys 3
// Description: This module is used to generate the immediate values for the Otter
// CPU.
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ImmdGen(
    // Inputs
    input [31:0] instruction, // 32-bit instruction

    // Outputs
    output logic [31:0] uTypeImmd,
    output logic [31:0] iTypeImmd,
    output logic [31:0] sTypeImmd,
    output logic [31:0] jTypeImmd,
    output logic [31:0] bTypeImmd
    );

    // U type immediate generation
    assign uTypeImmd = {instruction[31:12], 12'b0};

    // I type immediate generation
    assign iTypeImmd = {{20{instruction[31]}}, instruction[30:20]};

    // S type immediate generation
    assign sTypeImmd = {{21{instruction[31]}}, instruction[30:25], instruction[11:7]};

    // B type immediate generation
    assign bTypeImmd = {{20{instruction[31]}}, instruction[7], instruction[30:25], instruction
        [11:8], 1'b0};

    // J type immediate generation
    assign jTypeImmd = {{12{instruction[31]}}, instruction[19:12], instruction[20], instruction
        [30:21], 1'b0};
endmodule
```

6 Conclusion

The arithmetic logic unit is the heart of the Otter processor and performs all of the actions as described by the RiscV specification. The immediate generator is equally important and parses all of the immediate values that are frequently fed into the ALU. In this project, an ALU and immediate generator were created and tested. Through the verification process, both modules were tested and passed all test cases showing that there is reasonable confidence that the modules will work as expected in the Otter processor. All code for this assignment can be found here.