LAB REPORT
CSE4010 Computer Architecture
Instructor: Lawrence Orijuela

Name:__Cameron Harling____________________ SCORE: ___/30

Student ID: __007735059_____________________ DUE: __10/29/2024__

LAB: 3

Report

How can addition and subtraction of large numbers be performed by computers of they

are only aware of numbers 0 and 1?

Addition and subtraction of large numbers are performed by using the binary representation of the numbers and are sent through a series of half adders and half subtractors. XOR gates are used to get the sum/difference and AND gates are used to find the carry/borrow.

<u>Source Code for Parts A and B</u>

Part A

fullAdder.v

```verilog
// Simulates a half adder
module halfAdder(op1, op2, sum, carry); // Create module called halfAdder

    // inputs and outputs
    input op1, op2;
    output sum, carry;

    assign sum = op1 ^ op2;    // op1 XOR op2
    assign carry = op1 & op2; // op1 AND op2

endmodule // End module halfAdder

// Simulates a full adder
module fullAdder(A, B, carryIn, sum, carryOut);  // Create module called fullAdder

    // inputs and outputs
    input A, B, carryIn;
    output sum, carryOut;

    // intermediary wires
    wire c, d, e, f;

    halfAdder u1(A, B, c, d);        // XOR result assigned to c, AND result assigned to d
    halfAdder u2(carryIn, c, e, f);  // XOR result assigned to e, AND result assigned to f
    assign carryOut = f | d; // f OR d result assigned to carryOut

    assign sum = e;          // Assign e to sum


endmodule  // End module fullAdder
```

fullAdder _tb.v

```
`timescale 1 ns / 1 ns  // Sets time interval of fullAdder

`include "fullAdder.v"  // Links fullAdder.v to this file


module fullAdder_tb; // Create module called fullAdder_tb


    reg A, B, carryIn;   // Input wires A, B, and carryIn

    wire sum, carryOut;  // Output wires sum and carryOut


    fullAdder uut(A, B, carryIn, sum, carryOut); // Instantiate fullAdder as uut


    initial begin // Starts clock


        $dumpfile("fullAdder_tb.vcd"); // Sets file for output of the test

        $dumpvars(0, fullAdder_tb);


        {A, B, carryIn} = 3'd0; #20; // Set A = 0, B = 0, and carryIn = 0. Wait
20ns

        {A, B, carryIn} = 3'd1; #20; // Set A = 0, B = 0, and carryIn = 1. Wait
20ns

        {A, B, carryIn} = 3'd2; #20; // Set A = 0, B = 1, and carryIn = 0. Wait
20ns

        {A, B, carryIn} = 3'd3; #20; // Set A = 0, B = 1, and carryIn = 1. Wait
20ns
```

```
        {A, B, carryIn} = 3'd4; #20; // Set A = 1, B = 0, and carryIn = 0. Wait
20ns
        {A, B, carryIn} = 3'd5; #20; // Set A = 1, B = 0, and carryIn = 1. Wait
20ns
        {A, B, carryIn} = 3'd6; #20; // Set A = 1, B = 1, and carryIn = 0. Wait
20ns
        {A, B, carryIn} = 3'd7; #20; // Set A = 1, B = 1, and carryIn = 1. Wait
20ns
        $display("Finished additions!");
    end // Ends clock

endmodule  // End module fullAdder_tb
```

PartB

fullSubtractor.v

```
// Simulates a half subtractor
module halfSubtractor(op1, op2, difference, borrow); // Create module called
halfSubtractor

    // inputs and outputs
    input op1, op2;
    output difference, borrow;

    assign difference = op1 ^ op2;   // op1 XOR op2
    assign borrow = !op1 & op2; // NOT op1 AND op2

endmodule // End module halfSubtractor

// Simulates a full subtractor
module fullSubtractor(A, B, borrowIn, difference, borrowOut);  // Create module
called fullSubtractor
```

```verilog
    // inputs and outputs

    input A, B, borrowIn;

    output difference, borrowOut;


    // intermediary wires

    wire c, d, e, f;


    halfSubtractor u1(A, B, c, d);        // XOR result assigned to c, AND result
assigned to d

    halfSubtractor u2(borrowIn, c, e, f);  // XOR result assigned to e, AND
result assigned to f


    assign borrowOut = f | d; // f OR d result assigned to borrowOut

    assign difference = e;    // Assign e to difference


endmodule  // End module fullSubtractor
```

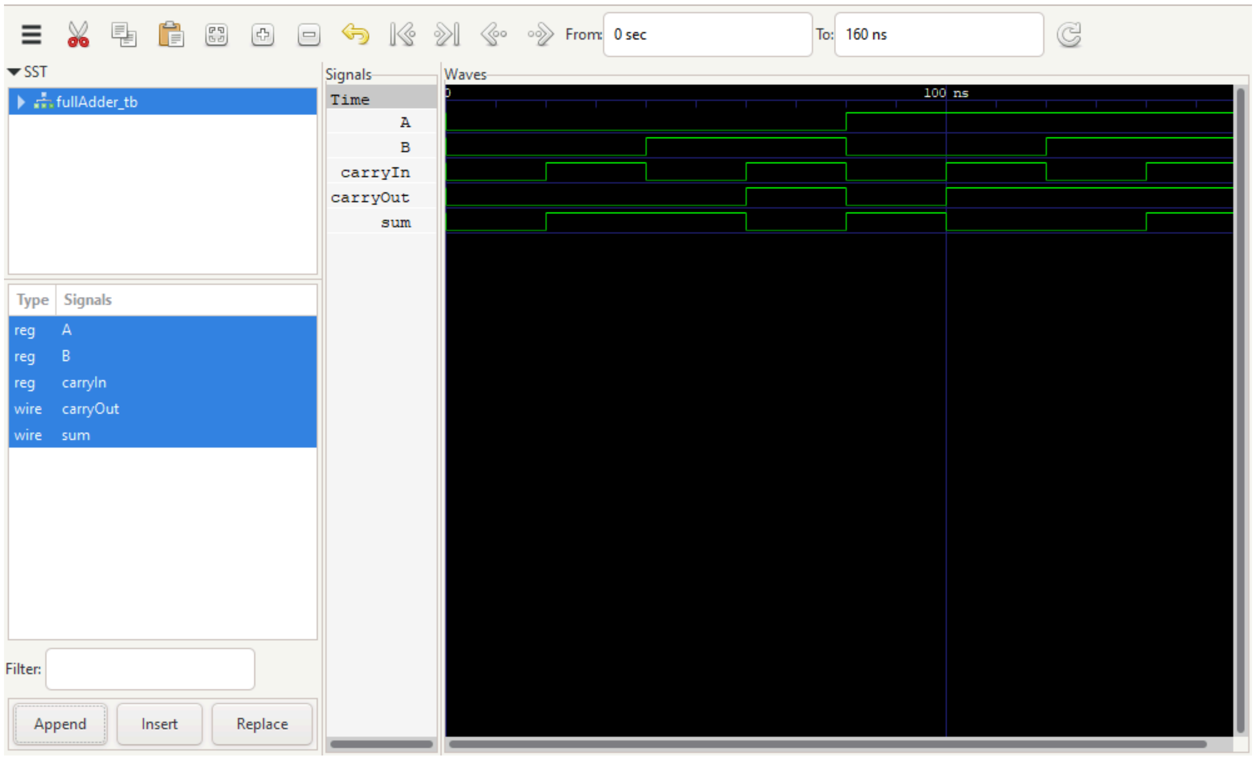
fullSubtractor _tb.v

```verilog
`timescale 1 ns / 1 ns  // Sets time interval of fullSubtractor

`include "fullSubtractor.v"  // Links fullSubtractor.v to this file


module fullSubtractor_tb; // Create module called fullSubtractor_tb


    reg A, B, borrowIn;   // Input wires A, B, and borrowIn

    wire difference, borrowOut;  // Output wires difference and borrowOut


    fullSubtractor uut(A, B, borrowIn, difference, borrowOut); // Instantiate
fullSubtractor as uut

```

```
    initial begin // Starts clock

        $dumpfile("fullSubtractor_tb.vcd"); // Sets file for output of the test
        $dumpvars(0, fullSubtractor_tb);

        {A, B, borrowIn} = 3'd0; #20; // Set A = 0, B = 0, and borrowIn = 0. Wait
20ns
        {A, B, borrowIn} = 3'd1; #20; // Set A = 0, B = 0, and borrowIn = 1. Wait
20ns
        {A, B, borrowIn} = 3'd2; #20; // Set A = 0, B = 1, and borrowIn = 0. Wait
20ns
        {A, B, borrowIn} = 3'd3; #20; // Set A = 0, B = 1, and borrowIn = 1. Wait
20ns
        {A, B, borrowIn} = 3'd4; #20; // Set A = 1, B = 0, and borrowIn = 0. Wait
20ns
        {A, B, borrowIn} = 3'd5; #20; // Set A = 1, B = 0, and borrowIn = 1. Wait
20ns
        {A, B, borrowIn} = 3'd6; #20; // Set A = 1, B = 1, and borrowIn = 0. Wait
20ns
        {A, B, borrowIn} = 3'd7; #20; // Set A = 1, B = 1, and borrowIn = 1. Wait
20ns
        $display("Finished subtractions!");
    end // Ends clock

endmodule  // End module fullSubtractor_tb
```

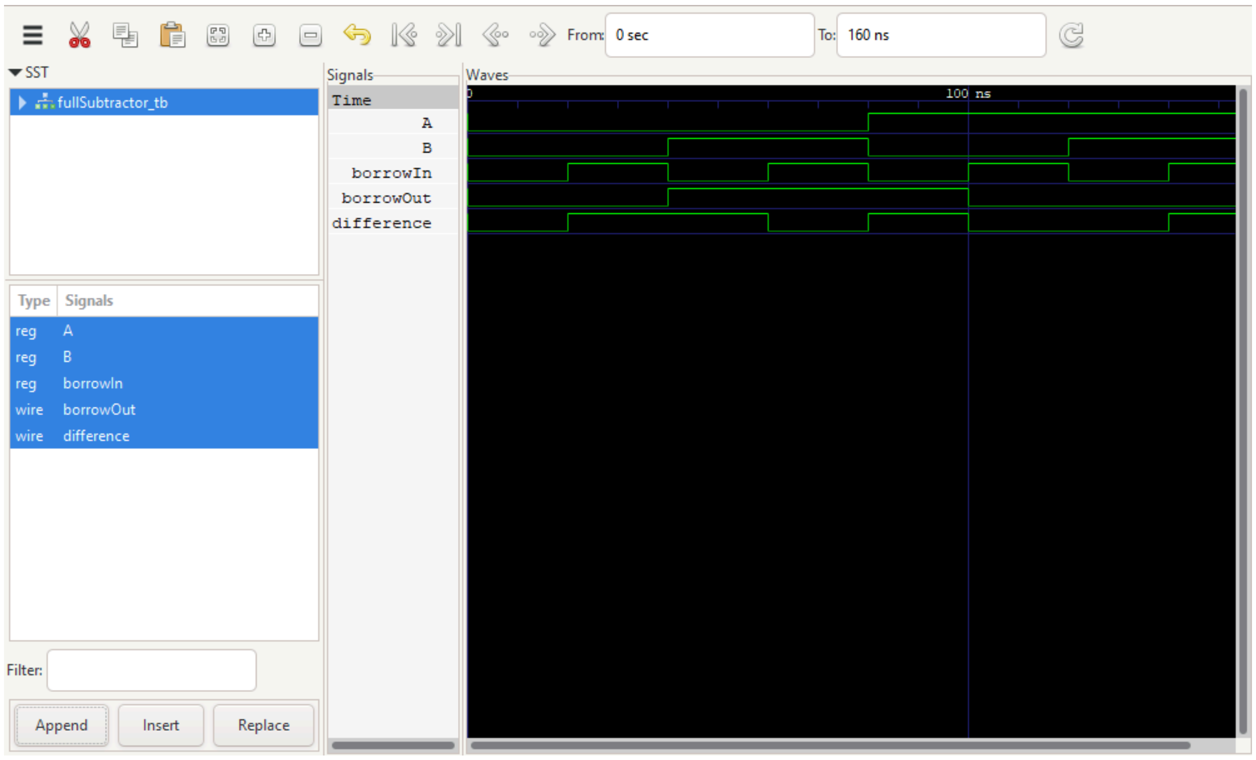
Screenshots for Parts A and B

Part A



Type	Signals
reg	A
reg	B
reg	carryIn
wire	carryOut
wire	sum

Part B



Type	Signals
reg	A
reg	B
reg	borrowIn
wire	borrowOut
wire	difference