Lab Report
CSE4010 Computer Architecture

Name: Zachary A. Hampton **Score:** /10
Student ID: 008339494 **Due:** 02-23-2025
Lab: Lab 3 - Full Adder and Full Subtractor Implementation

Report

- How can addition and subtraction of large numbers be performed by computers if they are only aware of numbers 0 and 1?
  - Computers perform arithmetic operations on large numbers by breaking them down into binary digits and processing them one bit at a time. For multi-digit numbers:
    - Addition is performed using full adders in cascade, where each full adder processes one bit position and handles the carry from the previous stage
    - Subtraction is similarly performed using full subtractors in cascade, where each subtractor handles one bit position and processes the borrow from the previous stage
    - The carry/borrow propagation between stages allows for handling numbers of any size
  - This binary arithmetic is implemented using combinational circuits that:
    - Process bits from least significant to most significant position
    - Use carry/borrow propagation for managing digit transitions
    - Can be extended to any number of bits by cascading multiple stages

Source Code

fullAdder.v

```
/*
 * File: fullAdder.v
 * Description: Implements a 1-bit full adder using half adders
 * A full adder adds three bits (A, B, and carry-in) and produces sum and carry-out
 * A full adder is built from two half adders and an OR gate;
 * The first half adder adds A and B and produces a sum and a carry
 * The second half adder adds the carry from the first half adder to the carry-in
 * The OR gate adds the carries from the two half adders to produce the final carry-out
 */

// Half Adder Module - Adds two bits and produces sum and carry
module halfAdder(op1, op2, sum, carry);

    // Inputs - two 1-bit operands
    input op1, op2;
    // Outputs - sum and carry bits
    output sum, carry;

    // XOR operation - sum is 1 if either op1 or op2 is 1, but not both
    assign sum = op1 ^ op2; // ^ is the xor operator
    // AND operation - carry is 1 only if both op1 and op2 are 1
    assign carry = op1 & op2; // & is the and operator

endmodule

// Full Adder Module - Adds three bits (A, B, carry-in) and produces sum and carry-out
module fullAdder(A, B, Cin, sum, Cout);

    // Input ports
    input A, B, Cin;  // A and B are the bits to add, carryIn is the carry from previous addition
    // Output ports
    output sum, Cout; // sum is the result bit, carryOut propagates to next addition

    // Internal connections - wires to connect the half adders
    wire c;  // Sum output from first half adder
    wire d;  // Carry output from first half adder
    wire e;  // Sum output from second half adder
    wire f;  // Carry output from second half adder

```

```
40      // First half adder - adds A and B
41      halfAdder u1(A, B, c, d);
42      // Second half adder - adds carryIn and the sum from first
           half adder
43      halfAdder u2(Cin, c, e, f);
44
45      // Final carry out is OR of both half adder carries
46      assign Cout = f | d;  // | is the OR operator
47      // Final sum is the sum output from second half adder
48      assign sum = e;
49
50  endmodule
```

fullAdder_tb.v

```
/*
 * File: fullAdder_tb.v
 * Description: Testbench for the 1-bit full adder
 * Tests all possible input combinations (0-7) for A, B, carryIn
 * The testbench uses a loop to iterate through all possible input combinations
 * It displays the results of each test case
 */

// Define timescale for simulation: 1 nanosecond time unit, 1 nanosecond precision
`timescale 1 ns / 1 ns
// Include the module to be tested
`include "fullAdder.v"

// Testbench module - no external ports
module fullAdder_tb;

// Test inputs - declared as registers since they will be driven
reg A, B, Cin;
// Test outputs - declared as wires since they will be monitored
wire sum, Cout;

// Instantiate the Unit Under Test (UUT) - connect to test signals
fullAdder uut(A, B, Cin, sum, Cout);

// Test sequence
initial begin
    // Generate VCD file for waveform viewing
    $dumpfile("fullAdder_tb.vcd");
    // Dump variables from the testbench (0 means all variables)
    $dumpvars(0, fullAdder_tb);

    // Test case 0: A=0, B=0, Cin=0
    {A, B, Cin} = 3'd0; #20;
    // Test case 1: A=0, B=0, Cin=1
    {A, B, Cin} = 3'd1; #20;
    // Test case 2: A=0, B=1, Cin=0
    {A, B, Cin} = 3'd2; #20;
    // Test case 3: A=0, B=1, Cin=1
    {A, B, Cin} = 3'd3; #20;
    // Test case 4: A=1, B=0, Cin=0
    {A, B, Cin} = 3'd4; #20;
    // Test case 5: A=1, B=0, Cin=1
    {A, B, Cin} = 3'd5; #20;
    // Test case 6: A=1, B=1, Cin=0
    {A, B, Cin} = 3'd6; #20;
    // Test case 7: A=1, B=1, Cin=1
```

```
47      {A, B, Cin} = 3'd7; #20;
48      // Display a message to indicate completion of test cases
49      $display("Finished␣additions!");
50
51  end
52
53  endmodule
```

fullSubtractor.v

```
/*
 * File: fullSubtractor.v
 * Description: Implements a 1-bit full subtractor using half subtractors
 * A full subtractor subtracts three bits (A, B, and borrow-in) and produces difference and borrow-out
 * A full subtractor is built from two half subtractors and an OR gate;
 * The first half subtractor subtracts B from A and produces a difference and a borrow
 * The second half subtractor subtracts the borrow-in from the difference of the first half subtractor
 * The OR gate combines the borrows from the two half subtractors to produce the final borrow-out
 */

// Half Subtractor Module - Subtracts one bit from another and produces difference and borrow
module halfSubtractor(op1, op2, diff, borrow);

    // Inputs - two 1-bit operands
    input op1, op2;
    // Outputs - difference and borrow bits
    output diff, borrow;

    // XOR operation - difference is 1 if op1 and op2 are different
    assign diff = op1 ^ op2;

    // AND operation with NOT on op2 - borrow is 1 if op1=1 and op2=0
    assign borrow = op1 & !op2;

endmodule

// Full Subtractor Module - Subtracts three bits (A, B, borrow-in) and produces difference and borrow-out
module fullSubtractor(A, B, Bin, diff, Bout);

    // Input ports
    input A, B, Bin;  // A is minuend, B is subtrahend, Bin is borrow-in from previous subtraction
    // Output ports
    output diff, Bout; // diff is the difference bit, Bout propagates borrow to next subtraction

    // Internal connections - wires to connect the half subtractors
    wire c;  // Difference output from first half subtractor
```

```
37      wire d;  // Borrow output from first half subtractor
38      wire e;  // Difference output from second half subtractor (
            unused)
39      wire f;  // Borrow output from second half subtractor
40
41      // First half subtractor - subtracts B from A
42      halfSubtractor u1(A, B, c, d);
43
44      // Second half subtractor - subtracts Bin from the result of
            first half subtractor
45      halfSubtractor u2(c, Bin, diff, f);
46
47      // Final borrow out is OR of both half subtractor borrows
48      assign Bout = f | d;  // | is the OR operator
49
50  endmodule
```

fullSubtractor_tb.v

```
/*
 * File: fullSubtractor_tb.v
 * Description: Testbench for the 1-bit full subtractor
 * Tests all possible input combinations (0-7) for A, B, Bin
 * The testbench uses a loop to iterate through all possible input combinations
 * It displays the results of each test case
 */

// Define timescale for simulation: 1 nanosecond time unit, 1 nanosecond precision
`timescale 1 ns / 1 ns
// Include the module to be tested
`include "fullSubtractor.v"

// Testbench module - no external ports
module fullSubtractor_tb;

// Test inputs - declared as registers since they will be driven
reg A, B, Bin;
// Test outputs - declared as wires since they will be monitored
wire diff, Bout;

// Instantiate the Unit Under Test (UUT) - connect to test signals
fullSubtractor uut(A, B, Bin, diff, Bout);

// Test sequence
initial begin
    // Generate VCD file for waveform viewing
    $dumpfile("fullSubtractor_tb.vcd");
    // Dump variables from the testbench (0 means all variables)
    $dumpvars(0, fullSubtractor_tb);

    // Test case 0: A=0, B=0, Bin=0
    {A, B, Bin} = 3'd0; #20;
    // Test case 1: A=0, B=0, Bin=1
    {A, B, Bin} = 3'd1; #20;
    // Test case 2: A=0, B=1, Bin=0
    {A, B, Bin} = 3'd2; #20;
    // Test case 3: A=0, B=1, Bin=1
    {A, B, Bin} = 3'd3; #20;
    // Test case 4: A=1, B=0, Bin=0
    {A, B, Bin} = 3'd4; #20;
    // Test case 5: A=1, B=0, Bin=1
    {A, B, Bin} = 3'd5; #20;
    // Test case 6: A=1, B=1, Bin=0
    {A, B, Bin} = 3'd6; #20;
    // Test case 7: A=1, B=1, Bin=1
```

```
47      {A, B, Bin} = 3'd7; #20;
48      // Display a message to indicate completion of test cases
49      $display("Finished subtractions!");
50
51  end
52
53  endmodule
```

Screenshots

Part A - Full Adder Simulation

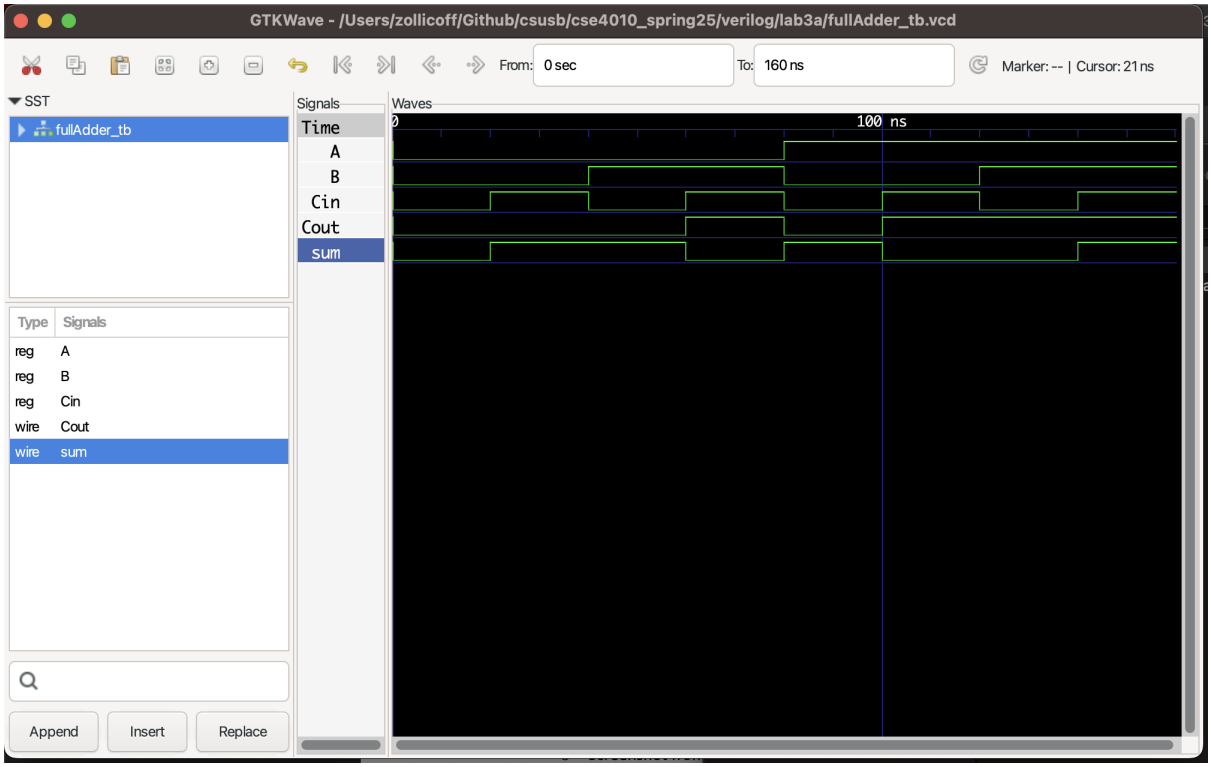

Figure 1: Full Adder Simulation Waveform showing all test cases with inputs A, B, Cin and outputs sum, Cout

Part B - Full Subtractor Simulation

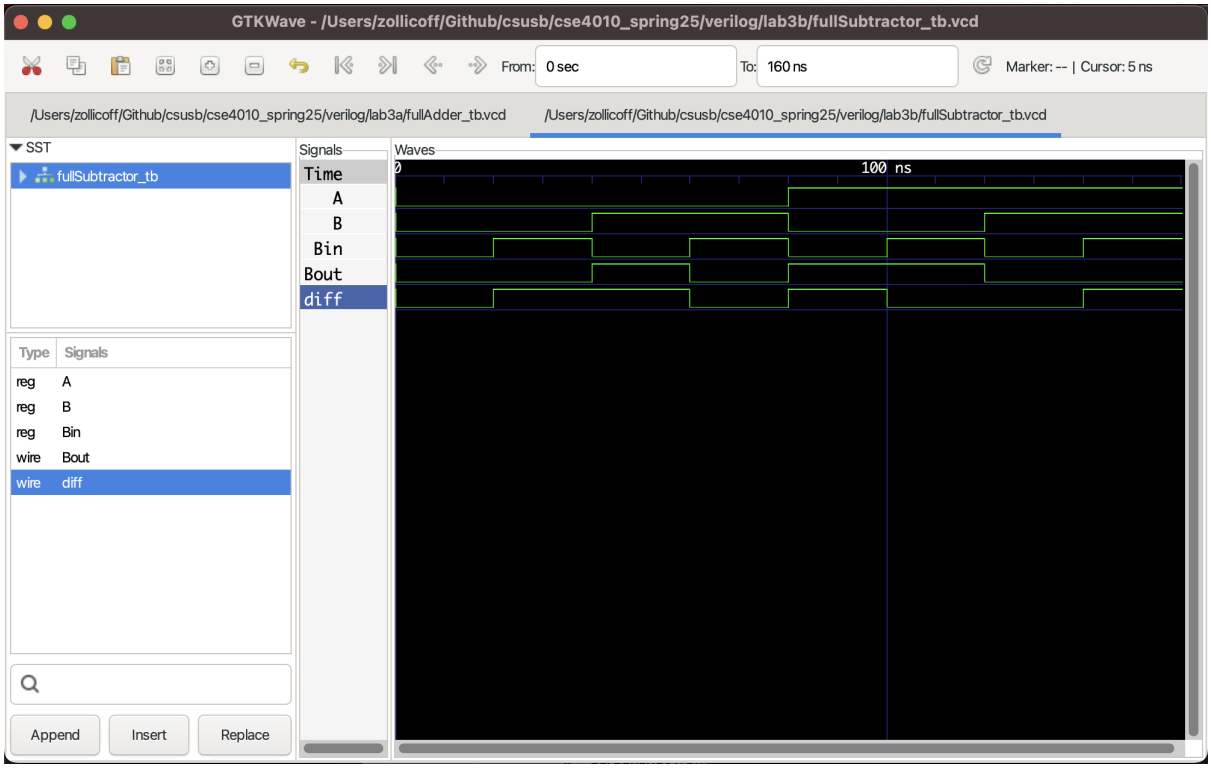

Figure 2: Full Subtractor Simulation Waveform showing all test cases with inputs A, B, Bin and outputs diff, Bout