

# Lab Report

## CSE4010 Computer Architecture

Name: Zachary A. Hampton

Score: \_\_\_\_\_/10

Student ID: 008339494

Due: 03-02-2025

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

```
1 /*
2  * File: fullAdder.v
3  * Description: Implements a 1-bit full adder using half adders
4  * A full adder adds three bits (A, B, and carry-in) and produces
   sum and carry-out
5  * A full adder is built from two half adders and an OR gate;
6  * The first half adder adds A and B and produces a sum and a
   carry
```

```

7  * The second half adder adds the carry from the first half adder
   to the carry-in
8  * The OR gate adds the carries from the two half adders to
   produce the final carry-out
9  */
10
11 // Half Adder Module - Adds two bits and produces sum and carry
12 module halfAdder(op1, op2, sum, carry);
13
14     // Inputs - two 1-bit operands
15     input op1, op2;
16     // Outputs - sum and carry bits
17     output sum, carry;
18
19     // XOR operation - sum is 1 if either op1 or op2 is 1, but
   not both
20     assign sum = op1 ^ op2; // ^ is the xor operator
21     // AND operation - carry is 1 only if both op1 and op2 are 1
22     assign carry = op1 & op2; // & is the and operator
23
24 endmodule
25
26 // Full Adder Module - Adds three bits (A, B, carry-in) and
   produces sum and carry-out
27 module fullAdder(A, B, Cin, sum, Cout);
28
29     // Input ports
30     input A, B, Cin; // A and B are the bits to add, carryIn is
   the carry from previous addition
31     // Output ports
32     output sum, Cout; // sum is the result bit, carryOut
   propagates to next addition
33
34     // Internal connections - wires to connect the half adders
35     wire c; // Sum output from first half adder
36     wire d; // Carry output from first half adder
37     wire e; // Sum output from second half adder
38     wire f; // Carry output from second half adder
39
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

```
1  /*
2  * File: fullAdder_tb.v
3  * Description: Testbench for the 1-bit full adder
4  * Tests all possible input combinations (0-7) for A, B, carryIn
5  * The testbench uses a loop to iterate through all possible
6  *   input combinations
7  * It displays the results of each test case
8  */
9  // Define timescale for simulation: 1 nanosecond time unit, 1
10 //   nanosecond precision
11 'timescale 1 ns / 1 ns
12 // Include the module to be tested
13 'include "fullAdder.v"
14
15 // Testbench module - no external ports
16 module fullAdder_tb;
17
18 // Test inputs - declared as registers since they will be driven
19 reg A, B, Cin;
20 // Test outputs - declared as wires since they will be monitored
21 wire sum, Cout;
22
23 // Instantiate the Unit Under Test (UUT) - connect to test
24 //   signals
25 fullAdder uut(A, B, Cin, sum, Cout);
26
27 // Test sequence
28 initial begin
29     // Generate VCD file for waveform viewing
30     $dumpfile("fullAdder_tb.vcd");
31     // Dump variables from the testbench (0 means all variables)
32     $dumpvars(0, fullAdder_tb);
33
34     // Test case 0: A=0, B=0, Cin=0
35     {A, B, Cin} = 3'd0; #20;
36     // Test case 1: A=0, B=0, Cin=1
37     {A, B, Cin} = 3'd1; #20;
38     // Test case 2: A=0, B=1, Cin=0
39     {A, B, Cin} = 3'd2; #20;
40     // Test case 3: A=0, B=1, Cin=1
41     {A, B, Cin} = 3'd3; #20;
42     // Test case 4: A=1, B=0, Cin=0
43     {A, B, Cin} = 3'd4; #20;
44     // Test case 5: A=1, B=0, Cin=1
45     {A, B, Cin} = 3'd5; #20;
46     // Test case 6: A=1, B=1, Cin=0
47     {A, B, Cin} = 3'd6; #20;
48     // 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

```

1  /*
2   * File: fullSubtractor.v
3   * Description: Implements a 1-bit full subtractor using half
4     subtractors
5   * A full subtractor subtracts three bits (A, B, and borrow-in)
6     and produces difference and borrow-out
7   * A full subtractor is built from two half subtractors and an OR
8     gate;
9   * The first half subtractor subtracts B from A and produces a
10     difference and a borrow
11   * The second half subtractor subtracts the borrow-in from the
12     difference of the first half subtractor
13   * The OR gate combines the borrows from the two half subtractors
14     to produce the final borrow-out
15   */
16
17 // Half Subtractor Module - Subtracts one bit from another and
18 // produces difference and borrow
19 module halfSubtractor(op1, op2, diff, borrow);
20
21     // Inputs - two 1-bit operands
22     input op1, op2;
23     // Outputs - difference and borrow bits
24     output diff, borrow;
25
26     // XOR operation - difference is 1 if op1 and op2 are
27     // different
28     assign diff = op1 ^ op2;
29
30     // AND operation with NOT on op2 - borrow is 1 if op1=1 and
31     // op2=0
32     assign borrow = op1 & !op2;
33
34 endmodule
35
36 // Full Subtractor Module - Subtracts three bits (A, B, borrow-in
37 // ) and produces difference and borrow-out
38 module fullSubtractor(A, B, Bin, diff, Bout);
39
40     // Input ports

```

```

31     input A, B, Bin; // A is minuend, B is subtrahend, Bin is
        borrow-in from previous subtraction
32     // Output ports
33     output diff, Bout; // diff is the difference bit, Bout
        propagates borrow to next subtraction
34
35     // Internal connections - wires to connect the half
        subtractors
36     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

```

1  /*
2  * File: fullSubtractor_tb.v
3  * Description: Testbench for the 1-bit full subtractor
4  * Tests all possible input combinations (0-7) for A, B, Bin
5  * The testbench uses a loop to iterate through all possible
        input combinations
6  * It displays the results of each test case
7  */
8
9  // Define timescale for simulation: 1 nanosecond time unit, 1
        nanosecond precision
10 `timescale 1 ns / 1 ns
11 // Include the module to be tested
12 `include "fullSubtractor.v"
13
14 // Testbench module - no external ports
15 module fullSubtractor_tb;
16
17 // Test inputs - declared as registers since they will be driven
18 reg A, B, Bin;
19 // Test outputs - declared as wires since they will be monitored
20 wire diff, Bout;

```

```

21
22 // Instantiate the Unit Under Test (UUT) - connect to test
    signals
23 fullSubtractor uut(A, B, Bin, diff, Bout);
24
25 // Test sequence
26 initial begin
27     // Generate VCD file for waveform viewing
28     $dumpfile("fullSubtractor_tb.vcd");
29     // Dump variables from the testbench (0 means all variables)
30     $dumpvars(0, fullSubtractor_tb);
31
32     // Test case 0: A=0, B=0, Bin=0
33     {A, B, Bin} = 3'd0; #20;
34     // Test case 1: A=0, B=0, Bin=1
35     {A, B, Bin} = 3'd1; #20;
36     // Test case 2: A=0, B=1, Bin=0
37     {A, B, Bin} = 3'd2; #20;
38     // Test case 3: A=0, B=1, Bin=1
39     {A, B, Bin} = 3'd3; #20;
40     // Test case 4: A=1, B=0, Bin=0
41     {A, B, Bin} = 3'd4; #20;
42     // Test case 5: A=1, B=0, Bin=1
43     {A, B, Bin} = 3'd5; #20;
44     // Test case 6: A=1, B=1, Bin=0
45     {A, B, Bin} = 3'd6; #20;
46     // 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

### Part B - Full Subtractor Simulation

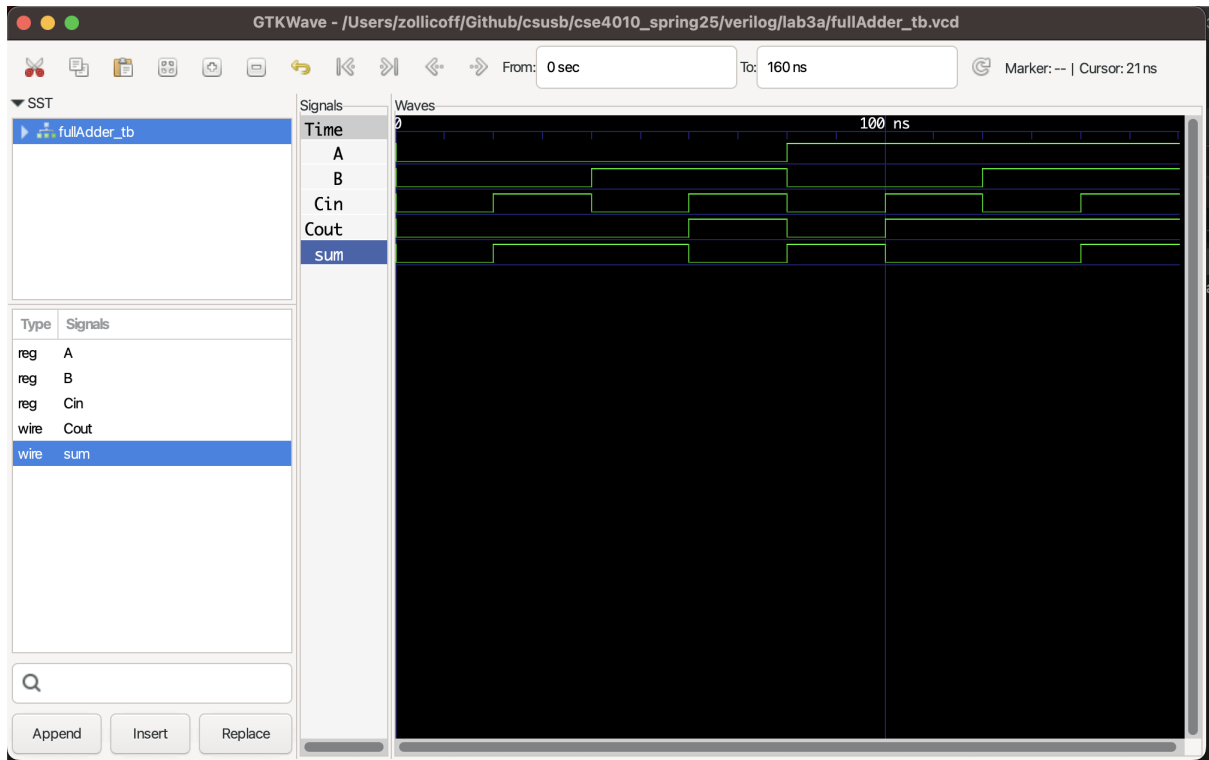


Figure 1: Full Adder Simulation Waveform showing all test cases with inputs A, B, Cin and outputs sum, Cout

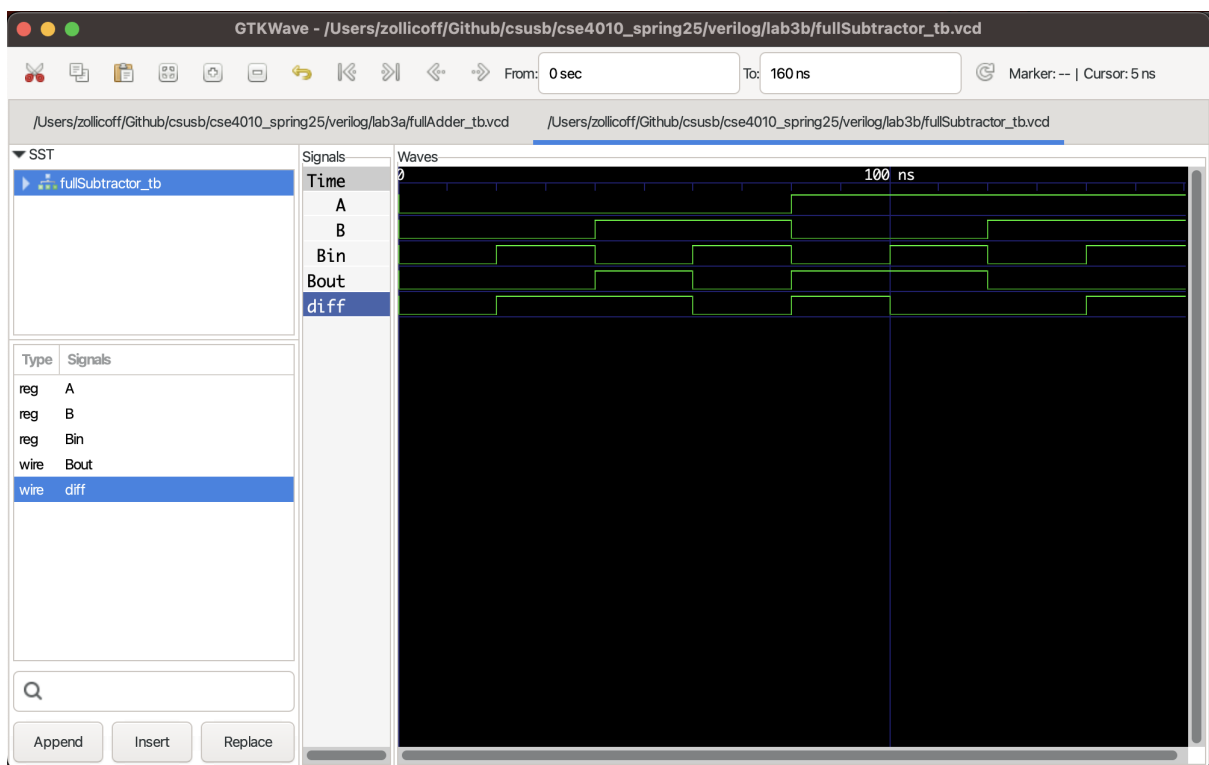


Figure 2: Full Subtractor Simulation Waveform showing all test cases with inputs A, B, Bin and outputs diff, Bout