**EE2026 Assignment**

**Design outline**

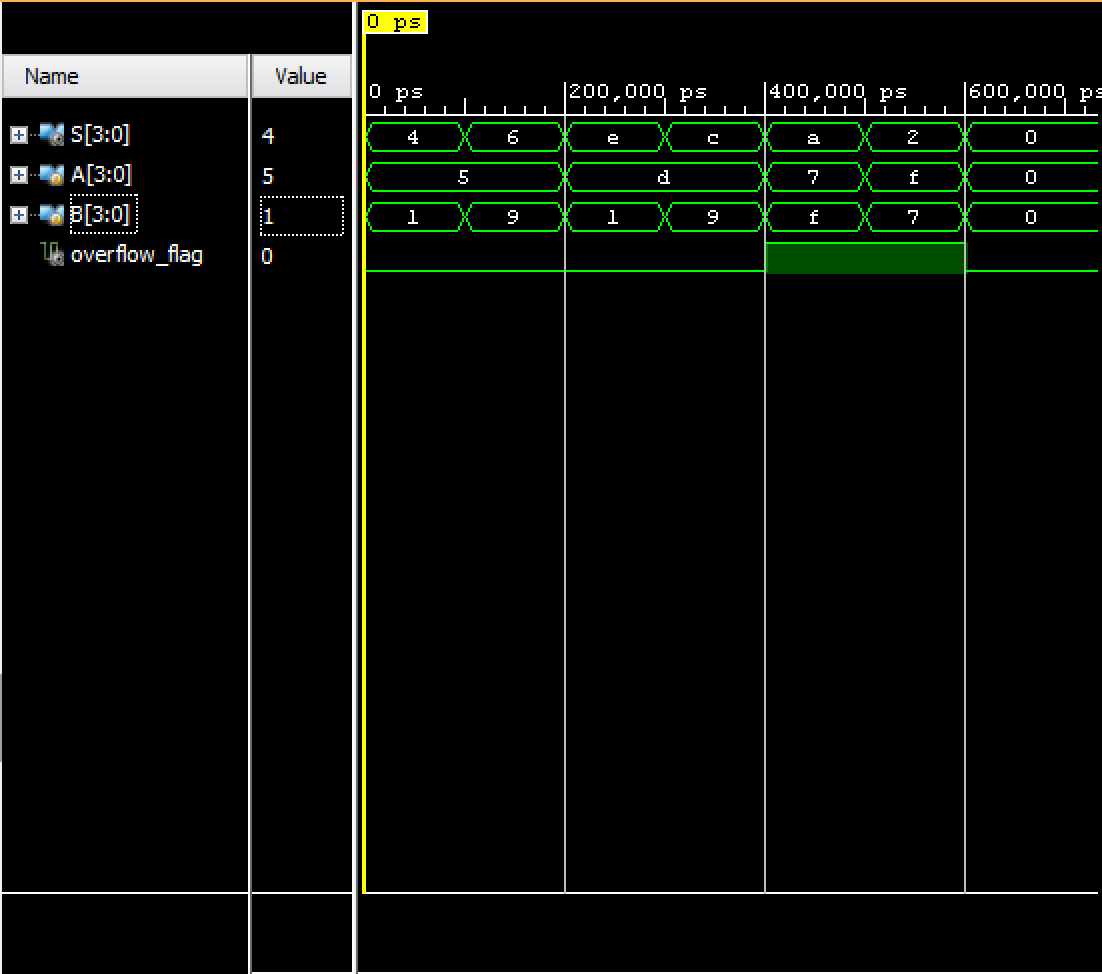
The assigned task was to construct a 4-bit subtractor using 2s complement and a 4-bit adder. We know that using a 2s complement representation of the negative numbers, we can simply add the values together using a 4-bit adder to get the result. We also know that the result will be in its normal form if it is positive, and in its 2s complement form if it is negative. Whether the number is positive or negative is given by the MSB (Most Significant Bit) of the sum. Additionally, we know that the range of values that this subtractor can operate on **and output** is -7 to +7. Thus, we realize that with certain input combinations, it is possible to go above the output range of the subtractor. Therein arises the need for an overflow flag to indicate the validity of the output of the subtractor.

My implementation of the subtractor intends to make it simpler for the user to use by utilizing a **4-bit signed magnitude representation** of the input and output values. In addition to that, as mentioned in the above section, an overflow flag is implemented. As such, the input and output testing criteria are slightly different for my implementation as compared to those provided in the lab manual.

**Results of simulation˜j**

The inputs of the tests are modified from the provided values such that they are converted to the 4-bit signed magnitude representation which is consistent with my intended implementation. As such, the results are summarized below in the following table. I have also added 2 additional tests for the overflow flag.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | S | overflow\_flag | Time (ps) |
| 0101 (+5) | 0001 (+1) | 0100 (+4) | 0 | 0 - 100,000 |
| 0101 (+5) | 1001 (-1) | 0110 (+6) | 0 | 100,000 – 200,000 |
| 1101 (-5) | 0001 (+1) | 1110 (-6) | 0 | 200,000 – 300,000 |
| 1101 (-5) | 1001 (-1) | 1100 (-4) | 0 | 300,000 – 400,000 |
| 0111 (+7) | 1111 (-7) | 1010 (-2) | 1 | 400,000 – 500,000 |
| 1111 (-7) | 0111 (7) | 0010 (+2) | 1 | 500,000 – 600,000 |
| 0000 (+0) | 0000(+0) | 0000 (+0) | 0 | >600,000 |



**Verilog Code**

|  |  |
| --- | --- |
| **module full\_fourbit\_subtractor(** input [3:0] A, input [3:0] B, output [3:0] S, output overflow\_flag ); wire [3:0] i1; // Intermediate wires wire [3:0] i2; wire [3:0] i3; wire [3:0] i4; wire [3:0] i5; wire [3:0] i6; wire [3:0] i7; wire [3:0] i8; wire [3:0] i9; wire carry; wire neg\_flag; wire zero\_flag, zero\_flag1; assign i7 = A & 4'b0111; // Masking out MSB to remove sign assign i8 = B & 4'b0111; twos\_complement C1 (i7, i1, zero\_flag); // Twos-complement for negative twos\_complement C2 (i8, i2, zero\_flag1); assign i3 = (A[3] == 0) ? i7 : i1; assign i4 = (B[3] == 1) ? i8 : i2; fourbit\_full\_adder A1 (i3, i4, 1'b0, i5, not\_neg\_flag); reverse\_twos\_complement C3 (i5, i6); assign neg\_flag = (i5[3] == 0) ? 0 : 1; assign i9 = (neg\_flag == 0) ? i5 : i6; assign S = (neg\_flag == 0) ? (i9 & 4'b0111) : (i9 | 4'b1000); // Adding sign back in assign overflow\_flag = (i3[3] != i4[3]) ? 0 : ((i3[3] == i5[3]) ? 0 : 1); **endmodule**  **module onebit\_full\_adder(** input A, input B, input CIN, output S, output COUT ); assign S = (A ^ B) ^ CIN; assign COUT = (A & B) | (CIN & (A ^ B)); **endmodule** | **module fourbit\_full\_adder(** input [3:0] A, input [3:0] B, input CIN, output [3:0] S, output COUT ); wire C1, C2, C3; onebit\_full\_adder A1 (A[0], B[0], CIN, S[0], C1); onebit\_full\_adder A2 (A[1], B[1], C1, S[1], C2); onebit\_full\_adder A3 (A[2], B[2], C2, S[2], C3); onebit\_full\_adder A4 (A[3], B[3], C3, S[3], COUT);**endmodule**  **module twos\_complement(** input [3:0] A, output [3:0] B, output zero\_flag ); fourbit\_full\_adder A1 ((~A), 4'b0001, 1'b0, B, zero\_flag);**endmodule**  **module reverse\_twos\_complement(** input [3:0] IN, output [3:0] OUT ); wire dummy; wire [3:0] intermediate; fourbit\_subtractor C1(IN, 1'b0001, intermediate, dummy); assign OUT = ~intermediate; **endmodule**  **module fourbit\_subtractor(** input [3:0] A, input [3:0] B, output [3:0] S, output neg\_flag ); wire [3:0] intermediate; wire not\_neg\_flag; wire zero\_flag; twos\_complement C1 (B, intermediate, zero\_flag); fourbit\_full\_adder A1 (A, intermediate, 1'b0, S, not\_neg\_flag); assign neg\_flag = (zero\_flag == 0) ? (~(not\_neg\_flag)) : (not\_neg\_flag); **endmodule**  **module simulation(** ); wire [3:0] S; wire overflow\_flag; reg [3:0] A; reg [3:0] B; full\_fourbit\_subtractor dut (A, B, S, overflow\_flag); initial begin A = 4'b0101; B = 4'b0001; #100; A = 4'b0101; B = 4'b1001; #100; A = 4'b1101; B = 4'b0001; #100; A = 4'b1101; B = 4'b1001; #100; A = 4'b0111; B = 4'b1111; #100; A = 4'b1111; B = 4'b0111; #100; A = 4'b0000; B = 4'b0000; #100; end**endmodule** |