ELC 2137 Lab 7: Binary Coded Decimal

Aaron Mendoza

October 14, 2020

Summary

The purpose of this lab is to use the seven segment module I built in Lab 6 and use it to make the switches display a number in both decimal and hex. In order to do this, I had to implement the double-dabble algorithm into my Basys3 board by programming a file onto it from Vivado.

First, I began by understanding how the algorithm works. In order to convert hex values into BCD, the double-dabble algorithm is used. In this algorithm, the bits are shifted left. If the bits that are shifted left are greater than four, then you would add three to those bits and shift afterwards. This would convert the bits into BCD.

Next, I had to write this module into SystemVerilog. I began by creating an add3 module. It would take a four-bit input and output another four bit number. If the input was greater than four, then I would add three to those bits. These modules are crucial in building the full double-dabble circuit because each circuit uses multiple add3 modules.

The second module I built was a 6-bit BCD converter module. It had an input of 6 bits; the three leftmost bits would read into the first add3 module with the leftmost bit of the add3 module being 0, and the output would cascade down into more add3 modules. The 6-bit BCD converter had only three add3 modules.

The third module I built was an 11-bit BCD converter module. It had an input of 11 bits, and followed the same pattern as the 6-bit module, but with fifteen add3 modules.

The final module built combined the sseg module created in lab 6 and the 11-bit BCD converter built in this lab. The ones and tens output from the 11-bit module was used as the A and B inputs in the sseg. I created a wrapper that gave the ability to toggle between hex and BCD by making two more MUX's prior to entering the first MUX. These worked by having the same "select" (sw[14]), which allowed the circuit to decide whether or not to convert the 11-bit input or not. If select was on, then the inputs are in hex and need to be converted so they would go through the 11-bit BCD converter. If select was off, then the input is already in BCD and it would pass through straight into the input of the MUX within the sseg.

To implement this onto the Basys3 board, I generated a bitstream and programmed the board with my code.

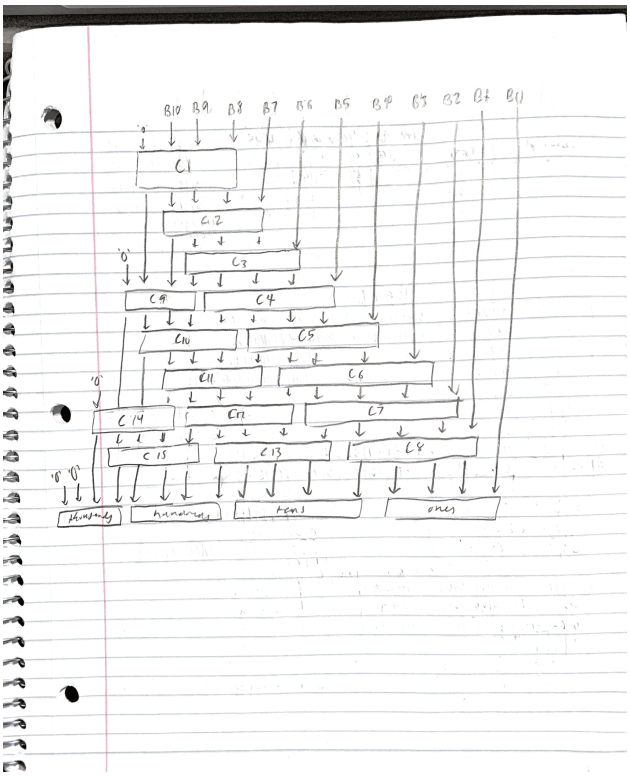

Figure 1: This is an image of the 11-bit BCD converter circuit diagram.

Results

Figure 2: ERT for Add 3 Module

num	modnum
0000	0000
0001	0001
0010	0010
0011	0011
0100	0100
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100
1010	1101
1011	1110
1100	1111
1101	0000
1110	0001
1111	0010

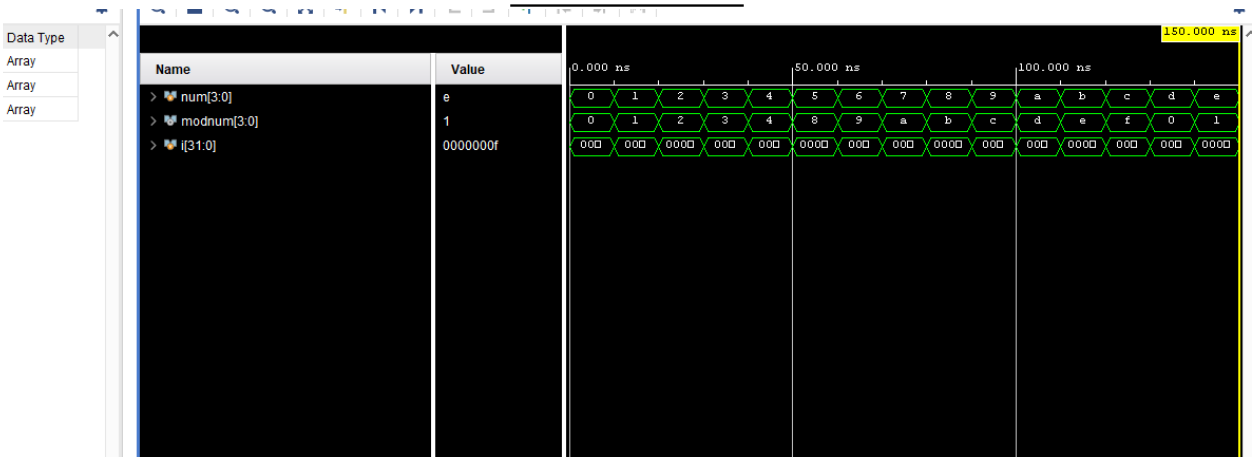

Figure 3: Add 3 Module Simulation

Figure 4: ERT for 6-Bit BCD Converter Module

input	tens	ones
000000	0000	0000
000001	0001	0001
000010	0000	0010
000011	0000	0011
000100	0000	0100
000101	0000	0101
.....	.....	.....
111110	0110	0011
111111	0110	0100

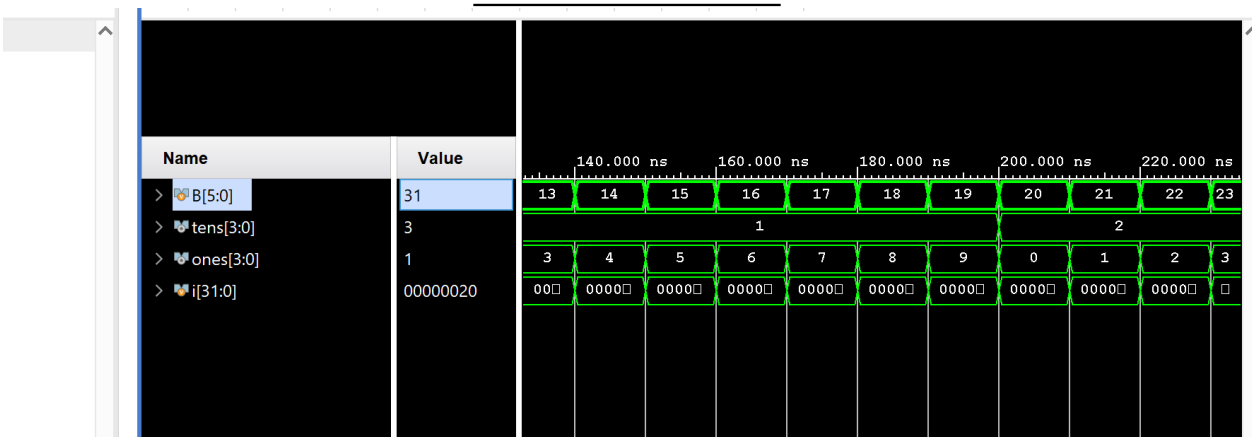

Figure 5: Six Bit Double-Dabble Module Simulation

Figure 6: ERT for 11-Bit BCD Converter Module

input	thousands	hundreds	tens	ones
00000000000	0000	0000	0000	0000
00000000001	0000	0000	0000	0001
00000000010	0000	0000	0000	0010
00000000011	0000	0000	0000	0011
.....	.....	.....	.....	.....
11111111110	0010	0000	0010	0111
11111111111	0010	0000	0010	1000

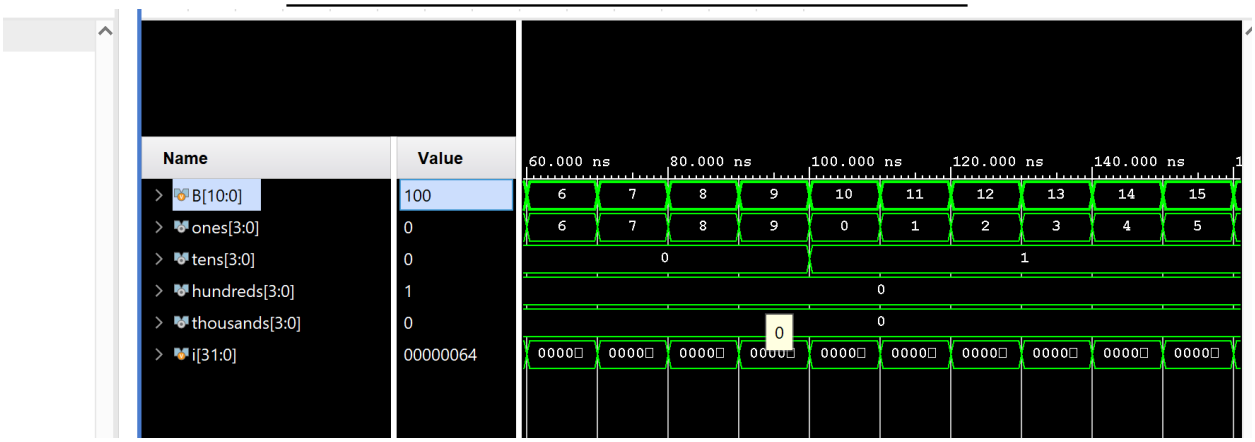

Figure 7: Eleven Bit Double-Dabble Module Simulation

Code

Listing 1: Add3 Module Code

```
module add3(
    input [3:0] num,
    output [3:0] modnum
    );

    reg [3:0] w;

    always @* begin
    if (num > 4'b0100)
        w = num + 4'b0011;
    else
        w = num;
    end

    assign modnum = w;

endmodule
```

—

Listing 2: Add3 TB

```
module add3_test();

reg [3:0] num;
wire [3:0] modnum;
integer i;

add3 dut(
    .num(num),
    .modnum(modnum)
    );

initial begin
    num[3:0]=4'h0000;
    for (i=0; i < 4'hF; i++) begin
        num = i;
        #10;
    end
    $finish;
end
```

—

Listing 3: 6-Bit BCD Converter Code

```
module six_bit_BCD(
    input [5:0] B,
    output [3:0] tens,
```

```
    output [3:0] ones
    );

    wire[3:0] c1_modnum;
    wire[3:0] c2_modnum;
    wire[3:0] c3_modnum;
    add3 c1(
        .num({1'b0,B[5:3]}),
        .modnum(c1_modnum)
        );
    add3 c2(
        .num({c1_modnum[2:0],B[2]}),
        .modnum(c2_modnum)
        );
    add3 c3(
        .num({c2_modnum[2:0],B[1]}),
        .modnum(c3_modnum)
        );

    assign tens = {1'b0,c1_modnum[3],c2_modnum[3],c3_modnum[3]};
    assign ones = {c3_modnum[2:0],B[0]};


endmodule
```

—

Listing 4: 6-Bit BCD Converter TB

```
module six_bit_BCD_test();
    reg [5:0] B;
    wire [3:0] tens;
    wire [3:0] ones;
    integer i;

    six_bit_BCD dut(
        .B(B),
        .tens(tens),
        .ones(ones)
        );

    initial begin
    B[5:0]=6'b000000;
    for (i=0; i < 6'b111111; i++) begin
        B = i;
        #10;
    end
    $finish;
end
endmodule
```

—

Listing 5: 11-Bit BCD Converter Code

```
module eleven_bit_BCD(
    input [10:0] B,
    output [3:0] ones,
    output [3:0] tens,
    output [3:0] hundreds,
    output [3:0] thousands
    );

    wire[3:0] c1_modnum;
    wire[3:0] c2_modnum;
    wire[3:0] c3_modnum;
    wire[3:0] c4_modnum;
    wire[3:0] c5_modnum;
    wire[3:0] c6_modnum;
    wire[3:0] c7_modnum;
    wire[3:0] c8_modnum;
    wire[3:0] c9_modnum;
    wire[3:0] c10_modnum;
    wire[3:0] c11_modnum;
    wire[3:0] c12_modnum;
    wire[3:0] c13_modnum;
    wire[3:0] c14_modnum;
    wire[3:0] c15_modnum;

    add3 c1(
        .num({1'b0,B[10:8]}),
        .modnum(c1_modnum)
        );
    add3 c2(
        .num({c1_modnum[2:0],B[7]}),
        .modnum(c2_modnum)
        );
    add3 c3(
        .num({c2_modnum[2:0],B[6]}),
        .modnum(c3_modnum)
        );
    add3 c4(
        .num({c3_modnum[2:0],B[5]}),
        .modnum(c4_modnum)
        );
    add3 c5(
        .num({c4_modnum[2:0],B[4]}),
        .modnum(c5_modnum)
        );
    add3 c6(
        .num({c5_modnum[2:0],B[3]}),
        .modnum(c6_modnum)
        );
    add3 c7(
        .num({c6_modnum[2:0],B[2]}),
        .modnum(c7_modnum)
        );
```

```
    add3 c8(
        .num({c7_modnum[2:0],B[1]}),
        .modnum(c8_modnum)
        );
    add3 c9(
        .num({1'b0,c1_modnum[3],c2_modnum[3],c3_modnum[3]}),
        .modnum(c9_modnum)
        );
    add3 c10(
        .num({c9_modnum[2:0],c4_modnum[3]}),
        .modnum(c10_modnum)
        );
    add3 c11(
        .num({c10_modnum[2:0],c5_modnum[3]}),
        .modnum(c11_modnum)
        );
    add3 c12(
        .num({c11_modnum[2:0],c6_modnum[3]}),
        .modnum(c12_modnum)
        );
    add3 c13(
        .num({c12_modnum[2:0],c7_modnum[3]}),
        .modnum(c13_modnum)
        );
    add3 c14(
        .num({1'b0,c9_modnum[3],c10_modnum[3],c11_modnum[3]}),
        .modnum(c14_modnum)
        );
    add3 c15(
        .num({c14_modnum[2:0],c12_modnum[3]}),
        .modnum(c15_modnum)
        );

    assign ones={c8_modnum[2:0],B[0]};
    assign tens={c13_modnum[2:0],c8_modnum[3]};
    assign hundreds={c15_modnum[2:0],c13_modnum[3]};
    assign thousands={1'b0,1'b0,1'b0,c14_modnum[3]};

endmodule
```

—

Listing 6: 11-Bit BCD Converter TB

```
module elev_BCD_test_();

    reg[10:0] B;
    wire[3:0] ones;
    wire[3:0] tens;
    wire[3:0] hundreds;
    wire[3:0] thousands;
    integer i;

    eleven_bit_BCD dut(
```

```
        .B(B),
        .ones(ones),
        .tens(tens),
        .hundreds(hundreds),
        .thousands(thousands)
        );

    initial begin
    B[10:0]=11'b00000000000;
    for (i = 0; i < 11'b1111111111; i++) begin
        B=i; #10;
        end
    end
endmodule
```

—

Listing 7: Top Level Module

```
module sseg1_bcd_wrapper(
    input [15:0] sw,
    input clk,
    output [3:0] an,
    output [6:0] seg,
    output dp
    );

    wire[7:0] num;
    wire[3:0] ones_wire;
    wire[3:0] tens_wire;
    wire[3:0] hundreds_wire;
    wire[3:0] thousands_wire;

    mux2_4b muxA(
    .in0(num[7:4]),
    .in1(num[3:0]),
    .sel(sw[14]),
    .out(ones_wire)
    );

    mux2_4b muxB(
    .in0(num[7:4]),
    .in1(num[3:0]),
    .sel(sw[14]),
    .out(tens_wire)
    );

    eleven_bit_BCD myelevBCD(
        .B(sw[10:0]),
        .ones(ones_wire),
        .tens(tens_wire),
        .hundreds(hundreds_wire),
        .thousands(thousands_wire)
        );
```

```
    sseg1_BCD my_sseg1(
    .A(ones_wire),
    .B(tens_wire),
    .sel(sw[15]),
    .seg_un(an[3:2]),
    .dp(dp),
    .sseg(seg),
    .seg_L(an[1]),
    .seg_R(an[0])
    );

endmodule
```

—