ELC 2137 Lab 07: Binary Coded Decimal

My Nguyen

October 12, 2020

Summary

This lab's purpose is to implement the double-dabble algorithm to convert hex values into binary coded decimal (BCD) using Verilog. The double-dabble algorithm, could be divided into to two parts shifting the code and adding to 3 to any binary number that is larger than 4. First, add3 file is created to detect any binary number larger than 4 and add 3 to it. Then, bcd6 and bcd11 modules were created to convert a 6-bit and 11-bit input to BCD using multiple add3 modules. Finally, create a top-level module named sseg1_BCD which uses all 16 switches, bcd11 module, mux2_4b and sseg_decoder and output to the seven segment display.

Table and Figure

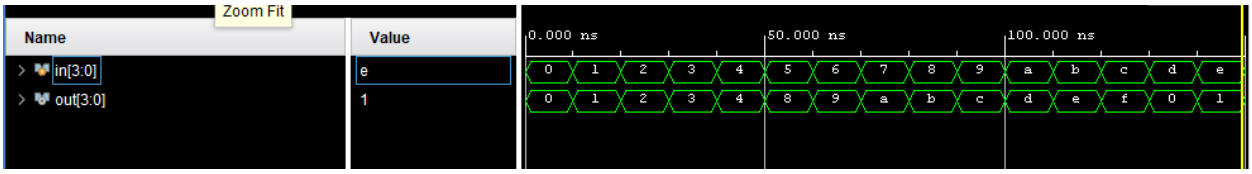

Figure 1: Add3 ERT

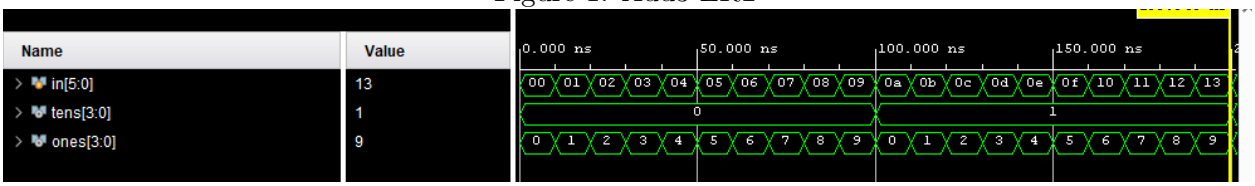

Figure 2: 6-bit BCD Convertor ERT

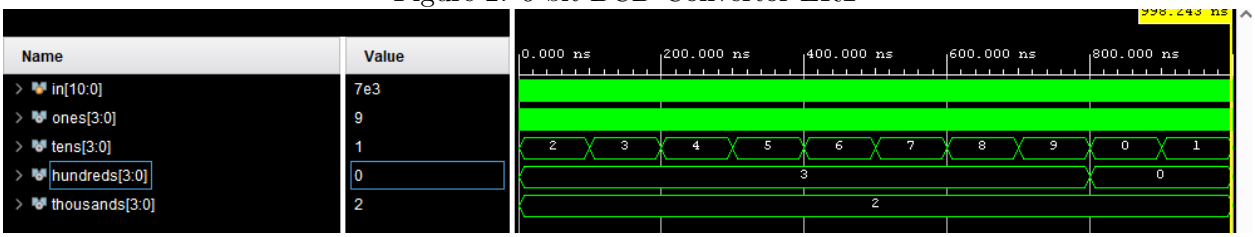

Figure 3: 11-bit BCD Convertor ERT

Code

Listing 1: Add3 Implementation

```
module add3(
```

```
    input [3:0] in,
    output [3:0] out
    );

    assign out = in > 4'b0100 ? in + 4'b0011 : in;
endmodule
```

Listing 2: Add3 Test Bench

```
module add3_test();
    reg [3:0] in;
    wire [3:0] out;

    add3 dut(
        .in(in),
        .out(out)
        );

    initial begin
        for (int i = 0; i <4'hF; i++) begin
            in = i; #10;
        end
        $finish;
    end
endmodule
```

Listing 3: 6-bit BCD Convertor Implementation

```
module bcd6(
    input [5:0] in,
    output [3:0] tens,ones
    );
    wire [2:0] tmp0;

    assign tens[3] = 1'b0;
    add3 c1(
        .in({1'b0 , in[5:3]}),
        .out({tens[2] , tmp0[2:0]})
    );

    wire [2:0] tmp1;
    add3 c2(
        .in({tmp0[2:0] , in[2]}),
        .out({tens[1] , tmp1[2:0]})
    );

    add3 c3(
        .in({tmp1[2:0] , in[1]}),
        .out({tens[0],ones[3:1]})
    );
    assign ones[0] = in[0];

endmodule
```

Listing 4: 6-bit BCD Convertor Test Bench

```
module bcd6_test();
    reg [5:0] in;
    wire [3:0] tens, ones;

    bcd6 dut(
        .in(in),
        .tens(tens),
        .ones(ones)
    );

    initial begin
        for (int i = 0; i < 6'b111111; i++) begin
            in = i; #10;
        end
        $finish;
    end
endmodule
```

Listing 5: 11-bit BCD Convertor Implementation

```
module bcd11(
    input [10:0] in,
    output [3:0] ones, tens, hundreds, thousands
    );

    assign thousands[3:2] = 2'b00;

    wire [3:0] tmpC1;

    add3 c1(
        .in({1'b0 , in[10:8]}),
        .out(tmpC1)
    );

    wire [3:0] tmpC2;
    add3 c2(
        .in({tmpC1[2:0],in[7]}),
        .out(tmpC2)
    );

    wire [3:0] tmpC3;
    add3 c3(
        .in({tmpC2[2:0],in[6]}),
        .out(tmpC3)
    );

    wire [3:0] tmpC4;
    add3 c4(
        .in({tmpC3[2:0],in[5]}),
        .out(tmpC4)
    );

    wire [3:0] tmpC5;
```

```
    add3 c5(
        .in({tmpC4[2:0],in[4]}),
        .out(tmpC5)
    );

    wire [3:0] tmpC6;
    add3 c6(
        .in({tmpC5[2:0],in[3]}),
        .out(tmpC6)
    );

    wire [3:0] tmpC7;
    add3 c7(
        .in({tmpC6[2:0],in[2]}),
        .out(tmpC7)
    );

    add3 c8(
        .in({tmpC7[2:0],in[1]}),
        .out({tens[0], ones[3:1]})
    );

    assign ones[0] = in[0];

    wire [3:0] tmpC9;
    add3 c9(
        .in({1'b0,tmpC1[3],tmpC2[3],tmpC3[3]}),
        .out(tmpC9)
    );

    wire [3:0] tmpC10;
    add3 c10(
        .in({tmpC9[2:0],tmpC4[3]}),
        .out(tmpC10)
    );

    wire [3:0] tmpC11;
    add3 c11(
        .in({tmpC10[2:0],tmpC5[3]}),
        .out(tmpC11)
    );

    wire [3:0] tmpC12;
    add3 c12(
        .in({tmpC11[2:0],tmpC6[3]}),
        .out(tmpC12)
    );

    add3 c13(
        .in({tmpC12[2:0],tmpC7[3]}),
        .out({hundreds[0], tens[3:1]})
    );

    wire [3:0] tmpC14;
```

```
    add3 c14(
        .in({1'b0,tmpC9[3],tmpC10[3],tmpC9[3]}),
        .out({thousands[1], tmpC14[2:0]})
    );

    add3 c15(
        .in({tmpC14[2:0],tmpC12[3]}),
        .out({thousands[0],hundreds[3:1]})
    );
endmodule
```

Listing 6: 11-bit BCD Convertor Test Bench

```
module bcd11_test();
    reg [10:0] in;
    wire [3:0] ones, tens, hundreds, thousands;

    bcd11 dut(
        .in(in),
        .ones(ones),
        .tens(tens),
        .hundreds(hundreds),
        .thousands(thousands)
    );

    initial begin
        for (int i = 11'b11110000000; i < 11'b11111111111; i++) begin
            in = i; #10;
        end
        $finish;
    end
endmodule
```

Listing 7: 11-bit BCD Convertor Test Bench

```
module sseg1_BCD(
    input [15:0] sw,
    output [3:0] an,
    output [6:0] seg,
    output dp
    );

    wire [3:0] bcd_ones, bcd_tens, convert, hundreds, thousands;

    bcd11 main(
        .in(sw[10:0]),
        .ones(bcd_ones),
        .tens(bcd_tens),
        .hundreds(hundreds),
        .thousands(thousands)
    );

    mux2_4b mux0(
        .in0(bcd_ones),
```

```
        .in1(bcd_tens),
        .se1(sw[15]),
        .out(convert)
    );

    sseg_decoder decoder0(
        .num(convert),
        .sseg(seg)
    );

    assign an[0] = sw[15];
    assign an[1] = ~sw[15];
    assign an[3:2] = 2'b11;
    assign dp = 1'b1;
endmodule
```

Screenshot

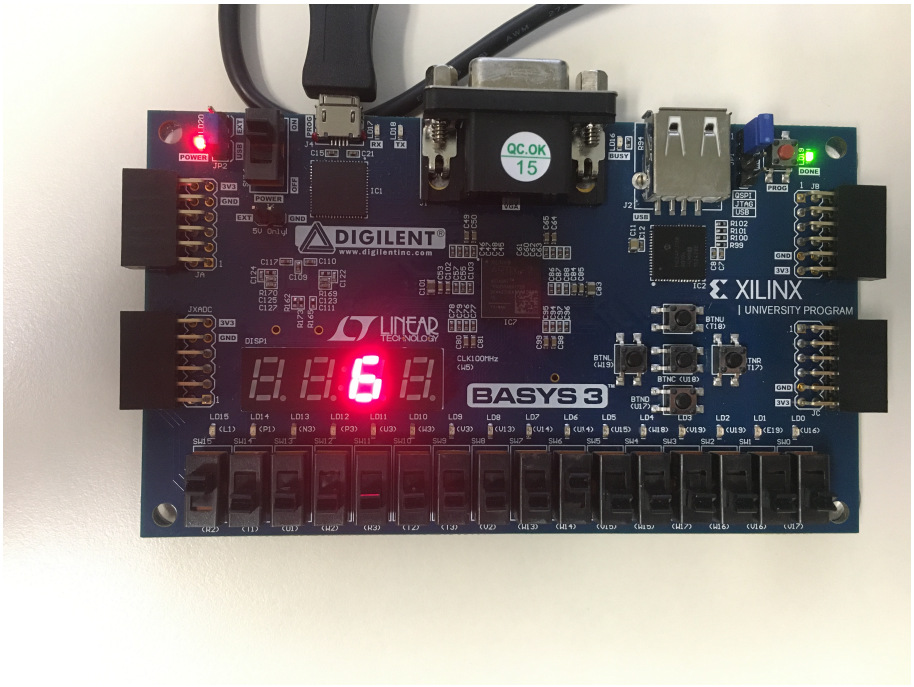

Figure 4: First digit

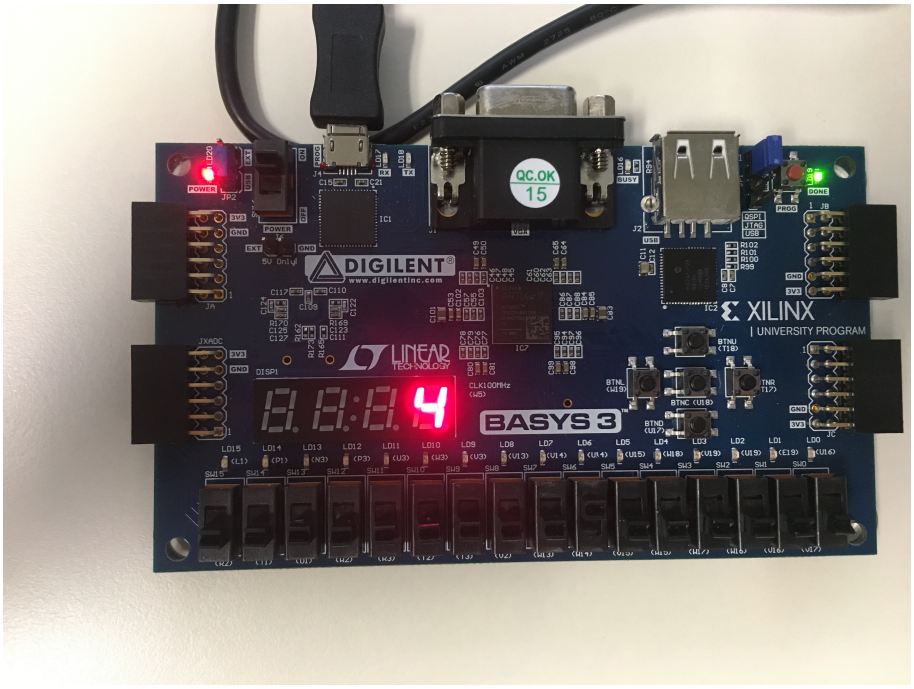

Figure 5: Second digit

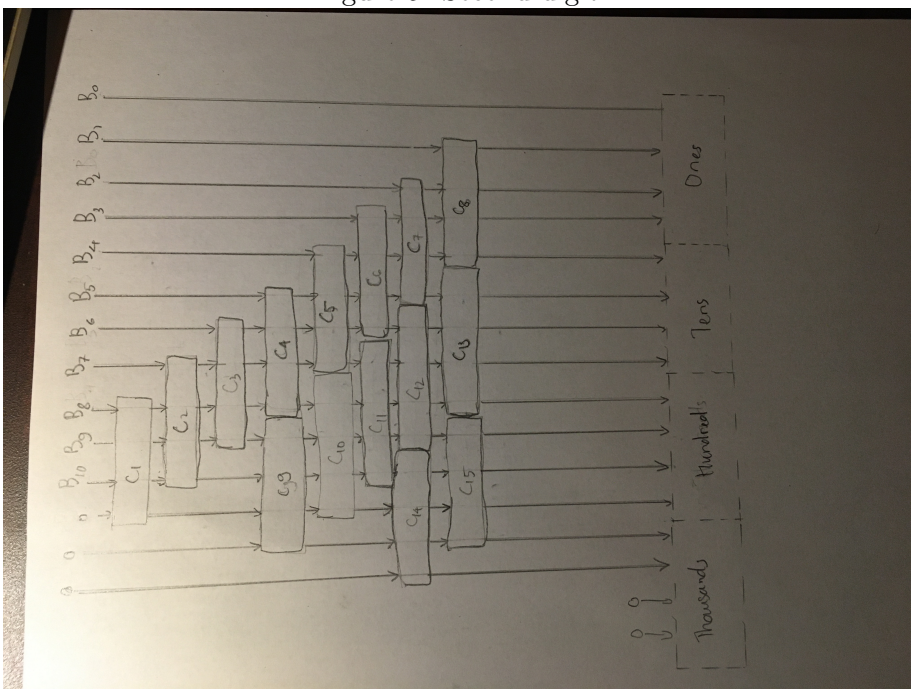

Figure 6: 11-bit Convertor Schematic