ELC 2137 Lab 09: ALU with Input Register

My Nguyen

October 26, 2020

Summary

This lab's purpose is to get a better understanding SR latch, D latch, D flip-flip and D-register with combinational and regular sequential logic. First part is to create a register module that uses parameter to generate register module with different number of bit for input and output. Then, create a 8-bit ALU module using the provided code. Afterward, create top_lab9 and import basys3.xdc with correct constraints as a top-level module to do on-board testing. Finally, import basys3_lab9.sv file to do top-level simulation.

Expected results tables

Table 1: *register* expected results table

Time (ns):	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40	40-45	45-50	50-55
D (hex)	0	0	A	A	3	3	0	0	0→6	6	6
clk	0	1	0	1	0	1	0	1	0	1	0
en	0	0	1	1	1→0	0→1	1→0	0	0→1	1	1
rst	0	0→1	0	0	0	0	0	0	0	0	0
Q (hex)	X	X→0	0	0→A	A	A	A	A	A→6	6	6

Table 2: *alu* expected results table skeleton

Time (ns):	0-10	10-20	20-30	30-40	40-50	50-60
in0	A	A	A	A	A	A
in1	B	B	B	B	B	B
op	0	1	2	3	4	5
out	15	FF	0A	0B	01	0A

Simulation Waveform

Pictures

Code

Listing 1: Register Implementation

```
module register #( parameter N =1)(
    input clk , rst , en ,
```

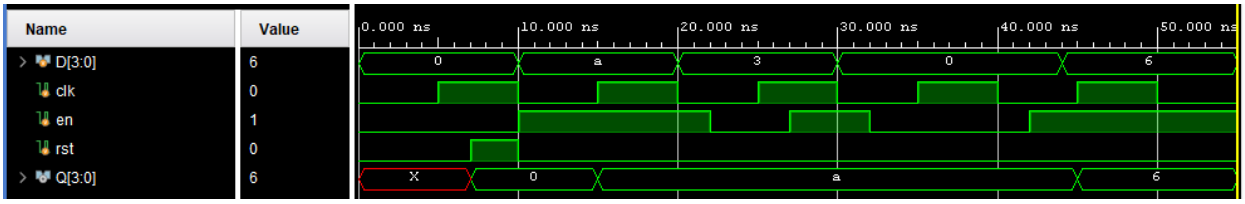

Figure 1: Register ERT

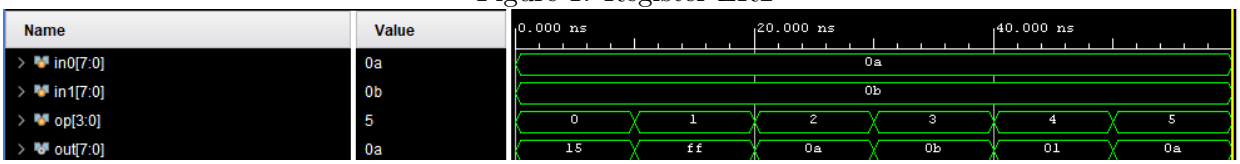

Figure 2: ALU ERT

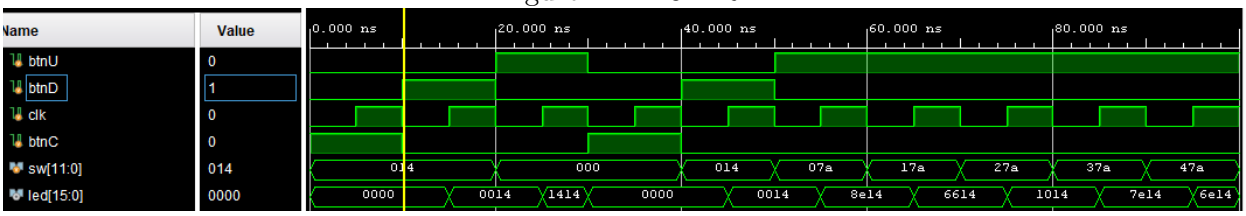

Figure 3: Top-Level Simulation ERT

```
    input [N -1:0] D,
    output reg [N -1:0] Q
    );
    always @( posedge clk , posedge rst)
    begin
        if (rst ==1)
            Q <= 0;
        else if (en ==1)
            Q <= D;
    end
endmodule
```

Listing 2: Register Test Bench

```
module register_test();
    reg [3:0] D;
    reg clk , en , rst ;
    wire [3:0] Q;
    register #(.N(4)) r(.D(D), .clk(clk),
    .en(en), .rst(rst), .Q(Q) );
    // clock runs continuously
    always begin
    clk = ~clk ; #5;
    end
    // this block only runs once
    initial begin
        clk =0; en =0; rst =0; D=4'h0; #7;
        rst = 1; #3;
        D = 4'hA; en = 1; rst = 0; #10;
```

```
        D = 4'h3; #2;
        en = 0; #5;
        en = 1; #3;
        D = 4'h0; #2;
        en = 0; #10;
        en = 1; #2;
        D = 4'h6; #11;
        $finish ;
    end
endmodule
```

Listing 3: ALU Implementation

```
module alu #( parameter N =8)(
    input [N-1:0] in0,
    input [N-1:0] in1,
    input [3:0] op,
    output reg [N-1:0] out
    );
    // Local parameters
    parameter ADD =0;
    parameter SUB =1;
    parameter AND =2;
    parameter OR =3;
    parameter XOR =4;
    always @*
        begin
            case(op)
                ADD: out = in0 + in1;
                SUB: out = in0 - in1;
                AND: out = in0 & in1;
                OR: out = in0 | in1;
                XOR: out = in0 ^ in1;
                default : out = in0 ;
            endcase
        end
endmodule
```

Listing 4: ALU Test Bench

```
module alu_test();
    reg [8-1:0] in0;
    reg [8-1:0] in1;
    reg [3:0] op;
    wire [8-1:0] out;

    alu #(.N(8)) alu(
        .in0(in0),
        .in1(in1),
        .op(op),
        .out(out)
        );

   initial begin
```

```
        in0 = 8'hA; in1 = 8'hB;
        for (int i = 0; i < 6; i++)begin
            op = i; #10;
        end
        $finish;
   end
endmodule
```

Listing 5: Top-Level Implementation

```
module top_lab9(
        input [11:0] sw,
        input btnU, btnD, btnC, clk,
        output [15:0] led
    );
    //in this circuit the first number inputed is stored as in1

    wire [7:0] regist_alu;
    register #(.N(8)) register0(
        .D(sw[7:0]),
        .en(btnD),
        .clk(clk),
        .rst(btnC),
        .Q(regist_alu)
        );
    assign led[7:0] = regist_alu;

    wire [7:0] alu_regist;
    alu #(.N(8)) alu0(
        .in0(sw[7:0]),
        .in1(regist_alu),
        .op(sw[11:8]),
        .out(alu_regist)
        );

    register #(.N(8)) register1(
        .D(alu_regist),
        .en(btnU),
        .clk(clk),
        .rst(btnC),
        .Q(led[15:8])
        );

endmodule
```

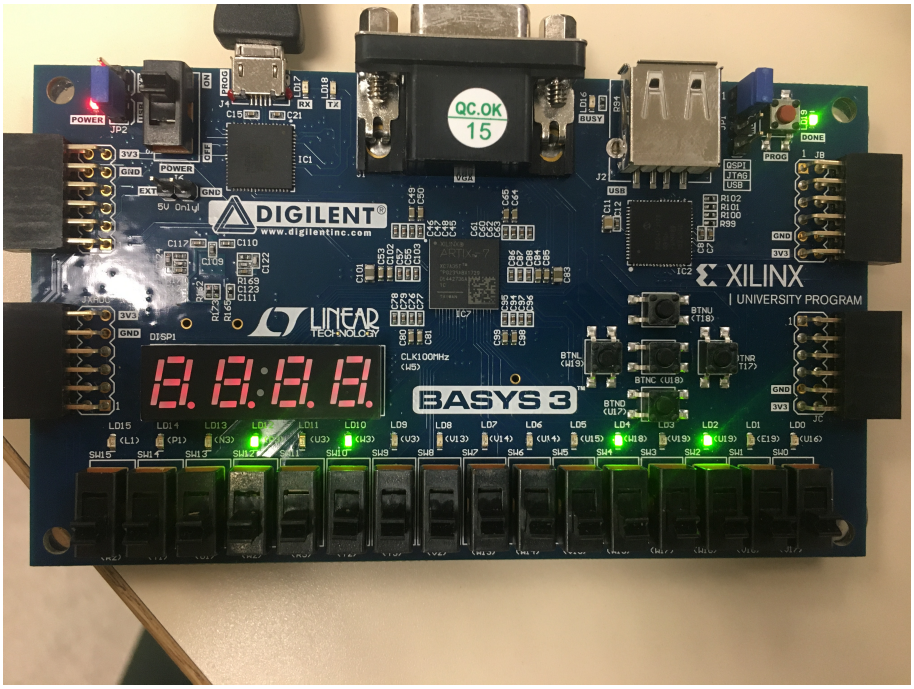

Figure 4: Adding 0

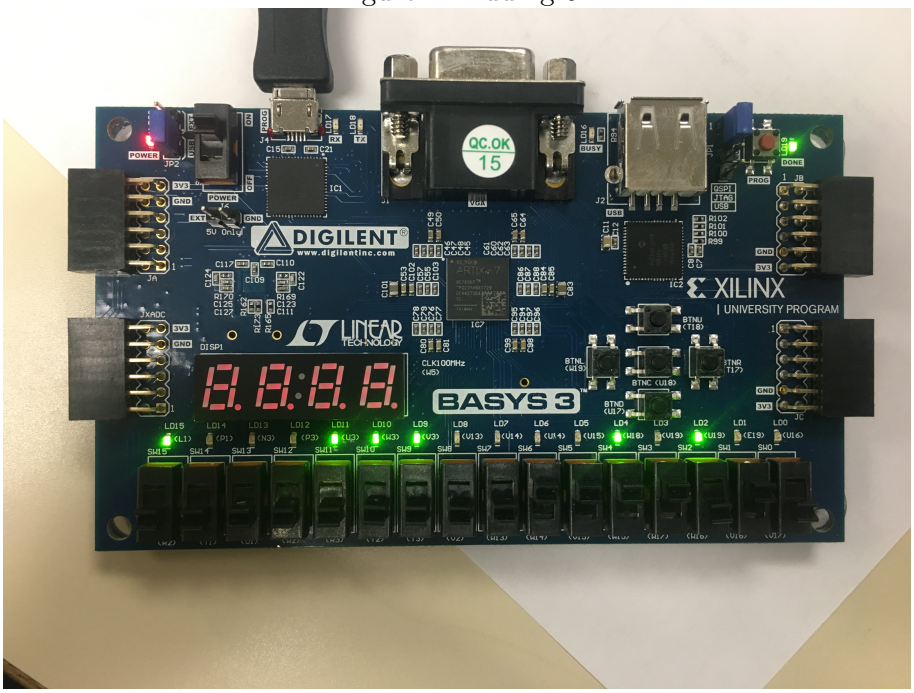

Figure 5: Addition Operation

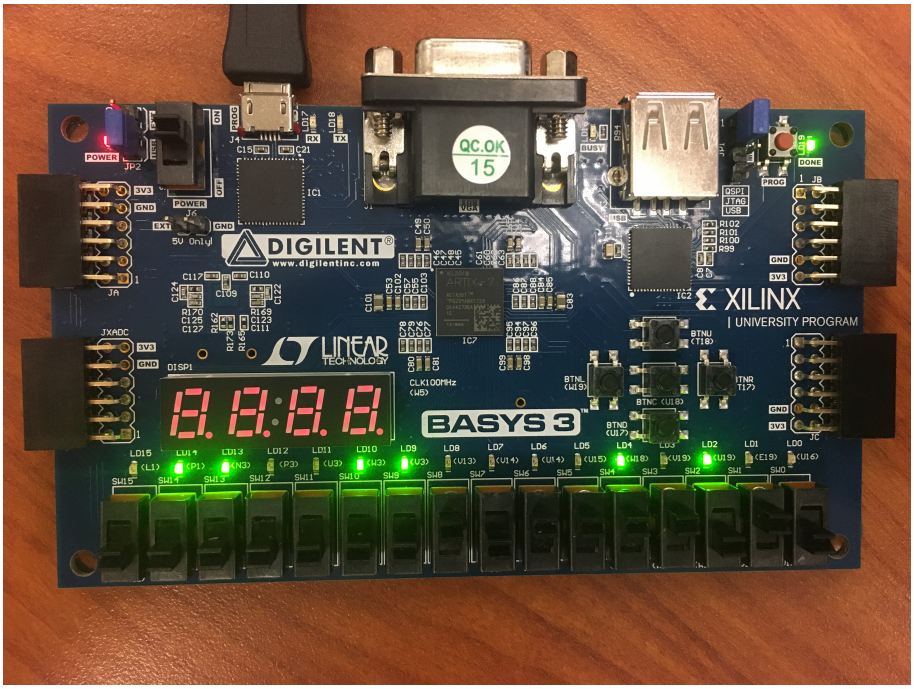

Figure 6: Subtraction Operation

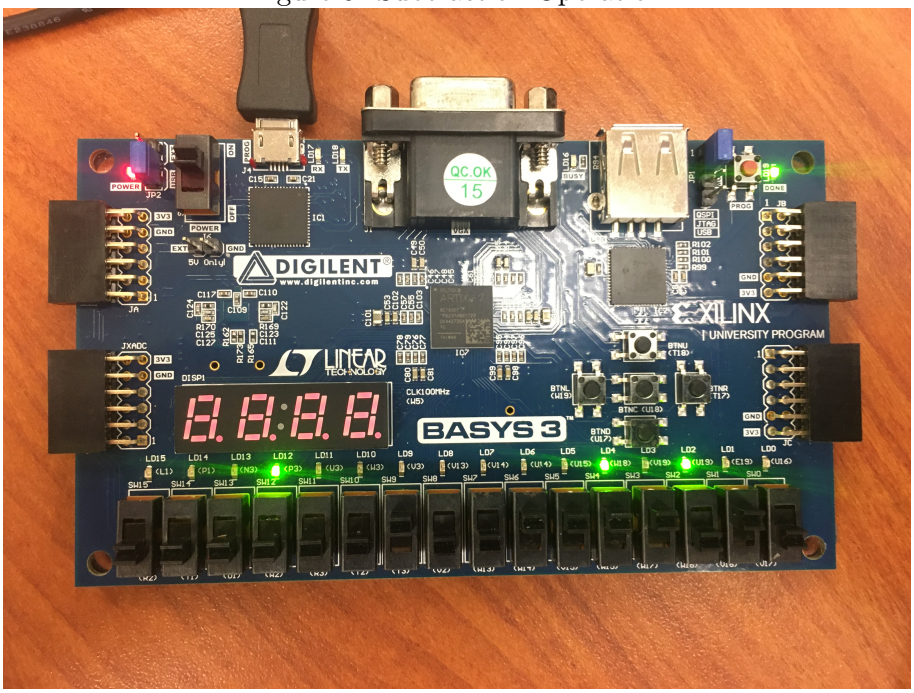

Figure 7: AND Operation

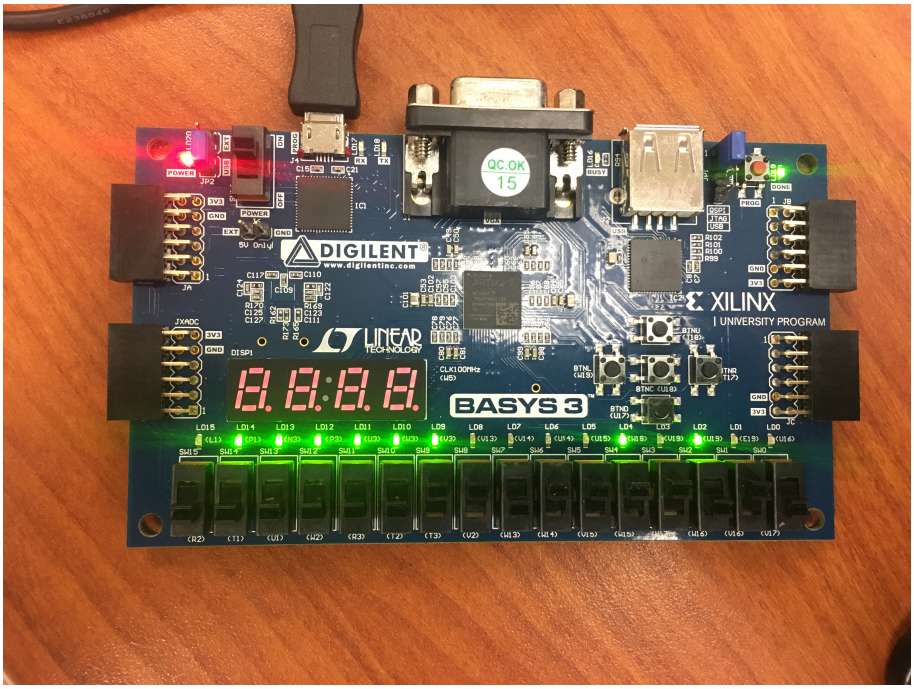

Figure 8: OR Operation

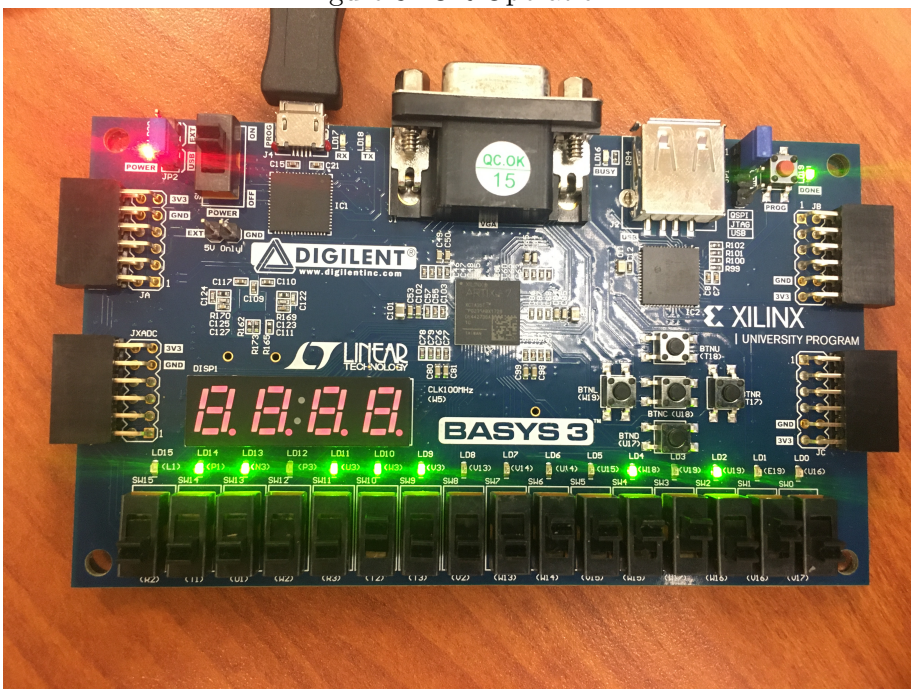

Figure 9: XOR Operation

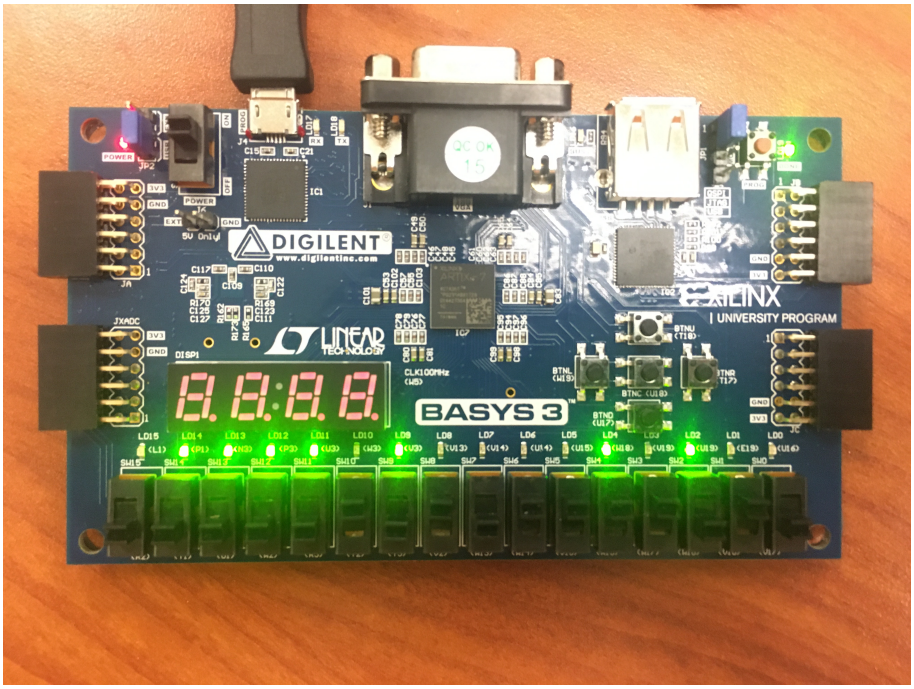

Figure 10: Default Operation