ELC 2137 Lab 9: ALU with Input Register

Mariah Montgomery

October 25, 2020

Summary

In this lab, we created a simple arithmetic logic unit (ALU) with an input register. The ALU needed two numbers to perform a mathematical operation, so we made the numbering using switches, and we had an input register store the numbers. The circuit we built that produced the top-level module is an example of regular sequential logic because it relies on time and past outputs, whereas combinational logic does not. The first step in this experiment was to build a register. The register took input and memorized the information on the positive edge of a clock change. The memorized input was then passed to the output Q. Next, we built an ALU, which is very similar to a multiplexer. The ALU accepted the Q and a number inputted via switches. Then, based on a set parameter, the ALU performed a mathematical operation between two numbers. The ALU output was then passed to a second register that stored the input during the positive edge of a clock change. At the end of this experiment, we used our Basys3 boards to verify our ALUs.

When testing the register, we had to use the non-blocking technique because we tested sequential logic rather than combinational logic. Nonblocking does not assign elements procedurally, so it mimics hardware more accurately than blocking assignments in test benches.

Q&A

Table 1: *Register* Expected Results Table

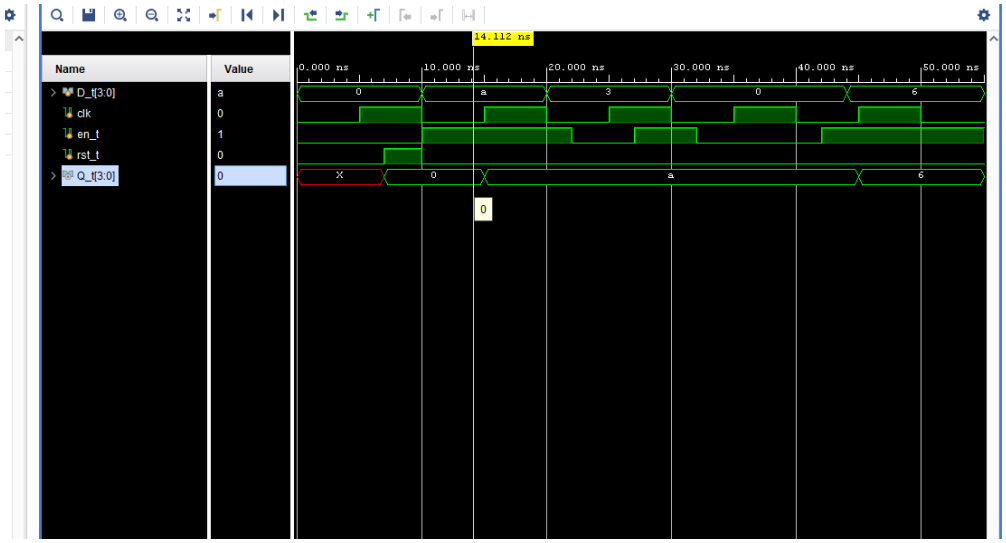
Time (ns):	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40	40-45	45-50	50-55
D (hex)	0	0	A	A	3	3	0	0	0→6	6	6
clk	0	1	0	1	0	1	0	1	0	1	0
en	0	0	1	1	1→0	0→1	1→0	0	0→1	1	1
rst	0	0→1	0	0	0	0	0	0	0	0	0
Q (hex)	X	X→0	0	A	A	A	A	A	A	6	6

Table 2: *ALU* Expected Results Table

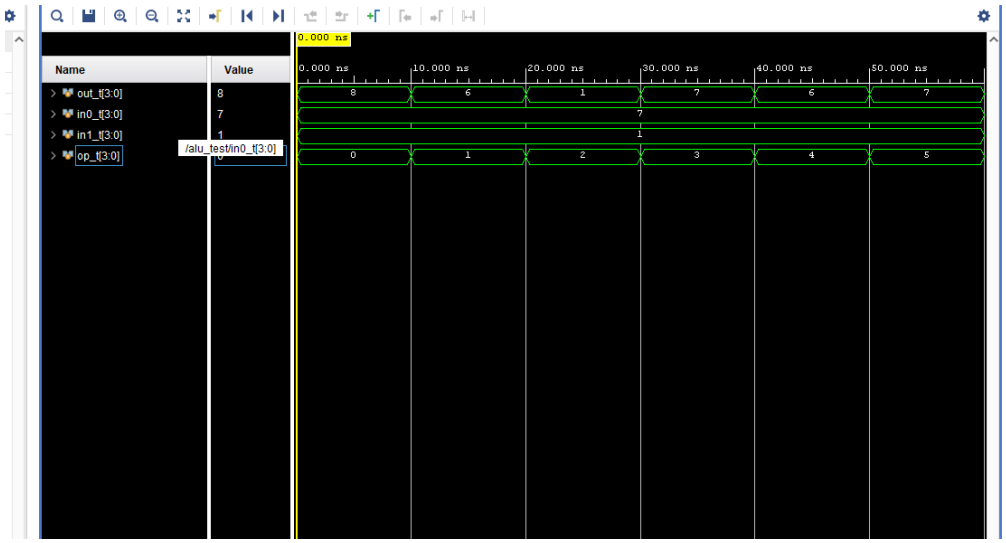
Time (ns):	0-10	10-20	20-30	30-40	40-50	50-60
in0	0111	0111	0111	0111	0111	0111
in1	0001	0001	0001	0001	0001	0001
op	0	1	2	3	4	5
out	1000	0110	0001	0111	0110	0111

Results

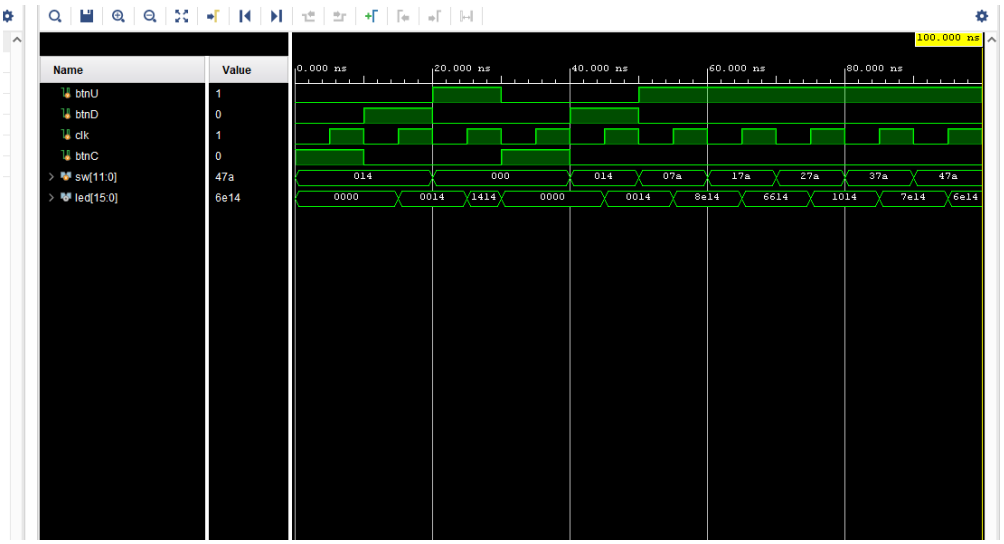
1. Register Test Simulation



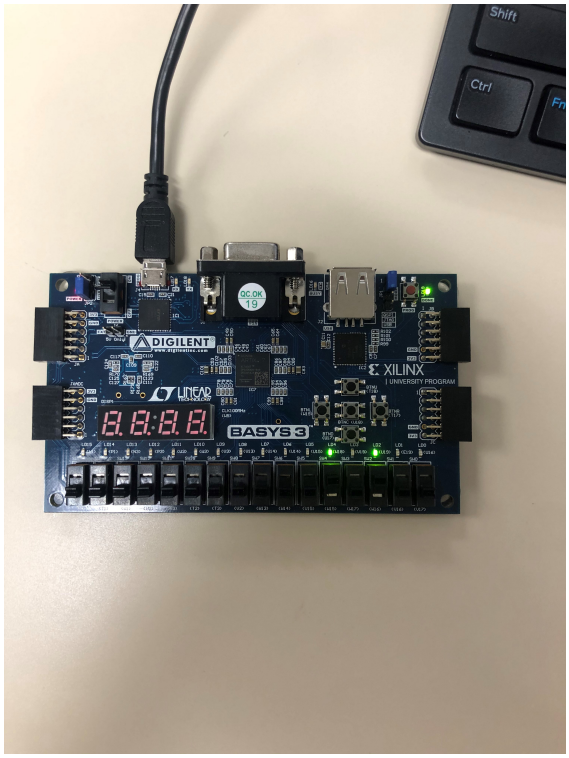
2. ALU Test Simulation



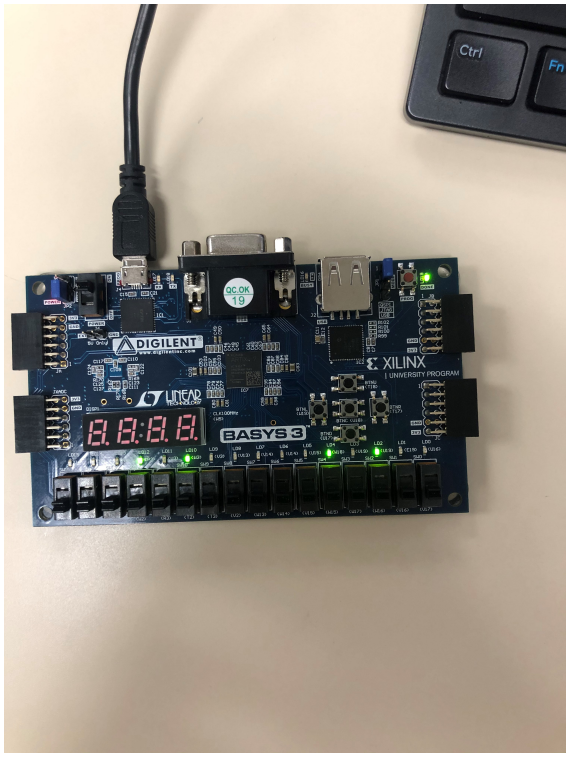
3. Top-Level Simulation



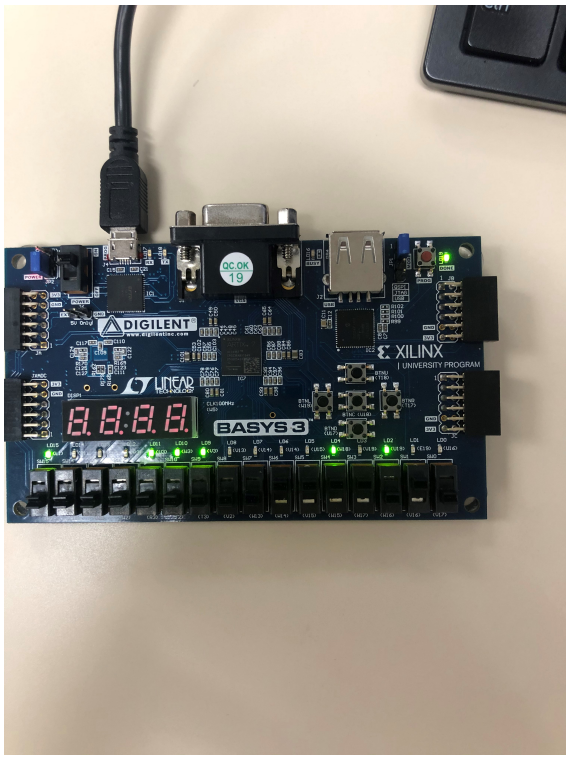
4. On Board Testing Step 1



5. On Board Testing Step 2

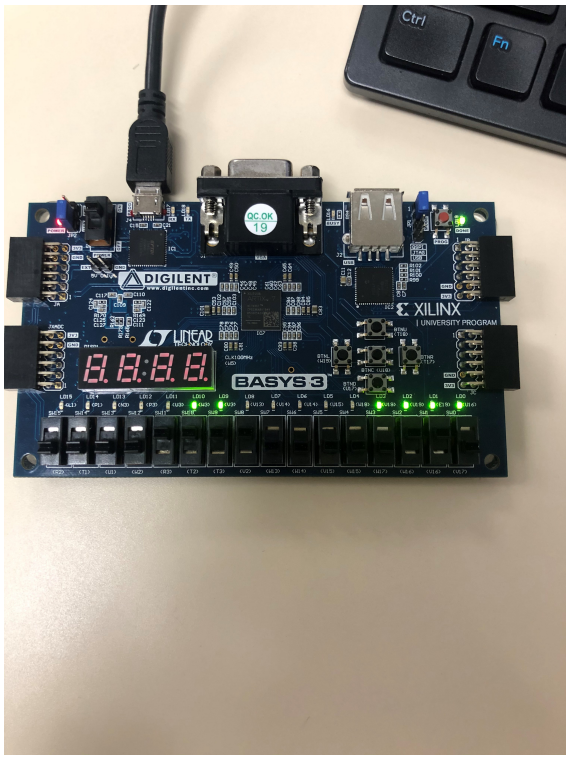


6. On Board Testing Step 3

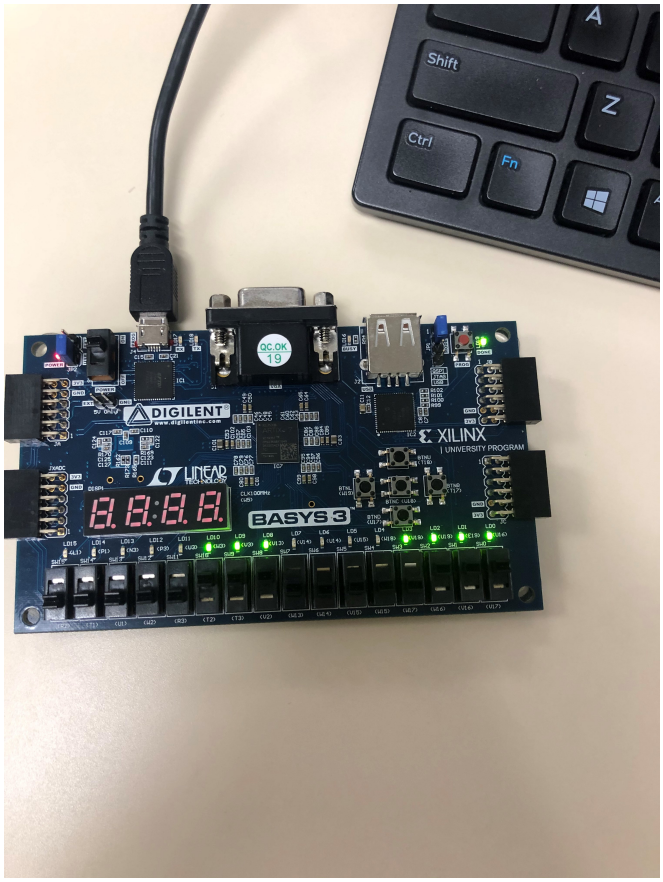


7. On Board Testing Step 4

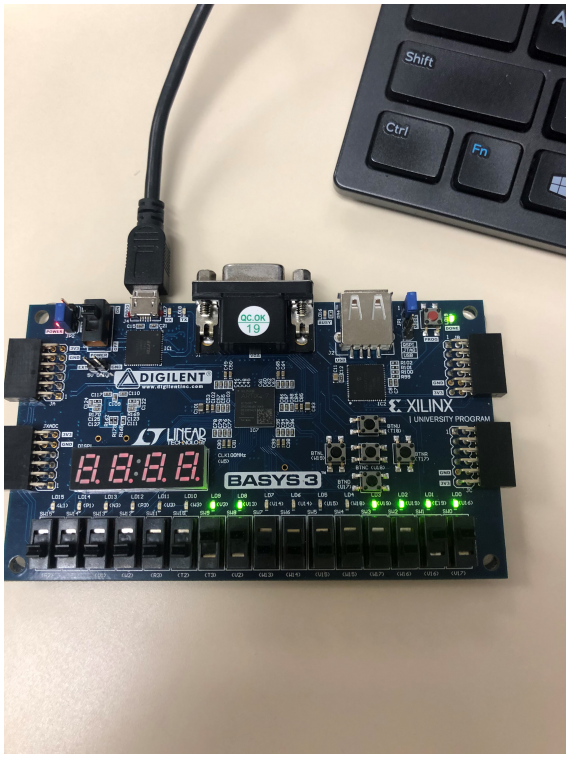
8. On Board Testing Step 5



9. On Board Testing Step 6



10. On Board Testing Step 7



Code

Listing 1: register

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

```
module register_test();

   reg [3:0] D_t;
   reg clk, en_t, rst_t;
   wire [3:0] Q_t;

   register #(.N(4)) dut(
      .D(D_t),
      .clk(clk),
      .en(en_t),
      .rst(rst_t),
      .Q(Q_t)
   );

   always begin
      clk = ~clk; #5;
   end

   initial begin
      clk = 0; en_t = 0; rst_t = 0; D_t = 4'h0; #7;
      rst_t = 1; #3;
      D_t = 4'hA; en_t = 1; rst_t = 0; #10;
      D_t = 4'h3; #2;
      en_t = 0; #5;
      en_t = 1; #3;
      D_t = 4'h0; #2;
      en_t = 0; #10;
      en_t = 1; #2;
      D_t = 4'h6; #11;
      $finish;
   end
```

```
endmodule
```

Listing 3: ALU

```
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
   );

   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

```
module alu_test();

   reg [3:0] in0_t;
   reg [3:0] in1_t;
   reg [3:0] op_t;
   reg [3:0] out_t;

   alu #(.N(4)) dut (
      .out(out_t),
      .in0(in0_t),
      .in1(in1_t),
      .op(op_t)
   );

   initial begin
      in0_t = 4'h7; in1_t = 4'h1; op_t = 0; #10;
      in0_t = 4'h7; in1_t = 4'h1; op_t = 1; #10;
      in0_t = 4'h7; in1_t = 4'h1; op_t = 2; #10;
      in0_t = 4'h7; in1_t = 4'h1; op_t = 3; #10;
      in0_t = 4'h7; in1_t = 4'h1; op_t = 4; #10;
      in0_t = 4'h7; in1_t = 4'h1; op_t = 5; #10;
```

```
      $finish;
   end
endmodule
```

Listing 5: Top-level Module

```
module top_lab9(
   input btnU,
   input btnD,
   input btnC,
   input [11:0] sw,
   input clk,
   output [15:0] led
   );

   wire [7:0] my_register1_out;
   wire [7:0] my_alu_out;
   wire [7:0] my_register2_out;

   register #(.N(8)) my_register1(
      .D(sw[7:0]),
      .clk(clk),
      .en(btnD),
      .rst(btnC),
      .Q(my_register1_out)
   );

   alu #(.N(8)) my_alu (
      .in0(sw[7:0]),
      .in1(my_register1_out),
      .op(sw[11:8]),
      .out(my_alu_out)
   );

   register #(.N(8)) my_register2(
      .D(my_alu_out),
      .clk(clk),
      .en(btnU),
      .rst(btnC),
      .Q(my_register2_out)
   );

   assign led = {my_register2_out, my_register1_out};
endmodule
```