EE/CE/TE 1202– Introduction to Electrical Engineering II

Dr. Amir Khoobroo

Lab 3: Designing the Toy Processor Datapath

Assigned: 10/05/2015 **Due: 10/19/2015**

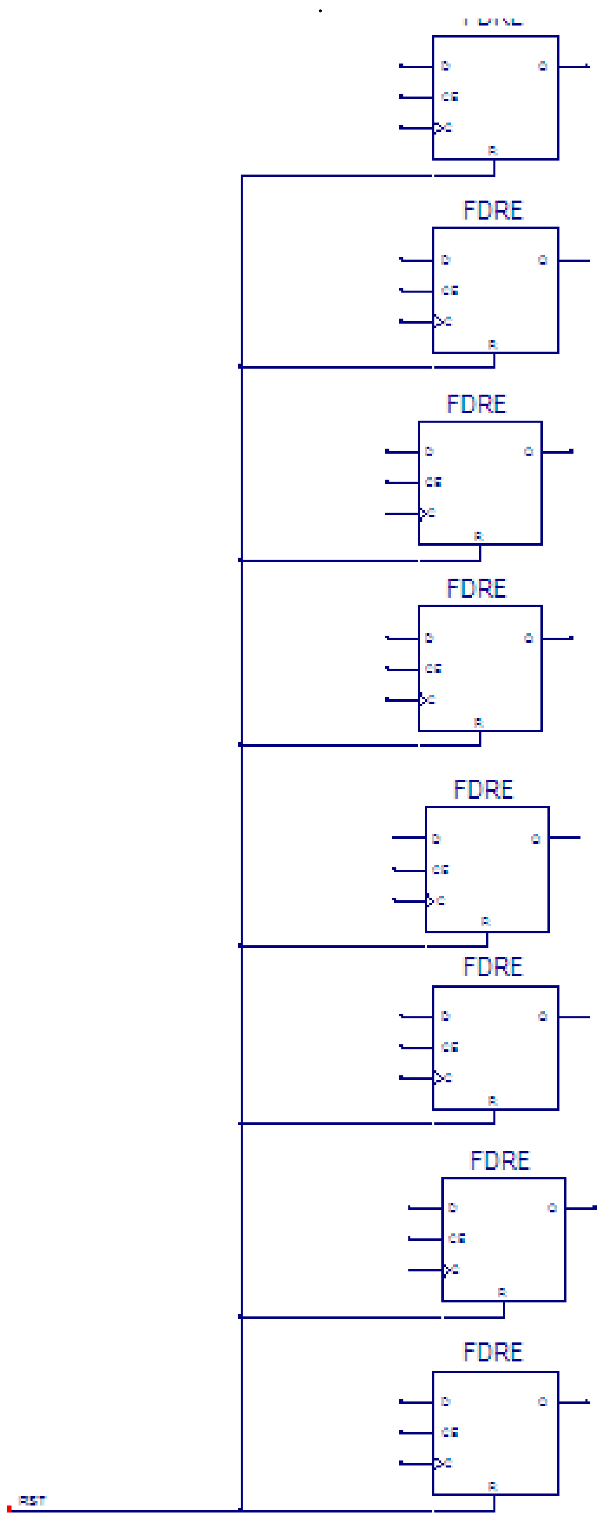
Contents

Part 1

Building an 8-bit Register
We will build an 8-bit register using eight 1-bit registers. The 1-bit register symbol is called FDRE.

1. Create a new schematic and name it **reg_sch.sch.**
2. On your drawing sheet, arrange eight FDRE symbols from the ***ISE*** library as shown below. Connect their R ports with a net. Name the net RST.

Note: The FDRE symbol is made up of a D-flip-flop and a 2-1 mux. When CE is high, it loads the current value of D. If CE is low, D is ignored. If RST is high the FDRE is reset.

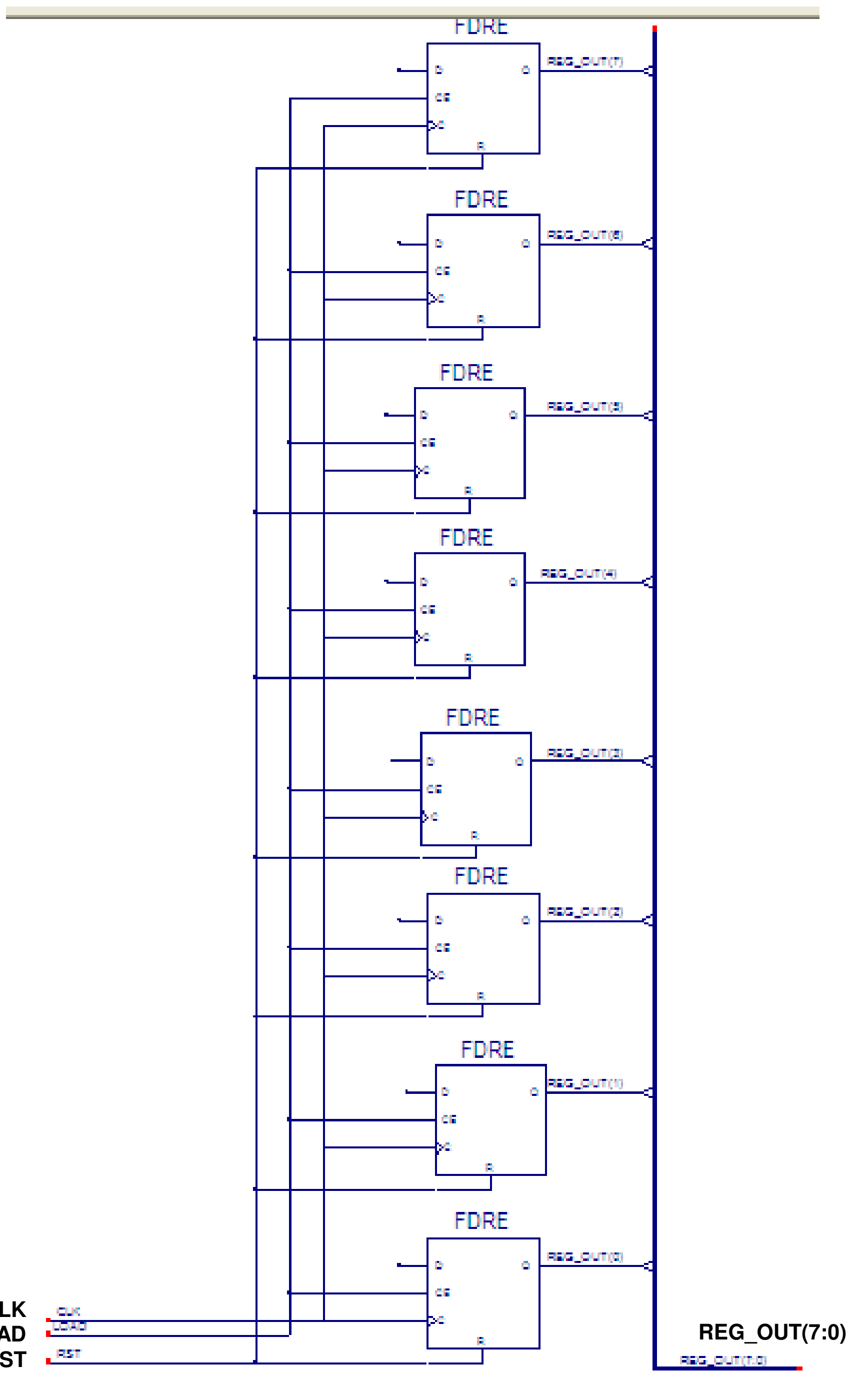


8 FDRE with R-ports connected to net RST.

3. Similarly connect the CE pins to a single net named LOAD to allow for a common enable signal for all eight components. Do the same for the clock pins C. Name the clock net CLK.

4. Connect the output ports Q to a bus called REG_OUT(7:0).

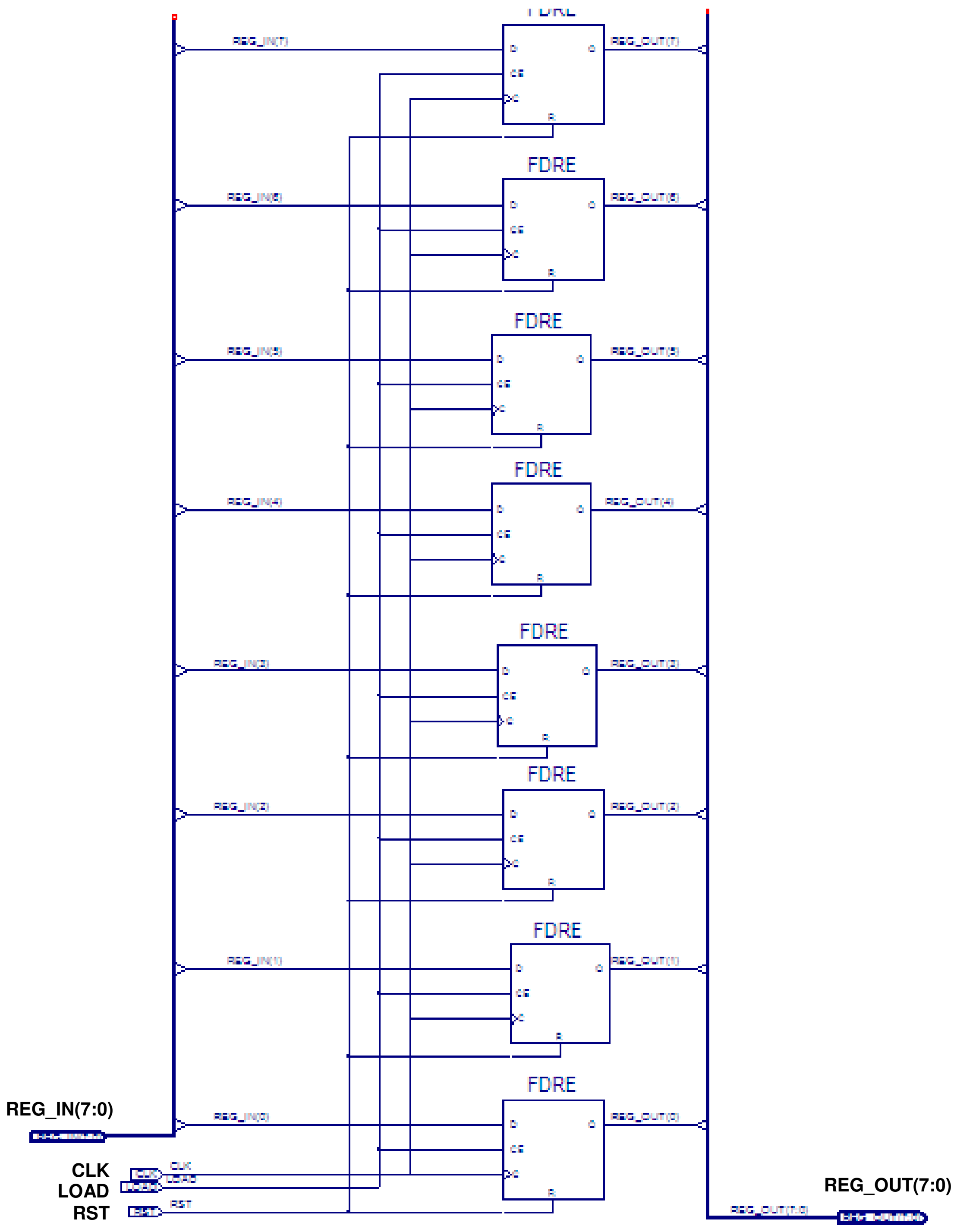
   Note: Remember that the bus tap nets must be named with the pattern REG_OUT(7)…REG_OUT(0).
   Your schematic should look something like the following now:



REG_OUT(7:0) bus connected, CLK, LOAD, RST connected

5. Connect the D ports in a bus named REG_IN(7:0).

6. Add I/O ports to the schematic. CLK, RST, LOAD and REG_IN(7:0) are all inputs. REG_OUT(7:0) is an output. Your final schematic will look like this:



Finished register with inputs CLK, LOAD, RST, REG_IN(7:0) and output REG_OUT(7:0)

Creating a Test bench waveform

1. Use the following code to create **reg_tb.v**. If asked by Associate Source to "Select a source with which to associate the new source", choose the **reg_sch**.

```
//reg_tb.v

`timescale 1ns/1ps

module reg_tbw_tb_0;
   reg CLK = 1'b0;
   reg LOAD = 1'b0;
   reg [7:0] REG_IN = 8'b00000000;
   reg RST = 1'b0;
   wire [7:0] REG_OUT;

   parameter PERIOD = 200;
   parameter real DUTY_CYCLE = 0.5;
   parameter OFFSET = 100;

   initial    // Clock process for CLK
   begin
      #OFFSET;
      forever
      begin
         CLK = 1'b0;
         #(PERIOD-(PERIOD*DUTY_CYCLE)) CLK = 1'b1;
         #(PERIOD*DUTY_CYCLE);
      end
   end

   reg_sch UUT (
      .CLK(CLK),
      .LOAD(LOAD),
      .REG_IN(REG_IN),
      .RST(RST),
      .REG_OUT(REG_OUT));

   initial begin
      // -------------  Current Time:  185ns
      #185;
      LOAD = 1'b1;
      // -------------------------------------
      // -------------  Current Time:  385ns
      #200;
      REG_IN = 8'b00000001;
      // -------------------------------------
      // -------------  Current Time:  585ns
```

```
        #200;
        LOAD = 1'b0;
        REG_IN = 8'b00000010;
        // ------------------------------------
        // ------------  Current Time:  785ns
        #200;
        REG_IN = 8'b00000011;
        // ------------------------------------
        // ------------  Current Time:  985ns
        #200;
        LOAD = 1'b1;
        RST = 1'b1;
        REG_IN = 8'b00000100;
        // ------------------------------------
        // ------------  Current Time:  1185ns
        #200;
        RST = 1'b0;
        REG_IN = 8'b00000101;
        // ------------------------------------
        // ------------  Current Time:  1385ns
        #200;
        REG_IN = 8'b00000110;
        // ------------------------------------
        // ------------  Current Time:  1585ns
        #200;
        REG_IN = 8'b00000111;
        // ------------------------------------
    end

endmodule
```

ISIM : Running the simulation

2. Choose the **Simulation** option on top of the **Design** window to enter into simulation mode.
3. In ***Behavioral window,*** highlight **reg_tb.v.** The ***Processes for Current Source*** will display an expandable toolbox tree ***ISim* Simulator.** Expand ***ISim* Simulator** and double-click **Simulate Behavioral Model.** This will run the ***ISim*** simulator process.

   What is the expected output?

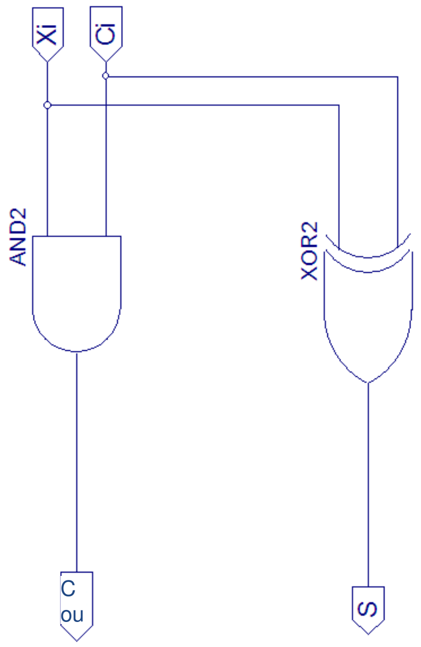
4. Does your register update REG_OUT to the current REG_IN only when LOAD is high? Does RST reinitialize all inputs and outputs? Make sure everything is working correctly.

You just finished building the 8-bit register.

Part 2: Let's make a Counter

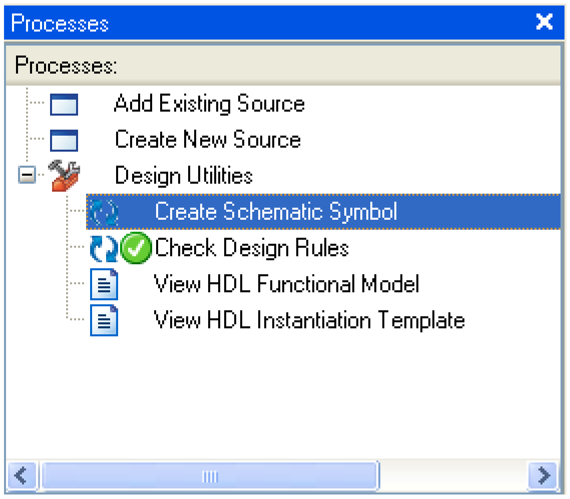
We will first build a half-adder.

1. Click **Project->New Source** in ***Project Manager.*** It is going to be a schematic circuit. Name it **ha_sch.sch.**
2. In ***ECS***, draw the half-adder circuit as shown below:



Half-adder circuit

3. Change the view to **/Implementation,** highlight **ha_sch.sch**, then expand the **Design Utilities**'s menu in the lower section. Double-click **Create Schematic Symbol** in the **Processes** tab. This creates a black-box type symbol for the half-adder. (You can create the symbol through Tools/Symbol Wizard as well)

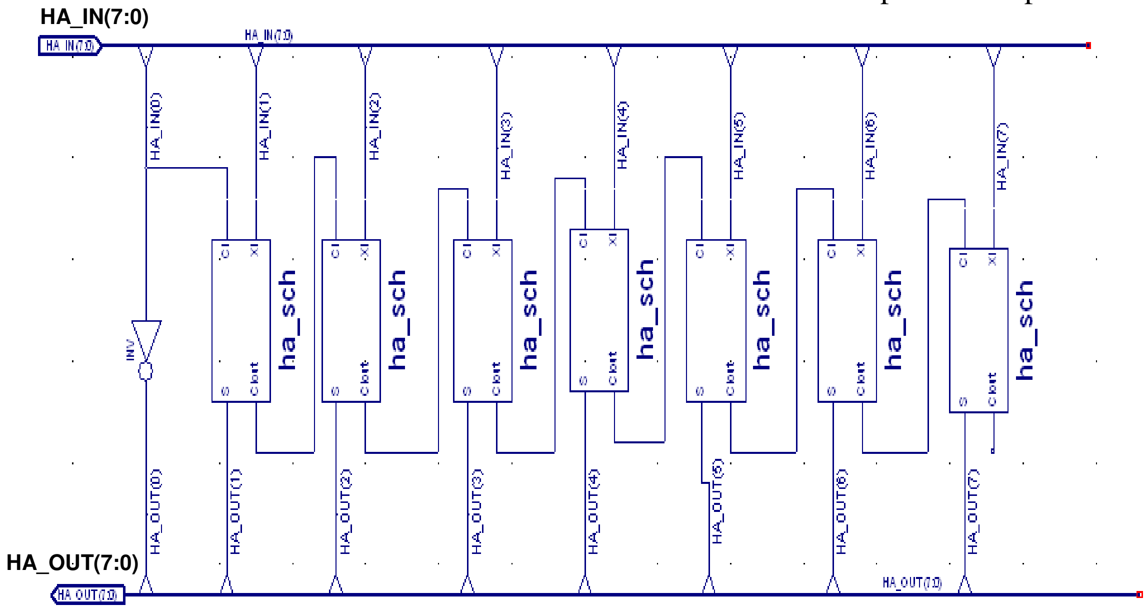


Symbol Wizard

Next we are going to build a half-adder array. This array will be used in making the counter.

4. Click **Project->New Source** in ***Project Manager.*** It is going to be a schematic circuit. Name it **ha8_sch.sch.**
5. In ***ECS***, draw the half-adder circuit as shown below Place 7 **ha_sch** symbols onto a new schematic sheet. Instead of using an 8th **ha_sch** on the least significant bit, we will use an inverter (NOT) gate since the half-adder effectively inverts its value.
6. Connect each Ci pin on one symbol to the Cout pin on the symbol to its left. This means the Cout pin on the rightmost **ha_sch** will be left unconnected.
7. Connect each Xi and the input of the INV symbol to an input bus named HA_IN(7:0).
8. Connect each S and the output of the INV symbol to an output bus named HA_OUT(7:0).

   Note: Make sure that INV is connected to the LSB of the input and output buses.



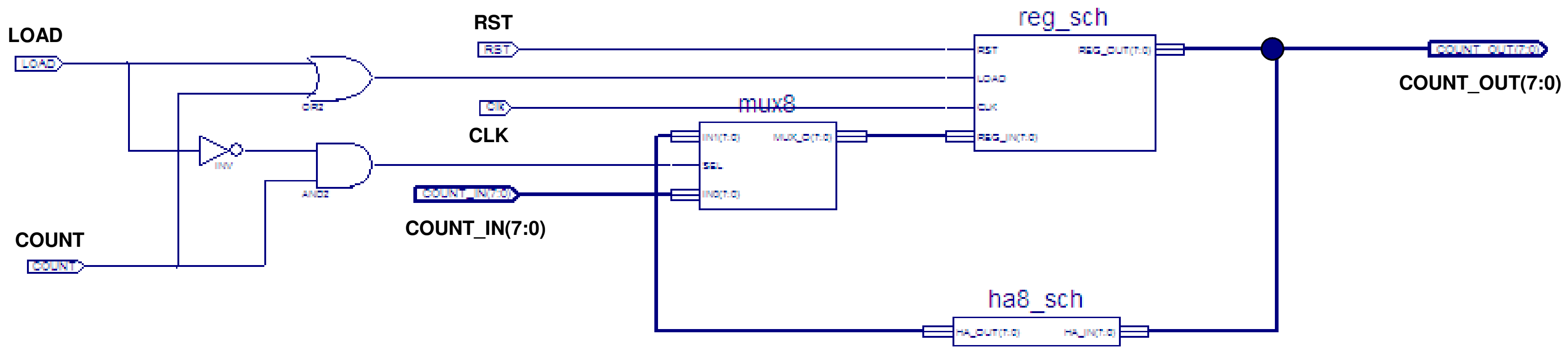
Note: Cout on the MSB of the half-adder is unconnected.

9. Save **ha8_sch.sch.** Create a symbol from **ha8_sch.sch** by invoking the ***Symbol Wizard*** from the ***Tools*** menu.

We are now going to build the counter.

10. In ***ISE,*** create a **New Source** of type schematic and name it **counter_sch.sch.**
11. Place the symbols **mux8_sch, ha8_sch** and **reg_sch** on your schematic drawing sheet. Your schematic should look something like this:

Note: The input of **ha8_sch** is connected to the output of **reg_sch.** The output of **ha8_sch** is connected to the input IN1(7:0) of **mux8_sch.**



Complete counter circuit.

12. Create a test bench called **counter_tb.v** (shown below, associated to **counter_sch**)**.** Use the following table to verify the correctness of your counter:

RST	LOAD	COUNT	COUNT_IN(7:0)	COUNT_OUT(7:0)
1	0	0	COUNT_IN(7:0)	0
1	0	1	COUNT_IN(7:0)	0
1	1	0	COUNT_IN(7:0)	0
1	1	1	COUNT_IN(7:0)	0
0	0	0	COUNT_IN(7:0)	Hold Previous Value
0	1	0	COUNT_IN(7:0)	COUNT_IN(7:0)
0	0	1	COUNT_IN(7:0)	COUNT_IN(7:0) + 1
0	1	1	COUNT_IN(7:0)	COUNT_IN(7:0)

Note: LOAD takes precedence over COUNT.

```
//counter_tb.v

`timescale 1ns/1ps

module counter_tbw_tb_0;
   reg CLK = 1'b0;
   reg COUNT = 1'b0;
   reg [7:0] COUNT_IN = 8'b00000000;
   reg LOAD = 1'b0;
   reg RST = 1'b0;
   wire [7:0] COUNT_OUT;

   parameter PERIOD = 200;
   parameter real DUTY_CYCLE = 0.5;
   parameter OFFSET = 100;

   initial    // Clock process for CLK
   begin
      #OFFSET;
      forever
      begin
         CLK = 1'b0;
         #(PERIOD-(PERIOD*DUTY_CYCLE)) CLK = 1'b1;
         #(PERIOD*DUTY_CYCLE);
      end
   end

   counter_sch UUT (
      .CLK(CLK),
      .COUNT(COUNT),
      .COUNT_IN(COUNT_IN),
    .LOAD(LOAD),
    .RST(RST),
    .COUNT_OUT(COUNT_OUT));

initial begin
    // -------------  Current Time:  185ns
    #185;
    RST = 1'b1;
    // ------------------------------------
    // -------------  Current Time:  585ns
    #400;
    COUNT_IN = 8'b00000001;
    // ------------------------------------
    // -------------  Current Time:  785ns
    #200;
    RST = 1'b0;
    COUNT_IN = 8'b00000010;
    // ------------------------------------
    // -------------  Current Time:  985ns
    #200;
    COUNT_IN = 8'b00000011;
    // ------------------------------------
    // -------------  Current Time:  1185ns
    #200;
    LOAD = 1'b1;
    COUNT_IN = 8'b00000100;
    // ------------------------------------
    // -------------  Current Time:  1385ns
    #200;
    COUNT = 1'b1;
    COUNT_IN = 8'b00000101;
```

```
// -------------------------------------
// -------------  Current Time:  1585ns
#200;
COUNT = 1'b0;
LOAD = 1'b0;
COUNT_IN = 8'b00000110;
// -------------------------------------
// -------------  Current Time:  1785ns
#200;
COUNT_IN = 8'b00000111;
// -------------------------------------
// -------------  Current Time:  1985ns
#200;
COUNT = 1'b1;
COUNT_IN = 8'b00001000;
// -------------------------------------
// -------------  Current Time:  2185ns
#200;
COUNT_IN = 8'b00001001;
// -------------------------------------
// -------------  Current Time:  2385ns
#200;
COUNT = 1'b0;
COUNT_IN = 8'b00001010;
// -------------------------------------
// -------------  Current Time:  2585ns
#200;
COUNT_IN = 8'b00001011;
// -------------------------------------
// -------------  Current Time:  2785ns
#200;
```

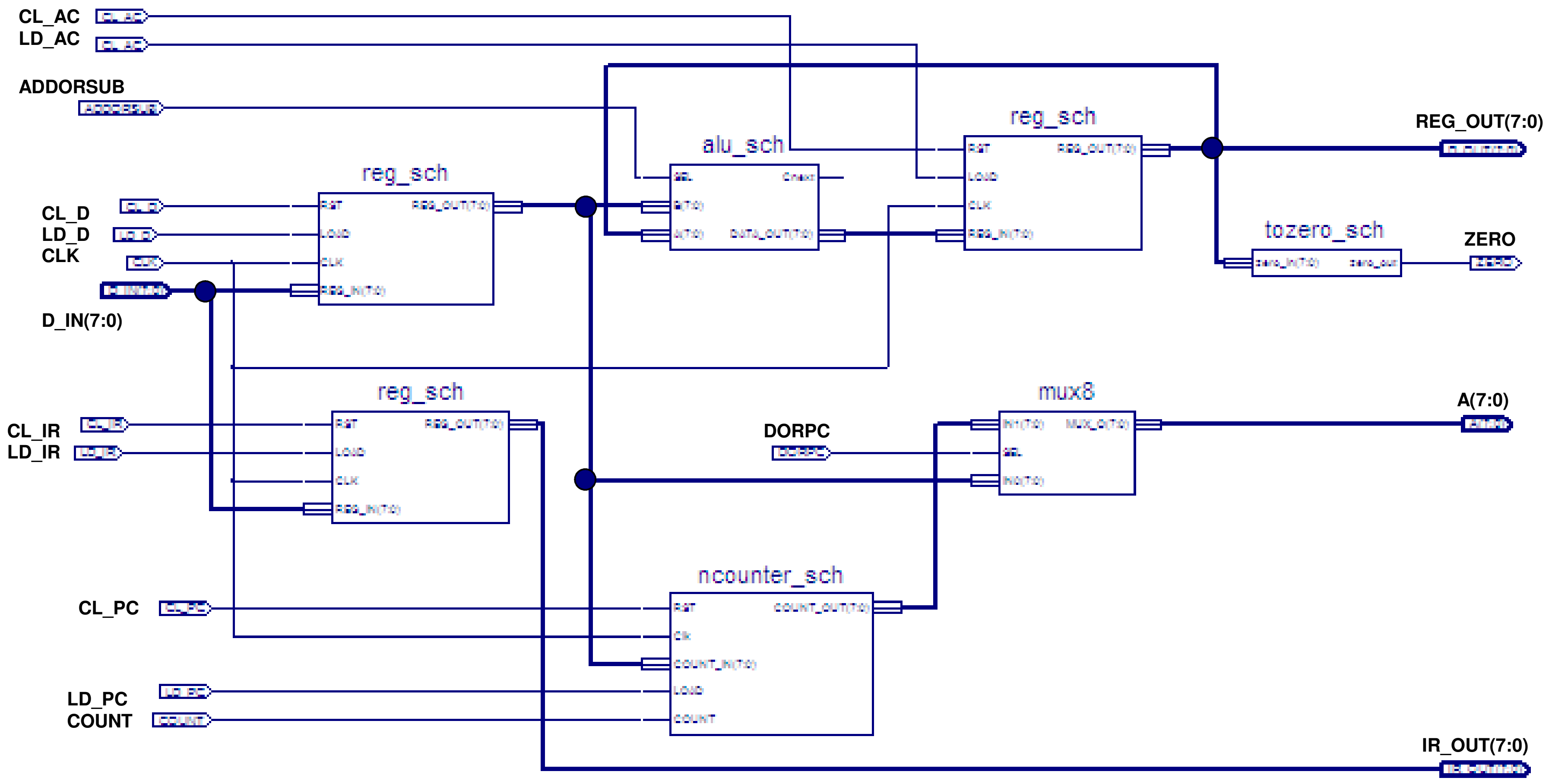
```
        COUNT_IN = 8'b00001100;
        // ------------------------------------
        // -------------  Current Time:  2985ns
        #200;
        COUNT_IN = 8'b00001101;
        // ------------------------------------
    end

endmodule
```

Now we have all the components necessary to build our datapath.

Part 3: Putting together the Datapath

Finally, use all the components that you designed in the directory ToyProcessor to build the Datapath. More specifically, create a new schematic, that you call "Datapath", and instantiate the necessary components as shown in the figure below. We will not simulate this datapath by itself, so be very careful in making the right connections. **Note:** Cnext on alu_sch is left disconnected.



CL_AC
LD_AC
ADDORSUB
CL_D
LD_D
CLK
D_IN(7:0)
CL_IR
LD_IR
CL_PC
LD_PC
COUNT
reg_sch
RST
LOAD
CLK
REG_IN(7:0)
REG_OUT(7:0)
alu_sch
SEL
B(7:0)
A(7:0)
Cnext
DATA_OUT(7:0)
tozero_sch
zero_in(7:0)
zero_out
mux8
IN1(7:0)
SEL
IN0(7:0)
MUX_O(7:0)
ncounter_sch
RST
Clk
COUNT_IN(7:0)
LOAD
COUNT
COUNT_OUT(7:0)
DORPC
REG_OUT(7:0)
ZERO
A(7:0)
IR_OUT(7:0)