EE 231

Homework 13

Due December 3, 2010

1. Explain in words and write the HDL statements for the operations specified by the following register transfer notations;

   (a) $R1 \leftarrow R1 - 1, R2 \leftarrow R1$

   Transfer the contents of $R1$ minus 1 into $R1$; at the same time, transfer the contents of $R1$ (before subtracting the 1) into $R2$.

   ```
   always @( posedge clock ) begin
       R1 <= R1 - 1;
       R2 <= R1;
   end
   ```

   (b) $R3 \leftarrow shr\ R3$

   Shift $R3$ right, and save the result into $R3$.

   ```
   always @( posedge clock )
       R3 <= R3 >> 1;
   ```

   (c) If $(S = 0)$ then $(R0 \leftarrow shr\ R0)$ else $(R0 \leftarrow shl\ R0)$

   If S is 0, shift $R0$ right and save the result into $R0$; otherwise, shift $R0$ left and save the result into $R0$.

   ```
   always @( posedge clock )
       if (S == 0) R0 <= R0 >> 1;
       else        R0 <= R0 << 1;
   ```

2. Construct a block diagram and an ASMD chart, and write a Verilog program, which controls a machine which dispenses a can of soda. The machine will accept pennies, nickels, dimes and quarters. Only one coin will go through the machine at a time, and that coin will be detected for exactly one clock cycle. The dispense output should go high for one clock cycle when the total amount of money inserted is equal to or greater than 60 cents. If the total is greater than 60 cents, the excess is held in the register to be used for the next soda. For example, if 75 cents is deposited, the machine should dispense a soda, and leave 15 cents towards the next soda.

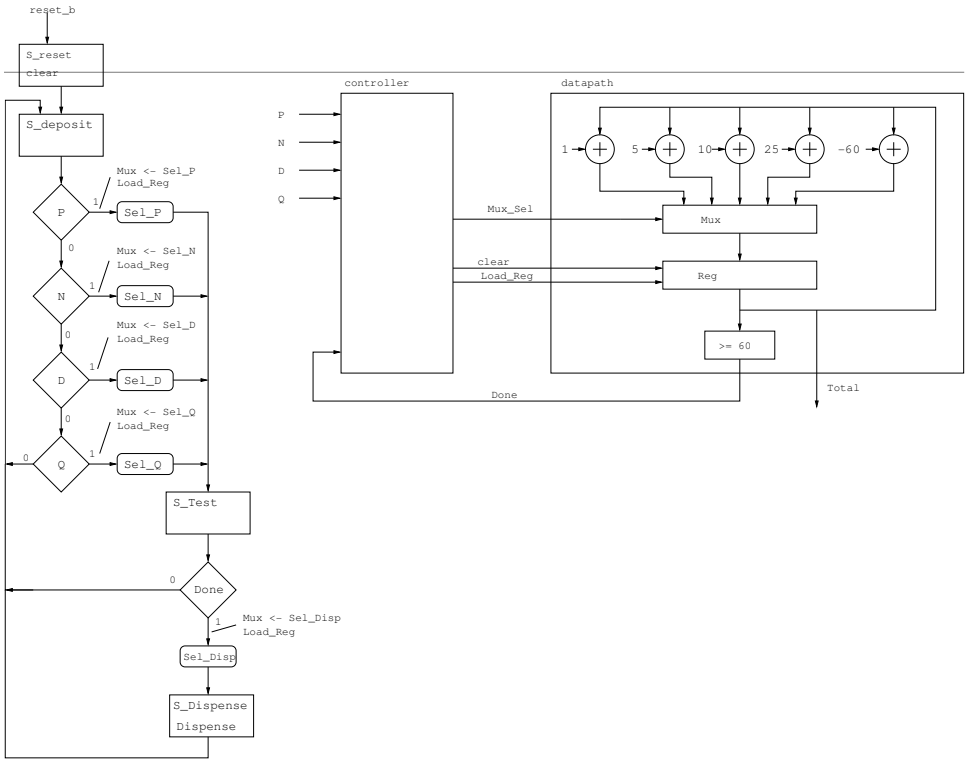
   The datapath should consist of a register to hold the total amount, a combinational circuit which can add the amount deposited to the register and subtract the cost of the soda from the register, and a display showing how much money is available for the soda.

   The controller should have a reset input to reset the count to zero.

   There are many ways to solve problems like this. Here is the way I did it.

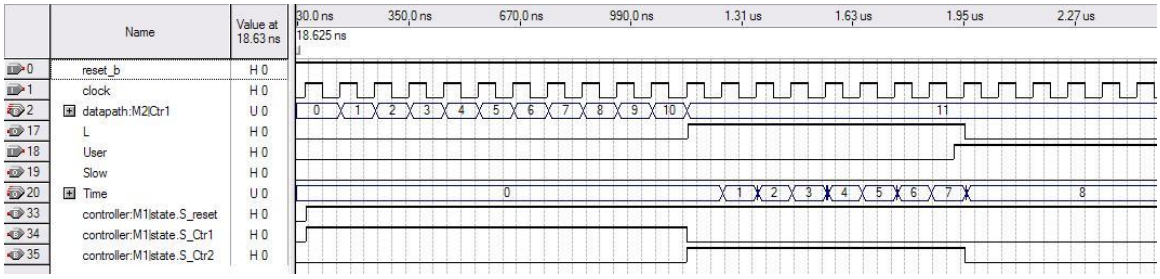
   For the datapath, I used one register to hold the total amount deposited. When a penny comes in, I add 1 to the total; when a nickel comes in I add 5 to the total, when a dime comes in I add 10 to the total, and when a quarter comes in I add 25 to the total. When a soda is dispensed I subtract 60 from the total. I have a comparator which checks to see if at least 60 cents has been deposited.

   The controller has four states: `S_reset` to clear the register which holds the amount deposited; `S_deposit` which checks to see if a coin has been deposited, and `S_Test` which checks to see if at least 60 cents has been deposited, and `S_Dispense` which dispenses the soda. (You could do this with one fewer states if Dispense were a Mealy output instead of a Moore output.)



```
module hw13_p2( input clock, reset_b, P, N, D, Q,
                output [6:0] Total, output Dispense);

wire [2:0] Mux_Sel;
wire Done, clear, load;

controller M1 ( clock, reset_b, P, N, D, Q, Done,
                clear, load, Dispense, Mux_Sel);

datapath M2 ( clock, clear, load, Mux_Sel,
              Total, Done);

endmodule


module controller (input clock, reset_b, P, N, D, Q, Done,
                   output reg clear, load, Dispense,
                   output reg [2:0] Mux_Sel );

parameter S_reset = 2'b00, S_deposit  = 2'b01,
          S_test  = 2'b10, S_dispense = 2'b11;

parameter Sel_P = 3'b000, Sel_N = 3'b001, Sel_D = 3'b010,
          Sel_Q = 3'b011, Sel_Disp = 3'b100;

reg [1:0] state, next_state;

always @( posedge clock, negedge reset_b )
    if (~reset_b) state <= S_reset;
    else state <= next_state;

/* Next state combinational block */
always @( state, Done, P, N, D, Q )
    case (state)
        S_reset:     next_state = S_deposit ;
        S_deposit :  if (P || N || D || Q) next_state = S_test;
                     else next_state = S_deposit;
        S_test:      if (Done) next_state = S_dispense;
                     else next_state = S_deposit;
        S_dispense:  next_state = S_deposit;
    endcase

/* Output combinational block */
always @( state, P, N, D, Q, Done) begin
    Mux_Sel  = 3'hx;
    clear    = 1'b0;
    load     = 1'b0;
    Dispense = 1'b0;
```

```
    case (state)
        S_reset:    clear = 1'b1;
        S_deposit:  if (P)      begin Mux_Sel = Sel_P;    load = 1'b1; end
                    else if (N) begin Mux_Sel = Sel_N;    load = 1'b1; end
                    else if (D) begin Mux_Sel = Sel_D;    load = 1'b1; end
                    else if (Q) begin Mux_Sel = Sel_Q;    load = 1'b1; end
        S_test:     if (Done)   begin Mux_Sel = Sel_Disp; load = 1'b1; end
        S_dispense: Dispense = 1'b1;
    endcase
end
endmodule

module datapath(input clock, clear, load, input [2:0] Mux_Sel,
                output reg [6:0] Total, output Done);

wire [6:0] mux_out;

parameter Sel_P = 3'b000, Sel_N = 3'b001, Sel_D = 3'b010,
          Sel_Q = 3'b011, Sel_Disp = 3'b100;

assign mux_out = (Mux_Sel == Sel_P)    ? Total + 7'd1  :
                 (Mux_Sel == Sel_N)    ? Total + 7'd5  :
                 (Mux_Sel == Sel_D)    ? Total + 7'd10 :
                 (Mux_Sel == Sel_Q)    ? Total + 7'd25 :
                 (Mux_Sel == Sel_Disp) ? Total - 7'd60 :
                 Total;

/* Enough money to dispense when Total >= 60 cents */
assign Done = (Total >= 7'd60);

always @( posedge clock )
        if (clear) Total <= 7'd0;
        else if (load) Total <= mux_out;
        else Total <= Total;

endmodule
```



3. Design a reaction timer system, which measures the amount of time elapsed between turning on a light and the user pressing a button. The system has three inputs: a 1 kHz clock, a reset button, and the user button B. It has three outputs: a one-bit output L to turn on an LED, a 12-bit output T to display the reaction time, and a one-bit output S which is activated if the user is too slow. After the reset button is pushed, the system waits for 10 seconds, then activates the L output, and starts counting on the T register. When the user presses the B button, the T value is held. If the user is too slow, and fails to press the button within 2 seconds, the S output is activated, and the T register displays 2,000.

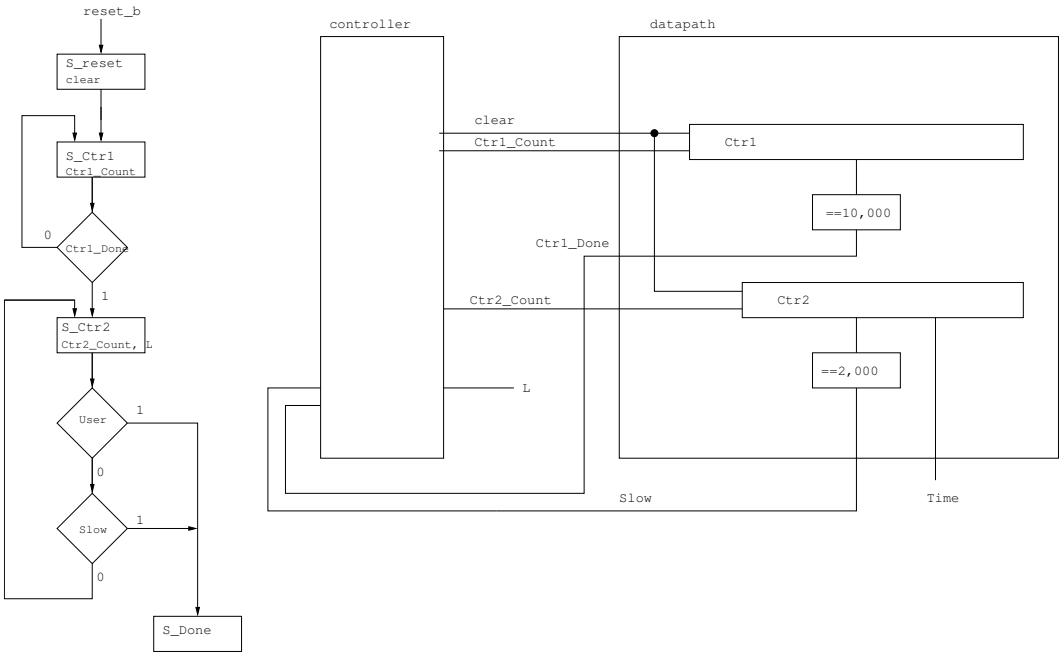
   Construct a block diagram (controller and datapath) and an ASMD chart, and write a Verilog program, to implement the reaction timer.

   Again, there are many ways to do this. Here is what I did.

   The datapath has two counters, one to count to 10,000 (to way 10 seconds), and the other to count to 2,000 (to wait 2 seconds). The datapath outputs a signal `Ctr1_Done` when Ctr1 has counted to 10,000; `Slow`, when the user fails to press the button in 2 seconds; and the reaction time.

   The controller has four states: `S_reset` which resets the two counters, `S_Ctr1` when Ctr1 counts, `S_Ctr2` when Ctr2 counts, and `S_Done` when the user has pushed the button or the time has elapsed. Ctr1 counts until the datapath asserts `Ctr1_Done`. Ctr2 counts until either the user pushes the button (B is asserted), or 2 seconds elasped (Slow is asserted).

   For testing, I had Ctr1 time out after 10 counts (instead of 10,000), and Slow was asserted after 20 counts (instead of 2,000).



```
module hw13_p3( input clock, reset_b, User,
                output [11:0] Time, output L, Slow);

wire clear, Ctr1_Count, Ctr2_Count, Ctr1_Done;

controller M1 ( clock, reset_b, Ctr1_Done, User, Slow,
                clear, Ctr1_Count, Ctr2_Count, L);

datapath M2 ( clock, clear, Ctr1_Count, Ctr2_Count,
              Time, Ctr1_Done, Slow);

endmodule


module controller (input clock, reset_b, Ctr1_Done, User, Slow,
                   output reg clear, Ctr1_Count, Ctr2_Count, L);

parameter S_reset = 2'b00, S_Ctr1 = 2'b01,
          S_Ctr2  = 2'b10, S_Done = 2'b11;

reg [1:0] state, next_state;

always @( posedge clock, negedge reset_b )
    if (~reset_b) state <= S_reset;
    else state <= next_state;

/* Next state combinational block */
always @( state, Ctr1_Done, User, Slow )
    case (state)
        S_reset: next_state = S_Ctr1;
        S_Ctr1:  if (Ctr1_Done) next_state = S_Ctr2;
                 else next_state = S_Ctr1;
        S_Ctr2:  if (User) next_state = S_Done;
                 else if (Slow) next_state = S_Done;
                 else next_state = S_Ctr2;
        S_Done:  next_state = S_Done;
    endcase

/* Output combinational block */
always @( state ) begin
    clear      = 1'b0;
    Ctr1_Count = 1'b0;
    Ctr2_Count = 1'b0;
    L          = 1'b0;
    case (state)
        S_reset:  clear = 1'b1;
        S_Ctr1:   Ctr1_Count = 1'b1;
        S_Ctr2:   begin Ctr2_Count = 1'b1; L = 1'b1; end
```

```
        default:  ;
    endcase
end
endmodule

module datapath(input clock, clear, Ctr1_Count, Ctr2_Count,
                output [11:0] Time, output Ctr1_Done, Slow);

reg [13:0] Ctr1;
reg [11:0] Ctr2;

assign Time = Ctr2;

assign Ctr1_Done = (Ctr1 >= 14'd10000) ? 1'b1 : 1'b0;
assign Slow = (Ctr2 >= 12'd2000) ? 1'b1 : 1'b0;

always @( posedge clock ) begin
        if (clear) begin  Ctr1 <= 14'd0; Ctr2 <= 12'd0;  end
        if (Ctr1_Count) Ctr1 <= Ctr1 + 14'd1;
        if (Ctr2_Count) Ctr2 <= Ctr2 + 12'd1;
    end

endmodule
```