Switching Circuits and Logic Design

Verilog Lab Questions

November 18th, 2013

(Deadline: December 6th)

In this problem set, we are going to introduce an algorithm calling "double-dabble" that converts binary numbers to binary coded decimal (BCD) numbers. We will design a circuit to perform this algorithm and use gate-level design approach to implement the circuit using verilog.

In the following problems about the code written in verilog, **you can only apply the gate level structure to design.** Moreover, only **AND, OR, and NOT** gates, no other gates, are allowed to be used.

The basic component in a double-dabble conversion circuit is a four-bit *add-3 module,* whose relationship between the input *X* and the output *Y* is shown in the following truth table. Symbol *D* means "don't-care."

X_3	X_2	X_1	X_0	Y_3	Y_2	Y_1	Y_0	X_3	X_2	X_1	X_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1
0	0	0	1	0	0	0	1	1	0	0	1	1	1	0	0
0	0	1	0	0	0	1	0	1	0	1	0	*D*	*D*	*D*	*D*
0	0	1	1	0	0	1	1	1	0	1	1	*D*	*D*	*D*	*D*
0	1	0	0	0	1	0	0	1	1	0	0	*D*	*D*	*D*	*D*
0	1	0	1	1	0	0	0	1	1	0	1	*D*	*D*	*D*	*D*
0	1	1	0	1	0	0	1	1	1	1	0	*D*	*D*	*D*	*D*
0	1	1	1	1	0	1	0	1	1	1	1	*D*	*D*	*D*	*D*

The input of the converter ranges from $(0000)_2$ to $(1001)_2$. The converter checks whether the decimal value of the input is greater than 4 or not. If the input is less than or equal to 4, the output follows the input. If the input is greater than 4, the output is given by $Y = X + 3$.

1. **[20 points]** (MA***dd3 module)*** Use Karnaugh maps to find minimum SOP of Y_3, Y_2 ,Y_1 and Y_0 to implement the converter.

$$\begin{cases} Y_3 = ? \\ Y_2 = ? \\ Y_1 = ? \\ Y_0 = ? \end{cases}$$

2. **[20 points]** Using the result you have derived in problem 1, design the ***MAdd3 module*** by filling the blanks underlined or remarked as "TO BE FILLED" in the code below.

MAdd3.v

```
module MAdd3 ( x, y );
  input [3:0] x;
  output [3:0] y;
  wire [1:0] t3;
  wire _____ t2;
  wire _____ t1;
  wire _____ t0;

  wire [3:0] xp;
  not n3 (xp[3], x[3]);
  not n2 (xp[2], x[2]);
```

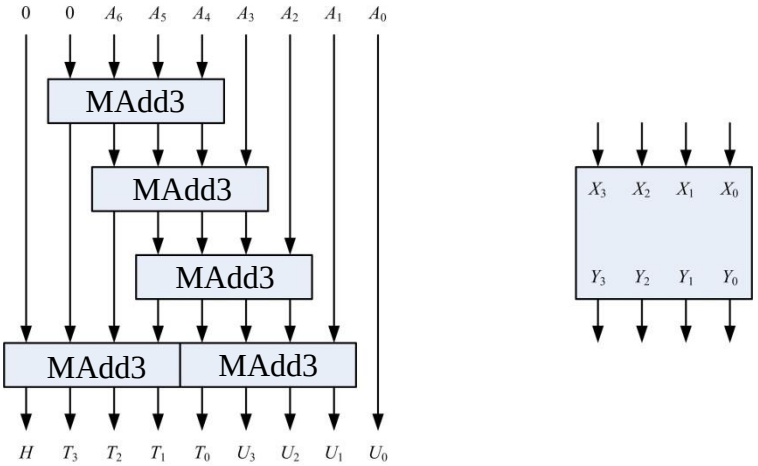
```
  not n1 (xp[1], x[1]);
  not n0 (xp[0], x[0]);

  /* and a32 (t3[2], x[3]); */
  and a31 (t3[1], x[2], x[0]);
  and a30 (t3[0], x[2], x[1]);
  or o3 (y[3], t3[1], t3[0]); //Check the y[3] is right answer or not.

  /* Blanks TO BE FILLED */

endmodule
```

3. **[20 points]** Verify your design in problem 2 by zooming out to view all of the simulation result (nWave) and capturing the simulation result window. Show the inputs and outputs in the same timing diagram. Figure out what outputs are when *X* are invalid values, $(A)_{16}$ ~ $(F)_{16}$.

MAdd3TB.v

```
module MAdd3TB;
  reg [3:0] x;
  wire [3:0] y;

  MAdd3 add31 (.x(x), .y(y));

  initial #256 $finish;

  initial begin
     $fsdbDumpfile("MAdd3.fsdb");
     $fsdbDumpvars;
    x = 4'h0; #16 x = 4'h1; #16 x = 4'h2; #16 x = 4'h3;
    #16 x = 4'h4; #16 x = 4'h5; #16 x = 4'h6; #16 x = 4'h7;
    #16 x = 4'h8; #16 x = 4'h9; #16 x = 4'ha; #16 x = 4'hb;
    #16 x = 4'hc; #16 x = 4'hd; #16 x = 4'he; #16 x = 4'hf;
  end

endmodule
```

We are ready to construct the whole design using verilog now. The design of the 7-bit binary-to-BCD converter is shown in the following left figure. (The right figure shows the ***MAdd3 module*** in the left figure.)



In this design, the 7-bit binary input *A* is converted into (*H*, *T*, *U*), and *H*, *T*, *U* represents the 100, 10, and 1 parts of the BCD code, respectively.

4. **[20 points]** Design the 7-bit binary-to-BCD converter by filling the blanks remarked as "TO BE FILLED" in the code shown as follows. You need to apply the module you have designed in problem 2.

Mbinary2BCD.v

```
module Mbinary2BCD ( a, h, t, u );
  input [6:0] a;
  output h;
  output [3:0] t;
  output [3:0] u;
  /* Blanks TO BE FILLED */
endmodule
```

Hint: Use "Net Concatenation" to assign the input signals for each MAdd3 module.

5. **[20 points]** Verify your design by capturing the simulation result window using the testbench shown below. Does this design implement the binary-to-BCD converter correctly according to your test?

Mbinary2BCDTB.v

```
module Mbinary2BCDTB;
    reg [6:0] a;
    wire hunds;
    wire [3:0] tens;
    wire [3:0] units;

    Mbinary2BCD b2bcd1 (.a(a), .h(hunds), .t(tens), .u(units));

    initial #256 $finish;

    initial begin
        $fsdbDumpfile("Mbinary2BC.fsdb");
        $fsdbDumpvars;
        a = 7'h0  (1)  ;
        #32 a = 7'h1  (2)  ;
        #32 a = 7'h2  (3)  ;
        #32 a = 7'h3  (4)  ;
        #32 a = 7'h4  (5)  ;
        #32 a = 7'h5  (6)  ;
        #32 a = 7'h6  (7)  ;
        #32 a = 7'h7  (8)  ;
    end

endmodule
```

Note: Use your student ID to fill the labeled underline blanks. For example,
if your student ID is B00901789, then you should fill blank (1) with 0, (2) with 0, (3) with 9, (4) with 0, (5) with 1, (6) with 7, (7) with 8, and (8) with 9;
if your student ID is B99B01987, then you should fill blank (1) with 9, (2) with 9, (3) with B, (4) with 0, (5) with 1, (6) with 9, (7) with 8, and (8) with 7.

More about "double-dabble" algorithm:
http://en.wikipedia.org/wiki/Double_dabble