硬體描述語言設計　Homework Assignment #4

Due 11/24/2017 (Friday)

Q1. (30%)

Design a universal synchronous 4-bit counter with the following features

module universal_counter (data_in, load, clear, mode, enable, count)

input [3:0] data_in;　　/* initial value of the counter */

input load;　　/* if load =1, count[3:0] is set to data_in[3:0] at the positive edge of the clock*/

input clear;　　/* if clear = 1, count[3:0] is reset at the positive edge of the clock*/

　　/* clear has the priority over the load signal */

input mode;　　/* up counting if mode = 1, down counting if otherwise */

input enable;　　/* counter counts when enable = 1 */

output [3:0] count;　　/* counter output */

/* priority of control signals: clear > load > enable */

For input stimulus, all signals (except clk) change at the negative edge of the clock

Counter changes at the positiv eedge of the clock

Clk 0 means the initial state, set clock period equal to 10 time units and the initial value of clk as 1

Clk #	0	1	2	…	7	…	10	…	15	16	…	24	25	…	30	
clear	0	0	0		0		0		1	0		1	0		0	
load	0	1	0		0		0		1	0		0	0		0	
enable	1	1	1		0		1		1	1		1	1		0	
data_in	0	6	0		0		0		8	8		8	4		4	
mode	1	1	1		1		0		0	0		1	1		1	

Q2. (30%)

8-bit Barrel shifter design

module barrel_shifter (data_in, mode, size, data_out)

input [7:0] data_in;　　/* input of the barrel shifter */

input [1:0] mode;　　/* mode control, 00 shift left, 01 logic shift right

　　10 rotate left, 11 rotate right*/

input [1:0] size;　　/* the displacement size of the shift-rotate opeartions, 0 ~ 3 bits */

output [7:0] data_out;　　/* output of the shifter */

For input stimulus,

time	0	5	10	15	20	25	30	35	
data_in	8'h6B	8'h6B	8'h6B	8'h6B	8'h6B	8'h6B	8'h6B	8'h6B	
mode	00	00	10	10	01	01	11	11	
size	00	10	01	11	01	10	11	00	

Q3. (40%)

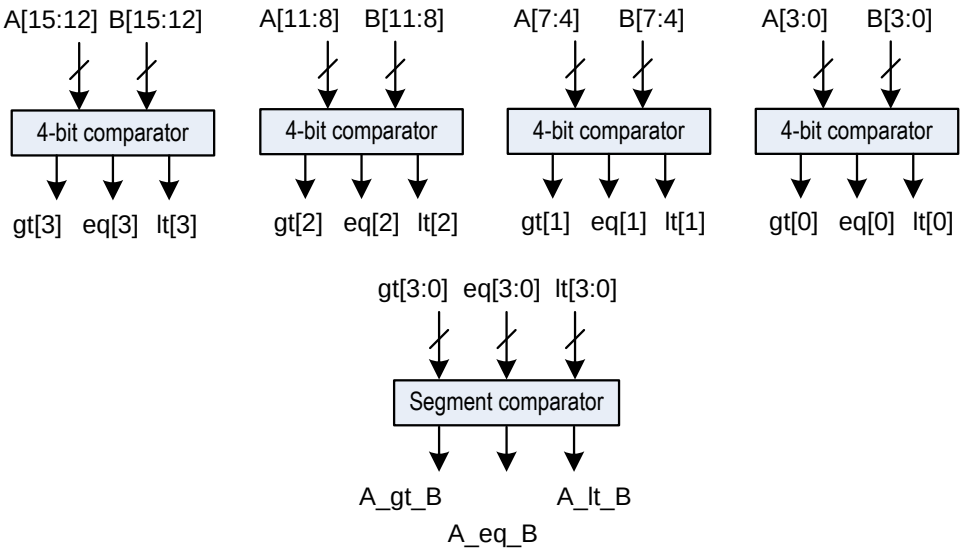
First design a 4-bit magnitude comparator. Next construct a 16-bit magnitude comparator design using following block diagram, i.e., Instantiate four 4-bit comparators and design a second level

“segment comparator”. Use “name mapping” when you instantiate the 4-bit magnitude comparator modules.

```
/* 4-bit magnitude comparator module*/
module comp4b(data_a, data_b, gt, eq, lt)
input [3:0] data_a, data_b;
output gt;      /* gt = 1 if data_a > data_b */
output eq;      /* eq = 1 if data_a == data_b */
output lt;      /* gt = 1 if data_a < data_b */

/* segment comparator takes the output from four 4-bit magnitude comparator modules and
generate the final 16-bit magnitude comparison results */
/* this design shows how to construct a large comparator from small comparators */
module seg_comp(gt, eq, lt, A_gt_B, A_eq_B, A_lt_B);
input [3:0] gt;
input [3:0] eq;
input [3:0] lt;
output A_gt_B, A_eq_B, A_lt_B;
/* hint: each 16-bit input operand is divided into four 4-bit operands with different weightings.
Four separate 4-bit comparisons are conducted first. A higher weighted comparator always prevails
over a lower weighted comparator. For example, if gt[3] from the highest weighted comparator
(A[15:12] vs B[15:12]) equals 1, then A_gt_B = 1 regardless of the comparison results from the
remaining three comparators. If eq[3] ==1, the comparison result of the second highest weighted
comparator (A[11:8] vs B[11:8]) would dominate the comparison result. By the same token, the
result of A[7:4] vs B[7:4] prevails when gt[3]=gt[2]=1. Use this principle to design your segment
comparator */
```

For input stimulus (initial begin A = 16’h04F8; B = 4’b04F7; #10 B=4’b04FA; #10 A=4’b04FA, #10 B=4’b24FA; end)



Q4. (加分題，30%) 你可以選擇不做這一題

Design a 8-bit ALU (Arithmetic Logic Unit) supporting the following instructions

```
module alu8b (Y, C, Z, A, B, code)
input [7:0] A, B;     /* two 8-bit wide input operands, both are unsigned (positive) number */
input [2:0] code;     /* 3-bit instruction code, MSB = 0 for logical operations, MSB = 1 for arithmetic
                      operations */
output [7:0] Y;       /* 8-bit output of the ALU */
output Z;             /* status flag "zero", Z = 1 if Y == 0 */
output C;             /* status flag "carry", In addition/increment, C = 1 if A+B > 255
                                                In subtraction/decrement, C = 1 if A-B < 0
                                                In logic operations, C is always set to 0 *//
:
:
endmodule
```

Instruction type	code[2:0]	operations	Status update
Logical	000	(Bitwise OR) Y = A | B;	Z (C is always 0)
	001	(bitwise AND) Y = A & B;	Z (C is always 0)
	010	(bitwise XOR) Y = A ^ B;	Z (C is always 0)
	011	(negation) Y = ~A;	Z (C is always 0)
arithmetic	100	(Addition) Y = A + B;	Z, C
	101	(Increment) Y = A + 1;	Z, C
	110	(subtraction) Y = A – B;	Z, C
	111	(decrement) Y = A – 1;	Z, C

- "Status update" means Z and C values should be re-evaluated (updated) if Y changes