Solver: Chia Su Chi Faith

Email Address: fchia001@e.ntu.edu.sg

1. (a)

Possible answer #1

```
module satadd(input signed [3:0] a,
b, output [1:0] res);
always @ * begin
      if((a+b)>3)
            res = 3;
      else if ((a+b)<0)
            res = 0;
      else
            res = a+b;
end
endmodule
```

Possible answer #2

```
module satadd(input signed [3:0] a,
b, output [1:0] res);
assign res = ((a+b)<0) ? 0 :
(a+b)>3) ? 3 : (a+b);
endmodule
```

Comments:

- Cannot change port declaration, so clk/rst not used. Combinational always block (always @ *)
- Output `res` is 2 bits unsigned – range from 0 to +3
- `assign` statement can be nested, but it's a lot easier to read answer #1.

(b) (i) `satadd uut(.a(a_sim), .b(b_sim), .res(res_sim);`

(ii) `always #10 b = b+3;`

(iii)
```
b = 4'b1011; a = 4'b0000;
#10 a = 4'b0100;
#10 a = 4'b1111;
#20 a = 4'b1010;
#10 a = 4'b1110;
#10 a = 4'b0111;
$finish;
```

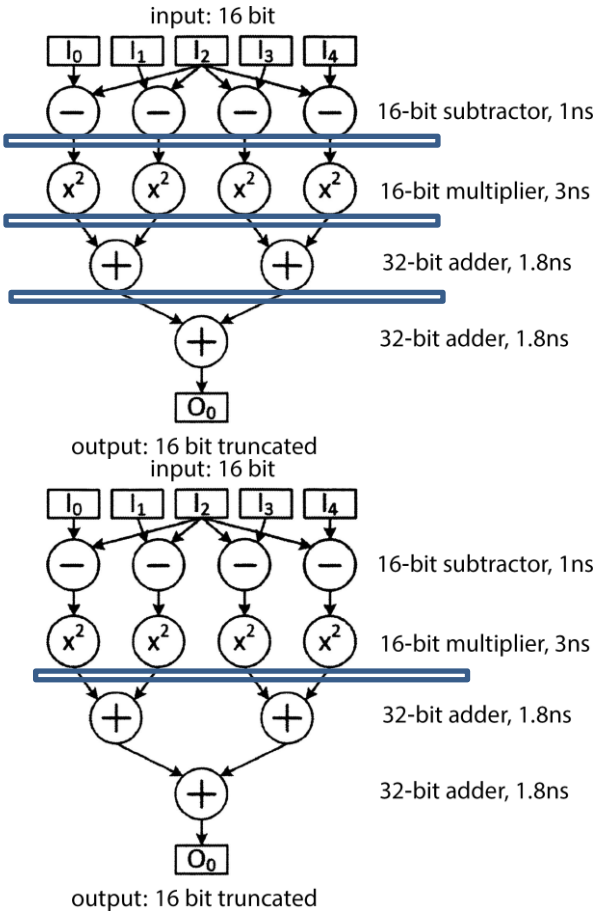
(c)

a_sim[3:0]	**0000**	**0100**	**1111**		**1010**	**1110**	**0111**
Decimal value of a_sim	0	+4	-1		-6	-2	+7
b_sim[3:0]	**1011**	**1110**	**0001**	**0100**	**0111**	**1010**	**1010**
Decimal value of b_sim	-5	-2	+1	+4	+7	-6	-3
Actual value of a+b	-5	+2	0	+3	+1	-8	+4
res_sim[1:0]	**00**	**10**	**00**	**11**	**01**	**00**	**11**
Actual res	0	+2	0	+3	+1	0	+3

Comments:

- Non-bolded rows show my working, don't include in the actual diagram.
- `b_sim` is unsigned, so it increments by 3 as if it were a normal unsigned number, but the module logic will read this input as a signed number.

2. (a)



(i)
Max throughput = 1 output every 3ns.
Max operating frequency = 1/3ns = 333.33MHz
Latency = 12ns (4 cycles * 3ns per cycle) from input to output.

(ii)
Max throughput = 1 output every 4ns.
Max operating frequency = 1/4ns = 250MHz
Latency = 8ns (2 cycles * 4ns per cycle) from input to output, an improvement by 4ns over the circuit in (i).

(iii)
```
module gradient(input [15:0] I0, I1, I2, I3, I4, input clk, rst,
                        output reg [15:0] O);
      reg [31:0] a, b, c, d, e;
      always @ (posedge clk) begin
            if(rst) begin
                  a <= 0; b <= 0; c <= 0; d <= 0; O <= 0;
            end else begin
                  a <= (I0-I2)*(I0-I2);
                  b <= (I1-I2)*(I1-I2);
                  c <= (I3-I2)*(I3-I2);
                  d <= (I4-I2)*(I4-I2);
                  e <= (a+b)+(c+d);
            end
      end
      assign O = e[15:0]; // truncation to keep 16 LSB
      endmodule
```

(b) Inputs *a*, *b*, *c*: 1 integer, 7 fractional; output *out*: 2 integer, 10 fractional; range of *out* is +/-1.9921875. The range of a*b+c is also +/-1.9921875. Hence, the code will produce the correct truncated result.

Comment: Deciding how many integer and fractional bits to use requires a tradeoff between accuracy and range. More integer bits for range, or more fractional bits for accuracy. Since the question asks for highest possible accuracy, limit the number of integer bits to a minimum.

Term	Integer, Fractional bits	Actual Range given that a, b, c = +/-0.995
`a, b, c`	1 int, 7 frac	-1 (1.0000000_2) to +0.9921875 (0.1111111_2)
`a*b`	(1int7frac)*(1int7frac) = 2int14frac See module 4, slide 30	-0.9921875 to +1 Most negative number is -1*+0.9921875 Most positive number is -1*-1
`{c[7], c, 7'b0000000}`	2int, 14frac	-1 to +0.9921875 Sign extension and padding does not change this term's actual value. This is done to align c with a*b so that the addition with a*b yields a correct result.
`tout`	2int14frac + 2int14frac = 3int14frac	+/-1.9921875 While 3 integer bits are used, realistically the range of values of a*b and c are such that 100.x is impossible.
`out`	2int10frac	+/-1.9921875

3. (a) (i)
```
SLTI: begin
        ALUop = 4'b1101; Sel_ALUsrc1 = 1'b0;
    end
```

Control signal	Value	Comment
ALUop	4'b1101	No similar instruction in the table. Also, in 3aii, SLTI = 13 = 1101_2.
PC_select	1'b0	S1 mux. Enabled only if next PC != PC+1.
sel_ALUsrc1	1'b1	S2 mux. Enabled if immed value is used by the ALU.
mem2Reg	1'b0	S3 mux. Enable if memory data is written back to register file, disable if ALUResult is used. Since 1 or 0 is written back, no usage of memory; disable.
RFwriteEnab	1'b1	Enabled if the register file is modified. Since register file destination address will be set to 1 or 0, enable.
memWriteEnab	1'b0	Enable if there is a write back to memory. No write back to memory, only write is to the register file.
memEnab	1'b0	Enable if there is access to memory. No reading from memory, only read is to the register file.

(ii)

Possible answer #1
```
SLTI: begin
   if (ALUData2 < ALUData1)
      ALUResult = 1;
   else
      ALUResult = 0;
end
```

Possible answer #2
```
SLTI: ALUResult = (ALUData2 <
ALUData1) ? 1 : 0;
```

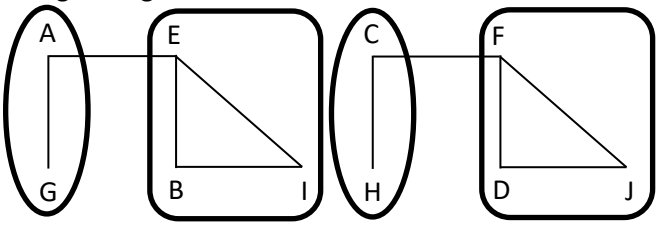
Comment: immed value is at ALUData1 (output of S2 mux), RF[rs] is at ALUData2

4. (a)

Present State	w_2w_1 00	01	11	10	Output z
A	**A**	G	-	E	0
B	-	I	**B**	E	0
E	A	-	B	**E**	0
G	A	**G**	C	-	0
I	A	**I**	B	-	0
C	-	F	**C**	H	1
D	**D**	F	-	J	1
F	D	**F**	C	-	1
H	A	-	C	**H**	1
J	D	-	C	**J**	1

Tip: Sort the table according to output, it's much easier to compare same-output states this way.

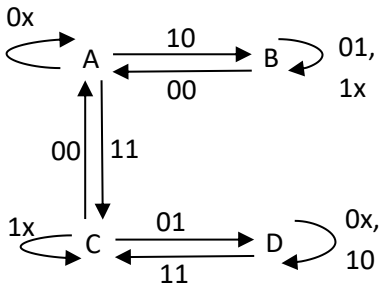
Merger Diagram:



Reduced table:

Present State	w_2w_1 00	01	11	10	Output z
A	**A**	**A**	C	B	0
B	A	**B**	**B**	**B**	0
C	A	D	**C**	**C**	1
D	**D**	**D**	C	**D**	1

(b)



Assigning A = 00, B = 01, C = 10, D = 11, Excitation Table:

Present State, y_2y_1	$y_2^+y_1^+$ if w_2w_1 is… 00	01	11	10	Output z
A = 00	**00**	**00**	10	01	0
B = 01	00	**01**	**01**	**01**	0
D = 11	**11**	**11**	10	**11**	1
C = 10	00	11	**10**	**10**	1

Tip: Get a hazard free assignment by drawing out the state transition diagram then moving the letters around to remove diagonal arrows. Don't just assign A = 00, B = 01, C = 11, D = 10 first.
Tip: **Always** check that the rows/columns go from 00 -> 01 -> 11 -> 10, changing one bit at a time.
Comment: There are multiple possible combinations of state assignments and so multiple correct answers.

y_2y_1	y_2^+ when w_2w_1 is 00	01	11	10	Output z
00	0	0	1	0	0
01	0	0	0	0	0
11	1	1	1	1	1
10	0	1	1	1	1

2 square loops with 4 1s
1 horizontal loop with 4 1s
1 vertical loop with 2 1s

y_2y_1	y_1^+ when w_2w_1 is 00	01	11	10	Output z
00	0	0	0	1	0
01	0	1	1	1	0
11	1	1	0	1	1
10	0	1	0	0	1

4 vertical loops with 2 1s
4 horizontal loops with 2 1s

Output = $z = y_2$

$y_2^+ = y_2y_1 + y_2w_1 + y_2w_2 + y_1'w_2w_1$

$y_1^+ = y_2y_1w_2' + y_2y_1w_1' + y_2'y_1w_1 + y_2'y_1w_2 + w_2'w_1y_1 + w_2'w_1y_2 + w_2w_1'y_2' + w_2w_1'y_1$

$= y_2y_1(w_2' + w_1') + y_2'y_1(w_1 + w_2) + w_2'w_1(y_1 + y_2) + w_2w_1'(y_2' + y_1)$

(c) A static hazard occurs when an input changes, causing the output to temporarily change when it should hold its value.

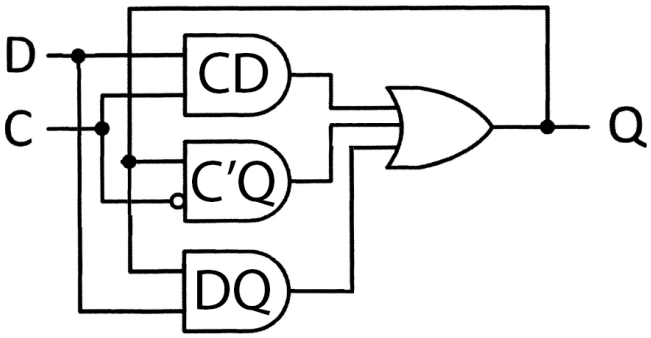
From the circuit, $Q^+ = D*C + C'*Q$. The static hazard occurs because when C changes its value, the inputs to the AND gates do not change at the exact same time. The C input at the first AND gate will change before the C input at the second AND gate. This is due to the inverter at the second AND gate. There are different path lengths for the C input into the AND gates.

Present State	Q+ if DC is			
Q	00	01	11	10
0	0	0	1	0
1	1	0	1	1

Eliminate the static hazard by including all prime implicants, ie add the term D*Q (dotted loop).

Corrected equation: $Q^+ = D*C + C'*Q + D*Q$

Corrected circuit:



Comment: this question is exactly the same as the example for static hazard used in the lecture slides. (Module 9, slide 12)

Example: if D = 1, C = 1, then Q = 1. If C changes to 0, there will be a moment where

- D*C = 0, (D = 1, C = 0 since the value of C is updated to 0 already)
- Q*C' = 0 (Q = 1, C' = 0 since the new value of C hasn't yet propagated through the inverter)
- D*C + Q*C = 0 which is incorrect; should be 1 instead; see table below.

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺