SC1005 Digital Logic Tutorial 1

Introductory concepts and number systems

1. What is the largest decimal number that can be represented using 16 bits in
   a) binary?
   b) BCD?

2. How many bits are needed to represent a decimal integer value not exceeding 350000_{10}?

3. Give the BCD representation of these decimal numbers:
   a) 285
   b) 47.19

4. Give the decimal value for each of these representations:
   a) $0011\ 1000_2$
   b) 0011 1000 (in ASCII)

5. Perform the following conversions:
   a) 101111.0111_2 = ?$_{16}$ = ?$_{10}$
   b) $15C.38_{16}$ = ?$_8$ = ?$_{10}$
   c) 1435_{10} = ?$_{16}$ = ?$_2$
   d) 7436.11_8 = ?$_{16}$ = ?$_{10}$

6. Convert the decimal fraction 0.8254 into an 8-bit binary fraction of the form $0.b_{-1}b_{-2} \ldots b_{-8}$.

7. Determine the parity bit to be generated for each of the following code words before transmission. Assuming even parity is used.
   a) 0110011
   b) 0x43 (0x is a common notation for hexadecimal)
   c) 0100 0111 0011

Answers

1. a) 65535
   b) 9999

2. 19

3. a) 0010 1000 0101
   b) 0100 0111. 0001 1001

4. a) 56
   b) 8

5.
   a) $101111.0111_2 = 2F.7_{16} = 47.4375_{10}$
   b) $15C.38_{16} = 534.16_8 = 348.21875_{10}$
   c) $1435_{10} = 59B_{16} = 10110011011_2$
   d) $7436.11_8 = F1E.24_{16} = 3870.140625_{10}$

6. 0.11010011_2

7.
   a) 0
   b) 1
   c) 0

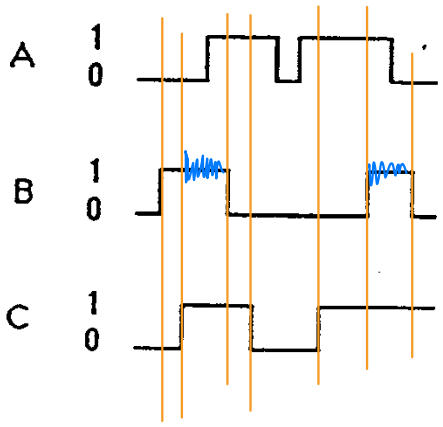
SC1005 Digital Logic Tutorial 2

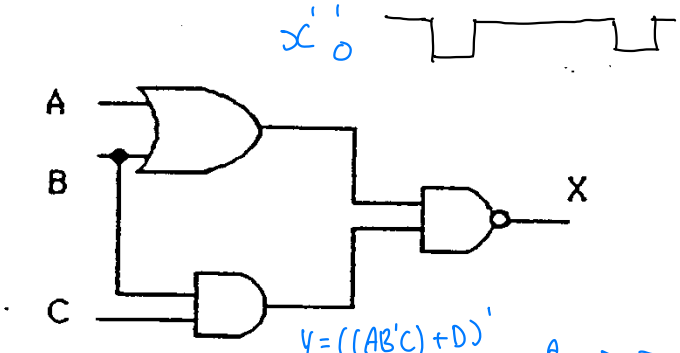
Logic gates and Boolean algebra

1. From the following figure,

    a) Express X as a Boolean function of inputs A, B and C.

    b) Sketch the output waveform at point X.





2. A logic circuit's output expression is Y = [(A B' C) + D]'.

    (a) Draw the logic circuit diagram for Y.

    (b) Construct the truth table for the circuit.

3. Simplify the following expressions using Boolean algebra:

    a) X = A'B'C'D' + A'B'CD' + A'BCD' + ABCD' + AB'CD'

    b) X = [AB (C+D)'AB]'

    c) X = A(AB)' + A'B'C + ABC

4. Show using Boolean algebra that the following equation is true.

    AB + ABC′D + ABDE′ + A′BC′E + A′B′C′E = AB + A′C′E

5. Implement the function (A+B)(C+D) using only

    a) Two-input NAND gates

    b) Two-input NOR gates

3a. $X = A'B'C'D' + A'B'CD' + A'BCD' + ABCD' + AB'CD'$

$= A'B'D'(C'+C) + BCD'(A'+A) + AB'CD'$

$= B'D'(A'+AC) + BCD'$

$= B'D'(A'+C) + BCD'$

$= A'B'D' + B'CD' + BCD'$

$= A'B'D' + CD'(B'+B)$

b. $X = (AB(C+D)'AB)'$

$= (AB)' + (C+D) + (AB)'$

$= A'+B' + C+D + A'+B'$

$= A'+B' + C+D$

c. $X = A(AB)' + A'B'C + ABC$

$= A(A'+B') + A'B'C + ABC$

$= AA' + AB' + A'B'C + ABC$

$= A(BC+B') + A'B'C$

$= A(C+B') + A'B'C$

$= AC + AB' + A'B'C$

$= C(A+A'B') + AB'$

$= CA + CB' + AB'$

4 $AB + ABC'D + ABDE' + A'BC'E + A'B'C'E$

$= AB + ABC'D + ABDE' + A'C'E(B+B')$

$= AB(1 + C'D + DE') + A'C'E$

$= AB + A'C'E$

Show using Boolean algebra that the following equation is true.

AB + ABC'D + ABDE' + A'BC'E + A'B'C'E = AB + A'C'E

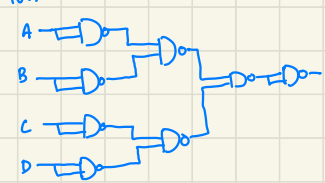
5. $(A+B)(C+D)$

Implement the function (A+B)(C+D) using only

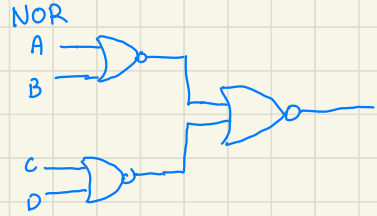
a) Two-input NAND gates

b) Two-input NOR gates

NAND



NOR



Answers

1. (a) X = [(A + B) B C]’

2. (b) There are 7 1’s in the Y column of the truth table

3. a) X = A'B'D' + CD'

   b) X = A' + B' + C+D

   c) X = AB' + AC + B'C

5. a) 8 two-input NAND gates needed.

   b) 3 two-input NOR gates needed.

SC1005 Digital Logic Tutorial 3

Digital arithmetic

1. Perform the following unsigned binary addition and subtraction.

   a. 1100110 + 1111001

   b. 11100011 - 1011101

2. Perform the following two's complement additions. Clearly indicate whether or not an overflow occurs.

   a. 11010100 + 11101011

   b. 10111111 + 11011111

   c. 01011101 + 00110001

   d. 01100001 + 00011111

3. Perform the following two's complement subtractions. Clearly indicate whether or not an overflow occurs. Check by converting to decimal values.

   a. 00110110 - 01000101

   b. 01110101 - 11010110

   c. 11010111 - 11101100

   d. 10000011 - 10001111

4. Perform the following unsigned binary multiplications. Verify with decimal values.

   a. 110101 x 1110

   b. 010110 x 1101

5. Perform the following signed 2's complement binary multiplications. Verify with decimal values.

   a. 110101 x 1110

   b. 010110 x 1101

SC1005 Digital Logic Tutorial 3

Digital arithmetic

1. Perform the following unsigned binary addition and subtraction.

   a. 1100110 + 1111001

   b. 11100011 - 01011101

2. Perform the following two's complement additions. Clearly indicate whether or not an overflow occurs.

   a. 11010100 + 11101011

   b. 10111111 + 11011111

   c. 01011101 + 00110001

   d. 01100001 + 00011111

3. Perform the following two's complement subtractions. Clearly indicate whether or not an overflow occurs. Check by converting to decimal values.

   a. 00110110 - 01000101

   b. 01110101 - 11010110

   c. 11010111 - 11101100

   d. 10000011 - 10001111

4. Perform the following unsigned binary multiplications. Verify with decimal values.

   a. 110101 x 1110

   b. 010110 x 1101

5. Perform the following signed 2's complement binary multiplications. Verify with decimal values.

   a. 110101 x 1110

   b. 010110 x 1101

Answers

1.

a. 11011111
b. 10000110

2.

a. 10111111 (no overflow)
b. 10011110 (no overflow)
c. 10001110 (overflow)
d. 10000000 (overflow)

3.

a. 11110001 (no overflow)
b. 10011111 (overflow)
c. 11101011 (no overflow)
d. 11110100 (no overflow)

4.

a. 1011100110
b. 100011110

5.

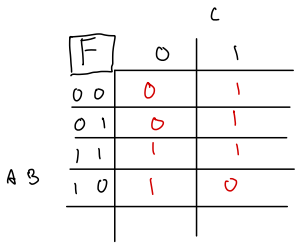
a. 000010110
b. 110111110

SC1005 Digital Logic Tutorial 4

Combinational Logic

1. Given the following truth table, determine:
   a. The canonical sum-of-minterms expression
   b. The minimum cost sum-of-products (SOP) expression from (a) using algebraic manipulation.
   c. The canonical product-of-maxterms expression
   d. The minimum cost product-of-sums (POS) expression from (c) using algebraic manipulation.
   e. The NAND gate only implementation of the minimum cost SOP expression.
   f. The NOR gate only implementation of the minimum cost POS expression.
   g. That the two expressions in (b) and (d) are identical using algebraic manipulation.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



a) $\Sigma_{ABC} = (m1, m3, m4, m6, m7)$
$= A'B'C + A'BC + AB'C' + ABC' + ABC$

b) $A'B'C + A'BC + AB'C' + ABC' + ABC$
$= A'C + AC' + ABC$
$= (A' + AB)C + AC'$
$= A'C + BC + AC'$

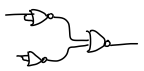
c) $\Pi_{ABC} = (M0, M2, M5)$
$= (A+B+C)(A+B'+C)(A'+B+C')$

d) $(A+B+C)(A+B'+C)(A'+B+C')$
$= (A+C)(A'+B+C')$

e) $A'C + BC + AC'$
$= \left((A'C)'(BC)'(AC')'\right)'$

f) $(A+C)(A'+B+C')$
$= \left((A+C)' + (A'+B+C')'\right)'$

NOR



$(A+A')' + (B+B)')' = (A'+B)'$
$= AB$

g) $(A+C)(A'+B+C')$
$= AA' + AB + AC' + CA' + CB + CC'$
$= C'A + CB + AB + CA'$
$= C'A + CB + CA'$

1.

a. F_{SOP} = A'B'C + A'BC + AB'C' + ABC' + ABC

b. F = AB + A'C + AC'

c. F_{POS} = (A+B+C).(A+B'+C).(A'+B+C')

d. F = (A+C).(A'+B+C')

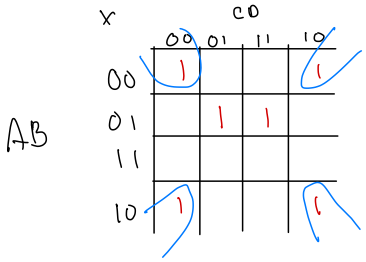
e. F = ((AB)'. (A'C)'. (AC')')'

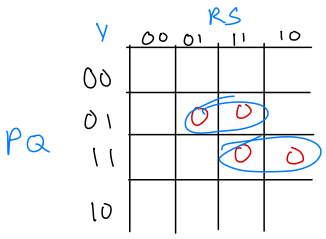
f. F = ((A+C)' + (A'+B+C')')'

1.

a. F_{SOP} = A'B'C + A'BC + AB'C' + ABC' + ABC
$= A'C(B' + B) + AC'(B' + B) + ABC$
$= A'C + AC' + ABC$
$= A'C + A(C' + BC)$
$= A'C + A(C' + B)$
$= A'C + AC' + AB$

$(A+C)(A'+B+C')$
$= AA' + AB + AC' + CA' + CB + CC'$
$= AC' + A'C + AB + CB$
$= AC' + A'C + AB + CBA' + CBA$
$= AC' + A'C + A'CB + AB + ACB$
$= AC' + A'C(1 + B) + AB(1 + C)$
$= AC' + A'C + AB$





Karnaugh maps

2. Simplify the following expressions using Karnaugh map:

   (a) X = AB'D' + A'BC'D + A'BCD + A'B'D' = B'D' + A'BD

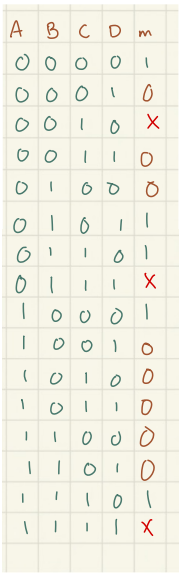
   (b) Y = (P+Q'+R+S')(P+Q'+R'+S')(P'+Q'+R+S')(P'+Q'+R'+S') = (P'+Q+S)(P+Q+R)

3. Simplify the Boolean function F together with the don't care condition d, using the K-map method. Give your answer in SOP.

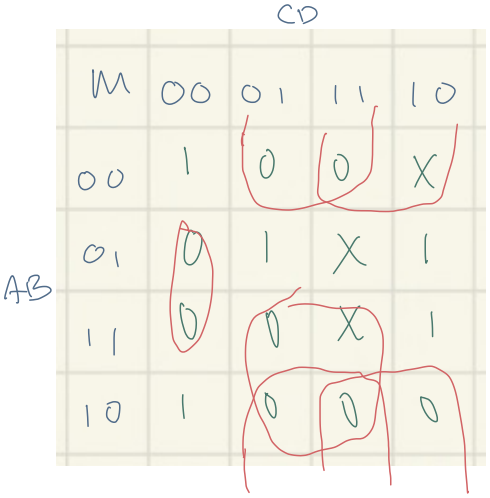
   F(A, B, C, D) = Σ m (0, 5, 6, 8, 14)

   d (A, B, C, D) = Σ m (2, 7, 15)

4. Repeat Question 3. Give your answer in POS.



A	B	C	D	m
0	0	0	0	1
0	0	0	1	0
0	0	1	0	X
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	X



SOP = B'C'D' + BC + A'BD

POS = (A'+D)(B+D')(B+C')(B'+C+D)

F = (A'+D')(B+D')(B+C')(B'+C+D)

Answers

1.

a. F_{SOP} = A'B'C + A'BC + AB'C' + ABC' + ABC

b. F = AB + A'C + AC'

c. F_{POS} = (A+B+C).(A+B'+C).(A'+B+C')

d. F = (A+C).(A'+B+C')

e. F = ((AB)'. (A'C)'. (AC')')'

f. F = ((A+C)' + (A'+B+C')')'

2. (a) X = B'D' + A'BD

   (b) Y = Q' + S'

3. (a) F = BC + A'BD + B'C'D'

   (b) F = (A'+D')(B+D')(B+C')(B'+C+D)

SC1005 Digital Logic Tutorial 5

1.

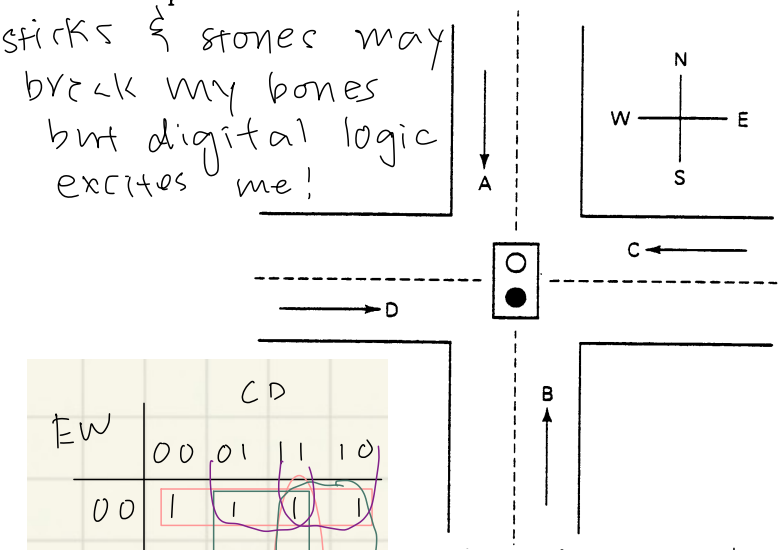
Figure 4-50 shows the intersection of a main highway with a secondary access road. Vehicle-detection sensors are placed along lanes *C* and *D* (main road) and lanes *A* and *B* (access road). These sensor outputs are LOW (0) when no vehicle is present and HIGH (1) when a vehicle is present. The intersection traffic light is to be controlled according to the following logic:

1. The east-west (E-W) traffic light will be green whenever *both* lanes *C* and *D* are occupied.
2. The E-W light will be green whenever *either C* or *D* is occupied but lanes *A* and *B* are not *both* occupied.
3. The north-south (N-S) light will be green whenever *both* lanes *A* and *B* are occupied but *C* and *D* are not *both* occupied.
4. The N-S light will also be green when *either A* or *B* is occupied while *C* and *D* are *both* vacant.
5. The E-W light will be green when *no* vehicles are present.

Using the sensor outputs *A*, *B*, *C*, and *D* as inputs, design a logic circuit to control the traffic light. There should be two outputs, N-S and E-W, that go HIGH when the corresponding light is to be *green*. Simplify the circuit as much as possible and show *all* steps.

A	B	C	D	EW	NS
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0

sticks & stones may break my bones but digital logic excites me!



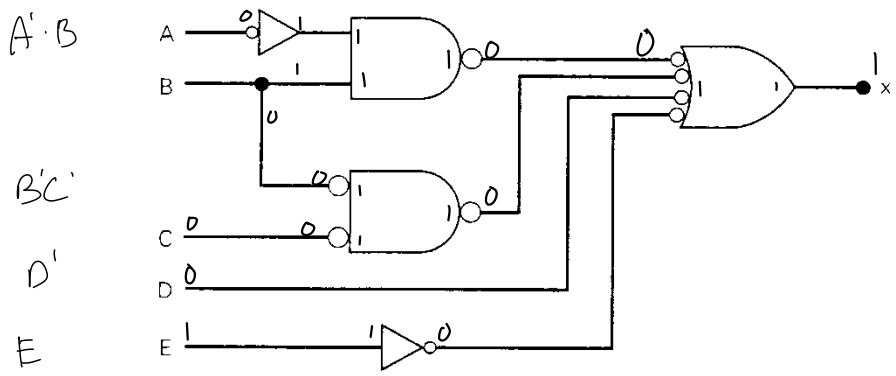
EW	CD 00	01	11	10
AB 00	1	1	1	1
01	0	1	1	1
11	0	0	1	0
10	0	1	1	1

$EW(A,B,C,D) = A'B' + CD + A'D + A'C + BD + B'C$

NS	CD 00	01	11	10
AB 00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	0	0	0

$NS(A,B,C,D) = AC'D' + ABC' + BC'D' + ABD'$

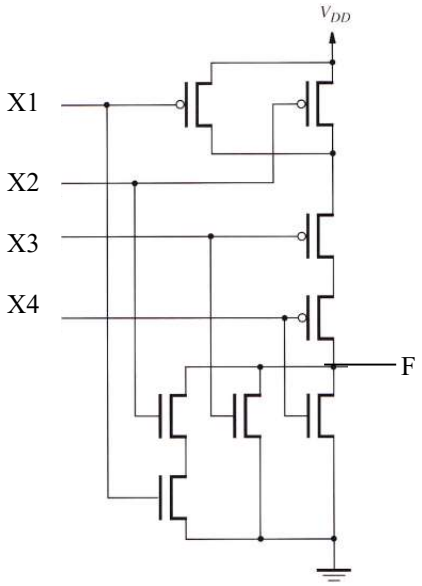
2. From the diagram alone, determine the input conditions that will cause output X to go high. Note that the matched bubbles make the task easy.



$X = A'B + B'C' + D' + E$

3. Determine the truth table for the following CMOS logic circuit.

X_1	X_2	X_3	X_4	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



$F(X_1, X_2, X_3, X_4) = \sum m(0, 4, 8)$

Answers

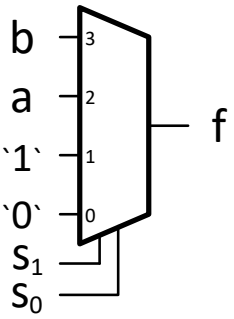
1. N/S = AC'D' + BC'D' + ABD' + ABC'

2. X = A'B + B'C' + D' + E

3. F (X1, X2, X3, X4) = ∑ m (0, 4, 8)

SC1005 Digital Logic
Tutorial 6

Q1 Determine the minimized sum-of-products expression for the 4-1 multiplexer below, and hence sketch the equivalent circuit using just AND and OR gates. Bubble inputs are allowed.



Q2. (a) Draw the circuit represented by the following Verilog module. Label the gates and wires.

```
module whatisit (input a, b, c, d, e,
                 output x, y);
    not  n1  (nb, b);
    not  n2  (ne, e);
    and  a1  (w1, a, b);
    and  a2  (w2, nb, c);
    nor  no1 (w3, d, e);
    nand na1 (w4, w2, w3);
    or   o1  (x, w1, w4);
    and  a3  (y, ne, w1);
endmodule
```

(b) Write down a logic expression for each of the outputs.

Q3. You are required to design a ferry boarding system to direct cars to one of four boarding ramps. The input to the circuit is a 2-bit binary number representing which ramp to use, and a 1-bit signal which is high when all ramps are full. These signals are generated for you (possibly by the human gatekeeper). The circuit has outputs that control four green lights, one above each of the ramps, with only one active at any time. If all ramps are full, no lights should be illuminated indicating that vehicles should not proceed to join any ramp.

(a) Sketch a block diagram for the circuit using a single 2-4 decoder (with enable). A 2-4 decoder takes a 2-bit binary input and produces a one-hot 4-bit output if enable is TRUE.

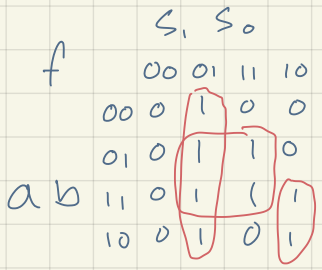
(b) Write a Verilog module to implement the boarding system that instantiates a predefined decoder module with the following declaration:

module dec2to4 (**input** [1:0] i, **input** en, **output** [3:0] d);

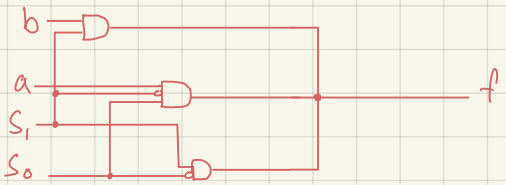
Q4. Rewrite the Verilog module of Q2 using a single assign statement for each output.

1)

a	b	S_1	S_0	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

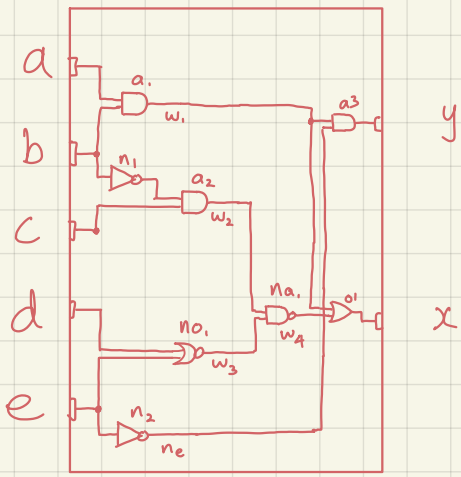


$f = S_0 S_1' + S_0 S b + S_0' S_1 a$



2)

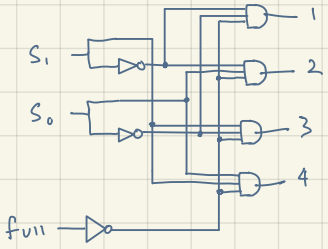
```
module whatisit (input a, b, c, d, e,
                 output x, y);
    not  n1  (nb, b);
    not  n2  (ne, e);
    and  a1  (w1, a, b);
    and  a2  (w2, nb, c);
    nor  no1 (w3, d, e);
    nand na1 (w4, w2, w3);
    or   o1  (x, w1, w4);
    and  a3  (y, ne, w1);
endmodule
```



$x = (a b) + ((c b')(d + e)')'$

$y = a b e'$

3)




```
module answer (input [1:0] i, input en, output [3:0] d, output stop);
    dec2to4 (.a(i), .enable(en), .d(d), .output stop);
endmodule
```

```
module controller (input [1:0] i, en,
                   output [3:0] d)
    assign d[0] = en ? 1'b0 : ~(i[0] || i[1]);
    assign d[1] = en ? 1'b0 : i[0] && ~i[1];
    assign d[2] = en ? 1'b0 : ~i[0] && i[1];
    assign d[3] = en ? 1'b0 : i[0] && i[1];
endmodule
```

SC1005 Digital Logic
Tutorial 7

Q1. A traffic light for a toy car track has a 3-bit one-hot vector input, with MSB corresponding to red and the LSB to green. However, the controller delivered with the track has only a 2-bit output for the lights, encoded as: red = “00”, yellow = “01” and green = “10”. The controller outputs “11” when there are no lights active.

Design a Verilog interface (using assign statements) to go between the controller and the lights.

Q2. A digital thermostat has two 8-bit unsigned binary inputs representing the target temperature and the actual temperature in degrees centigrade (°C). The thermostat has two outputs: one to turn a heater on when the actual temperature is 4°C below the target, and one to turn a cooler on when the actual temperature is 4°C above the target

(a) Design a Verilog module, *thermo*, with two 8-bit inputs, Tset and Tact and two 1-bit outputs Hon and Con. Use a parameter statement to specify the 8-bit width.

(b) The module designed in part (a) is instantiated in another module. Write the Verilog statement to instantiate the thermo module with identifier U1 using the same signal names in the upper module.

(c) Part way through the design process, the design team finds out that the temperature sensor and the target set-point both have 12-bit outputs. Describe a simple change to the Verilog statement to instantiate the thermo module so that this will not affect the operation.

Q3. The following Verilog module has a number of mistakes/errors that would result in it synthesizing incorrectly.

(a) Identify the mistakes/errors and rewrite the module to correct them.

(b) How could you rewrite that module using a single conditional assignment?

```
module thingamajig (input [3:0] a, b, c,
                    input [1:0] sel,
                    output [5:0] result);

      always @ (a,b)
      begin
         case (sel)
            2’b00 : result = a;
            2’b01 : result = b;
            2’b10 : result = c;
         endcase
      end
endmodule;
```

Q4. (a) Write a Verilog module that implements a 3-to-8 decoder using a case statement in an always block.

(b) What modification would you make in order to add an enable (en) to the circuit?

Q1

A traffic light for a toy car track has a 3-bit one-hot vector input, with MSB corresponding to red and the LSB to green. However, the controller delivered with the track has only a 2-bit output for the lights, encoded as: red = "00", yellow = "01" and green = "10". The controller outputs "11" when there are no lights active.

Design a Verilog interface (using assign statements) to go between the controller and the lights.

red	100	00
orang	010	01
green	001	10
n/L	000	11

```
module control (input [1:0] i, output [2:0]out);
    assign i[2] = ~ i[1] & ~i[0];
    assign i[1] = ~ i[1] & i[0];
    assign i[0] =   i[1] & ~i[0];

endmodule
```

A digital thermostat has two 8-bit unsigned binary inputs representing the target temperature and the actual temperature in degrees centigrade (°C). The thermostat has two outputs: one to turn a heater on when the actual temperature is 4°C below the target, and one to turn a cooler on when the actual temperature is 4°C above the target

(a) Design a Verilog module, *thermo*, with two 8-bit inputs, Tset and Tact and two 1-bit outputs Hon and Con. Use a parameter statement to specify the 8-bit width.

```
module thermo #(PARAMETER SIZE = 8)(
                input [size-1:0] Tset, Tact,
                output Hon, Con);

    assign Hon = (Tact + 3'd4) < Tset;
    assign Con = (Tact - 3'd4) > Tset;

end module
```

The module designed in part (a) is instantiated in another module. Write the Verilog statement to instantiate the thermo module with identifier U1 using the same signal names in the upper module.

```
thermo U1 (.Tset(Tset), .Tact(Tact), .Hon(Hon), .Con(Con));
thermo #(.SIZE(12)) U1 (.Tset(Tset), .Tact(Tact), .Hon(Hon), .Con(Con));
```

The following Verilog module has a number of mistakes/errors that would result in it synthesizing incorrectly.

(a) Identify the mistakes/errors and rewrite the module to correct them.

(b) How could you rewrite that module using a single conditional assignment?

```
module thingamajig (input [3:0] a, b, c,
                    input [1:0] sel,
                    output [5:0] result);
      always @ (a,b)
      begin
         case (sel)
            2'b00 : result = a;
            2'b01 : result = b;
            2'b10 : result = c;
         endcase
      end
endmodule;
```

(reg) [5:0] → [1:0] ??

Q4. (a) Write a Verilog module that implements a 3-to-8 decoder using a case statement in an always block.

(b) What modification would you make in order to add an enable (en) to the circuit?

```
module decoder3to8 #PARAMETER SIZE=8, IN=3) (input [IN-1:0] in,
                                             output reg [SIZE-1:0] out);

    always @ *
    begin
      case (in);
        3'b000:   out[0] = 1'b0;
```

Q1

A traffic light for a toy car track has a 3-bit one-hot vector input, with MSB corresponding to red and the LSB to green. However, the controller delivered with the track has only a 2-bit output for the lights, encoded as: red = "00", yellow = "01" and green = "10". The controller outputs "11" when there are no lights active.

Design a Verilog interface (using assign statements) to go between the controller and the lights.

100 red
010 yellow
001 green

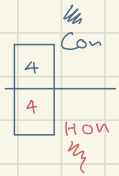
00
01
10
11

```
module trafficlight (input [2:0] d,
                     output [1:0] p);
    assign p[0] = d[1];
    assign p[1] = d[2] | d[0];
endmodule
```

Q2

A digital thermostat has two 8-bit unsigned binary inputs representing the target temperature and the actual temperature in degrees centigrade (°C). The thermostat has two outputs: one to turn a heater on when the actual temperature is 4°C below the target, and one to turn a cooler on when the actual temperature is 4°C above the target

(a) Design a Verilog module, *thermo*, with two 8-bit inputs, Tset and Tact and two 1-bit outputs Hon and Con. Use a parameter statement to specify the 8-bit width.




```
module thermo #(parameter IN=8, OUT=1)(
    input [SIZE-1:0] Tset, Tact,
    output reg [SIZE-1:0] Hon, Con);
    always @*
    begin
    if((Tset + 3'd4) < Iact)
    Con = 1'b1 ;
    else
    Con = 1'b0 ;
    if((Tset - 3'd4) > Tact)
    Ton = 1'b1 ;
    else
    Ton = 1'b0 ;
    end
endmodule
```

b) The module designed in part (a) is instantiated in another module. Write the Verilog statement to instantiate the thermo module with identifier U1 using the same signal names in the upper module.

```
thermo U1 (.Tset(Tset),
           .Tact(Tact),
           .Con(Con),
           .Hon(Hon));
```

Q3

The following Verilog module has a number of mistakes/errors that would result in it synthesizing incorrectly.

(a) Identify the mistakes/errors and rewrite the module to correct them.

(b) How could you rewrite that module using a single conditional assignment?

```
module thingamajig (input [3:0] a, b, c,
                    input [1:0] sel,
                    output [5:0] result);
    always @ (a,b)
    begin
        case (sel)
            2'b00 : result = a;
            2'b01 : result = b;
            2'b10 : result = c;
        endcase
    end
endmodule;
```

reg

← c

3 into 5 bit (-ve will be 00100)

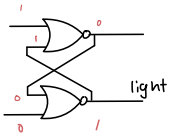
2'b11

SC1005 Digital Logic
Tutorial 8

Q1. Using a behavioural description, write a Verilog module that has three 8-bit inputs, a, b, and c, and outputs the largest (max) and smallest (min) of them. Add a third output (diff) that outputs the difference between the two. You should use if statement(s). You can ignore equal inputs.

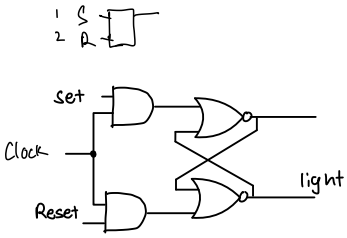
Q2. The *clk* input in the timing diagram is connected to the control/enable/clk input of the following circuits:

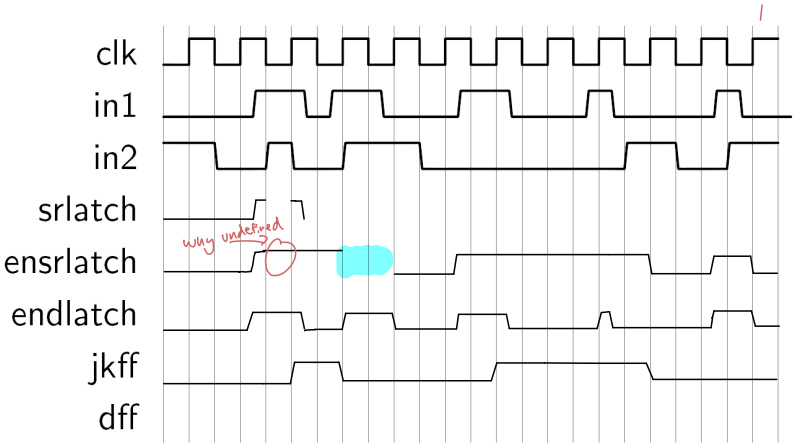
- SR-latch (no enable/control)
- Enabled SR-latch
- Enabled D-type latch
- JK flip-flop
- D flip-flop



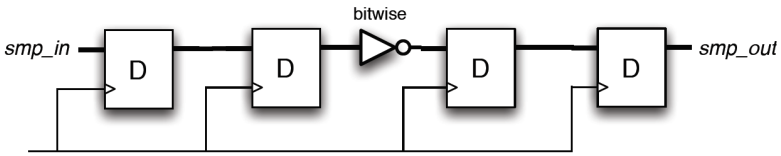
For the D-type latch and D flip-flop, the *in1* input is connected to the D input, and *in2* is left disconnected. For the SR-latches, *in1* is connected to the S input and *in2* is connected to the R input. For the JK flip-flop, *in1* is connected to the J input and *in2* is connected to the K input.

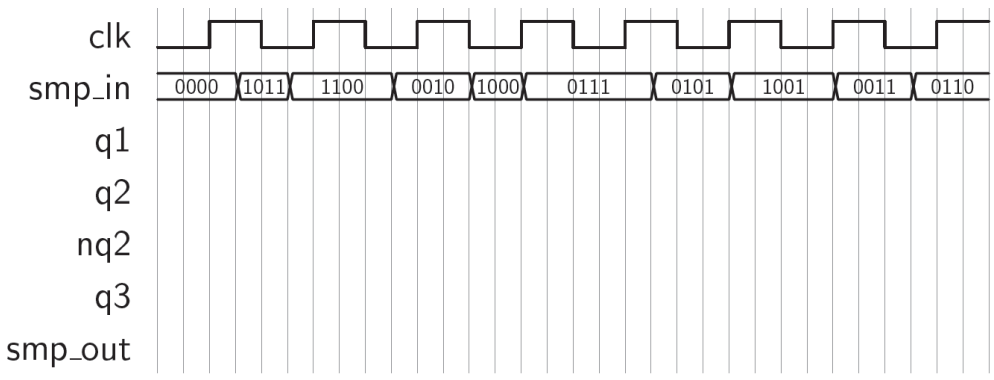
Complete the timing diagram showing the outputs of the four sequential circuits. Indicate any undefined output by shading the relevant area.

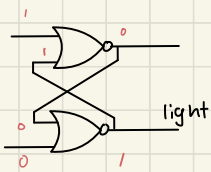


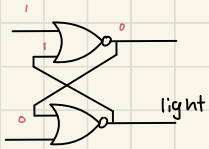


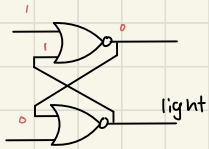
Q3. A bank of 4-bit registers is arranged as per the diagram below, to form a FIFO, with a bitwise inversion in the middle. The input shown in the diagram is applied to the leftmost register. All registers share the same clock. Show a timing diagram for the output at the rightmost register. A bitwise inversion simply inverts each bit of the signal individually. Write down the resulting hexadecimal number sequence at the output.






light


light


light

Q1

Using a behavioural description, write a Verilog module that has three 8-bit inputs, a, b, and c, and outputs the largest (max) and smallest (min) of them. Add a third output (diff) that outputs the difference between the two. You should use if statement(s). You can ignore equal inputs.

```
module compare #(parameter SIZE=8)(
                input [SIZE-1:0] a,b,c,
                Output reg [SIZE-1:0] min,max,dif);

    always @*
    begin
    if(a>b && a>c)
        max=a;
    else if(b>a && b>c)
        max=b;
    else
        max=c;
    if(a<b && a<c)
        min=a;
    else if(b<a && b<c)
        min=b;
    else
        min=c;

    dif = (min+max) / 2'b10
    end
endmodule
```

```
if  a>(b or c)
  wmax = a
else if  b>(a or c)
  wmax = b
else if  c>(a or b)
  wmax = c
```

SC1005 Digital Logic
Tutorial 9

Q1. (a) Explain what is wrong with the following code for a counter, and correct it:

```
module countwrong (input clk, rst, output [5:0] cnt_out);

always@*
begin
cnt_out = cnt_out + 1'b1;
end
endmodule
```

(b) The following combinational module is to be converted into a synchronous module. Add a register after each combinational stage in the original description, by rewriting the module using only a single synchronous always block:

```
module arch1 (input [6:0] a, b, output [13:0] total);

wire [6:0] int1;

assign int1 = a + b;
assign total = int1 * int1;

endmodule
```

Q2. A monitoring circuit has an input, *evnt*, that is high whenever a certain condition is met. It has an internal counter that counts the number of cycles in which *evnt* is high. When this count exceeds a threshold, determined by the 6-bit *thresh* input, it sounds an alarm by asserting the alarm output and stops the counter. The human operator can then check for problems and reset the system by asserting *rst*. Design a Verilog module that implements this circuit. (Hint: alarm should be a combinational circuit determined from the count value and threshold. It can be used to determine whether or not the counter counts in any given cycle).

Q3. (a) Write a Verilog description of a 5-bit binary counter that counts up to 20 and then wraps round to zero.

(b) Modify the counter in (a), adding a new 5-bit input, *countmax*. The counter should now wrap around at the *countmax* value.

(c) Write a Verilog module that implements a 6-bit counter that counts down from an initial value. The initial value should be loaded on reset from a 6-bit input, *start_val*.

A monitoring circuit has an input, *evnt*, that is high whenever a certain condition is met. It has an internal counter that counts the number of cycles in which *evnt* is high. When this count exceeds a threshold, determined by the 6-bit *thresh* input, it sounds an alarm by asserting the alarm output and stops the counter. The human operator can then check for problems and reset the system by asserting *rst*. Design a Verilog module that implements this circuit. (Hint: alarm should be a combinational circuit determined from the count value and threshold. It can be used to determine whether or not the counter counts in any given cycle).

```
module monitor (input evnt, clk, rst,
                input [5:0] tresh,
                output reg alarm)
  reg [5:0] timer;
  always @ (posedge clk)
  begin
    if (rst)
      begin
      timer <= 6'b000000;
      alarm <= 1'b0;
      end
    else
      if (evnt)
        begin
          timer <= timer + 1'b1;
          if (timer > tresh)
            alarm <= 1'b1;
        end
        else
          alarm <= 1'b0;
    end
endmodule
```

Counter must be 7 bits.
Tresh can be 111 111 + 1 = 000 000 ?

(a) Write a Verilog description of a 5-bit binary counter that counts up to 20 and then wraps round to zero.

(b) Modify the counter in (a), adding a new 5-bit input, *countmax*. The counter should now wrap around at the *countmax* value.

(c) Write a Verilog module that implements a 6-bit counter that counts down from an initial value. The initial value should be loaded on reset from a 6-bit input, *start_val*.

```
module counter (input [4:0] countmax, rst, clk,
                output [4:0] time);

  always @ (posedge clk)
    begin
    if (rst)
      time <= 5'b 00000;
    else
    begin
    if (time = countmax)
      time <= 5'b 00000;
    else
      time <= time + 1'b1;
    end
  end
endmodule
```

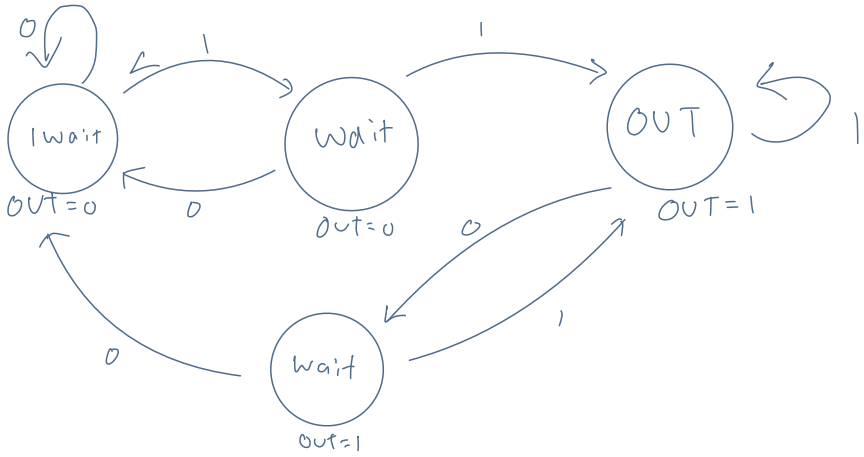
SC1005 Digital Logic
Tutorial 10

Q1. Design a finite state machine that has a single input and single output. It outputs a 1 from the second consecutive high input, and only then outputs a zero after the second consecutive low input. Hence, two consecutive 1 inputs, get a high output, that stays until two consecutive low inputs are received.

(a) Implement the finite state machine in Verilog using a combinational always block for the state transition logic.

(b) Redo the implementation using only assign statements for the state transition logic.

(c) Show how the state machine would respond to the following sequence of inputs:
0,1,0,1,1,0,1,1,0,0,0,1,0,1,1



```
module exam ( input in, clk, reset
              output out);
reg [1:0]st, nst;
assign out = ( st == (2'd2 || 2'd3))
always @ (posedge clk)
begin
   if (reset)
        st <= 2'b00 ;
   else
        st <= nst;
   end
always @ * begin
     nst = st
     case ( st )
        2'b000 : if (in) Nst = st +1 ;
        2'b001 : if (in) NST = st + 1;
                 else Nst = st - 1;
        2'b010 : if (~in) NSt = St + 1;
        2'b011 : if (~in) NSt = St + 1 ;
                 else Nst = st - 1;
```

```
module exam ( input in, clk, rst
              output out);
wire [1:0] st;
reg [1:0] nst;
assign =
assign :

always @ ( posedge clk)
begin
   if(rst)
      st<= 2'b00;
   else
      st<= nst;
end
```