CS207 · Digital Logic

Assignment 2

何泽安 (He Zean)
12011323
Nov. 1, 2021

1 DIGITAL DESIGN THEORY

Q. 1

(a) Let $F_1 = \Sigma m_i$ and $F_2 = \Sigma m_j$, then $E = F_1 + F_2 = \Sigma m_i + \Sigma m_j = \Sigma(m_i + m_j)$, which is the sum of the minterms of F_1 and F_2.

(b) Let $F_1 = \Sigma m_i$ and $F_2 = \Sigma m_j$, then $G = F_1 \cdot F_2 = \Sigma m_i \cdot \Sigma m_j = \Sigma_i \Sigma_j (m_i \cdot m_j)$, where $m_i \cdot m_j = \begin{cases} 0 (i \neq j) \\ 1 (i = j) \end{cases}$, in this case only the minterms where both in (say, common to) F_1 and F_2, a.k.a. appear to have the same i and j can be i in G.

Q. 2

(a) $F(x, y, z) = \Pi(0, 1, 2, 4, 6)$

(b) $F(A, B, C, D) = \Sigma(0, 1, 2, 4, 6, 7, 9, 10, 13, 14)$

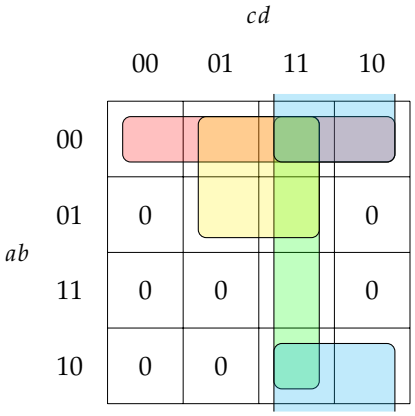
Q. 3

(a)

Converting to min/max-terms

$$
\begin{aligned}
(b' + d)(a' + b' + c)(a' + c) &= b'a'a' + b'a'c + b'b'a' + b'b'c + b'ca' + b'cc + \\
&\quad da'a' + da'c + db'a' + db'c + dca' + dcc \\
&= a'b' + a'b'c + a'b' + b'c + a'b'c + b'c + a'd + a'cd + a'b'd + b'cd + a'cd + cd \\
&= a'b' + a'b'c + b'c + a'd + a'cd + a'b'd + b'cd + cd \\
&= a'b' + b'c + a'd + a'b'd + cd \\
&= a'b' + b'c + a'd + cd
\end{aligned}
$$

Using K-map



$$(b' + d)(a' + b' + c)(a' + c) = a'b' + cd + a'd + b'c$$

(b)

$$\begin{aligned} xy + x'z' + x'yz &= xy(z + z') + x'(y + y')z' + x'yz \\ &= xyz + xyz' + x'yz' + x'y'z' + x'yz \\ &= \Sigma(0, 2, 3, 6, 7) \\ &= \Pi(1, 4, 5) \\ &= (x + y + z')(x' + y + z)(x' + y + z') \end{aligned}$$

Q. 4

(a)

$$\begin{aligned} y'z' + yz' + x'z &= (y' + y)z' + x'z && \text{(LHS)} \\ &= z' + x'z \\ &= x' + z' \\ x' + xz' &= x' + z' && \text{(RHS)} \end{aligned}$$

Thus LHS = RHS, the equation is true.

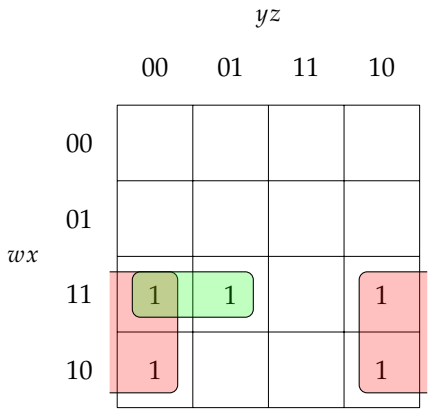
(b)

$$\begin{aligned} x'y + xz + y'z' &= x'y(z + z') + x(y + y')z + (x + x')y'z' && \text{(LHS)} \\ &= x'yz + x'yz' + xyz + xy'z + xy'z' + x'y'z' \\ &= (xy'z + xy'z') + (x'yz' + x'y'z') + (x'yz + xyz) \\ &= xy' + x'z' + yz && \text{(RHS)} \end{aligned}$$

Thus LHS = RHS, the equation is true.

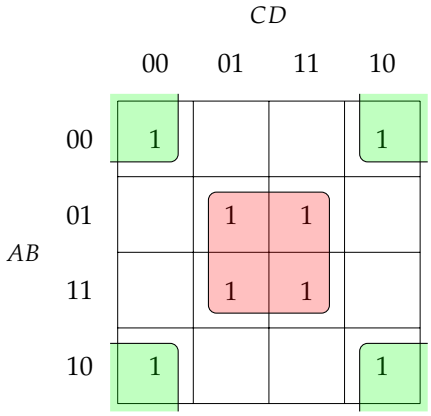
Q. 5

(a)



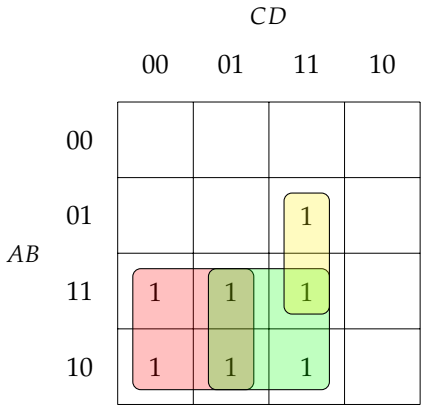
$$F(w,x,y,z) = (wy'z' + wyz') + wxy' = wz' + wxy'$$

(b)



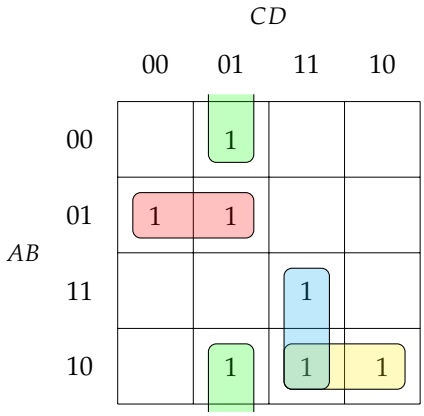
$$F(A,B,C,D) = BD + B'D'$$

(c)



$$AB'CD + AD + AC' + BCD = AC' + AD + BCD$$

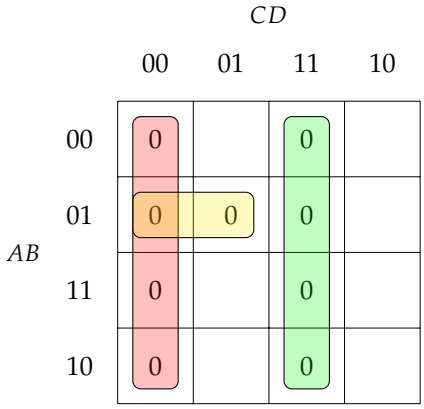
(d)



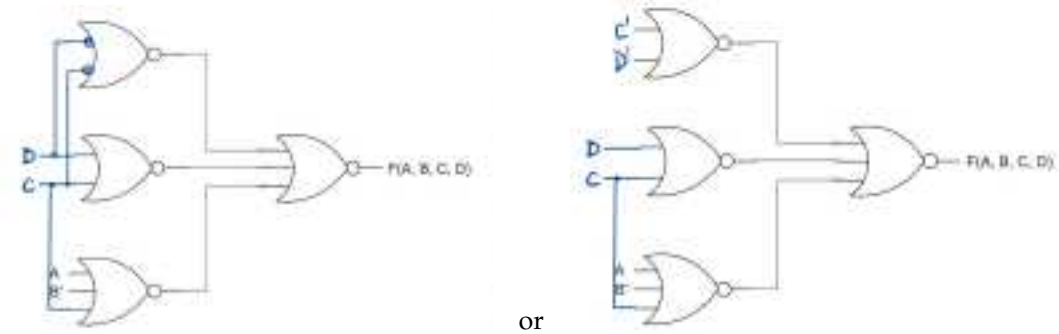
$$A'B'C'D + ABCD + A'BC' + AB'C + AB'D = A'BC' + B'C'D + ACD + AB'C$$

Q. 6

(a)



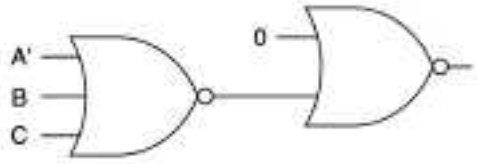
$$F(A, B, C, D) = (C + D)(C' + D')(A + B' + C)$$



(b)

AB \ CD	00	01	11	10
00				
01				
11				
10	0	0		

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



Q. 7

(a)

AB \ CD	00	01	11	10
00	d	0	0	1
01	1	0	1	d
11	1	0	0	0
10	d	0	0	1

$$F(A,B,C,D) = C'D' + B'D' + A'BC$$

$$= \Sigma(0,2,4,6,7,8,10,12)$$

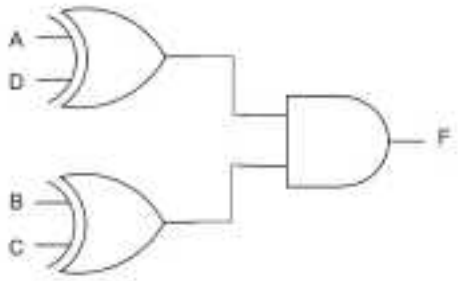
(b)

AB \ CD	00	01	11	10
00	d	d	0	1
01	1	d	0	0
11	1	0	0	1
10	d	0	0	1

$$F(A,B,C,D) = B'D' + C'D' + AD'$$
$$= \Sigma(0,2,4,8,10,12,14)$$

Q. 8

$$F = A'BC'D + AB'CD' + ABC'D' + A'B'CD$$
$$= (A'D + AD')BC' + (AD' + A'D)B'C$$
$$= (A'D + AD')(BC' + B'C)$$
$$= (A \oplus D)(B \oplus C)$$



2 DIGITAL DESIGN LAB

2.1 Task 1

```
`timescale 1ns / 1ps

module seg_tube (
    input [3:0] id,
    output reg [7:0] seg_out
);
    always @* begin
        case(id)
            0: seg_out=8'b1100_0000;          // 0
            1: seg_out=8'b1111_1001;          // 1
            2: seg_out=8'b1010_0100;          // 2
            3: seg_out=8'b1011_0000;          // 3
            default: seg_out = 8'b1111_1111;  // off
        endcase
    end
endmodule

module enc (
    input [3:0] bell,
    input hi_fst,
    output [7:0] seg_out, seg_en
);
    assign seg_en = 8'b1111_1110;
    reg [3:0] dispid;
    always @* begin
        case(hi_fst)
            1: begin
                casex(bell)
                    'b0000: dispid <= 'b100;  // invalid flag
                    'b0001: dispid <= 0;
                    'b001x: dispid <= 1;
                    'b01xx: dispid <= 2;
                    'b1xxx: dispid <= 3;
                endcase
            end
            0: begin
                casex(bell)
                    'b0000: dispid <= 'b100;
                    'b1000: dispid <= 3;
                    'bx100: dispid <= 2;
                    'bxx10: dispid <= 1;
                    'bxxx1: dispid <= 0;
                endcase
            end
        endcase
    end
```

```
    seg_tube st(.id(dispid), .seg_out(seg_out));
endmodule
```

The design file contains two module, the assistant module that helps converting an input id into the seg-tube-code, while the main module *mux* takes a priority switch and the input switches, then select the one having the highest priority and convert it into seg-tube-code. To make it elegant (that can use the assistant module *seg_tube*), though the code of wards 0 - 3 can be represented in a 2-bit binary number, I use a 3-bit register that can represent the no-switch-state. (say, *dispid* actually acts as

{disable_output, id[1], id[0]}

, this is only for convenience.)

```
# seg_tube
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[7]}]
set_property PACKAGE_PIN A18 [get_ports {seg_en[7]}]
set_property PACKAGE_PIN A20 [get_ports {seg_en[6]}]
set_property PACKAGE_PIN B20 [get_ports {seg_en[5]}]
set_property PACKAGE_PIN E18 [get_ports {seg_en[4]}]
set_property PACKAGE_PIN F18 [get_ports {seg_en[3]}]
set_property PACKAGE_PIN D19 [get_ports {seg_en[2]}]
set_property PACKAGE_PIN E19 [get_ports {seg_en[1]}]
set_property PACKAGE_PIN C19 [get_ports {seg_en[0]}]
set_property PACKAGE_PIN E13 [get_ports {seg_out[7]}]
set_property PACKAGE_PIN C15 [get_ports {seg_out[6]}]
set_property PACKAGE_PIN C14 [get_ports {seg_out[5]}]
set_property PACKAGE_PIN E17 [get_ports {seg_out[4]}]
set_property PACKAGE_PIN F16 [get_ports {seg_out[3]}]
set_property PACKAGE_PIN F14 [get_ports {seg_out[2]}]
set_property PACKAGE_PIN F13 [get_ports {seg_out[1]}]
set_property PACKAGE_PIN F15 [get_ports {seg_out[0]}]

# priority/bell switchs
set_property IOSTANDARD LVCMOS33 [get_ports {bell[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {bell[1]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {bell[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {bell[3]}]
set_property PACKAGE_PIN T5 [get_ports {bell[3]}]
set_property PACKAGE_PIN T4 [get_ports {bell[2]}]
set_property PACKAGE_PIN R4 [get_ports {bell[1]}]
set_property PACKAGE_PIN W4 [get_ports {bell[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports hi_fst]
set_property PACKAGE_PIN U5 [get_ports hi_fst]
set_property DRIVE 12 [get_ports {seg_en[7]}]
set_property DRIVE 12 [get_ports {seg_en[6]}]
set_property DRIVE 12 [get_ports {seg_en[5]}]
set_property DRIVE 12 [get_ports {seg_en[4]}]
set_property DRIVE 12 [get_ports {seg_en[3]}]
set_property DRIVE 12 [get_ports {seg_en[2]}]
set_property DRIVE 12 [get_ports {seg_en[1]}]
set_property DRIVE 12 [get_ports {seg_en[0]}]
```

Here we set the right-most 4 switches as the wards' bell input, and the 5^{th} switch as the priority switch.

(a) When no ward press the bell, display nothing



(b) Showing in both high/low-first mode, when only one ward calls, display the corresponding room number

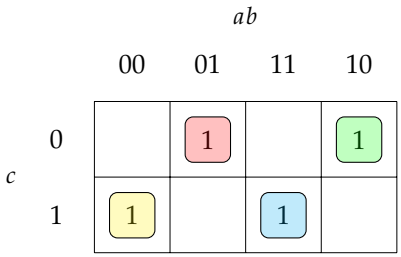

(c) Showing in high-first mode, bell 0&2 on → display 2; bell 0&1&2&3 on → display 3


(d) Showing in low-first mode, bell 1&3 on → display 1; bell 0&1&2&3 on → display 0

Figure 1: Verifying Task 1

2.2 Task 2

a	b	c	$F(a,b,c) = a \oplus b \oplus c$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$$F(a,b,c) = a'b'c + a'bc' + abc + ab'c' \quad \text{(SOP)}$$
$$= (a+b+c)(a+b'+c')(a'+b'+c)(a'+b+c') \quad \text{(POS)}$$

Design file

```
`timescale 1ns/1ps

module task2 (
    input a, b, c,
    output sop, pos
);
    assign sop = ~a & ~b & c | ~a & b & ~c | a & b & c | a & ~b & ~c;
    assign pos = (a | b | c) & (a | ~b | ~c) & (~a | ~b | c) & (~a | b | ~c);
endmodule
```

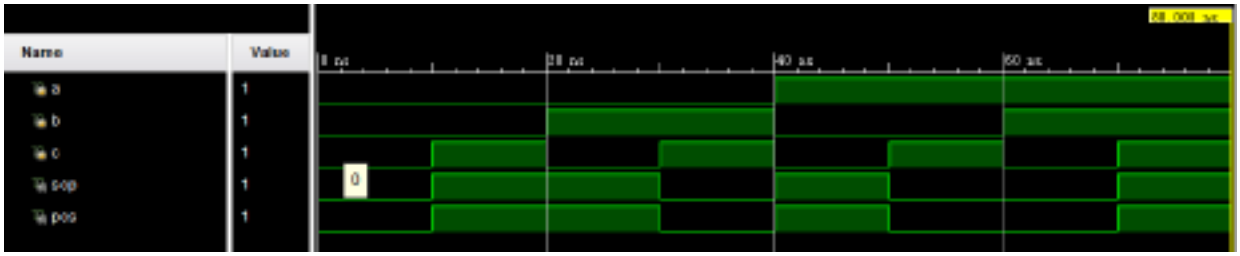
Simulation file

```
`timescale 1ns/1ps

module tsk2tb ();
    reg a, b, c;
    wire sop, pos;

    task2 t(a, b, c, sop, pos);
```

```
8
9       initial begin
10          {a, b, c} = 3'b000;
11          while ({a, b, c} < 3'b111)
12              #10 {a, b, c} = {a, b, c} + 1;
13          #10 $finish();
14      end
15  endmodule
```



From the waveform above we see:

① The truth-values of SOP and POS design are always the same.

② When the number of *true* in the three variables a, b, c is odd, the truth value of $a \oplus b \oplus c$ is *true*, otherwise is *false*. The result keeps the same as the truth table.

2.3 Problems & Solutions

(1) At first I directly used part of the constraint file (about the seg-tube) provided on the lab scission, then I found there always light up two numbers together. After checking the design file, constraint file and the minisys user manual carefully, I found there's a mis-constrained port and fixed it.

(2) Designing the seg-tubes that are low-level effective is no that intuitive and need much care. An intuitive way is designing them in high-level way that use 1 to active one, and add a *bitwise not* to transverse them.

(3) Maybe the next assignment could be harder and contains more questions :)