Homework 2

PB20111686 Ruixuan Huang

Problem. 1

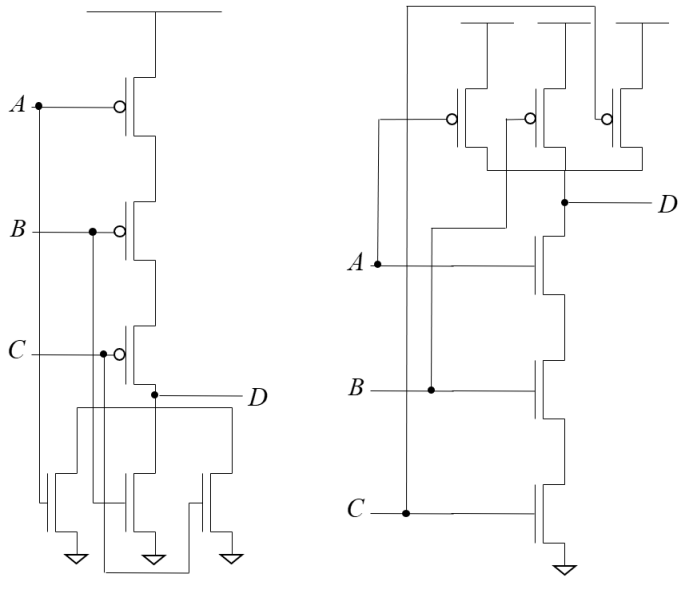
0 00000001 00000000000000000000000 1.0×2^{-126}

Problem. 2

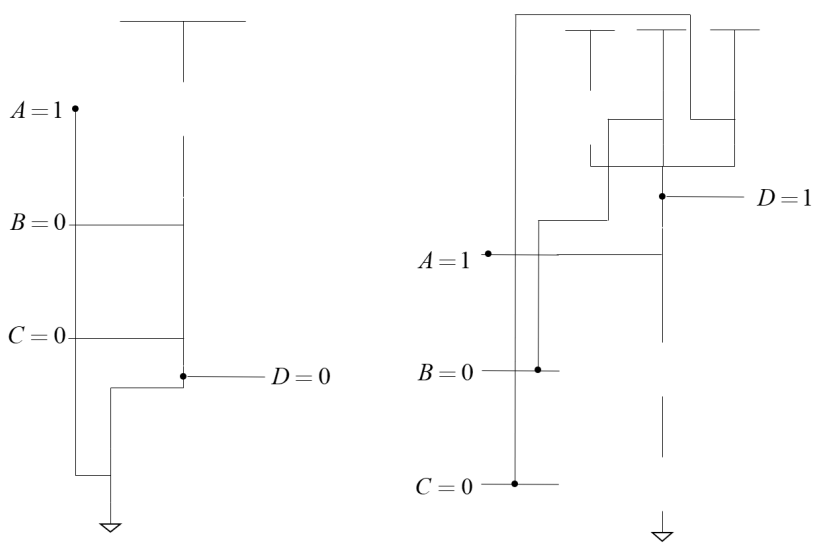
01111111111111111111111111111111 $2^{31} - 1$

Problem. 3

a.



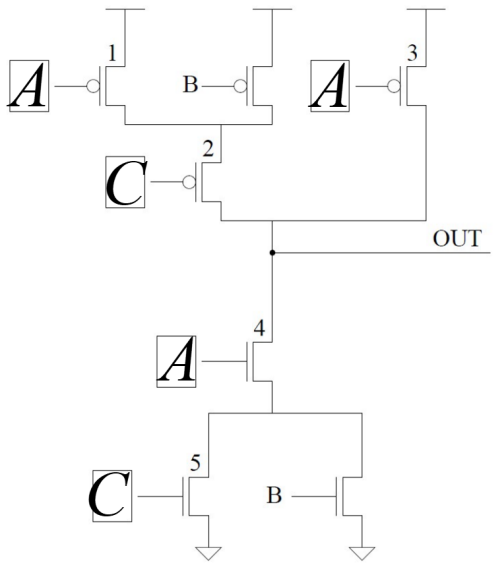
b.



c.

There're 2 possible patterns.

(1)



A	B	C	OUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

(2)

A	B	C	OUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Problem. 4

	a	b	c	d	e	f
X	X	1	0	1	0	0

Problem. 5

The output D in circuit 1 is only related to the input A, B, C at present. However, the output D in circuit 2 is related to not only the input A, B at present but also the status of D before the inputs changes.

Problem. 6

(1)

S1	S0	A	B	C	D	OUT
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	0
0	0	0	1	0	0	0
0	0	0	1	0	1	0
0	0	0	1	1	0	0
0	0	0	1	1	1	0
0	0	1	0	0	0	1
0	0	1	0	0	1	1
0	0	1	0	1	0	1
0	0	1	0	1	1	1
0	0	1	1	0	0	1
0	0	1	1	0	1	1
0	0	1	1	1	0	1
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	0	0	1	0
0	1	0	0	1	0	0
0	1	0	0	1	1	0
0	1	0	1	0	0	1
0	1	0	1	0	1	1
0	1	0	1	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	0	0
0	1	1	0	0	1	0
0	1	1	0	1	0	0
0	1	1	0	1	1	0
0	1	1	1	0	0	1
0	1	1	1	0	1	1
0	1	1	1	1	0	1
0	1	1	1	1	1	1

S1	S0	A	B	C	D	OUT
1	0	0	0	0	0	0
1	0	0	0	0	1	0
1	0	0	0	1	0	1
1	0	0	0	1	1	1
1	0	0	1	0	0	0
1	0	0	1	0	1	0
1	0	0	1	1	0	1
1	0	0	1	1	1	1
1	0	1	0	0	0	0
1	0	1	0	0	1	0
1	0	1	0	1	0	1
1	0	1	0	1	1	1
1	0	1	1	0	0	0
1	0	1	1	0	1	0
1	0	1	1	1	0	1
1	0	1	1	1	1	1
1	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	0
1	1	0	0	1	1	1
1	1	0	1	0	0	0
1	1	0	1	0	1	1
1	1	0	1	1	0	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	0	0	1	1
1	1	1	0	1	0	0
1	1	1	0	1	1	1
1	1	1	1	0	0	0
1	1	1	1	0	1	1
1	1	1	1	1	0	0
1	1	1	1	1	1	1

(2) The Verilog code of my implementation is as follows.

```
module MUX4_1(input a, b, c, d, sel0, sel1, output out);
    wire o1, o2;
    MUX2_1 S1(o1, a, b, sel1);
    MUX2_1 S2(o2, c, d, sel1);
    MUX2_1 S3(out, o1, o2, sel0);
endmodule
```

(3) The Verilog code of my implementation is as follows.

```
module xor(input A, B, output out);
    wire o1;
    MUX2_1 S1(o1, B, 0, A);
    MUX2_1 S2(out, A, o1, B);
endmodule
```

Problem. 7

(a) 3.

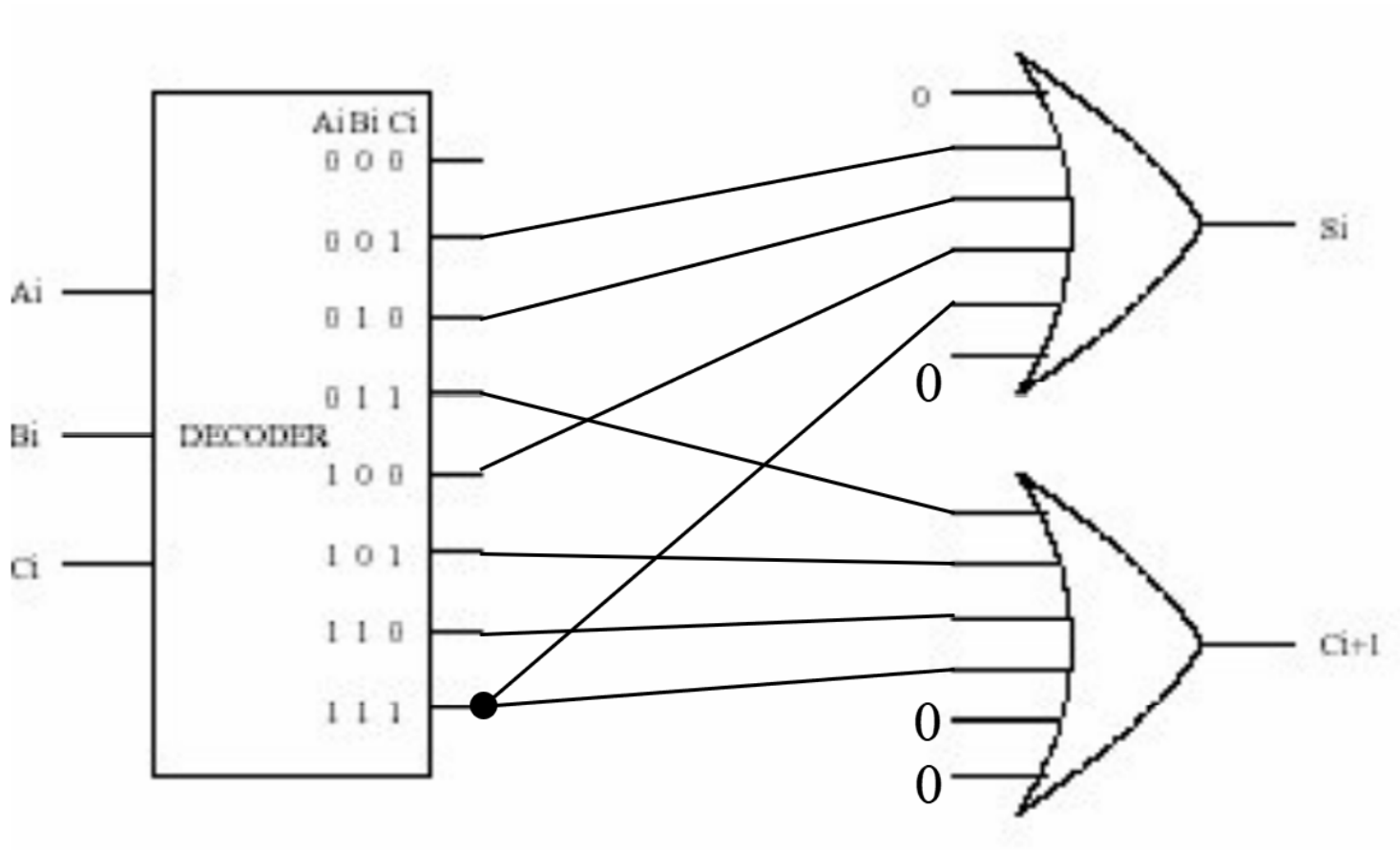
(b) 12.

(c) The Verilog code of my implementation is as follows.

```
module figure2(input z, output a, b, c, d, e);
    wire w1, w2, w3, w4;
    and
    (w1, a, b),
    (w2, c, d),
    (w3, e, e),
    (w4, w1, w2),
    (z, w3, w4);
endmodule
```

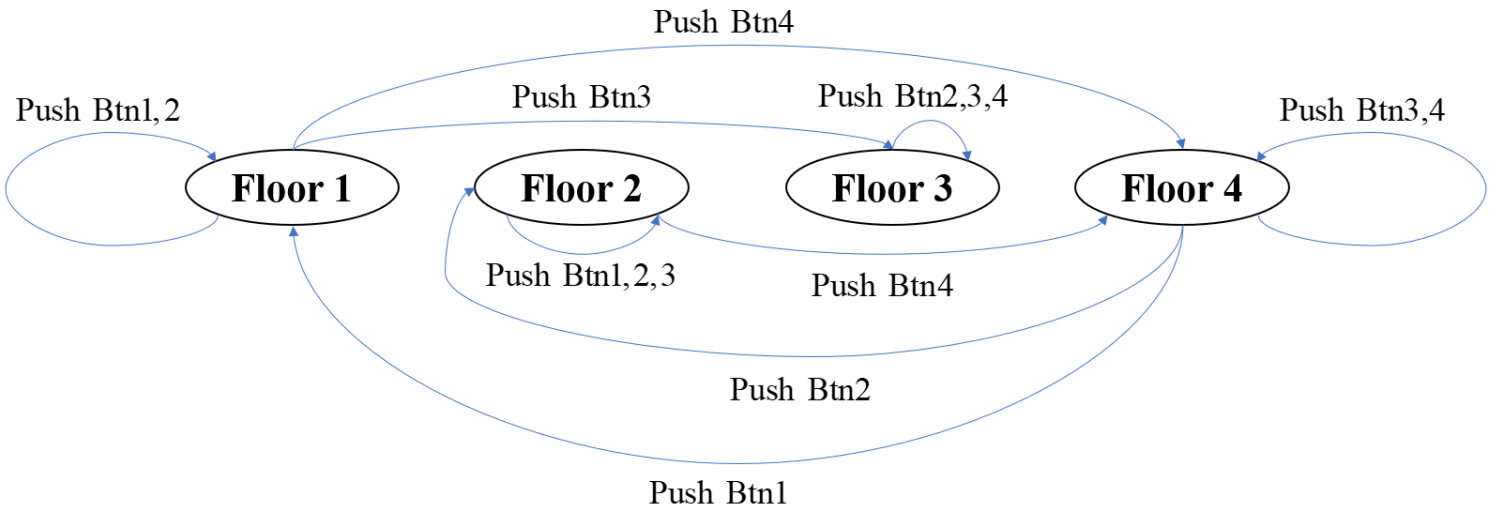
Now the propagation delay is decreased from 4 to 3.

Problem. 8



Problem. 9

(a)



(b) The truth table I built is as follows. A[1:0] is the current status of the elevator. For example, A[1:0] = 2'b10 represents the elevator is located at the third floor. B[1:0] is the button I push. C[1:0] is the status of the elevator after I push the button.

A[1]	A[0]	B[1]	B[0]	C[1]	C[0]
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	1	1

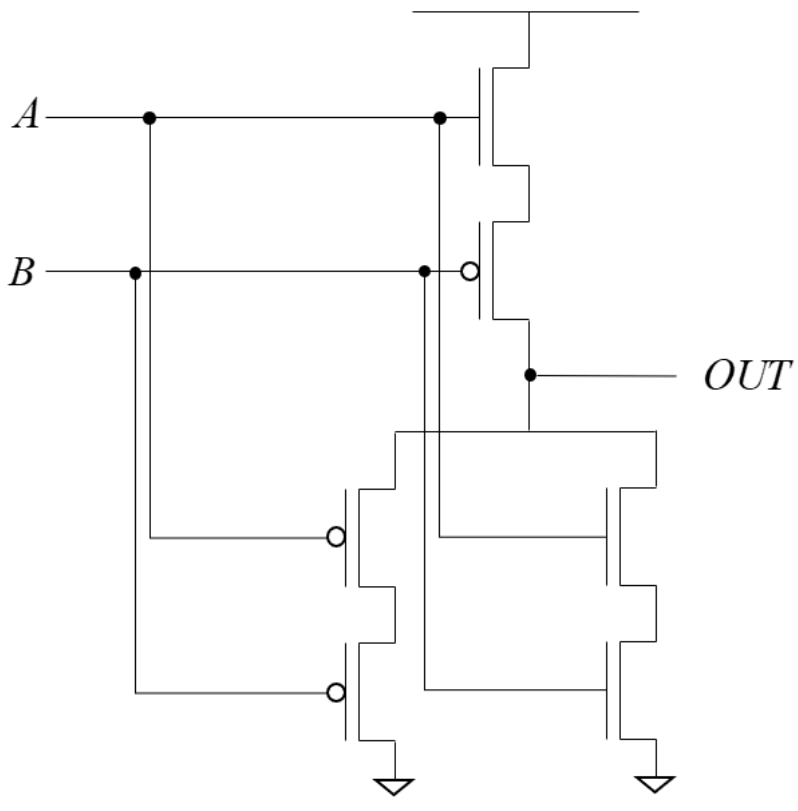
Problem. 10

According to the circuit, the status will be changed as the following pattern.

000000(t_0) - 100000($c_1 end$) - 111000($c_2 end$) - 111110 - 011111 - 000111 - 000001 - 100000($c_7 end$).....

So it takes 6 cycles for a specific state to show up again. After 50 cycles, the state will be "111000".

Problem. 11

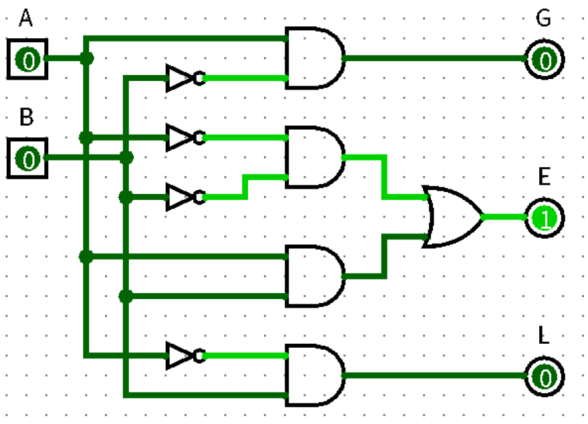


Problem. 12

a.

A	B	G	E	L
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

b. My implementation circuit is as follows.



c.

The logic expressions are as follows.

$$OUT = G[3] + E[3]G[2] + E[3]E[2]G[1] + E[3]E[2]E[1]G[0]$$

The Verilog code of my implementation is as follows.

```
module prog12c(input G[3:0], E[3:1], output OUT);
    wire w1, w2, w3;
    and
    (w1, E[3], G[2]),
    (w2, E[3], E[2], G[1]),
    (w2, E[3], E[2], E[1], G[0]);
    or(OUT, G[3], w1, w2, w3);
endmodule
```

Problem. 13

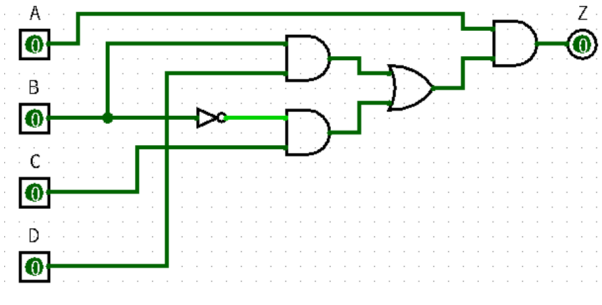
The logic expressions are as follows.

$$Z = A(BD + \bar{B}C)$$

The truth table is as follows.

A	B	C	D	Z
0	x	x	x	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

The gate-level diagram is as follows.



Problem. 14

Let $A\&B$ be $\overline{A\&B}$(**NAND**).

$$\bar{A} = A \not\& A$$
$$A|B = \bar{A} \not\& \bar{B}$$
$$A \& B = \overline{\bar{A}|\bar{B}} = (\bar{A}|\bar{B}) \not\& (\bar{A}|\bar{B}) = (A \not\& B) \not\& (A \not\& B)$$

We know that **AND** & **OR** & **NOT** is logically complete, so the **NAND** is logically complete.