Question 1. An image processing system gives us pixels with four levels of grey: $ab = 00, 01, 10, 11$ (respectively black, dark, light and white). We want to build a machine that gets those ab values and sets its output Z to 1 whenever it detects the end of a full ascending or descending sequence: $00-01-10-11$ or $11-10-01-00$. We can simplify this problem by assuming that two successive pixels will never have the same value.

States (We have to add the initial state - state 1 -)

Build the Huffman table of this machine.

	00	01	11	10	Z
1	2	1	6	1	0
2	2	3	1	1	0
3	1	3	1	4	0
4	1	1	5	4	0
5	1	1	5	7	1
6	1	1	6	7	0
7	1	8	1	7	0
8	9	8	1	1	0
9	9	3	1	1	1

We chose stable states for ab = 01 & ab = 10, because ab = 11 or ab = 00 are states in which we could eventually have Z = 1, whereas we want an initial neutral state.

→ 00 - 01 - 10 - 11 → 10 - 01 - 00

→ We go back to the beginning!

→ If we re-enter the same ab values that we put to arrive in this state, we rest in the same state forever (by def. of stable state)

11 - 10 - 01 - 00 → 01 - 10 - 11

From our initial stable state, we could start from the ascending: state 2 or the descending: state 6

Question 2. Two evil weasels are attacking Metropolis. Superman is the only one who can save the town. However, the dreadful Mustelida are tough and Superman decides to ask Dr Hamilton for help. The professor thus gives him a Huffman table explaining the sequence of power required to triumph. Deceitfully, he also includes a sequence that would turn the weasels into immortal fire breathing monsters. The following instructions are added:

> a: laser vision; b: ice breath; Z_1: weasels defeated; Z_2: invulnerable weasels.

	ab					
Y_1Y_2	00	01	11	10	Z_1	Z_2
1	1	2	-	5	0	0
2	1	**2**	-	3	0	0
3	1	4	-	**3**	0	0
4	**4**	**4**	**4**	**4**	1	0
5	1	6	-	**5**	0	0
6	1	**6**	-	7	0	0
7	**7**	**7**	**7**	**7**	0	1

What does Superman need to do in order to save Metropolis?

$1 \xrightarrow{01} 2 \xrightarrow{10} 3 \xrightarrow{01} 4$: ice breath, laser, ice breath

Question 3. According to a legend created for this session, envious of the Konami code[1] success, Nintendo would have implemented a similar feature in his NES games. The "Nintendo code" is simpler though: in order to activate a hidden bonus, the player simply needs to press (A), then (B) while maintaining (A) and finaly to press SELECT while still holding the first two buttons down. If one of the buttons is released, or if they are pressed in the wrong order, the sequence is canceled.

Build a digital circuit implementing this feature.

a ← (A)

b ← (B)

c ← SELECT

States:

a	b	c	
0	0	0	"Initial neutral state"
1	0	0	
1	1	0	
1	1	1	

encoded on 4 lines ⇒ 2 Y_i

Y_1Y_2	000	001	011	010	100	101	111	110	Z
00	00	00	00	00	01	00	00	00	0
01	00	00	00	00	01	00	00	11	0
11	00	00	00	00	00	00	10	11	0
10	00	00	00	00	00	00	10	00	1

y_1y_2

Huffman Table

y_1y_2 \ abc	000	001	011	010	100	101	111	110
00								
01								1
11							1	1
10							1	

$$Y_1 = abcy_1 + ab\bar{c}y_2$$

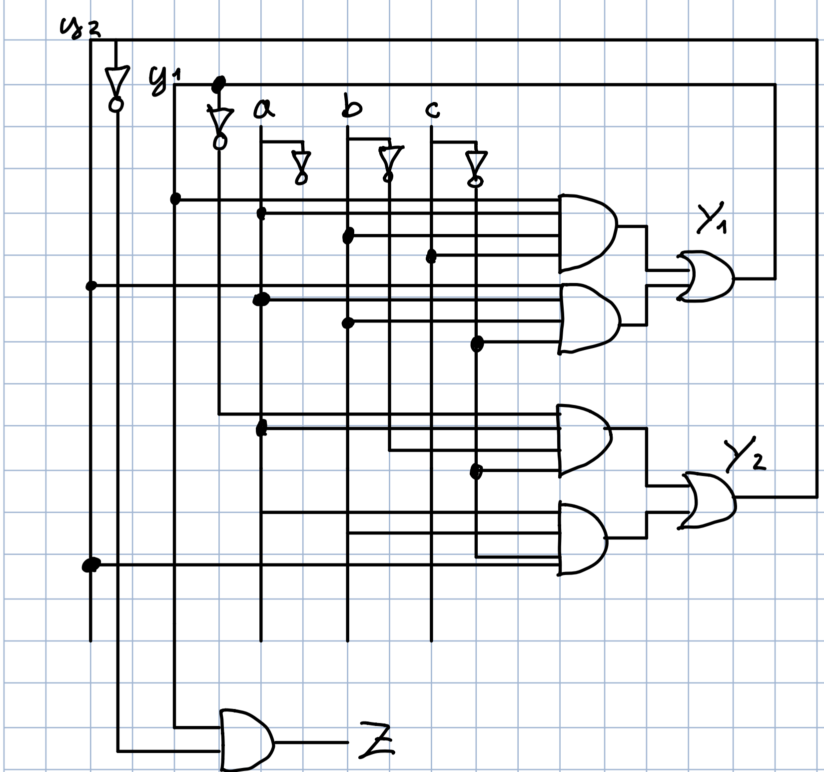
y_1y_2 \ abc	000	001	011	010	100	101	111	110
00					1			
01					1			1
11								1
10								

$$Y_2 = \overline{abcy_1} + ab\bar{c}y_2$$

y_1y_2 \ abc	000	001	011	010	100	101	111	110
00								
01								
11								
10	–	–	–	–	–	–	1	–

For Z first look at stable states where Z = 1 (10 111)

$$Z = y_1\bar{y}_2$$



Solving race conditions :

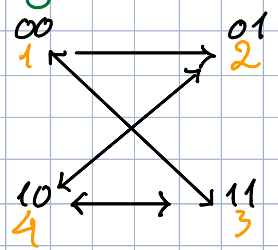
Method 1 —if did not fully work→ Method 2 —if both did not work→ Method 3

Method 1 - State encoding

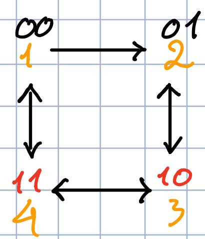
Try to change state encoding so that there are no race cond.

Systematic way : Build state graph without transition conditions and self transitions

	Y_1	Y_2	00	01	11	10
1	0	0	00	01	11	01
2	0	1	01	01	10	01
3	1	1	00	11	11	10
4	1	0	01	11	10	10



⚠ Diagonals will be races cond.



Change binary coding BUT NOT the pos. of 1, 2, ..., 4

The binary repr. of states will not change in the K-map.

Method 2 - Modifying transitions

Modify sub-transitions so that we do not have suddenly a hamming dist. of 2.

Example : sudden 00 —▷ 11 —▷ 11

We have to find :
- Existing transitions
- Don't cares

NOT sudden
00 —▷ 01 —▷ 11 —▷ 11
00 —▷ -- —▷ 11 —▷ 11

Add it to the transition with race conditions

Method 3 - Adding extra state variable

Add a new Y_{n+1}
↳ 2^n more states

Use them for method 2.

⚠ State optimization is ≠ in Moore & Meally machines

Moore state optimization

We cannot merge states having the same Z

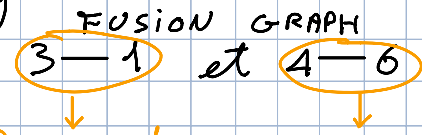
So we build the "equivalences condition table" putting × if Z is different.

	ab 00	01	11	10	Z
1	**1**	2	–	3	0
2	1	**2**	4	–	1
~~3~~	1	–	4	**3**	0
4	–	5	**4**	–	1
5	6	**5**	4	–	0
~~6~~	**6**	5	–	3	1

2	×				
3	OK	×			
4	×	2-5	×		
5	1-6 2-5	×	1-6	×	
6	×	1-6 2-5	×	OK	×
	1	2	3	4	5

2	×				
3	OK	×			
4	×	~~2-5~~ ×	×		
5	~~1-6 2-5~~ ×	×	~~1-6~~ ×	×	
6	×	~~1-6 2-5~~ ×	×	OK	×
	1	2	3	4	5

FUSION GRAPH



Becomes 1

Becomes a new 3 since we don't have one.

⚠ After fused table, CHECK for RACES

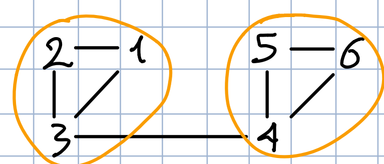
Meally state optimization

We can merge states having different Z EXCEPT those differing by two different STABLE cells!

From Moore → All × become n-n or OK except

2	OK				
3	OK	OK			
4	2-5	2-5	OK		
5	1-6 2-5	×	1-6	OK	
6	×	1-6 2-5	1-6	OK	OK
	1	2	3	4	5

2	OK				
3	OK	OK			
4	~~2-5~~ ×	~~2-5~~ ×	OK		
5	~~1-6 2-5~~ ×	×	~~1-6~~ ×	OK	
6	×	~~1-6 2-5~~ ×	~~1-6~~ ×	OK	OK
	1	2	3	4	5



becomes 1

becomes 2 (because tied to 1)

Fusion table

	00	01	11	10
1	1/0	1/1	2	1/0
2	2/1	2/0	2/1	1

2 became 1 but it's output is kept

We use old don't care cells to tie the 2 new states