## 1. FSM I

**- Analyze the FSM shown as follows. The input of this FSM is X, clock signal is CLK, and output is Q. Write the state transition table, output table. Sketch the state transition diagram. (10')**

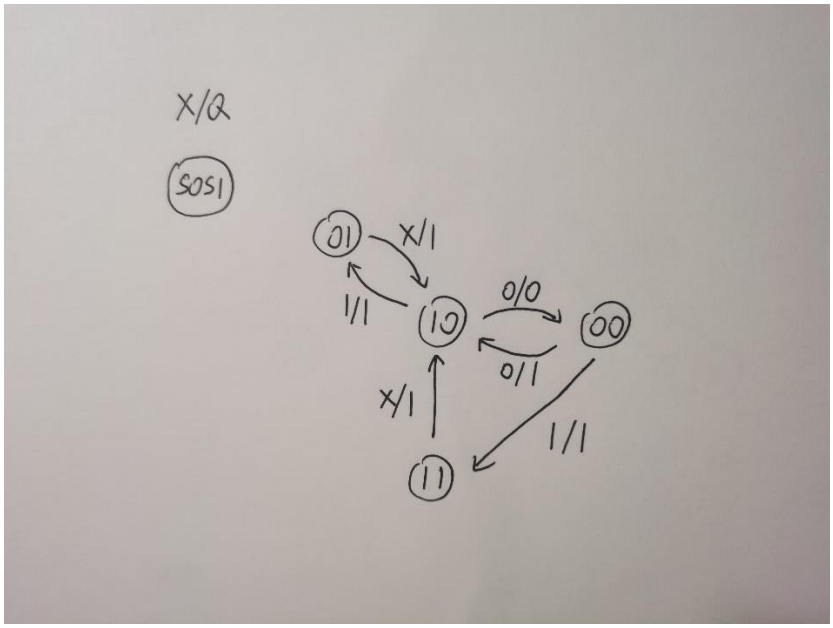
State transition table(S0 for the left, S1 for the right):

| Current State | | Input | Next State | |
|---|---|---|---|---|
| S0 | S1 | X | S0* | S1* |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |

Output table(S0 for the left, S1 for the right):

| Current State | | Input | Output |
|---|---|---|---|
| S0 | S1 | X | Q |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |

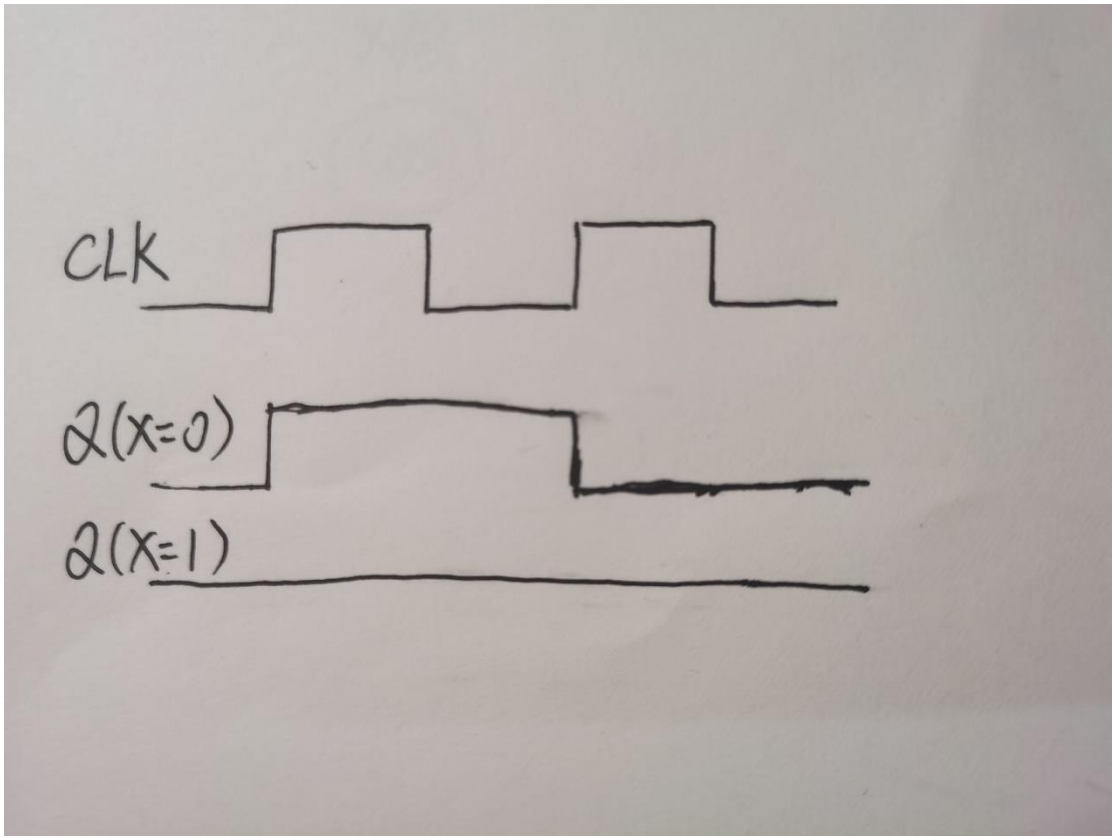
State transition diagram:



**- Suppose the clock frequency is 50 Hz. For this FSM, if the input X is HIGH, what will the output be? What if the input X is LOW? Describe both situation in words. You can show hand-drawn waveform, but you can't use simulation results. (10')**

If X is HIGH, the output will be always HIGH.

If X is LOW, the output will change periodically between HIGH and LOW at the frequency of 50Hz.




## 2. FSM II

**- The state transition diagram of an FSM is shown below. The input is X. State encoding and output encoding is shown in the table below. Draw the state transition table of this FSM (20')**

State transition table:

| Current State | Current State encoding | | Input | Next State | Next State encoding | |
|---|---|---|---|---|---|---|
| S | S[0] | S[1] | X | S* | S*[0] | S*[1] |
| S0 | 0 | 0 | 0 | S0 | 0 | 0 |
| S0 | 0 | 0 | 1 | S1 | 0 | 1 |
| S1 | 0 | 1 | 0 | S2 | 1 | 0 |
| S1 | 0 | 1 | 1 | S1 | 0 | 1 |
| S2 | 1 | 0 | 0 | S2 | 1 | 0 |
| S2 | 1 | 0 | 1 | S3 | 1 | 1 |
| S3 | 1 | 1 | 0 | S0 | 0 | 0 |
| S3 | 1 | 1 | 1 | S3 | 1 | 1 |

**– According to the transition table you’ve drawn, simplify the state logic and output logic, implement the circuit in Multisim. The clock frequency should be 1000 Hz and the initial state should be S0. Test its performance. (20’)**

State logic:

$S^*[0] = \overline{S[0]}\ S[1]\ \overline{X} + S[0]\ \overline{S[1]}\ \overline{X} + S[0]\ \overline{S[1]}\ X + S[0]\ S[1]\ X$

$= \overline{S[0]}\ S[1]\ \overline{X} + S[0]\ S[1]\ X + S[0]\ \overline{S[1]}$

$S^*[1] = \overline{S[0]}\ \overline{S[1]}\ X + \overline{S[0]}\ S[1]\ X + S[0]\ \overline{S[1]}\ X + S[0]\ S[1]\ X$

$= X$

Output logic:

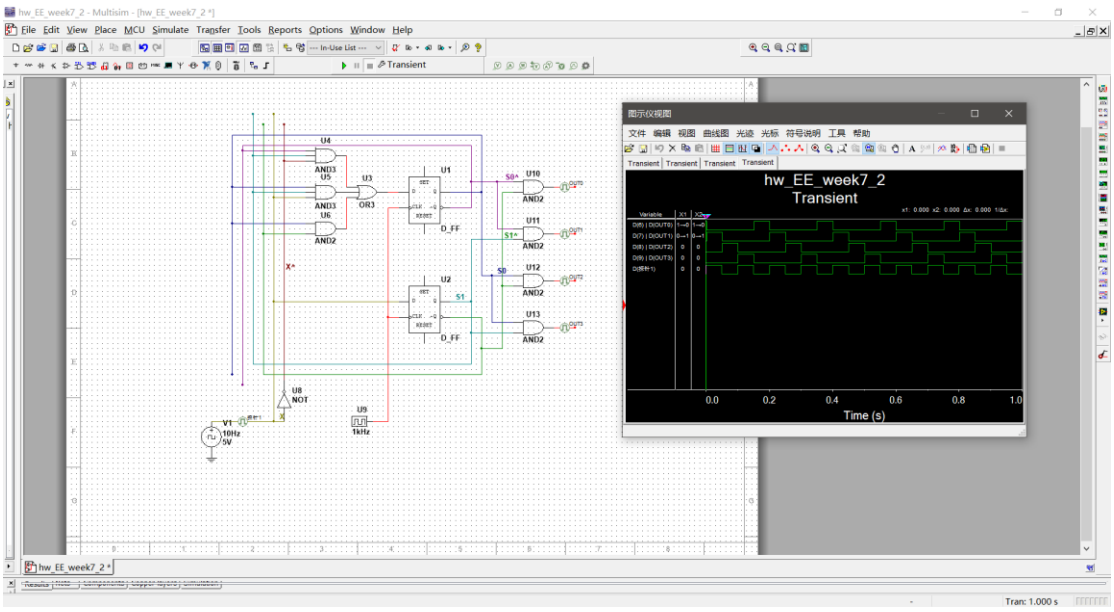
$Output[0] = \overline{S^*[0]}\ \overline{S^*[1]}$

$Output[1] = \overline{S^*[0]}\ S^*[1]$

$Output[2] = S^*[0]\ \overline{S^*[1]}$

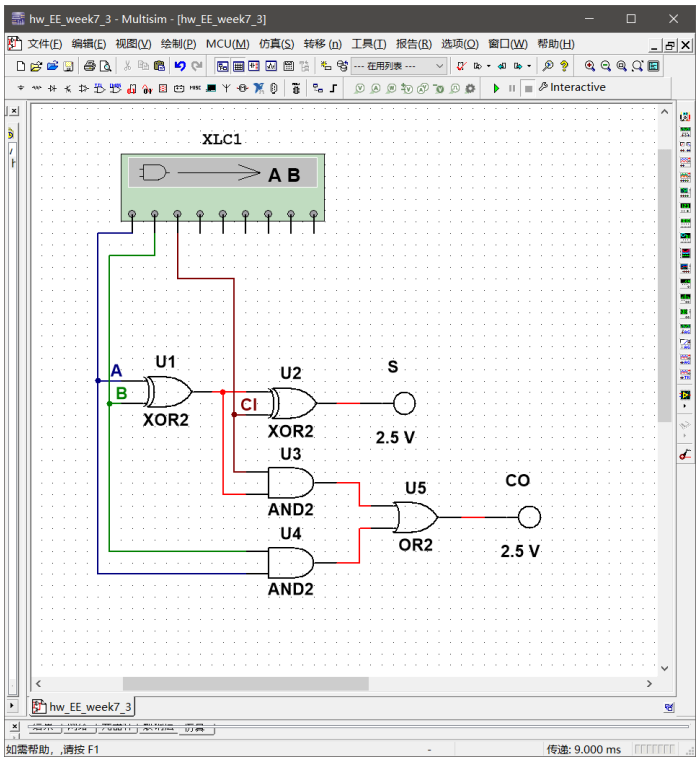
$Output[3] = S^*[0]\ S^*[1]$

Performance(Red for Output[0], Orange for Output[1], Yellow for Output[2], Green for Output[3]):

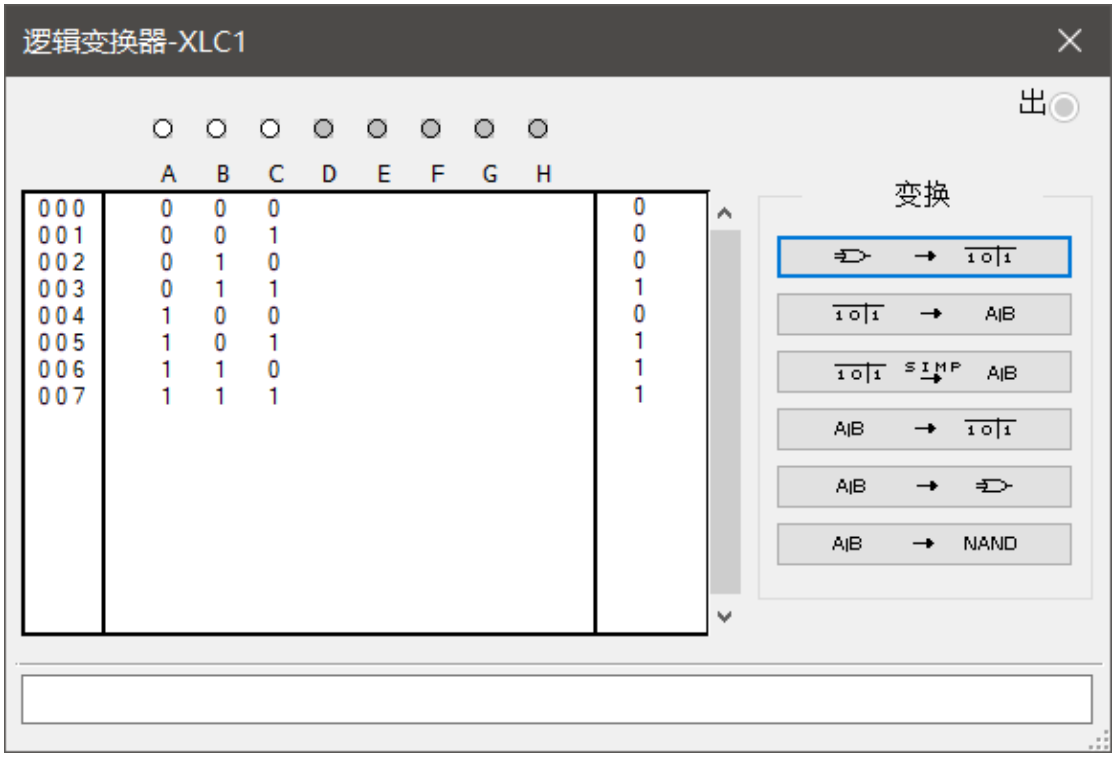


3. More combinational logic

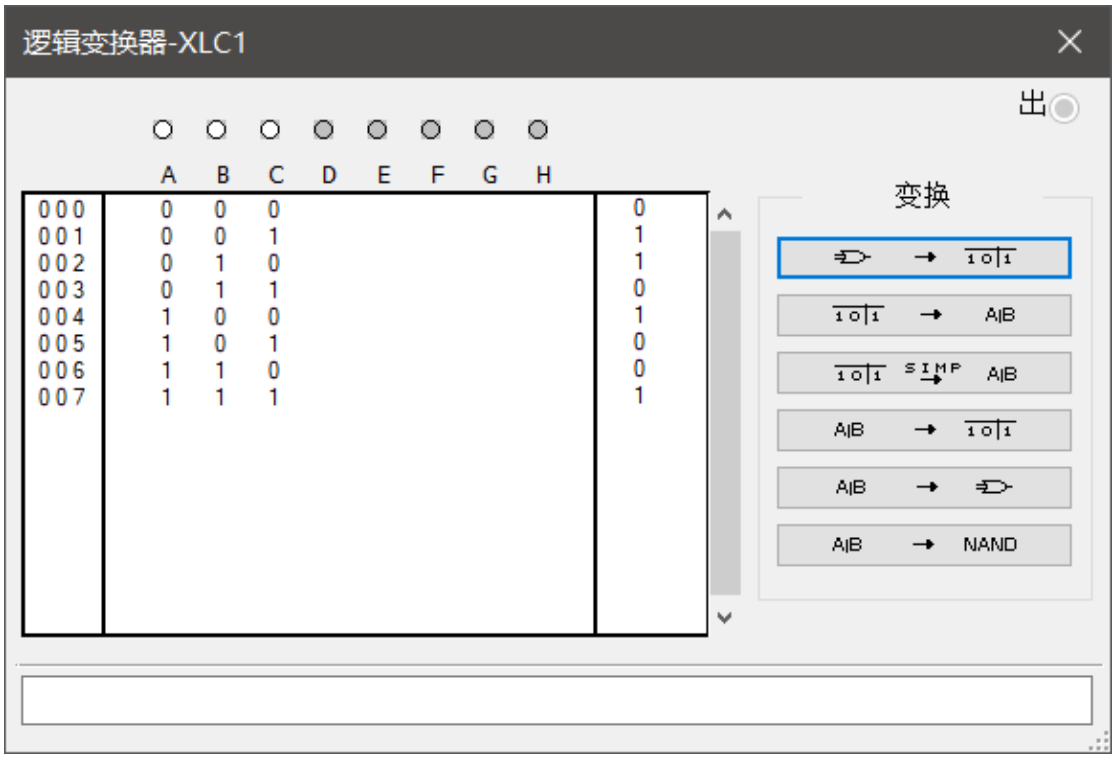
- Implement a full adder with basic logic gate (NOT, AND, OR, NOR, NAND, etc.). The inputs are A, B and CI (carry input). Outputs are S and CO (carry output). Use logic converter to show its truth table.




Truth table for CO:



Truth table for S:

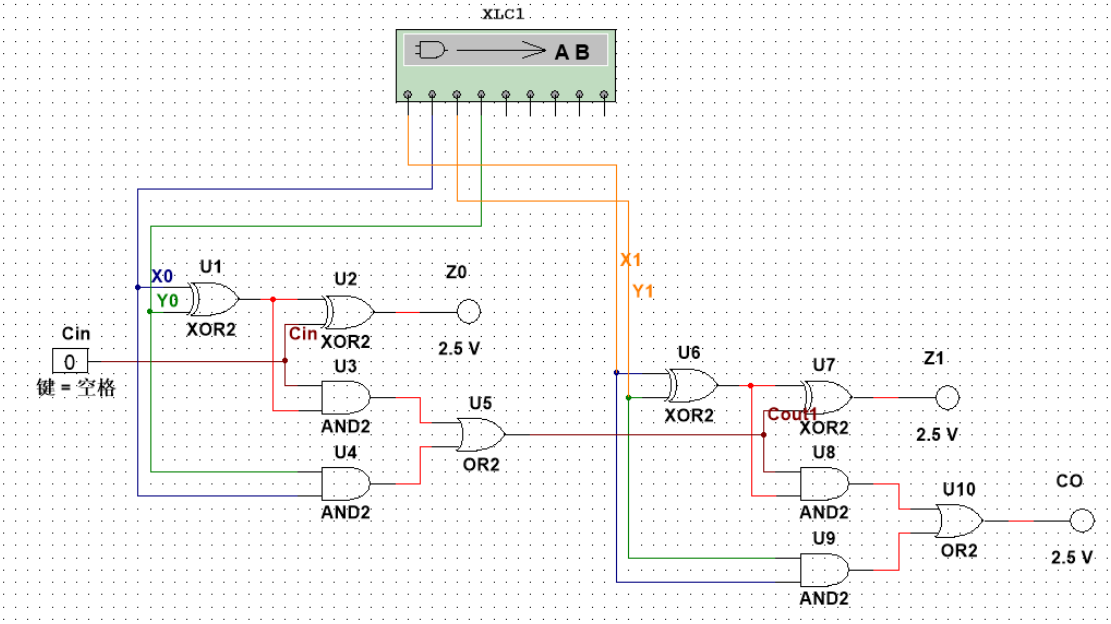



Truth table:

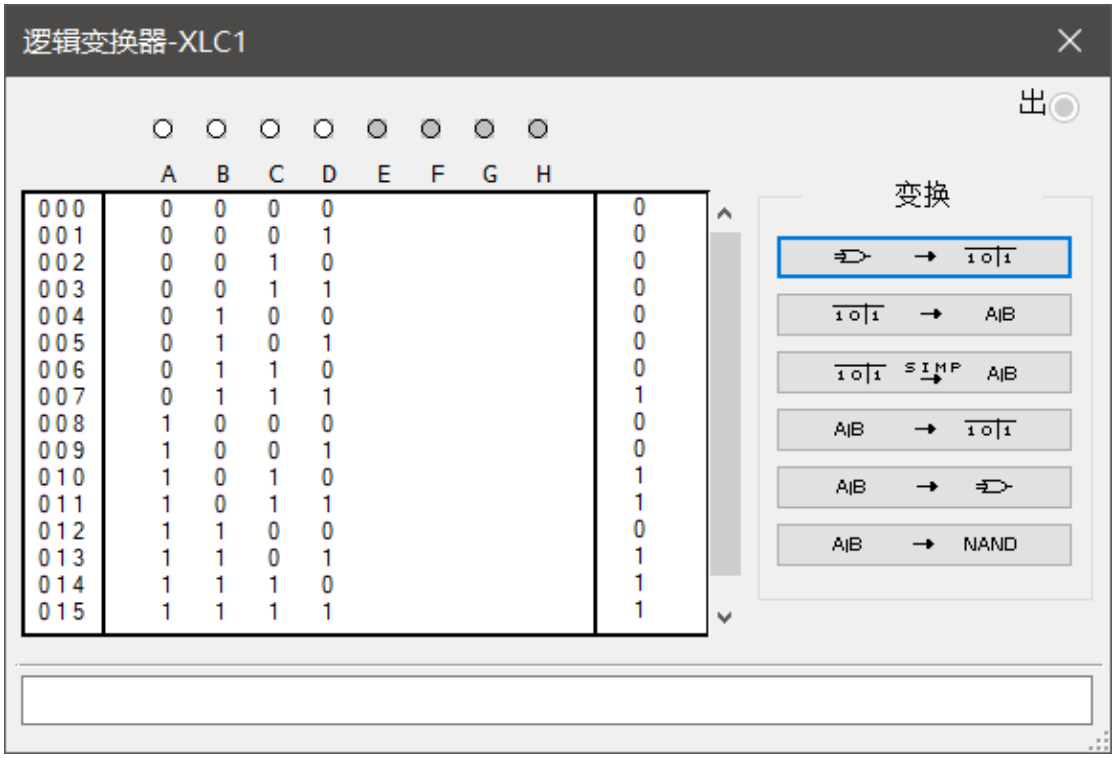
| Inputs | | | Outputs | |
|---|---|---|---|---|
| A | B | CI | CO | S |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

- Use the full adder you've built in the last problem, implement a 2 bits adder. The inputs are two binary number X and Y, each containing two bits. For X, it's X1, X0, and for Y it's Y1, Y0; Outputs are Z1, Z0 and one carry bit CO. The purpose is to calculate X+Y. Show your circuit schematic. Use logic converter to show the truth table of Z1, Z0 and C. (10')

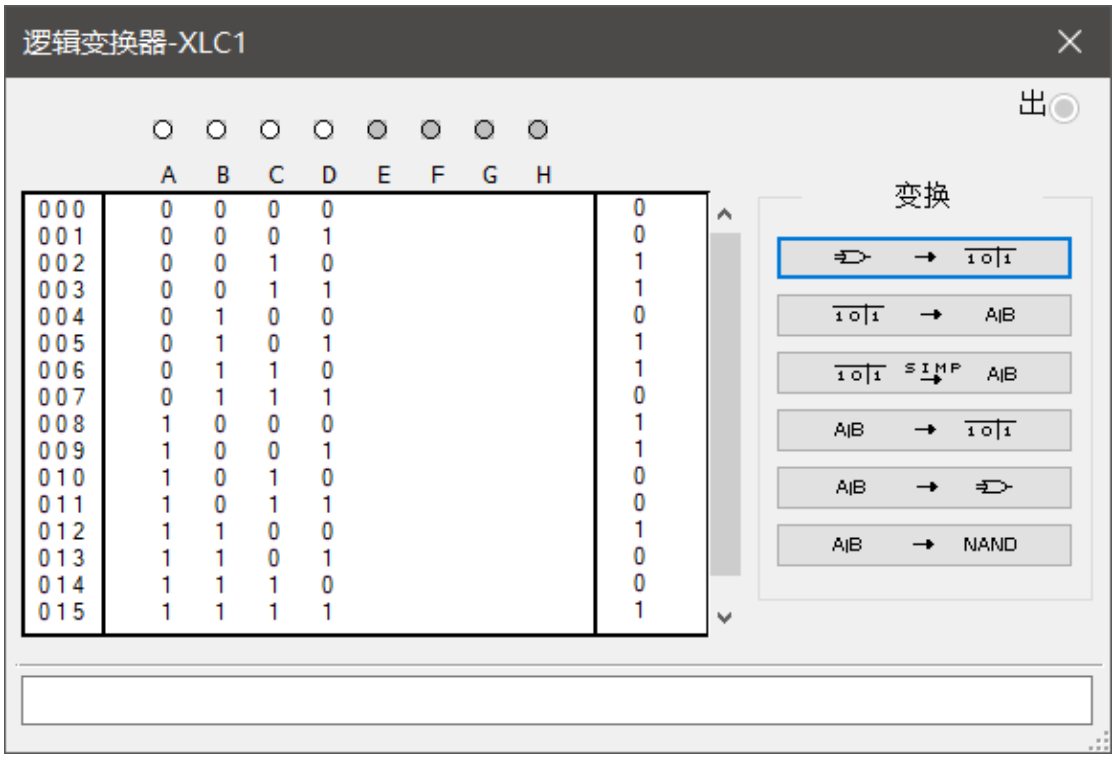
The 2 bits adder:



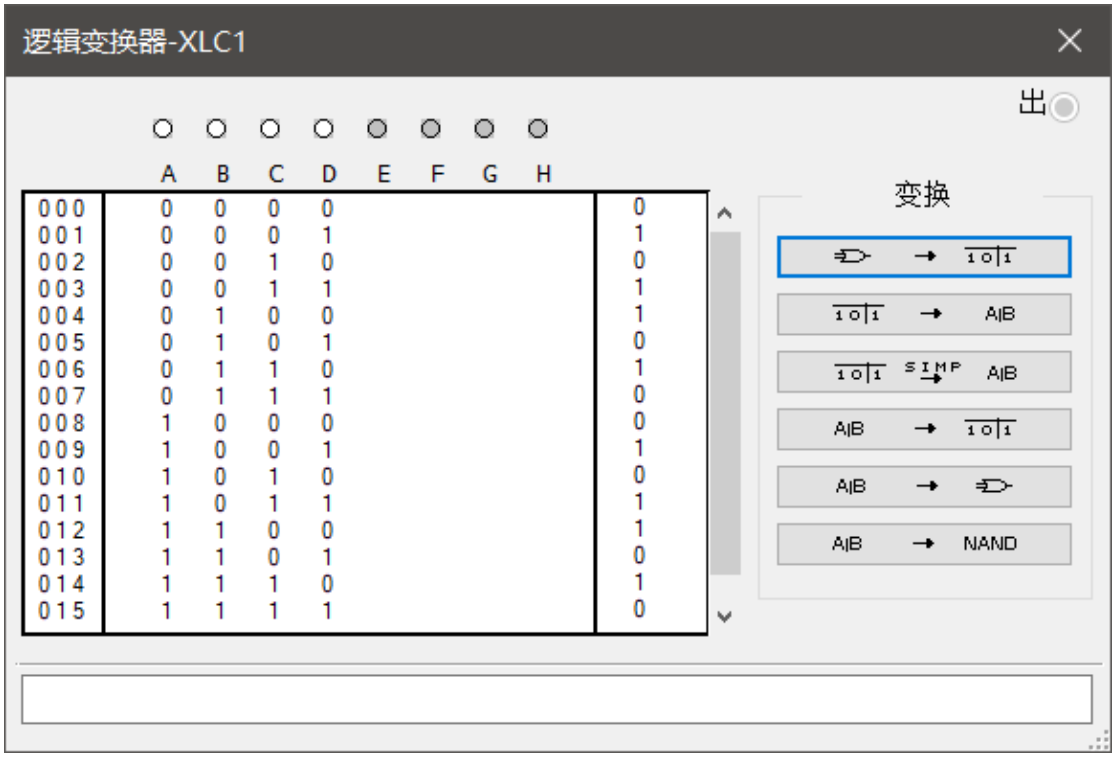

Truth table for CO:



Truth table for Z1:



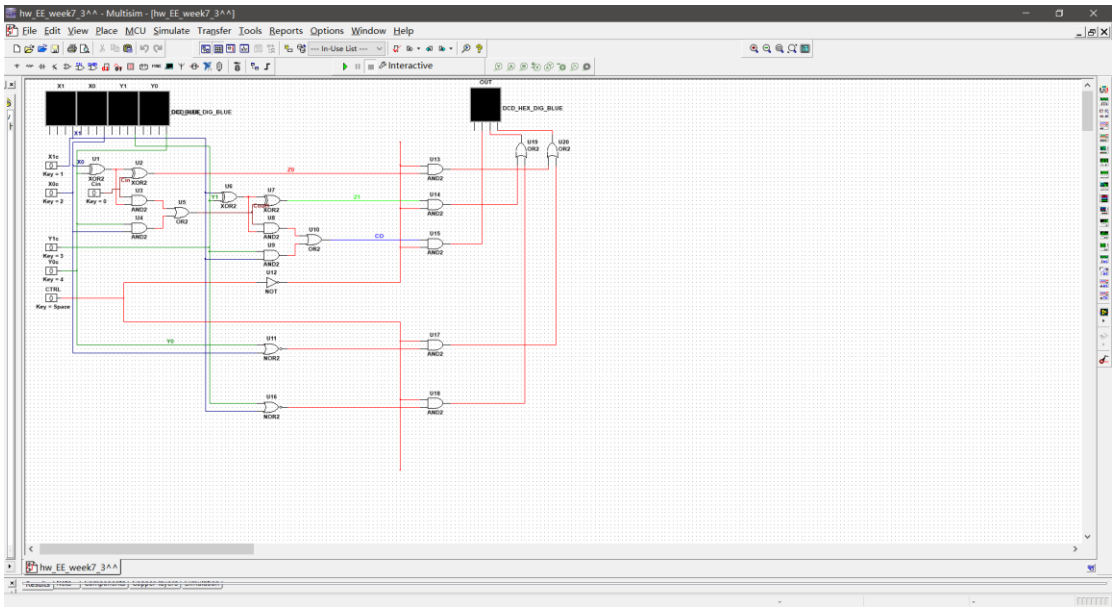

Truth table for Z0:



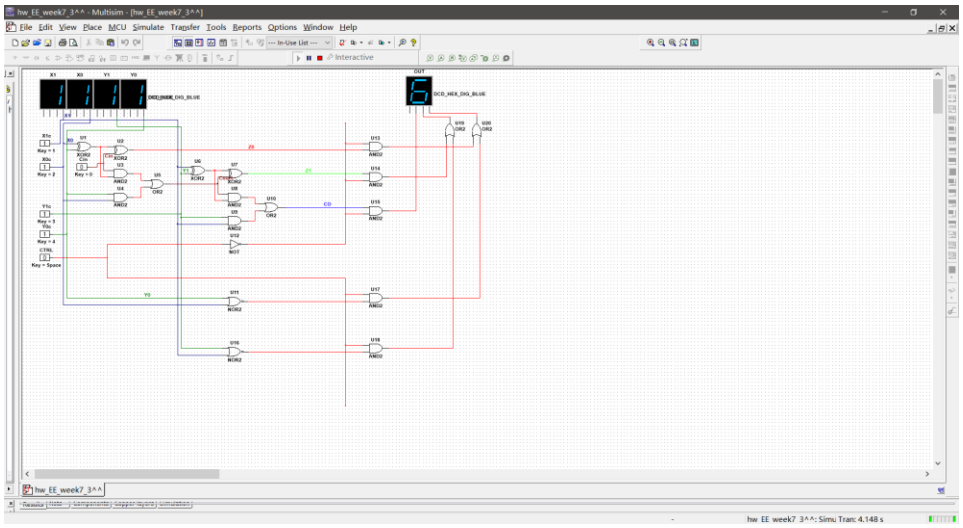
Truth table:

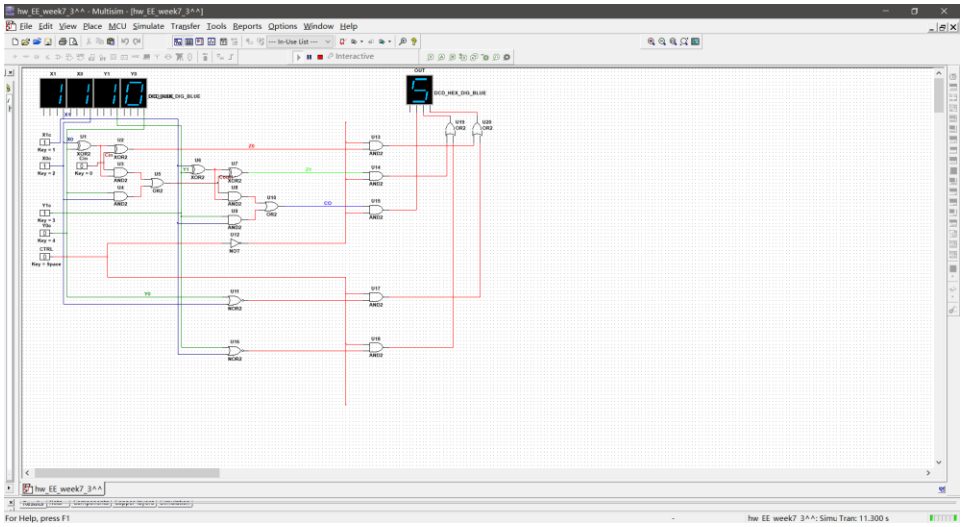
| Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|
| X1 | X0 | Y1 | Y0 | CO | Z1 | Z0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |

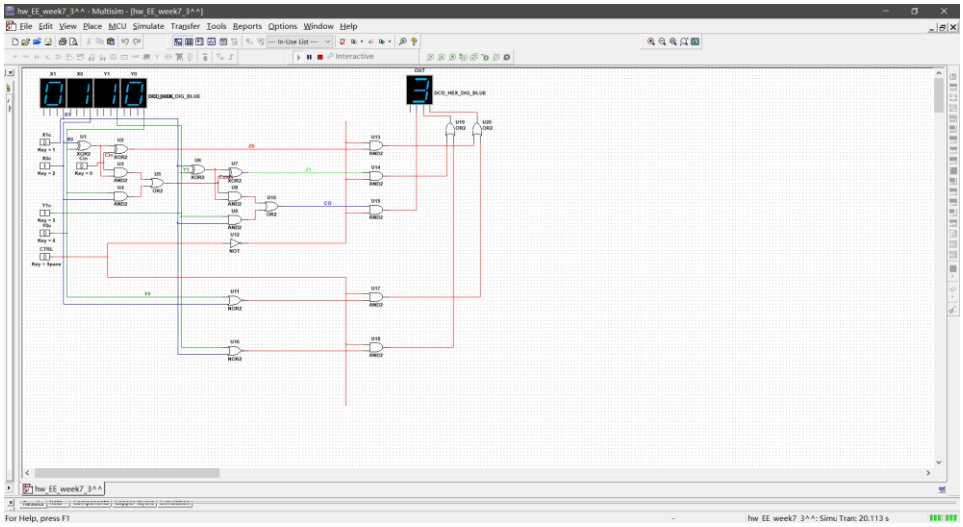
- Bonus: Implement an 2bit ALU. The control bit is CTRL, when CTRL is LOW, your ALU will perform addition like the full adder in the last question. When CTRL is HIGH, your ALU will perform NOR operation (In other words, When CTRL=1, Z0 = X0 nor Y0, Z1 = X1 nor Y1). Use 7-segment display to show the input and output. (10’)

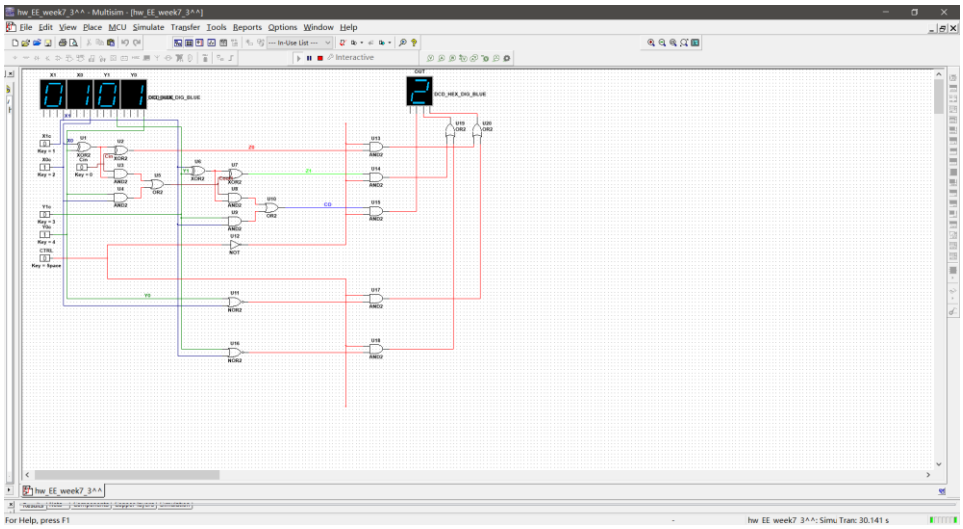


CTRL=0:

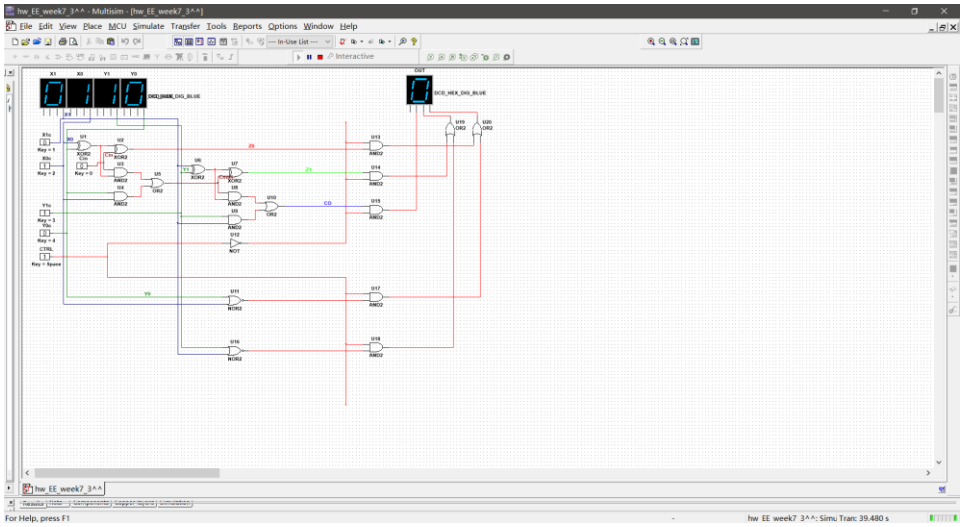

hw_EE_week7_3^^ - Multisim - [hw_EE_week7_3^^]
File Edit View Place MCU Simulate Transfer Tools Reports Options Window Help
hw_EE_week7_3^^
hw_EE_week7_3^^: Simu Trans: 4.148 s

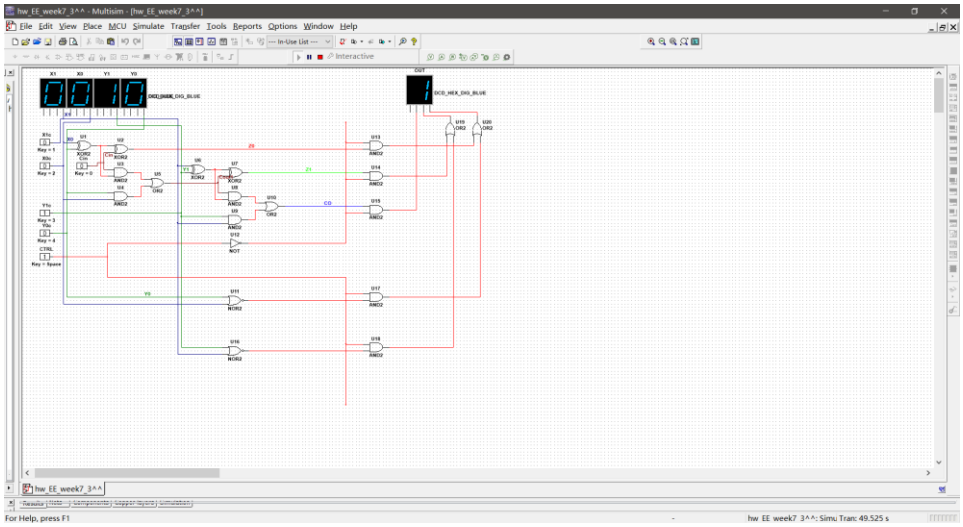

hw_EE_week7_3^^ - Multisim - [hw_EE_week7_3^^]
File Edit View Place MCU Simulate Transfer Tools Reports Options Window Help
hw_EE_week7_3^^
For Help, press F1
hw_EE_week7_3^^: Simu Trans: 11.300 s

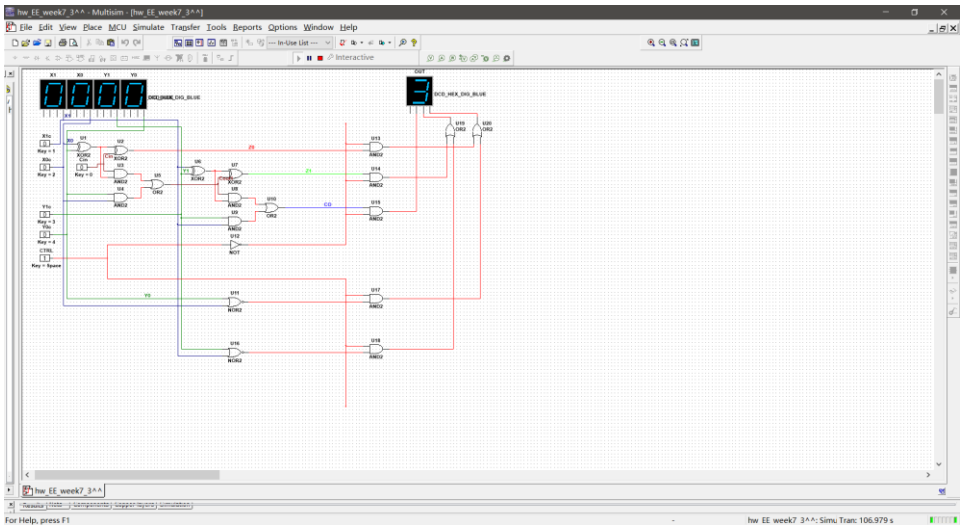

hw_EE_week7_3^^ - Multisim - [hw_EE_week7_3^^]
File Edit View Place MCU Simulate Transfer Tools Reports Options Window Help
hw_EE_week7_3^^
For Help, press F1
hw_EE_week7_3^^: Simu Trans: 20.113 s

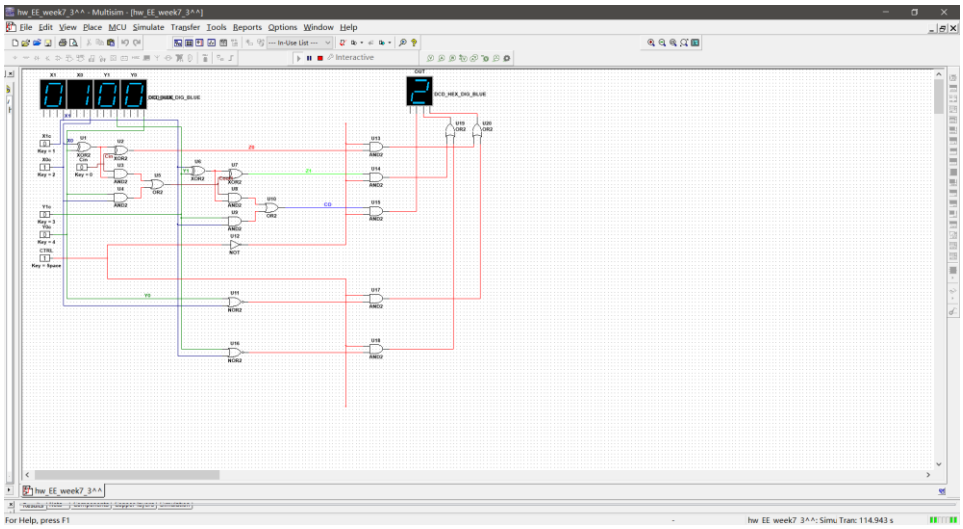


CTRL = 1 :









## 4. MCU development

**Control the brightness of the on board led of your ESP32 by using PWM. Let the LED’s brightness increase linearly from 0% to 100%, then let it decrease from 100% to 0%, then turn off the LED. The speed of increase and decrease should be the same, and the whole process should be done between 8 to 12 seconds. (20’)**

```
from machine import Pin, PWM
import time
led = PWM(Pin(2))
led.freq(1000)

for i in range(0,1024):
    led.duty(i)
    time.sleep_ms(5)

for i in range(1023, -1, -1):
    led.duty(i)
```

```
    time.sleep_ms(5)


led.value(0)
```