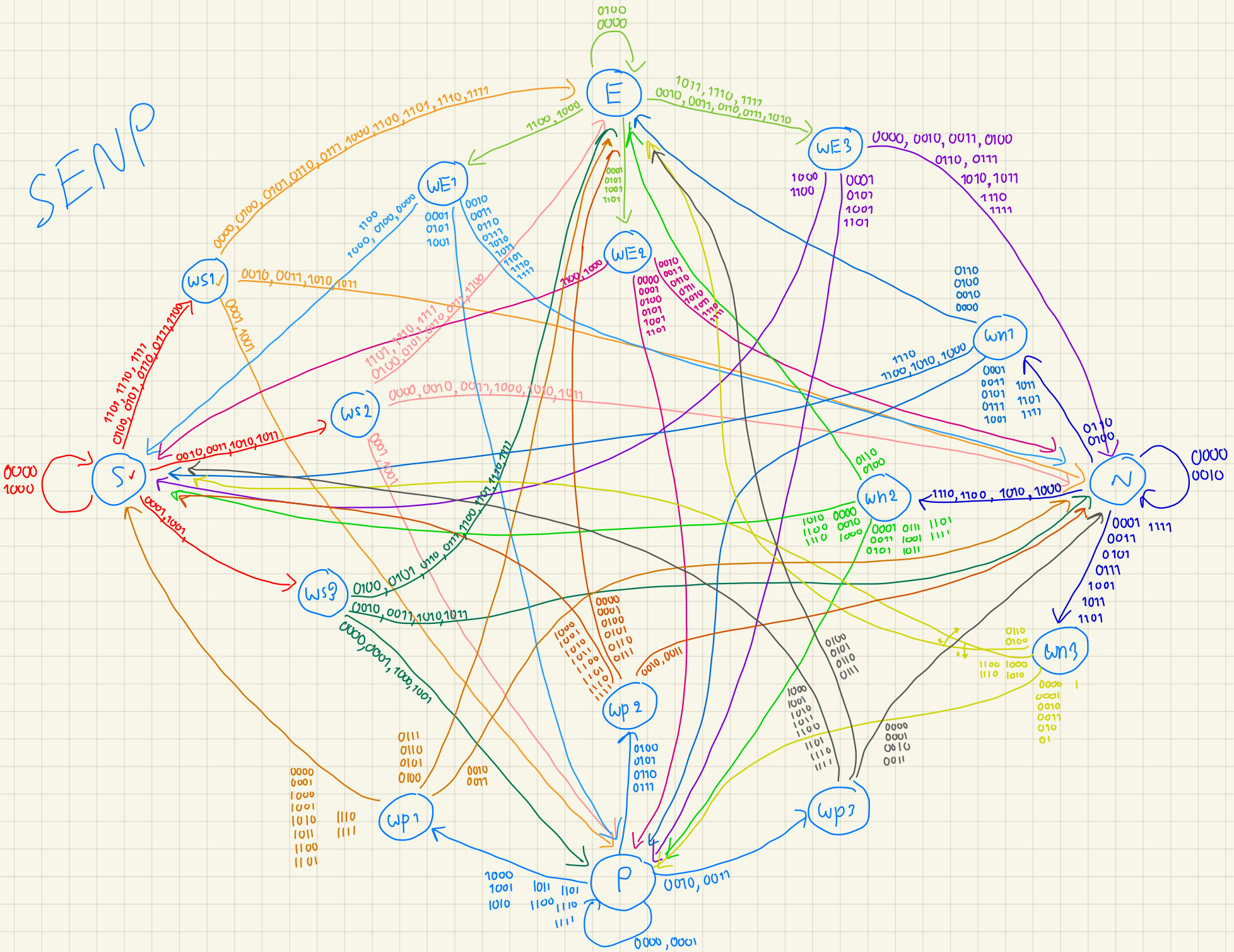
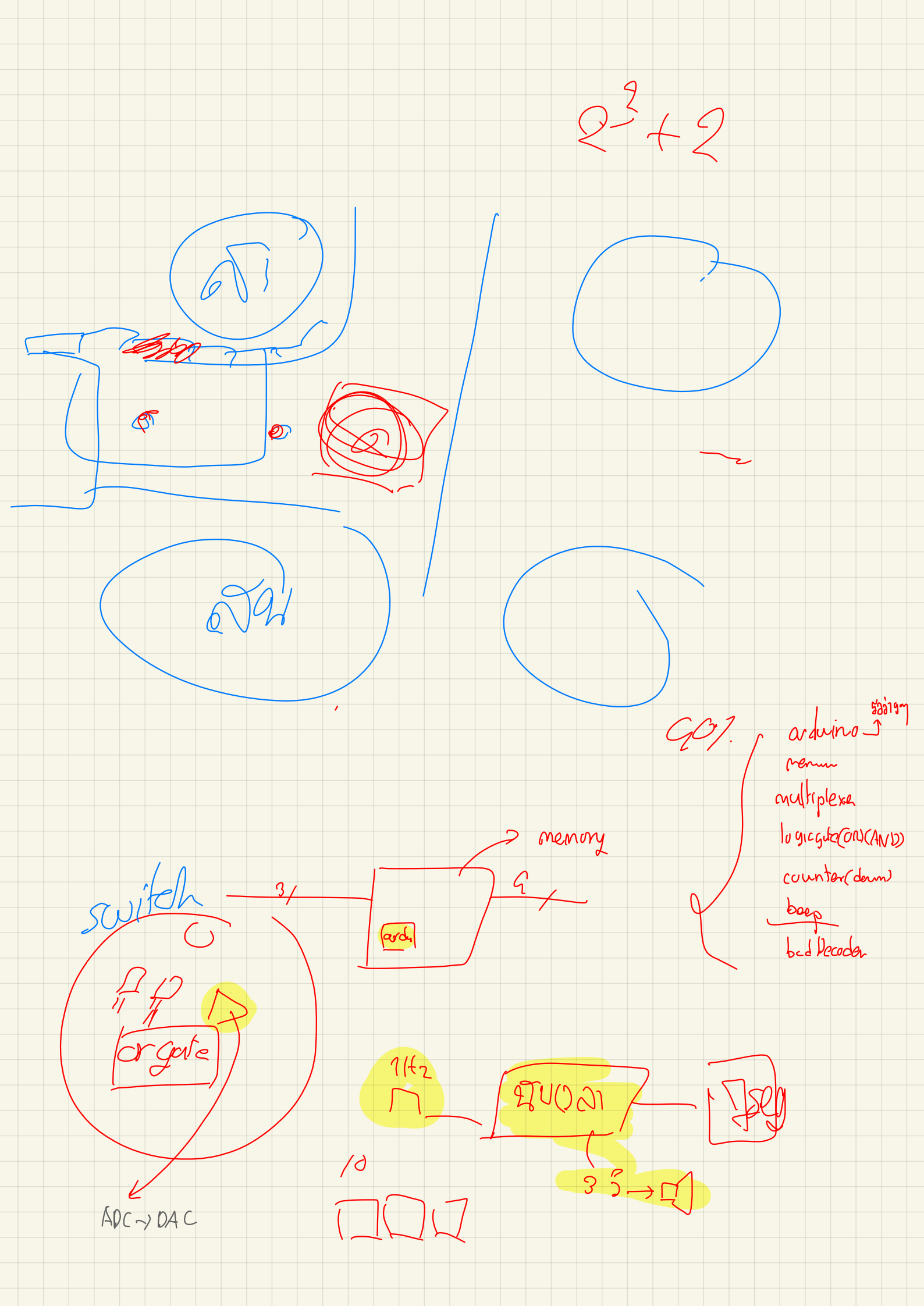
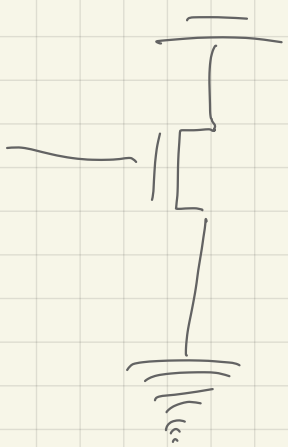

SENP
E
WE1
WE2
WE3
WS1
WS2
WS3
WN1
WN2
WN3
S
N
WP1
WP2
WP3
P


memory
switch
3
4
ard
or gate
arduino
memory
multiplexer
logic gate(OR)(AND)
counter(down)
beep
bcd Decoder
ADC → DAC

000 → 111
001 110
010 101
011 100



Finite State Machine Using VHDL

การใช้งานเบื้องต้น (tip)

↳ ดูวงจรที่ gen ออกมาจาก vhdl

↳ สร้างไฟล์แบบ vhdl

↳ ไปที่ design > synthesize > view technology schematic ✓

↳ ms test input ที่บรรยายที่สร้างจาก vhdl

↳ (right click) on vhdl module >> new source >> vhdl test bench >> (input) [fileName] >>next >>next >> finish

↳ ลิ้นแรกที่เราควรทำ

↳ ลบบรรทัดที่มีคำว่า constant <clock>_period : time := 10 ns; //62

และ

```
-- Clock process definition                    // 74-81
<clock>_process :process
begin
    <clock> <= '0';
    wait for <clock>_period/2;
    <clock> <= '1';
    wait for <clock_period>/2;
end process;

wait for <clock>_period/2;     //82
```

↳ สำคัญใน vhdl test bench เราจะมาแก้ไข input ไม่ยุ่งกับ output

ตรงส่วน code

```
stim_proc: process
begin
    wait for 100 ns;
    -- insert stimulus here
    ไม่ syntax →  name_port_input <= 'logic';
    wait;
end process
```

↳ คำสั่งเพิ่มเติม
↳ wait
↳ รูปแบบ wait for [time] [unit];

```
wait for 10 ns;
```

↳ เมื่อเขียน vhdl work bench เสร็จแล้ว
↳ ขั้นตอน เปลี่ยนจาก implementation เป็น simulation
↳ ไปที่ tab design >> simulate behavioral >> yes
↳ ตัวนี้จะเรียกขึ้น simulate ใน new window

↳ **keyword : signal**
↳ represent wires within a circuit.

```
architecture [name] of [entity] is
    signal a, b, temp : std_logic;
begin
    temp <= a and b;
    c <= temp and D;
end [name];
```

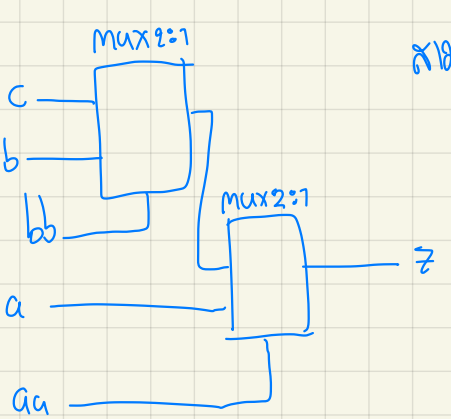
ใช้เพื่อสร้างสายไฟ
↳ ต้องเอาไป assign กับ input และ output ของ entity ด้วยนะ

↳ **How to use "when"** (concurrent statement)
↳ Select using when statement

```
z <= a when aa = '1' else
     b when bb = '1' else
     ⋮
     c;
```

เหมือนการวนลูป

↳ เป็น priority logic
↳ เช่น aa = '1' จะไม่สนใจเงื่อนไขที่เหลือเลย



สายสัญญาณ 'z'
↳ tri state

↳ **How to use 'with' state ment** (concurrent statement)
↳ syntax

```
with [condition_expression] select
    target <= expression1 when choice1, -- เช่น choice1 = condition_expression
              expression2 when choice2,
              expressionN when choiceN;
    - - '0' when others; -- optional
```

เป็น non priority logic → ลูกศรแบ่งกัน

↳ Process and Sequential statements
  ↳ how does process work?
  ↳ sequential constructs → if statement
                          ↳ case statement
  ↳ type of process
  ↳ hardware modeling example

↳ Process
  ↳ process are executed one after another in the order in which they are written.

process ( ) — sensitivity list → the change of the signal will execute this.
  begin
  ═══
  ═══ } these statement will execute sequentially (เหมือนการเขียนโปรแกรม)
  ═══
end process;

while concurrently

  ↳ เมื่อมีหลายๆ process มันจะทำงานแบบ concurrently แต่ภายในทำงานแบบ sequentially

  ↳ variable can only be used inside the process.

  ↳ process is primary concurrently VHDL statement, described sequential behavior
    ↳ เช่น ff, latch, reg
    ↳ การทำงานแบบ sequential การที่มันจะใช้งานแบบไหนก็ขึ้นอยู่กับตัวเรา

  ↳ NOTE: sequential statement does not always generate sequential hardware.

  ↳ ยกเว้น wait statement, put only one driver to the signal

  ↳ If statement
    ↳ concept is the same of c, but the result to hw can be different.

```
if [condition] then [sequential_statement]
elsif [condition] then [sequential_statement]
else [~~] end if;
```

  ↳ Case statement
    ↳ เหมือน with select

```
case [expression] is
        when [choice] => [statement]
        :
        when others => [statement]
end case;
```

→ tip

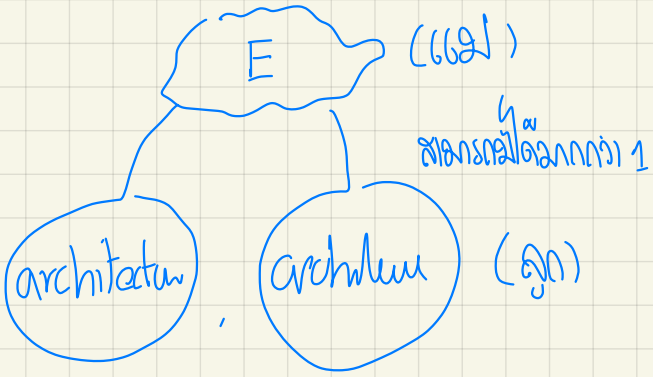
```
when 0 to 4 => x <= '1';
```

(เขียนแบบรวม range(0-4))

ห้ามมีการ overlap

Entity

↳ กำหนดชื่อของ input, output



Architecture

↳ ส่วนนี้

↳ behavior → how my design work?

↳ function → how output = f(input)

↳ reration between input and output of entity

↳ มี syntax

```
architecture [architecture_name] of [entity_name] is
  -- declarations
begin
   -- concurrent_statements   ทำงานพร้อมกัน execute at the same time
end [architecture_name];
```

↳ a design can be described in an architecture using various levels of abstraction

↳ in gate level, behavior

↳ if you know how the input and output is

↳ gate level

↳ เขียนคล้ายการต่อวงจรดิจิตอล

```
architecture [name] of [entity] is
    begin
        c <= a and b;
        x <= a xor d;
end [name];
```

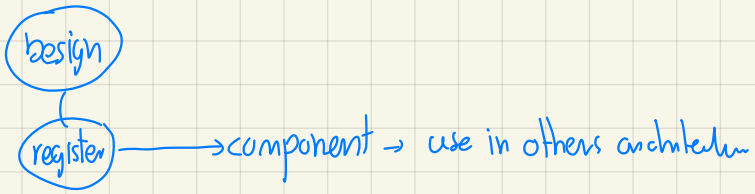
↳ behavioral style

↳ more higher level

```
architecture [name] of [entity] is
    begin
        process(a,b)
            begin
                if a = '1' and b = '1' then c <= '1';
                else c <= '0';
                end if;
        end process;
end [name];
```

รันคำสั่งแบบลำดับ

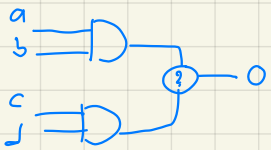
↳ structural
↳ it is possible to mix 3 modeling style in a single architecture body
↳ ใน designs vhdl มี libraries อยู่มากมาย เวลาเรียกใช้งานก็เรียกเป็น component



Driver

↳ เกิดขึ้นเมื่อมีการเขียน o<=a and b;

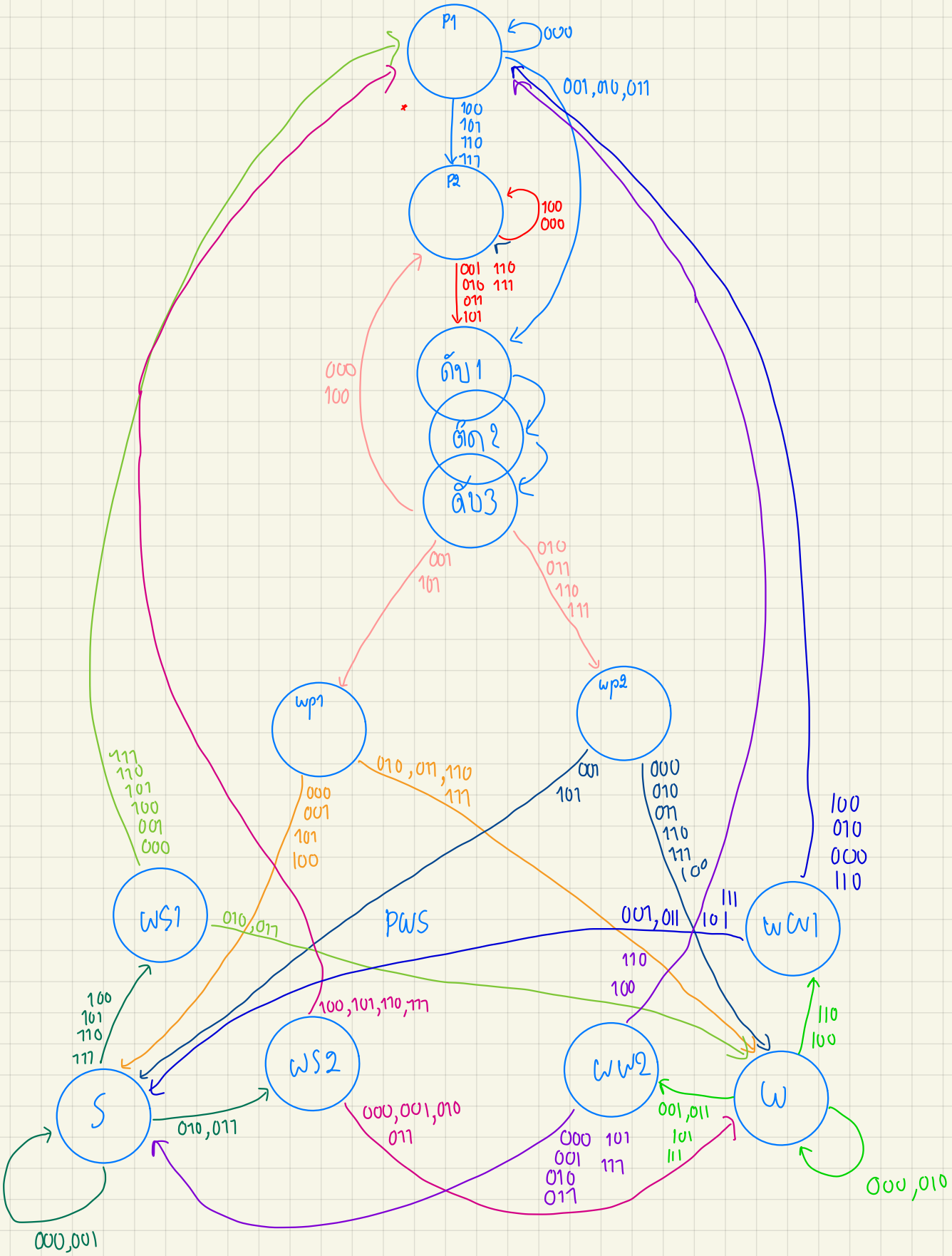
↳ ถ้าเขียน o <= a and b;
o <= c and d;

multiple driver 

ในการออกแบบ "เราไม่ต้องการ"

sig <= a when sel=1 else
↳ whe '1' else
default_value;

with sel select
sig <= a when "01",
z when others;


P1
000
001,010,011
100
101
110
111
P2
100
000
001
010
011
101
110
111
ดีบ 1
ดีบ 2
ดีบ 3
000
100
001
101
010
011
110
111
wp1
wp2
111
110
101
100
001
000
010,011,110
111
000
001
101
100
001
101
000
010
011
110
111
100
100
010
000
110
WS1
010,011
PWS
001,011
111
101
WW1
110
100
100
101
110
111
100,101,110,111
WS2
WW2
W
110
100
S
010,011
000,001,010
011
001,011
101
111
000
001
010
011
101
111
000,010
000,001

	rp	yp	gp	rs	ys	gs	rw	yw	gw
p1	0	0	1	1	0	0	1	0	0

p2

ตัวบ1

ตัวบ2

ตัวบ3

s

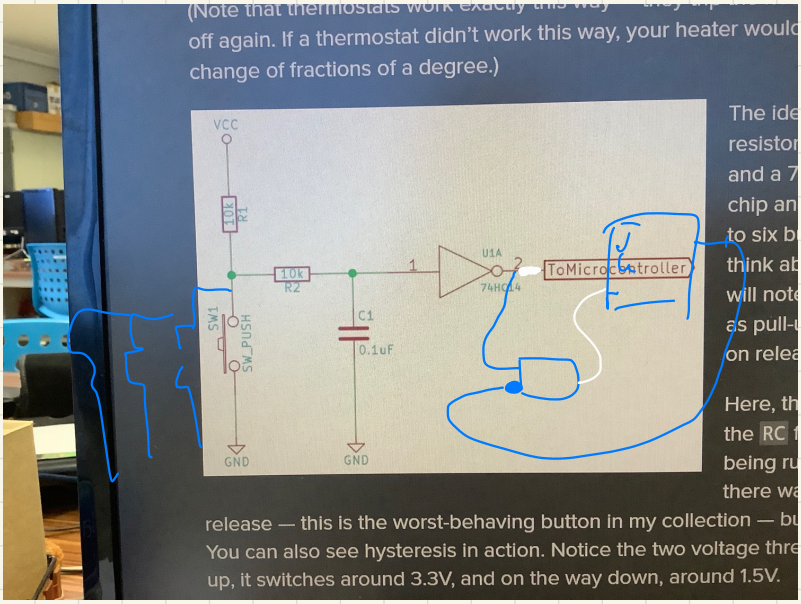
ws1

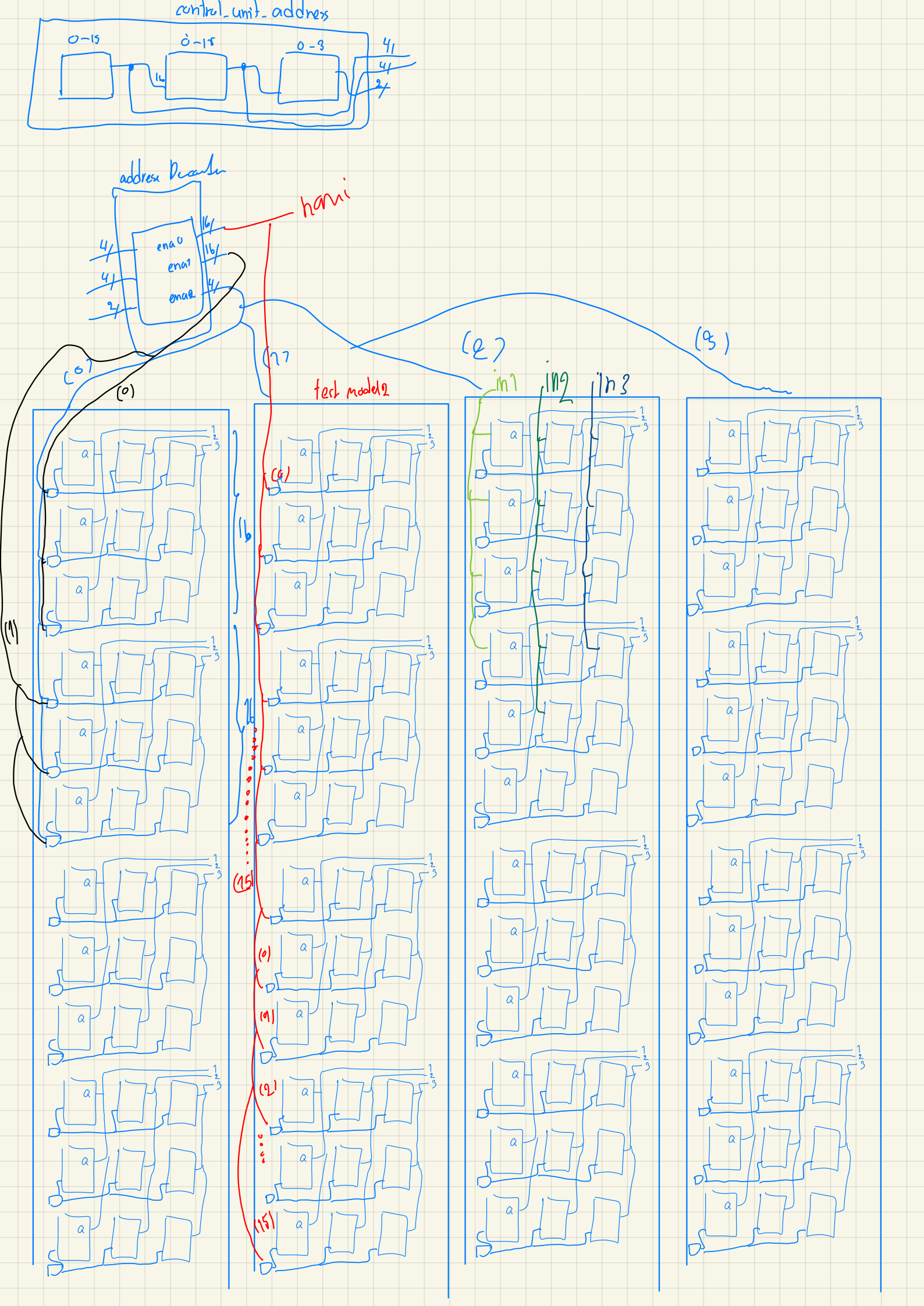
ws2

w

ww

ww2




control_unit_address
0-15
0-15
0-3
4
4
2
address decoder
hani
ena0
ena1
ena2
16
16
4
4
2
(0)
(7)
(2)
(3)
(1)
test model2
in1
in2
in3
16
(0)
(15)
(0)
(1)
(2)
(15)

RAM16X4S

↳ เมื่อ WE = '0'

↳ WCLK → ไม่สนใจ , ข้อมูลใน RAM อยู่

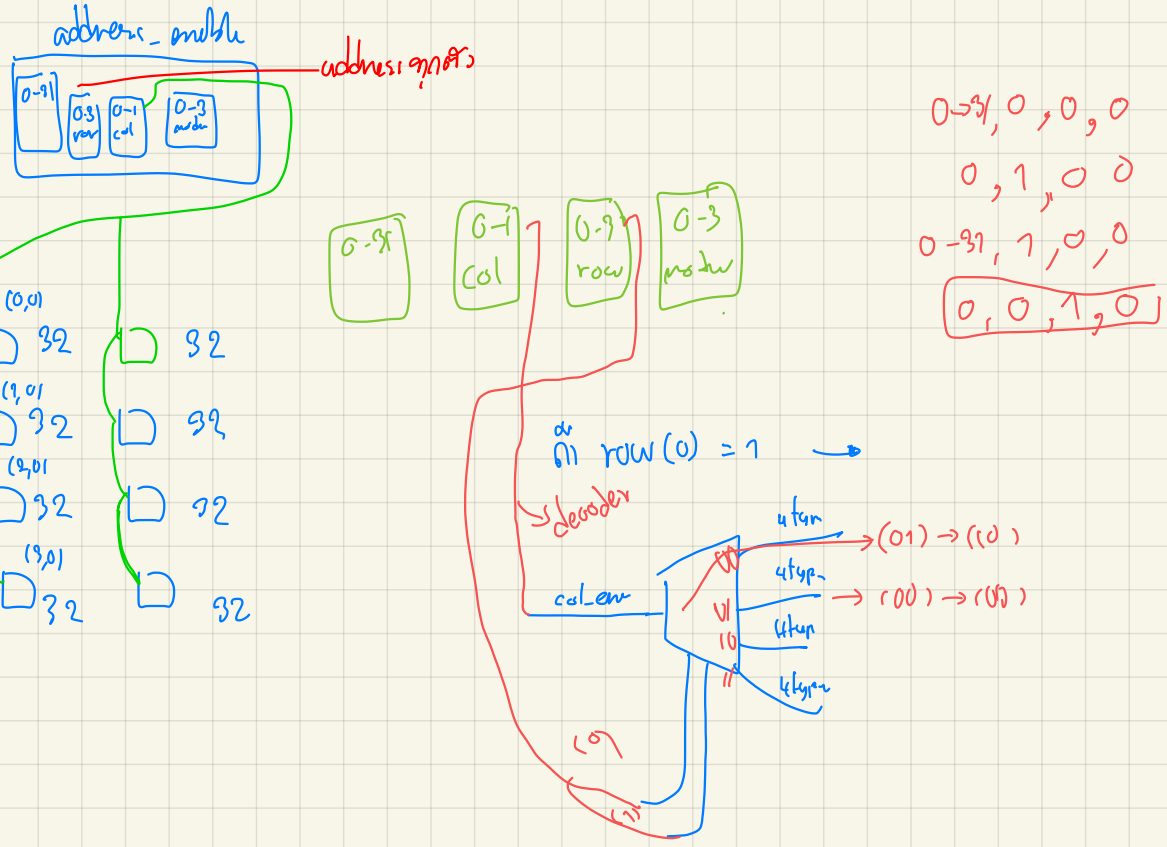
↳ เมื่อ WE = '1'

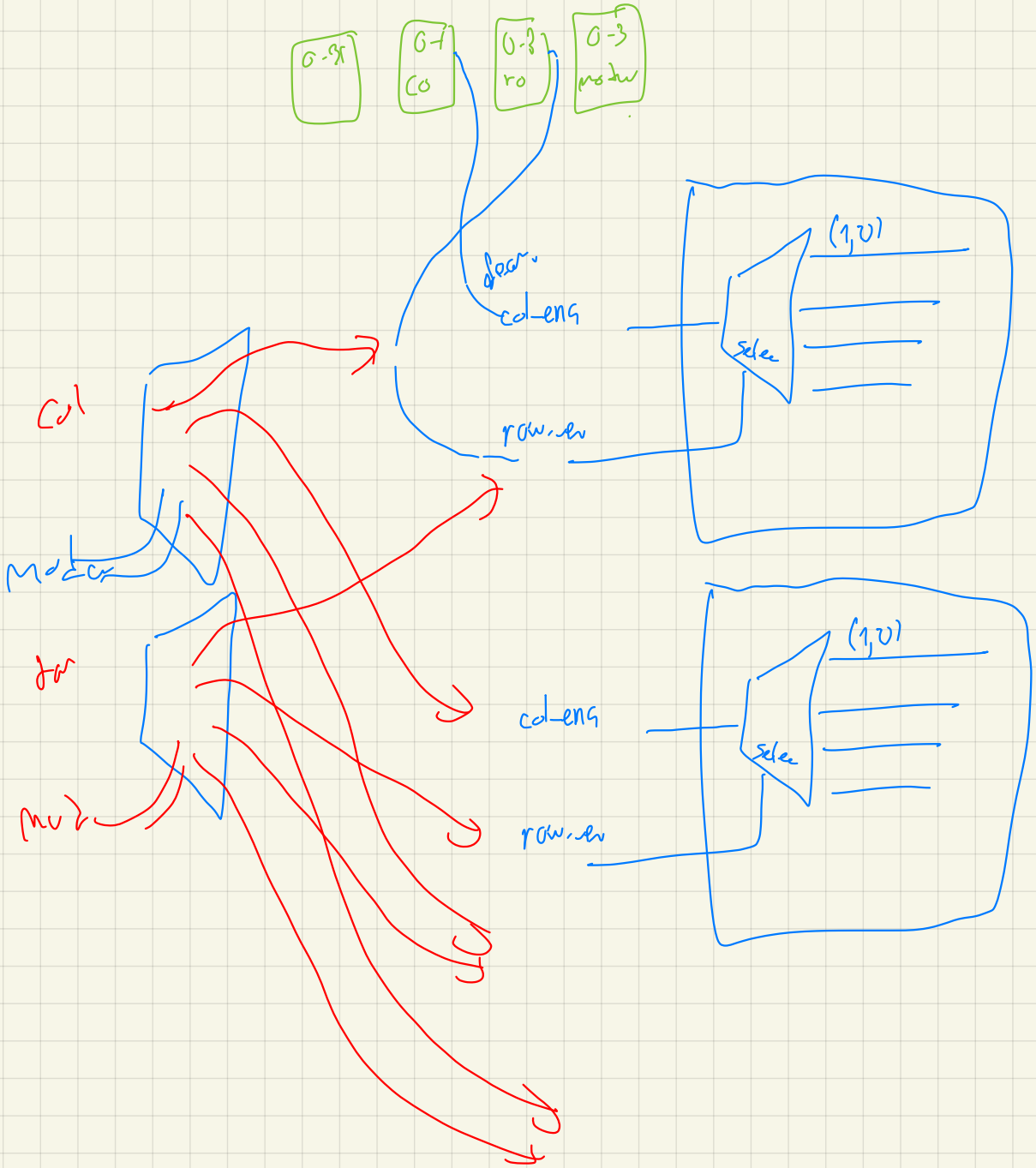
↳ WCLK = '1' , load data form (D3:D0) into the word selected by 4 bit add (A3:A0)

↳ ที่อยู่และข้อมูลต้องรอที่ ก่อนการเขียน ขึ้น ⎍ กับ WCLK

↳ ถ้าค่าเท่า 1, 0 ไม่ทำการเขียน

Inputs			Outputs
WE (mode)	WCLK	D3:D0	O3:O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D3:D0	D3:D0
1 (read)	↓	X	Data
Data = word addressed by bits A3:A0.			



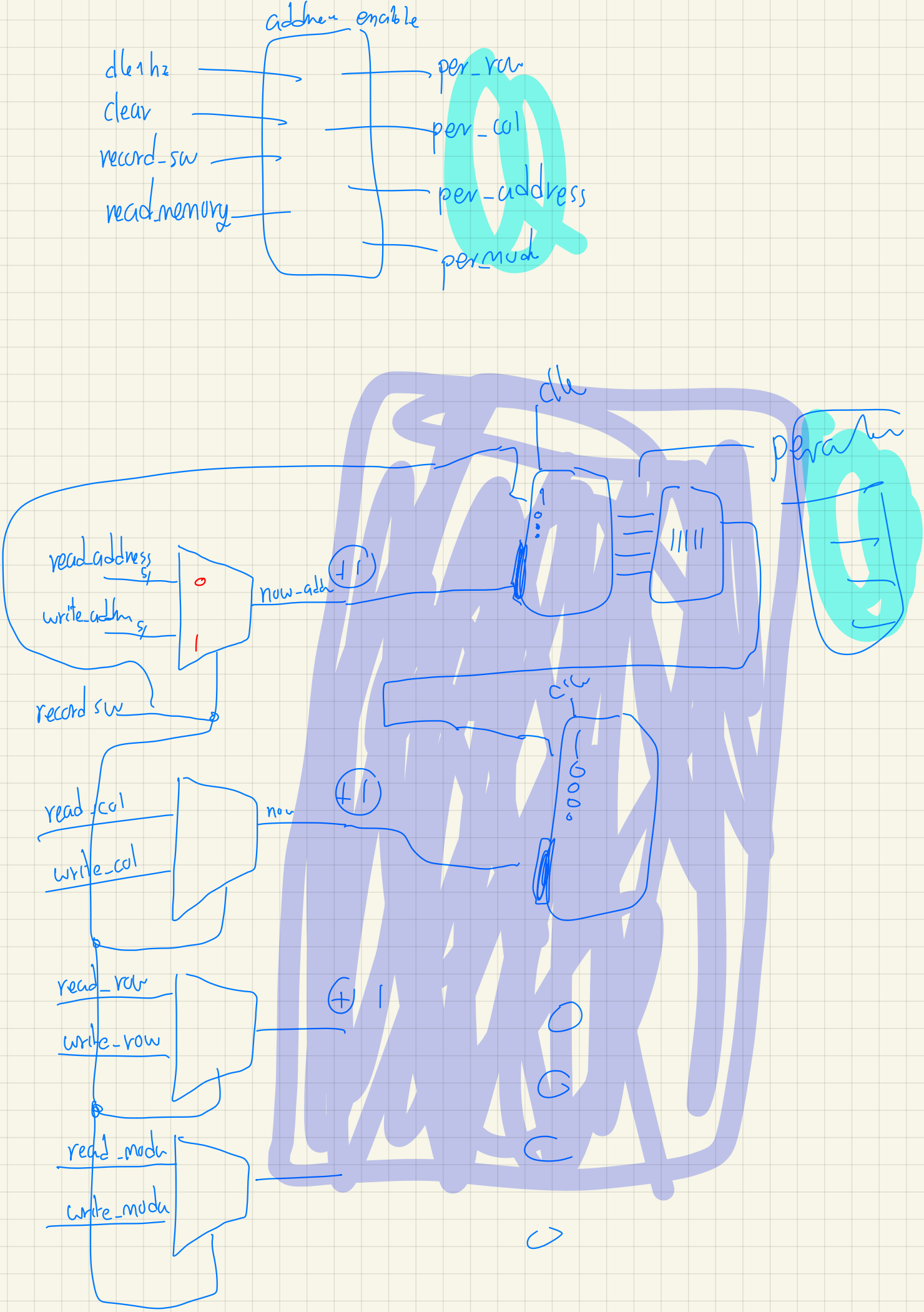


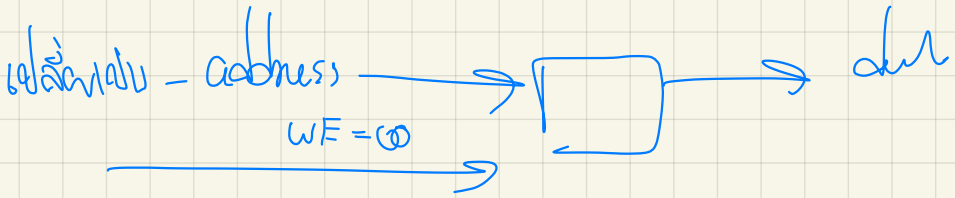
ram64 ใช้ selector ตัว col (decode) ดูว่าจะเลือก data ออก

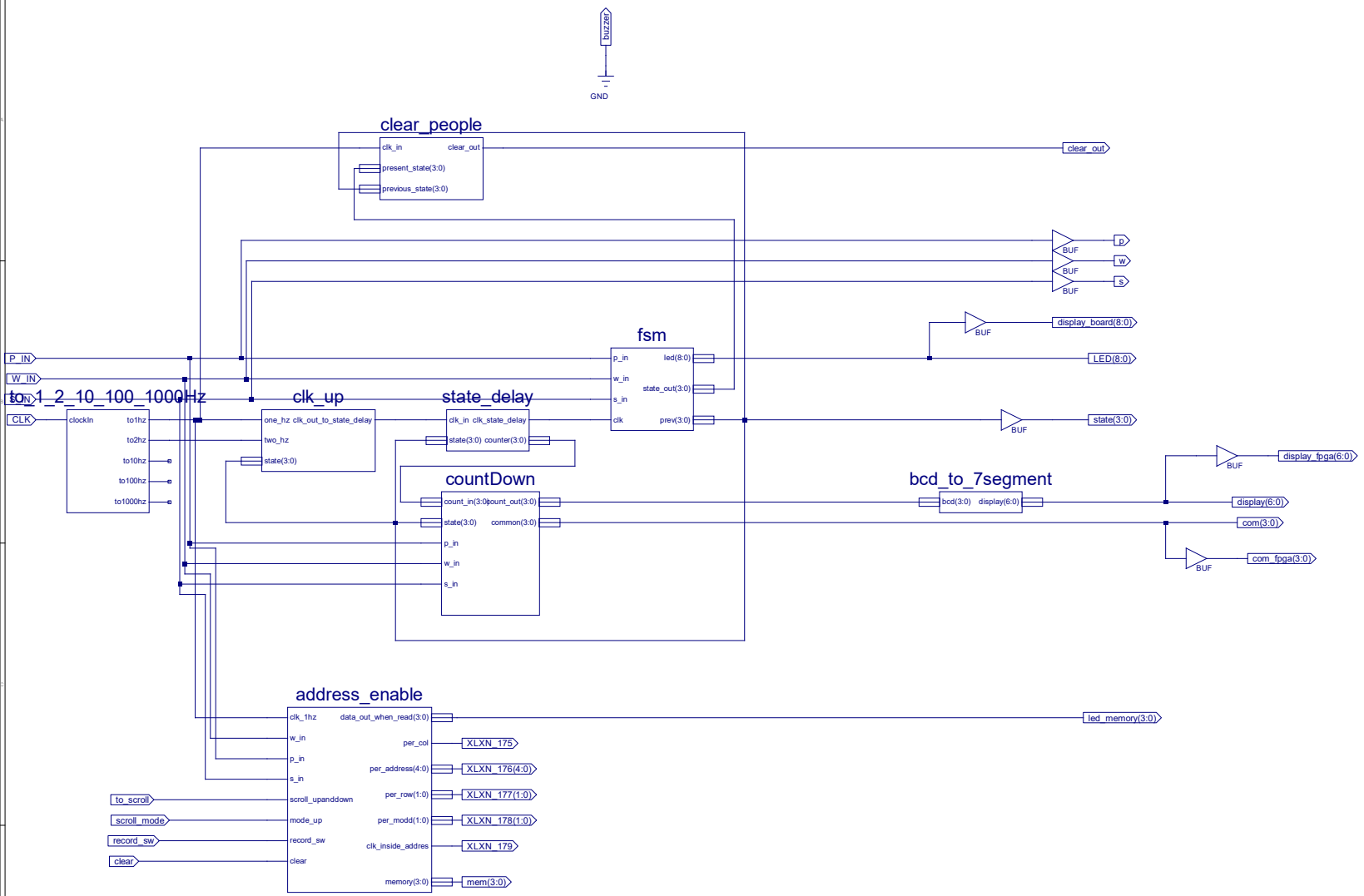
ram256 ใช้ selecte ตัว row (a_0, a_1) → ดูว่าจะเลือก data out

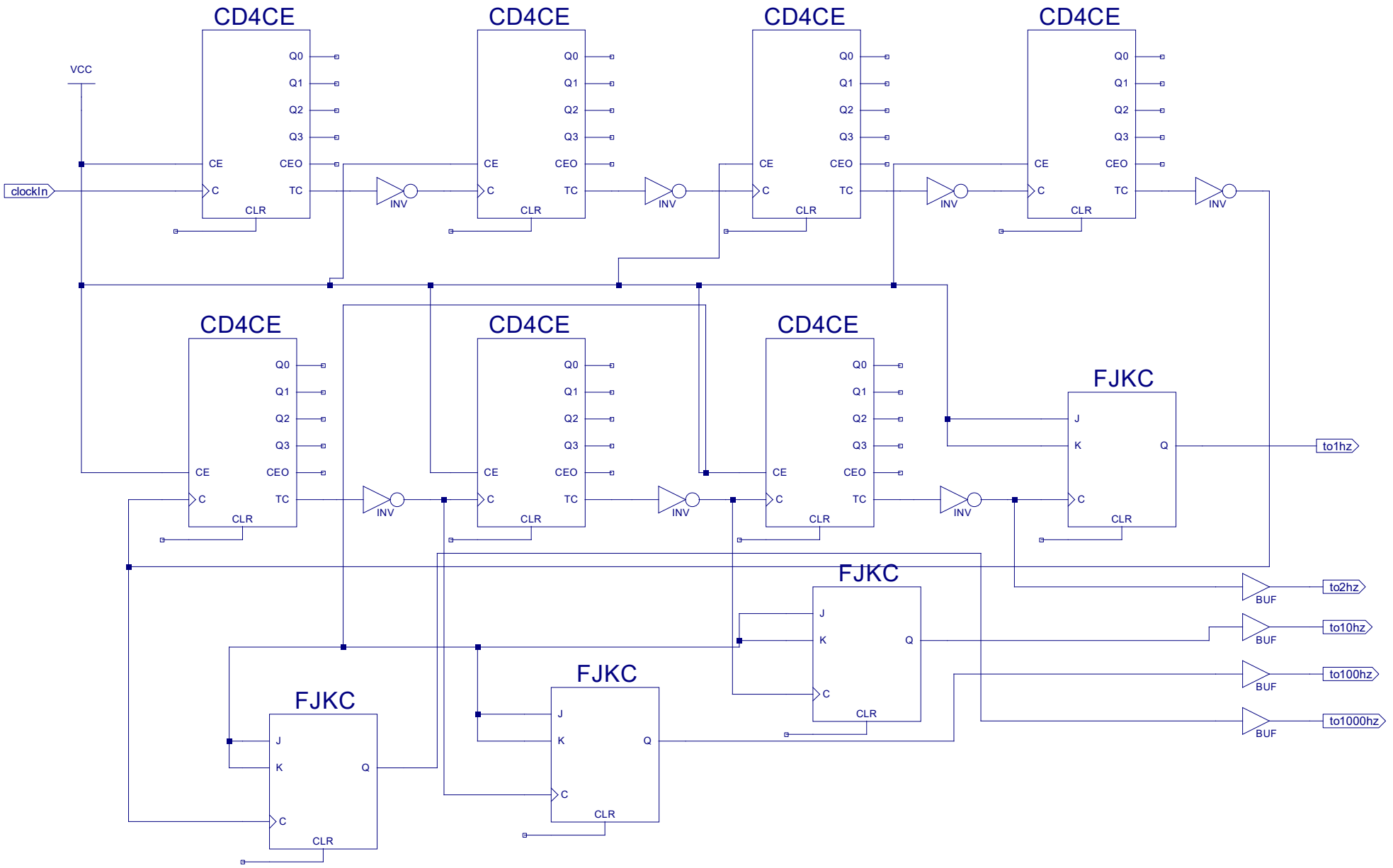
└ 00, 01, 10, 11

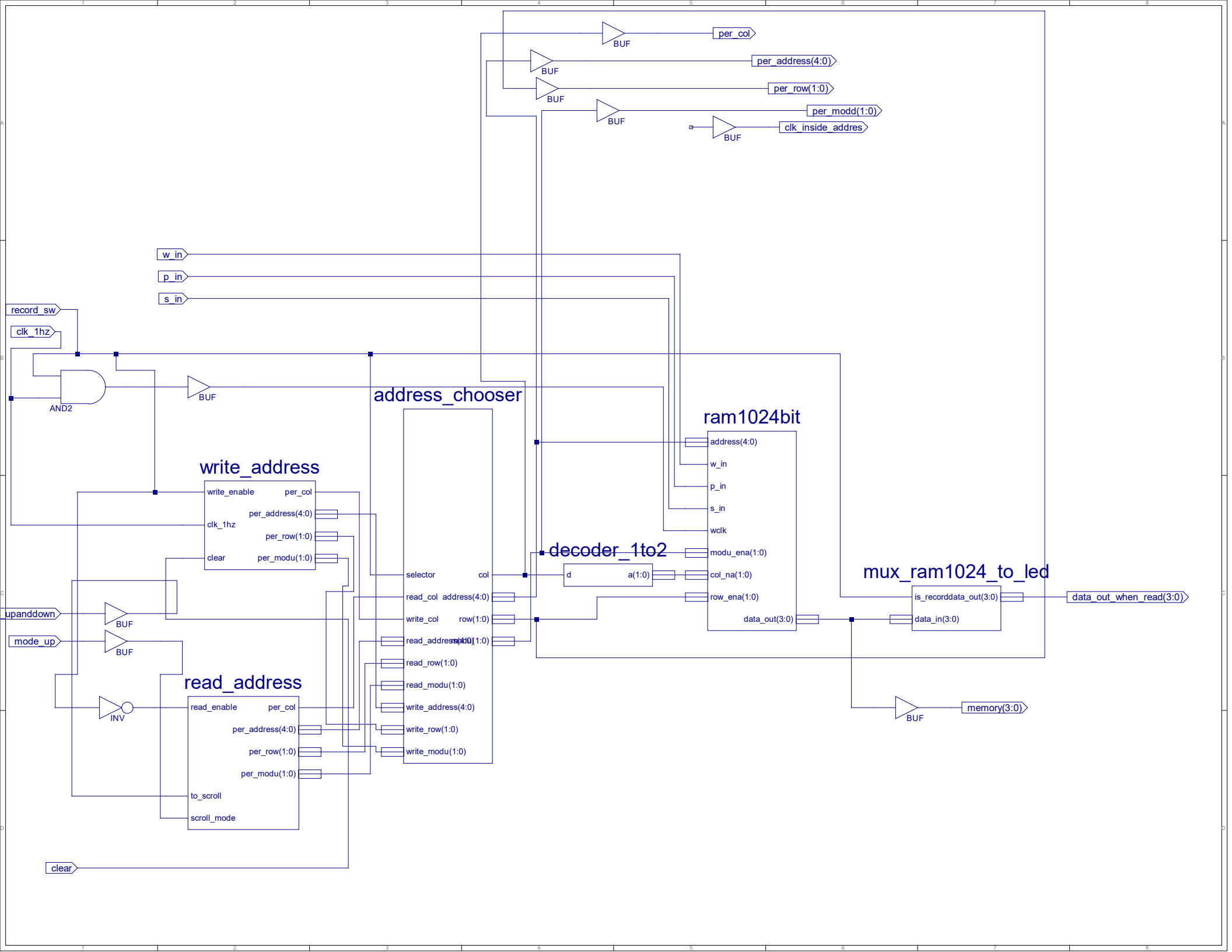
ram1024 เรียก ——— mode (00, 01, 10, 11) ———

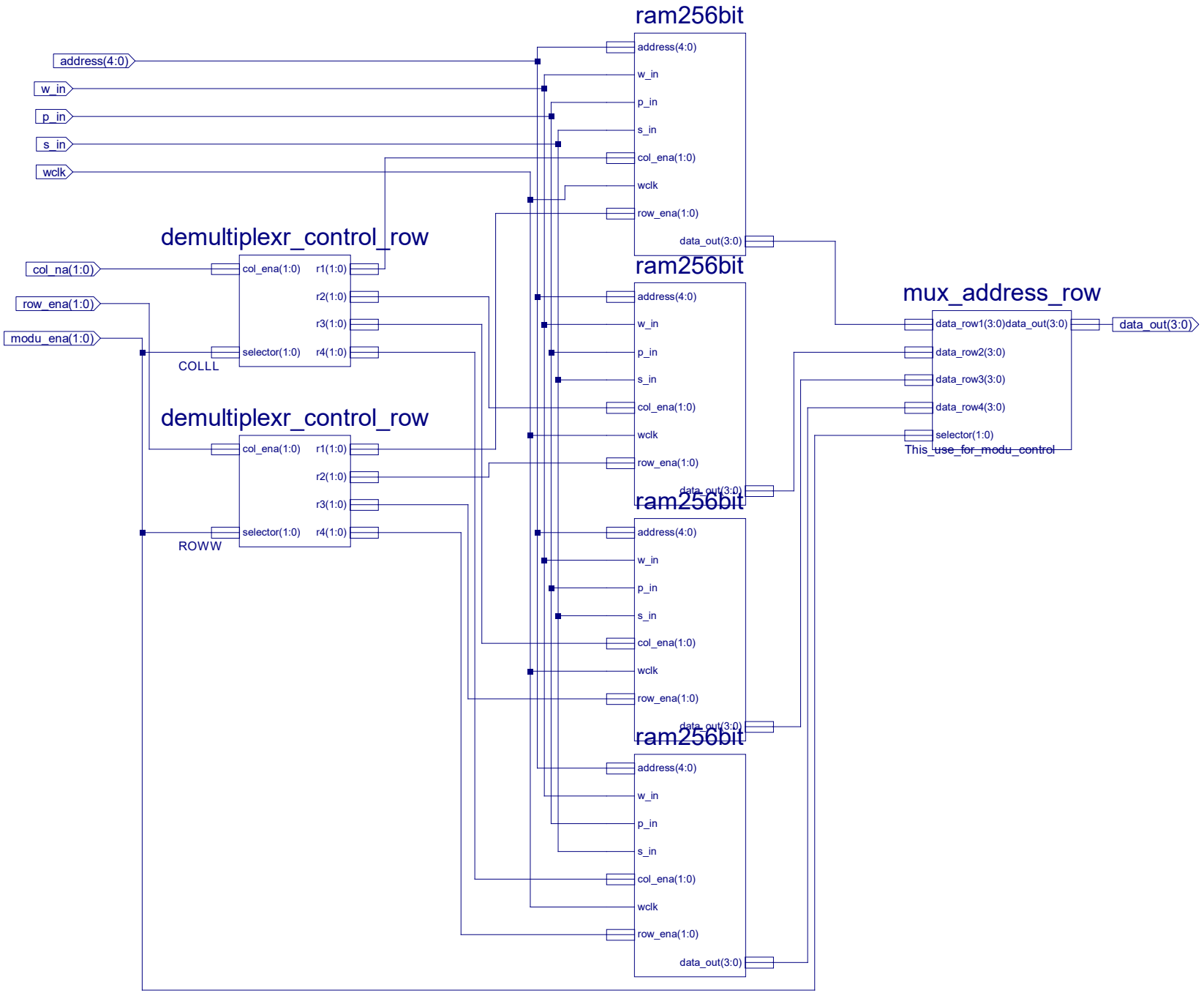

address_enable
clk_1hz
clear
record_sw
read_memory
per_row
per_col
per_address
per_mode
clk
pera
read_address
write_addr
now_addr
record sw
read_col
write_col
now
read_row
write_row
read_mode
write_mode
clk

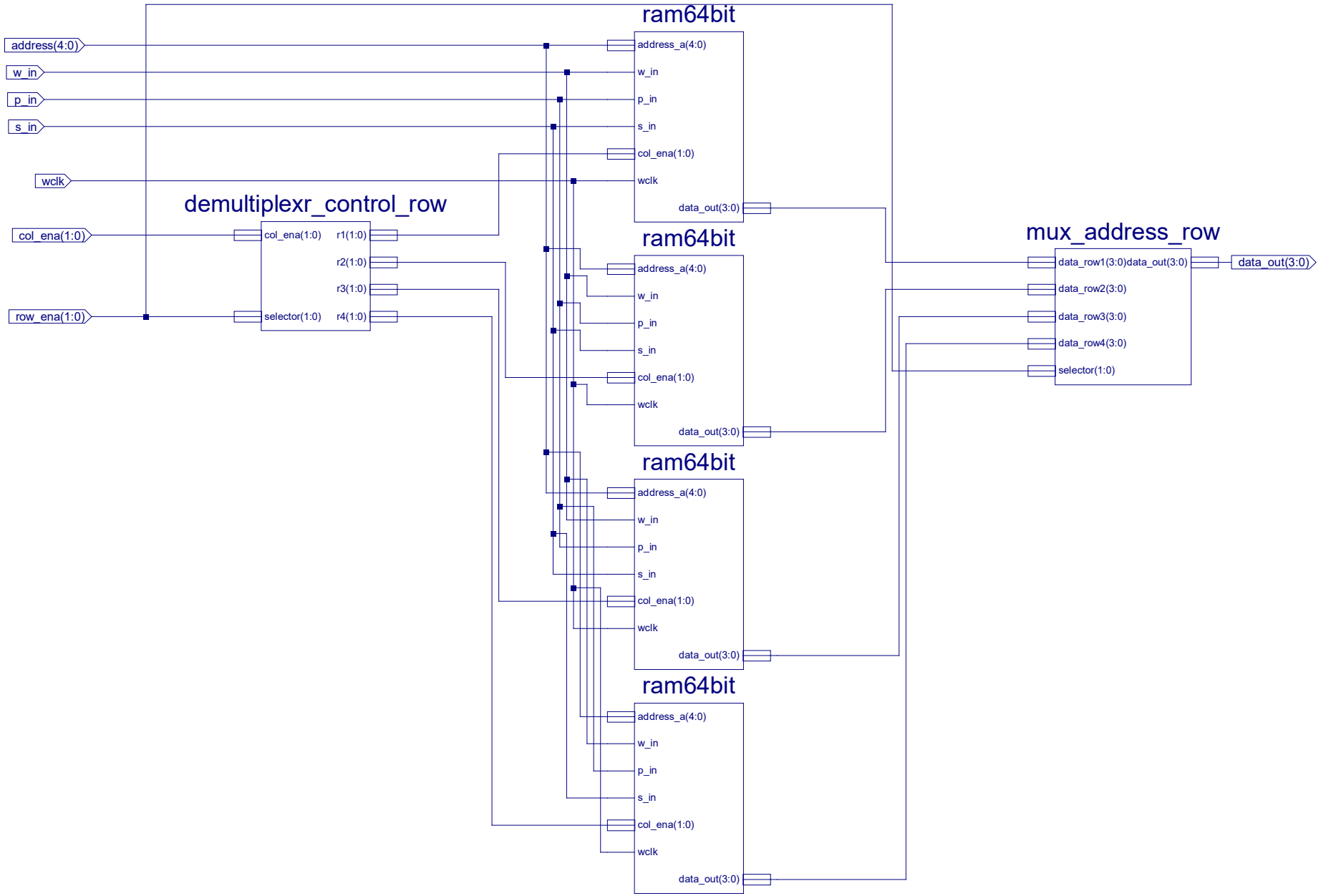

เปลี่ยนเฉย - address
WE = 0
data

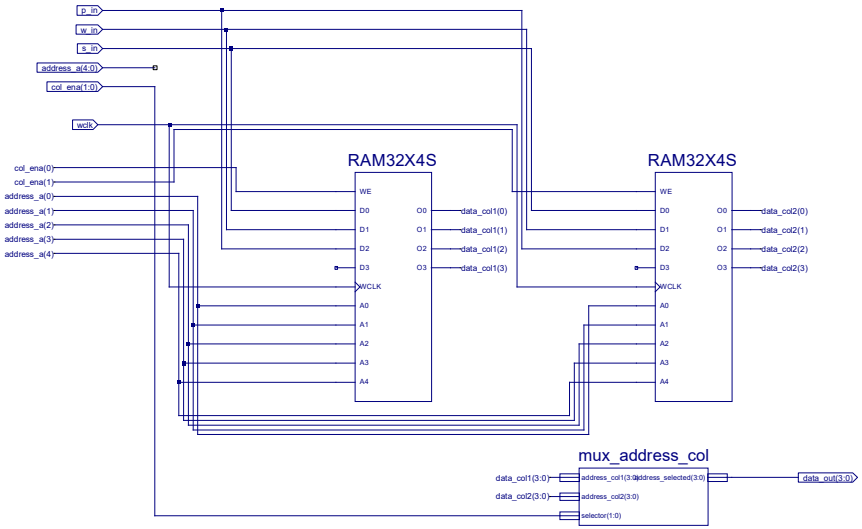

buzzer
GND
clear_people
clk_in
clear_out
present_state(3:0)
previous_state(3:0)
clear_out
BUF
p
w
s
display_board(8:0)
fsm
p_in
led(8:0)
w_in
state_out(3:0)
s_in
clk
prev(3:0)
LED(8:0)
state(3:0)
P_IN
W_IN
ON
CLK
to_1_2_10_100_1000Hz
clockIn
to1hz
to2hz
to10hz
to100hz
to1000hz
clk_up
one_hz clk_out_to_state_delay
two_hz
state(3:0)
state_delay
clk_in clk_state_delay
state(3:0) counter(3:0)
countDown
count_in(3:0)
count_out(3:0)
state(3:0)
common(3:0)
p_in
w_in
s_in
bcd_to_7segment
bcd(3:0)
display(6:0)
display_fpga(6:0)
com(3:0)
com_fpga(3:0)
address_enable
clk_1hz
data_out_when_read(3:0)
led_memory(3:0)
w_in
per_col
XLXN_175
p_in
per_address(4:0)
XLXN_176(4:0)
s_in
per_row(1:0)
XLXN_177(1:0)
to_scroll
scroll_upanddown
per_modd(1:0)
XLXN_178(1:0)
scroll_mode
mode_up
clk_inside_addres
XLXN_179
record_sw
record_sw
clear
memory(3:0)
mem(3:0)


CD4CE
CD4CE
CD4CE
CD4CE
VCC
clockIn
Q0
Q1
Q2
Q3
CE
CEO
C
TC
CLR
INV
CD4CE
CD4CE
CD4CE
FJKC
J
K
Q
C
CLR
to1hz
BUF
to2hz
to10hz
to100hz
to1000hz
FJKC
FJKC
FJKC


per_col
per_address(4:0)
per_row(1:0)
per_modd(1:0)
clk_inside_addres
BUF
w_in
p_in
s_in
record_sw
clk_1hz
AND2
address_chooser
ram1024bit
write_address
write_enable
per_col
per_address(4:0)
clk_1hz
per_row(1:0)
clear
per_modu(1:0)
selector
col
read_col
address(4:0)
write_col
row(1:0)
read_address
read_row(1:0)
read_modu(1:0)
write_address(4:0)
write_row(1:0)
write_modu(1:0)
address(4:0)
w_in
p_in
s_in
wclk
modu_ena(1:0)
col_na(1:0)
row_ena(1:0)
data_out(3:0)
decoder_1to2
d
a(1:0)
mux_ram1024_to_led
is_recorddata_out(3:0)
data_in(3:0)
data_out_when_read(3:0)
upanddown
mode_up
INV
read_address
read_enable
per_col
per_address(4:0)
per_row(1:0)
per_modu(1:0)
to_scroll
scroll_mode
clear
memory(3:0)


ram256bit
address(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
row_ena(1:0)
data_out(3:0)
address(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
row_ena(1:0)
data_out(3:0)
address(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
row_ena(1:0)
data_out(3:0)
address(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
row_ena(1:0)
data_out(3:0)
address(4:0)
w_in
p_in
s_in
wclk
demultiplexr_control_row
col_ena(1:0)
r1(1:0)
r2(1:0)
r3(1:0)
selector(1:0)
r4(1:0)
COLLL
demultiplexr_control_row
col_ena(1:0)
r1(1:0)
r2(1:0)
r3(1:0)
selector(1:0)
r4(1:0)
ROWW
mux_address_row
data_row1(3:0)data_out(3:0)
data_row2(3:0)
data_row3(3:0)
data_row4(3:0)
selector(1:0)
This use for modu control
data_out(3:0)
col_na(1:0)
row_ena(1:0)
modu_ena(1:0)


address(4:0)
w_in
p_in
s_in
wclk
col_ena(1:0)
row_ena(1:0)
demultiplexr_control_row
col_ena(1:0)
selector(1:0)
r1(1:0)
r2(1:0)
r3(1:0)
r4(1:0)
ram64bit
address_a(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
data_out(3:0)
ram64bit
address_a(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
data_out(3:0)
ram64bit
address_a(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
data_out(3:0)
ram64bit
address_a(4:0)
w_in
p_in
s_in
col_ena(1:0)
wclk
data_out(3:0)
mux_address_row
data_row1(3:0)data_out(3:0)
data_row2(3:0)
data_row3(3:0)
data_row4(3:0)
selector(1:0)
data_out(3:0)


p_in
w_in
s_in
address_a(4:0)
col_ena(1:0)
wclk
col_ena(0)
col_ena(1)
address_a(0)
address_a(1)
address_a(2)
address_a(3)
address_a(4)
RAM32X4S
WE
D0
D1
D2
D3
WCLK
A0
A1
A2
A3
A4
O0
O1
O2
O3
data_col1(0)
data_col1(1)
data_col1(2)
data_col1(3)
RAM32X4S
WE
D0
D1
D2
D3
WCLK
A0
A1
A2
A3
A4
O0
O1
O2
O3
data_col2(0)
data_col2(1)
data_col2(2)
data_col2(3)
mux_address_col
data_col1(3:0)
data_col2(3:0)
address_col1(3:0)
address_col2(3:0)
selector(1:0)
address_selected(3:0)
data_out(3:0)