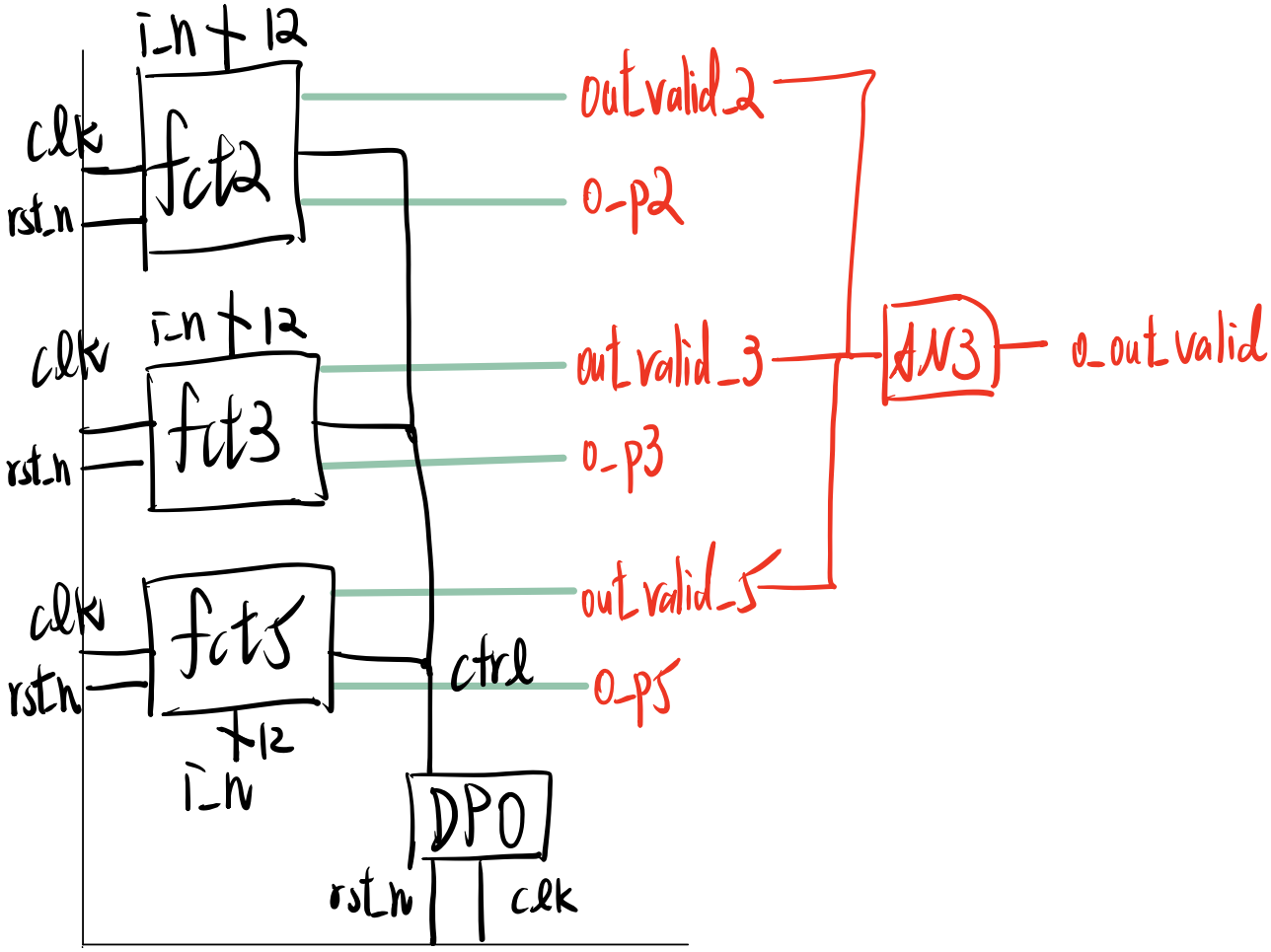
積體電路設計**hw4 Report**
B09901081 施伯儒

a.Simulation

use RECURSIVE and minimum clock cycle = 7.8ns

```
================================================
                  Summary
================================================
        Clock cycle:           7.8 ns
        Number of transistors: 3710
        Total excution cycle:  2378
        Correctness Score:     40.0
        Performance Score:     68814564.0
================================================
```

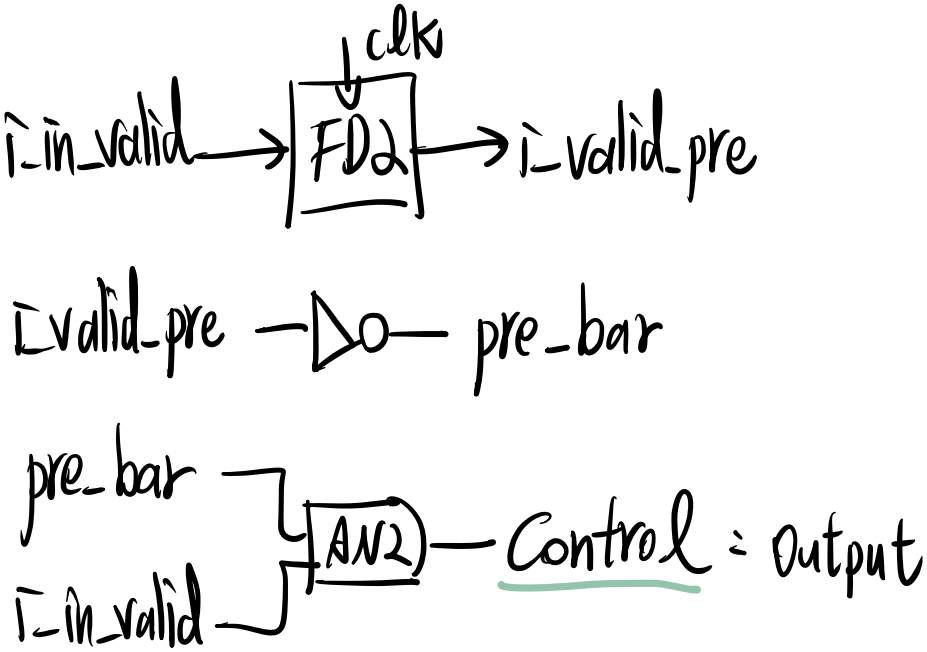
factor =



Critical Path 在 fct5 內部 (詳見 fct5)

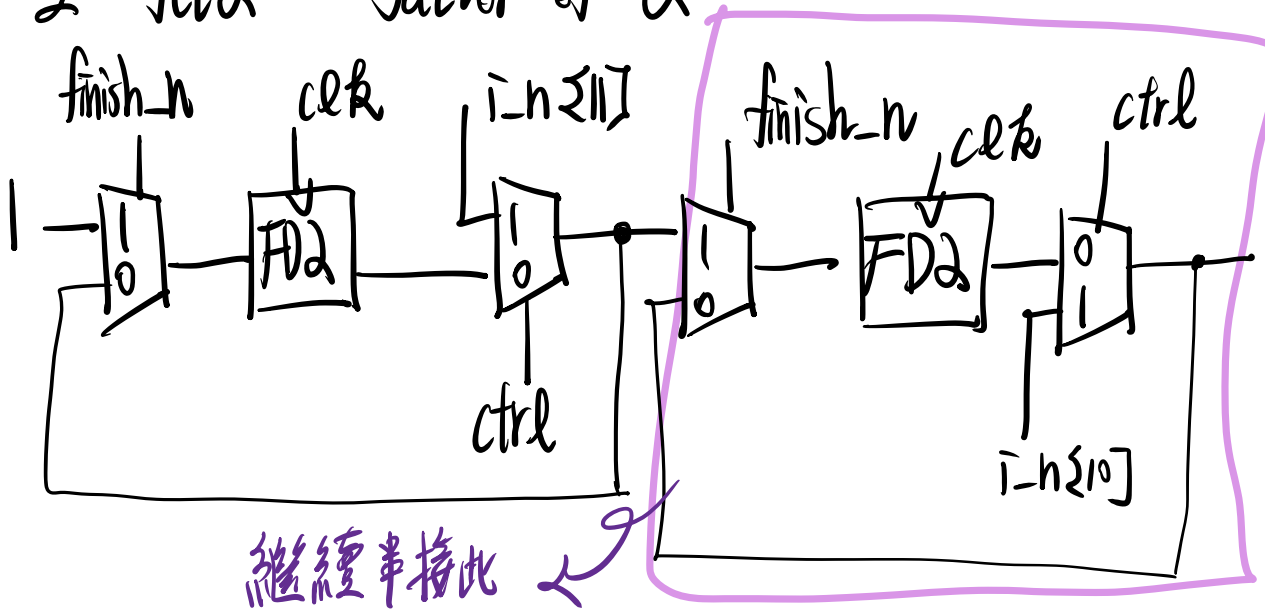
1° DPO：Detect_Posedge

功用：侦测 i_in_valid 的 posedge 来了没

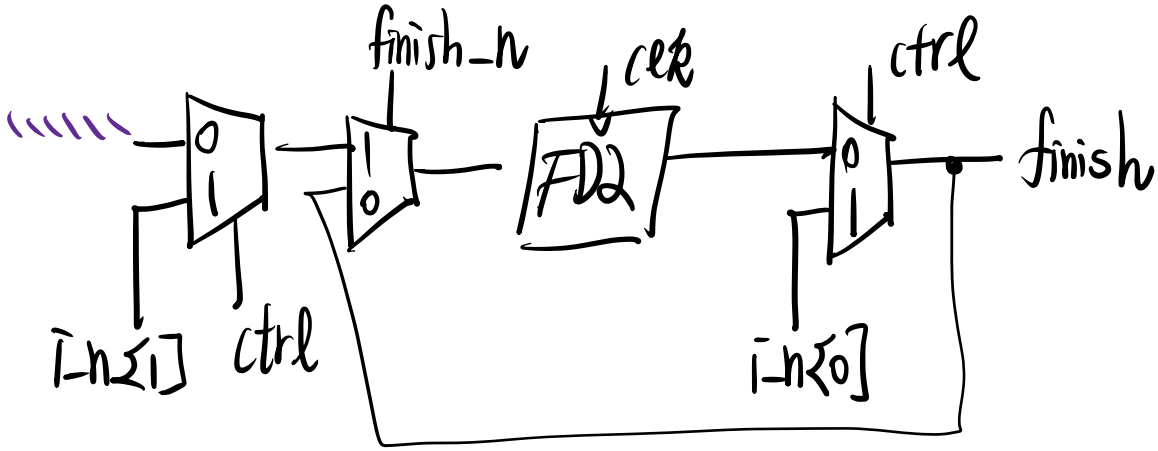


若 pre_bar=1 且 valid 为 1 表示侦测到

2° fd2 = factor of 2

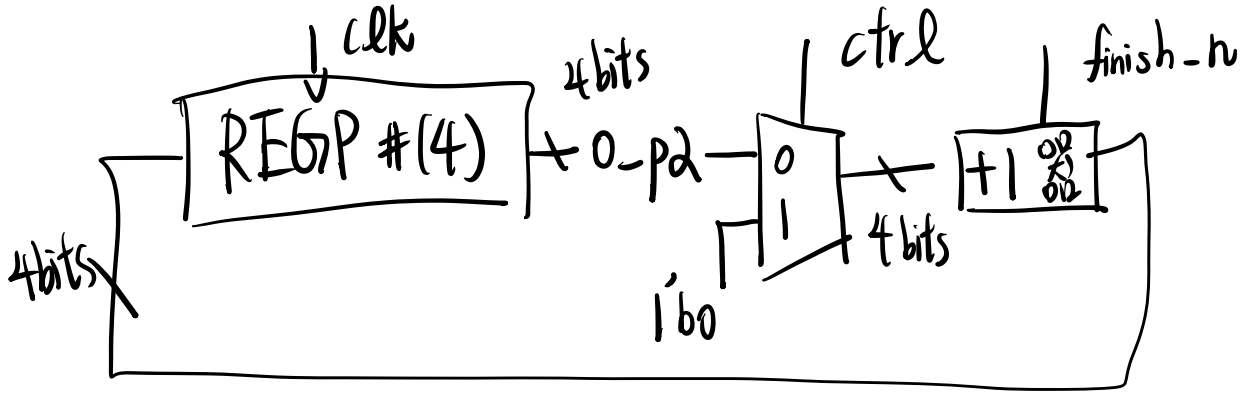


繼續串接此 Block，但是 i_n[ ] 的 i 逐個遞減

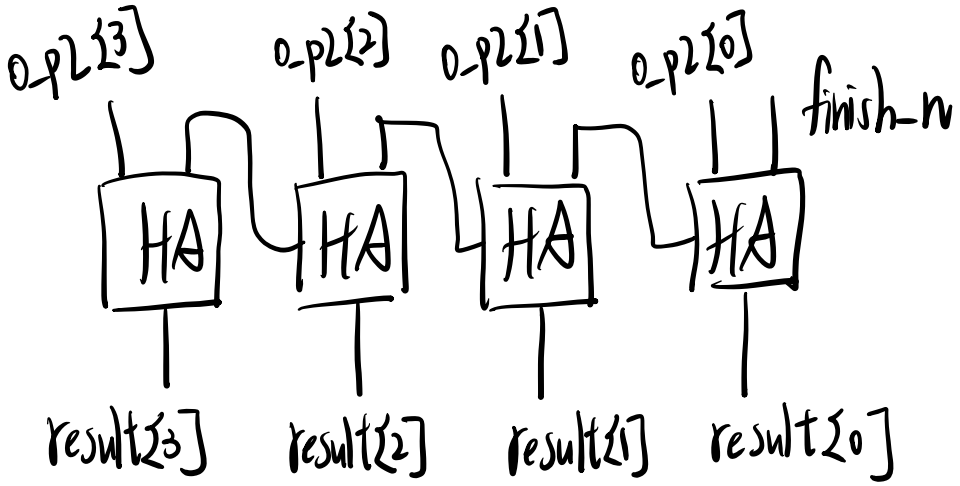


finish —▷o— finish_n

finish — FD2 — o_out_valid
clk



+1器 = 4bit 加1器



原理：就第0位開始數過到幾個0
才過到1，得到的數字就是 o_p2

實作：用 12 個 FD2，當 ctrl=1 時 (i_in_valid posedge)
就會 load 一次。

等clk來時，會把上個FD2的值load進來

如此可以逐一看過所有bits in i_n

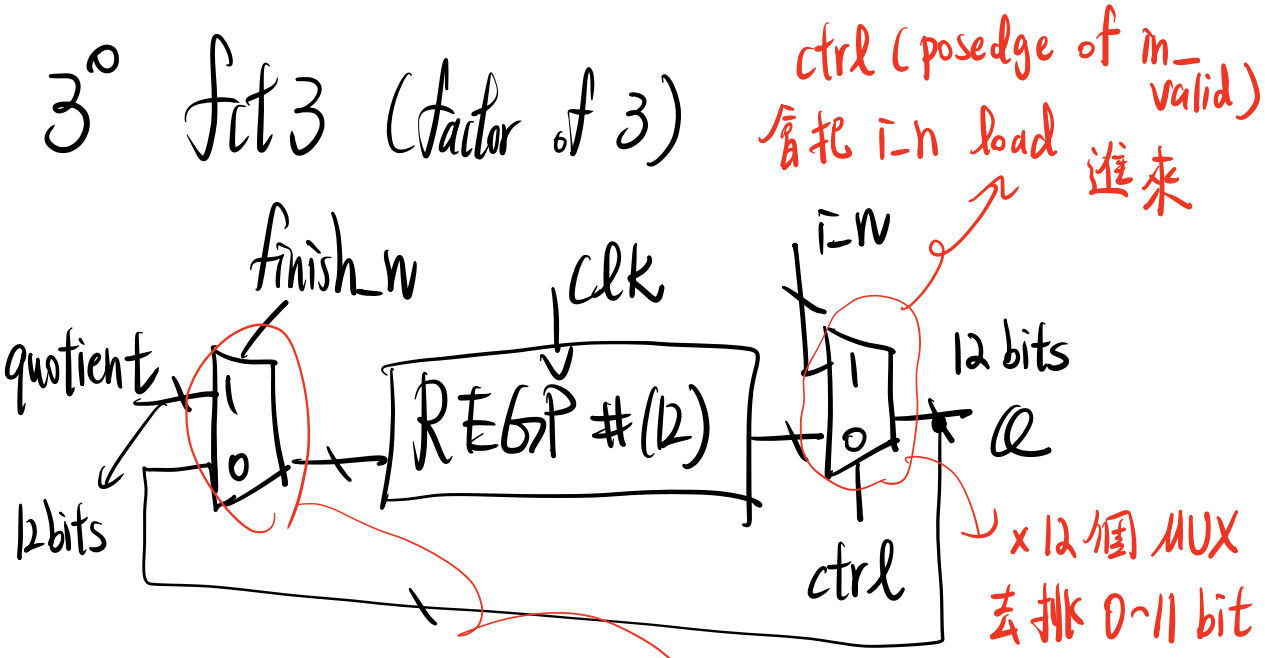
而第0個FD2的Q即是當下看到的值

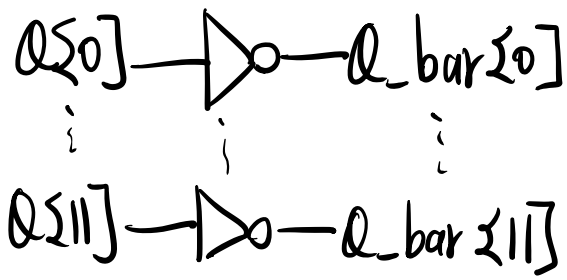
若為0，可被2整除，finish=0

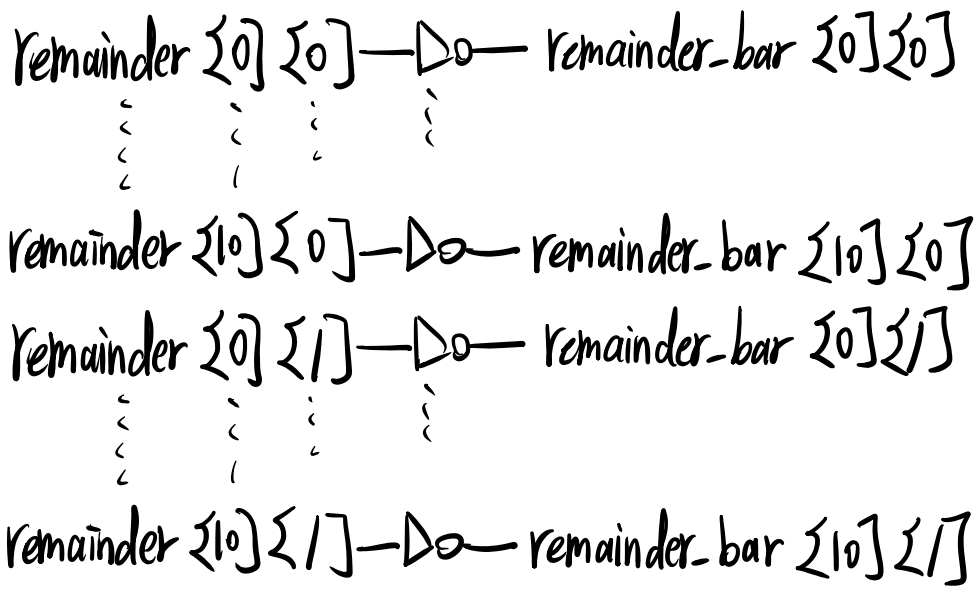
/ 不" " " =1

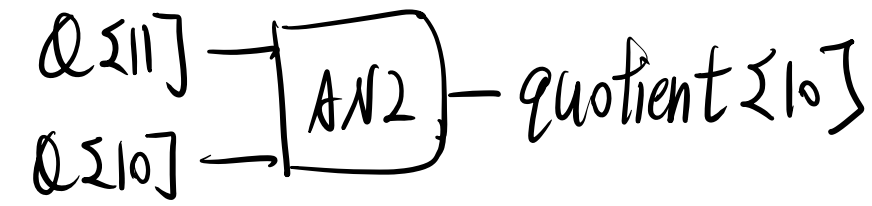
再經過D0得到finish_n加到o_p2上

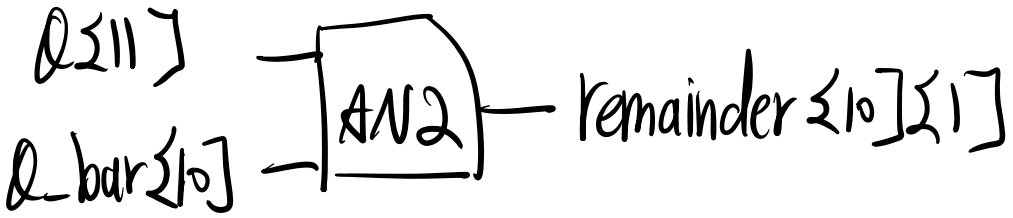
3° fct3 (factor of 3)

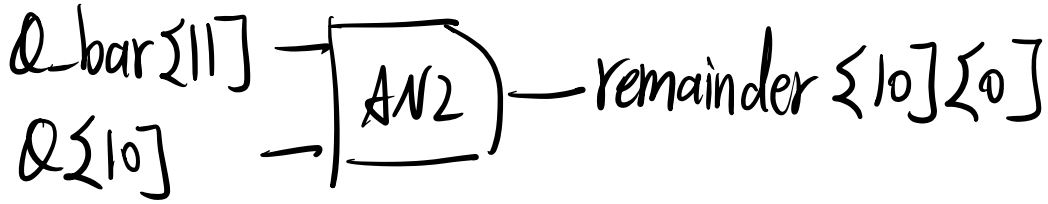


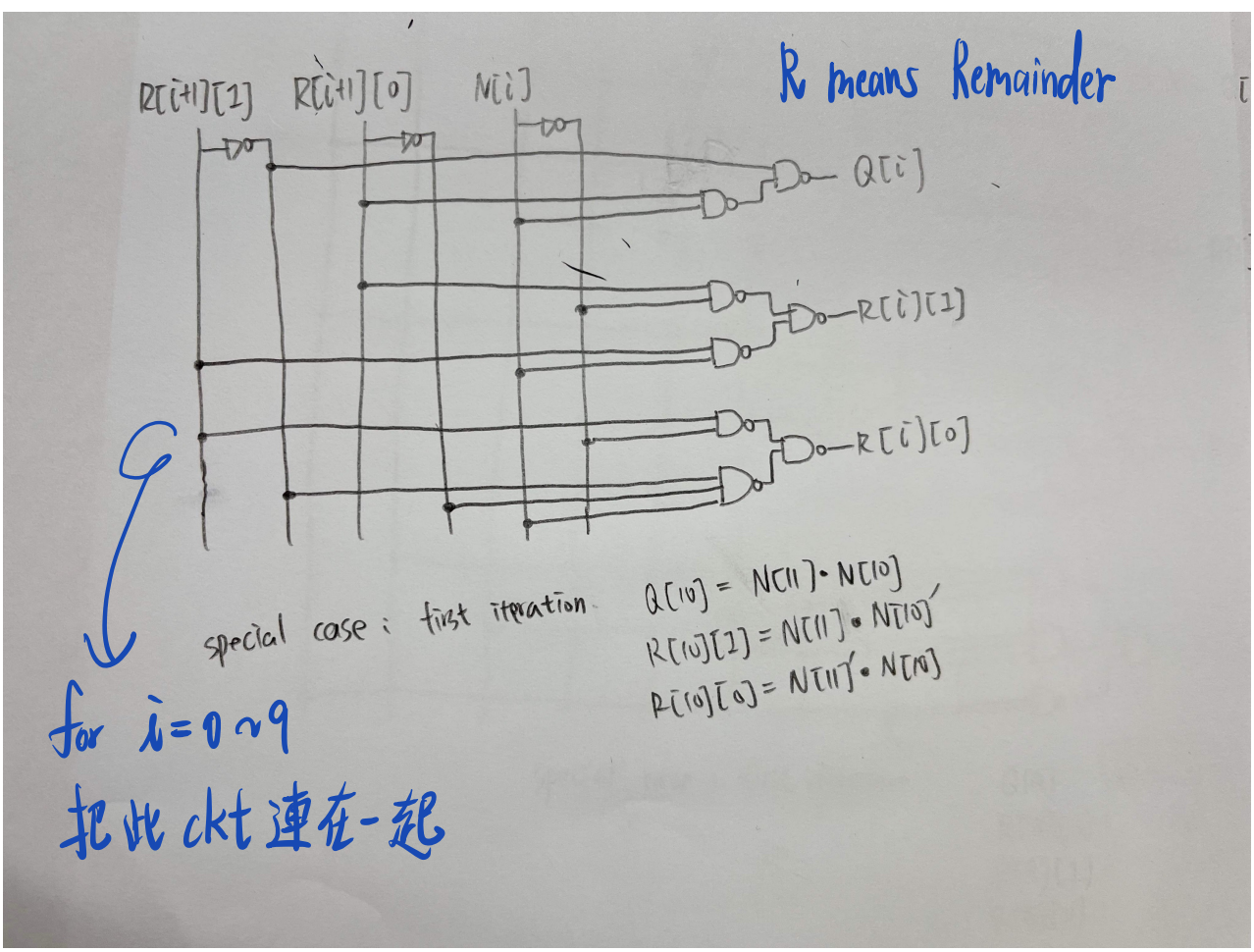


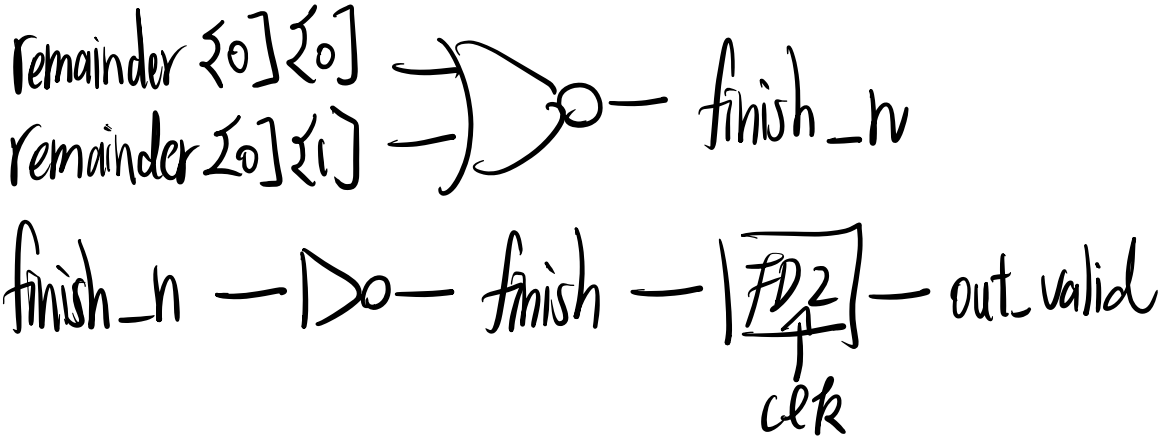


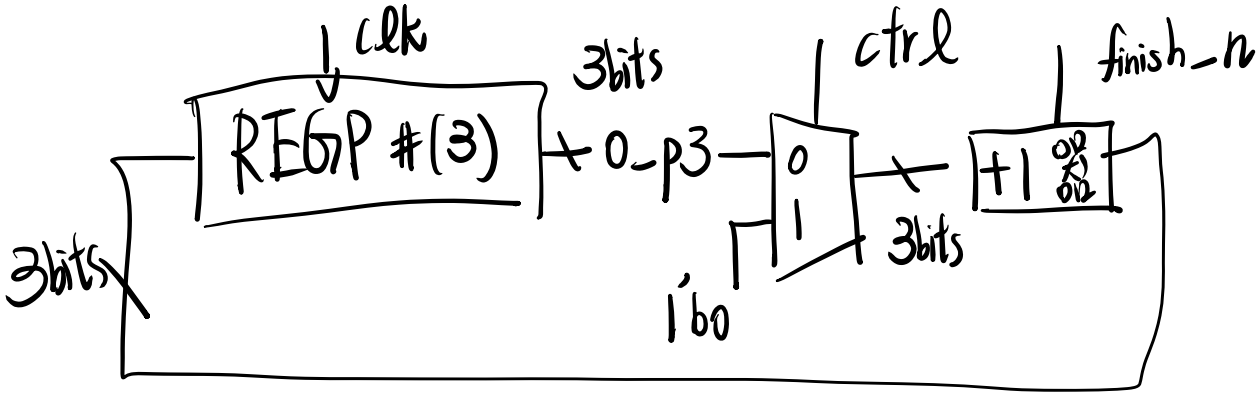

Q[11]
Q[10]
AN2
quotient[10]


Q[11]
Q_bar[10]
AN2
remainder[10][1]

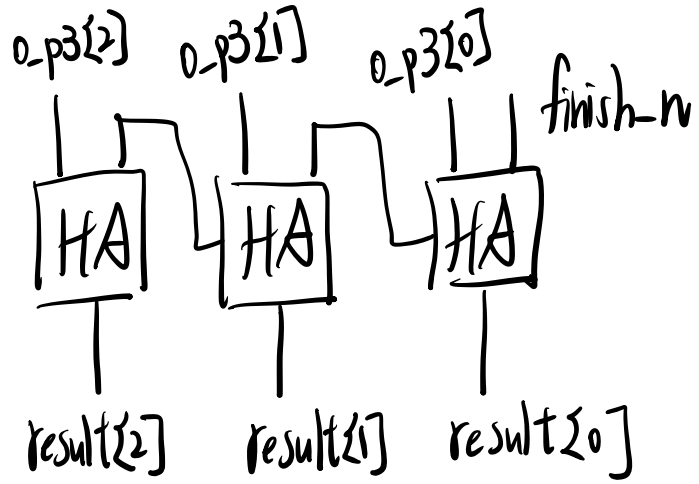

Q_bar[11]
Q[10]
AN2
remainder[10][0]

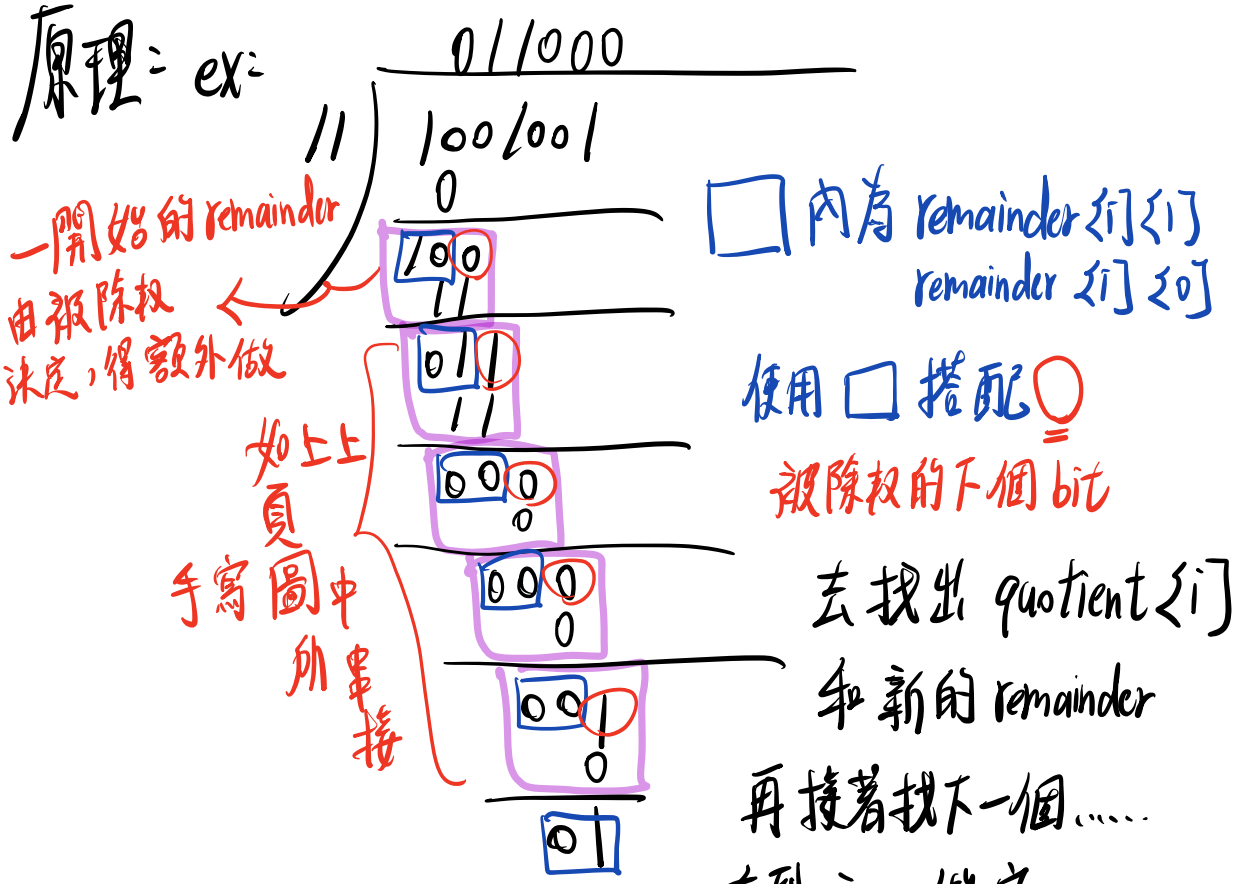

R means Remainder
R[i+1][1]
R[i+1][0]
N[i]
Q[i]
R[i][1]
R[i][0]
special case : first iteration
Q[10] = N[11] · N[10]
R[10][1] = N[11] · N[10]'
R[10][0] = N[11]' · N[10]
for i=0~9
把此 ckt 連在一起





+1 연산 = 3bit adder 연산



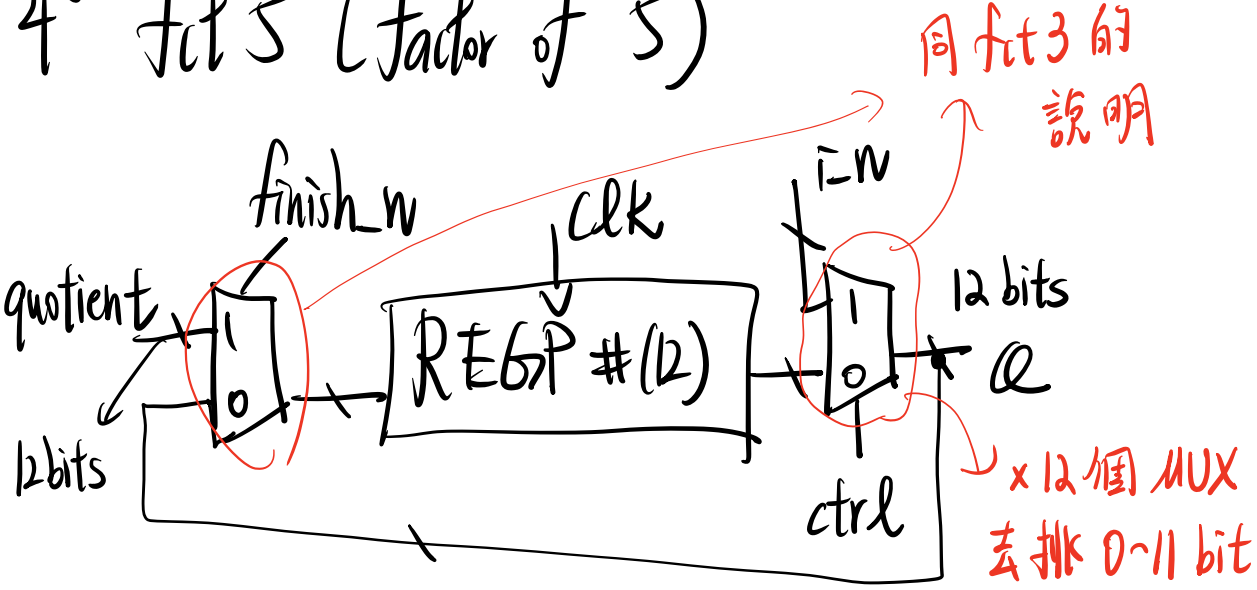


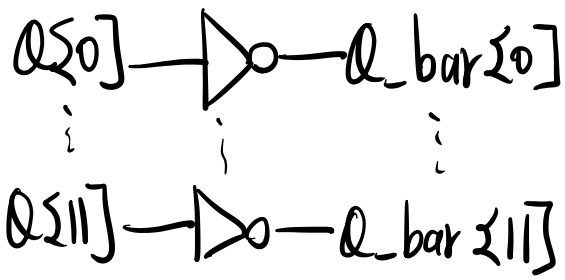
求出的 remainder (i)(0) 若為 0 則 quotient load 回去
remainder (i)(1) 0

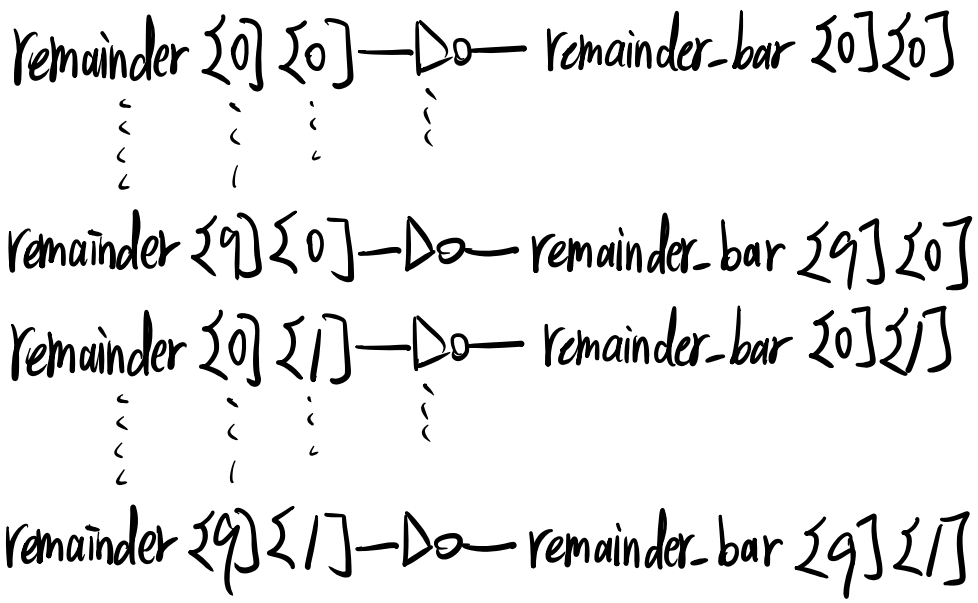
繼續除，並把 0_p3 加1，反之則結束

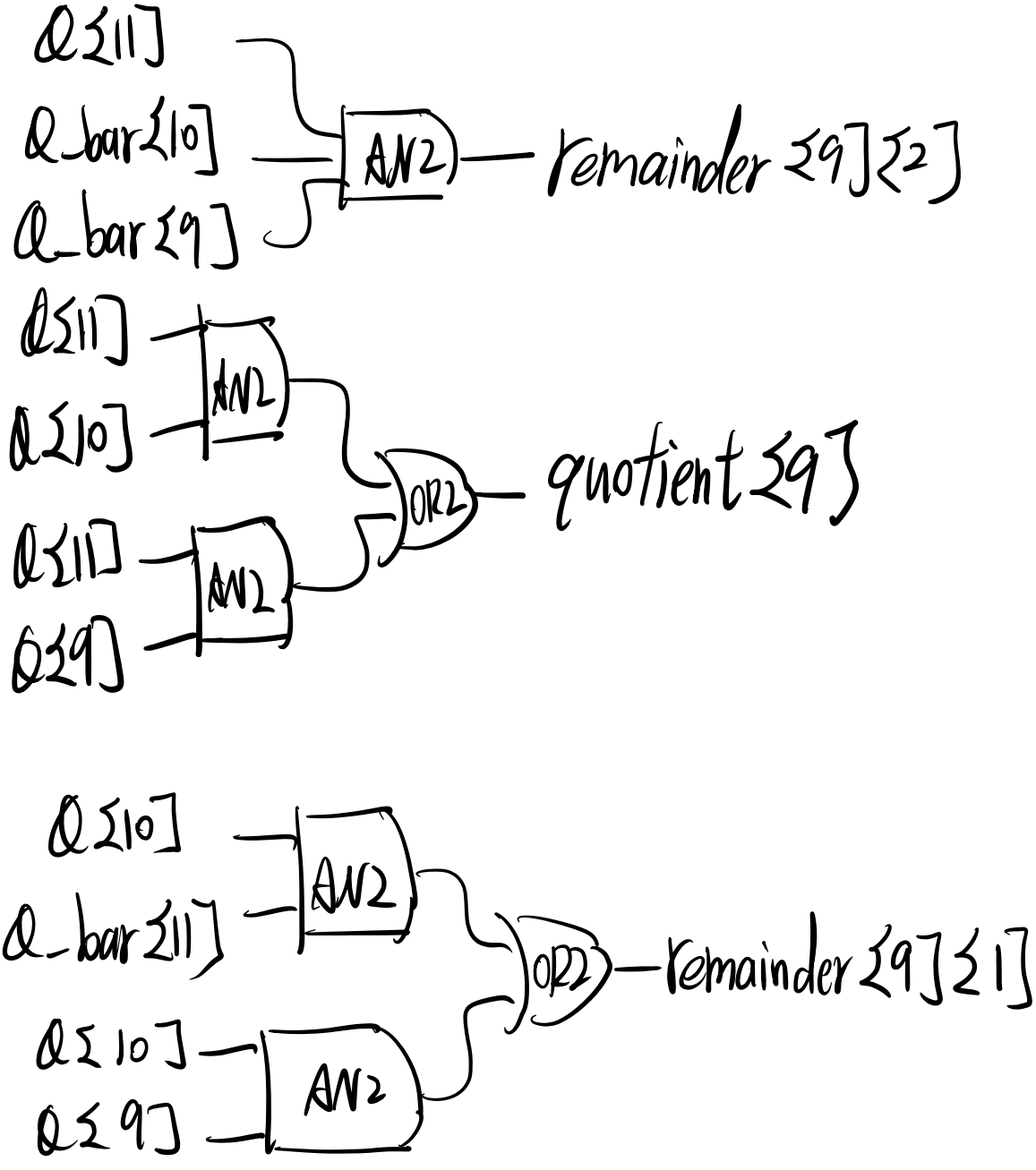
內使用LUT去求出下一個remainder行行，2列
和quotient行

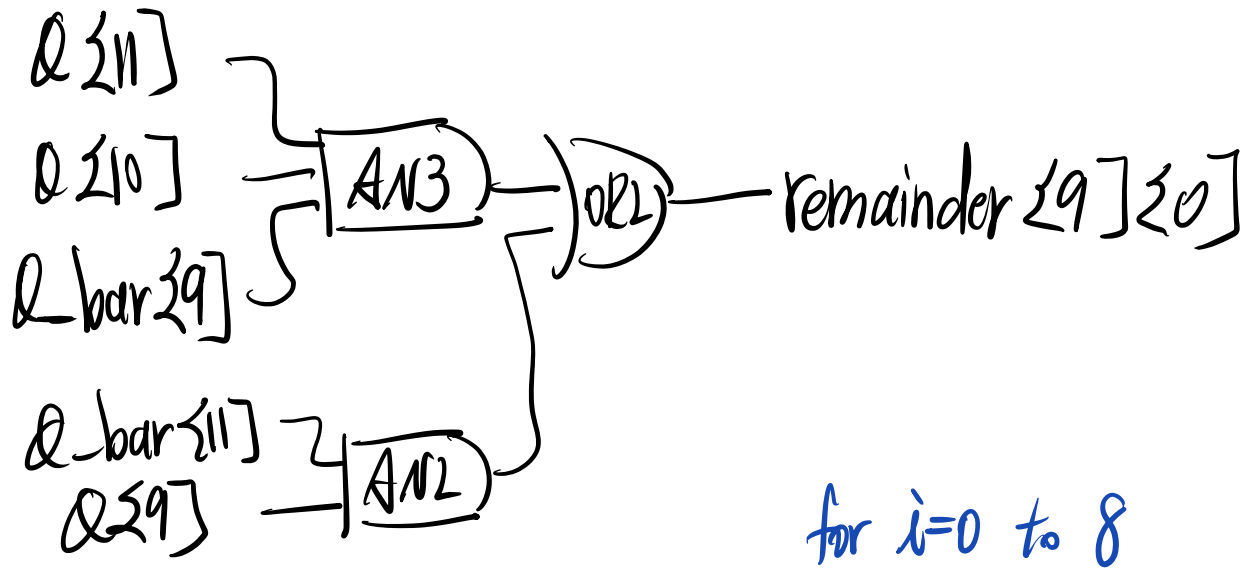
4° fct5 (factor of 5)





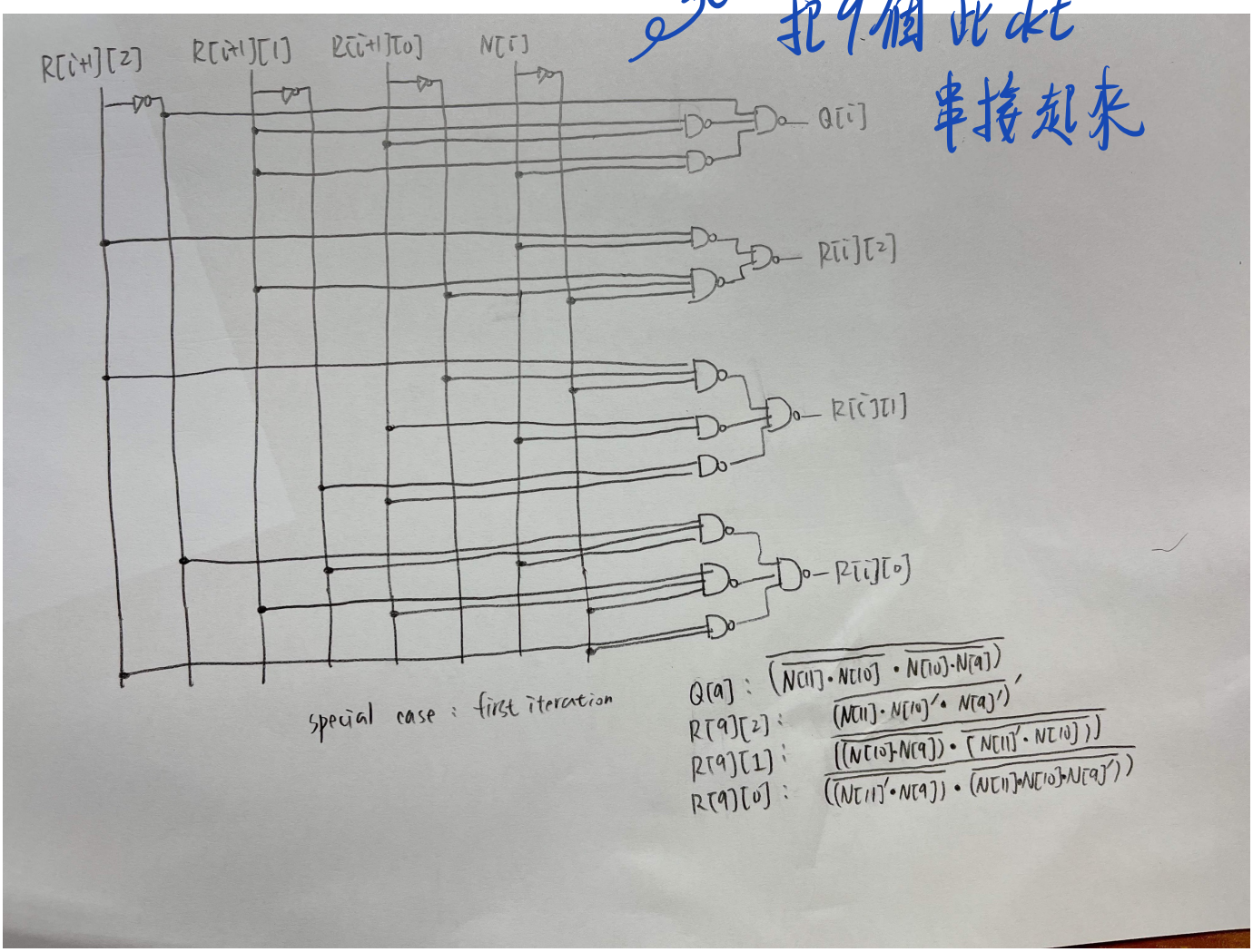


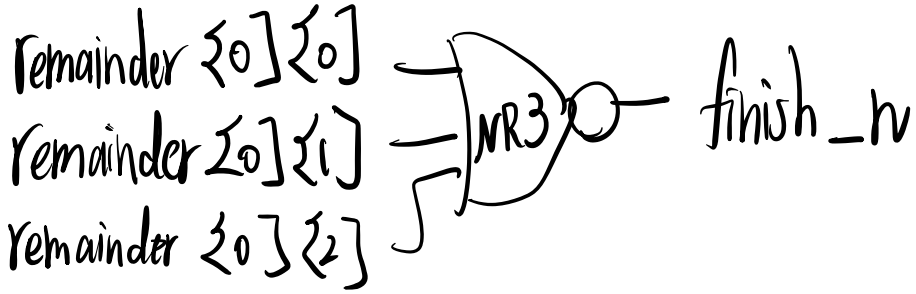

Q[11]
Q_bar[10]
Q_bar[9]
AN2
remainder[9][2]
Q[11]
Q[10]
AN2
Q[11]
Q[9]
AN2
OR2
quotient[9]
Q[10]
Q_bar[11]
AN2
Q[10]
Q[9]
AN2
OR2
remainder[9][1]



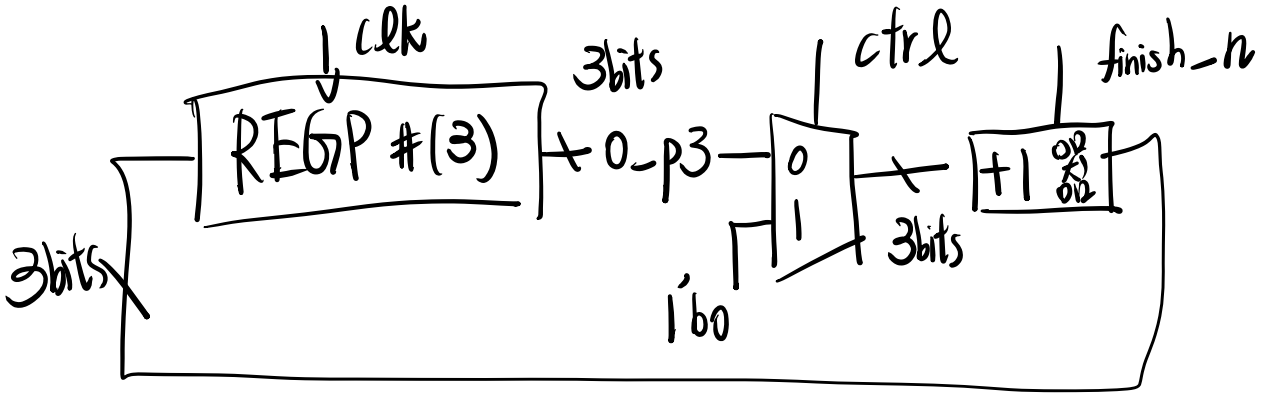
for i=0 to 8

把9個此ckt 串接起來

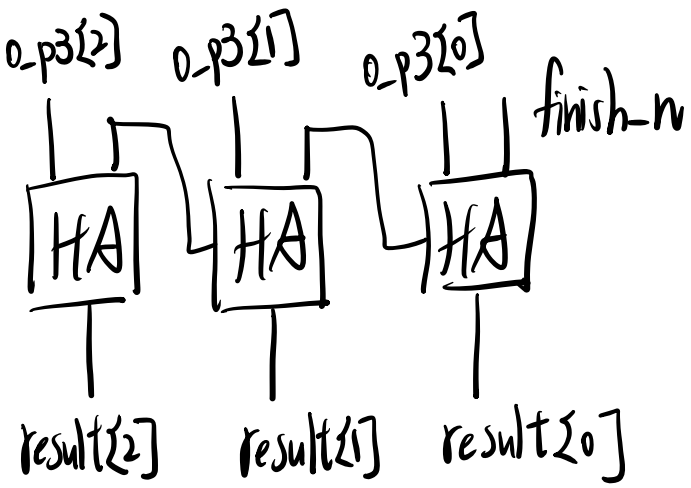



remainder[0][0]
remainder[0][1]
remainder[0][2]
NR3
finish_n


finish_n
finish
FD2
out_valid
clk


clk
3bits
REGP #(3)
o_p3
ctrl
finish_n
0
1
+1
3bits
3bits
1'b0

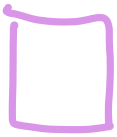
+1 = 3bit adder


o_p3[2]
o_p3[1]
o_p3[0]
finish_n
HA
HA
HA
result[2]
result[1]
result[0]

原理：同 factor of 3，只是除以 101

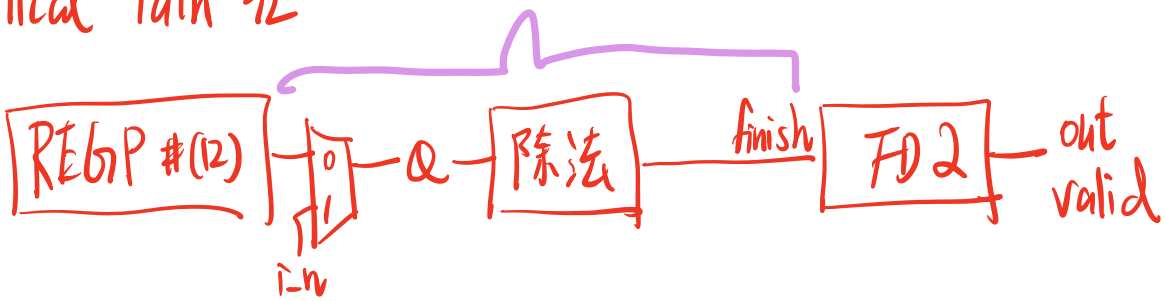
有一些地方不一樣

① remainder 2'b 有 (0, 1, 2) 3個 bits

②  內 LUT 的部分變成用 remainder 2'b 的 3 bits 和 被除數 2'b 去 找出 quotient 2'b 和下個 remainder 2'b 2'b, 2'b, 2'b

⇒ 化簡後可得上上頁的手寫圖中的電路

Critical Path 在



其餘部份和 ÷3 雷同

故不重複

c.Discussion

1.Introduce your design

⇒　在b.部分已附上

2.How do you cut your pipelined or recursive design?

⇒　我使用recursive，並未使用pipelined register除了在output得依規定buffer以外，其他的額外register會增加cycles，所以不用

3.How do you improve your critical path and the number of transistors?

⇒　使用較快的gate，例如:NAND，或者是除法器設計上使用部分的LUT(用交電教的化簡)，而非是用原汁原味的除法器，有特別為除三除五去設計

4.How do you trade-off between area and speed?

⇒　用萬用的除法器去跑，雖然可以等除三做完再做除五，面積可以少一半，但同時也耗很多cycles。因此我使用兩個除法器同時除三和除五，花比較多area，但快很多，performance也更好

5.Compare with other architectures you have designed (if any).

⇒ (discussion with B09901178)

64 / 101

Division by Constants (2/2)

- $\frac{1}{d} = \frac{m}{2^n - 1} = \frac{m}{2^n(1-2^{-n})} = \frac{m}{2^n}(1+2^{-n})(1+2^{-2n})(1+2^{-4n})\cdots$
- For example, for 24-bit precision:

$d = 5, \Rightarrow m = 3, n = 4$

Easy for hardware implementation

$$\frac{z}{5} = \frac{3z}{2^4 - 1} = \frac{3z}{16(1-2^{-4})} = \frac{3z}{16}(1+2^{-4})(1+2^{-8})(1+2^{-16})$$

Next term ($1+2^{-32}$) does not contribute anything to 24-bit precision

我實作的另一個架構是這個division by constants的架構，缺點是面積很大且會用到好幾個adders串接，導致cycle也很長，反而最直觀的除法器表現很好:)

附上很爛的performance:

```
==========================================
                Summary
==========================================
       Clock cycle:           10.2 ns
       Number of transistors: 6143
       Total excution cycle:  2378
       Correctness Score:     40.0
       Performance Score:     149002150.8
==========================================
```