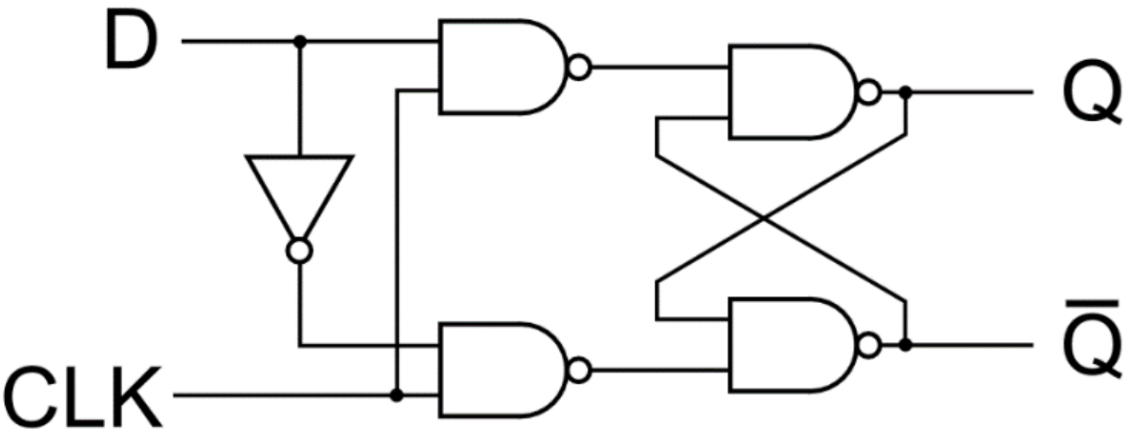
Lab 3 Report

Course	Logic Design Lab
Done	☐
Last edited time	@Oct 22, 2020 8:01 PM
URL	

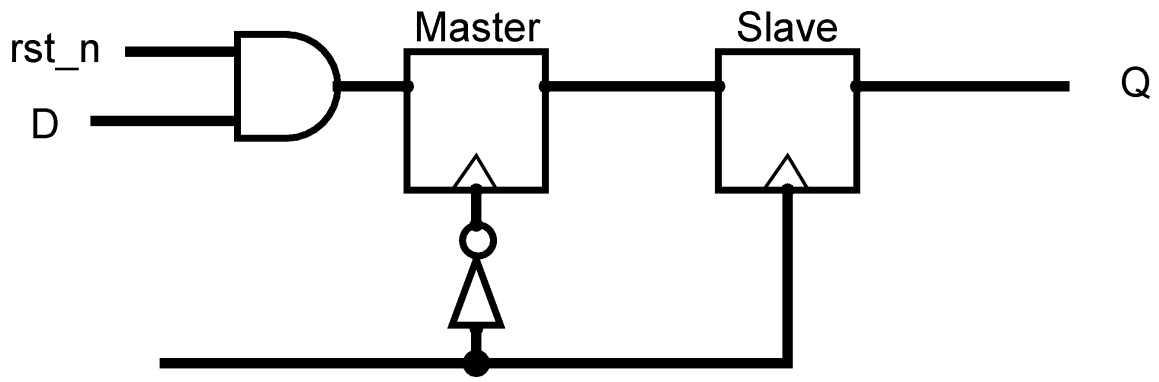
Advanced Question 1

DFF

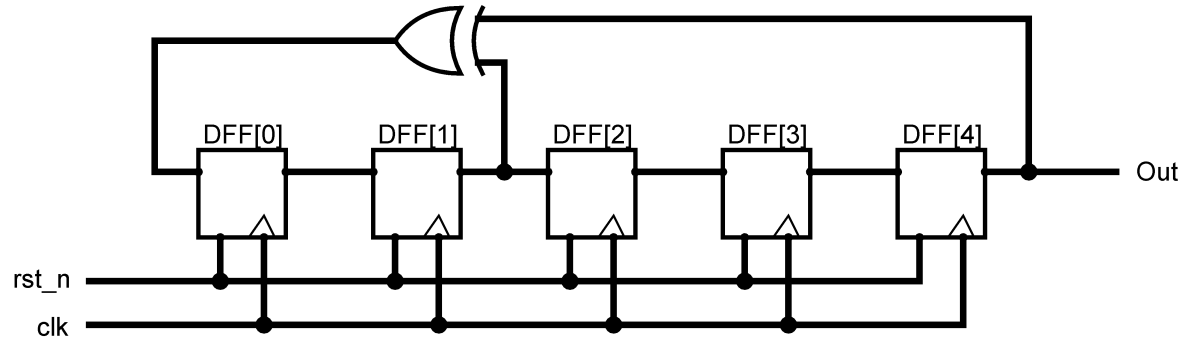
為了做出DFF，我們要先刻出D-Latch



然後按照Lab 2 Basic的做法，刻出DFF，不過在輸入前多加上一個偵測rst_n的部份，讓 `rst_n == 0` 時可以輸出0



Q1 circuit.



Testbench

這次的testbench沒有什麼重點，製造出一個 `clk`，然後確定開始前有先reset一次就可以了。剩下可以藉由觀察waveform來確定結果正確。

（由於輸出只有 `DFF[4]` ，因此觀察waveform是比較快的方法。）

```
always #5 clk = ~clk;

initial begin
    rst_n = 0;
    #10
    rst_n = 1;

    repeat (2 ** 5) begin
        #5
        ;
    end
end
```

State Transition

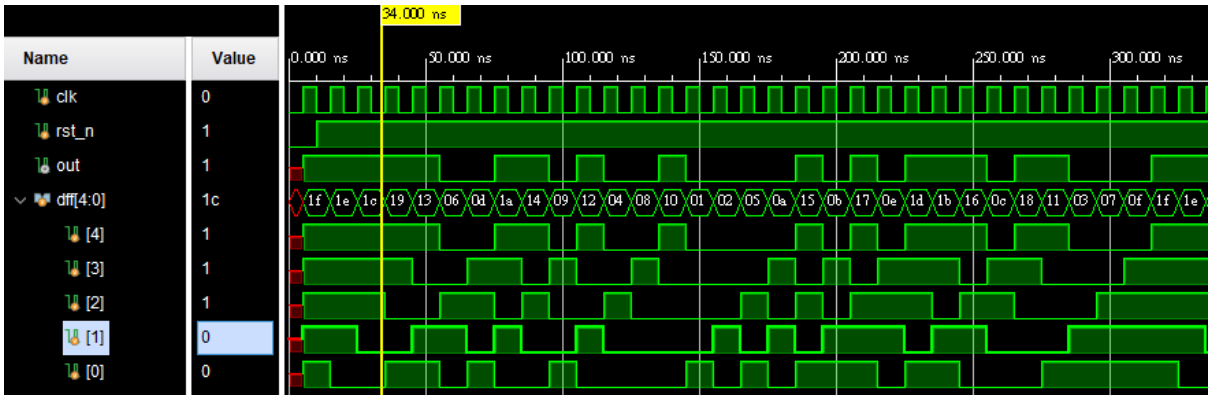
他太長了……所以我直接打了份code輸出state transition，因為從頭到尾的state diagram就是一個環狀圖，結果如下：

當 `rst_n == 1` 時，每次trigger會讓他變成下一個state。

當 `rst_n == 0` 時， `DFF[4:0] == 5'b11111` 。

```
11111 -> 01111 -> 00111 -> 10011 -> 11001 ->
01100 -> 10110 -> 01011 -> 00101 -> 10010 ->
01001 -> 00100 -> 00010 -> 00001 -> 10000 ->
01000 -> 10100 -> 01010 -> 10101 -> 11010 ->
11101 -> 01110 -> 10111 -> 11011 -> 01101 ->
00110 -> 00011 -> 10001 -> 11000 -> 11100 ->
11110 -> 11111
```

Waveform



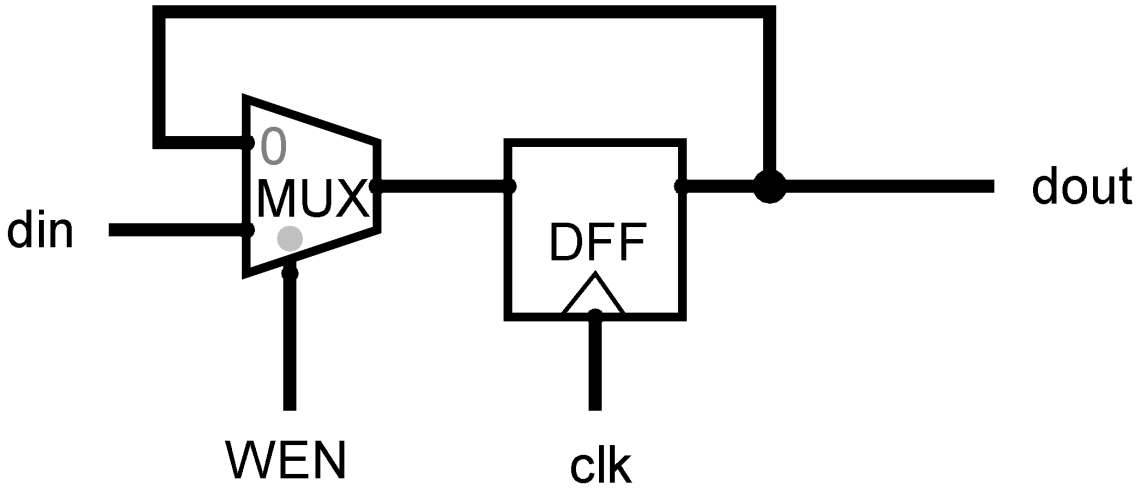
What happens if we reset the DFFs to 5'b00000?

觀察可以發現 `DFF[0](t+1) = (rst_n ? DFF[1](t) ^ DFF[4](t) : 0)` ，因此不論如何，之後的 `DFF[4:0]` 永遠都是 `5'b00000` ， `out` 也會永遠都是0。

Advanced Question 2

Register

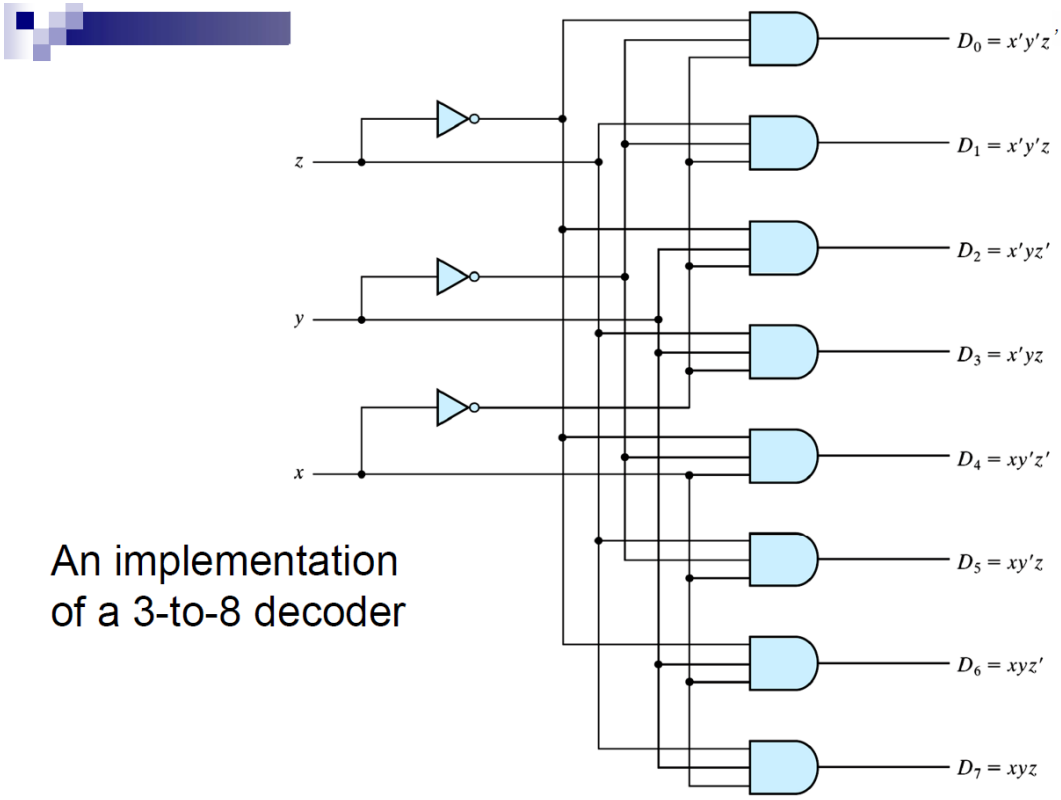
實作memory的部份會用到register，因此我們可以先實作一個可以儲存資料的register

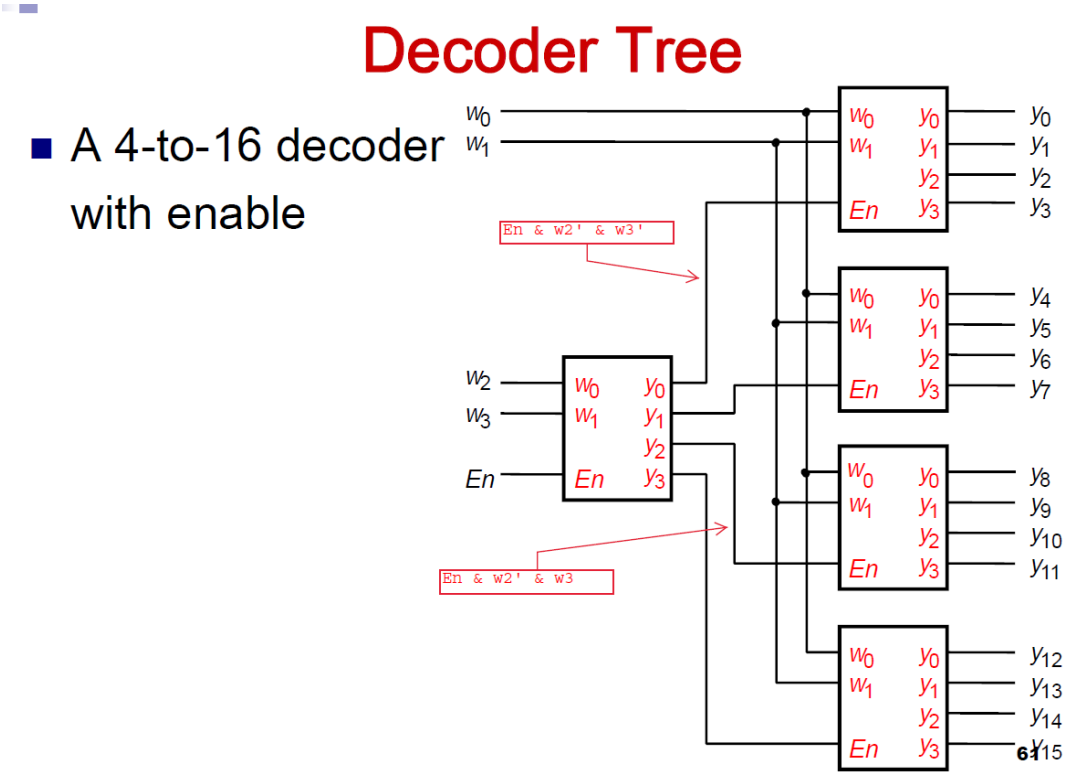


DFF與MUX的部份在Q1與前幾次的Lab都實作過，因此不再重複一遍。做好register後，我們需要將他複製很多~~~份，構造一個 `[7:0] memory [127:0]` 的register陣列。

Decoder

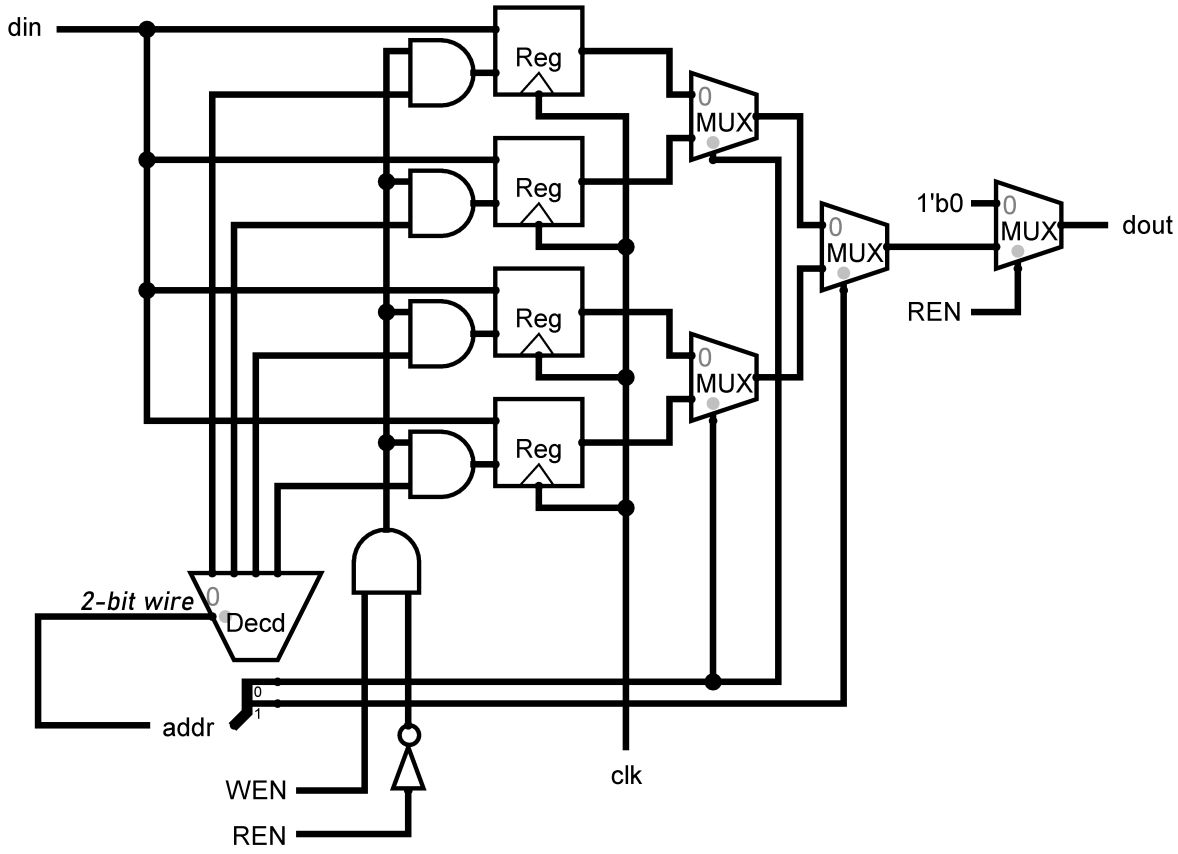
接著實作一個Decoder，等等會使用到（這邊直接擷取上學期Decoder的範例，上圖是列舉所有可能性，下圖則是使用Decoder Tree的方法）





Q2 circuit

接著，使用Decoder與MUX，可以構造出一個可以讀寫至不同位址的memory。



- `din`：資料輸入 (1bit)
- `dout`：資料輸出 (1bit)
- `addr`：memory位址 (2bits)
- `WEN`：寫入
- `REN`：讀取

運作方法如下，當 `WEN` 是1時，會根據 `addr` 提供的位址，轉換成Decoder輸出的訊號，經過AND gate之後，active相對應位置的register。

output則是使用MUX，抓取相對應位置的data，當 `REN` 是1時，藉由dout輸出，否則輸出 `1'b0` 。

圖例是一個 `memory [3:0]` 的memory，如果要實作出 `[7:0] memory [127:0]` 的memory的話，就多做幾個.....（`[7:0]` 的部份就多copy幾份，讓每個module處理各自的bit；`[127:0]` 的部份，就在每個module內接到128個register就可以了）

Testbench

這份testbench內，我定義了一個

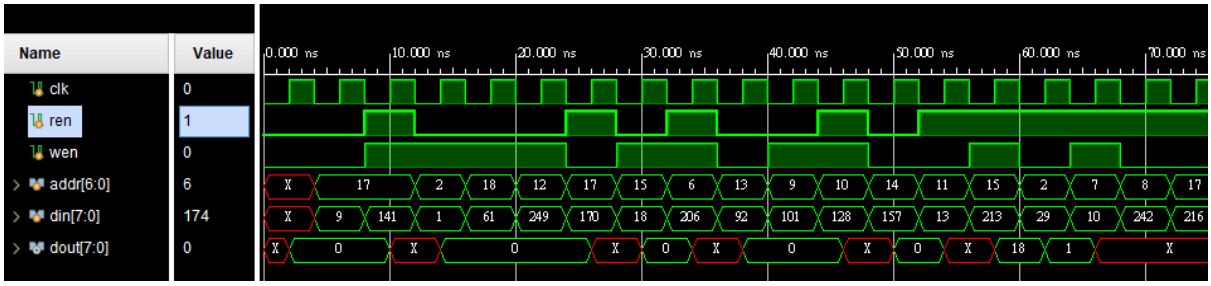
`parameter test = 20`，目的是想說既然構造方法一樣，那我只需要測試0~19 這個區間，如果正常，那整體就沒有問題。

前面隨機給值測試，最後一次觀察memory中儲存的值是否正確。

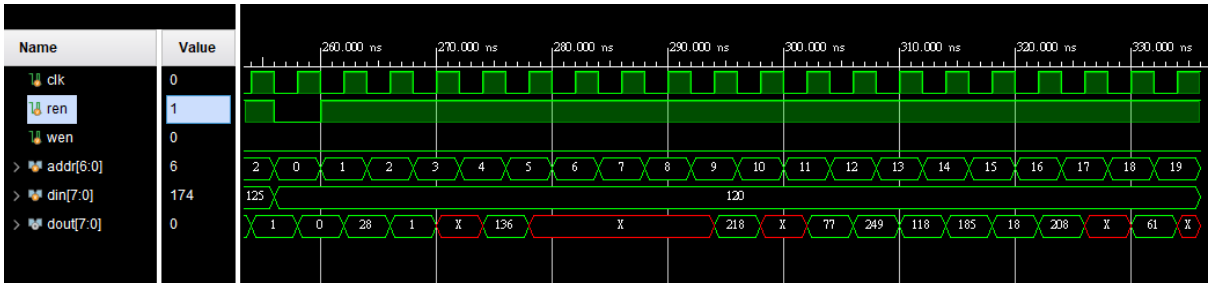
```
repeat (2 ** 6) begin

    @(negedge clk) begin
        {ren, wen} = $random;
        addr = {$random} % test;
        din = $random;
    end
end

addr = 0;

repeat (test) begin

    @(negedge clk) begin
        {ren, wen} = 2'b10;
        addr = addr + 1;
    end
end
```

Waveform

隨機測試：



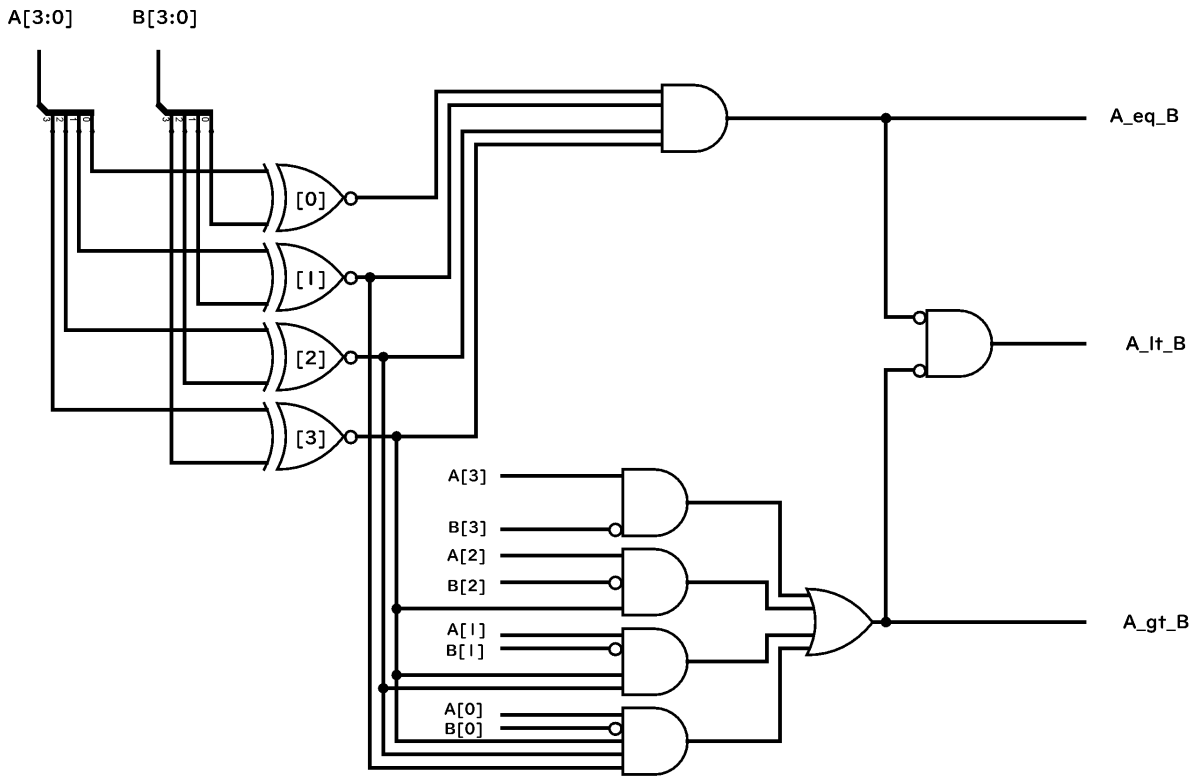
最後觀察全部memory存放的值：



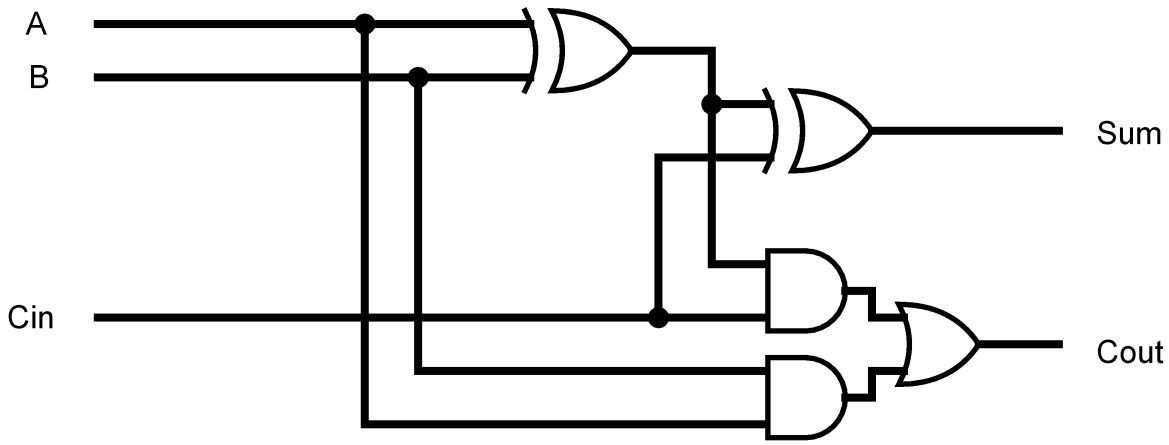
Advanced Question 3

Comparator

我們使用了Lab 1的Comparator，等等可以用來比較`min`, `max`與`count`的值。

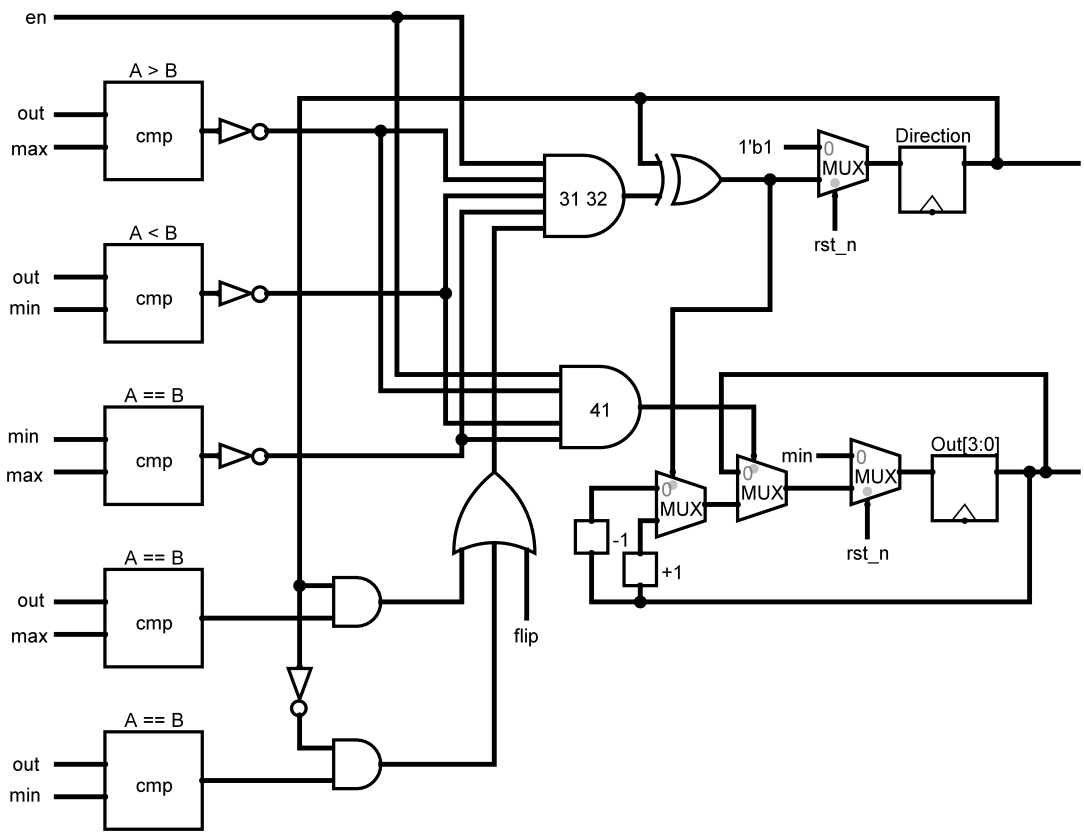


Adder



就用Adder做出+1或-1的功能（-1就轉換2's complement再+1，利用XOR minus signal做到轉換2補數的效果，我好像把圖弄丟了QQ），等等用來當`count`的遞增或遞減。

Q3 Circuit



直接附上組合電路的code，這樣子看AND的線路來源會比較方便（AND上的數字代表第幾行的if）

```
29  always @(*) begin
30
31      if (enable && out >= min && out <= max && min != max) begin
32          if ((direction && (out == max)) || (!direction && (out == min)) || flip)
33              next_direction = ~direction;
34          else
35              next_direction = direction;
36      end
37      else
38          next_direction = direction;
39
40
41      if (enable && out >= min && out <= max && min != max) begin
42          next_out = (next_direction ? out + 1 : out - 1);        You, 35 minutes ago
43      end
44      else
45          next_out = out;
46  end
47  endmodule
```

判斷方法基本上跟Basic的版本一樣，只是多了 `max` , `min` , `flip` 要判斷，畫圖好累QQQ

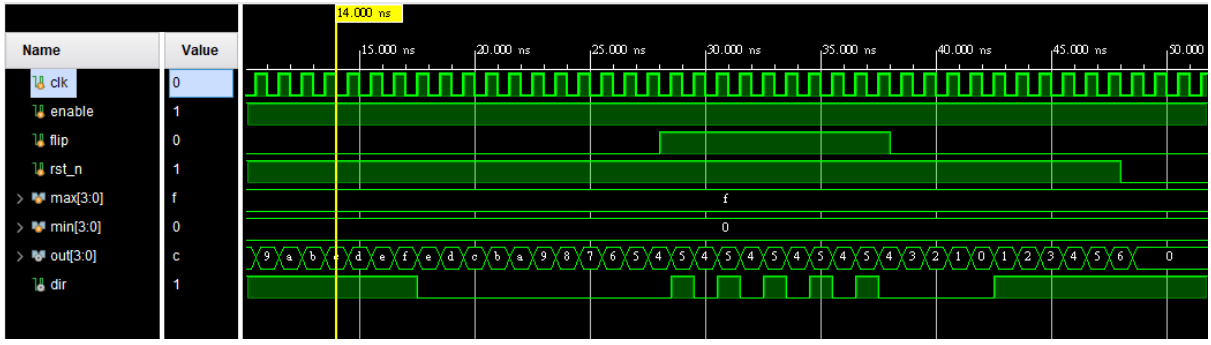
Testbench

送入 `reset` , `flip` , `enable` 觀察有沒有變成正確的結果。

```
initial begin
    clk = 0;
    enable = 1;
    flip = 0;
    rst_n = 0;
    max = 15;
    min = 0;

    @(negedge clk) rst_n = 0;
    @(negedge clk) rst_n = 1;

    #26
    flip = ~flip;

    #10
    flip = ~flip;
```

```
    #10
    @(negedge clk) rst_n = 0;
    #5
    @(negedge clk) rst_n = 1;

    #5
    enable = 0;
    #10
    enable = 1;
    #5
    $finish;
    // ....
end
```

Waveform



Contributions

Q1

- Module: 林諭震
- Testbench: 莊景堯

Q2

- Module: 林諭震
- Testbench: 莊景堯

Q3

- Module: 林諭震
- Testbench: 莊景堯

CAD Test

- 林諭震

FPGA

- 莊景堯

Report

- 莊景堯

Learned

1. 寫report真的好累QQ
2. FPGA
    - 除頻頻率我不會抓，FPGA燒進去時有時候顯示變成每次+2（實測認為大概是跳過了1個clk，我推測可能是同時使用改變數字的clk與顯示的clk導致出現了problem之類的，不過看simulate也正常），有時候只會顯示第二個跟第四個數字，改個除頻頻率又正常，重燒一次再度爆炸，超崩潰（我覺得時間都浪費在改個除頻的數字又要重新generate一次）
        - 10/22 update: Demo一次過，但是我覺得在除頻的部份，還是要多練習QQ
    - `flip` 跟 `rst_n` 直接炸開來，原本我把 `flip` 寫在組合電路判斷 `next_direction` 的值（就是Q3 Circuit圖那樣），但是燒進FPGA之後 `flip` 一點用都沒有，我也有嘗試過改成寫在DFF裡面跟 `rst_n` 一樣用判斷的，但是一樣沒反應，超奇妙，明明 `rst_n` 也是一樣的寫法，就可以正常發揮功能……
    - 同一個if只寫一個判斷的變數我沒有很get到，如果我分層寫if的話，不就會變成MUX大串燒會造成嚴重的delay嗎

    - 10/22 update: 盡量讓MUX寫成2-to-1 MUX，可以在debug上會比較easy一點，也可以避免不小心寫了一個可能8-to-1 MUX，但是只接上6個input的情況
- 基於以上各種雜七雜八的問題，我覺得有一部份原因是因為還沒建立起好的coding習慣，不知道可不可以在Lab結束之後，公布一個助教們的FPGA範本，讓我們檢討哪裡可以改進嗎，謝謝QQQ