Lab1 Report

Questions：

1、 Mux_8bits

2、 4_to_16_Decoder

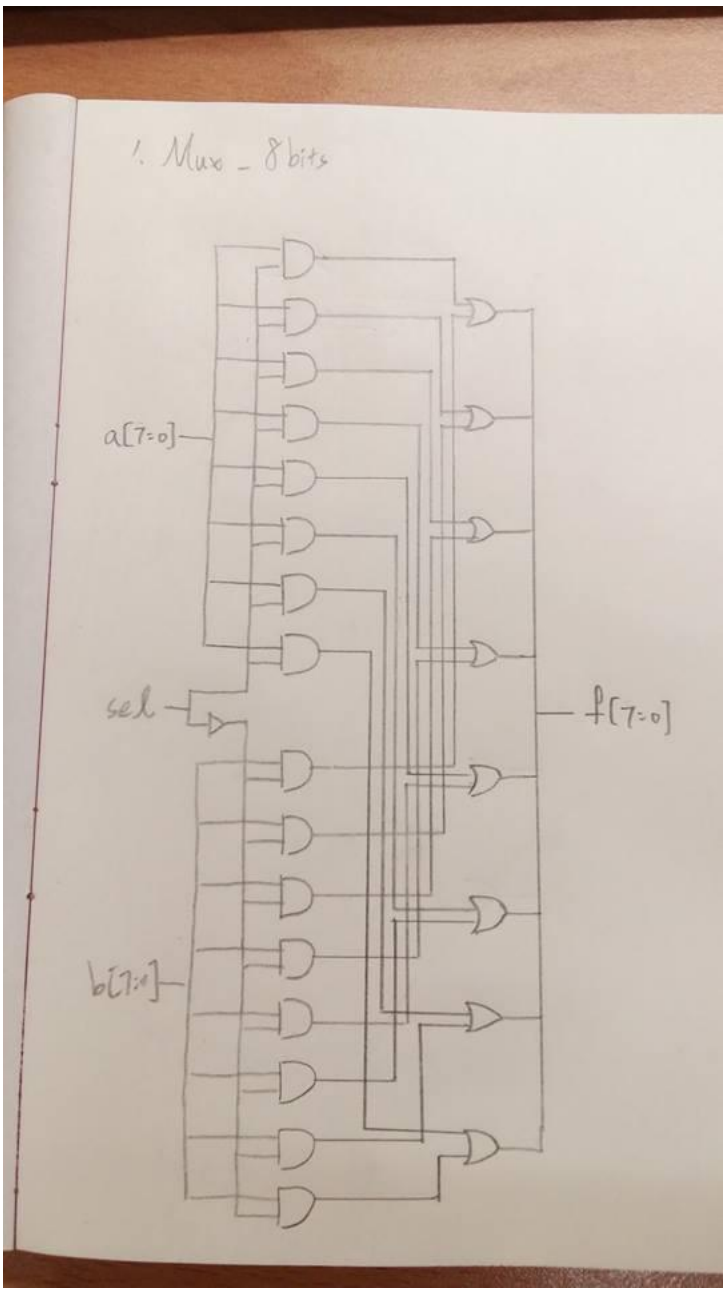
3、 Comparator_3bits

4、 RippleCarryAdder

組長：蔡登瑞、組員：蔡政諺

1. Mux_8bits

(1) Logic Gates



(2) Designs in Detail

以 **f[0]**為例，a[0]和 sel、b[0]和 sel 的 negation 分別通過 AND gate 後，再通過 OR gate 接到 f[0]。

Boolean Expression: f[0]=(a[0]^sel)∨ (b[0]^~sel)

sel 就是一個選擇器的概念，

sel=1 時，f[0]=(a[0]^1)∨ (b[0]^0)=a[0]∨ 0=a[0]，會輸出 a[0]的值；

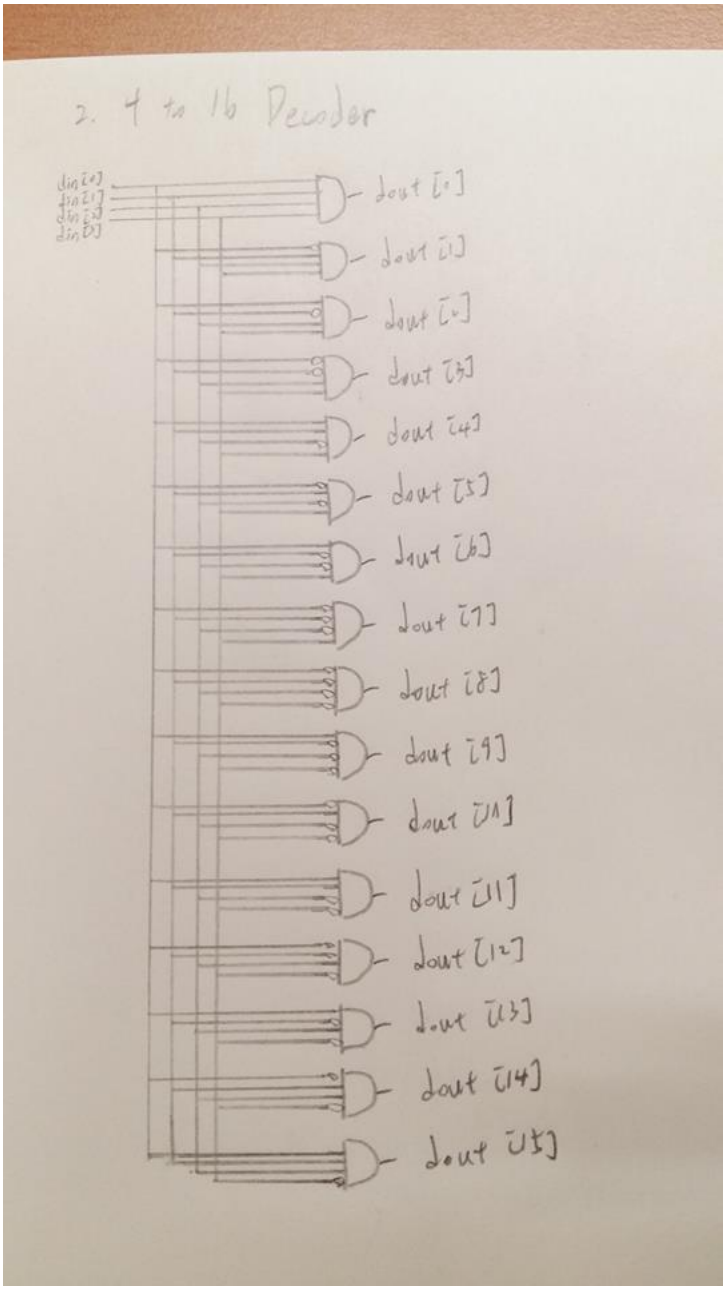
sel=0 時，f[0]=(a[0]^0)∨ (b[0]^1)=0∨ b[0]=b[0]，會輸出 b[0]的值。

同理，**f[1] ~ f[7]**也是遵照相同的邏輯。

所以當 sel=1 時，會輸出 a[7:0]；當 sel=0 時，會輸出 b[7:0]。

2. 4 to 16 Decoder

(1) Logic Gates



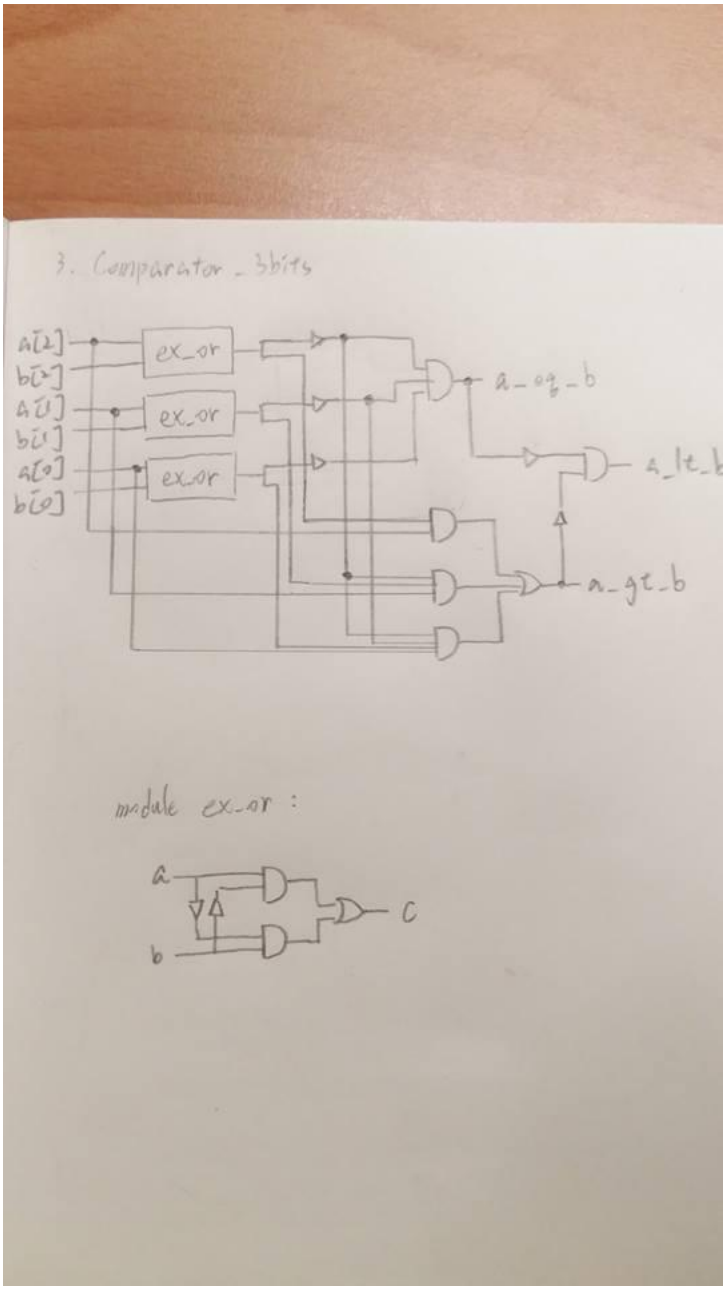
(2) Designs in Detail

{din, dout} = {1111, 0000000000000001};
{din, dout} = {1110, 0000000000000010};
{din, dout} = {1101, 0000000000000100};
{din, dout} = {1100, 0000000000001000};
{din, dout} = {1011, 0000000000010000};
{din, dout} = {1010, 0000000000100000};
{din, dout} = {1001, 0000000001000000};
{din, dout} = {1000, 0000000010000000};
{din, dout} = {0000, 0000000100000000};
{din, dout} = {0001, 0000001000000000};
{din, dout} = {0010, 0000010000000000};
{din, dout} = {0011, 0000100000000000};
{din, dout} = {0100, 0001000000000000};
{din, dout} = {0101, 0010000000000000};
{din, dout} = {0110, 0100000000000000};
{din, dout} = {0111, 1000000000000000};

誠如上面所見，最簡單方便的方式就是硬爆。取得 din[3] ~ din[0]的反向 Ndi[3] ~ Ndi[0]，就用 16 個 AND gate 將各個 output 所需要的條件分別放入 input。最後就可以得到一個 4 to 16 的 Decoder。

3. Comparator_3bits

(1) Logic Gates



(2) Designs in Detail

由於布林代數 1、0 不能一次就分辨出>、=、<。

因此我先分成(=)跟(>、<)兩類，再去分別是(>)還是(<)。

1) a_eq_b

先利用 ex_or，也就是 XOR，若 XOR 後的值為 0，代表兩者同為 1 或同為 0，亦即 a=b。從最高位數到最低位數總共會用到三次的 ex_or。最後把三個 wire 反向後接到一個 AND gate，就可以得到 a_eq_b 的值。若 a、b 的各位數皆同為 1 或同為 0，代表 a 與 b 數值相同，a_eq_b 就會等於 1。

a_eq_b = ~(a[2] ⊕ b[2]) ^ ~(a[1] ⊕ b[1]) ^ ~(a[0] ⊕ b[0])。

2) a_gt_b

前面 a[2] ⊕ b[2]得到的值如果是 1，代表 a≠b。接著把 a[2] ⊕ b[2]的值跟 a[2]接到一個 AND gate 上， 其結果若為 1，代表說 a[2]為 1，相對的 b[2]就為 0，因此可以判斷 a>b。 相反來說，結果若為 0，則 a<b。但若是 a[2] ⊕ b[2]得到的值為 0，可是 a[1] ⊕ b[1]為 1，那就把~(a[2] ⊕ b[2]) 跟 (a[1] ⊕ b[1]) 還有 a[1] 接到一個 and 上。會把~(a[2] ⊕ b[2]) 接上 and 是因為當 a[2]<b[2]時，(a[2] ⊕ b[2]) ^a[2] =0。這時如果沒有~(a[2] ⊕ b[2])，最後 a_gt_b 就會依據 (a[1] ⊕ b[1]) ^ a[1]的值。 這樣一來就會發生錯誤。相同的，在(a[0] ⊕ b[0]) ^ a[0]時一樣加上 ~(a[2] ⊕ b[2]) ^ ~(a[1] ⊕ b[1])。最後把三個 and 的結果 or 起來，就可以得到 a_gt_b。

a_gt_b = ((a[2] ⊕ b[2]) ^ a[2])
V(~(a[2] ⊕ b[2]) ^ (a[1] ⊕ b[1]) ^ a[1])
V(~(a[2] ⊕ b[2]) ^ ~(a[1] ⊕ b[1]) ^ (a[0] ⊕ b[0]) ^ a[0])

3) a_lt_b

經過了上面 a_eq_b 跟 a_gt_b 重重難關後，剩下最後的 a_lt_b。

a_lt_b 其實有兩種寫法，第一種方法是跟上述 a_gt_b 一模一樣，這樣一來，就會需要更多的 Logic Gates，更多的 Logic Gates 就會需要更多的時間跑跟花費更多的面積，因此我們選擇了第二種方法。

第二種方法，就是利用已經找到的 a_eq_b 跟 a_gt_b 來得到 a_lt_b。

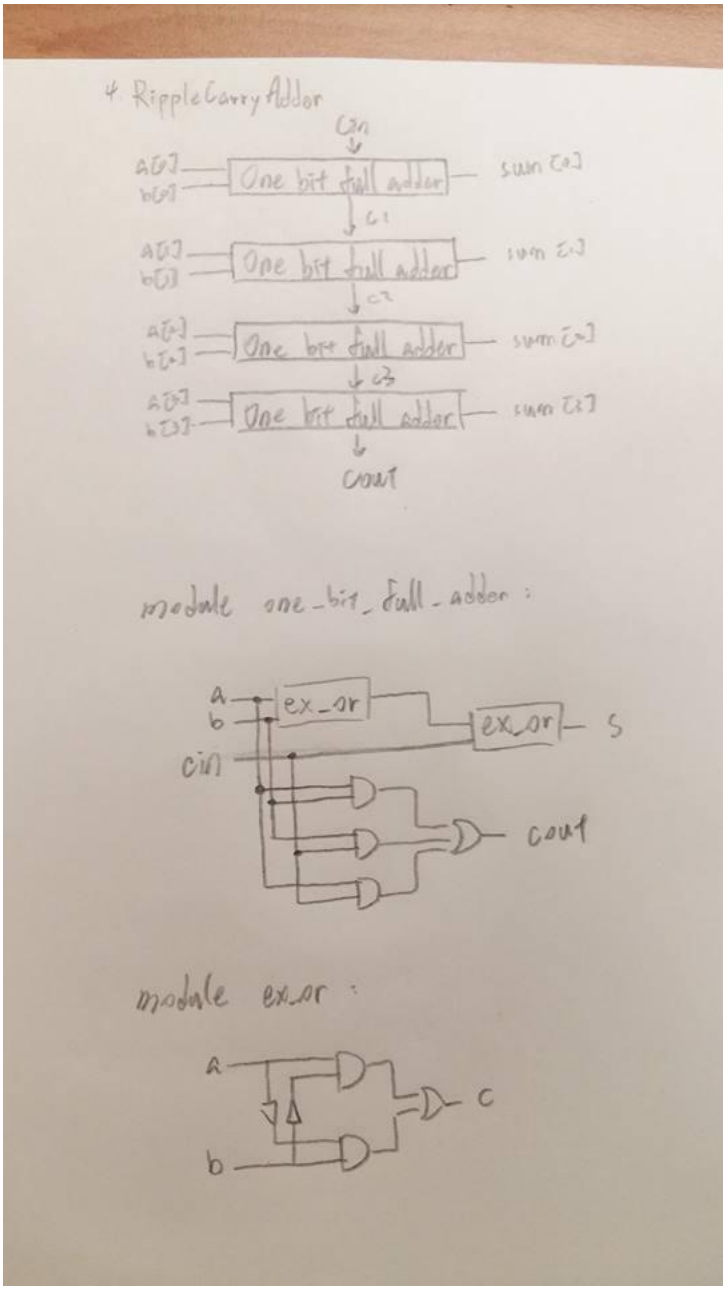
這個方法是用到相對的概念，因為 a 跟 b 比大小只會有大於、等於或小於，不會有同時大於等於或是同時等於小於的狀況發生。

因此，當 a_eq_b 跟 a_gt_b 同時為 0 時，a_lt_b 必為 1。

a_lt_b = ~a_eq_b ^ ~a_gt_b。

4. RippleCarryAdder

(1) Logic Gates



(2) **Designs in Detail**

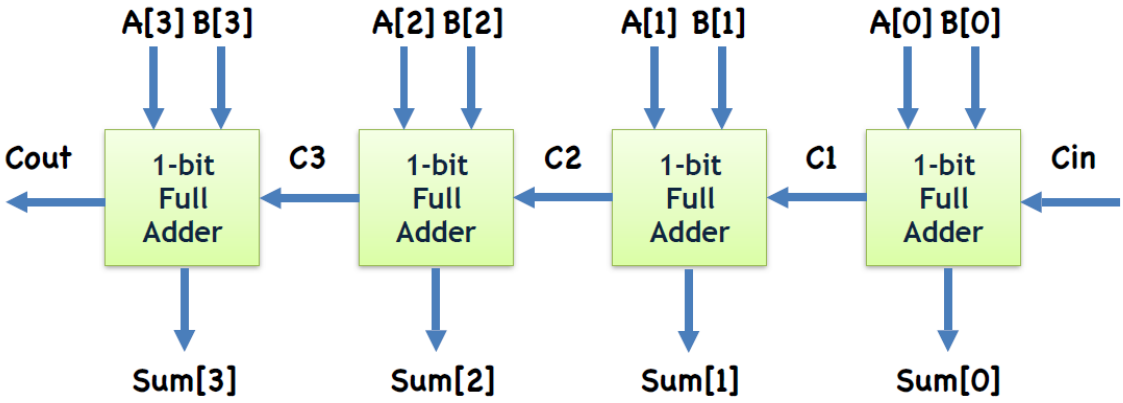
因為不能直接使用內建的 XOR gate，所以我組了一個 **ex_or** 取代其功能。

Boolean Expression: c=(a^~b)V(~a^b)

接著用 ex_or(以下以⊕表示)和其他 logic gates 組出 **1-bit** 的 **FullAdder**。

Boolean Expression:
s=(a⊕b)⊕cin
cout=(a^b)V(a^cin)V(b^cin)

RippleCarryAdder 由 4 個 1-bit FullAdder 構成，串接方式如助教在 pdf 中提供的圖：



低位的 1-bit FullAdder 的 cout 會成為高一位元的 1-bit FullAdder 的 cin。

透過此系統，可以自低位(sum[0])逐一運算至高位(sum[3]、cout)，達到 4 位數二進位加法的目的。

- **How we test our designs?**

```
1  `timescale 1ns/1ps
```

首段的 timescale 為時間單位的宣告。

```
2
3  module Comparator_t;
4  reg [3-1:0] a = 3'b0;
5  reg [3-1:0] b = 3'b0;
6  wire a_lt_b, a_gt_b, a_eq_b;
7
8  Comparator_3bits C1 (
9    .a (a),
10   .b (b),
11   .a_lt_b (a_lt_b),
12   .a_gt_b (a_gt_b),
13   .a_eq_b (a_eq_b)
14 );
15
```

接下來要在 testbench 裡宣告 input 與 output (input 要定義好數值)，並接到我們設計好的 module。

```
16 initial begin
17   repeat (2 ** 6) begin
18     #1 {a, b} = {a, b} + 1'b1;
19   end
20   #1 $finish;
21 end
22
23 endmodule
```

若有 n bits 的 input，我們就執行一個重複 2 的 n 次方的迴圈，以 run 過每一種測資。

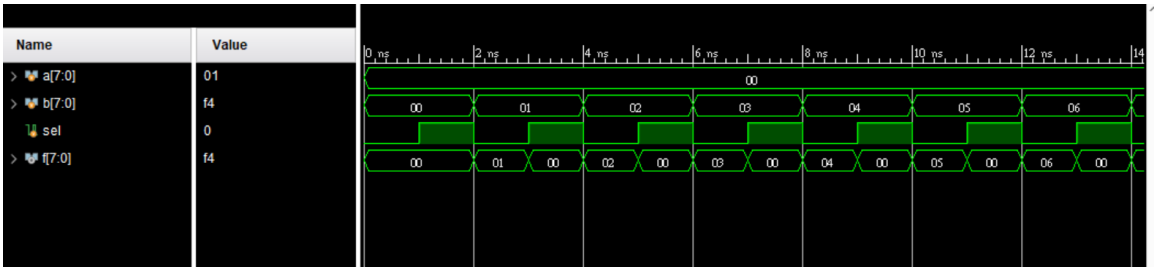
以 3-bit Comparator 為例，a、b 各為 3-bit 的 input，合起來為 6 bits，所以我在 testbench 的開頭先 reg a、b 為 0，接著跑一個 2^6 次的迴圈，每次都使{a, b}的數值加 1。這樣就可以使得{a, b}從 000000 跑到 111111。

以此類推，8-bit Mux 的 input 包含 8-bit 的 a、b，1-bit 的 sel，總共為

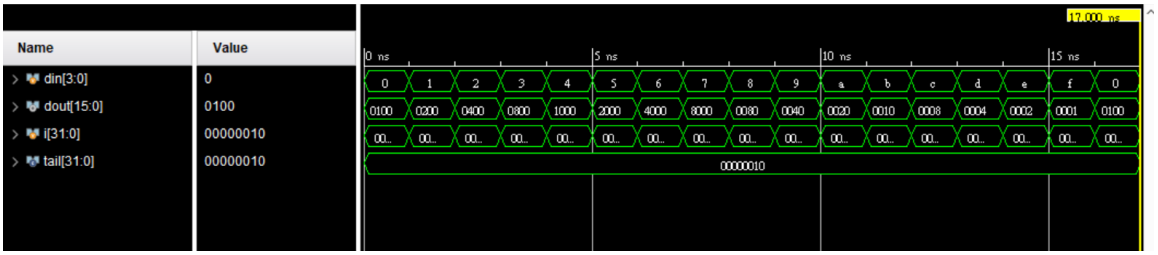
17 bits，就執行一個 2^{17} 次的迴圈。RippleCarryAdder 的 input 包含 4-bit 的 a、b，1-bit 的 cin，總共為 9 bits，就執行一個 2^{9} 次的迴圈。Decoder 的 input 包含 4-bit 的 din，總共為 4 bits，就執行一個 2^{4} 次的迴圈。

- **Waveforms**

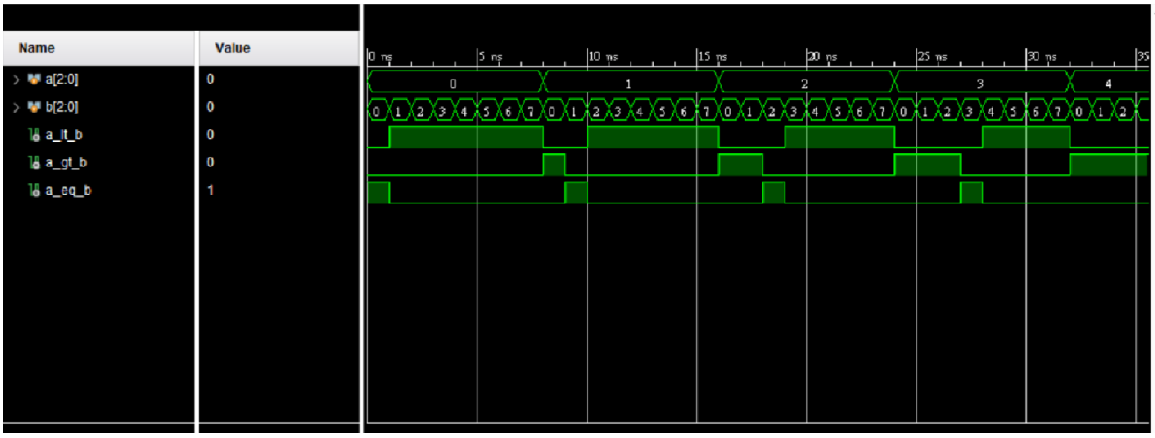
1. Mux_8bits

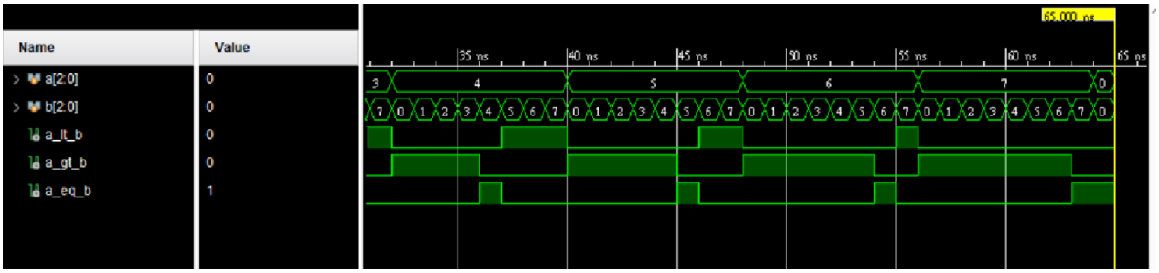


2. 4 to 16 Decoder

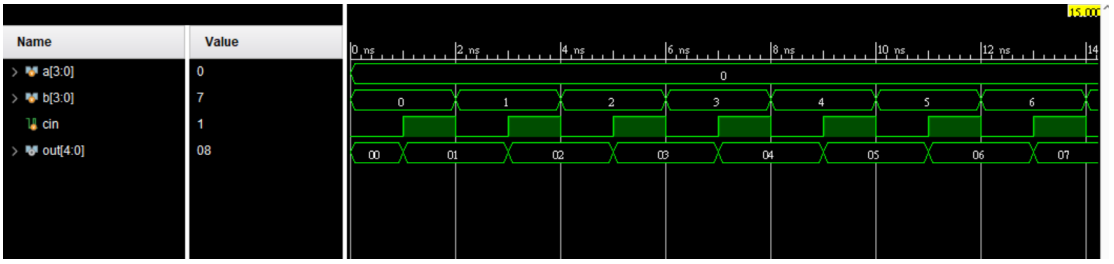


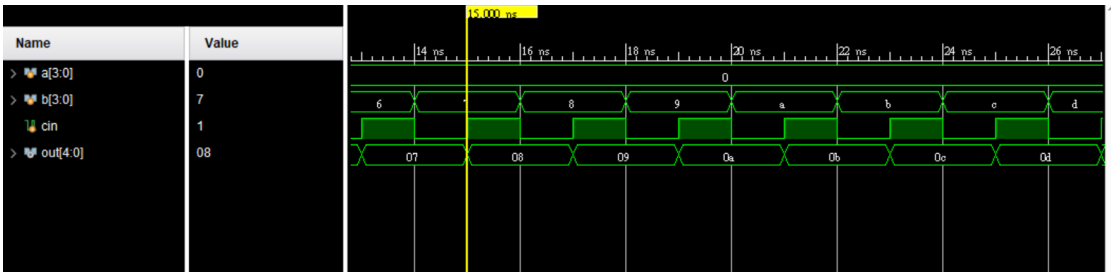
3. Comparator_3bits

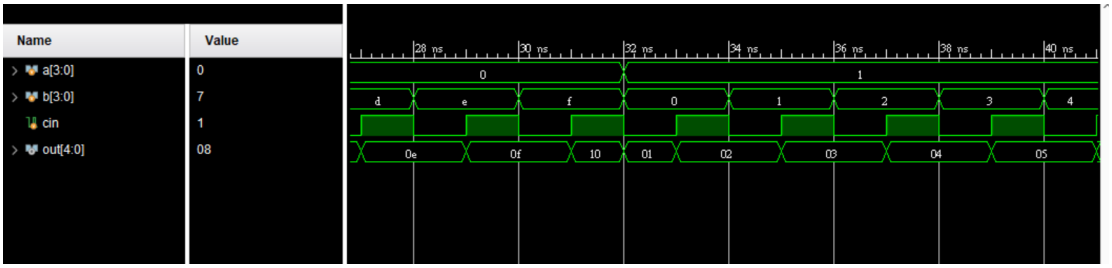




4. RippleCarryAdder







- **What we have learned in Lab1?**

我們從這次的 lab 中重新認識 verilog，並且認知到看似最簡單的 Gate Level 原來有這麼大的用處。此外，經過這次的 lab，我們對於 Vivado 的操作愈來愈嫻熟，相比於上學期使用 MobaXterm 登入工作站寫 code，Vivado 的介面更直白明瞭，我們相信這對我們的實作能力會造成飛躍性的幫助。這次 lab 也認識了 FPGA 這個硬體設備，學會藉由建立 XDC 檔將 verilog 語言燒到 FPGA 板上，終於能使用 testbench 以外的管道測試我們的 design。

- **Contributions**

1、 分工：蔡登瑞，負責寫 code、testbench。

蔡政諺，負責畫 gate-level circuit、燒 FPGA。

2、 合作：寫 report。