Lab 1

學號: 110011138 姓名: 楊立慈

A. Lab Implementation

(a) Lab1_1

1. 下圖 1 為 lab1_1 的 block diagram，其中斜線上方數字代表該 variable 有多少 bit。我們可看到 input 分別有 4-bit 的 a 跟 b、2-bit 的 op，output 則是 4-bit 的 d。

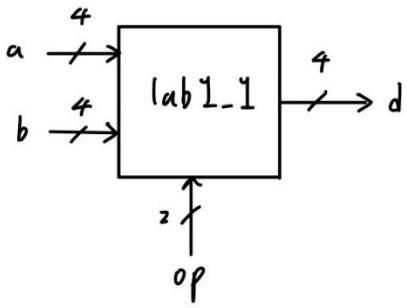

圖 1: lab1_1 block diagram

2. 下圖 2 為 lab1_1 的主要代碼。由於使用 assign statement 來寫可能會造成可讀性不佳，因此選擇用 always 語法來寫，並使用 case statement 來枚舉所有 op 的值以及對應到該值時要做的運算。Default case 則是為了避免萬一有某個 case 沒有被枚舉到會造成 latch inference (個人習慣不管需不需要都會寫 default case)。

```
8     always @(*) begin
9         case (op)
10            2'b00: d = a & b;
11            2'b01: d = a << b;
12            2'b10: d = a | b;
13            2'b11: d = a >> b;
14            default: d = 0;
15        endcase
16    end
```

圖 2: lab1_1 partial code

(b) Lab1_2

1. 下圖 3 為 lab1_2 的 block diagram。最外圍的框框代表整個 lab1_2 block，左側是此 block 的 6 個 input，右側則是 1 個 output。由於針對 op_0、op_1 使用的運算都與 lab1_1 一樣，只有使用到的 source 會不一樣，因此下圖在 lab1_2 內部使用了兩個 lab1_1 的 block 並接上相對應 op_0、op_1 的 sources，並將 output 分別命名為 op_0_result、op_1_result。由於根據不同 request 值我們需要給 result 不同運算的值，因此使用一個 4-to-1 MUX，以 2-bit 的 request 作為 selection line，並依照 spec 所述，request 是 00 代表沒有 request 所以要給 2'b0000 的值、01 要接 op_0_result、10 要接 op_1_result、11 由於 op_0 的順位比較高，所以要接 op_0_result。

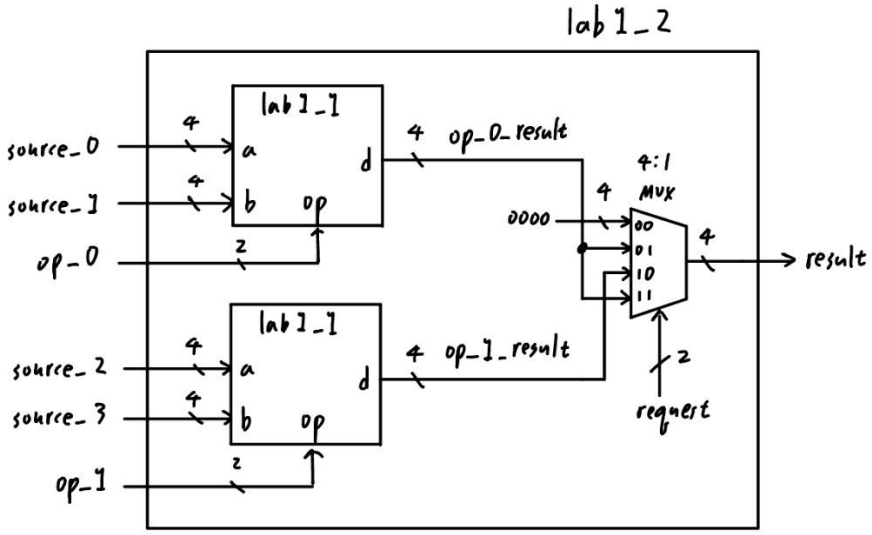

圖 3: lab1_2 block diagram

2. 下圖 4 為 lab1_2 的主要代碼。首先我們 declare 了兩個 4-bit variable，op_0_result、op_1_result，並將其接到兩個 lab1_1 instances 的 output，至此就完成 block diagram 中的兩個 lab1_1 block 了。剩下的就是做出 block diagram 中 MUX 的功能，其中 01 跟 11 可都當作 request[0] == 1 的情況將 result 設為 op_0_result，10 則當作 request[1] == 1 的情況(request == 2'b11 的情況已經在第一個 if 就處理了，所以不會進到這個 else-if block)，其餘情況則都設為 2'b0000。

```
12      wire [3:0] op_0_result;
13      wire [3:0] op_1_result;
14
15      lab1_1 m1(.op(op_0), .a(source_0), .b(source_1), .d(op_0_result));
16      lab1_1 m2(.op(op_1), .a(source_2), .b(source_3), .d(op_1_result));
17
18      always @(*) begin
19          if (request[0] == 1) begin
20              result = op_0_result;
21          end
22          else if (request[1] == 1) begin
23              result = op_1_result;
24          end
25          else begin
26              result = 4'b0000;
27          end
28      end
```

圖 4: lab1_2 partial code

B. Questions and Discussions

(a) Question A

由於該 input assignment 需要加法等等運算，而在電路上來看這些運算需要用一些 gate 接起來才能實現，而 gate 會有 delay，因此會有 gate delay 的問題，也就是說這個運算是需要時間的。如果我們沒有#DELAY 在 output verification 之前的話，會造成我們在 verify 時 input assignment 可能還沒運算完，導致我們根本是在 verify 錯誤的值。

(b) Question B

我們希望在 request == 2'b11 時以 op_0、op_1 的值決定順位，優先做有 2'b00 的 operation，再來是 2'b01、2'b10、2'b11，當 op_0 和 op_1 一樣時，則 op_0 的順位較高。

我們定義 op_0 = AB (A 是 MSB，B 是 LSB，i.e. A = op_0[1]、B = op_0[0])、op_1 = CD，並定義 1-bit 的 variable R 為當 R = 0 時 result = op_0_result，當 R = 1 時 result = op_1_result。以 AB、CD 為 input，R 為 output，我們可以畫出下圖 5 中左側的 truth table。

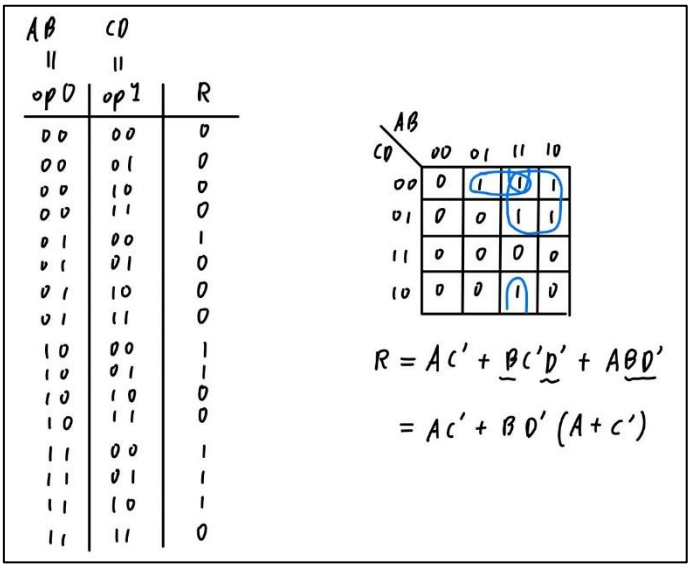

圖 5: Question B truth table and K-map

接著依照 truth table 畫出如圖 5 右側的 K-map 以及寫出 minimum SOP form，並做化簡。再來只要將原本 lab1_2 always block 裡的 if-else statement 改為 request == 00、01、10、11 的情形，並在 request == 11 時使用上述的 R 的式子來判斷要給 op_0_result 還是 op_1_result。實作 code 如下圖 6。

```
30      always @(*) begin
31          if (request == 2'b01) result = op_0_result;
32          else if (request == 2'b10) result = op_1_result;
33          else if (request == 2'b11) result =
34              (op_0[1] & ~op_1[1] | op_0[0] & ~op_1[0] & (op_0[1] | ~op_1[1]))
35              ? op_1_result : op_0_result;
36          else result = 4'b0000;
37      end
```

圖 6: Question B code

C. Problem Encountered

(a) 在 lab1_2 裡 declare op_0_result 和 op_1_result 時忘記要寫[3:0]，變成相較於 4-bit 它們只有 1-bit，導致跑測試出現一百多個 error。為了 debug，我先確認一遍 code 本身的邏輯沒問題後，就利用 testbench 的 display 去確認問題，後來發現有 error 的 result 都是 2'b0000 或 2'b0001，看起來好像前三位都沒被算到，才終於找到 bug 在存 op_0 跟 op_1 的 result 只有 1 個 bit。

(b) 一開始 op_0_result 跟 op_1_result declare 成 reg type，而不是 wire type，導致在 Vivado run

simulation 時會失敗跑不了，並出現錯誤訊息: Concurrent assignment to a non-net op_0_result is not permitted。在上網查詢後，才知道要 instantiate module 時的 output 要用 wire 去接才可以，不能用 reg。仔細想想也蠻合理的，因為在 always/initial block 之外基本上不會用 reg type，而且可以想成我們是不斷 assign 該 module 的 output 值給 op_0_result，就好像實際接一條線過去一樣，因此不能用 reg。

D. **Suggestions**

關於 Lab 的 demo 方式，目前是學生上去白板寫學號，並由 TA 一個一個叫號碼並過去 demo，感覺整堂 demo 下來 TA 要一直走來走去。我在想或許可以改成每次 lab 都開一個 google sheet，讓寫完 lab 準備要 demo 的同學上去填學號，然後教室固定一個位置專門給 demo 的人用，一個學生 demo 完之後在 google sheet 上他的那格就被劃掉之類的，下一個準備 demo 的學生自己看到 sheet 上前幾個人被劃掉之後就可以到 demo 位置旁邊準備。這樣 TA 就只要在 demo 位置上就好，學生也可以省掉排隊寫黑板的時間。