Lab 3

學號: 108032053 姓名: 陳凱揚

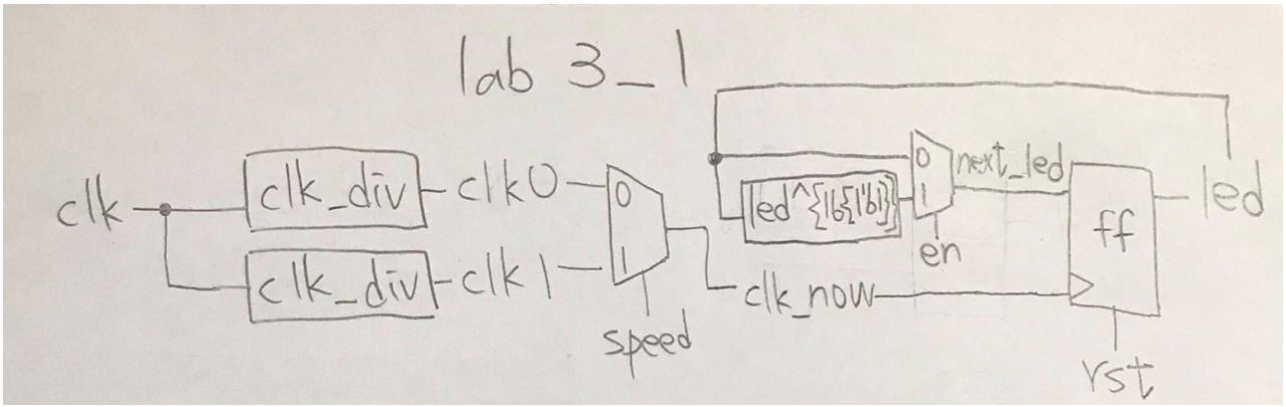
1. 實作過程

(1) clock_divider

我以一個 n bits 的 counter，每隔一個 cycle 加 1，再將 output 接上這個 counter 的 MSB，即可將 clk 除頻2^n倍。

(2) lab3_1

如下圖 1，我先利用前小題做出的 clock_divider，分別得到除頻2^{27}和2^{24}倍的 clk1 和 clk0，接著以 speed 選出現在的速度，作為 led 的 clk 訊號，而 led 則是在 en = 1 的情況下，不停的在全亮和全暗間轉換。

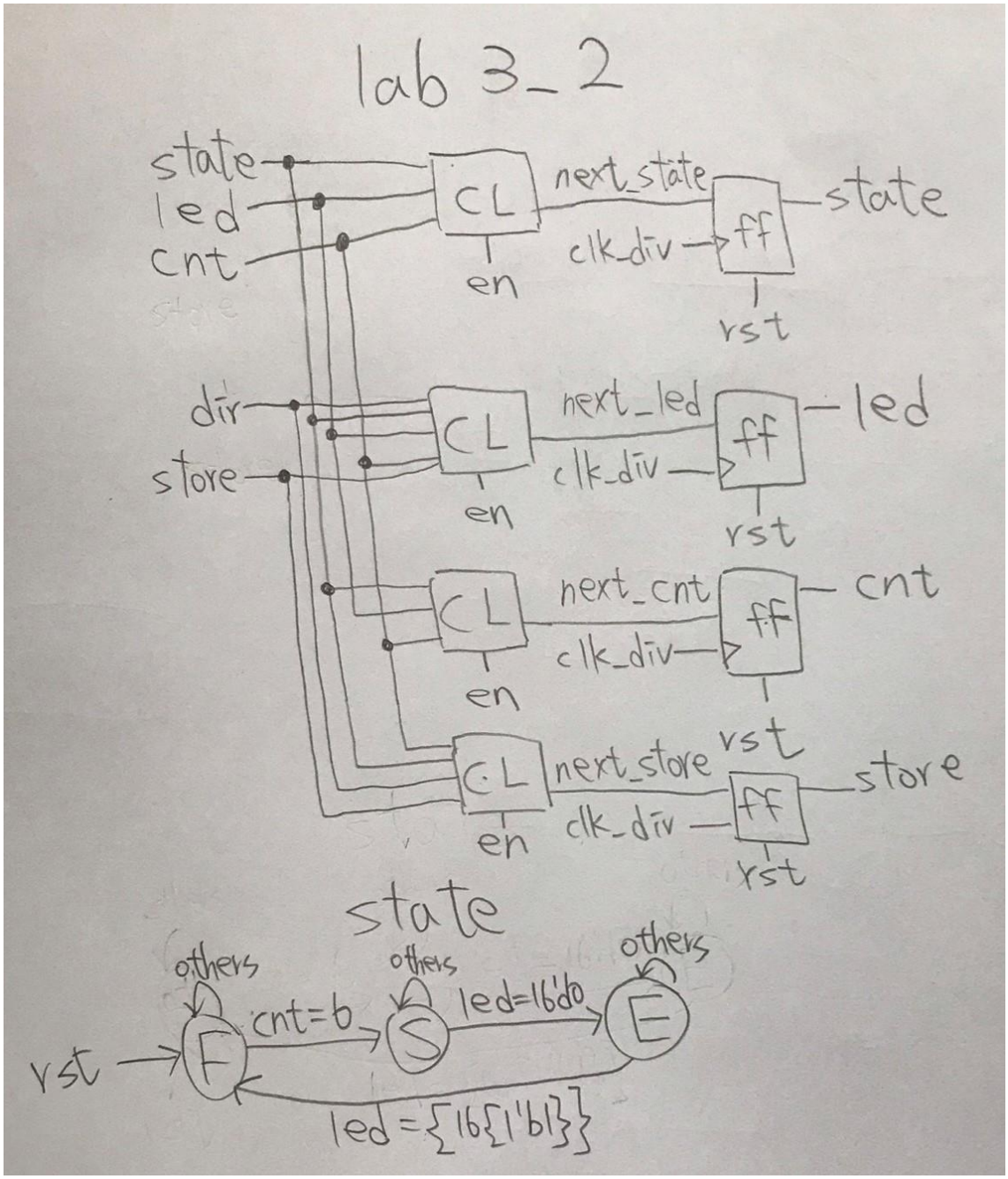


▲ 圖 1

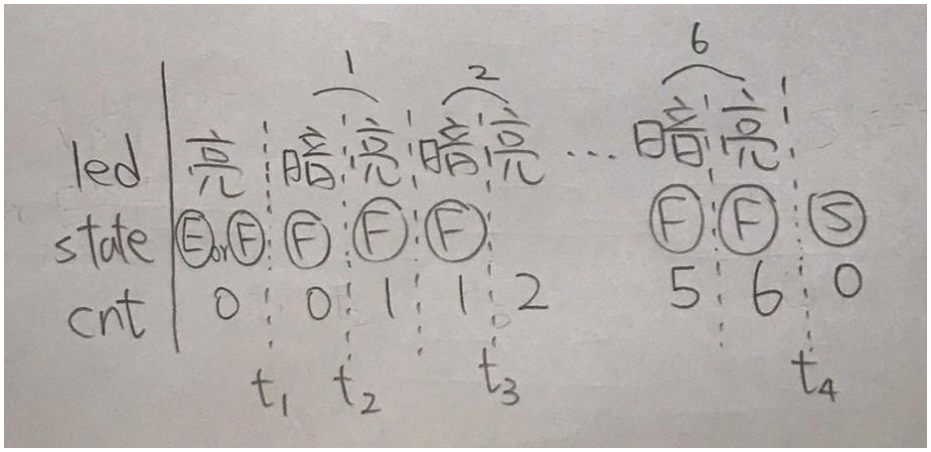
(3) lab3_2

如下圖 2，我主要由 4 個 filp-flop 來記錄資訊，第一個是 state，有 FLASH, SHIFT, EXPAND 等三種狀態，並依照 state diagram 來決定下一個 state；第二個是 led，會依照當前狀態及 input 來決定下一個 cycle 的輸出；第三個是 cnt，於 FLASH 時使用，增加的條件為在 FLASH 狀態且 led 全暗，在其他 state 時皆為 0，沒有功能，如下圖 3，在 t1 時進入 FLASH 狀態，t2, t3 時 led 全暗，因此皆增加 1，接著在 t4 時，就恰好可以 cnt = 6 為條件進入 SHIFT 狀態；第四個是 store，於 SHIFT 時使用，有 48 bits，是 led 的 3 倍，用來儲存因為跑到邊界而不見的亮燈，可以在轉向時，依然保持有 8 顆亮燈出現，在其他 state 時，store 皆沒有功能。

在 code 的部分，我運用了很多 concatenate，像是在 FLASH 時，next_led = led^{16{1'b1}}，在 EXPAND 時，next_led = dir ? {1'b0, led[15:9], led[6:0], 1'b0} : {led[14:8], 2'b11, 1ed[7:1]}，來控制亮燈的擴張與縮小，而在 SHIFT 時，store 的中間 16 個 bits 即代表 led 目前的亮燈，即使 led2 的亮燈因到達邊界而消失，store 的左右 16 個 bits 依然能夠儲存住資訊，在 dir 轉向後，以 >> 和 << 推回原先的位元。


lab 3_2
state
led
cnt
dir
store
CL
en
next_state
clk_div
ff
rst
state
next_led
led
next_cnt
cnt
next_store
store
state
others
cnt=0
led=16'd0
rst
led={16{1'b1}}

▲ 圖 2

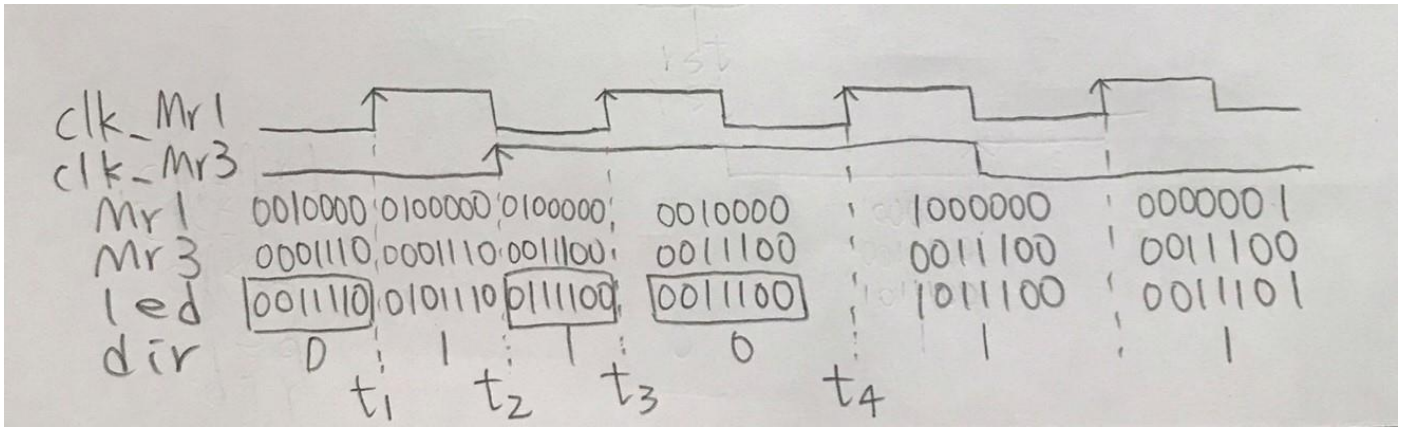

led
state
cnt
亮
暗
E or F
F
S
0
1
2
5
6
t1
t2
t3
t4

▲ 圖 3

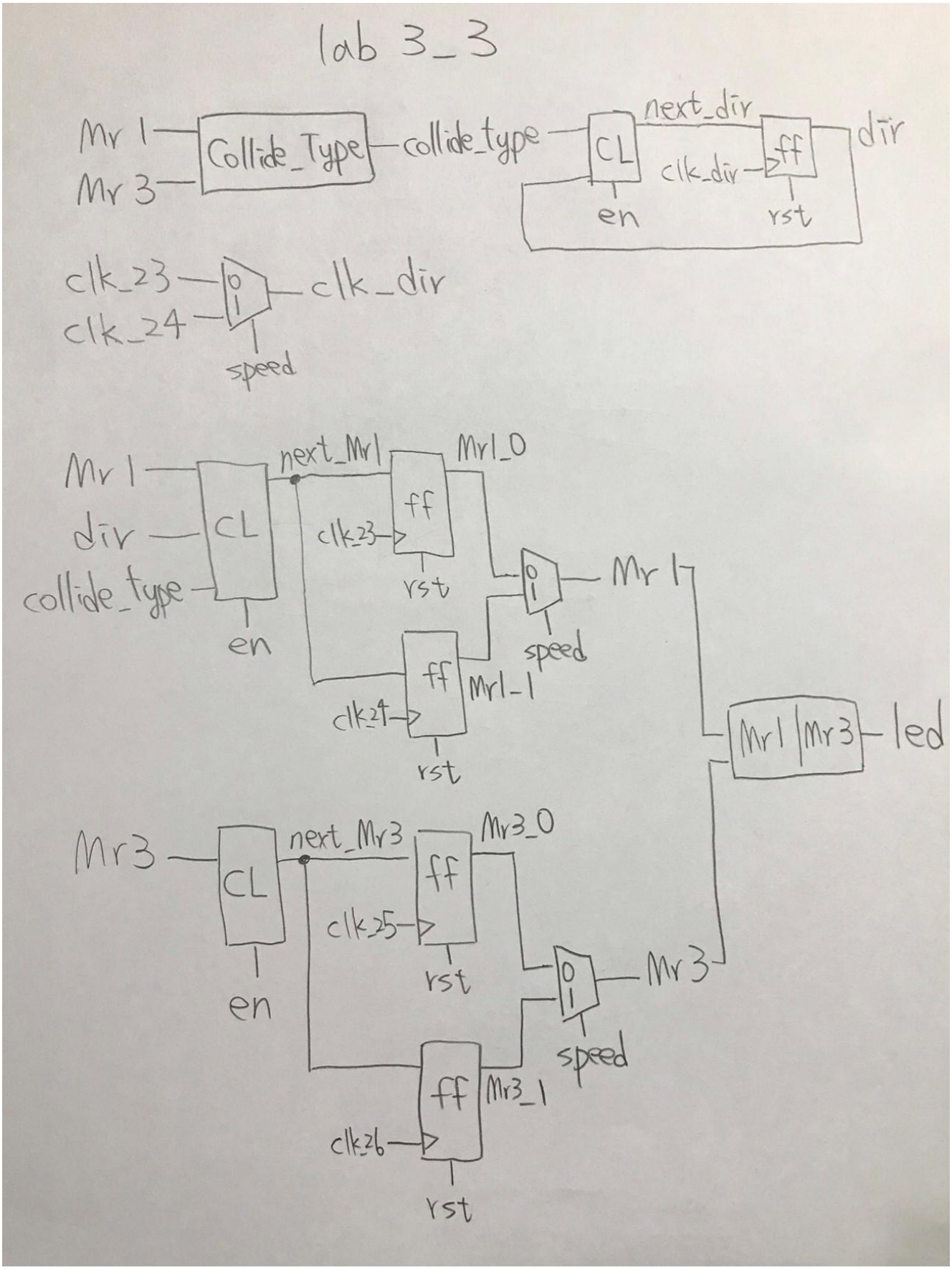
(4) lab3_3

如下圖 5，先看下方的大架構，我以 4 個 flip-flop 儲存 Mr1 和 Mr3 的各兩種不同速度更新位置，再以 speed 作為 select 選出現在的 Mr1 和 Mr3，而 led = Mr1 | Mr3。在計算 Mr3 的下一個位置相當容易，就只要一直向左前進即可，next_Mr3 = {Mr3[14:0], Mr3[15]}，而計算 Mr1 的下一個位置比較複雜，需要以目前的碰撞情況、目前前進方向、目前位置來決定。碰撞情況我寫成一個 module 來判斷，output 有四種結果，NONE、TOUCH、OVERLAP_EDGE、OVERLAP_MID，而 Mr1 的下一個位置即是以 collide_type 來決定碰撞時要往反向跳幾步，像是 TOUCH 只要往反向跳 1 步就能離開碰撞，另外兩種 OVERLAP 的情形分別是往反向跳 2 步和跳 3 步才能離開碰撞，NONE 只要依原始方向前進一步，而 collide_type 也會用來決定下一步的 dir。

至於為什麼在一發生碰撞就會反向的規則下，還是有可能發現 OVERLAP 的情況，是因為 Mr1 和 Mr3 的 clock 不相等，以下圖 4 來看，Mr3 的 clock 週期是 Mr1 的 4 倍長，原先 Mr1 向右前進，在 t1 時發生碰撞，collide_type 為 TOUCH，因此向後退一步，方向改為向左；在 t2 時，Mr3 向左前進一步，造成了在 t3 時，又發生了一次 TOUCH 碰撞，但這時 Mr1 的方向為向左，因此他往反向跳一步，反而更接近了 Mr3；則在 t4 時，即檢查出了 OVERLAP_EDGE 碰撞，讓 Mr1 向左跳了兩步，這時才結束了不停碰撞的情形。



▲ 圖 4


lab 3_3
Mr 1
Mr 3
Collide_Type
collide_type
CL
next_dir
clk_dir
ff
dir
en
rst
clk_23
clk_24
0
1
clk_dir
speed
Mr 1
next_Mr1
Mr1_0
dir
collide_type
CL
en
ff
clk_23
rst
speed
ff
Mr1_1
clk_24
rst
Mr1 | Mr3
led
Mr3
next_Mr3
Mr3_0
CL
en
ff
clk_25
rst
Mr3
speed
ff
Mr3_1
clk_26
rst

▲ 圖 5

2. 學到的東西與遇到的困難

我覺得活用位元運算和 concatenate 在硬體設計中很重要，能夠省下一些不必要的 flip-flop，不僅讓程式碼更簡潔，設計出來也更有效率，而且在檢查某些狀態時，位元運算相當好用，可以很快的算出結果。此外，我覺得 try and error 也很重要，像是我一開始無法想出為什麼 lab3_3 的碰撞會發生重疊的情形，所以我就先假設這個情況會發生並先設計實作，且自己寫 testbench、跑模擬、觀察波形圖，透過這些方法，我很快就找出了發生的原因，比起憑空想像還要更有效率了許多。

3. 想對老師或助教說的話

有一天薯條家庭在聊天
孫子：爺爺你怎麼那麼怕貓
…
…
…
…
…
爺爺：因為我是老薯