Logic Design Laboratory
Final Project Report

-Typeracer-

學生:

110062208 林書辰、110062205 楊景遇

組別:25

contribution:

林書辰:遊戲架構、圖片處理

楊景遇:VGA顯示、紅外線

中華民國 一百一十二 年 一 月 十三 日

Table of Contents

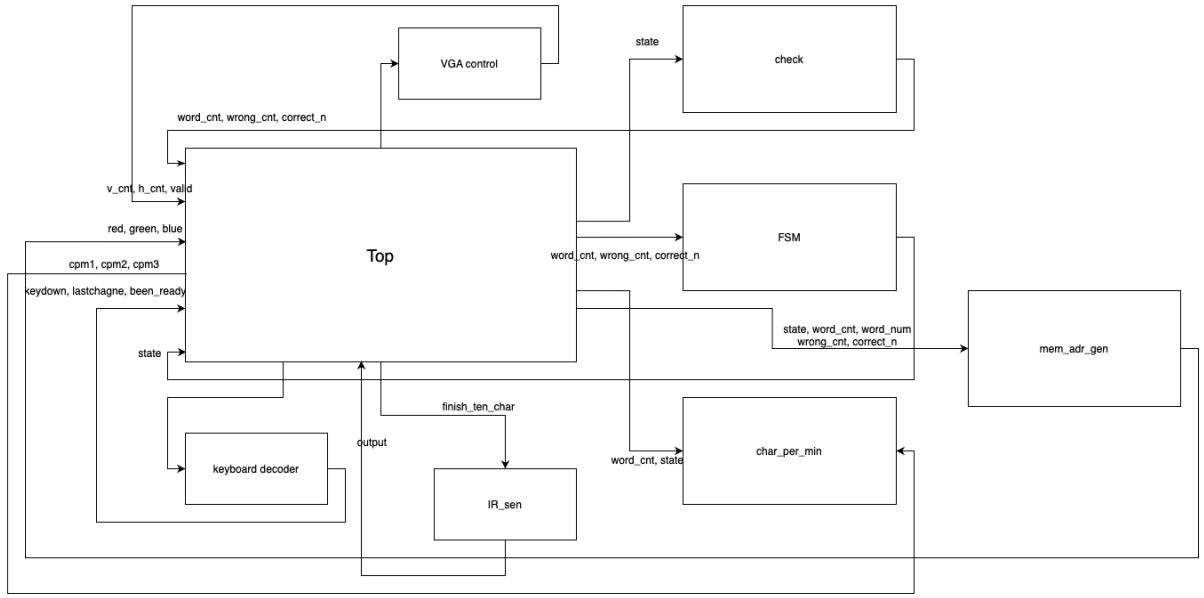
1. Motivation

我們兩個組員都喜歡打 typeracer，有事沒事都在打 typeracer，我們認為打字富有藝術，打在正確的位置上，感受鍵盤的回饋，體驗著飆車的快感，都讓人愛不釋手。雖然大部分的人可能不能理解我們為什麼喜歡打字，但我們熱愛著他，所以我們覺得我們在做 project 的時候會更有動力去完成。

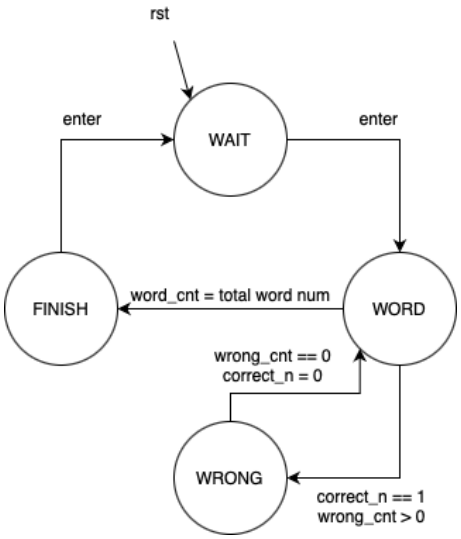
我們做這個 project 最主要的目的就是推廣 typeracer，希望大家能感受到他的樂趣，跟親朋好友一起 pk 打字速度。

2. Diagram

2.1 Block Diagram



2.2 State Diagram



3. Design

3.1 Modules and Variables

- Top module
  這是最主要的 module，用來呼叫其他所有的 modules，在這個 modules 裡面，我存了非常多的變數，為了在各個 module 中間呼叫。

  1. state，遊戲的 state，有 WAIT, WORD, WRONG, FINISH
  2. cpm1, cmp2, cmp3，這些是打字的速度，會在char_per_min裡面計算。
  3. correct_n，代表現在有沒有打錯，主要用來判斷要不要進到WRONG state
  4. word_cnt，用來記錄現在打到哪一個字。
  5. wrong_cnt，用來記錄打錯幾個字，只在WRONG state計算。
  6. output_word，計算v_cnt, h_cnt，告訴VGA現在要輸出什麼字。
  7. Red, Green, Blue，暫存VGA output顏色，等vga_control valid==1 輸出。
  8. finish_ten_char，紀錄打完10個字母，要讓車子跑。
  9. Keyboard, VGA variables, rst
  10. clk_d25，cursor 閃爍的速度。
  12. ir_send，表示輸出紅外線訊號。
  11. 其他，用來在 fpga 上面 debug。

  在這個 module裡面，我也開了一個 [5:0] adr [300:0] 來記錄文章。需要裡面的字母的時候就用 adr[word_cnt] 或 adr[output_word] 來傳入其他modules。

- FSM module
  這是 Finite State Machine 的 module，這個 module 傳入的 input 有 key_down last_change, been_ready, correct_n, wrong_cnt, word_cnt，output 則是 state

- CHECK module
  這個 module 是計算現在打到哪一個字還有有沒有打錯的 module，input 有 key_down, last_change, been_ready, state, word，output 有 word_cnt, correct_n, worng_words。

- char_per_min module
  這是用來計算打字速度的 module，input 有 word_cnt, state，output 則是 cpm1, cpm2, cpm3。這裡我回傳的是三個已經轉成 10 進位的 digits。

- mem_addr_gen module
  這是用來控制 VGA 的 module，input 有 state, h_cnt, v_cnt, word_num, wrong_cnt, letter, correct, word_cnt, cpm1, cpm2, cpm3, cursor，output 只有 red, green, blue。

  這應該是最大的 module，因為VGA的控制幾乎需要所有的變數。

  word_num 傳入的是 top.v 的 output_word，表示螢幕需要輸出第幾個字元。
  letter 傳入的是 top.v 的 adr[output_word]，表示輸出的是什麼字元。

  這個 module 裡面還有存我需要的圖片，到 VGA control 的時候再多做說明。

- IR_send module
  input ready, output out。用來控制紅外面的頻率，需要38khz。

- Other modules
  De_one_clk，裡面是 debounce, one_pulse, div_clk modules。
  vga_controller，上課提供的 VGA code。
  KeyboardDecoder，上課提供的 keyboard decoder。

3. 2 Game Design

首先，這個遊戲我設計了四個 state，分別是 WAIT, WORD, WRONG, FINISH

1. 在 WAIT 的時候，顧名思義，我們在等待開始，當按下 enter 鍵遊戲便會開始。
2. 在WORD 的時候，表示我們正在打字，並且現在沒有打錯任何字，一旦打錯便進到 WRONG state，如果打完了，就進到 FINISH state。
3. 在 WRONG 的時候，需要打 backspace 來刪除打錯的字，如果沒有的話，打錯的字數會持續增加，直到把錯字刪完才會回到 WORD state。
4. 在FINISH 的時候，會顯示這一場的 cpm，等到按下 enter 回到 WAIT state。

再來是有關文章讀取的部分，這一部分我是在 top.v 裡面完成的，我自定義每一個 character 的數字，在用 c++ 把文章轉成 verilog code。

```
cout << "assign adr[" << i << "] = ";
if (a[i] == ' ') cout << 26 << ";\n";
else if (a[i] == ',') cout << 27 << ";\n";
else if (a[i] == '.') cout << 28 << ";\n";
else if (a[i] == '\'') cout << 29 << ";\n";
else if (a[i] >= 'a' && a[i] <= 'z') cout << a[i] - 'a' << ";\n";
else cout << a[i] - 'A' + 30 << ";\n";
```

```
167     assign adr[0] = 38;
168     assign adr[1] = 26;
169     assign adr[2] = 6;
170     assign adr[3] = 14;
171     assign adr[4] = 19;
```

一部份生成的 code。
因為我的文章只有這些字元，所以只有定義這些，當然如果要更多的字元也是沒有問題的。

再來是 CHECK module，在這裡我會紀錄下一個要打的字，又或者現在有沒有打錯字、打錯幾個字。我先記錄了所有可能會用到的按鍵的 key code。

這裡的 input 有 word，表示現在要打的是什麼字，我在這裡用 cur_word 紀錄了現在要打的字的 key_code，圖下是其中的一部分。

```
always @(*) begin
    case (word)
        0 : cur_word = KEY_A;
        1 : cur_word = KEY_B;
        2 : cur_word = KEY_C;
        3 : cur_word = KEY_D;
        4 : cur_word = KEY_E;
        5 : cur_word = KEY_F;
        6 : cur_word = KEY_G;
        7 : cur_word = KEY_H;
        8 : cur_word = KEY_I;
        9 : cur_word = KEY_J;
        10 : cur_word = KEY_K;
```

在 word_cnt 的部分，只需要在 WORD state 紀錄，因為 cur_word 中，大寫和小寫的字母我都用一樣的 keycode，所以我需要多判斷 input 的 word 是大寫還是小寫，在判斷現在有沒有按著 shift，也就是 keydown[KEY_SHIFT] 等不等於一。

```
WORD : begin
    if (been_ready && key_down[last_change]) begin
        if (last_change != cur_word) next_word_cnt = word_cnt;
        else begin
            if (word >= 30 && !key_down[KEY_LEFT_SHIFT] && !key_down[KEY_RIGHT_SHIFT]) next_word_cnt = word_cnt;
            else if (word < 30 && (key_down[KEY_LEFT_SHIFT] || key_down[KEY_RIGHT_SHIFT])) next_word_cnt = word_cnt;
            else next_word_cnt = word_cnt + 1;
        end
    end else next_word_cnt = word_cnt;
end
```

接下來是 wrong_cnt 的部分，先討論 WORD state，我忽略了 last_change == enter, backspace, shift 按鍵，然後就是判斷現在打的字和需要打的字一不一樣，大致上和 word_cnt 相似。

```
WORD : begin
    if (record && key_down[last_change]) begin
        if (last_change == KEY_LEFT_SHIFT || last_change == KEY_RIGHT_SHIFT || last_change == KEY_BACK || last_change == KEY_ENTER) begin
            next_wrong = 0;
        end else if (last_change != cur_word) begin
            next_wrong = 1;
        end else begin
            if (word >= 30 && !key_down[KEY_LEFT_SHIFT] && !key_down[KEY_RIGHT_SHIFT]) next_wrong = 1;
            else if (word < 30 && (key_down[KEY_LEFT_SHIFT] || key_down[KEY_RIGHT_SHIFT])) next_wrong = 1;
            else next_wrong = 0;
        end
    end else next_wrong = 0;
end
```

再來是 WRONG state，我一樣忽略了last_change == shift, enter，和 WORD state中部一樣的地方就是需要判斷有沒有按下 backspace，有就減一，沒有的話 wrong_cnt就持續增加，因為要回到 WORD state的方法只有 wrong_cnt == 0。

```
WRONG : begin
    if (last_change == KEY_LEFT_SHIFT || last_change == KEY_RIGHT_SHIFT || last_change == KEY_ENTER) begin
        next_wrong = cnt_wrong;
    end else if (been_ready && key_down[last_change]) begin
        if (last_change == KEY_BACK) next_wrong = cnt_wrong - 1;
        else if (last_change == KEY_ENTER) next_wrong = cnt_wrong;
        else next_wrong = cnt_wrong + 1;
    end
    else next_wrong = cnt_wrong;
end
```

correct_n 的部分就是利用 wrong_cnt 等不等於零來判斷，0 代表沒錯，1代表錯了，FSM也是透過 correct_n來判斷要不要進到 WRONG state。

打字速度是打字遊戲最重要的一環，我用 char_per_min module 來記錄打字速度。我計算的方式是每 0.1 秒算一次，module 裡面有 counter, dsec, cpm 三個自己的變數，每當coutner 數到 10000000，也就是 0.1 秒，他就會歸零，然後 dsec += 1。

cpm 則是計算 char per minutes。

```
assign cpm = (state == 3'b100) ? (298 * 600 / (1+dsec)) : (word_cnt*600 / (1 + dsec));
```

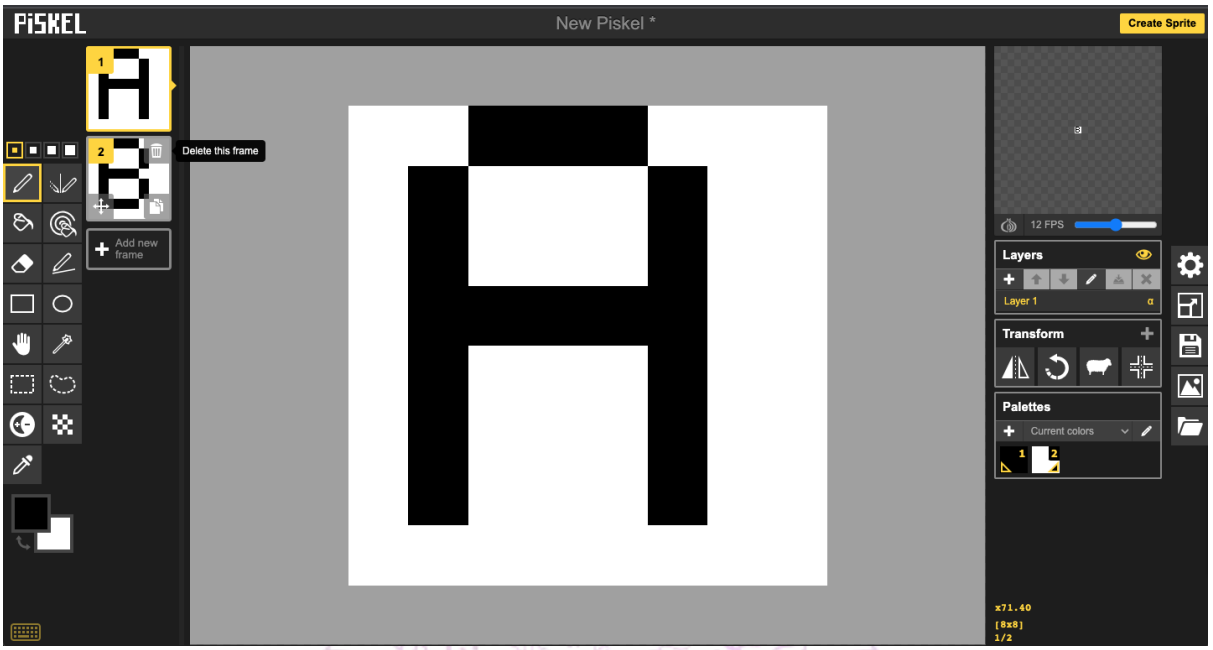
但因為我們希望輸出的時候三個是十進位的，所以就有了 cpm1, cpm2, cpm3 來記錄三個 digit 分別是多少。

```
assign cpm1 = (cpm)%10; // get the first digit of cpm
assign cpm2 = (cpm/10)%10; // get the second digit of cpm
assign cpm3 = (cpm/100)%10; // get the third digit of cpm
```

只有在 WORD, WRONG state 讀秒才會繼續，FINISH state會固定秒數，WAIT則是歸零。

3.3 VGA Control

這裡我要先介紹圖片的讀取，我沒有用上課教的方法把需要的照片轉成 coe 檔然後開 IP，這裡我直接在 mem_addr_gen module 裡面開 memory 存。我先用 piskel 畫出所有我需要的東西，像是字母，數字。
8*8 pixel 示意圖。



再將照片匯出，然後自己寫了一個讀取 png 檔案然後將所有 piskel 轉成 rgb representation 的 code，因為正常的 rgb representation 有8*3 bit，但因為 VGA 只允許 4*3 bit 的顏色，所以我把紅綠藍的數字都除以 16。我在用 c++ 生成 verilog code 丟到 module 裡面，module 裡面的 Upperletter, Lowerletter, otherletter, CPM 存的都是我需要的圖片，分別是大寫字母、小寫字母、標點符號以及CPM 還有數字。因為是一個 pixel 一個pixel存進去的，所以 code看起來很長，但主要是因為 CPM 的圖片是 32 * 32 bits 的，所以很多行，除了 synthesis 比較久以外看起來沒什麼異常。

我用這個方式的原因是因為用 coe 檔讀取的話畫面的解析度只有 320 * 240，但這麼做可以來到 640 * 480，需要高解析度的原因是因為文章中的字偏小，如果用 320 * 240 會幾乎看不清楚文章，而且轉成 coe 檔似乎會有糊邊的效果，讓字幾乎看不到。我這樣的實作方式可以清楚地看到文章，因為我是對每一個pixel做處理。
這是其中 CPM 我所畫的圖片，是輸出 CPM, 數字的時候用的。

mem_addr_gen module 最主要的功能是透過 h_cnt, v_cnt 指定需要的 vga_red, vga_green, vga_blue。

這裡有一些特別的變數，主要是用 h_cnt, v_cnt, letter, cpm來判斷現在要輸出 Upperletter, Lowerletter, otherletter, CPM中的哪一個圖片。
place1 -> Upperletter, place2 -> Lowerletter, place3 -> otherletter
cpms and CCPPMM are for CPM

```
assign place = (h_cnt % 8) + 8*(v_cnt % 8) + (64 * letter);
assign place2 = (h_cnt % 8) + 8*(v_cnt % 8) + (64 * (letter - 26));
assign place3 = (h_cnt % 8) + 8*(v_cnt % 8) + (64 * (letter - 30));
assign cpm_digit1 = (h_cnt % 32) + 32 * (v_cnt % 32) + (1024 * (3 + cpm1));
assign cpm_digit2 = (h_cnt % 32) + 32 * (v_cnt % 32) + (1024 * (3 + cpm2));
assign cpm_digit3 = (h_cnt % 32) + 32 * (v_cnt % 32) + (1024 * (3 + cpm3));
assign count_for_cpm = (h_cnt - 224) / 32;
assign CCPPMM = (count_for_cpm < 3) ? (h_cnt % 32) + 32 * (v_cnt % 32) + (1024 * count_for_cpm) : 0;
```

再來是影像輸出的部分，我只有分成兩個 state 來做不一樣的輸出處理，第一個是 FINISH state，在這裡我只會輸出最終的打字速度結果，我讓 CPM 還有數字出現在

螢幕正中間。

```
if (v_cnt >= 224 && v_cnt <= 256 && h_cnt >= 224 && h_cnt <= 416) begin
    if (count_for_cpm < 3) begin
        {red, green, blue} = CPM[CCPPMM];
    end else begin
        if (count_for_cpm == 5) {red, green, blue} = CPM[cpm_digit1];
        else if (count_for_cpm == 4) {red, green, blue} = CPM[cpm_digit2];
        else if (count_for_cpm == 3) {red, green, blue} = CPM[cpm_digit3];
    end
end
```

count_for_cpm 代表我要輸出 C, P, M 還是第幾位數字。

接下來是正文的部分，這裡我有傳入一個變數，就是 word_num，代表現在 h_cnt, v_cnt 數到的是哪一個字母的位置，在 top module 計算之後傳入的。

```
(h_cnt >= 80 && v_cnt >= 208) ? ( ((h_cnt - 80) / 8) + (60 * ((v_cnt-208) / 16))) : 0;
```

在知道要輸出什麼字之後，就像上面 CPM 一樣輸出，再配合其他的功能。

1. 正確的字。如果現在要輸出的字小於我要輸入的字，代表他已經被打對，這時候字會變成綠色。

```
end else if (word_num < word_cnt) begin
    if (letter <= 25) {red, green, blue} = (Lowerletter[place] == 12'b111111111111) ? 12'b111111111111 : 12'b000011110000;
    else if (letter >= 26 && letter <= 29) {red, green, blue} = (otherletter[place2] == 12'b111111111111) ? 12'b111111111111 : 12'b000011110000;
    else {red, green, blue} = (Upperletter[place3] == 12'b111111111111) ? 12'b111111111111 : 12'b000011110000;
```

2. 打錯字，因為我希望像 typeracer 一樣打錯字之後，紅色的部分會往後延伸，所有我有把 wrong_cnt 也傳進來，這樣我就可以用 correct_n, word_num, word_cnt, wrong_cnt 來計算現在的字的背景需不需要反紅。

```
if (correct && (word_num >= word_cnt) && (word_num < word_cnt + wrong_cnt)) begin
    if (letter <= 25) {red, green, blue} = (Lowerletter[place] == 12'b0) ? 12'b0 : 12'b111100110000;
    else if (letter >= 26 && letter <= 29) {red, green, blue} = (otherletter[place2] == 12'b0) ? 12'b0 : 12'b111100110000;
    else {red, green, blue} = (Upperletter[place3] == 12'b0) ? 12'b0 : 12'b111100110000;
```

3. Cursor。如果打字的時候有 Cursor 一定會更順暢更快，所以我們設計了一個 cursor，每 0.3 秒左右會閃一次，寫法就是取 not，非常直觀。

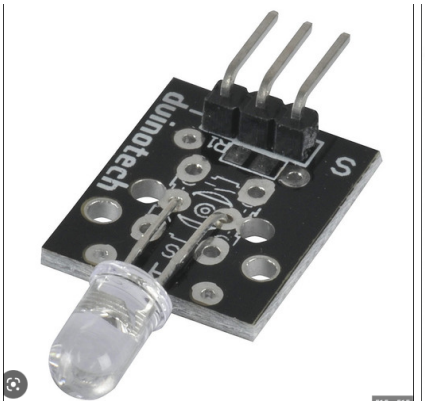
```
end else begin
    if (letter <= 25) {red, green, blue} = (cursor) ? ~Lowerletter[place] : Lowerletter[place];
    else if (letter >= 26 && letter <= 29) {red, green, blue} = (cursor) ? ~otherletter[place2] : otherletter[place2];
    else {red, green, blue} = (cursor) ? ~Upperletter[place3] : Upperletter[place3];
```

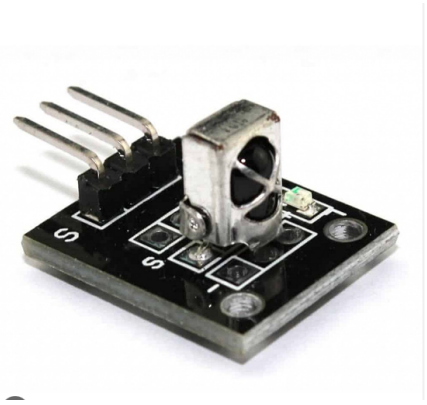
cursor 算是一個除頻過後的 clk_d25。

因為這是我們自己想的顯示方法，如果有問題的話還請教授助教提點。

3.4 IR Sensor

這裡我們是用紅外線發射模組還有 KY-022，也就是 IR-receiver。





發射端的部分，我們有一個變數 finish_ten_char，每當word_cnt % 10 == 0 的時候，就會讓 finish_ten_char = 1，然後傳到 IR_send module裡面，在 module 裡面就是reday 的訊號，不用擔心停留在同一個位置車子會一直往前，因為紅外線的訊號只有在 0 -> 1 的時候會偵測到，維持在 1 不會一直收到訊號。

module 的部分，變數有 tk38, tk38_half, cnt，可以知道原本的 clk 是 10^8hz，所以要除以大約 2631變成38k。所以 tk38 = 2631。tk38_half 的目的是為了讓 enable high 維持一段時間，所以多少影響並不大。主要是 tk38, cnt 讓output 可以維持在 38khz。

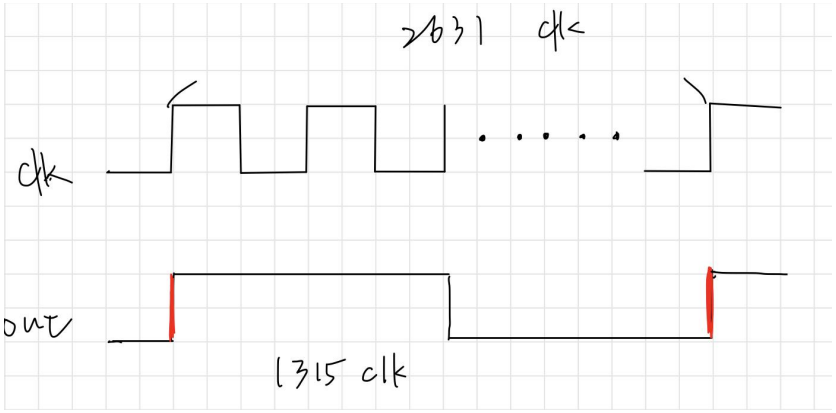
```
assign tk38 = 2631;
assign tk38_half = 1315;
assign cnt_next = cnt + 1;
```

因為 KY-022 裡面是預設接收 38khz 的紅外線，所以我在 module 裡面寫了一個 counter，讓 clk 變成 38khz。

```
always @(posedge clk) begin
    if (rst) cnt <= 0;
    else begin
        if (cnt >= tk38) cnt <= 0;
        else cnt <= cnt_next;
    end
end

assign out = ready ? ((cnt <= tk38_half) ? 1 : 0) : 0;
```

ready 是傳入 module 的訊號，代表說現在有沒有打到 10 char，需不需要傳送訊號給車子。可以注意到說因為 receiver 只有在訊號從 0 -> 1的那個瞬間才會偵測到，所以用 tk38_half 維持一點時間並不影響。可以看到下圖，只有在紅色的 posedge，receiver才會收到訊號。



3.5 Car Design

車子的設計用到了紅外線感測及遊戲本體這兩個module，遊戲本體會在每打十個字元時使用紅外線發送訊號告知車子，而每次車子收到訊號將會啟動counter，從而持續前進0.1秒的時間。

```
always @(*) begin
    if (!ir_signal) begin
        next_cnt = 10000000;
    end else if (cnt == 0) next_cnt = 0;
    else next_cnt = cnt - 1;
end
```

每次收到ir_signal都會將倒數時間設為0.1秒，而再倒數結束前車子都會持續前進

```
assign run = (cnt > 0) ? 2'b11 : 2'b00;
```

4. Problems

在這次Final Project中，我們一開始對使用紅外線或無線電接發訊號不知所措，我們嘗試思考一個Protocol使得能傳輸超過1bit的訊號卻毫無結果。這導致我們沒辦法讓遊戲本體所在的FPGA板直接傳送當前打過的字元數給車子。於是後來我們改成使用”每打10個字元跑0.1秒”的設計，這樣只要在每次打到10個字元時改變傳輸的訊號，而車子感測到這樣的改變再跑0.1秒。也幸好車子的控制只有前進或不前進兩種，我們才能輕鬆達成不寫protocol便能控制車子的設計。

5. What we have learnt

邏設實驗這門課算是這學期 loading 最重的課，但可以看到經過了一個學期，我們對於 verilog 已經可以算是熟悉，能夠在學期末用自己學到的東西做出一個自己有興趣的 Final Project，我們覺得非常的開心、有成就感。

我認為在這次的 Final Project 中，最有趣的是螢幕的顯示，我們研究了滿久的時間，看能不能克服 load coe 檔會出現的模糊的問題，我們認為這一個解法算是不錯，成果也可以看出畫面滿漂亮的，我覺得除了在前面利用老師教過的方法延伸寫出一個自己的遊戲架構外，能夠自己找到方法解決問題也是一個很好的學習。我們認為這次的 project 表現還算不錯，有達到我們預期的效果，也希望有機會的話，可以有一塊自己的板子自己摸索。

我們認為這次最可惜的部分在於紅外線的部分，除了沒有製作賽道，讓車子沒有走直線外，我們沒有做到紅外線分頻的功能，因為網路上關於 不同頻率的紅外線的 encode 的資料較少，我們沒有足夠的時間完成，如果有機會的話希望可以做出來，這樣就可以達到雙人對戰的效果，就等於從 typeracer 的 self practice 變成 和別人對戰，我們認為這樣的成果指日可待。