Final Project: Flappy Bird Game

106033233 周聖諺, 107091021 曾燕茹

2022-06-11

Contents

Final Project: Flappy Bird Game

106033233 資工大四 周聖諺 107091021 電資大三 曾燕茹

Proposal

本期末專題將設計與實現 flappy bird 遊戲，玩家在進行本遊戲時，需要控制畫面中的 bird，使其不要碰到障礙物 (圖中綠色水管)，方可累積分數；若碰到障礙物則遊戲結束。

本期末專題利用 FPGA 板實現遊戲，並用 VGA 連接螢幕，呈現出 bird 以及障礙物的畫面；遊戲一開始時，VGA 螢幕畫面會呈現開始畫面，等偵測到玩家按下鍵盤後即開始遊戲，進入遊戲畫面。

當遊戲開始時，bird 會自動下降其飛行高度，此時玩家需要透過控制鍵盤讓畫面中的 bird (按一下按鍵往上飛一些，若沒有持續按鍵盤，則 bird 飛行高度會再次下降) 躲避障礙物 (圖中綠色水管) 方可進行遊戲。

Overview

以下是架構圖，`top` 為本專案的頂層模組，`global` 為全域變數。

- `global`
- `top`
    - `clock_divisor`
    - `onepulse`
        * `debounce`
    - `KeyboardDecoder`
        * `KeyboardCtrl_0`
        * `OnePulseKB`
    - `game`
        * `bg_ctrl`
            · `bg_mem_addr_gen`
            · `blk_mem_gen_bg_big`
        * `pipe_ctrl`
            · `pipe_mem_addr_gen`
            · `blk_mem_gen_pipe`
        * `bird_ctrl`

            - bird_mem_addr_gen
            - bird_pos_ctrl
            - blk_mem_gen_bird
        * ctrl
        * scence_ctrl
            - score2font
              - dec2font
            - text_ctrl
              - font_ctrl
                - font_mem_addr_gen
                - blk_mem_gen_font
- audio_ctrl
    * fre_div
    * song_ctrl
        - up_counter
        - fruit_pudding_mem
        - angry_bird_mem
        - flap_mem
        - bump_mem
    * note_gen
    * speaker_control
        - frequency_divider
- vga_controller
- dec_disp
    * segment7
    * display_7seg
        - segment7_frequency_divider

Design Specification

Module: global

Global variables

Module: top

Inout: PS2_DATA, PS2_CLK

Input: clk, rst, btn_u, btn_m, btn_d, btn_r, btn_l

Output: [COLOR_BIT_N-1:0] vgaRed, [COLOR_BIT_N-1:0] vgaGreen, [COLOR_BIT_N-1:0] vgaBlue, [LED_N-1:0] leds,[SEGMENT_7_DISPALY_DIGIT_N-1] d_sel, [0:SEGMENT_7_SEGMENT_N-1:0] d_out, hsync, vsync, mclk, lrck, sck, sdin

Module: clock_divisor

Input: clk

Output: clk1, clk21, clk22

Module: onepulse

Input: clk, rst, push

Output: push_onepulse, push_onepulse_long, push_debounced, push_debounced_long, push_sig, push_sig_long

Module: debounce

Input: rst, clk, push

Outpout: push_debounced

Module: KeyboardDecoder

Inout: PS2_DATA, PS2_CLK

Input: rst, clk

Output: [511:0] key_down, [8:0] last_change, key_valid

Module: OnePulseKB

Input: signal, clock

Output: signal_single_pulse

Module: display_7seg

Output: [0:3] d_sel, [7:0] d_out

Input: clk, rst, [7:0] d0, [7:0] d1, [7:0] d2, [7:0] d3

Design Implementation

Module: global

The global variables are used across the whole project.

Module: top

此為本遊戲的頂層模組，此模組調用 `clock_divisor` 為背景滾動、水管滾動、小鳥移動和拍動翅膀提供 clock 作為 trigger。並將這些 clock 傳進模組`game`，並依據 VGA 座標(`h_cnt`, `v_cnt`)回傳 pixel 的資料，再傳進模組`vga_controller`，並用模組`dec_disp`使分數同步顯示於七段顯示器上。

Module: clock_divisor

為背景滾動、水管滾動、小鳥移動和拍動翅膀提供 clock 作為 trigger。

Module: onepulse

用一個計數器來計算按鈕按下的 clock cycles，若按鈕按下的時間較長，會觸發`push_onepulse_long`，反之，若按鈕按下的時間較短，則會觸發`push_onepulse`。

I use a counter to count the clock cycles during the button is pressed. If the counting exceed a threshold, it will trigger a long press pulse `push_onepulse_long`. Otherwise, it will trigger a click pulse `push_onepulse`.

Module: debounce 在每次按按鈕的時候，此模組會延遲 4 個 clock cycle 且產生一個" debounce pulse"。在模組中，使用 4 個 registers 來達成延遲 4 的 clock cycle，並在此 4 個 registers 中的值皆為 1 的時候，輸出一個 pulse。

For each click, the module will delay 4 clock cycle and then raise the debounce pulse. I use 4 registers to represent the delay state and send a pulse while 4 registers are all 1s.

Module: KeyboardDecoder

此為 lab 8 助教提供的 keyboard 模組之一。

Module: KeyboardCtrl_0 此為 lab 8 助教提供的 IP。

Module: OnePulseKB 此為 lab 8 助教提供的 keyboard 模組之一。

Module: game

此為控制遊戲邏輯的主要模組，主要由五個模組構成，分別是：`bg_ctrl`、`pipe_ctrl`、`bird_ctrl`、`ctrl`、`scence_ctrl`。接下來將於本節詳細描述。

Module: bg_ctrl

此模組的作用在於控制背景滾動，我們用 `bg_mem_addr_gen` 和 `blk_mem_gen_bg_big` 來實現此功能。其中 `bg_mem_addr_gen` 產生對應該 VGA 座標所需的 pixel 的 address，並放入 `blk_mem_gen_bg_big` 將對應 address 的背景圖資料讀出。

Module: bg_mem_addr_gen `bg_mem_addr_gen` 依據滾動速率在每單位時間都將背景圖片向左 shift 一個單位，並依據輸入的 VGA 座標輸出相對應的 pixel 的 address。

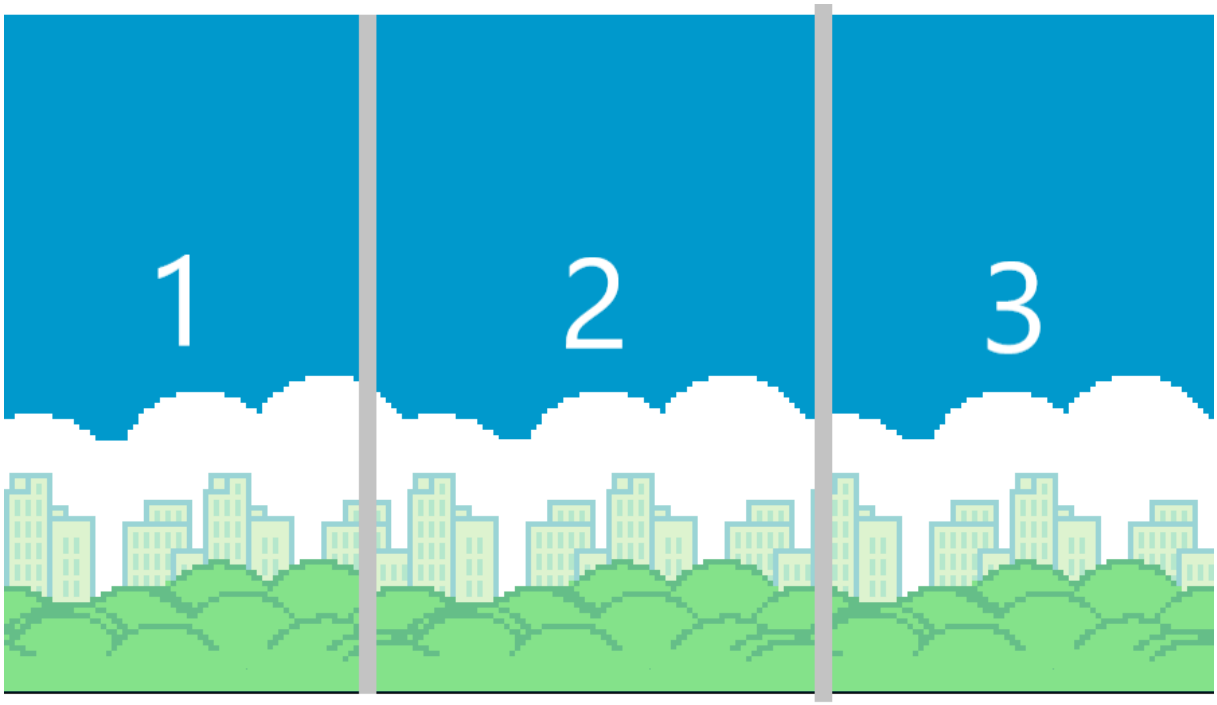
Module: blk_mem_gen_bg_big 此為 Vivado 內建的 RAM IP 模組，我們將背景圖片如下放進 RAM 中儲存並讀取。

Module: pipe_ctrl

此模組控制畫面中的綠色水管的長度與水管間的間隔，以及在螢幕畫面中移動的方式。其中，`pipe_mem_addr_gen` 控制了水管的移動方式、長度及間隔，並依據 VGA 輸入所需要的座標輸出相對應所需的 pixel 資料。而 `blk_mem_gen_pipe` 則是依據輸入的記憶體地址輸出相對應的 pixel 資料。兩者組合在一起就可以做出水管移動並且有不同高度與間隔的效果。

另外，`dout`輸出對應 VGA 座標的 pixel 資料，`px_valid` 則輸出在此 VGA 座標是否需要顯示出此 pixel，會如此設計是因為水管圖片周圍其實會有一層非透明的框，因此我們就直接針對方框的顏色予以剃除，同時必須在不用畫出水管的區域讓水管這一個圖層保持透明，以方便疊圖。

Module: pipe_mem_addr_gen 此模組的功能在於控制水管的長度、間隔與移動方式，並對應輸入的 VGA 座標輸出 address。由於在畫面中只會出現三根水管，因此水管每移動 1/3 個螢幕就必須讓下一根水管出現，然而，每根水管必須從右到左將整個螢幕掃過一次才會消失，因此其實我們必須設計一個 shift register 如下，其中 `pipe_gaps` 儲存水管間的間隔，而 `pipe_lens` 儲存水管的高度，每當水管走過 1/3 個螢幕時，就將下一個水管 shift 進來。




```
reg [CNT_BITS_N-1:0] pipe_gaps [`PIPE_NUM-1:0];
reg [CNT_BITS_N-1:0] pipe_lens [`PIPE_NUM-1:0];

pipe_gaps[14] <= pipe_gaps[0];
pipe_gaps[0] <= pipe_gaps[1];
.
.
.
pipe_gaps[13] <= pipe_gaps[14];

pipe_lens[14] <= pipe_lens[0];
pipe_lens[0] <= pipe_lens[1];
.
.
.
pipe_lens[13] <= pipe_lens[14];
```

而水管的移動速度是依據 input `clk_scroll`的跳動速率決定，每一個 clock period pipe 皆會向左移動一個單位。同時，因為水管每走 1/3 個螢幕就必須 shift 一個新的水管進來，所以我們將螢幕從左到右切分成三等份，第 1 等分顯示第 0 個水管，也就是`pipe_gaps[0]`及`pipe_lens[0]`，第二等分顯示第 1 個水管，第三等份顯示第 2 個

水管。每個水管在超出該等分所顯示的範圍之後 (其實就是水管每走完 1/3 個螢幕時)，就會 shift 到左邊下一個等分繼續顯示，並且 shift register 會 shift 進一個新的水管到第三等份的螢幕中。以下為當水管的位置 `pos` 到達第一等分`h_h_cnt < pos + PIPE_WIDTH_CNT`的位置時，若 VGA 的座標分別為`h_h_cnt`與`h_v_cnt`的話，需要水管圖片的`(addr_h_cnt, addr_v_cnt)`座標的 pixel。

```
end else if(h_h_cnt > pos && h_h_cnt < pos + PIPE_WIDTH_CNT) begin
    if(h_v_cnt < pipe_lens[1]) begin
        addr_h_cnt <= h_h_cnt - pos;
        addr_v_cnt <= pipe_lens[1] - h_v_cnt;
        valid <= 1'b1;
    end else if(h_v_cnt > pipe_lens[1] + pipe_gaps[1]) begin
        addr_h_cnt <= h_h_cnt - pos;
        addr_v_cnt <= h_v_cnt - pipe_lens[1] - pipe_gaps[1];
        valid <= 1'b1;
    end else begin
        addr_h_cnt <= 0;
        addr_v_cnt <= 0;
        valid <= 1'b0;
    end
```

值得注意的是，其實做到這樣只能確保水管在進場和滑動的時候沒有問題，但是這個做法卻沒有考慮到水管除如何出場，因此，我們而外在多加了一個第 0 等分，目的在於我們希望當水管在走完第一等分的螢幕畫面必須出場時，會接續由第零等分顯示並出場。

```
if(is_pass_first_pipe && h_h_cnt > 0 && PIPE_WIDTH_CNT > (`PHASE1_CNT -
    pos) && h_h_cnt < PIPE_WIDTH_CNT - (`PHASE1_CNT - pos)) begin
    if(h_v_cnt < pipe_lens[0]) begin
        addr_h_cnt <= h_h_cnt + (`PHASE1_CNT - pos);
        addr_v_cnt <= pipe_lens[0] - h_v_cnt;
        valid <= 1'b1;
    end else if(h_v_cnt > pipe_lens[0] + pipe_gaps[0]) begin
        addr_h_cnt <= h_h_cnt + (`PHASE1_CNT - pos);
        addr_v_cnt <= h_v_cnt - pipe_lens[0] - pipe_gaps[0];
        valid <= 1'b1;
    end else begin
        addr_h_cnt <= 0;
        addr_v_cnt <= 0;
        valid <= 1'b0;
    end
```

而畫面中的上下兩根水管其實就將水管的 pixel 的垂直座標上下顛倒就行。

Module: blk_mem_gen_pipe 此為 Vivado 內建的 RAM IP 模組，我們將水管圖片如下放進 RAM 中儲存並讀取。

Module: bird_ctrl

此模組依據玩家的輸入，控制小鳥的飛行位置，並同時實現小鳥拍打翅膀的動畫與模仿地心引力的下墜。dout輸出對應 VGA 座標的 pixel 資料，px_valid 則輸出在此 VGA 座標是否需要顯示出此 pixel，若px_valid=0則此圖層為透明無色，同時，與繪製水管的考量相仿，因為小鳥的圖片周圍也有一層非透明的方框，因此必須檢測該方框顏色並予以剔除。

Module: bird_mem_addr_gen 此模組會依據目前小鳥的位置(pos_h_cnt, pos_v_cnt)產生對應 pixel 的 address，同時會依據clk的頻率更新小鳥拍動翅膀的圖片。具體的作法為，由於小鳥拍動翅膀的動畫是由三個 frame 組成，我們會依據clk更新小鳥處在不同的 frame 始知看起來像在拍動翅膀。另外，我們依據(pos_h_cnt, pos_v_cnt)來判斷小鳥在畫面上所處的位置，若 VGA 的座標位置落在顯示小鳥的區域上的話，則輸出對應的 pixel。而具體做法就是檢查 VGA 座標是否落在以 (pos_h_cnt, pos_v_cnt) 作為顯示小鳥區域的左上 anchor，往右及往下小鳥的高度及寬度所框起來的區域，即為h_h_cnt >= pos_h_cnt && h_h_cnt < pos_h_cnt + BIRD_WIDTH_CNT && h_v_cnt >= pos_v_cnt && h_v_cnt < pos_v_cnt + BIRD_HEIGHT_CNT。

```
always@(posedge clk) begin
    if(rst) begin
        phase <= 0;
    end else begin
        if(phase == 0) begin
            phase <= 10;
        end else if(phase == 10) begin
            phase <= 20;
        end else if(phase == 20) begin
            phase <= 0;
        end
    end
end

always@(*) begin
    pixel_addr <= addr_h_cnt % BIRD_WIDTH_CNT + phase + BIRD_WIDTH_CNT
       * 3 * (addr_v_cnt % BIRD_HEIGHT_CNT);
end

always@(*) begin
    if(h_h_cnt >= pos_h_cnt && h_h_cnt < pos_h_cnt + BIRD_WIDTH_CNT &&
        h_v_cnt >= pos_v_cnt && h_v_cnt < pos_v_cnt + BIRD_HEIGHT_CNT)
           begin
        addr_h_cnt <= h_h_cnt - pos_h_cnt;
        addr_v_cnt <= h_v_cnt - pos_v_cnt;
        valid <= 1'b1;
    end else begin
        addr_h_cnt <= 0;
        addr_v_cnt <= 0;
        valid <= 1'b0;
```

```
29      end
30  end
```

Module: bird_pos_ctrl 此模組依據使用者的輸入，控制小鳥的位置。使用者只需要按住空白鍵，小鳥就會一直往上飛直至碰到螢幕的頂部，放開空白鍵的話小鳥就會以自由落體的速度下墜，也就是說會以時間平方成正比的速度下墜。

我們的做法是，若 input `btn_fly` 為 1，也就是空白鍵被按下時，則每過一個 `clk_move` 小鳥的垂直座標`pos_v_cnt`會減一；若否，則小鳥的垂直座標`pos_v_cnt`會以放開空白鍵的 clock 數量的平方增加。code 邏輯如下。

```
1  if(btn_fly && ~is_dead && pos_v_cnt > 0) begin
2      if(pos_v_cnt - 2 <= 1) begin
3          pos_v_cnt_next <= 0;
4      end else begin
5          pos_v_cnt_next <= pos_v_cnt - 2;
6      end
7      drop_count_next <= 0;
8      is_clicked_next <= 1;
9  end else if((is_dead && pos_v_cnt_next < HEIGHT_CNT) || (~btn_fly &&
       is_clicked && pos_v_cnt < HEIGHT_CNT)) begin
10     pos_v_cnt_next <= pos_v_cnt + drop_count * drop_count / 32;
11     drop_count_next <= drop_count + 1;
12 end
```

Module: blk_mem_gen_bird 此為 Vivado 內建的 RAM IP 模組，我們將小鳥的圖片如下放進 RAM 中儲存並讀取。

Module: ctrl

此模組有四大功能

1. 判斷小鳥是否有撞上水管：讀取當前小鳥的位置和在畫面中第一等分水管的位置，並檢查小鳥的 pixel 是否和水管的 pixel 是否同時 `px_valid=1`，若有，則判斷小鳥已經撞上水管，並將變數 `is_bump=1`；若否，則將變數 `is_bump=0`。

2. 融合圖層：本遊戲有四個圖層，顯示的優先順序從高到低分別為文字 -> 小鳥 -> 水管 -> 背景，若叫高優先的圖層的 px_valid=1則會以該圖層蓋掉較低優先順率的圖層。
3. 控制畫面場景：使用 push_onepulse_d 及 is_bump 來控制狀態機，狀態機有三個狀態，分別是 a. 開始狀態 b. 遊戲狀態 c. 結束狀態

    a. 開始狀態：is_start=0、is_dead=0、is_game_over=0，若遇到push_onepulse_d=1，也就是 Enter 鍵被按下，則跳到遊戲狀態
    b. 遊戲狀態：is_start=1、is_dead=0、is_game_over=0，若遇到小鳥撞上水管is_bump=1，則跳到結束狀態。
    c. 結束狀態：is_start=0、is_dead=1、is_game_over=1，若遇到push_onepulse_d=1，也就是 Enter 鍵被按下，則跳到開始狀態

4. 計算分數：若水管的位置 pos <= 0(代表水管的左側已經碰到螢幕的左側) 且小鳥沒有死is_dead=0，則分數加一，若(~is_start)&& (~is_dead) 為真，也就是在開始狀態，分數則歸零。

Module: scence_ctrl

此模組依據 input is_start、is_dead 判斷狀態機所處的狀態，並顯示相對應的文字，

a. 開始狀態：當is_start=0、is_dead=0，在畫面正中央顯示：**FLAPPY BIRD**
b. 遊戲狀態：當is_start=1、is_dead=0，在畫面頂部顯示玩家分數：**SCORE: 0001**
c. 結束狀態：is_start=0、is_dead=1，，在畫面正中央顯示兩行：第一行為固定文字：**GAME OVER**，第二行為玩家分數：**SCORE: 0001**

Module: score2font 此模組能將 4 位數的十進位數字score轉成font_ctrl的文字代碼d0_font、d1_font、d2_font和d3_font，分別是千位、百位、十位以及個位數。具體作法其實就只是用除法和 mod 就可以達成。

Module: dec2font 此模組能將二進位數字dec轉成font_ctrl模組的文字代碼font。

Module: text_ctrl

此模組擴展 font_ctrl 的功能，給定一個座標(pos_h_cnt, pos_v_cnt)將多個文字alphabets_1d於指定座標水平顯示。也就是說，此模組提供類似於文字方塊的功能，將文字內容水平顯示。具體的作法是檢查 VGA 座標 (h_cnt, v_cnt) 看看座標是落在水平文字的第幾個文字上，若是落在第二個文字上，就將 font_ctrl 的 input 換成第二個文字的座標(text_h_cnt, text_v_cnt)和文字編碼alphabet。代碼如下

```
always@(posedge clk) begin
    if(h_h_cnt >= pos_h_cnt && h_v_cnt >= pos_v_cnt) begin
```

```
        if((h_h_cnt - pos_h_cnt) < FONT_WIDTH_CNT * ALPHABET_N) begin
            text_h_cnt <= pos_h_cnt + FONT_WIDTH_CNT * alpha_idx;
            text_v_cnt <= pos_v_cnt;
            alphabet <= alphabets[alpha_idx];
        end
    end
end
```

Module: font_ctrl

此模組可以依據指定位置(pos_h_cnt, pos_v_cnt)將單個文字alphabet在指定位置畫出，其中dout輸出對應該 VGA 座標(h_cnt, v_cnt)的 pixel 資料，而px_valid輸出該 VGA 座標是否會有文字 pixel，也就是文字圖層在該 VGA 座標是否為透明。

Module: font_mem_addr_gen 此模組依據輸入的文字座標位置(pos_h_cnt, pos_v_cnt)並將此座標當作文字框的左上錨點輸出 input alphabet對應文字圖片的 address pixel_addr。具體作法是先將文字圖片切成 7 * 8 個方格，其中每個方格為 8 * 8 pixels，並從左上到右下依序從小到大編號，當 VGA 座標若位於([pos_h_cnt, pos_h_cnt+8), [pos_v_cnt, pos_v_cnt+8))區域內，則將 input alphabet 對應的文字的 address 輸出。

```
always@(*) begin
    blk_h_cnt <= alphabet % FONT_NUM_COL;
    blk_v_cnt <= (alphabet / FONT_NUM_COL) % FONT_NUM_ROW;
end

always@(*) begin
    pixel_addr <= (addr_h_cnt + FONT_WIDTH_CNT * blk_h_cnt) + (
        FONT_WIDTH_CNT * FONT_NUM_COL * (addr_v_cnt + FONT_HEIGHT_CNT *
        blk_v_cnt));
    // pixel_addr <= addr_h_cnt + (FONT_WIDTH_CNT * FONT_NUM_COL *
        addr_v_cnt);
end

always@(*) begin
    if(h_h_cnt >= pos_h_cnt && h_h_cnt < pos_h_cnt + FONT_WIDTH_CNT &&
       h_v_cnt >= pos_v_cnt && h_v_cnt < pos_v_cnt + FONT_HEIGHT_CNT)
          begin
        addr_h_cnt <= (h_h_cnt - pos_h_cnt) % FONT_WIDTH_CNT;
        addr_v_cnt <= (h_v_cnt - pos_v_cnt) % FONT_HEIGHT_CNT;
        valid <= 1'b1;
    end else begin
        addr_h_cnt <= 0;
        addr_v_cnt <= 0;
        valid <= 1'b0;
    end
end
```

Module: blk_mem_gen_font 此為 Vivado 內建的 RAM IP 模組，我們將文字圖片如下放進 RAM 中儲存並讀取。

Module: audio_ctrl

此模組主要功能是將所有聲音的模組結合在一起，變成

Module: fre_div

Module: song_ctrl

Module: up_counter

Module: fruit_pudding_mem

Module: angry_bird_mem

Module: flap_mem

Module: bump_mem

Module: note_gen

Module: speaker_control

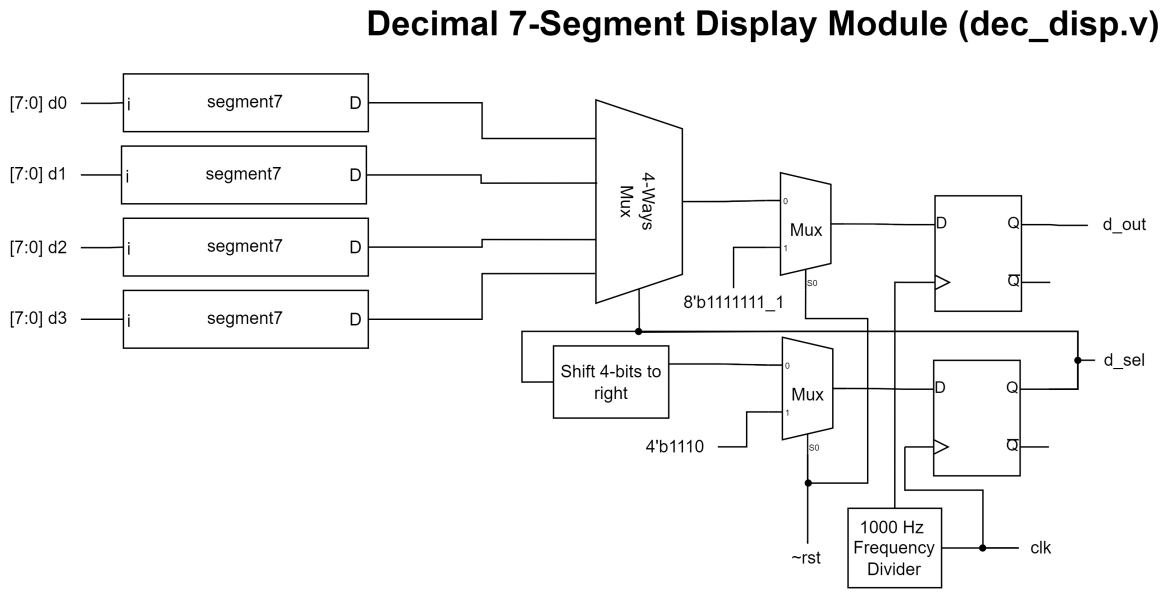
Module: frequency_divider

Module: vga_controller

Module: dec_disp

此模組在七段顯示器上會顯示`d0`, `d1`, `d2`, 和 `d3` 的二進制數。此模組會將二進制的`d0`, `d1`, `d2`, 和 `d3` 轉換成七段顯示模式，並將這些訊號傳入`display_7seg`模組，即可將`d0`, `d1`, `d2`, 和 `d3`以七段顯示顯示出來。

The module shows the binary number `d0`, `d1`, `d2`, and `d3` on the 7-segment display. It convert the binary number `d0`, `d1`, `d2`, and `d3` to 7-segment pattern and then, put them into the module `display_7seg` to show the number on the 7-segment display.



Module: segment7 此模組將 4-bit 二進制數字轉換為七段顯示器

Convert 4-bit binary number to 7-segment display with switch-case syntax.

Module: display_7seg 當要控制七段顯示器時，由於每次只能控制一個位數，因此將此模組設計為以四位數為輸入，並在每次時鐘訊號上升的時候，即在顯示器上顯示一個位數。每當時鐘訊號上升時，此模組會控制`d_sel`切換到不同的位數，並顯示相對應的數字。例如: 當時鐘訊號第一次上升時，模組會設定`d_sel = 4'b1110` 和 `d_out = d0`；時鐘訊號第二次上升時，模組則會設定`d_sel = 4'b1101` 和 `d_out = d1` 等等。

Since we can only control one digit of the 7-segment display each time, I design a module that takes the 4-digit patterns as input and shows the 1 digit on the display when the clock raises. Whenever the clock raises, the module will switch the control d_sel to different digit and shows the corresponding digit. Take an example, when the first clock raise occur, the module will set `d_sel = 4'b1110` and

`d_out = d0`. As for second clock pulse, the module will output `d_sel = 4'b1101` and `d_out = d1` and so on.

Module: segment7_frequency_divider 為了要產生 1000 Hz 的時鐘訊號，在此模組使用了變數`counter_in` 和 `counter_out` ，並使變數從 0 數到 50000。變數`counter_in` 會儲存下一個時間狀態要用到的值，並在時鐘訊號上升時將此值傳給`counter_out`。在每次時鐘訊號上升的時候，才會觸發記數，因此在此模組需要從 0 到 50000 的記數，且在每兩次 clock pulses ，記數會多 1。

To generate the 1000 Hz clock, I use variables `counter_in` and `counter_out` to count from 0 to 50000. The counter_in will store the value for the next time step and pass the value to the `counter_out` when the clock raises. The reason why we need 50000 counting is each counting is triggered only when the clock raises, so the circuit will count 1 more for every twice clock pulses.

Conclusion