Final Project: Flappy Bird Game

106033233 周聖諺, 曾燕茹

2022-06-11

Contents

Final Project: Flappy Bird Game

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

Proposal

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

Module: clock_divisor

Module: onepulse

Module: debounce

Module: KeyboardDecoder

Inout PS2_DATA, PS2_CLK

Input rst, clk

Module: OnePulseKB

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

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

Module: clock_divisor

Module: onepulse

Module: debounce

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。

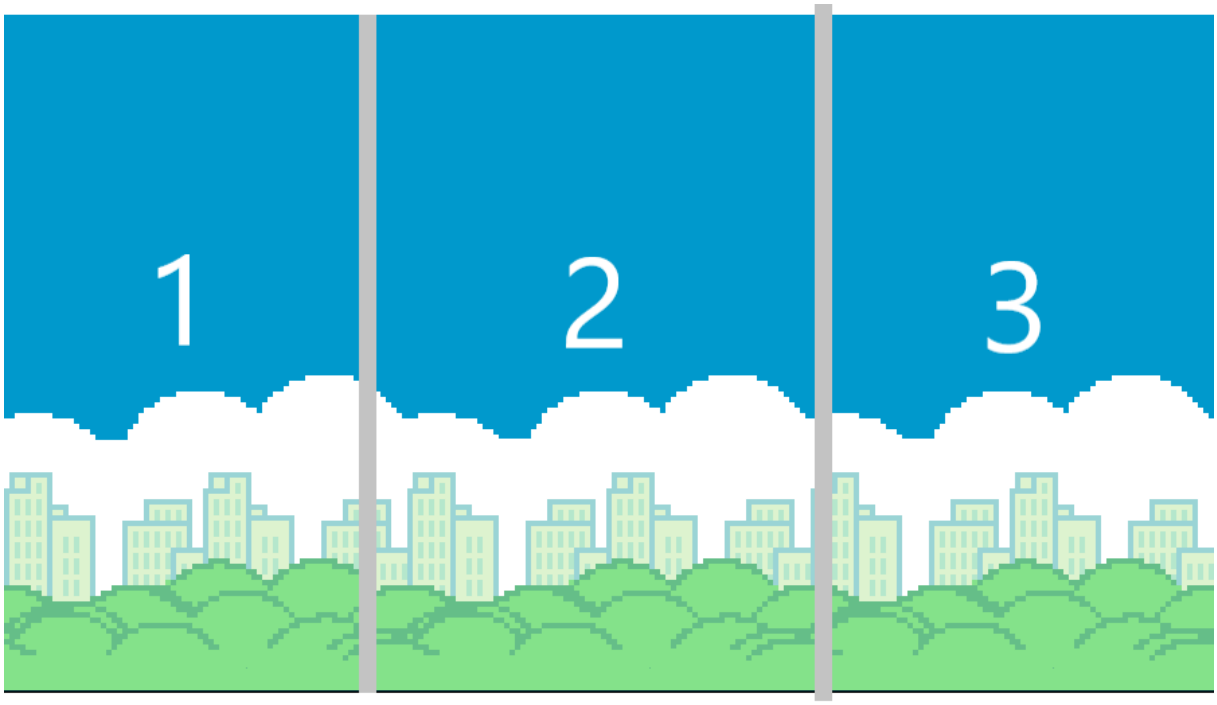
Module: blk_mem_gen_bg_big 此為 Vivado 內建的 RAM IP 模組，我們將背景圖片如下放進 RAM 中儲存並讀取。

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。

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。

Module: bird_mem_addr_gen

Module: bird_pos_ctrl

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

Module: scence_ctrl

Module: score2font

Module: dec2font

Module: text_ctrl

Module: font_ctrl

Module: font_mem_addr_gen

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

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

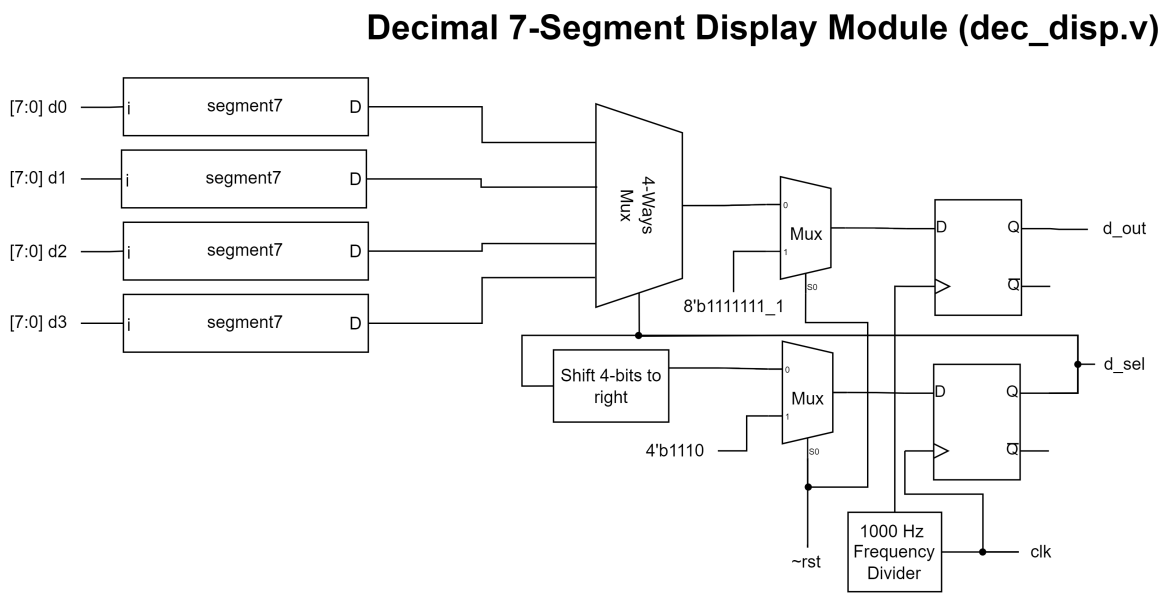
Module: speaker_control

Module: frequency_divider

Module: vga_controller

Module: dec_disp

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

Module: display_7seg 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 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