Final Project_Team4
Doodle Jump

submitted by 鄧華予 106062106

李思佑 Szuyu Lee 106062328

CS201401 Hardware Design and Lab

National Tsing Hua Univeristy

Motivation

Doodle Jump(released in 2009) is a popular game in our childhood, when IPhone just started to become prevalent. We were looking for some games that are easy to be programmed and this was what firstly came up.

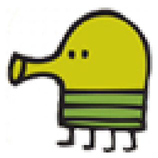
Introduction

Adapted from the phone-based game Doodle-Jump.
We keep most of the elements in the original game.

Goal: Player need to guide the doodle up to a series of platforms without falling.
Victory is set as score=9999.

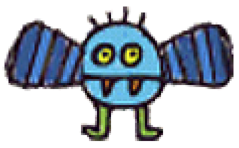
Characters:

Doodle:



Represents the player, can shoot with infinite bullet, but only one bullet is allowed to present on the map. Controlled by keyboard.

Monster:



Represents the villain of the game, kill the player upon contact.
A player can destroy monsters by shooting bullet towards it. Monsters spawned every 500 points after its death.

Environment:

Map:

map_width = 640, map_height = 480

Platform:

Platform are generated randomly, initially they are immobile. However, when player scores 5000, platforms will start to move horizontally.

User Manual:

keyboard:

A->left

D->right

1,2,3,4,6,7,8,9->shooting(in 8 directions)

Enter,Esc->progress in the game

FPGA board:

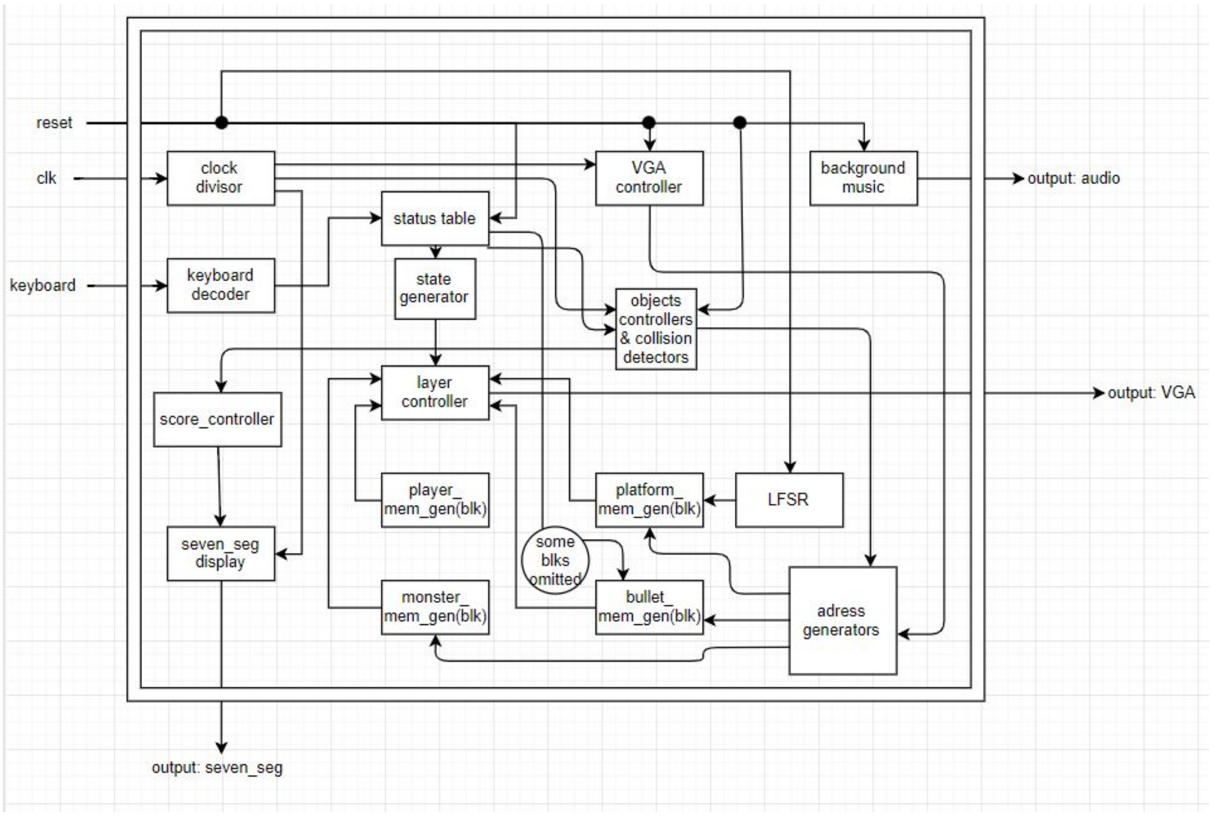
7-segment: current score

rightmost switch: turn on(off) music

LED: will shine in patterns after victory

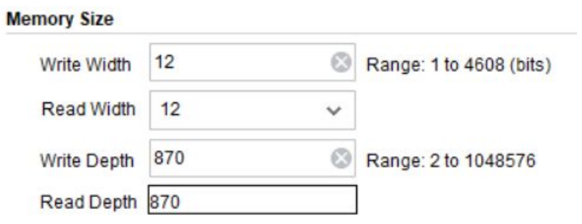
Game Design

Rough block diagram



1.block memory generators

We use memory IPs to create multiple block memory generators, block memory generators take a lot of space of RAMs and the size of it is obtained by depth*width.

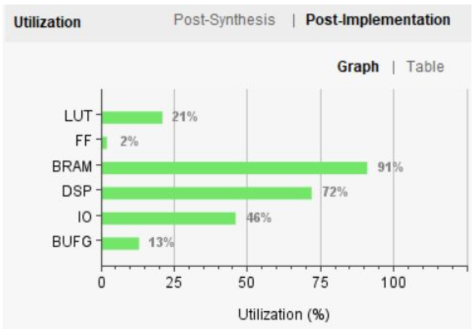


Memory Size

Write Width	12	Range: 1 to 4608 (bits)
Read Width	12	
Write Depth	870	Range: 2 to 1048576
Read Depth	870	

(ex. the block memory of platform take 12*(58*15)=10,400bits,1/180 of total)

We use 14 block generators in total which consumes



91% of the total usage(1800,000bits). We've calculated take the best advantage of usage by cutting big pictures into smaller pieces. But to have clearer texts on the screen, we need to do some sacrifice.

DOODLE JUMP

ex.

(440*56)*12=295,680 295,680/1800,000=16.4%

2. controllers(monster,player,bullet)

a) player controller—

The module is designed to handle player movement and position data. It takes two input from keyboard_decoder, these affects player's speed in horizontal direction. The module will update player's position by its current speed, however, in the case that collision occurs, position will updated as being bounced instead. Collision detector gives a signal "hit" to control the process.

b) monster controller—

This module is used to handle monster spawning, movement and position data. Monster is spawn by 500 score interval, and will automatically move horizontally until it reached map bottom or being hit by bullet.

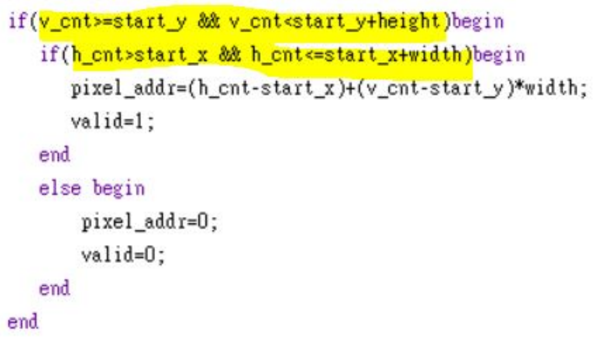
c) bullet controller—

This module records bullet existence, speed and position. There can only be at most one bullet on a map, thus the player has to wait the bullet to went out of the map to reload.

To determine if there is a bullet exist on the map, two register exist_x, exist_y are used and only when both are "high" the bullet is considered exist. After player fires a bullet, exist_x and exist_y are set to "high", and will return to "low" when reaching the edge of the map.

3. memory address generators

it give address to block memory generator , at clk_25MHZ
tells it which part of the memory should be read.

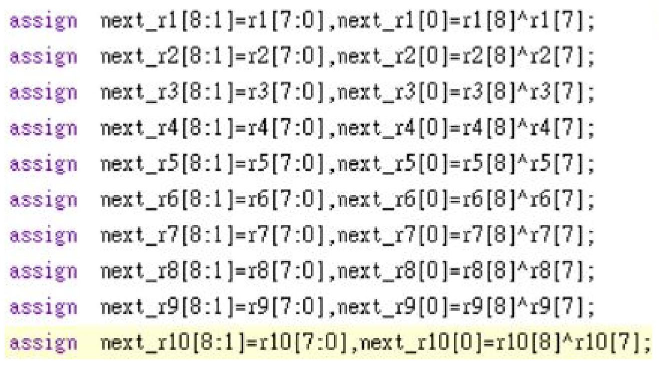


```
if(v_cnt>=start_y && v_cnt<start_y+height)begin
   if(h_cnt>start_x && h_cnt<=start_x+width)begin
      pixel_addr=(h_cnt-start_x)+(v_cnt-start_y)*width;
      valid=1;
   end
   else begin
       pixel_addr=0;
       valid=0;
   end
end
```

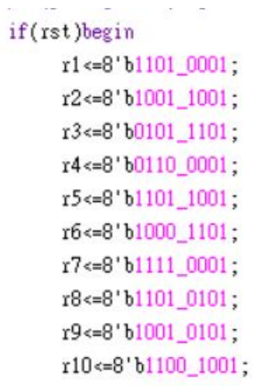
as the method above, it reads in the height*width region, otherwise it reads 0.
(In all our figures when addr==0 ----> pixel==12'hfff)

4. random platform generator(LFSR)

10 registers are created to
every time the when platform's position > screen_height
'new' platform comes from top of screen, with random horizontal position.
We use LFSR here.



```
assign  next_r1[8:1]=r1[7:0],next_r1[0]=r1[8]^r1[7];
assign  next_r2[8:1]=r2[7:0],next_r2[0]=r2[8]^r2[7];
assign  next_r3[8:1]=r3[7:0],next_r3[0]=r3[8]^r3[7];
assign  next_r4[8:1]=r4[7:0],next_r4[0]=r4[8]^r4[7];
assign  next_r5[8:1]=r5[7:0],next_r5[0]=r5[8]^r5[7];
assign  next_r6[8:1]=r6[7:0],next_r6[0]=r6[8]^r6[7];
assign  next_r7[8:1]=r7[7:0],next_r7[0]=r7[8]^r7[7];
assign  next_r8[8:1]=r8[7:0],next_r8[0]=r8[8]^r8[7];
assign  next_r9[8:1]=r9[7:0],next_r9[0]=r9[8]^r9[7];
assign  next_r10[8:1]=r10[7:0],next_r10[0]=r10[8]^r10[7];
```



```
if(rst)begin
     r1<=8'b1101_0001;
     r2<=8'b1001_1001;
     r3<=8'b0101_1101;
     r4<=8'b0110_0001;
     r5<=8'b1101_1001;
     r6<=8'b1000_1101;
     r7<=8'b1111_0001;
     r8<=8'b1101_0101;
     r9<=8'b1001_0101;
     r10<=8'b1100_1001;
```

It generate pseudourandomness with methods above.

5. collision detector

The module only detects collision between two objects by checking overlapping of x and y coordinates. If both events are true, then collision is expected and sends out a "hit" signal.

6. audio

Uses pmod to output music. The work from lab5 is utilized.
We use Doraemon's theme to make it sounds happier and relaxing.

7. **keyboard_decoders**
   Uses the keyboard decoder from lab6.

8. **seven-segments**
   Uses register[15:0]'s first bit do run the four concurrently.

9. **top module**
   top module deal with

   1)LED controls

   LED flashes when player finally reach score 9999!! There are four display pattern and each with a duration of 16 cycles.

   a) right-shifting
   All LEDs will be set to high initially, and then turned off sequentially from left to right.

   b) left-shifting
   This state is similar to right-shifting, but turning off in the opposite direction.

   c) Crossing
   This state combines two previous pattern, but each bounced back when it reaches middle LEDs.

   d) Twingling
   This final state sets LEDs into pairs such as 1100_1100_0011_0011.
   In each cycle the LEDs invert its state and thus create a glittering effect.

   The LEDs repeat these four states until the game is reset.

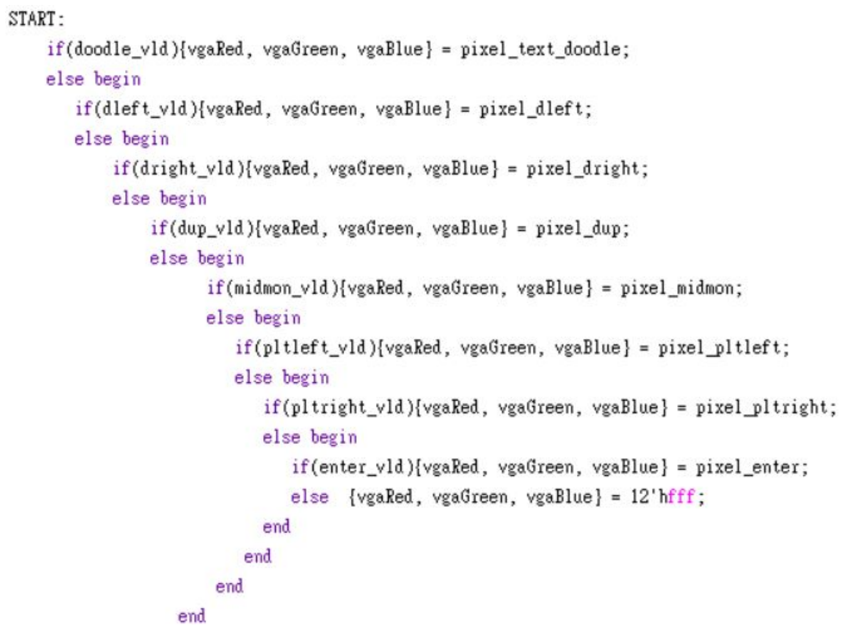
   2)state & score controls ()

   There are 4 states:
   START: menu page
   RUN0: main stage when everything keep still
   RUN1: main stage when game runs
   END: the ending page
   state signal are delivered into many modules, to tell them only to trigger changes in specific states.

   score is added by 3***advance**, which is a special signal saying how much height it jumps, generated by move_controller of player.

   3)VGA ouput layers control (line 251)

The layers are streched by a simple if-else structure.
We setted valid_signal for every object.
the order listed as:
START: actually there’s no regulations here,
since graphs are (almost)stable at menu pagefor
only for uniformity.



```
START:
    if(doodle_vld){vgaRed, vgaGreen, vgaBlue} = pixel_text_doodle;
    else begin
       if(dleft_vld){vgaRed, vgaGreen, vgaBlue} = pixel_dleft;
       else begin
          if(dright_vld){vgaRed, vgaGreen, vgaBlue} = pixel_dright;
          else begin
             if(dup_vld){vgaRed, vgaGreen, vgaBlue} = pixel_dup;
             else begin
                  if(midmon_vld){vgaRed, vgaGreen, vgaBlue} = pixel_midmon;
                  else begin
                     if(pltleft_vld){vgaRed, vgaGreen, vgaBlue} = pixel_pltleft;
                     else begin
                        if(pltright_vld){vgaRed, vgaGreen, vgaBlue} = pixel_pltright;
                        else begin
                           if(enter_vld){vgaRed, vgaGreen, vgaBlue} = pixel_enter;
                           else  {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
                        end
                      end
                   end
                end
```

RUN0,RUN1: blt_vld > plyr_vld >mon_vld >bg_vld

Conclusion:

Hardware environment is not common to be used by game programming **.**In this project, we have implemented a primary game environment that includes collision detectors, layer controls. Some advanced features like gravity simulation and random generators are also involved. We overcome the problems of VGA signals and discontinuity between block memories and controllers. Not only did we apply the knowledge we have learned, new questions encountered in the assignment also lead to new discoveries. The game, with some imperfections need to be improved,comes out as an enjoyable one to play with.

Appendix: Individual reports

李思佑—

Firstly, The debugging cost a lot of time when programming since every time the synthesis take several minutes, even sometimes half an hour. It’s very different from the software projects we’re used to program.

By setting the systhesis -directive* to Runtime-optimize can slightly mitigate the problem.

The second problem I encounter is loading pictures. The pixel on the right is missing sometimes since the h_cnt and v_cnt are not processed properly(sometimes simple delay will cause serious problems).

I make sure the correctness by setting the region in memory address generator in specific pattern.

The third problem I encounter is block memory. Since the limit of Basys3 is only 1800,000 bits. Many delicate pictures cannot be used(or need to be processed very complicatedly).

To solve this, I calculated and resize every graph’s size to make best composition and usage.

The project made me think of a lot of explorational things, which I might have never deep down into with normal exams and assignments. Although it’s far from enough, that’s what learnings should be like.

鄧華予ー

In this assignment, the ability to identify bugs before generating bitstream is much important because it’s such a time consuming process. Plus, as the project grows larger, from time to time I would forget my progress and result in trivial errors such as forget to connect ports in top module. I guess using some notes may help, just like making version control.