ECE 385

Fall 2022
Final Report

Doodle Jump on FPGA

Jiabao Shen (jiabaos3)

Chuanrui Chen (cc86)

TA: Hongshuo Zhang

1. Introduction

We made a simple version of the popular game Doodle Jump on FPGA. Based on the previous labs about the moving ball, we implemented the basic functions of the original game, including jumping, obstacles (monsters, moving stairs), additional tools (springs) and the score record. The goal of the game is to let the character (doodler) jump as high as it can. The doodler will keep jumping until it touches a monster or fail to stand on a stair on the screen. The doodler will jump higher and faster if it jumps on a spring on the stair. The monster can be destroyed if shot by the bullets from the doodler or trampled under the doodler's feet.

When we start the game, there will be a starting page. After pressing the ENTER button, it comes to the playing page. You can press key "A" or "D" to move the doodler to left or right. You can also press key "⇧" to shoot the bullet straight up, "⇦" to shoot diagonally to the upper left, and "⇨" to shoot diagonally to the upper right (one bullet per press). When the doodler jumps out of one side of the screen, it will jump back to the screen from another side. When the doodler jumps to the upper half of the screen, the background (including the stairs) will move down to make sure that the doodler will not jump off the screen. The stairs will appear and move randomly. The spring can help the doodler jump faster and higher. The score increases as the distances increases. When the doodler touched the monster or fail to stand in a stair, it will die and there will be an ending page showing your score. After pressing the ENTER button, it will come back to the start page and wait for another round.

2. Details of the design and List of features

2.1 List of features

Starting page

The starting page is picture of size 300*480 pixels. At the beginning of the game, you can press enter to start.

Figure 1: Starting page

Three Doodler states

The doodler has three states, including left doodler, right doodler and shooting doodler. The state of doodler is decided by if it is going to right or left, or it is shooting bullets. When you press “A” on keyboard, the doodler will move to left. When you press “D” on keyboard, the doodler will move to right. When you press “⇦” or “⇨” or “⇧”, it will shoot bullets.

Figure 2: Three doodler states

Two monster states

In the game we have a monster moving horizontally. It has two states decided by if it turns right or left.

When doodler hits the head of monster, monster will die. Otherwise, doodler will die.

Figure 3: Two monster states

Bullets

Bullets are used to shoot monster. When bullets shoot monster, monster will die. When doodler is shooting, bullets will be shot. There are three directions of bullets decided by which arrow key on keyboard you press. Press key "⇧" to shoot the bullet straight up, "⇦" to shoot diagonally to the upper left, and "⇨" to shoot diagonally to the upper right. Every time you press one key, one bullet will be shot.

Figure 4: Bullet

Stairs

We have fourteen stairs which are initially set and when the doodler moves up, new stairs are set randomly in ten cases. When one stair disappears on the bottom of the screen, the other one is decided to appear or not on the top of the screen. Sometimes there will be moving stairs and it is also decided by cases set previously.

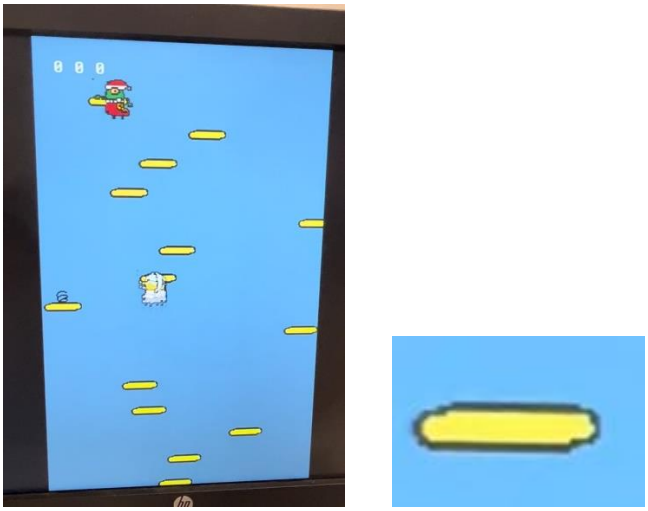

Figure 5: Stairs

Spring

There will be some springs on some stairs. When the doodler touches it , it will speed up.

Figure 6: Spring

Score

When doodler is jumping, the distance of jumping increases so we use score to accumulate it. There is 3 digits score number on the top left of the screen.

Figure 7: Score

Ending page

When you touch a monster or fall down, you will die and the screen will turn to an ending page. You will have your total score on this page. Then when you press enter, the page will turn to starting page and you can press enter again to start.

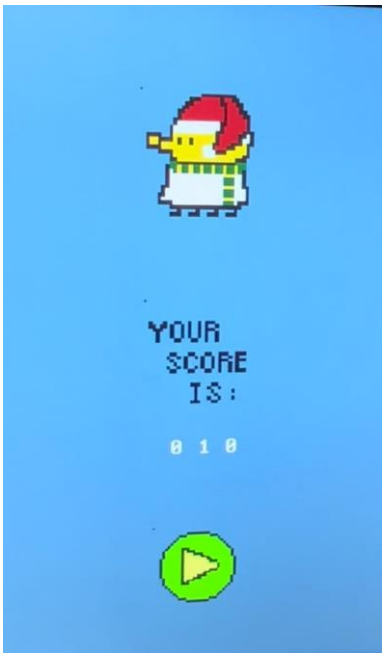

Figure 8: Ending page

2.2 Details

How to make stairs move and generate random stairs?

When doodler collides with stairs and moves up after it approaches to half of screen, the doodler will be still and stairs will move down as the speed of doodler. One stair disappears on the bottom of screen, the other stair will be decided to appear or not on the top of the screen. We have eleven cases for 14 initial stairs to decide it will appear again or not after it disappears. We use the total distance stairs move down to decide which cases it will use. Every time total distance achieves to 1000 will cause a change of game page. The game page is controlled by three 14 bits signals called moving message, active message and tool message, deciding if one stair should move, if one stair should appear and if there will be one spring on this stair separately.

```
if (Ball_Y_Pos <= 10'd239 && Ball_Y_Motion > 10'd30)
   begin
      Ball_Y_Pos <= Ball_Y_Pos ;
      distance <= Ball_Y_Motion;
   end
```

```
for (int i = 0; i < 14; i++)
   begin
      Stair_Y[i] <= Stair_Y[i] - distance;

      if(Stair_Y[i] > 10'd479)
         begin
            if(active_message[i])
               begin
                  //Stair_X[i] <= random_num[9:0];
                  Stair_Y[i] <= 10'd0;
               end
            else
               begin
                  Stair_X[i] <= 10'd600;
                  Stair_Y[i] <= 10'd600;
               end
         end

   end

   for (int k = 0; k < 14; k++)
      begin
         if (move_message[k])
            Stair_X[k] <= stair_x_in[k] + Stair_X[k];
      end
```

```
module random_num (input frame_clk, Reset,
                   input [31:0] distance_sum,
                   output [13:0] ac_mes,
                                 mv_mes,
                                 tl_mes
                   );
 //logic [4:0] ac_addr, mv_addr, tl_addr;
 logic [3:0] changing;

 always_comb
begin
 changing = distance_sum / 10000;

case (changing)
 4'd0 :
begin
 ac_mes =14'b11111111111111;
 mv_mes =14'b00000100000100;
 tl_mes =14'b00000010000000;
 end

 4'd1 :
begin
 ac_mes =14'b01111111111101;
 mv_mes =14'b00100000100000;
 tl_mes =14'b00000100001000;
 end

 4'd2 :
begin
 ac_mes =14'b01111101101101;
 mv_mes =14'b01000100001000;
 tl_mes =14'b00100000000000;
 end
```

Figure 9: Codes about how to move stairs and how to generate new stairs.

How to calculate scores?

We write a module to calculate the total distance stairs moving and use it to calculate the score number, then we match three digits to font rom to decide which number should be printed out to the screen.

```
score3 = score_num/100;
score1 = (score_num - 100*(score_num/100))/10; //two digits num
score2 = score_num%10;

nd
```

```
//------------------------------------------------------------
module total_distance ( input Reset, frame_clk,
                        input drop,
                        input [9:0] distance,
                        output logic [31:0] distance_sum
                       );
    logic [31:0] sum;
    assign distance_sum = sum;

    always_ff @ (posedge Reset or posedge frame_clk)
       begin
          if (Reset)
             sum <= 0;
          else if (drop)
             sum <= sum;
          else
             sum <= sum + distance % 1000;
       end
endmodule
```

Figure 10: Codes about how to calculate total distance and three digits of score.

3. Design procedure

We almost follow the timeline in proposal to finish our report. The following is the general sequence and process of our completion of this project.

-Draw our pictures and use 385 tools on website to convert our picture from png to txt using palette method.

-Add 14 initial stairs on the screen and change ball in lab62 to doodler with left and right states. Now doodler can only move horizontally.

-Add moving monster on the screen.

-Write collision between stairs and doodler.

-Add vertical motion of doodler and let it can collide with stairs. Now the doodler will jump with gravity on stairs and can move up and turn left or right using A or D.

-Add some springs on stairs. Make doodler gain speed if it touches the spring.

-Extend keyboard so that there can be two keys input at the same time.

-Add shooting doodler state.

-Write bullet module. Now use three arrow keys the bullet will be shot successfully.

-Add collision between bullet and monster. If bullet touch monster, monster will disappear immediately.

- Add distance parameter in project and let stairs can move down.

- Add fixed active message, tool message and moving message to decide how to generate a new stair on top of the screen if one stair disappears on the bottom of the screen.

-Add drop signal and dead signal. Doodler is dropping when it touches almost bottom of screen and it is dead when it touches monster.

-Add start page and state machine, containing start, play, dropping, death, halt states. If you press enter you will start the game. When doodler drops for a period or touches a monster, you will achieve to death state and a blank screen will appear. You can press enter to come back to starting page again.

- Modify ending page.

- Add score calculation systems and show the final score on ending page.

-Modify active message, tool message and moving message. Set 11 cases for game page having different stairs, springs and moving stairs. Let them be randomly decided by how many 1000 the total distance of stairs moving has.

-Adjust active message, tool message and moving message to let the game have a reasonable difficulty.

4. Block Diagram

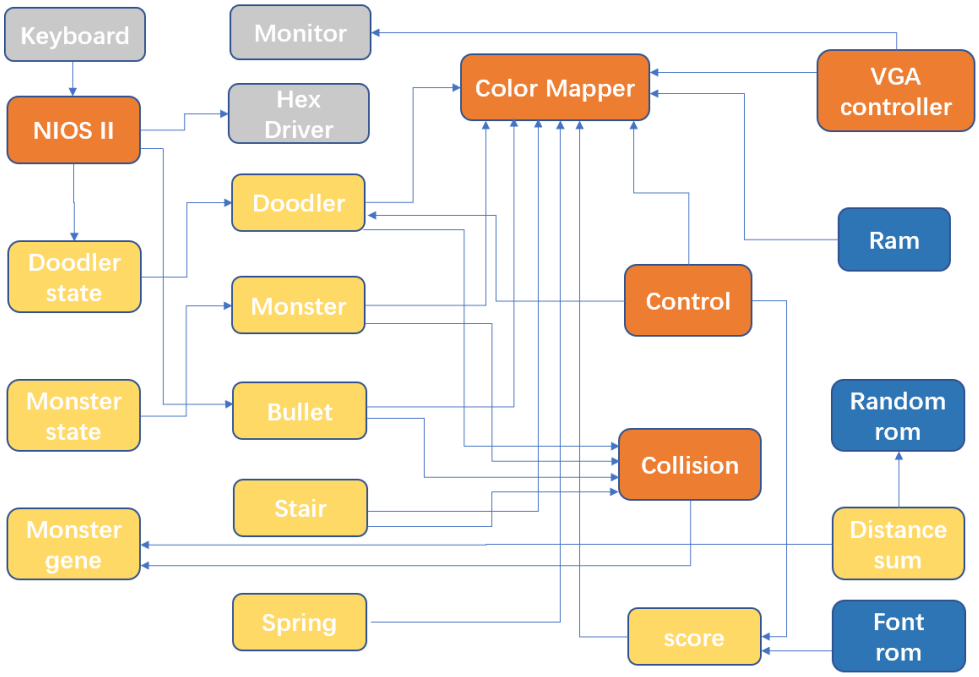

Figure 11: Block Diagram

5. Module Description

a. All sv. Modules

Module: lab62

Inputs & Outputs:

```
////////// Clocks //////////
input     MAX10_CLK1_50,
////////// KEY //////////
input    [ 1: 0]   KEY,
////////// SW //////////
input    [ 9: 0]   SW,
////////// LEDR //////////
output   [ 9: 0]   LEDR,
////////// HEX //////////
output   [ 7: 0]   HEX0,
output   [ 7: 0]   HEX1,
output   [ 7: 0]   HEX2,
output   [ 7: 0]   HEX3,
output   [ 7: 0]   HEX4,
output   [ 7: 0]   HEX5,
////////// SDRAM //////////
output             DRAM_CLK,
output             DRAM_CKE,
output   [12: 0]   DRAM_ADDR,
output   [ 1: 0]   DRAM_BA,
inout    [15: 0]   DRAM_DQ,
output             DRAM_LDQM,
output             DRAM_UDQM,
output             DRAM_CS_N,
output             DRAM_WE_N,
output             DRAM_CAS_N,
output             DRAM_RAS_N,
////////// VGA //////////
output             VGA_HS,
output             VGA_VS,
output   [ 3: 0]   VGA_R,
output   [ 3: 0]   VGA_G,
output   [ 3: 0]   VGA_B,
////////// ARDUINO //////////
inout    [15: 0]   ARDUINO_IO,
inout              ARDUINO_RESET_N
```

Description: This is the top level for the program. It combines all the submodules together.

Purpose: It is the top-level module.

Module: Color Mapper

Inputs & Outputs:

```
input Clk, //frame_clk, Reset is added 11/29
input [9:0] BallX, BallY, DrawX, DrawY, Ball_size,
input [13:0][9:0] Stair_X, Stair_Y, // pos of stair
input [9:0] MonsterX, MonsterY, // added for monster position
input [13:0][9:0] toolX, toolY, toolS,//added 11/29
input [9:0] BulletX, BulletY, BulletS, //added 12/1
input [9:0] SBX, SBY, STX, STY, SNX, SNY, //added 11/29 start page
input [9:0] EDX, EDY, RBX, RBY, YSX, YSY, //added 12/5 end page
input [13:0] find,  // check if to draw stair
input [2:0] is_doodler,
input logic is_monster,
input [13:0] is_tool,
input logic is_bullet, //added 12/1
input logic is_sb, is_st, is_sn, //added 11/29 for start page
input logic is_ed, is_rb, is_ys,//added 12/5 for end page
input logic is_score, //added 12/5
input [1:0] doodler_state,
input logic monster_state, //added for left and right monster
input appear, //dead, //drop,//added 11/27
input [2:0] show, // 0:play, 1:start, 2:game over, added 11/27
input [7:0] keycode, //addeed 11/27
input [7:0] keycode_ext,
output logic [7:0]  Red, Green, Blue );
```

Description:

Purpose: Color mapper needs to have as inputs the horizontal and vertical position counters and maps out color either to foreground color or background color.

Purpose: It is used to combine it with VGA controller to draw the shapes of the components for the game.

Module: control

Inputs & Outputs:

```
( input logic Clk, Reset, frame_clk,
  input logic [7:0] keycode,
  input logic [7:0] keycode_ext,
  input dead, drop,
  output logic [2:0] show,
  output logic restart);
```

Description: This is the state machine of the game. It contains mainly five states: start, play, drop, dead, halt. The details are on the “state machine” part.

Purpose: the state machine for the game.

Module: doodler

Inputs & Outputs:

```
( input Clk, Reset, frame_clk,
  input [9:0] DrawX, DrawY,
  input [7:0] keycode,
  input [7:0] keycode_ext,
  input [1:0] doodler_state,
  input logic collision, //add 22/11/27
  input logic gain,
  input logic dead, //add 22/12/7
  output logic drop, //added 12/4
  input [31:0] distance_sum, //added 12/4
  output [9:0] distance, //add 12/3
  output [2:0] is_doodler,
  output [9:0]  BallX, BallY, BallS, Ball_Y_Step_out);
```

Description: This is the module for most operations of the doodler. The doodler has three states, which are right, left and shooting. This module is connected to the doodler_state module. The input signals collision, gain, dead and drop tells which state of doodler we use.

Purpose: Used for the display and control of the doodler.

Module: monster

Inputs & Outputs:

```
input Clk, Reset, frame_clk,
input gene, hit, beat_monster,//added to check
input [9:0] DrawX, DrawY,
input logic monster_state, // added to check th
input [9:0] distance, //added 12/3
output logic is_monster,
output [1:0] turn, //added to change the state
output [9:0]  Monster_X, Monster_Y, Monster_S,
output logic  appear  // if monster should appe
```

Description: This is the module for most operations of the monster. The monster has two states, which are right, left. It moves horizontally. When it touches one side of the screen, it will turn and move in the opposite direction. This module is connected to the monster_state module. The input signals hit, gene and beat_monster tell when the monster should appear or disappear.

Purpose: Used for the display and control of the monster.

Module: stair

Inputs & Outputs:

```
(
 input Reset, Clk, frame_clk,
 input [9:0] DrawX, DrawY,
 output [13:0][9:0] Stair_X, Stair_Y,
 output [13:0] find,
 output logic [9:0] stair_size,
 ///// 22/11/29
 input [13:0] tool_signal, // add for making su
 output logic [13:0][9:0] toolX, toolY, toolS,
 input [13:0] move_message, active_message,
 input [9:0] distance
 );
```

Description: This is the module for most operations of the stair. The original number of stairs per screen is 14. We use the signal active_message to determine which stairs to appear. The signal move_message is used to choose which stairs to move horizontally. The signal tool_signal is used to choose which stairs have springs. The signal When the moving stair touches one side of the screen, it will turn and move in the opposite direction.

Purpose: Used for the display and control of the stairs.

Module: spring

Inputs & Outputs:

```
( input Clk, Reset, frame_clk,
  input [9:0] DrawX, DrawY,
  input [9:0] BallX, BallY, BallS, Ball_y_step,
  input [13:0][9:0] toolX, toolY, toolS,
  output logic gain,
  output [13:0] is_tool
```

Description: This is the module for the display of the spring. It will output a signal gain to let the doodler jump higher and faster when it jumps onto a spring.

Purpose: Used for the display and control of the springs.

Module: bullet

Inputs & Outputs:

```
( input Reset, frame_clk, Clk,
  //input hit, //
  //input [1:0] direction,
  input [9:0] DrawX, DrawY,
  input [7:0] keycode, keycode_ext,
  input [9:0] bullet_x, bullet_y, bullet_s,
  output logic fly,
  output logic is_bullet,
  output [9:0]  BulletX, BulletY, BulletS );
```

Description: This is the module for most operations of the bullet. The keycode will

control the direction of the bullet, including straight up, diagonally left up and right up. The output signal fly is for the moving process of the bullet.

Purpose: Used for the display and control of the bullets.

Module: doodler_state

Inputs & Outputs:

```
( input Reset, frame_clk,
  input [7:0] keycode,
  input [7:0] keycode_ext,
  output [1:0] state
 );
```

Description: This is the module for the state of the doodler. The keycode will control the state of the doodler, including left, right and shooting. The output state will be given to the module doodler.

Purpose: Used for the state of the doodler.

Module: monster_state

Inputs & Outputs:

```
( input Reset, frame_clk,
  input [1:0] turn,
  output logic state
 );
```

Description: This is the module for the state of the doodler. When the monster touches one side of the screen, it will turn to another direction by the input signal turn. The output state will be given to the module doodler.

Purpose: Used for the state of the monster.

Module: Collision

Inputs and Outputs:

```
module doodler_stair ( input Clk, Reset, frame_clk,
                       input [9:0] BallX, BallY, BallS, Ball_y_step,
                       input [13:0][9:0] StairX, StairY,
                       input [9:0] StairS,
                       output logic collision
                     );
```

```
module bullet_monster ( input Clk, Reset, frame_clk,
                        input [9:0] BulletX, BulletY, BulletS,
                        input [9:0] MonsterX, MonsterY, MonsterS,
                        input logic appear, fly, //what is fly ???
                        output logic hit
                      );

module doodler_monster ( input Clk, Reset, frame_clk,
                         input [9:0] BallX, BallY, BallS, Ball_y_step,
                         input [9:0] MonsterX, MonsterY, MonsterS,
                         input logic appear,
                         output logic dead,
                         output logic beat_monster
                       );
```

Description: This includes three kinds of collision: doodler jumping onto a stair, bullet hits the monster, and the doodler touches or beats the monster.

Purpose: This is the module for the collision in the game.

Module: vga_controller

Input: Clk, Reset

Output: hs, vs, pixel_clk, blank, sync,

[9:0] DrawX, DrawY

Description: It is the controller for the VGA to determine how the character will be drawn including its location and color. according to the vertical synchronous pulse and the horizontal synchronous pulse on the screen and the RGB.

Purpose: It is used to control the display of the character.

Module: rams

Input: [10:0] addr,

Output [7:0] data

Description: The is the ram for all the components that are to be displayed on the screen. The txt. File provides the RGB information for different location of the pixals.

Purpose: ram for the components to be displayed.

Module: font_rom

Input: [10:0] addr,

Output [7:0] data

Description: It is the on-chip memory VRAM. The address changes from 10 to 12 bits for the week2 design to accommodate the additional VRAM and palette.

Purpose: The on-chip memory for the Week2's design.

Module: HexDriver

Input: logic [3:0] In0

Output: logic [6:0] Out0

Description: It is used to show the signals in HEX on the FPGA.

Purpose: Show signals in the HEX on FPGA.

b. platform design

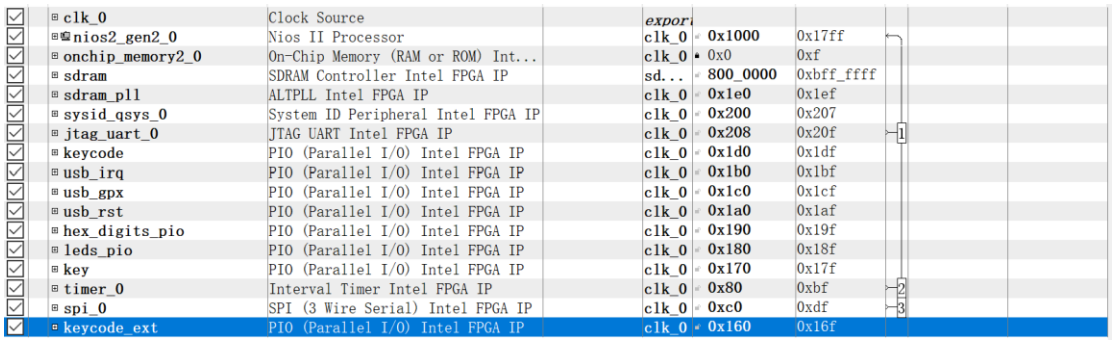

Figure 12: Platform design

- clk_0: This is the clock module. The clock is given to all other modules except sdram. The actual clock frequency is 50MHz, and the clock phase shift is 0ns.
- nios2_gen2_0: This is the Nios II Processor. We use economy version of the processor, which is the resource-optimized 32-bit RISC. It is used to compile C code.
- onchip_memory2_0: This is the on-chip memory. The memory type is RAM and the total memory size is 16 bytes.
- sdram: This is the off-chip SDRAM. We use it to store the software program. We choose 32M*16 chips, and the total amount of memory is 512MBits.

  sdram_pll: This is the PLL component of SDRAM. It provides the clock signal

for the SDRAM chip. The phase shift is -1ns.

- sysid_qsys_0: This is the System ID Peripheral Intel FPGA. It is used to ensure the compatibility of the hardware and the software.
- jtag_uart_0: It is the JTAG UART peripheral.
- keycode: This is the PIO for keycode. It is connected to the Avalon data bus.
- keycode_ext: This is the extended PIO for keycode. It is connected to the Avalon data bus.
- usb_irq: This is the PIO for USB_IRQ. It is connected to the Avalon data bus.
- usb_rst: This is the PIO for USB_RST. It is connected to the Avalon data bus.
- usb_gpx: This is the PIO for USB_GPX. It is connected to the Avalon data bus.
- hex_digits_pio: This is the PIO for HEX digits. It is connected to the Avalon data bus.
- key: This is the PIO for key. It is connected to the Avalon data bus.
- timer_0: It is the internal timer.
- spi_0: This is the Seral Peripheral Interface used for the FPGA to communicate with the signals in peripherals.

6. State Machine

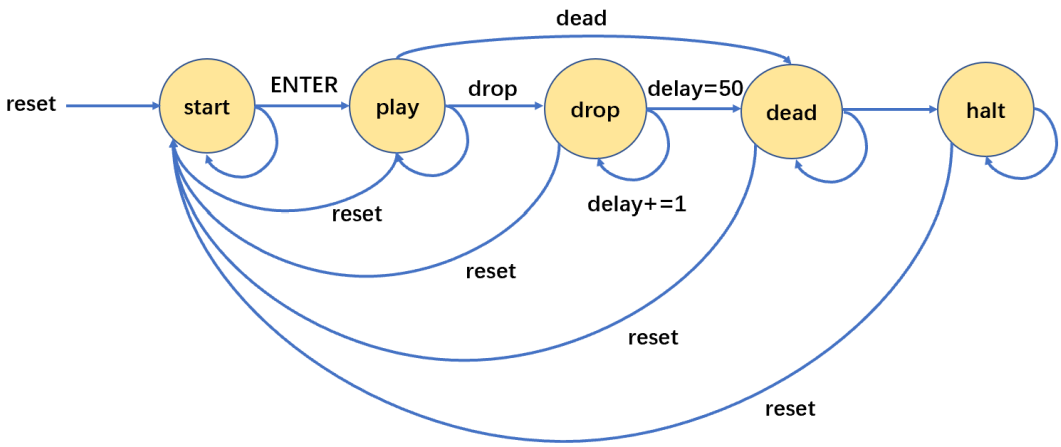

Figure 13: State Machine

Generally, we have five states for the whole game. From the restart, we get to the start state, which shows a starting page. When you press the ENTER button, it comes

to the play page and the doodler starts jumping. When you fail to let the doodler jump on one stair, it starts to drop and comes to the drop state. The drop will have 50 delays until it comes to the dead state, which there will be an ending page. When your doodle touched the monster on the play state, it will come to the dead state. If you don't press, it will come to the halt state. When you press ENTER, it will comes back to the reset and the start page and there will be another round.

7. Design Statics

LUT	11979
DSP	0
Memory(BRAM)	1,677,312 bits
Flip-Flop	3401
Frequency	71.08MHz
Static Power	97.06mW
Dynamic Power	173.03mW
Total Power	292.12mW

Table 1: Design Statics

8. Conclusion and known bugs

Our final project achieves all baseline features provided in proposal and the whole game works well. We draw all pictures by hand and all pictures are clear. We use three digits score number to calculate the score. Our doodler and monster work well. Bullets which have three directions can also shoot monster successfully. There are springs on stairs and some stairs will move. There are starting page and ending page controlled by a state machine to restart the game.

Besides, there are also some places to improve. Firstly, sometimes the score will still increase when doodler touches a monster. That is because though the doodler is dead and the final score is shown on the screen, the doodler can still jump up invisibly on the stair until it can not jump up or it falls. That is a place of score accumulation

system to improve. Besides, our random stairs, springs and moving stairs are set by cases previously. Which case of stairs will be used is decided by the distance of doodler moving up (or stairs moving down, they are the same). We use hardware instead of software. It will be more convenient and trustful to use software. This is another place we can improve.

The process of final project is extremely hard, and it is unbelievable that we spend almost half time of our fall break to debug for one place. Every time we add new elements to final project, there will be a bug waiting for us. But the result is really good, and we are amazed that we really finish the game by ourselves and every baseline feature we provide is achieved. The whole process is challenging but we learn a lot and gradually we find we really enjoy the process of improving our code and game. Thanks for ECE385, which gives us a precious experience of coding.