ECE 385

Fall 2017

Final Project

Mingxi Zou & Lilan Yang

ABD / Tuesday 8am
Xinying Zhao

INTRODUCTION

For the final project, we construct a dress-up game based on the construction of Lab 8, adding multiple game states to switch between interfaces that serves the function of selecting gender, skin, hair and clothes by pressing the numbers on the keyboard.

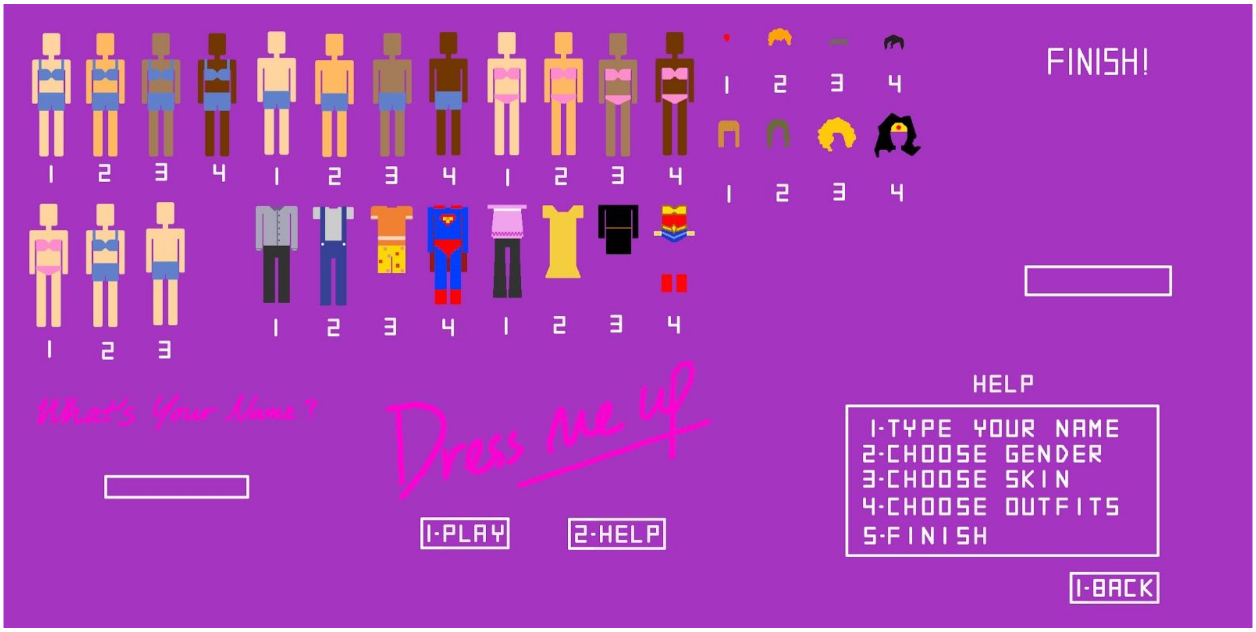
PREPARATIONS

1. Coming Up with the Idea

To think outside of the box, we want to build a game that stands out graphically with beautiful choices of colors and stylish designs. Also to satisfy the childhood nostalgia and to properly use our knowledge on basic systems working, we get to combine the two and try to make something awesome.

2. Drawing the Figures

Pixel by pixel, with the usage of *Pixel Art Studio*, we draw all the mini figures as below:



This image is the combination of our pre-drawn 12 interfaces. The image with title “Dress me up” is the first interface of our game and after pushing button 1 the interface will change to choosing gender. Those images with little bodies are interfaces for choosing gender, male skin, bi skin. And there are also images for choosing male clothes/hair and female clothes/hair. The right upper one with “FINISH!” is the one we use to show the final figure with the choices from previous state. The with help instruction is the interface player could go from the first interface by pushing button 2.We used want to use the image with “What’s your name?” to let players enter their name but we give up on it.

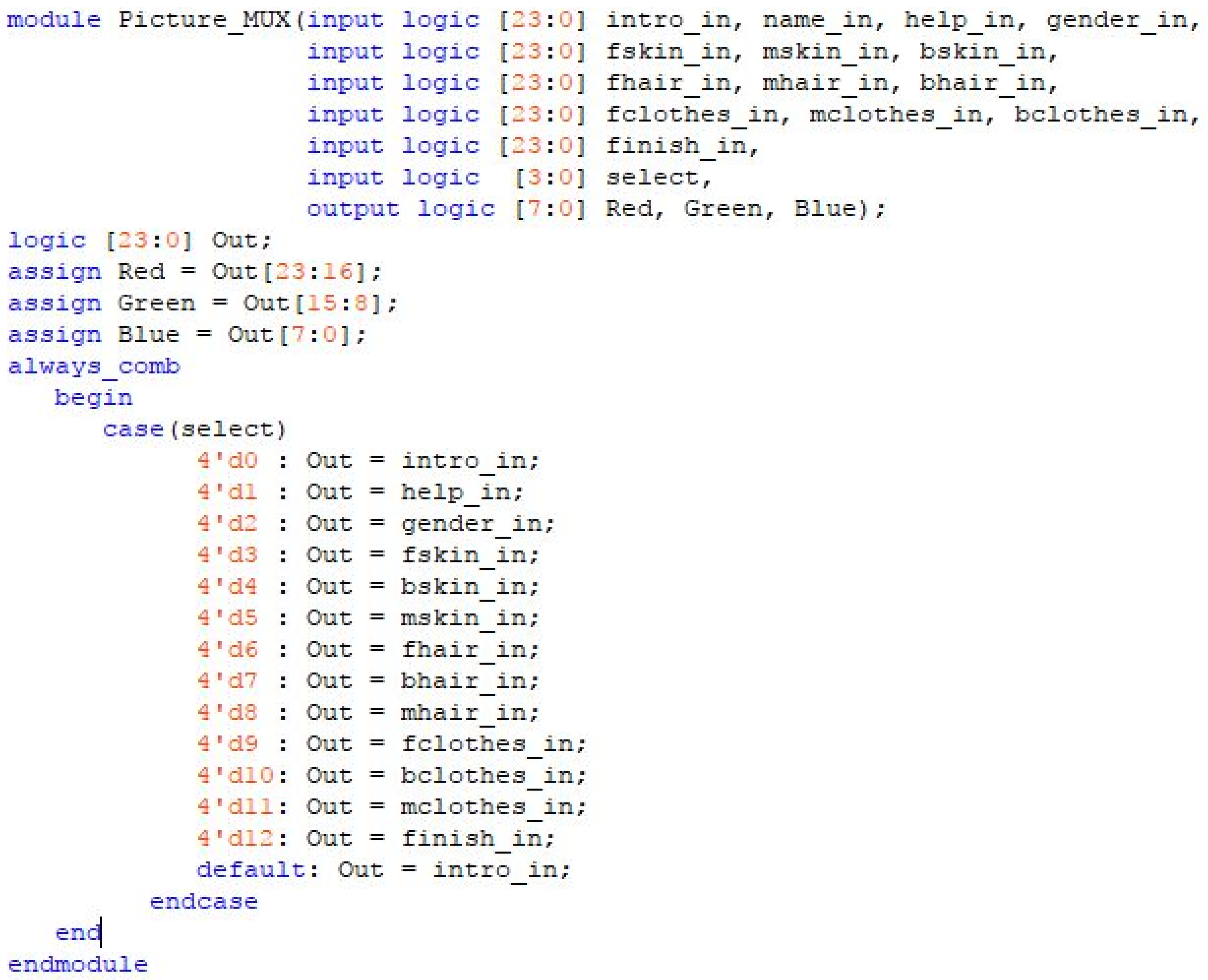
3. Converting PNG Image into TXT File

Using Rishi's ECE 385 Helper Tools and Anaconda, by getting all the colors in the right order for the color palette, PNG images has been successfully converted into TXT files such that the first color in the color palette would be presented by 0, second by 1, etc., then with the correctly generated files. We construct the ram.sv as discussed below in details.

WRITTEN DESCRIPTION

The whole project could be divided into three parts - mux, ram, and game_state.

- **Mux**

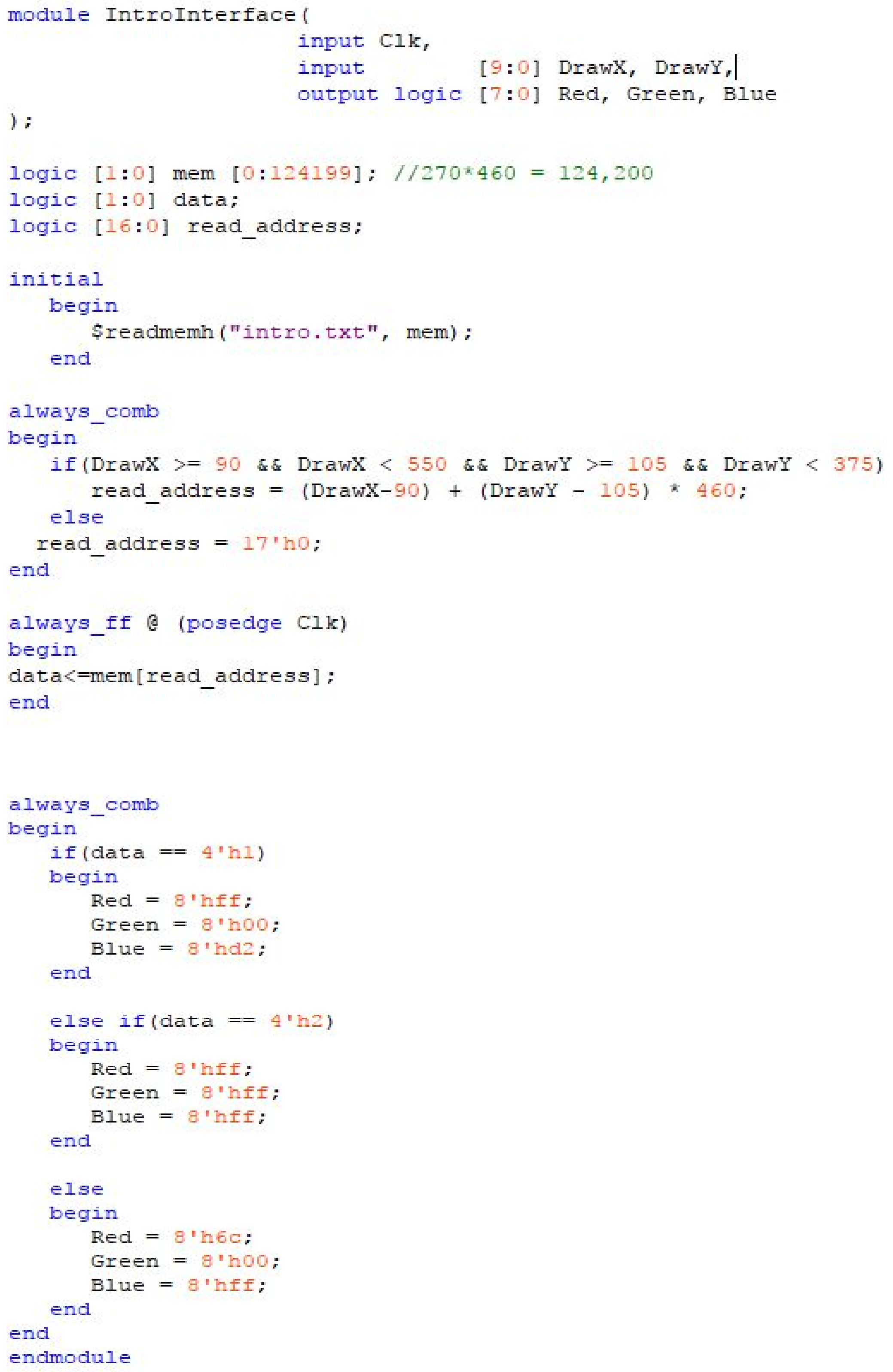


```
module Picture_MUX(input logic [23:0] intro_in, name_in, help_in, gender_in,
                   input logic [23:0] fskin_in, mskin_in, bskin_in,
                   input logic [23:0] fhair_in, mhair_in, bhair_in,
                   input logic [23:0] fclothes_in, mclothes_in, bclothes_in,
                   input logic [23:0] finish_in,
                   input logic  [3:0] select,
                   output logic [7:0] Red, Green, Blue);
logic [23:0] Out;
assign Red = Out[23:16];
assign Green = Out[15:8];
assign Blue = Out[7:0];
always_comb
   begin
      case(select)
            4'd0 : Out = intro_in;
            4'd1 : Out = help_in;
            4'd2 : Out = gender_in;
            4'd3 : Out = fskin_in;
            4'd4 : Out = bskin_in;
            4'd5 : Out = mskin_in;
            4'd6 : Out = fhair_in;
            4'd7 : Out = bhair_in;
            4'd8 : Out = mhair_in;
            4'd9 : Out = fclothes_in;
            4'd10: Out = bclothes_in;
            4'd11: Out = mclothes_in;
            4'd12: Out = finish_in;
            default: Out = intro_in;
         endcase
   end
endmodule
```

In this part we built up a big mux with 13 different output images. Both select bit and the RGB values of each images are the inputs of the mux and the outputs are the RGB values.

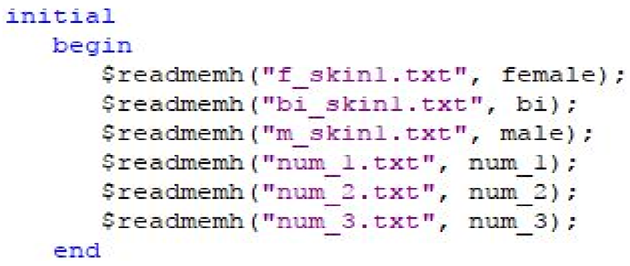
- **Ram**

We build up modules for each interfaces in the ram.sv. The simplest one is the IntroInterface module.



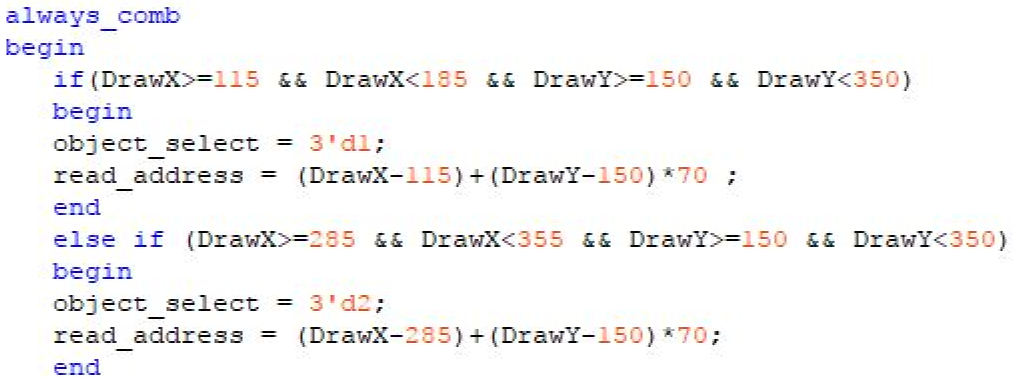
```
module IntroInterface(
                     input Clk,
                     input        [9:0] DrawX, DrawY,
                     output logic [7:0] Red, Green, Blue
);

logic [1:0] mem [0:124199]; //270*460 = 124,200
logic [1:0] data;
logic [16:0] read_address;

initial
   begin
      $readmemh("intro.txt", mem);
   end

always_comb
begin
   if(DrawX >= 90 && DrawX < 550 && DrawY >= 105 && DrawY < 375)
      read_address = (DrawX-90) + (DrawY - 105) * 460;
   else
  read_address = 17'h0;
end

always_ff @ (posedge Clk)
begin
data<=mem[read_address];
end



always_comb
begin
   if(data == 4'h1)
   begin
      Red = 8'hff;
      Green = 8'h00;
      Blue = 8'hd2;
   end

   else if(data == 4'h2)
   begin
      Red = 8'hff;
      Green = 8'hff;
      Blue = 8'hff;
   end

   else
   begin
      Red = 8'h6c;
      Green = 8'h00;
      Blue = 8'hff;
   end
end
endmodule
```

In this module, the first parts is to create a mem and use $readmemh to store the converted text to the mem. Next we use an always_comb to set the X,Y axis of the whole image on the screen. Then we use always_ff to read each data stored in mem out we stored the color palette in a always_comb to specify the color value of each value read from mem. This is basically how ram works although the following modules is more complicated.

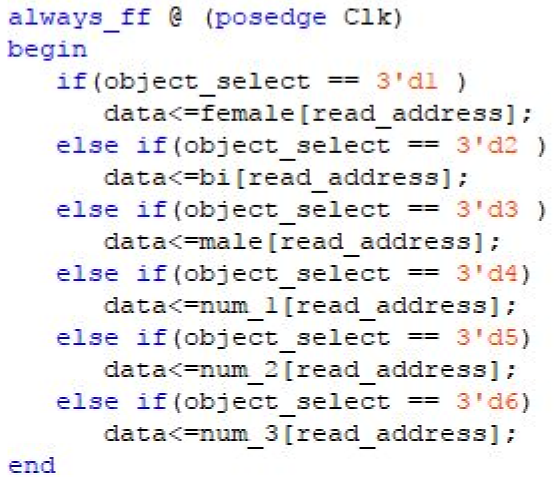


```
initial
   begin
      $readmemh("f_skin1.txt", female);
      $readmemh("bi_skin1.txt", bi);
      $readmemh("m_skin1.txt", male);
      $readmemh("num_1.txt", num_1);
      $readmemh("num_2.txt", num_2);
      $readmemh("num_3.txt", num_3);
   end
```

As for Gender, FemaleSkin, MaleSkin, BiSkin, FemaleHair, MaleHair, BiHair, FemaleClothes, MaleClothes, BiClothes modules. We stored the txt files of each figure elements into different mem. Below is part of the always_comb to set the x,y position of each little figures in the whole interface.



```
always_comb
begin
   if(DrawX>=115 && DrawX<185 && DrawY>=150 && DrawY<350)
   begin
   object_select = 3'd1;
   read_address = (DrawX-115)+(DrawY-150)*70 ;
   end
   else if (DrawX>=285 && DrawX<355 && DrawY>=150 && DrawY<350)
   begin
   object_select = 3'd2;
   read_address = (DrawX-285)+(DrawY-150)*70;
   end
```

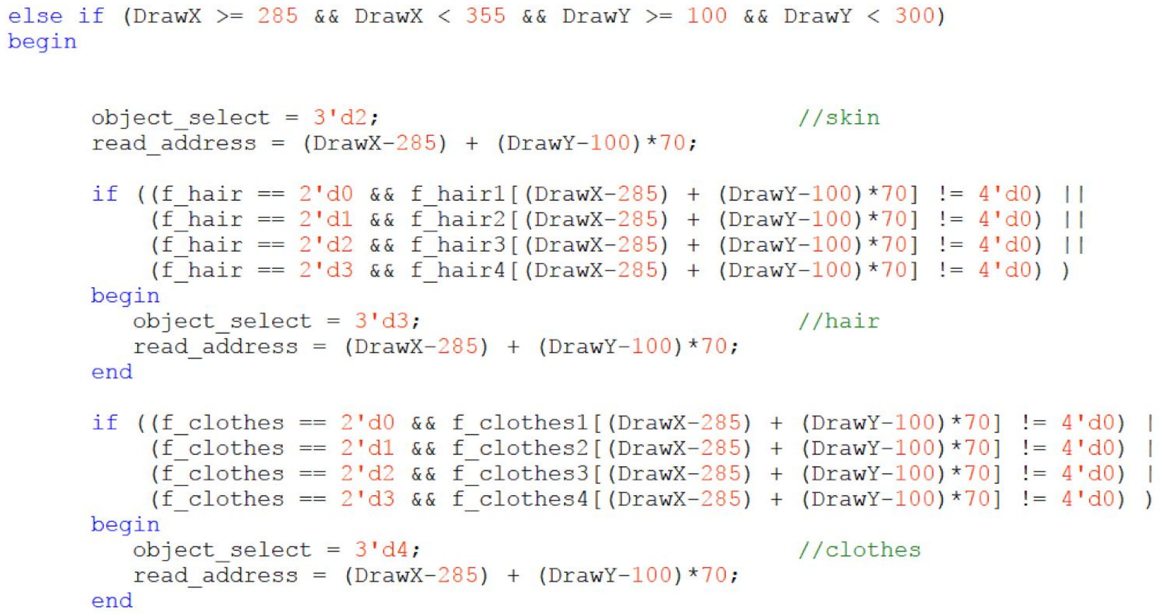


```
always_ff @ (posedge Clk)
begin
   if(object_select == 3'd1 )
      data<=female[read_address];
   else if(object_select == 3'd2 )
      data<=bi[read_address];
   else if(object_select == 3'd3 )
      data<=male[read_address];
   else if(object_select == 3'd4)
      data<=num_1[read_address];
   else if(object_select == 3'd5)
      data<=num_2[read_address];
   else if(object_select == 3'd6)
      data<=num_3[read_address];
end
```

We set the object_select in order to read different figures we want into data.

The last always_comb is the also the color palette of this interface.

The only interface that is kind of different from other is the finish interface.

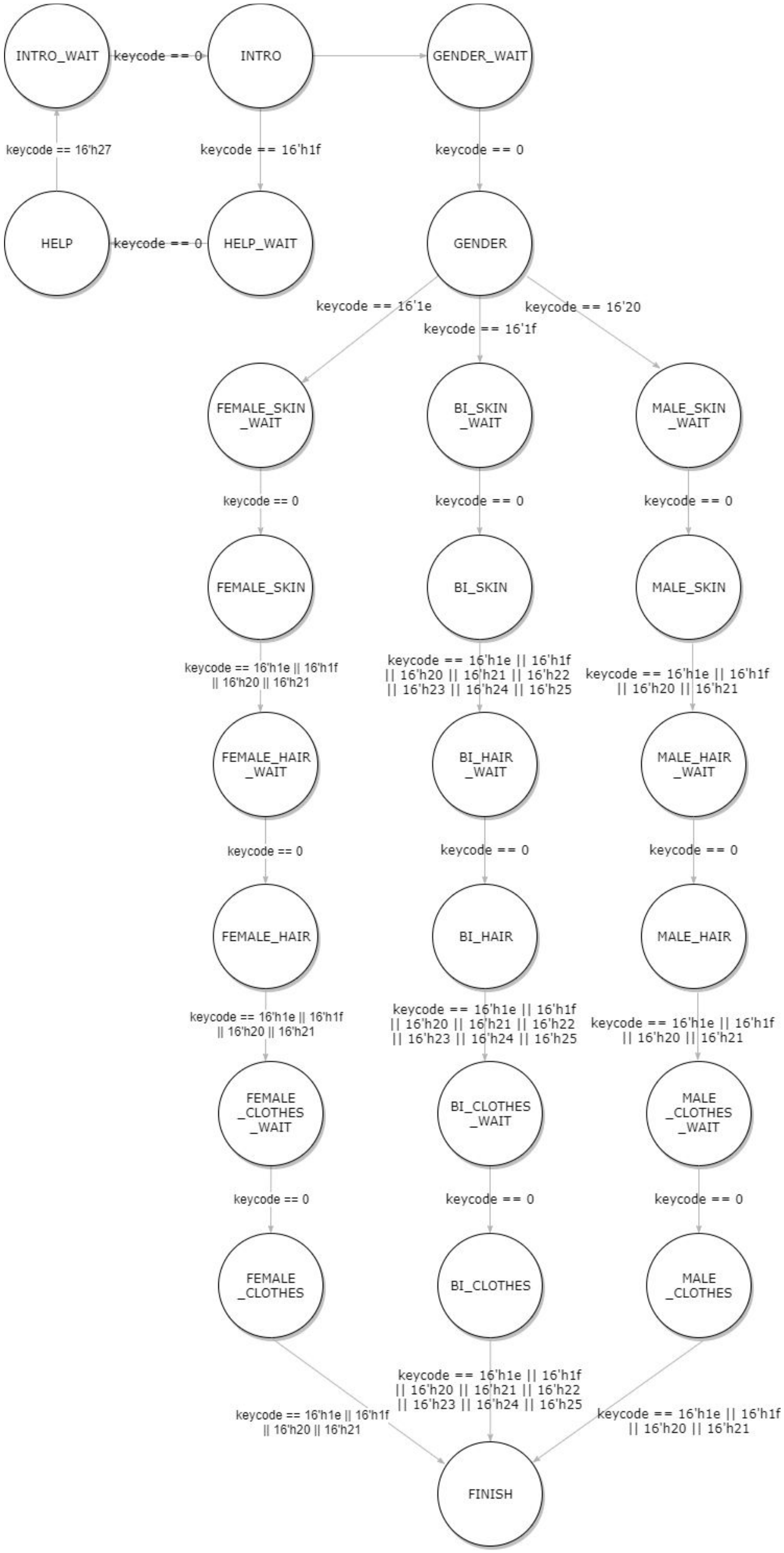


```
else if (DrawX >= 285 && DrawX < 355 && DrawY >= 100 && DrawY < 300)
begin


      object_select = 3'd2;                              //skin
      read_address = (DrawX-285) + (DrawY-100)*70;

      if ((f_hair == 2'd0 && f_hair1[(DrawX-285) + (DrawY-100)*70] != 4'd0) ||
          (f_hair == 2'd1 && f_hair2[(DrawX-285) + (DrawY-100)*70] != 4'd0) ||
          (f_hair == 2'd2 && f_hair3[(DrawX-285) + (DrawY-100)*70] != 4'd0) ||
          (f_hair == 2'd3 && f_hair4[(DrawX-285) + (DrawY-100)*70] != 4'd0) )
      begin
         object_select = 3'd3;                           //hair
         read_address = (DrawX-285) + (DrawY-100)*70;
      end

      if ((f_clothes == 2'd0 && f_clothes1[(DrawX-285) + (DrawY-100)*70] != 4'd0) ||
          (f_clothes == 2'd1 && f_clothes2[(DrawX-285) + (DrawY-100)*70] != 4'd0) ||
          (f_clothes == 2'd2 && f_clothes3[(DrawX-285) + (DrawY-100)*70] != 4'd0) ||
          (f_clothes == 2'd3 && f_clothes4[(DrawX-285) + (DrawY-100)*70] != 4'd0) )
      begin
         object_select = 3'd4;                           //clothes
         read_address = (DrawX-285) + (DrawY-100)*70;
      end
```

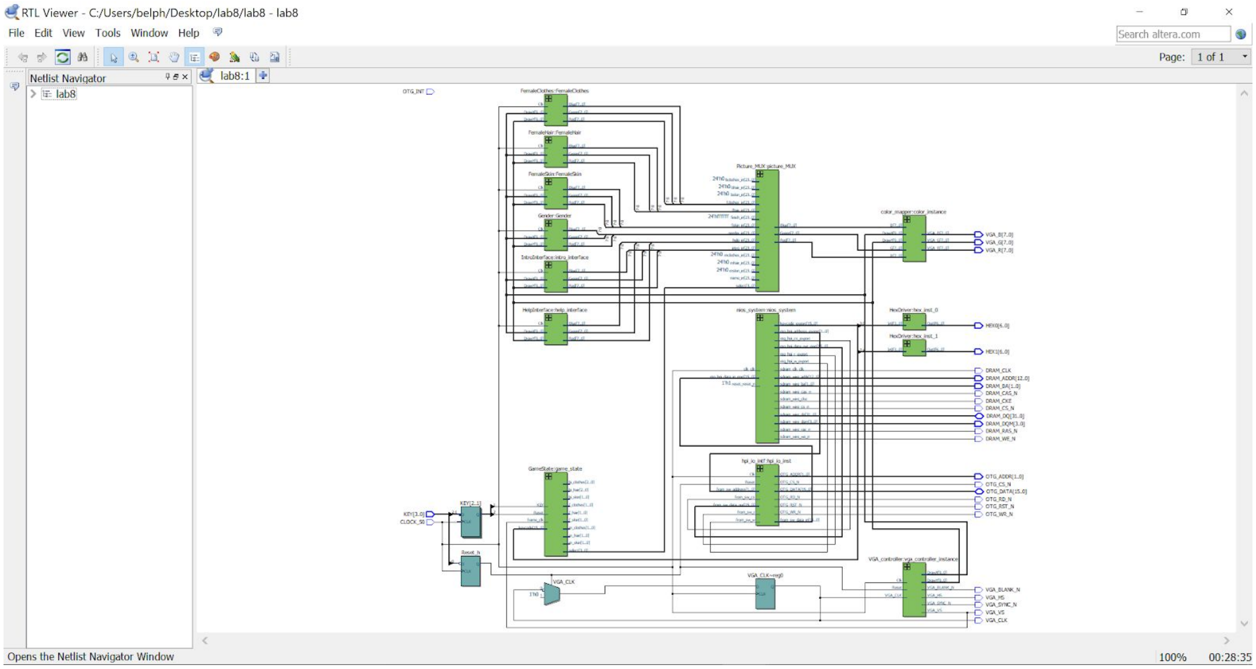
The screenshot code above enables multiple layers overlapping on each other. Taking the final build up for female's choices, we have the skin as the base when object_select has been given 3'd2. Then considering the if conditions for hair and clothes. Previously on the color palette the background color has been given data value of 0, so whenever in the range of the front layer, data that is non-zero would be overwritten by a new value as the the new color of pixel on the top. Then the final figure can be perfectly drawn.

- **Game_state**

We have totally 25 states in the whole project. Details is shown in the graph below:


INTRO_WAIT
keycode == 0
INTRO
GENDER_WAIT
keycode == 16'h27
keycode == 16'h1f
keycode == 0
HELP
keycode == 0
HELP_WAIT
GENDER
keycode == 16'1e
keycode == 16'1f
keycode == 16'20
FEMALE_SKIN _WAIT
BI_SKIN _WAIT
MALE_SKIN _WAIT
keycode == 0
keycode == 0
keycode == 0
FEMALE_SKIN
BI_SKIN
MALE_SKIN
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21 || 16'h22 || 16'h23 || 16'h24 || 16'h25
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21
FEMALE_HAIR _WAIT
BI_HAIR _WAIT
MALE_HAIR _WAIT
keycode == 0
keycode == 0
keycode == 0
FEMALE_HAIR
BI_HAIR
MALE_HAIR
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21 || 16'h22 || 16'h23 || 16'h24 || 16'h25
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21
FEMALE _CLOTHES _WAIT
BI_CLOTHES _WAIT
MALE _CLOTHES _WAIT
keycode == 0
keycode == 0
keycode == 0
FEMALE _CLOTHES
BI_CLOTHES
MALE _CLOTHES
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21 || 16'h22 || 16'h23 || 16'h24 || 16'h25
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21
keycode == 16'h1e || 16'h1f || 16'h20 || 16'h21
FINISH

BLOCK DIAGRAMS



Module Descriptions

Shown in **Appendix A*.

POST-LAB

1.

LUT	10,079
DSP	(specified in later tables)
Memory (BRAM)	534,992
Flip-Flop	2249

Frequency	166.89 MHz
Static Power	102.48 mW
Dynamic Power	0.76 mW
Total Power	170.38 mW

2.

Fitter DSP Block Usage Summary

Statistic	Number Used	Available per Block	Maximum Available
Simple Multipliers (9-bit)	0	2	532
Simple Multipliers (18-bit)	2	1	266
Embedded Multiplier Blocks	2	-	266
Embedded Multiplier 9-bit Elements	4	2	532

CONCLUSIONS

After approximately 50 hours or more time working on the final project from scratch, we work through from the very beginning from drawing every elements in the pixels to uploading on the FPGA board and using VGA display and keyboard to implement the game. Separately for faster compiling and debugging, we tested out the states to be working perfectly, displaying with the right colors but a few pixels going out of places, and presenting the overlaying feature in the final interfaces. Yet it is unfortunate that when trying to combine all the working parts, it fails to compile altogether due to RAM megafunction converts. Other than that, every separated parts work fine.

In the middle of the process, we have encountered lots of problems such as having to re-draw the figures again and again in order to fit the requirement of our finish state. Also Eclipse has always been painful in the testing phase, facing problems of “downloading ELF errors” or “routine mailbox data” being assigned either to some specific value or zero.

It has really been a journey of exploring, experimenting, and making theoretical ideas into building actual games.

**APPENDIX A* - MODULES DESCRIPTIONS

1. **Module**: lab8.sv

    Inputs: CLOCK_50, [3:0] KEY, [6:0] HEX0, [6:0] HEX1, OTG_INT

    Outputs: [7:0] VGA_R, [7:0] VGA_G, [7:0] VGA_B,VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, [1:0] OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK

    Inout: [15:0] OTG_DATA, [31:0] DRAM_DQ

    Description: Top-level file

    Purpose: Make connections with VGA Interface, CYC67200 Interface and SDRAM Interface for NIOS II Software

2. **Module**: hpi_io_intf.sv

    Inputs: Clk, Reset, [1:0] from_sw_address, [15:0] from_sw_data_out, from_sw_r, from_sw_w, from_sw_cs

    Outputs: [15:0] from_sw_data_in, [1:0] OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_R

    Inout: [15:0] OTG_DATA

    Description: Hardware Tristate Buffer

Purpose: To connect the PIO's from NIOS II processor to the pins that controls CY7C67200 chip

3. **Module**: VGA_controller.sv

   Inputs: Clk, Reset, VGA_CLK

   Outputs: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, [9:0] DrawX, DrawY

   Description: VGA controller

   Purpose: Controlling the VGA signal to display

4. **Module**: HexDriver.sv

   Inputs: [3:0] In0

   Outputs: [6:0] Out0

   Description: List all the cases of FPGA output with the matching hexadecimal input

   Purpose: To show the Hex on FPGA board

5. **Module**: Color_Mapper.sv

   Inputs: is_ball, [9:0] DrawX, DrawY

   Outputs: [7:0] VGA_R, VGA_G, VGA_B

   Description: Assign color to the interface

   Purpose: To decide which color to be the output to VGA for each pixel

6. **Module**: nios_system.v

Inputs: clk_clk, [15:0] otg_hpi_data_in_port, reset_reset_n

Outputs: [15:0] keycode_export, [1:0] otg_hpi_address_export, otg_hpi_cs_export, [15:0] otg_hpi_data_out_port, otg_hpi_r_export, otg_hpi_w_export, sdram_clk_clk, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, [3:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n

Inout: [31:0] sdram_wire_dq

Description: Qsys generated module

Purpose: I/O for CPU

7. **Module**: nios_system.v

   Inputs: clk_clk, [15:0] otg_hpi_data_in_port, reset_reset_n

   Outputs: [15:0] keycode_export, [1:0] otg_hpi_address_export, otg_hpi_cs_export, [15:0] otg_hpi_data_out_port, otg_hpi_r_export, otg_hpi_w_export, sdram_clk_clk, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, [3:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n

   Inout: [31:0] sdram_wire_dq

   Description: Qsys generated module

   Purpose: I/O for CPU

8. **Module:** GameState.sv

   Inputs: [15:0] keycode, input KEY, frame_clk,Reset, [2:0] bi_hair, bi_clothes

   Outputs:[3:0] select,[1:0] f_skin,m_skin,bi_skin,f_hair,m_hair,f_clothes,m_clothes,

   Description: Having total 24 states to control the progress of the game

   Purpose: To shift between different interfaces

9. **Module:** Picture_MUX.sv

Inputs: [23:0] intro_in, name_in, help_in, gender_in, [23:0] fskin_in, mskin_in, bskin_in, [23:0] fhair_in, mhair_in, bhair_in, [23:0] fclothes_in, mclothes_in, bclothes_in, [23:0] finish_in, [3:0] select

Outputs: [7:0] Red, Green, Blue

Description: a multiplexor for all different interfaces

Purpose: To select a specific interface to present

10. **Module:** IntroInterface.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Introduction interface of the game

    Purpose: Help to display the first interface

11. **Module:** Gender.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs:[7:0] Red, Green, Blue

    Description: Stored the Gender choosing interface of the game

    Purpose: Help to display the gender choosing interface

12. **Module:** FemaleSkin.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing Female Skin

    Purpose: Help to display the Interface for Choosing Female Skin

13. **Module:** BiSkin.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing Bi Skin

    Purpose: Help to display the Interface for Choosing Bi Skin

14. **Module:** MaleSkin.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing Male Skin

    Purpose: Help to display the Interface for Choosing Male Skin

15. **Module**: FemaleHair.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing female hair

    Purpose: Help to display the Interface for Choosing female hair

16. **Module**: BiHair.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing bi hair

    Purpose: Help to display the Interface for Choosing bi hair

17. **Module**: MaleHair.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing male hair

    Purpose: Help to display the Interface for Choosing male hair

18. **Module**: FemaleClothes.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing female clothes

    Purpose: Help to display the Interface for Choosing female clothes

19. **Module**: MaleClothes.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing male clothes

    Purpose: Help to display the Interface for Choosing male clothes

20. **Module**: BiClothes.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing bi clothes

    Purpose: Help to display the Interface for Choosing bi clothes

21. **Module**: FemaleClothes.sv

    Inputs: Clk, [9:0] DrawX, DrawY

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the Interface for Choosing female clothes

    Purpose: Help to display the Interface for Choosing female clothes

22. **Module**: Finish.sv

    Inputs: input Clk, [9:0] DrawX, DrawY, [1:0] f_skin, bi_skin, m_skin, [1:0] f_hair, m_hair, [1:0] f_clothes, m_clothes, [2:0] bi_hair, bi_clothes,

    Outputs: [7:0] Red, Green, Blue

    Description: Stored the chosen figures from previous states

    Purpose: To display the final figure from the choices made before