Final Project Report

Project Title: Fastest Finger First
Contributions: Maitreyi Joshi and Abhinav Sanjeeva Prasad

Section 1.0
Introduction

The goal of the project was to build a multiplayer buzzer quiz game, where every player has to answer each question correctly and as quickly as possible. The player with the quickest response and the most correct answers is the winner of the game. Thus, the name of the game is titled "Fastest Finger First". These types of games are commonly found in many game shows on the television for eg, Temptation, Who Wants to Be a Millionaire, etc.

In many buzzer games, we often encounter cases where at least two or more players press the buzzer around the same time, having a really small time difference. In these cases, it is somewhat impossible for a person to judge who pressed the buzzer first. Nonetheless, with the aid of digital system electronics, it is possible to measure time with a precision of picoseconds hence, allowing us to make decisions more accurately and fairly.

The basic gameplay: there will be 5 multiple choice questions where each question has 4 options out of which only one option is correct. There will be 3 players who will answer each question with the keyboard or the switches. Moreover, the keys of the fpga will act as a buzzer for the players. The question is displayed on the VGA, player's correct response is displayed on the HEX and the LEDS, and at the end of the 5 questions, the winner is displayed on the HEX display.

Section 2.0
The Design

Sub Module-1.Game-Control(Figure 1.1)

a.Timer circuit.

1. Functionality-Select the player who takes the minimum time to answer the given question.
2. Working-A counter clocked by 50 Mhz clock on the De-1Soc, using the KEYS on the board as inputs, is used to select the player who answers the fastest.
3. Each of the 3 players are assigned a separate counter-which keeps counting the number of clock cycles passed until the player presses the KEY.The result is sent to the main module.
4. The counter is reset at the beginning of each game cycle and also after every question. Hence after every question the time taken by a particular player to answer that question is stored in a register in the main module.
5. Once a player has given inputs for a question, no further buzzer beats are recorded until the next question.

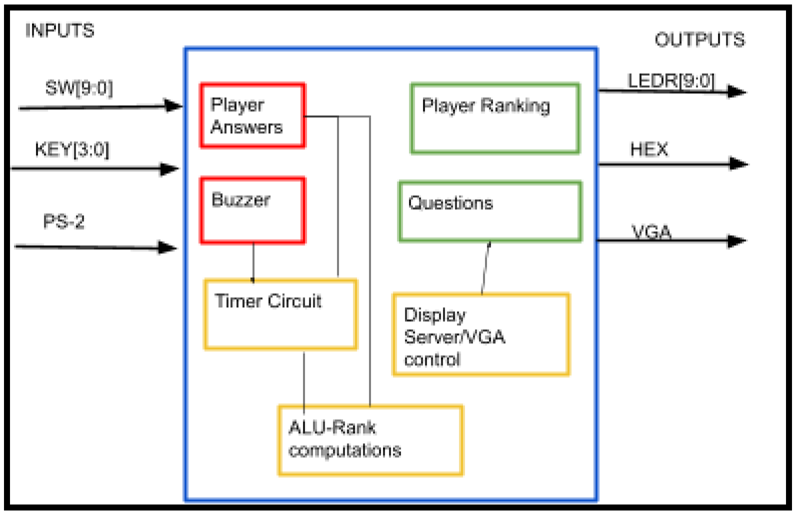


Figure 2.1-The top level module of the design

b.Correct Answer verification circuit

1. Functionality-Verify that the player has selected the correct option for a question. The correct answers are coded for in the main module, and compared to the player responses (coming from the switches and the keyboard(PS-2).

c.Final Rank Computation

1. Functionality- Compute the rank of the player based on the number of questions correctly answered and the time taken to answer to those questions.
2. The time taken to answer each question is stored in registers in the main module and is summed up using an Arithmetic Logic Unit. A counter is used to track each players correct responses to the 5 questions.
3. When the game ends, a comparator is used to compare the time taken by the different players to answer the 5 questions. A higher priority is given to the player with more correct answers than a player who answers faster. If two or more players answered the same number of questions correctly, the comparator circuit assigns a higher rank to player with the minimum time.
4. This output is then sent to the HEX Display to display the rank of the player .

d.Game flow logic

1. A finite state machine is used to track the various stages in the game flow including the begin, end of game and changing from one question to another.
2. The FSM also sends appropriate feedback signals to VGA adaptor to display the background, followed by the questions, one after the other.

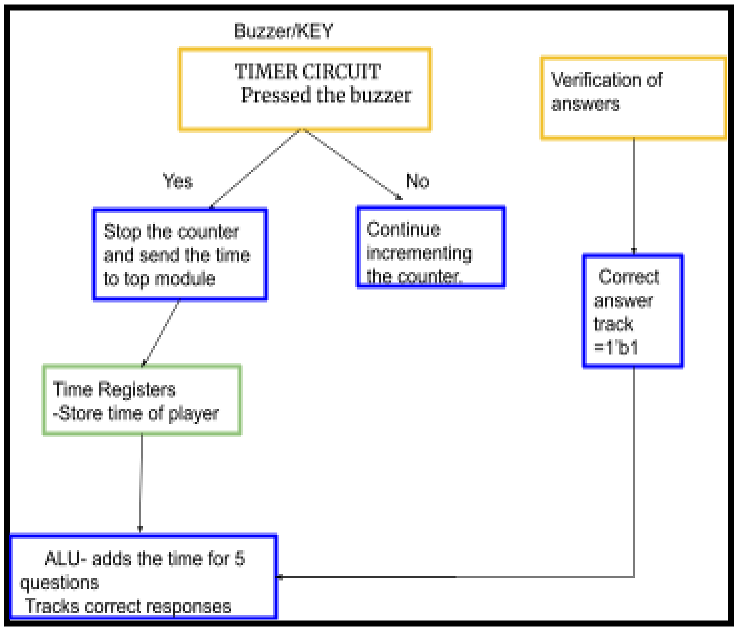

Buzzer/KEY
TIMER CIRCUIT Pressed the buzzer
Verification of answers
Yes
No
Stop the counter and send the time to top module
Continue incrementing the counter.
Correct answer track =1'b1
Time Registers -Store time of player
ALU- adds the time for 5 questions Tracks correct responses

Figure 2.2 VGA to display the questions and the start of the game.

VGA Display: In this project, the VGA displayed the start of the game and the questions with the options. To draw onto the VGA, a finite state machine was formulated ,consisting of control path , data path and the vga adapter module. In order to draw pixels on the VGA, several steps were carried out as shown in the diagrams below. (refer schematic in figure-2.3 & 2.4).

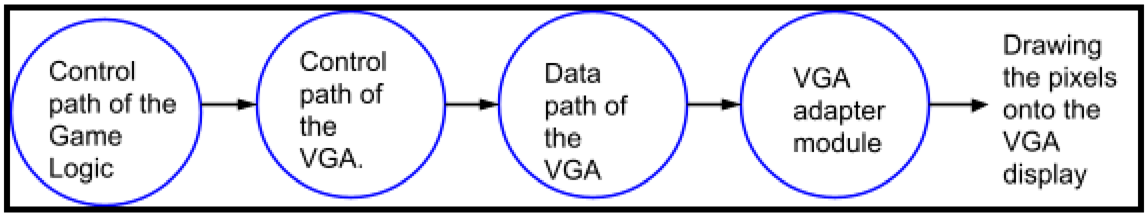

Control path of the Game Logic
Control path of the VGA.
Data path of the VGA
VGA adapter module
Drawing the pixels onto the VGA display

Figure2.3: The diagram describes the path taken to draw the pixels onto the VGA.

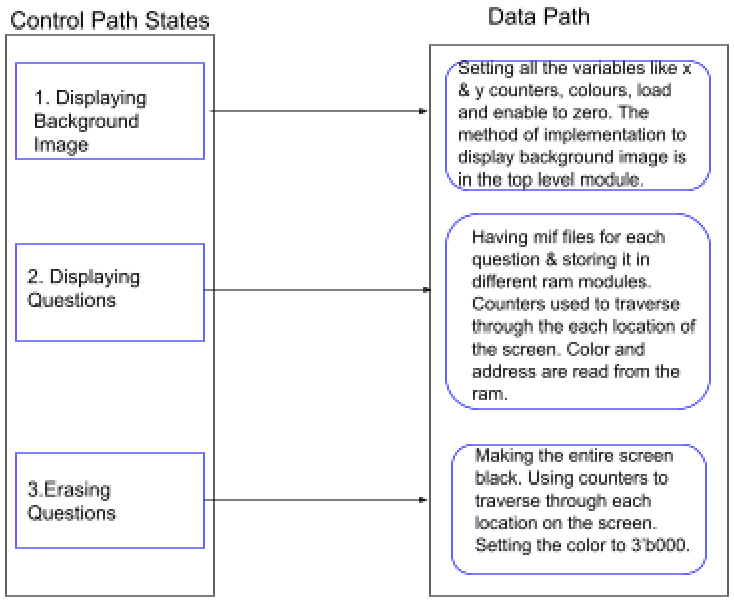

Control Path States
Data Path
1. Displaying Background Image
Setting all the variables like x & y counters, colours, load and enable to zero. The method of implementation to display background image is in the top level module.
2. Displaying Questions
Having mif files for each question & storing it in different ram modules. Counters used to traverse through the each location of the screen. Color and address are read from the ram.
3.Erasing Questions
Making the entire screen black. Using counters to traverse through each location on the screen. Setting the color to 3'b000.

Figure2.4: Overview of the VGA's control and data path.

a. Displaying Background Image

The background image of the game signifies the start of the game. Drawing of the background image is done by converting the background image in the mif file and setting the parameter "VGA.BACKGROUND_IMAGE" as the mif file name of the background image.

b. Displaying the questions

Components needed: RAM, counters, vga adapter and a multiplexer.

Every question is stored in different ram modules. The output port of each ram is set to 8 bits and each ram has 76800 words(since the resolution 320x240=76800).

The questions and the background are of the same size. When the load signal for a specific question is high, the net counter starts incrementing by 1 bit in every positive edge of the 50MHz clock cycle. The net counter increments at every positive edge of the clock cycle, until it reaches the decimal value of 76800 . When the net counter reaches the value of 76800, the process of drawing pixel stops.

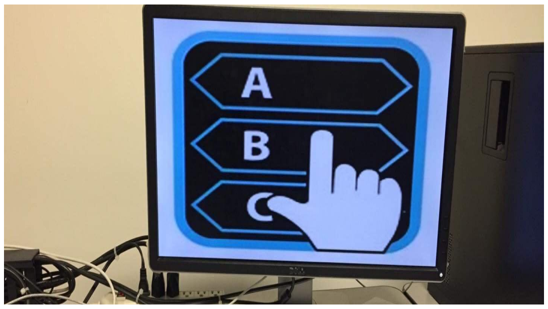
In order to draw a pixel at a specific coordinate/location of the screen, an address in terms of x and y counter (x and y counters represent the coordinates of the pixel on the screen) is passed to the address port of the instantiated ram(see appendix, page 26, assignment of address). The address is sent in a way that the ram can reach the location of the pixel inside the memory. It then reads the value of the output stored in that particular address. The value of the output at that location is the value of the colour of the pixel.

In the top level module a multiplexer is formed, the function of the multiplexer is to select which question to display/erase. A question is selected when the load signal of that particular question is high. The multiplexer assigns the x, y and the colour values of the VGA adapter to the x,y counter and the colour values of the selected question. (See appendix, page 11, the always block is basically a mux).

3.0 Report on Success

In this project, we successfully implemented the entire game. We successfully took all the inputs from the keyboard, switches and the buzzer(the keys) and displayed the correct answer on the LEDs/HEX.We were able to compute the rankings and display the winner on the HEX. We succeeded in drawing the background image and the questions onto the vga and also gained success while interfacing the vga with the game logic. Nonetheless, due to the limitation of the available memory we could not load all the images into the ram. Hence, we redesigned our project with fewer images.

3.1 Pictures of the working project



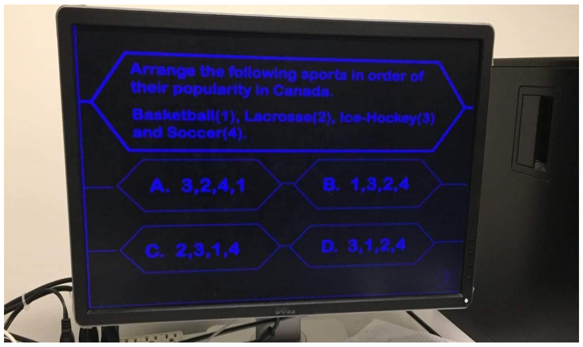
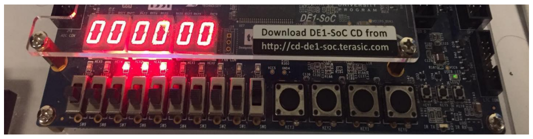
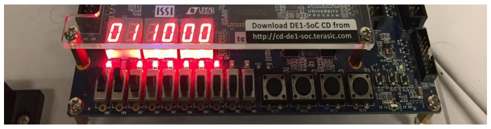
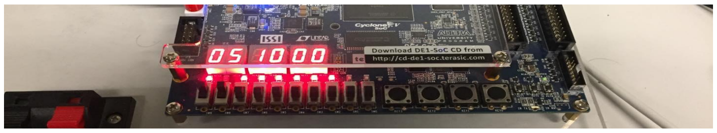

Arrange the following sports in order of their popularity in Canada.
Basketball(1), Lacrosse(2), Ice-Hockey(3) and Soccer(4).
A. 3,2,4,1
B. 1,3,2,4
C. 2,3,1,4
D. 3,1,2,4

Figure 3.1: The game start display.

Figure3.2: Drawing of the question

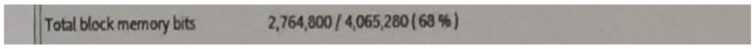






Figures 3.3-3.5-from top left- rank at the beginning of the game,rank after answering the first question, final ranking(Player C has rank 1)

After loading all the images of the questions into the ram and compiling the project, we discovered an error message stating that the ram cannot fit all the data stored. Due to this, we were not able to display all the images onto the VGA. After inspecting the problem, we discovered that loading background image and drawing one question alone occupies the 68% of the memory(See Figure 3.6). Thus, we had to redesign our project for fewer images.



Total block memory bits 2,764,800 / 4,065,280 (68 %)

Figure 3.6: The total block memory bits occupied is 68%.

4.0 What would you do differently ?

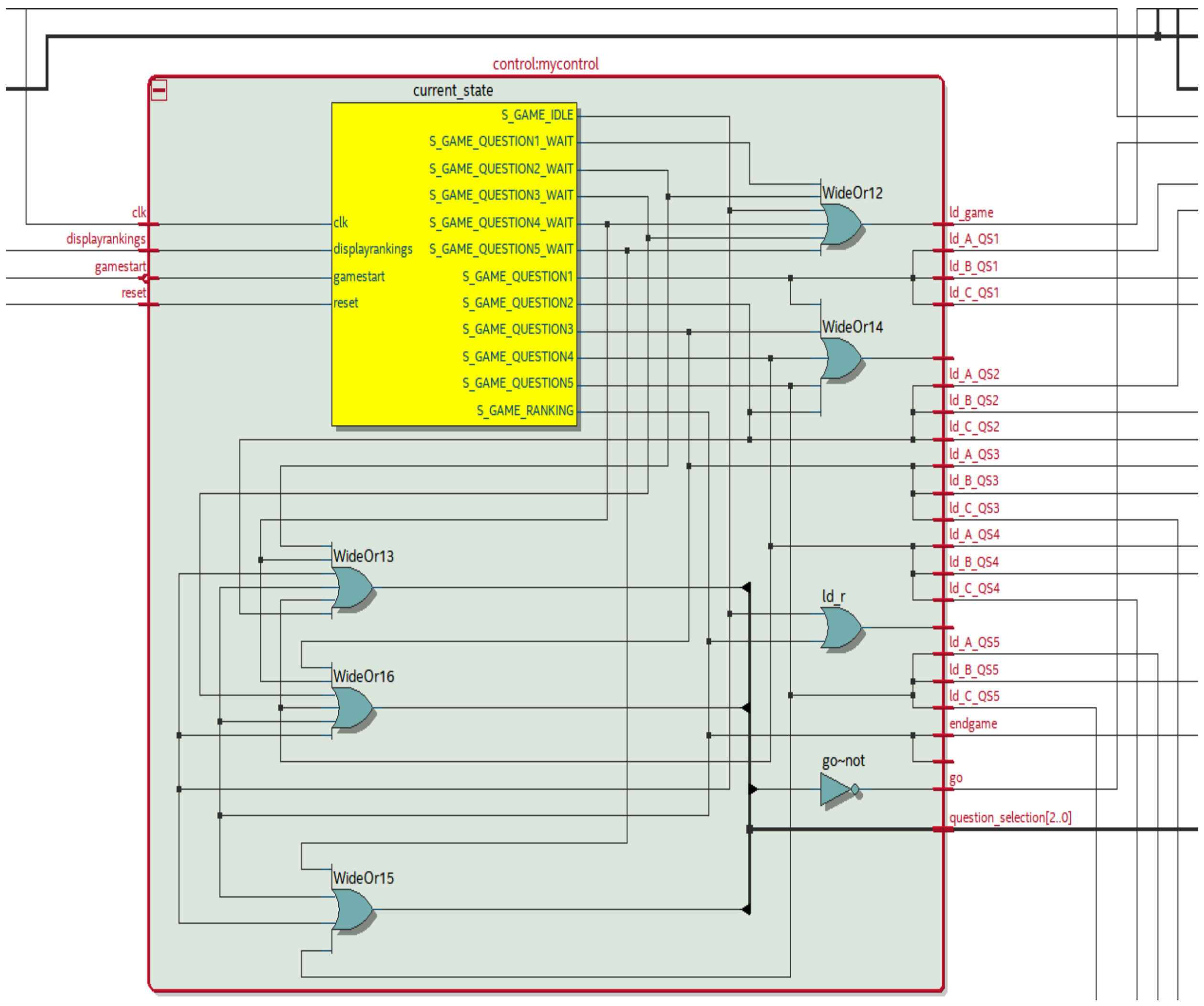
a. We will change our approach to test a module, by running the simulations first. Our usual approach was programming the fpga and running the circuit.

b. We would have a prior knowledge of how a memory is allocated in the fpga, how large is the memory space, how much memory space each image will occupy and then strategically devise a plan on how to successfully store all the images in the memory without exceeding its maximum limit.

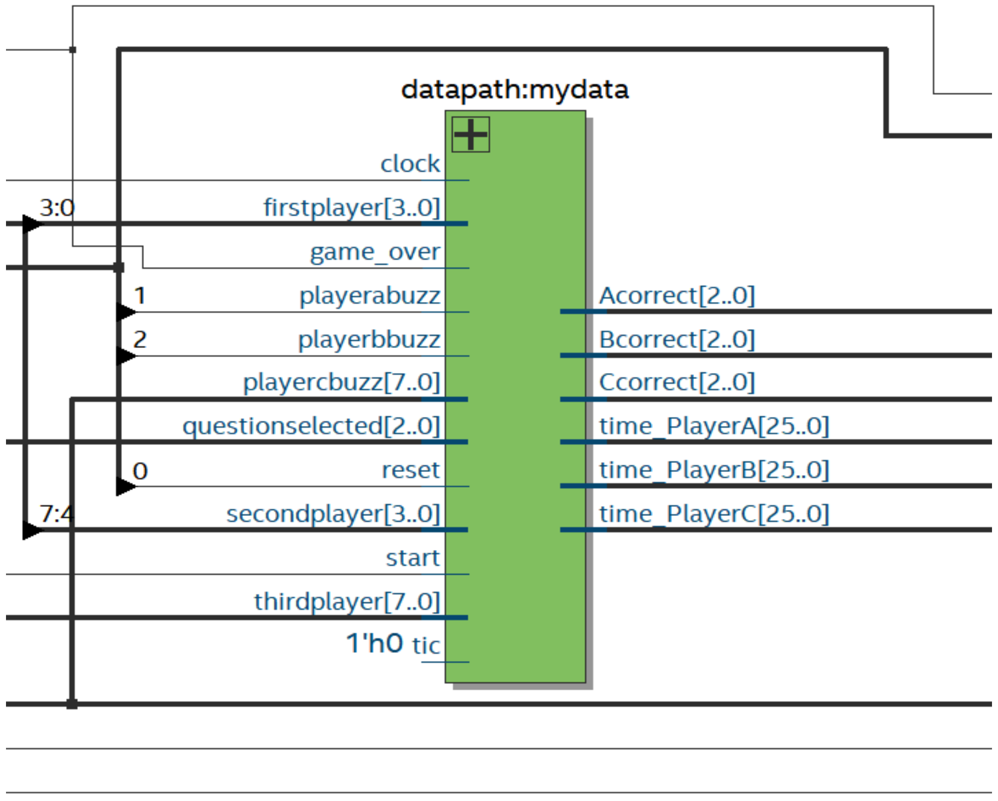
c. We would plan to begin with the VGA module first and then the game logic, as that would help us reduce the workload in the final week of project.

Appendix

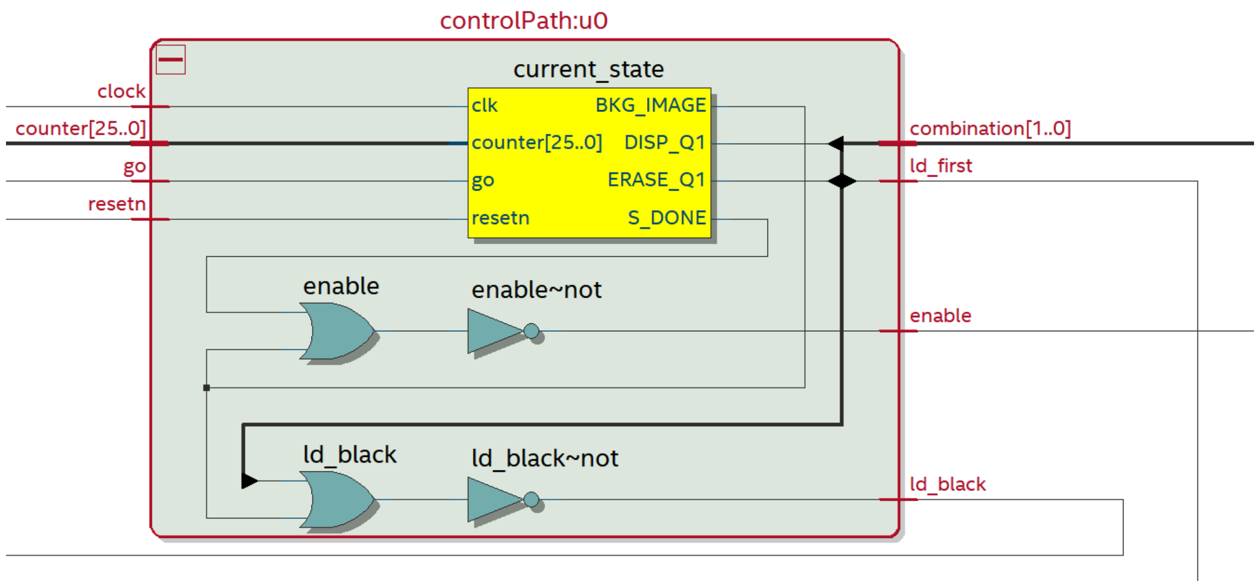
Schematics-Game Control Module



Game Datapath module



VGA Controller



Code:

Fastestfingerfirst.v

```
module Fastestfingerfirst( SW
,KEY,CLOCK_50,LEDR,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,PS2_CLK,PS2_DAT,VGA_CLK,
                              //     VGA Clock
              VGA_HS,                                         //     VGA H_SYNC
              VGA_VS,                                         //     VGA V_SYNC
              VGA_BLANK_N,                                    //     VGA BLANK
              VGA_SYNC_N,                                     //     VGA SYNC
              VGA_R,                                          //     VGA Red[9:0]
              VGA_G,                                          //     VGA Green[9:0]
              VGA_B );                                        //     VGA Blue[9:0])

              input [9:0]SW;
              input[3:0]KEY;
                input CLOCK_50;
              output [6:0]HEX0;
              output [6:0]HEX1;
              output [6:0]HEX2;
              output [6:0]HEX3;
              output [6:0]HEX4;
              output [6:0]HEX5;
              output [6:0]HEX6;
              output [9:0]LEDR;
              wire govga;




               reg [23:0] colour; // parameters of the vga adapter
                reg [8:0] x;
                reg[7:0] y;
               wire writeEn;
               wire [3:0]playerAanswer;//options of player a on switch
               wire [3:0]playerBanswer;//options of player b on switch
               wire [7:0]playerCanswer;//options of player C-7 bit wide comming from the
controller

               //wire resetfordatapath;//afterevery qs
               wire clock;//clock_50

               wire ldA;//load signal for reg A to store time from control
               wire ldB;//load signal
               wire ldC;
               wire ldR;
               wire Abuzzer;//key pressed by player
```

```
             wire Bbuzzer;//key pressed by player
             wire [7:0]Cbuzzer;
             wire game_start;//switch 9
             wire ranking_display;//SW[8]
             wire resetgame;//for controller



             //10 wires or load signals from controller
             wire ldAQS1,ldAQS2,ldAQS3,ldAQS4,ldAQS5,ldBQS1,ldBQS2,ldBQS3,ldBQS4,ldBQS5;
             wire ldCQS1,ldCQS2,ldCQS3,ldCQS4,ldCQS5;

             reg [25:0]timeA1;
             reg [25:0]timeA2;
             reg [25:0]timeA3;
             reg [25:0]timeA4;
             reg [25:0]timeA5;
             reg [25:0]timeB1;
             reg [25:0]timeB2;
             reg [25:0]timeB3;
             reg [25:0]timeB4;
             reg [25:0]timeB5;
             reg [25:0]timeB6;


             reg [25:0]timeC1;
             reg [25:0]timeC2;
             reg [25:0]timeC3;
             reg [25:0]timeC4;
             reg [25:0]timeC5;

            // Declare your inputs and outputs here
            // Do not change the following outputs
            output              VGA_CLK;                        //     VGA Clock
            output              VGA_HS;                         //     VGA H_SYNC
            output              VGA_VS;                         //     VGA V_SYNC
            output              VGA_BLANK_N;                    //     VGA BLANK
            output              VGA_SYNC_N;                     //     VGA SYNC
            output  [7:0]  VGA_R;                         //     VGA Red[7:0] Changed
from 10 to 8-bit DAC
            output  [7:0]  VGA_G;                         //     VGA Green[7:0]
            output  [7:0]  VGA_B;                         //     VGA Blue[7:0]


            //instantiating the VGA adapter
            vga_adapter VGA(
                   .resetn(resetgame),
                   .clock(CLOCK_50),
            .colour(colour),
		.x(x),
			.y(y),
			.plot(writeEn),

			.VGA_R(VGA_R),
			.VGA_G(VGA_G),
			.VGA_B(VGA_B),
			.VGA_HS(VGA_HS),
			.VGA_VS(VGA_VS),
		.VGA_BLANK(VGA_BLANK_N),
		.VGA_SYNC(VGA_SYNC_N),
			.VGA_CLK(VGA_CLK));

		defparam VGA.RESOLUTION = "320x240";
		defparam VGA.MONOCHROME = "FALSE";
		defparam VGA.BITS_PER_COLOUR_CHANNEL = 8;
		defparam VGA.BACKGROUND_IMAGE = "Front5";


		wire [1:0]select; //select signals will be used by the multiplexer to select
which question to display or erase


		wire load_First; //load signal for the first question
		wire  load_Black; //load signal for erasing the question
		wire [23:0] colorF;
		wire [23:0] colorS;
		wire [8:0] xF;
		wire [7:0] yF;
		wire [8:0] xS;
		wire [7:0] yS;
		wire [7:0] yB;
		wire [8:0] xB;
		wire [23:0] colorB;


		wire[25:0] main; //main is just a counter that counts till 76800 clock cycles

		//Instantiating controlPath of the VGA
		controlPath u0 (.counter(main),.enable(writeEn), .clock(CLOCK_50),
		.go(govga), .resetn(resetgame), .ld_first(load_First),
		.ld_black(load_Black),.combination(select));

		//instantiating the dataPath of the VGA
		dataPath u1 (.counterF(main),.clock(CLOCK_50), .go(change), .resetn(resetgame),
.ld_first(load_First), .ld_black(load_Black),
				.x_outF(xF),  .y_outF(yF),  .color_outF(colorF),
.x_outB(xB), .y_outB(yB), .color_outB(colorB));
```

```
		//This is a multiplexer which basically selects which question should be drawn
or erased on the VGA
		always@(CLOCK_50)
		begin
		case(select)
		2'b00:begin
			//DISPLAY BKG IMAGE
		end
		2'b01:begin
			x=xF;
			y=yF;
			colour=colorF;
		end
		2'b10:begin
			x=xB;
			y=yB;
			colour=colorB;
		end
		 default:begin
			//DISPLAY BKG IMAGE
		end

		endcase
		end


		//similarly registers to store correct answers of players
		 reg correctA1;
		 reg correctA2;
		 reg correctA3;
		 reg correctA4;
		 reg correctA5;
		 reg correctB1;
		 reg correctB2;
		 reg correctB3;
		 reg correctB4;
		 reg correctB5;
		  reg correctc1;
		 reg correctc2;
		 reg correctc3;
		 reg correctc4;
		 reg correctc5;




		inout PS2_CLK;

		 inout PS2_DAT;
```

```
 wire         [7:0]  ps2_key_data;
 wire         ps2_key_pressed;
 reg [7:0] last_data_received;

 always@(posedge CLOCK_50)
 begin
 if(KEY[0]==1'b0)
   last_data_received<=8'h00;
else if(ps2_key_pressed==1'b1)
       last_data_received<=ps2_key_data;
end

 PS2_Controller PS2 (
              // Inputs
              .CLOCK_50(CLOCK_50),
              .reset(~KEY[0]),

              // Bidirectionals
              .PS2_CLK(PS2_CLK),
              .PS2_DAT(PS2_DAT),

              // Outputs
              .received_data(ps2_key_data),
              .received_data_en(ps2_key_pressed)
);



 reg assignedrankA;//variable that says rank has been assigned dont enter again
 reg assignedrankB;
 reg assignedrankC;

 wire[25:0]timeA;//comming from datapath-can contain time for different qs
 wire[25:0]timeB;//comming from datapath
 wire [25:0]timeC;
 reg [25:0]data_resultA;//storing the total time


 reg [25:0]data_resultPlayerB;
  reg [25:0] data_resultC;                          //storing total time of B
 reg [2:0]data_correctA;//register storing the count of player a'scorrect
answers
 reg [2:0]data_correctB;
 reg [2:0]data_correctC;
```

```
                wire [2:0]playerAtrack;
                wire [2:0]playerBtrack;
                wire [2:0]playerCtrack;



                reg [2:0]correct_counterA;
                reg [2:0] correct_counterB;
                reg [2:0]correct_counterC;
                //internal wires

                wire begingame;//from controller to datapath
                wire gameover;//from controller to top level module

                wire [2:0]questions_selected;//the current questionselected controller to
datapath

                wire enableforclock;
                reg PlayerArank;
                reg PlayerBrank;
                reg PlayerCrank;



                assign playerAanswer=SW[3:0];
                assign playerBanswer=SW[7:4];
                assign playerCanswer=last_data_received[7:0];
                assign resetgame=KEY[0];//reset for the controller

                assign Abuzzer=KEY[1];
                assign Bbuzzer=KEY[2];
                assign Cbuzzer=ps2_key_data[7:0];//from the keyboard-input third player

                //assign game_start=SW[9];
                assign game_start=~KEY[3];
                assign ranking_display=SW[8];

                wire tictic;


               datapath
mydata(.clock(CLOCK_50),.Acorrect(playerAtrack),.Bcorrect(playerBtrack),.Ccorrect(playerCtrac
k),

.tic(tictic),.reset(resetgame),.firstplayer(playerAanswer),.secondplayer(playerBanswer),.thir
dplayer(playerCanswer),

.time_PlayerA(timeA),.time_PlayerB(timeB),.time_PlayerC(timeC),.start(begingame),
.questionselected(questions_selected),.playerabuzz(Abuzzer),.playerbbuzz(Bbuzzer),.playercbuz
z(Cbuzzer),
                      .game_over(gameover));


               control
mycontrol(.go(govga),.cctenable(enableforclock),.clk(CLOCK_50),.reset(resetgame),.gamestart(g
ame_start),.displayrankings(ranking_display),

.ld_game(begingame),.question_selection(questions_selected),.endgame(gameover),.ld_r(ldR),

                                            .ld_A_QS1(ldAQS1),
                                            .ld_A_QS2(ldAQS2),
                                            .ld_A_QS3(ldAQS3),
                                            .ld_A_QS4(ldAQS4),
                                            .ld_A_QS5(ldAQS5),
                                            .ld_B_QS1(ldBQS1),
                                            .ld_B_QS2(ldBQS2),
                                            .ld_B_QS3(ldBQS3),
                                            .ld_B_QS4(ldBQS4),
                                            .ld_B_QS5(ldBQS5),
                                            .ld_C_QS1(ldCQS1),
                                            .ld_C_QS2(ldCQS2),
                                            .ld_C_QS3(ldCQS3),
                                            .ld_C_QS4(ldCQS4),
                                            .ld_C_QS5(ldCQS5));





   //compute rankings
   //this is the time register for both players for all the questions s
   always@(posedge CLOCK_50) begin
        if(!resetgame) begin

               correct_counterA<=3'b000;
               correct_counterB<=3'b000;


                timeA1<=25'b0;
                timeA2<=25'b0;
                timeA3<=25'b0;
                timeA4<=25'b0;
                timeA5<=25'b0;
                timeB1<=25'b0;
                 timeB2<=25'b0;
                timeB3<=25'b0;
```

```
          timeB4<=25'b0;
          timeB5<=25'b0;
          timeB6<=25'b0;
          correctA1<=1'b0;
          correctA2<=1'b0;
          correctA3<=1'b0;
          correctA4<=1'b0;
           correctA5<=1'b0;
           correctB1<=1'b0;
          correctB2<=1'b0;
          correctB3<=1'b0;
          correctB4<=1'b0;
          correctB5<=1'b0;

            timeC1<=25'b0;
            timeC2<=25'b0;
           timeC3<=25'b0;
           timeC4<=25'b0;
           timeC5<=25'b0;




         correctc1<=1'b0;
         correctc2<=1'b0;
         correctc3<=1'b0;
         correctc4<=1'b0;
          correctc5<=1'b0;


end

 else begin
          if(ldAQS1)
          begin
                timeA1<=timeA  ;
                correctA1<=playerAtrack;
         end
          if(ldAQS2)
          begin
                timeA2<=timeA  ;
                correctA2<=playerAtrack;
          end


         if(ldAQS3)
          begin
                timeA3<=timeA;
                correctA3<=playerAtrack;
```

```
end
if(ldAQS4)
  begin
        timeA4<=timeA  ;
        correctA4<=playerAtrack;
end
if(ldAQS5)
  begin
        timeA5<=timeA;
        correctA5<=playerAtrack;
end

if(ldBQS1)
  begin
        timeB1<=timeB;
        correctB1<=playerBtrack;
end
if(ldBQS2)
  begin
        timeB2<=timeB;
        correctB2<=playerBtrack;
end
if(ldBQS3)
  begin
        timeB3<=timeB;
        correctB3<=playerBtrack;
 end
if(ldBQS4)
  begin
        timeB4<=timeB;
        correctB4<=playerBtrack;
end


if(ldBQS5)
  begin
        timeB5<=timeB;
        correctB5<=playerBtrack;
end


if(ldCQS1)
  begin
        timeC1<=timeC;
        correctc1<=playerCtrack;
end
if(ldCQS2)
  begin
        timeC2<=timeC;
```

```
                    correctc2<=playerCtrack;
            end
            if(ldCQS3)
              begin
                    timeC3<=timeC;
                    correctc3<=playerCtrack;
            end
            if(ldCQS4)
              begin
                    timeC5<=timeC;
                    correctc4<=playerCtrack;
             end
            if(ldCQS5)
                    begin
                    timeC5<=timeC;
                    correctc5<=playerCtrack;
             end




    end
    end




    // The ALU
    always @(*)
    begin : ALU
     // alu


      data_resultA=timeA1+timeA2+timeA3+timeA4+timeA5;
      data_resultPlayerB=timeB1+timeB2+timeB3+timeB4+timeB5;
      data_correctA=correctA1+correctA2+correctA3+correctA4+correctA5;
      data_correctB=correctB1+correctB2+correctB3+correctB4+correctB5;
      data_resultC=timeC1+timeC2+timeC3+timeC4+timeC5;
      data_correctC=correctc1+correctc2+correctc3+correctc4+correctc5;
   end





//final results computation logic
always@( *)

   begin
       if(!resetgame)
```

```
         begin
                   PlayerArank<=1'b0;
                   PlayerBrank<=1'b0;
                  assignedrankA<=1'b0;
                  assignedrankB<=1'b0;
                  PlayerCrank<=0;
                   assignedrankC<=0;


                                    end
       else
begin
       if(gameover)//only whwen controlled signals end of game compare


         begin
           if(assignedrankA==1'b0 & assignedrankB==1'b0 )
               begin
                            if(data_correctA<data_correctB)//b answwered more questions
correctly
                               begin
                                   PlayerArank<=1'b0;
                                   PlayerBrank<=1'b1;
                                   assignedrankA<=1'b1;
                                   assignedrankB<=1'b1;
                                   end
                            else if(data_correctA>data_correctB)
                                begin
                                   PlayerArank<=1'b1;
                                   PlayerBrank<=1'b0;
                                   assignedrankA<=1'b1;
                                   assignedrankB<=1'b1;
                                   end







       else
              begin

                      if(data_resultA<data_resultPlayerB)//player A took less time to answer
                         begin
                               PlayerArank<=1'b1;
                               PlayerBrank<=1'b0;
                            assignedrankA<=1'b1;
                            assignedrankB<=1'b1;
```

```
                          end
                     else if (data_resultA>data_resultPlayerB)//in case player B took more
time to answer
                       begin
                          PlayerArank<=1'b0;
                          PlayerBrank<=1'b1;
                         assignedrankA<=1'b1;
                          assignedrankB<=1'b1;

                           end
                     else
                         begin
                           PlayerArank<=1'b0;
                           PlayerBrank<=1'b0;
                          assignedrankA<=1'b1;
                           assignedrankB<=1'b1;

                         end

                  end

              end
              if(assignedrankC==0 &&assignedrankB==1 && assignedrankA==1)// a and B rank
assignement done
                    begin
                            if(PlayerArank==1'b1)
                             begin
                                   if(data_correctC>data_correctA)
                                      begin
                                             PlayerCrank<=1;
                                             assignedrankC<=1;
                                            PlayerArank<=0;//overwrite the rank of player A
                                       end
                                else if(data_correctC<data_correctA)
                                       begin
                                           PlayerCrank<=0;
                                           assignedrankC<=1;//dont overwrite
                                        end
                                else//if both C and A answered the same no of qs correctly
                                   begin
                                      if(data_resultC<data_resultA)//case where C took less
time to answer
                                       begin
                                           PlayerCrank<=1'b1;
                                           assignedrankC<=1;
                                           PlayerArank<=0;
                                       end
                                     else if (data_resultC>data_resultA)
                                       begin
```

```
                                            PlayerCrank<=0;
                                            assignedrankC<=1;//dont overwrite
                                      end
                                      else
                                    begin
                                    end

                            end
                       end

                else if (PlayerBrank==1'b1)
                         begin
                                        if(data_correctC>data_correctB)
                                          begin
                                                   PlayerCrank<=1;
                                                   assignedrankC<=1;
                                                   PlayerBrank<=0;
                                         end
                                        else if(data_correctC<data_correctB)
                                             begin
                                                          PlayerCrank<=0;
                                                          assignedrankC<=1;
                                             end
                                         else//if both C and A answered the same no of qs
correctly
                                            begin
                                             if(data_resultC<data_resultPlayerB)//case where C
took less time to answer
                                              begin
                                                   PlayerCrank<=1'b1;
                                                   assignedrankC<=1;
                                                   PlayerBrank<=0;
                                              end
                                          else if (data_resultC>data_resultPlayerB)
                                             begin
                                                      PlayerCrank<=0;
                                                      assignedrankC<=1;
                                            end
                                          else
                                              begin
                                             end


                              end
               end

              else //case where none of the players A or B are assigned a rank of 1
              begin
```

```
       if (data_correctC>data_correctA &data_correctC>data_correctB)
       begin
              PlayerCrank<=1'b1;
              assignedrankC<=1'b1;
              PlayerArank<=1'b0;
              PlayerBrank<=1'b0;
              assignedrankA<=1'b1;
              assignedrankB<=1'b1;
       end
else if(data_resultC<data_resultPlayerB & data_resultC<data_resultA)
begin
       PlayerCrank<=1'b1;
       assignedrankC<=1'b1;
       PlayerArank<=1'b0;
       PlayerBrank<=1'b0;
       assignedrankA<=1'b1;
       assignedrankB<=1'b1;
end

else
begin
       PlayerCrank<=1'b0;
       assignedrankC<=1'b1;
end

end
end



       end


end

end




assign LEDR[0]=PlayerArank;

assign LEDR[1]=PlayerBrank;
assign LEDR[2]=PlayerCrank;
assign LEDR[4]=assignedrankA;
assign LEDR[3]=assignedrankC;
assign LEDR[5]=questions_selected[0];//for debugging
assign LEDR[6]=questions_selected[1];
assign LEDR[7]=questions_selected[2];
```

```
//assign LEDR[8]=Cbuzzer;
assign LEDR[9]=assignedrankB;
hex_decoder c1(.hex_digit(data_correctB[2:0]),.segments(HEX2));

hex_decoder d2(.hex_digit(data_correctC[2:0]),.segments(HEX4));

hex_decoder d3(.hex_digit(data_correctA[2:0]),.segments(HEX5));






reg [3:0]display1;
reg[3:0]display2;
reg [3:0] display3;
always@(posedge CLOCK_50)
begin

if(!resetgame)
begin
       display1<=0;
       display2<=0;
       display3<=0;
end
else
begin
if(gameover)
begin
if(PlayerArank==1'b1&PlayerBrank==0 &PlayerCrank==0)
display1=4'b0001;
else if(PlayerBrank==1'b1 &PlayerArank==0& PlayerCrank==0)
display2=4'b0001;
else if(PlayerCrank==1 && PlayerBrank==0 &PlayerArank==0)
display3=4'b0001;

else
begin
       display1=4'b0000;
       display2=4'b0000;
       display3=4'b0000;
end

end
end
end
```

```
              hex_decoder hexdisplay1(.hex_digit(display1),.segments(HEX0));//displays rank
of player one -time for debugging

              hex_decoder hexdisplay2(.hex_digit(display2),.segments(HEX1));//displays rank
of player 2


              hex_decoder d1(.hex_digit(display3),.segments(HEX3));




   endmodule




module hex_decoder(hex_digit, segments);
    input [3:0] hex_digit;
    output reg [6:0] segments;

    always @(*)
        case (hex_digit)
            4'h0: segments = 7'b100_0000;
            4'h1: segments = 7'b111_1001;
            4'h2: segments = 7'b010_0100;
            4'h3: segments = 7'b011_0000;
            4'h4: segments = 7'b001_1001;
            4'h5: segments = 7'b001_0010;
            4'h6: segments = 7'b000_0010;
            4'h7: segments = 7'b111_1000;
            4'h8: segments = 7'b000_0000;
            4'h9: segments = 7'b001_1000;
            4'hA: segments = 7'b000_1000;
            4'hB: segments = 7'b000_0011;
            4'hC: segments = 7'b100_0110;
            4'hD: segments = 7'b010_0001;
            4'hE: segments = 7'b000_0110;
            4'hF: segments = 7'b000_1110;
            default: segments = 7'h7f;
        endcase
endmodule







//control path of the VGA
module controlPath( counter,clock,go, resetn, enable, ld_first, ld_black,combination);
```

```
        input [25:0] counter; //a main counter that counts till 76800, indicating that drawing
on the entire screen has been completed
        input clock;
        input go; //signalling the transition from one state to the other
        input resetn;
        output reg enable; //enable signal implies plotting
        reg [3:0]next_state;
        reg [3:0]current_state;
        output reg [1:0]combination; //combination is a combination of a code, this defines
the select signal of the mux used for selecting which state to display.
        output reg ld_first; //load signals used to determine which state to load on the VGA

        output reg ld_black;




        localparam        BKG_IMAGE =3'd0, //defining the states of the FSM
                         DISP_Q1   =3'd1,
                         ERASE_Q1  =3'd2,
                         S_DONE    =3'd4;



        always@(*)
        begin: state_table
                case(current_state)
                        BKG_IMAGE: next_state=go?DISP_Q1:BKG_IMAGE;

                        DISP_Q1:begin
                              if(counter>18'b10010110000000000)//if the counter hasn't reached
76800 it keeps looping in the state.
                                          begin
                                              if(go)next_state=ERASE_Q1;
                                              else  next_state=DISP_Q1;
                                              end
                             else
                                         next_state=DISP_Q1;
                        end

                        ERASE_Q1: next_state = go?S_DONE:ERASE_Q1;
                    S_DONE: next_state = go?S_DONE:BKG_IMAGE;
                endcase
        end

        always@( *)
        begin: enable_signals
        //By default, all our signals are 0
        enable = 1'b0;
        ld_first = 1'b0;
        ld_black = 1'b0;
        combination=2'b11;

        case (current_state)
               BKG_IMAGE: begin
                      enable=1'b0;
                      ld_first = 1'b0;
                      ld_black = 1'b0;
                      combination=2'b00;
               end
               DISP_Q1: begin
                      ld_first = 1'b1;
                      enable = 1'b1;
                      ld_black = 1'b0;
                      combination=2'b01;
               end

               ERASE_Q1: begin

                      ld_first = 1'b0;
                      enable = 1'b1;
                      ld_black = 1'b1;
                      combination=2'b10;
               end



               S_DONE: begin
                      enable = 1'b0;
                      ld_first = 1'b0;
                      ld_black = 1'b1;
                      combination=2'b00;
               end
        endcase
        end


    // current_state registers
    always@(posedge clock)
    begin: state_FFs
        if (!resetn)
            current_state <= BKG_IMAGE;
        else
            current_state <= next_state;
    end // state_FFS
endmodule


//data path
```

```
module dataPath(counterF,clock,go, resetn, enable, ld_first, ld_black,
              x_outF,y_outF, x_outB, y_outB, color_outF,color_outB);

       input go;
       input clock;
       input resetn;
       input enable;

       input ld_first;
       input ld_black;
       reg   [8:0] X_counterFirst; //X and Y Counters for the first question
       reg   [7:0] Y_counterFirst;




       output  reg   [25:0] counterF; //this output is synced with the counter in top level
module and controlPath
                                   //counter for the first question. If it reaches 76800,
drawing is completed.


        reg   [25:0] counterB;
       reg   [8:0] X_counterBlack;
       reg   [7:0] Y_counterBlack;
       wire  [23:0]  colourFirst;

       output reg [8:0] x_outF;

       output reg [7:0] y_outF;

       output reg [8:0] x_outB;
       output reg [7:0] y_outB;

       output reg [23:0] color_outF;

       output reg [23:0] color_outB;


       wire [16:0] address;
       assign address = (17'd320)*(Y_counterFirst)+X_counterFirst; //the address is in this
form because the drawing takes place row wise.

       q1Mem  u1 (.address(address), .q(colourFirst), .wren(1'b0), .clock(clock));
//instantiating the q1 RAM


       always@ (posedge clock) begin
              if (!resetn) begin
```

```
                        X_counterFirst <= 9'd0;
                        Y_counterFirst <= 8'd0;
                        x_outB<=0;
                        y_outB<=0;
                        x_outF<=0;
                        y_outF<=0;
                        color_outB<=0;
                        color_outF<=0;
                end

                else
                begin
                if(ld_first) begin

                        if(X_counterFirst == 9'd319 && Y_counterFirst!=8'd239 ) begin
                                X_counterFirst <= 0;                    //since the width is
320, if x reaches the rightmost pixel in a row
                                Y_counterFirst <= Y_counterFirst+1'b1;          // then reset x
to 0, and increment y by 1 bit.
                        end
                        else begin
                                X_counterFirst<=X_counterFirst+1'b1;    //else keep
incrementing x
                        end
                                x_outF<=X_counterFirst;
                                y_outF<=Y_counterFirst;
                                color_outF<=colourFirst;
                                counterF <= counterF+1'b1;

                end




        else if(ld_black) begin

                         if(X_counterBlack == 9'd319 && Y_counterBlack!=8'd239 ) begin
                                X_counterBlack <= 0;
                                Y_counterBlack <= Y_counterBlack+1'b1;
                        end
                        else begin
                                X_counterBlack<=X_counterBlack+1'b1;
                        end
                                x_outB<=X_counterBlack;
                                y_outB<=Y_counterBlack;
                                color_outB<=24'b0;
                                counterB <= counterB+1'b1;

        end
```

```
end
end


endmodule
```

Control of the game logic :control.v

```
module
control(rankdisplay,go,ld_A_QS1,ld_A_QS2,ld_A_QS3,ld_A_QS4,ld_A_QS5,ld_B_QS1,ld_B_QS2,ld_B_QS
3,ld_B_QS4,ld_B_QS5,
 ld_C_QS1,ld_C_QS2,ld_C_QS3,ld_C_QS4,ld_C_QS5,
cctenable,clk,ld_r,reset,gamestart,displayrankings,ld_game,question_selection,endgame);

//gamestart is switch 9

//displayrankings is sw8

//a total of 10 load signals for the registers
output reg
ld_A_QS1,ld_A_QS2,ld_A_QS3,ld_A_QS4,ld_A_QS5,ld_B_QS1,ld_B_QS2,ld_B_QS3,ld_B_QS4,ld_B_QS5;
output reg  ld_C_QS1,ld_C_QS2,ld_C_QS3,ld_C_QS4,ld_C_QS5;

output reg  go;//to the VGA
output reg rankdisplay;//to VGA

input clk,reset,gamestart,displayrankings;
//output to the VGA need to fill this
//output to the datapath
reg [5:0] current_state;
reg [5:0]next_state;
output reg ld_game;//going to the datapath
output reg [2:0] question_selection;
//output reg ld_time_A;//the load signal for the register
//output reg ld_time_B;//the load signal for time registerB
//output reg mux_selectA;//select signal for mux on top of A register
output reg ld_r;//load signal for results register
//output reg  mux_selectB;
output reg cctenable;
output reg  endgame;//to datapath to stop comparing ans and compute the avg time
localparam      S_GAME_IDLE    = 5'd0,

                S_GAME_QUESTION1 = 5'd1,
                                    S_GAME_QUESTION1_WAIT=5'd2,
                S_GAME_QUESTION2 = 5'd3,
                                    S_GAME_QUESTION2_WAIT=5'd4,
                S_GAME_QUESTION3 = 5'd5,
                                    S_GAME_QUESTION3_WAIT=5'd6,
                S_GAME_QUESTION4 = 5'd7,
                                     S_GAME_QUESTION4_WAIT=5'd8,
                S_GAME_QUESTION5 = 5'd9,
                                     S_GAME_QUESTION5_WAIT=5'd10,
                S_GAME_RANKING      = 5'd11;


 // Next state logic aka our state table
    always@(*)
    begin: state_table
            case (current_state)
                S_GAME_IDLE: next_state = gamestart ? S_GAME_QUESTION1 : S_GAME_IDLE; // Loop
in current state until value is input

                 S_GAME_QUESTION1: next_state = gamestart ?
S_GAME_QUESTION1:S_GAME_QUESTION1_WAIT; // Loop in current state until value is input

                                      S_GAME_QUESTION1_WAIT:next_state = gamestart ?
S_GAME_QUESTION2:S_GAME_QUESTION1_WAIT;




                                    S_GAME_QUESTION2: next_state = gamestart ?
S_GAME_QUESTION2:S_GAME_QUESTION2_WAIT; // Loop in current state until go signal goes low

                                    S_GAME_QUESTION2_WAIT:next_state = gamestart ?
S_GAME_QUESTION3:S_GAME_QUESTION2_WAIT;


                                     S_GAME_QUESTION3: next_state = gamestart ?
S_GAME_QUESTION3 : S_GAME_QUESTION3_WAIT; // Loop in current state until value is input

                                     S_GAME_QUESTION3_WAIT: next_state = gamestart ?
S_GAME_QUESTION4 : S_GAME_QUESTION3_WAIT;
                                     S_GAME_QUESTION4: next_state = gamestart ?
S_GAME_QUESTION4 : S_GAME_QUESTION4_WAIT; // Loop in current state until go signal goes low

S_GAME_QUESTION4_WAIT:next_state=gamestart?S_GAME_QUESTION5:S_GAME_QUESTION4_WAIT;


                                     S_GAME_QUESTION5:next_state=gamestart?
S_GAME_QUESTION5:S_GAME_QUESTION5_WAIT;

                                     S_GAME_QUESTION5_WAIT: begin
                                     if(gamestart==0)
                                       next_state=S_GAME_QUESTION5_WAIT;
                                     else
                                       next_state=S_GAME_RANKING;
                                    end

                                     S_GAME_RANKING: begin
                                     if(displayrankings)
                                       next_state=S_GAME_RANKING;
                                           else
                                           next_state=S_GAME_IDLE;



                                     end
                                     default:     next_state = S_GAME_IDLE;
        endcase
    end // state_table


// Output logic aka all of our datapath control signals
    always @(*)
    begin: enable_signals
        // By default make all our signals 0 to avoid latches.
        // This is a different style from using a default statement.
        // It makes the code easier to read.  If you add other out
        // signals be sure to assign a default value for them here.
        ld_game = 1'b0;
                question_selection=3'b111;//noe of the questions are slected //game not
started yet
        endgame=1'b0;
//              ld_time_A=1'b1;//always load the shift registere??
//              ld_time_B=1'b1;
//              mux_selectA=1'b0;
//              mux_selectB=1'b0;
         ld_A_QS1=1'b0;
                    ld_A_QS2=1'b0;
                    ld_A_QS3=1'b0;
                    ld_A_QS4=1'b0;
                    ld_A_QS5=1'b0;
                    ld_B_QS1=1'b0;
                    ld_B_QS2=1'b0;
                    ld_B_QS3=1'b0;
                    ld_B_QS4=1'b0;
                    ld_B_QS5=1'b0;
                    ld_C_QS1=1'b0;
                    ld_C_QS2 =1'b0;
                    ld_C_QS3=1'b0;
                    ld_C_QS4 =1'b0;
                    ld_C_QS5=1'b0;
         ld_r=1'b0;
```

```
                    go=0;
                    rankdisplay=0;


               //set the vga oputputs here

        case (current_state)
            S_GAME_IDLE: begin
                                ld_game=1'b1;//signal to datapath for beginning of a game
                //DO NOTHING
                                    //CALL THE DISPLAY SIGNAL FOR BASIC FIRST DISPLAY WHEN
GAME OPENS
                                    go=0;




                                    //also reset the counter
//                              ld_time_A=1'b1;//load time comming from datapath
//                              ld_time_B=1'b1;
//                              mux_selectA=1'b0;//select the time from datapath
//                              mux_selectB=1'b0;
                                ld_r=1'b1;
                                cctenable=1'b0;
                                    ld_A_QS1=1'b0;
                    ld_A_QS2=1'b0;
                    ld_A_QS3=1'b0;
                    ld_A_QS4=1'b0;
                    ld_A_QS5=1'b0;
                    ld_B_QS1=1'b0;
                    ld_B_QS2=1'b0;
                    ld_B_QS3=1'b0;
                    ld_B_QS4=1'b0;
                    ld_B_QS5=1'b0;
                    ld_C_QS1=1'b0;
                    ld_C_QS2 =1'b0;
                    ld_C_QS3=1'b0;
                    ld_C_QS4 =1'b0;
                    ld_C_QS5=1'b0;

                end
            S_GAME_QUESTION1: begin
                //CALL THE VGA FOR FIRST QUESTION DISPLAY
                                    question_selection=3'b000;
                                    ld_game=1'b0;//dont reset the data path game happening

//                               ld_time_A=1'b1;//load time comming from datapath
//                              ld_time_B=1'b1;
//                              mux_selectA=1'b0;//select the time from datapath
//                              mux_selectB=1'b0;
                                ld_A_QS1=1'b1;
                                ld_B_QS1=1'b1;


                                ld_A_QS2=1'b0;
                            ld_A_QS3=1'b0;
                            ld_A_QS4=1'b0;
                    ld_A_QS5=1'b0;
                    ld_B_QS2=1'b0;
                    ld_B_QS3=1'b0;
                    ld_B_QS4=1'b0;
                    ld_B_QS5=1'b0;
                    ld_C_QS1=1'b1;
                    ld_C_QS2 =1'b0;
                    ld_C_QS3=1'b0;
                    ld_C_QS4 =1'b0;
                    ld_C_QS5=1'b0;

                           //ld_r=1'b1;
                           //signal to click tic to begin counting time in milliseconds
                           cctenable=1'b1; //qs 1 is displayed
                           go=1'b1;


                           end
                   S_GAME_QUESTION1_WAIT:begin
                                question_selection=3'b000;
                                    ld_game=1'b1;//reset the datapath for the next question
//
//                                 ld_time_A=1'b1;//load time comming from datapath
//                                ld_time_B=1'b1;
//                                mux_selectA=1'b1;//select the time from alu
//                              mux_selectB=1'b1;
//
                           //ld_r=1'b1;
                           //now stop timer for that question
                           cctenable=1'b0;
                           ld_A_QS1=1'b0;
                                ld_A_QS2=1'b0;

                            ld_A_QS3=1'b0;
                            ld_A_QS4=1'b0;
                    ld_A_QS5=1'b0;

                     ld_B_QS1=1'b0;
                    ld_B_QS2=1'b0;
                    ld_B_QS3=1'b0;
                    ld_B_QS4=1'b0;
```

```
                    ld_B_QS5=1'b0;
                    ld_C_QS1=1'b0;
                    ld_C_QS2 =1'b0;
                    ld_C_QS3=1'b0;
                    ld_C_QS4 =1'b0;
                    ld_C_QS5=1'b0;
                    go=1'b1;



                end
            S_GAME_QUESTION2: begin
                //CALL VGA FOR DISPLAY 1 QUESTION
                                    question_selection=3'b001;//just select a question so
correct ans is known
                ld_game=1'b0;//reset the datapath for next question
//                                  ld_time_A=1'b1;//load time comming from datapath
//                             ld_time_B=1'b1;
//                             mux_selectA=1'b0;//select the time from datapath
//                           mux_selectB=1'b0;
                              //ld_r=1'b1;

                              ld_A_QS1=1'b0;
                               ld_A_QS2=1'b1;

                         ld_A_QS3=1'b0;
                         ld_A_QS4=1'b0;
                    ld_A_QS5=1'b0;

                     ld_B_QS1=1'b0;
                    ld_B_QS2=1'b1;
                    ld_B_QS3=1'b0;
                    ld_B_QS4=1'b0;
                    ld_B_QS5=1'b0;
                    ld_C_QS1=1'b0;
                    ld_C_QS2 =1'b1;
                    ld_C_QS3=1'b0;
                    ld_C_QS4 =1'b0;
                    ld_C_QS5=1'b0;

                           go=1'b0;




                           cctenable=1'b1;//begin timer for qs 2
```

```
                                          end

                             S_GAME_QUESTION2_WAIT:begin
                                       question_selection=3'b001;
                                           ld_game=1'b1;//reset the datapath for the next question
//
//                                      ld_time_A=1'b1;//load time comming from datapath
//                                     ld_time_B=1'b1;
//                                     mux_selectA=1'b1;//select the time from alu
//                                   mux_selectB=1'b1;
                                      //ld_r=1'b1;
                             cctenable=1'b0;
                                     go=1'b0;




                                     ld_A_QS1=1'b0;
                                     ld_A_QS2=1'b0;

                                 ld_A_QS3=1'b0;
                                 ld_A_QS4=1'b0;
                            ld_A_QS5=1'b0;

                             ld_B_QS1=1'b0;
                            ld_B_QS2=1'b0;
                            ld_B_QS3=1'b0;
                            ld_B_QS4=1'b0;
                            ld_B_QS5=1'b0;
                            ld_C_QS1=1'b0;
                            ld_C_QS2 =1'b0;
                            ld_C_QS3=1'b0;
                            ld_C_QS4 =1'b0;
                            ld_C_QS5=1'b0;


                      end



                     S_GAME_QUESTION3: begin
                      //CALL VGA TO DISPLAY 2 QUESTION
                                          question_selection=3'b010;
                                          ld_game=1'b0;
//                                         ld_game=1'b0;//reset the datapath for next question
//                                        ld_time_A=1'b1;//load time comming from datapath
//                                     ld_time_B=1'b1;
//                                     mux_selectA=1'b0;//select the time from datapath
//                              mux_selectB=1'b0;

                              //ld_r=1'b1;
                              cctenable=1'b1;

                              ld_A_QS1=1'b0;
                                     ld_A_QS2=1'b0;

                                 ld_A_QS3=1'b1;
                                 ld_A_QS4=1'b0;
                            ld_A_QS5=1'b0;

                             ld_B_QS1=1'b0;
                            ld_B_QS2=1'b0;
                            ld_B_QS3=1'b1;
                            ld_B_QS4=1'b0;
                            ld_B_QS5=1'b0;
                            ld_C_QS1=1'b0;
                            ld_C_QS2 =1'b0;
                            ld_C_QS3=1'b1;
                            ld_C_QS4 =1'b0;
                            ld_C_QS5=1'b0;
                            go=1'b1;




                      end

                              S_GAME_QUESTION3_WAIT:begin
                                       question_selection=3'b010;
                                           ld_game=1'b1;//reset the datapath for the next question
//                   ld_time_A=1'b1;//load time comming from datapath
//                                ld_time_B=1'b1;
//                                mux_selectA=1'b1;//select the time from alu
//                              mux_selectB=1'b1;
//                      ld_r=1'b1;
                                     cctenable=1'b0;

                                     ld_A_QS1=1'b0;
                                     ld_A_QS2=1'b0;

                                 ld_A_QS3=1'b0;
                                 ld_A_QS4=1'b0;
                            ld_A_QS5=1'b0;
```

```
                                ld_B_QS1=1'b0;
                               ld_B_QS2=1'b0;
                               ld_B_QS3=1'b0;
                               ld_B_QS4=1'b0;
                               ld_B_QS5=1'b0;
                               ld_C_QS1=1'b0;
                               ld_C_QS2 =1'b0;
                               ld_C_QS3=1'b0;
                               ld_C_QS4 =1'b0;
                               ld_C_QS5=1'b0;
                               go=1'b1;








                         end



                                      S_GAME_QUESTION4: begin
                        //CALL VGA TO DISPLAY QS 3
                                              question_selection=3'b011;
                                              ld_game=1'b0;
//                                             ld_game=1'b0;//reset the datapath for next question
//                                             ld_time_A=1'b1;//load time comming from datapath
//                                        ld_time_B=1'b1;
//                                        mux_selectA=1'b0;//select the time from datapath
//                                      mux_selectB=1'b0;
//                                      ld_r=1'b1;
                                        cctenable=1'b1;

                                       ld_A_QS1=1'b0;
                                        ld_A_QS2=1'b0;

                                    ld_A_QS3=1'b0;
                                    ld_A_QS4=1'b1;
                               ld_A_QS5=1'b0;

                                ld_B_QS1=1'b0;
                               ld_B_QS2=1'b0;
                               ld_B_QS3=1'b0;
                               ld_B_QS4=1'b1;
                               ld_B_QS5=1'b0;
                        ld_C_QS1=1'b0;
                               ld_C_QS2 =1'b0;
```

```
                    ld_C_QS3=1'b0;
                    ld_C_QS4 =1'b1;
                    ld_C_QS5=1'b0;
                    go=1'b0;






          end

                           S_GAME_QUESTION4_WAIT:begin
                               question_selection=3'b011;
                                   ld_game=1'b1;//reset the datapath for the next question
//                 ld_time_A=1'b1;//load time comming from datapath
//                             ld_time_B=1'b1;
//                             mux_selectA=1'b1;//select the time from alu
//                           mux_selectB=1'b1;
                      //ld_r=1'b1;


                                  go=1'b0;




                                  ld_A_QS1=1'b0;
                             ld_A_QS2=1'b0;

                         ld_A_QS3=1'b0;
                         ld_A_QS4=1'b0;
                    ld_A_QS5=1'b0;

                     ld_B_QS1=1'b0;
                    ld_B_QS2=1'b0;
                    ld_B_QS3=1'b0;
                    ld_B_QS4=1'b0;
                    ld_B_QS5=1'b0;
                    ld_C_QS1=1'b0;
                    ld_C_QS2 =1'b0;
                    ld_C_QS3=1'b0;
                    ld_C_QS4 =1'b0;
                    ld_C_QS5=1'b0;

                                  cctenable=1'b0;
              end
```

```
                                S_GAME_QUESTION5: begin
                                //CALL VGA TO DISPLAY QS 4
                            question_selection=3'b100;
                                ld_game=1'b0;
                                 ld_game=1'b0;//reset the datapath for next question
//                                        ld_time_A=1'b1;//load time comming from datapath
//                                   ld_time_B=1'b1;
//                                   mux_selectA=1'b0;//select the time from datapath
//                                 mux_selectB=1'b0;
//                                 ld_r=1'b1;
                                cctenable=1'b1;
                                ld_A_QS1=1'b0;
                                  ld_A_QS2=1'b0;

                              ld_A_QS3=1'b0;
                              ld_A_QS4=1'b0;
                         ld_A_QS5=1'b1;

                          ld_B_QS1=1'b0;
                         ld_B_QS2=1'b0;
                         ld_B_QS3=1'b0;
                         ld_B_QS4=1'b0;
                         ld_B_QS5=1'b1;
                         ld_C_QS1=1'b0;
                         ld_C_QS2 =1'b0;
                         ld_C_QS3=1'b0;
                         ld_C_QS4 =1'b0;
                         ld_C_QS5=1'b1;

                                go=1'b1;





                                end



                                S_GAME_QUESTION5_WAIT:begin
                                    question_selection=3'b100;
                                        ld_game=1'b1;//reset the datapath for the next question
//                      ld_time_A=1'b1;//load time comming from datapath
//                                   ld_time_B=1'b1;
//                                   mux_selectA=1'b1;//select the time from alu
//                                 mux_selectB=1'b1;
//                                 ld_r=1'b1;
                 cctenable=1'b0;
```

```
go=1'b1;





ld_A_QS1=1'b0;
                ld_A_QS2=1'b0;

            ld_A_QS3=1'b0;
            ld_A_QS4=1'b0;
       ld_A_QS5=1'b0;

        ld_B_QS1=1'b0;
       ld_B_QS2=1'b0;
       ld_B_QS3=1'b0;
       ld_B_QS4=1'b0;
       ld_B_QS5=1'b0;
       ld_C_QS1=1'b0;
       ld_C_QS2 =1'b0;
       ld_C_QS3=1'b0;
       ld_C_QS4 =1'b0;
       ld_C_QS5=1'b0;


 end


              S_GAME_RANKING: begin
          //call to topmodule to stop comparing and display results on VGA
              endgame=1'b1;
              ld_r=1'b1;//only load results register at end of game
              cctenable=1'b0;


              ld_A_QS1=1'b0;
                ld_A_QS2=1'b0;

            ld_A_QS3=1'b0;
            ld_A_QS4=1'b0;
       ld_A_QS5=1'b0;

        ld_B_QS1=1'b0;
       ld_B_QS2=1'b0;
       ld_B_QS3=1'b0;
       ld_B_QS4=1'b0;
       ld_B_QS5=1'b0;
       ld_C_QS1=1'b0;
       ld_C_QS2 =1'b0;
```

```
                        ld_C_QS3=1'b0;
                        ld_C_QS4 =1'b0;
                        ld_C_QS5=1'b0;
                        rankdisplay=1'b1;



                               end





                             // default:    // don't need default since we already made sure
all of our outputs were assigned a value at the start of the always block
        endcase
    end // enable_signals

    // current_state registers
    always@(posedge clk)
    begin: state_FFs
        if(!reset)//active low
            current_state <= S_GAME_IDLE;
        else
            current_state <= next_state;
    end // state_FFS
endmodule
```

Datpath of the game logic :datapath.v

```
module
datapath(tic,Acorrect,Bcorrect,Ccorrect,clock,reset,firstplayer,secondplayer,thirdplayer,time
_PlayerA,
time_PlayerB,time_PlayerC,start,questionselected,playerabuzz,playerbbuzz,playercbuzz,game_ove
r);

//ADD THIRD AND FOURTH PLAYER LATER
//FIRSTPLAYER IS OPTIONS
//QUESTION SELECTED IS THE QUESTION PATTERN SELECTED
//START IS GAME START
//player1buzz is the key from top module
```

```
input [3:0]firstplayer;//options
input [3:0]secondplayer;//options
input [7:0]thirdplayer;
reg [3:0]coptions;
//converting from keyboard to proper input
always @(*)
 begin
 case (thirdplayer)
 8'h 1C: coptions=4'b0001;
 8'h 32: coptions=4'b0010;
 8'h 21: coptions=4'b0100;
 8'h 23: coptions=4'b1000;
default: coptions=4'b0000;
endcase
end
```

```
input game_over;//comming from controller to display rankings
input clock,start,reset;//start is comming from the controller to begin a new game


input tic;//comes every 1 millisecond from controller/top module

input[2:0] questionselected;//from control-determines which qs is selected

input playerabuzz,playerbbuzz;
input [7:0]playercbuzz;



output reg [25:0] time_PlayerA;//the time taken by each player
output reg [25:0] time_PlayerB;
output reg [25:0] time_PlayerC;
```

```
output reg  [2:0] Acorrect;//one bit wires to determine wether or not the player answered
correctly
output reg [2:0]Bcorrect;
output reg [2:0]Ccorrect;

reg [25:0]qa;//the time counting machinery
reg [25:0]qb;
reg [25:0] qc;


reg A_pressed;//signal the pressing of the buzzer-different from Abuzz and Bbuzz
reg B_pressed;
reg C_pressed;


//the correctoptions coding
reg [3:0]correctanswers;
always@(posedge clock)
begin
case(questionselected)//commimg from control deciding which answer is orrect for particular
qs
3'b000: correctanswers= 4'b0001;//answer to question 1 is option a(if question1)//one hot
encoding intuitive
 3'b001: correctanswers=4'b0010;//answer to question 2 is option b
 3'b010: correctanswers=4'b0100;//answer to question3 is option c
 3'b011: correctanswers=4'b1000;//answer to question4 is option d
 3'b100: correctanswers=4'b1000;//answer to question5 is also d

 default:correctanswers=4'b0000;//set default to 0


endcase
end




    // what is happening for evry question
    always@(posedge clock) begin
        if((!reset)  ) begin//this is manual reset after every question

                            Acorrect<=3'b0;
                            Bcorrect<=3'b0;
                            Ccorrect<=3'b0;

                            time_PlayerA<=25'b0;
                            time_PlayerB<=25'b0;
                            time_PlayerC<=25'b0;

                            A_pressed<=1'b0;
                    B_pressed<=1'b0;
                            C_pressed<=1'b0;
```

```
                              qa<=0;
                     qb<=0;
                              qc<=0;


        end

else begin//if not reset

                      if(start)//game has begun reset all output signals-also reset after
every question
                begin

                              Acorrect<=3'b0;
                              Bcorrect<=3'b0;
                              Ccorrect<=3'b0;


                              time_PlayerA<=25'b0;
                              time_PlayerB<=25'b0;
                              time_PlayerC<=25'b0;




                              A_pressed<=1'b0;
                     B_pressed<=1'b0;
                              C_pressed<=1'b0;

                              qa<=0;
                     qb<=0;
                              qc<=0;


                     end




                      else if(game_over)//controller signalled the end of game
                           begin
                                //the logic to compute winner goes here-done in top level
module
                      Acorrect<=3'b0;
                              Bcorrect<=3'b0;
                              Ccorrect<=3'b0;



                              time_PlayerA<=25'b0;
```

```
                    time_PlayerB<=25'b0;
                    time_PlayerC<=25'b0;




                    A_pressed<=1'b0;
            B_pressed<=1'b0;
                    C_pressed<=1'b0;

                    qa<=0;
            qb<=0;
                 qc<=0;







             end




             else//game  not over yet, not new round so keep comp time and rankings
               begin

               //the timer logic
               if(playerabuzz & playerbbuzz)//SEEMS LIKE nobody pressed
      begin
             if(A_pressed==0 &B_pressed==0 )//nobody pressed//
                    begin
                        //start counter for both players times
                        qa<=qa+1;
                        qb<=qb+1;
                        time_PlayerA<=qa;
                        time_PlayerB<=qb;
                        end
              else if(A_pressed==1&B_pressed==0 )//A has already  given input B hasnt
yet
                     begin
                              qb<=qb+1;//continue timer of B
                              time_PlayerB<=qb;
                              //do nothing with player a's time-taken care below
                              end
             else if(A_pressed==0 & B_pressed==1)//B gave input A didnt yet
             begin
                               qa<=qa+1;
                              time_PlayerA<=qa;
                              end
                     else if(B_pressed==1 & A_pressed==1)//both players have given inputs
already and key debounced to 1
                            begin
                                    qb<=0;//do nothing??
                                    qa<=0;//dont increment counter keep time as it is to be
stored in register
                                    end
               end
              else if(playerabuzz==0 & playerbbuzz==1)//THE VERY INSTANCE when  player A
toggles his switch and B doesnt-sig delay
                    begin
                            if(A_pressed==0 &B_pressed==0)//first time input-ACCEPT-A
pressing first B not pressed yet
                              begin
                                  A_pressed<=1'b1;//make this one so it never enters loop
again untill reset pressed
                                          qa<=0;//stop counter of A while result intact in
register time_PlayerA
                                          qb<=qb+1;//keep counting B
                                          time_PlayerB<=qb;//assign and store time of B
                                          end
                                    else if(A_pressed==0 & B_pressed==1)//B gave input, A is
giving later-now
                                         begin
                                                 A_pressed<=1'b1;//make this one so it never
enters loop again
                                         qa<=0;//stop timer for A leave time_playerA intact
                                                 //dont touch B
                                         end
                                   else if(A_pressed==1 & B_pressed==0)//A has already given
input trying again-dont accept!!-B not yet given
                                        begin
                                            qb<=qb+1;//increment time for qb
                                            time_PlayerB<=qb;//store value in reg
                                            //dont do anything with qa-already set to zeroor
set to zero??

                                            end
                                   else if (A_pressed==1 &B_pressed==1)//both player gave
inputs-A trying again
                                    begin
                                            //do nothing ?? or set to zero??
                                            end
              end
              else if(playerabuzz==1 & playerbbuzz==0)//player B had has pressed the buzzer
                     begin
                            if(A_pressed==0 &B_pressed==0)//first time input-ACCEPT-B A not
pressed yet
                              begin
```

```
                                   B_pressed<=1'b1;//make this one so it never enters loop
again untill reset pressed
                                           qb<=0;//stop counter of B while result intact in
register time_PlayerB
                                           qa<=qa+1;//keep counting A
                                           time_PlayerA<=qa;//assign and store time of A
                                           end
                                     else if(A_pressed==0 & B_pressed==1)//B gave input
trying again!!, A not yet given
                                          begin
                                                  qa<=qa+1;//continue timer of A
                                          time_PlayerA<=qa;//store value in register
                                                  //dont touch B
                                          end
                                    else if(A_pressed==1 & B_pressed==0)//A has already given
input B is giving later-Now
                                         begin
                                             B_pressed<=1'b1;//so it never enters loop again
                                             qb<=0;//reset counter of B
                                             //time stotrein in reg time_playerB
                                             //dont touch A
                                             end
                                    else if (A_pressed==1 &B_pressed==1)//both player gave
inputs-B trying again
                                     begin
                                             //do nothing ?? or set to zero??
                                             end

                      end


        else if (playerabuzz==0 & playerbbuzz==0)//both actually pressed together-is it even
possible??

                        //really difficult for two player to synchronise inputs with the same
positive clock edge

                        begin
                             if(A_pressed==0 &B_pressed==0)//first time input-accept for both
A and B
                               begin
                                   B_pressed<=1'b1;//make this one so it never enters loop
again untill reset pressed
                                           qb<=0;//stop counter of B while result intact in
register time_PlayerB
                                           A_pressed<=1'b1;//A has also pressed
                                           qa<=0;//set counter of A to zero
                                           end
                                     else if(A_pressed==0 & B_pressed==1)//B gave input
trying again!!, A not yet given
                                          begin
                                                  A_pressed<=1'b1;//so it never enters loop
again

                                                  qa<=0;//stop timer of A now

                                                  //dont touch B
                                          end
                                    else if(A_pressed==1 & B_pressed==0)//A has already given
input trying again!B giving Now!!
                                         begin
                                             B_pressed<=1'b1;//so it never enters loop again
                                             qb<=0;//reset counter of B
                                             //time stotrein in reg time_playerB
                                             //dont touch A
                                             end
                                    else if (A_pressed==1 &B_pressed==1)//both player gave
inputs-B again trying again at same time !!
                                     begin
                                             //do nothing ?? or set to zero??
                                             end

                                    end

                                    //just trying the player C logic separately
                                    if(playercbuzz==0)//if keyboard is  not pressed
                                      begin
                                       if(C_pressed==0)
                                              begin
                                                  qc<=qc+1;
                                                  time_PlayerC<=qc;
                                                  end
                                           else begin //if player already given  dont do
anything

                                           end
                                           end
                                    else if(playercbuzz>8'h1)
                                          begin
                                                  if(C_pressed==0)//giving input first time
                                                  begin
                                                  qc<=0;//reset the timer store result in reg
time_playerc

                                                  C_pressed<=1'b1;
                                                  end
                                                  else
                                                  begin
                                                  end
                                           end
```

```
              //now checked the timing for all players
      //output to top level-time if player is correct
      if(A_pressed &B_pressed)//both players have pressed their buzzers
             begin
                   if(firstplayer==correctanswers)//check correct answer of player
                          begin
                              Acorrect<=3'b001;//SEND THIS TO top module sayinmg that player
is correct for this answer

                              end
                            if(secondplayer==correctanswers)
                              begin
                              Bcorrect<=3'b001;

                          end

                     end
         if(C_pressed)
            begin
                   if(coptions==correctanswers)
                      begin
                     Ccorrect<=3'b001;
                  end


        end









       end

    end
```

```
end

endmodule
```

RAM of question 1 :q1Mem.v

```
// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: q1Mem.v
// Megafunction Name(s):
//                    altsyncram
//
// Simulation Library Files(s):
//                    altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 18.0.0 Build 614 04/24/2018 SJ Standard Edition
// ************************************************************


//Copyright (C) 2018  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module q1Mem (
       address,
       clock,
       data,
       wren,
       q);
       input  [16:0]  address;
       input    clock;
       input  [11:0]  data;
       input    wren;
       output [7:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
       tri1     clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

       wire [11:0] sub_wire0;
       wire [7:0] q = sub_wire0[7:0];

       altsyncram    altsyncram_component (
                      .address_a (address),
                      .clock0 (clock),
                      .data_a (data),
                      .wren_a (wren),
                      .q_a (sub_wire0),
                      .aclr0 (1'b0),
                      .aclr1 (1'b0),
                      .address_b (1'b1),
                      .addressstall_a (1'b0),
                      .addressstall_b (1'b0),
                      .byteena_a (1'b1),
                      .byteena_b (1'b1),
                      .clock1 (1'b1),
                      .clocken0 (1'b1),
                      .clocken1 (1'b1),
                      .clocken2 (1'b1),
                      .clocken3 (1'b1),
                      .data_b (1'b1),
                      .eccstatus (),
                      .q_b (),
                      .rden_a (1'b1),
                      .rden_b (1'b1),
                      .wren_b (1'b0));
       defparam
              altsyncram_component.clock_enable_input_a = "BYPASS",
              altsyncram_component.clock_enable_output_a = "BYPASS",
              altsyncram_component.init_file = "q1.mif",
              altsyncram_component.intended_device_family = "Cyclone V",
              altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
              altsyncram_component.lpm_type = "altsyncram",
              altsyncram_component.numwords_a = 76800,
		altsyncram_component.operation_mode = "SINGLE_PORT",
		altsyncram_component.outdata_aclr_a = "NONE",
		altsyncram_component.outdata_reg_a = "CLOCK0",
		altsyncram_component.power_up_uninitialized = "FALSE",
		altsyncram_component.read_during_write_mode_port_a = "NEW_DATA_NO_NBE_READ",
		altsyncram_component.widthad_a = 17,
		altsyncram_component.width_a = 12,
		altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "q1.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "76800"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "17"
// Retrieval info: PRIVATE: WidthData NUMERIC "12"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "q1.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "76800"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING "NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "17"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "12"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 17 0 INPUT NODEFVAL "address[16..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 12 0 INPUT NODEFVAL "data[11..0]"
// Retrieval info: USED_PORT: q 0 0 12 0 OUTPUT NODEFVAL "q[11..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 17 0 address 0 0 17 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 12 0 data 0 0 12 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 12 0 @q_a 0 0 12 0
// Retrieval info: GEN_FILE: TYPE_NORMAL q1Mem.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL q1Mem.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL q1Mem.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL q1Mem.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL q1Mem_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL q1Mem_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```

References

PS2 Keyboard:

Intel.com. (2019). *University - University IP Cores*. [online] Available at: https://www.intel.com/content/www/us/en/programmable/support/training/university/materials-ip-cores.html?&ifup_version=9.1 [Accessed 2 Dec. 2019].