6.111 Final Report

Shuto Ogihara, Emmanuel Havugimana

December 2019

Contents

1 Abstract

Our project is about making a robot car that could track an object using a loaded camera, and move itself to follow the object. To make it simple, the object we used as were uni-color object/spheres, and color detection technique was used to track the object in the scene. Through VGA cables, the scene from the camera and visual feedback of the car motors were displayed on the screen to configure the parameters used to run the robot. Chase-bot will move to keep the object appear in the same location and the same size to keep the distance based on a control algorithm. By loading the FPGA, camera, batteries and all the other things necessary on the car, the robot is able to update the speeds of the motors at the frame rate of the camera which enables itself to track the object with minimum delay. Besides of the chasing task, we implemented another task using the same algorithm where the camera was attached to the side of the car and it would move only backward and forward to collide with the object, which we will refer to as the goal keeping task.

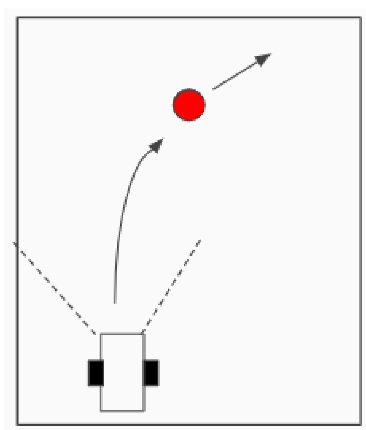

Figure 1: Chasing Task

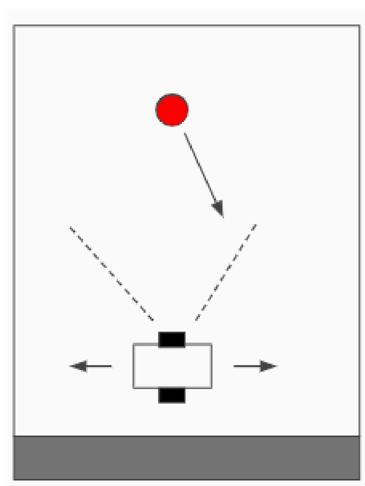

Figure 2: Goal Keeping Task

2 Project Goals

2.1 Baseline Goals

- Initialize the object to follow by a user interface with the camera input and a cursor on the screen
- Track the center and size of the object of interest using color detection

- Display a visual feedback of the tracking system showing which direction the car should move

2.2 Expected Goals

- Make the car chase the object by loading the camera in the front (Chasing Task)
- Make the car collide with the object by loading the camera on the side (Goal keeping)

2.3 Stretched Goals

- Make the car follow objects other than spheres
- Track itself using speed encoder
- Calculate real distance of the car and the object

3 System Design

3.1 High Level Block Diagram

The high level diagram of our system is shown in Figure 3. We have 3 major modules, tracker, controller, and initializer. The tracker module computes the radius and position of the tracking object, while the controller module generates adequate output to move the motor and drive the car. Initializer module deals with configuration of the object through visual interface and also takes care of the main FSM of this system since most of the states are related to initializing the object. The bram was used to store each frame when we display the camera input on the screen. It would lose 4 bit for each rgb due to capacity limitation. Although we needed to tune the parameters for tracking and controlling when debugging, we didn't have enough switches for that so we implemented a switch FSM which outputs the parameters based on the current switch values and the main state.

3.2 Main FSM

The main FSM of this system has 5 states. It starts with the initialize state where you choose which object to track by selecting a pixel (Figure 4) by moving the cursor using the buttons, and push the center button when necessary. After selection, it becomes the selected state, where the tracker would give the computed position and size on the screen as a white box. Then you can select it again to confirm you are satisfied with the tracking to move on the confirmed state. If you are not satisfied with the tracking, you could go back to initialize state by moving the cursor. The system would set the desired size of the object as the size of the object when you switch to confirmed state. The desired size

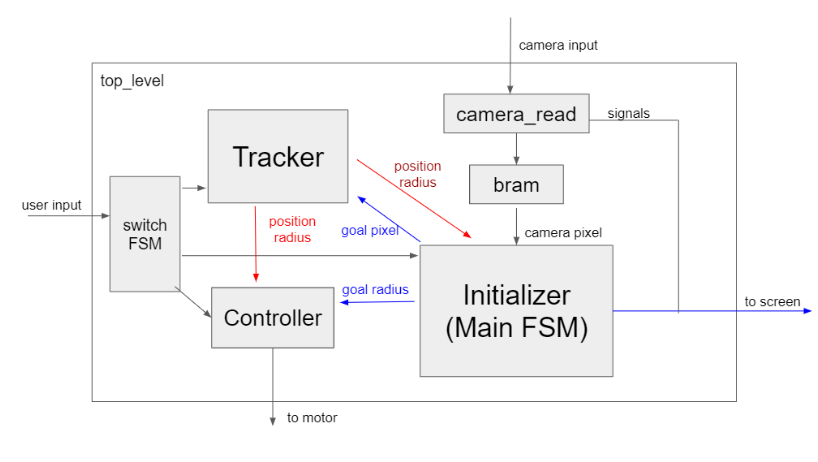

Figure 3: High level block diagram

corresponds to the distance between the car, so this transition would define how close the car would be to the object. In the confirmed state, the box would turn red, and there would be a visual feed back of the 2 motor outputs as 2 red bars on the right hand side of the screen.(Figure 5) The controller would compute the output for the motors, but it wouldn't actually output the signals since this state is for tuning the gains or setting the modes for control. Finally, when you are ready to move the car, you can deactivate the initializer to actually run! This is the move state, and you can always switch to pause mode where you can pause the car by pushing the center button.

Figure 4: initialize state

Figure 5: confirmed state

3.3 Hardware

2WD chassis

As the hardware of the robot, we used a 2 wheel driving chassis kit, which came with the body, 2 motors and wheels, battery box for the motor driver, and a passive wheel. Assembling the chassis was very straightforward. It was very intuitive with how to assemble the DC motors, wheels, and the battery box and solder the appropriate connections. Once the base of the chassis was assembled, the platform was installed by adding the motor driver at the bottom of the car (Figure 10) with wires connected to the battery box and the FPGA board loaded on the top side of the car. One issue that we had to confront was how to load all of the devices necessary on the limited space. This project required the FPGA, motor driver, battery pack for both motor driver and the camera all to be on the car. However, this platform was designed mainly in the use of Arduino where the board is quite smaller than Nexys 4 ddr. Therefore, we designed a second floor on the car using the acrylic plate and some spacers. This is quite similar to the FPGA stand in the 6.111 lab. As shown in Figure 7 the first floor had the battery pack for powering the FPGA and the FPGA was placed on the second floor. This provided enough area to fix the FPGA to the car in a stable manner and freedom to where the camera would be installed.

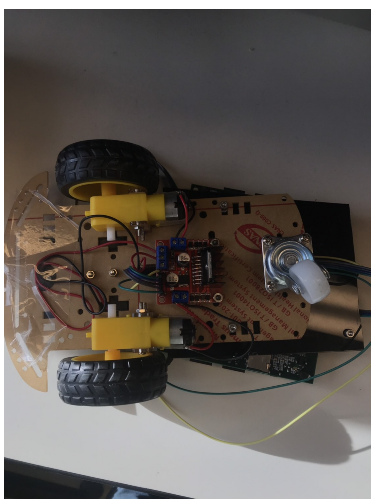

Figure 6: back view of the robot

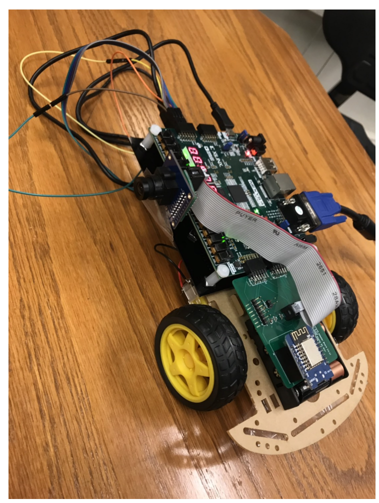

Figure 7: overview of the robot

L298N motor driver

In order to drive the motor appropriately, there were 2 problems; first one is that the output voltage from the FPGA would be 3.3V, which would be not enough for moving the car and the second one is that we cannot control the speed because we cannot output analog voltage. The L298 is a high voltage, high current dual full-bridge driver designed to accept not only DC motors but standard TTL logic levels and drive inductive loads such as relays, solenoids, and stepping motors. Two enable inputs were provided to enable or disable the motors independently of the input signals. L298N motor driver was used to amplify the voltage driven into the motors, and make it controllable using pulse width modulation (pwm) as the enable inputs. As shown in Figure 8, the width of the pulse, in other words the duty cycle, of the pwm would change the average voltage driven into each motors. The speed of the dc motors are proportional to the fed in voltage. Under the condition that the supply volatage was 6[V], the voltage would be described as

$$(DutyCycle) \times 6[V]$$

The circuit configuration of L298N is shown in Figure 9. Besides the enable input, there are 2 inputs, IN1 and IN2, to control the direction of each motor. Having IN1 high and IN1 low would drive the motor forward and vice versa.

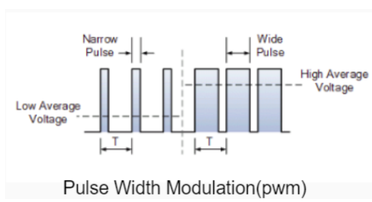

Figure 8: pulse width modulation

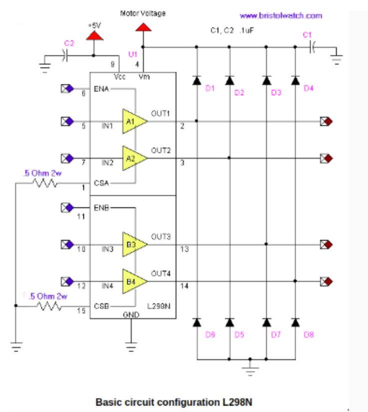

Figure 9: L298N circuit

OV7670 Camera

The camera we used was provided by the lab with arduino board for processing and some verilog codes useful for interfacing the camera. It had around 30 frame per seconds, each frame with 320*240 pixels. The ja and jb port of Nexys ddr 4 was occupied for camera interaction.

4 Modules

4.1 Tracker module [Emmanuel]

Tracker modules taken in a camera and color and outputs radius and position. A camera captures an image of the scene and that image is passed through a color space conversion module which gives out an image in an easy to color detect format. That image is then passed through a color detector[hereby named thresholder] which output the color when color in image match color of a selected pixel in initialization stage. After getting pixels that match the color, the xcenter and ycenter are computed by weighted sum and radius derived using number of pixels as area and then using square root and divider IP modules

$$x_{center} = \frac{\sum_0^n x_i}{n}$$

$$y_{center} = \frac{\sum_0^n y_i}{n}$$

$$radius = \sqrt{\frac{8 \times n}{22}}$$

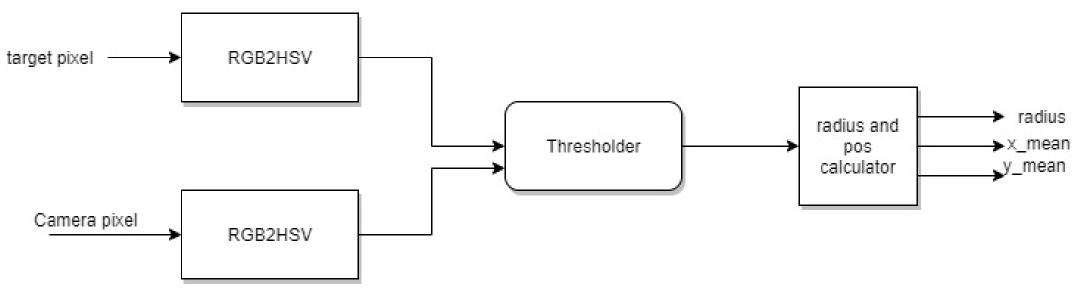

Figure 10: Block diagram of tracker module

for n pixels and x_i and y_i being the pixel locations of the detected object

RGB2HSV module

Since we are doing color detection and color detetion in RGB is not optimal. we need a module to change RGB pixel to HSV and then color threshold in HSV. HSV module had a delay of about 22 clock cycles that we had to correct for in computing the radius, y and x positions of an object.

color_class module

to detect colors first attempt was to have a margin over a color of a pixel selected in initialization stage. But we realized that a one margin for all colors was not great as color bands are not equal and sample of the color may be selected near another color. so we need a color class module to classify a pixel and use that class in detection.

4.2 Initializer module [Shuto]

The initializer module took in buttons for cursor control and selection, position and radius of the object for generating the box, speed of two motors for displaying visual feedback of the motor, and other signals necessary for display. The outputs are the output pixel,goal pixel for tracker, goal radius for control, and the state. The module could be broken down to several subsystems.

State Transition

The state transition of our FSM was quite simple. The buttons for controlling the cursors and the activation switch were the only ones to trigger transition. Our car would only move at the move state, and other states are only for configuration or pausing.

debouncer

Since some state transition rely on user input buttons, noise is likely to ruin the FSM. The debouncer make sure the button is high for at least 10 ms to actually say that the button is pressed.

box

Using information of the position and radius of the object from the tracker module, generates the box to show what it tracks. The color is white during initialize and selected state, and red during the other states.

cursor

Generates the cursor for the user to select the object. The center position would move accordingly to the button inputs, at the speed of 2 pixels per frame. The cursor would only be displayed at initialize and selected state.

colorpad

Generates a 30 * 30 blob in the color of the pixel of interest at the right bottom side of the screen. At the initialize state, it will display the color of the pixel at the center of the pixel, while in the other states, it will be the selected pixel instead. This allows access to which color that you are trying to track, which made debugging easier.

speed bar

Displays a visual feedback for the motor output on the right hand side of the screen. Consists of a horizontal base line and two red bars indicating the speed of each motor. Based on the 8 bit speed and forward/backward signals for each motors, the bar would be longer as the speed is higher and would face upwards if it is moving forward.

4.3 controller module [Shuto]

The controller module takes in the x,y position and the size of the object from the tracker module, and goal size of the object from the initializer module to generate signals to drive the motors properly based on control algorithm. This module has 4 subsystems.

control

This modules uses a PD control algorithm to generate the speed of the car and how much it turns as a 9 bit signed value, based on the error in both x position and radius. The desired x was set as the center of the camera, while the desired radius was an input from the initializer. It also takes in 12 bits parameter for the gains for the control, and 2 bits parameter indicating which mode to be in. There are 3 modes, debugging, chasing, and goal keeping. In the debugging mode, the 12 bits parameter would be the direct output of this module. This mode was helpful to check if the hardware was really working or not. In the chasing mode, the 12 bit parameter was for the proportional and differential gain for both turning and speed, 3 bits each. The speed was obtained by the PD control for the radius of the object, and turning was obtained by the PD control for the x position of the object. Note that the gains take care of the converting pixel space x and radius into the proportional space speed and turn. Since the sample rate is constant, we just used the difference between the recent two frames for the differential. The control equations for the chasing task would be as below,

$$speed = K_{sp}\tilde{r} + K_{sd}\delta\tilde{r}$$

$$turn = K_{tp}\tilde{x} + K_{td}\delta\tilde{x}$$

where

$$\tilde{r} = r_{desired} - r$$

$$\tilde{x} = x_{desired} - x$$

$$\delta\tilde{r} = \tilde{r_n} - \tilde{r_{n-1}}$$

$$\delta\tilde{x} = \tilde{x_n} - \tilde{x_{n-1}}$$

In the goal keeping mode, the first 6 bits of the parameters was the proportional and differential gain for the x position. The radius information was not used since in this task the movement won't be affected by how close the object is. The control equations for this task would be as follows.

$$speed = K_{sp}\tilde{x} + K_{sd}\delta\tilde{x}$$

$$turn = 0$$

motor out

This module takes in the speed and the turning of the car and breaks that down into the speed and direction of two motors. We used a simple model for this, where the average speed of the 2 motors would be the overall speed of the car, and the turning would be the difference in the speed of 2 motors (Figure 11). In the real hardware, because of the weight of all the devices, the car would not move until it had enough voltage to beat the friction. In our case, the voltage seemed to be around 1.2V, so we added an offset of 1V equivalent value to the speeds we obtained with the model above. This turned out to be helpful to make the car become more sensitive to the object position.

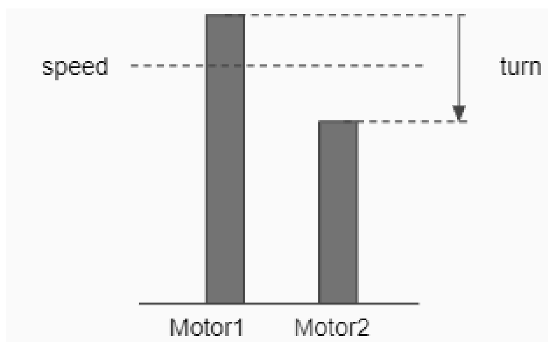

Figure 11: Conversion from car speed,turning to motor speed

After acquiring the speed and direction for both of the motors, this module generated the proper IN1,IN2, and enable signal for each using pwmgen below for the enable.

pwmgen

This module was used to generate the pwm based on the 8 bit input. We were able to generate 255 types of pwm signals in other words 255 types of speed for the motors.

xr_process

The estimated radius and x from the tracker module was noisy in that if the object was out of screen it reacted to the few pixel in the background and generated a bad radius and position. This caused the car to move randomly when it lost the object. Another problem was that since the camera angle was smaller than we expected, it was more likely to get out from the screen. This module took care of this 2 problems by pre-processing the radius and x from the tracker before it gets fed into the control. When the radius was too small, we estimated that the object was out of sight. We also made the car to move just the way it was for 10-15 frames after it lost the object and stop. This made the chance of getting the object in the scene again higher after it lost it once.

5 Challenges

For the color detection, the resolution of the camera was 4bit for the processed signals. While normal cameras have 8 bit each color channel R,G,B our camera had 4—5. With less resolution it was hard to tell colors which are very similar. In terms of the design of the hardware, since the car was much more smaller than I had expected, figuring out a way to load everything was a challenge. By making it a two floor car, we were able to make it not only packed but also robust. One regret is that we couldn't think of a way to fix the camera for each task. Another challenge was that the car did not act the way I thought. One example was that the with all the devices on board the car had required some torque to start making it move. We dealt with that by adding offset voltage to the motor out module. The chasing task turned out to perform well, but we had to figure out how to overcome the weakness of having a limited camera angle. Holding the output for few frames after the car loses the object definitely helped, but it wasn't enough to make it follow an object crossing right in front of the car. We did not have time to explore better solutions for that. The hardest thing about the goal keeping task was that the car does not move straight even though the 2 motor voltages are the same. Having an encoder for feedback or manually tuning the turning effect might have helped, but we did not have the time.

6 Future Work

As mentioned in the challenges, using the encoder would broaden the options of what we can do next. For example, having a feedback system or self tracking would be possible. For a simple extension of the project, a better camera with large angle of view would definitely help make a better tracking system and a control system . Future teams can also include robust noise reduction and make a more a better object detector. For example our object detection relied solely on color of the object, but other teams can explore shape recognition by correlation in 2D and use of a cascade of morphological processes, and hopefully track non-generic object like human faces.

7 Conclusion & Advices

We were able to track objects with different colors, and made our robot do two tasks which were goalkeeping and following. Most our modules were independently testable which made integration easier at the end.

We initially spent a lot of time on rgb2hsv module confused about divider module. Asking help earlier probably would have saved sometime. For future students, we recommend to integrate and start testing early. The first few iteration would never be perfect, so try testing and experimenting as early as you can. We believe that we were successful in that we set our goals doable, and had some time to improve the performance, but we still could have tested more and done more stretch goals.

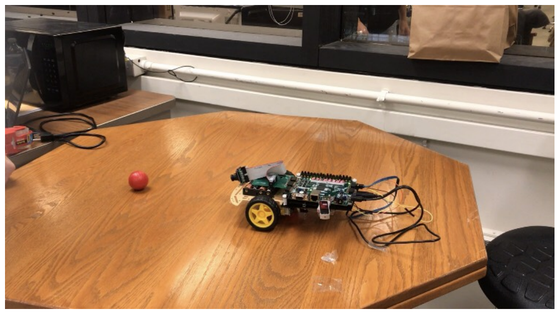

Figure 12: Chase-bot chasing a ball

A Appendix: Verilog Code

The code for the project can be found at `https://github.com/ehavugi/chasebot/tree/sources` sources branch have source files used.(we had issue with ip cores due to different vivado version). test1(a vivado project)

The list of modules

- track_init.sv
- tracker.sv
- rgb2hsv.sv
- blob.sv
- camera_read.sv
- control.sv
- xvga.sv
- clk_wiz_lab3.sv
- display_8hex.sv
- initialize.sv
- control.sv
- motor_out.sv
- color_class.sv

track_init.sv

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/26/2019 05:42:29 PM
// Design Name:
// Module Name: track_init
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File ‘timescale 1ns / 1ps
// ////////////////////////////////////////////////////////////////////////////////
//
// Updated 8/10/2019 Lab 3
// Updated 8/12/2018 V2.lab5c
// Create Date: 10/1/2015 V1.0
// Design Name:
// Module Name: labkit
//
// ////////////////////////////////////////////////////////////////////////////////

module track_init_control(
   input clk_100mhz,
   input[15:0] sw,
   input btnc, btnu, btnl, btnr, btnd,
   input  [7:0] ja,
   input  [2:0] jb,
   output   jbclk,
   input  [2:0] jd,
   output   jdclk,
   output[3:0] vga_r,
   output[3:0] vga_b,
   output[3:0] vga_g,
   output vga_hs,
   output vga_vs,
   output led16_b, led16_g, led16_r,
   output led17_b, led17_g, led17_r,
   output[15:0] led,
   output ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
   output[7:0] an,    // Display location 0-7
   output[7:0] jc   //for motor output
   );
    logic clk_65mhz;
    // create 65mhz system clock, happens to match 1024 x 768 XVGA timing
    clk_wiz_lab3 clkdivider(.clk_in1(clk_100mhz), .clk_out1(clk_65mhz));

    logic [31:0] data;      //  instantiate 7-segment display; display (8) 4-bit hex
    wire [6:0] segments;
    assign {cg, cf, ce, cd, cc, cb, ca} = segments[6:0];
    display_8hex display(.clk_in(clk_65mhz),.data_in(data), .seg_out(segments), .strobe_out(an));
    assign  dp = 1'b1;  // turn off the period
```

```
    assign led = sw;

    wire [10:0] hcount;    // pixel on current line
    wire [9:0] vcount;     // line number
    wire hsync, vsync, blank;
    wire [11:0] pixel;
    reg [11:0] rgb;
    xvga xvga1(.vclock_in(clk_65mhz),.hcount_out(hcount),.
        vcount_out(vcount),
          .hsync_out(hsync),.vsync_out(vsync),.blank_out(
              blank));


    // sw[0] button is user reset
    wire reset;
    debounce db1(.reset_in(reset),.clock_in(clk_65mhz),.
        noisy_in(sw[0]),.clean_out(reset));
    logic scale; //1 when twice scaling

    logic xclk;
    logic[1:0] xclk_count;

    logic pclk_buff, pclk_in;
    logic vsync_buff, vsync_in;
    logic href_buff, href_in;
    logic[7:0] pixel_buff, pixel_in;

    logic [11:0] cam;
    logic [11:0] frame_buff_out;
    logic [15:0] output_pixels;     //pixel from camera
    logic [12:0] processed_pixels;  //stored inside bram
    logic valid_pixel;  //1 if inside camera frame
    logic frame_done_out;   //pulse indicating the end of
        frame

    logic [16:0] pixel_addr_in;
    logic [16:0] pixel_addr_out;

    assign xclk = (xclk_count >2'b01);
    assign jbclk = xclk;
    assign jdclk = xclk;


    blk_mem_gen_0 jojos_bram(.addra(pixel_addr_in),
                             .clka(pclk_in),
                             .dina(processed_pixels),
                             .wea(valid_pixel),
                             .addrb(pixel_addr_out),
                             .clkb(clk_65mhz),
```

```
                                .doutb(frame_buff_out));

    always_ff @(posedge pclk_in)begin
        if (frame_done_out)begin
            pixel_addr_in <= 17'b0;
        end else if (valid_pixel)begin
            pixel_addr_in <= pixel_addr_in +1;
        end
    end

    always_ff @(posedge clk_65mhz) begin
        pclk_buff <= jb[0];//WAS JB
        vsync_buff <= jb[1]; //WAS JB
        href_buff <= jb[2]; //WAS JB
        pixel_buff <= ja;
        pclk_in <= pclk_buff;
        vsync_in <= vsync_buff;
        href_in <= href_buff;
        pixel_in <= pixel_buff;
        xclk_count <= xclk_count + 2'b01;
        processed_pixels = {output_pixels[15:12],
            output_pixels[10:7],output_pixels[4:1]};

    end

    assign pixel_addr_out = scale?((hcount>>1)+(vcount>>1)
        *32'd320):hcount+vcount*32'd320;
    assign cam = scale&&((hcount<640) &&  (vcount<480))?
        frame_buff_out:~scale&&((hcount<320) &&  (vcount<240)
        )?frame_buff_out:12'h000;
    assign {red,green,blue}=cam;


   camera_read  my_camera(.p_clock_in(pclk_in),
                          .vsync_in(vsync_in),
                          .href_in(href_in),
                          .p_data_in(pixel_in),
                          .pixel_data_out(output_pixels),
                          .pixel_valid_out(valid_pixel),
                          .frame_done_out(frame_done_out));


    //////////////////////////////for center positions and
        hsv////////////////////////
    logic [23:0] radius;
    logic [31:0] x_center,y_center;
    logic [11:0] thres;
    logic [11:0] pixel_out,goal_pixel,goal_rad;
    logic [7:0] h_t,s_t,v_t;
    logic show_thres,use_rgb;
```

```
    rgb2hsv  goal_px(.clock(clk_65mhz),.reset(reset),.r({
        goal_pixel[11:8],4'h0}),.g({goal_pixel[7:4],4'h0}), .
        b({goal_pixel[3:0],4'h0}), .h(h_t), .s(s_t), .v(v_t))
        ;

    tracker my_tracker(
    .clk(clk_65mhz),
    .use_rgb(use_rgb),
    .cam(cam),
    .hcount(hcount),
    .vcount(vcount),
    .goalpixel(goal_pixel),
    .vsync(vsync),
    .radius(radius),
    .x_center(x_center),
    .y_center(y_center),
    .thres(thres)
    );



    ////////////INITIALIZE////////////////
    logic signed [8:0] speed,turn;
    logic en1,ina1,inb1,ina2,inb2,en2;
    logic [7:0] speed_1,speed_2;
    logic signed [8:0] speed1,speed2;
    assign speed1 = {inb1,inb1?~speed_1 + 9'b1:speed_1};
    assign speed2 = {inb2,inb2?~speed_2 + 9'b1:speed_2};


    logic track,move;
    logic [3:0] direction;
    logic [2:0] state;
    assign direction = {btnu,btnd,btnl,btnr};

    initialize initializer(
          .clk_65mhz(clk_65mhz),
          .reset(reset),
          .hcount(hcount),
          .vcount(vcount),
          //.vsync(vsync_in),
          .vsync(frame_done_out),
          .directions(direction), //up,down,left,right
          .confirm_in(btnc),
          .activate_in(sw[1]),
          .sw2(scale), //whether to make the size twice
          .cam(show_thres?thres:cam),
          .cur_pos_x(x_center[8:0]),
          .cur_pos_y(y_center[8:0]),
```

```
            .cur_rad(radius),
            .speed1(speed1),
            .speed2(speed2),
            .pixel_out(pixel_out),
            .goal_pixel(goal_pixel),
            .goal_rad(goal_rad),
            .track(track),
            .move(move)
            //for debug
            ,
            .state(state)
//            .cursor_x(cursor_x),
//            .cursor_y(cursor_y),
//            .up(dir[3]),
//            .down(dir[1]),
//            .left(dir[2]),
//            .right(dir[0])
            );
    ////////////////////////////////////////////



    ////////////CONTROL/////////////////
    logic [3:0] Kps,Kpt;
    logic [2:0] Kds,Kdt;
    logic [1:0] mode;
    logic [15:0] params;
    logic [8:0] x,y;
    logic [6:0] rad;
    assign params = mode[1]?{Kps,Kds,Kpt,Kdt,mode}:{
        speed1_in,speed2_in,mode};

    assign x = &x_center?9'd160:x_center[8:0];
    assign y = y_center[8:0];
    assign rad = (radius<7'd10)?goal_rad:radius;

    control my_control( .clk_in(clk_65mhz),
                        .rst_in(reset),
                        .ready_in(frame_done_out),
                        .cur_pos_x(x),
                        .cur_pos_y(y),
                        .cur_rad(rad),
                        .goal_rad(goal_rad),
                        .params({Kps,Kds,Kpt,Kdt,mode}),
                        .speed(speed),
                        .turn(turn)
                        );

    motor_out my_motor( .clk_in(clk_65mhz),
                        .rst_in(reset),
```

```
                            .offset(8'd50),
                            .speed_in(speed),
                            .turn_in(turn),
                            .motor_out({en1,ina1,inb1,ina2,inb2,
                                en2}),
                            .speed_1(speed_1),
                            .speed_2(speed_2)
                            );

    assign jc[5:0] = move?{en1,en2,ina2,inb2,ina1,inb1}:6'b0
        ;
    //
        ///////////////////////////////////////////////////////////////


    ///////////////switch,segment display FSM
        /////////////////////////////////////
    parameter INITIALIZE = 0;
    parameter SELECTED = 1;
    parameter CONFIRMED = 2;
    parameter MOVE = 3;
    parameter PAUSE = 4;

    logic [1:0] seg_display;
    logic [7:0] speed1_in;  //signed
    logic [7:0] speed2_in;

    assign scale = 0;   //no scaling
    //assign mode[1] = 1;
    assign Kps[3] = 0;
    assign Kpt[3] = 0;
    assign speed1_in[1:0] = 0;
    assign speed2_in[1:0] = 0;

    // things to display on a 7 segment displays
    always_ff @(posedge clk_65mhz) begin
        case (seg_display)
            2'b00: data <= {7'b0,speed1,7'b0,speed2};   //
                xxx(center)xxxx(size)x(switch)
            2'b01: data <= {7'b0,x,3'b0,y, {1'b0,state}};
                // display 0123456 + sw[3:0]
            2'b10: data <= {{2'b0,goal_rad},{5'b0,state}};
            2'b11: data <= {rad,{1'b0,state}};
            default: data <= {x_center[15:0],y_center[11:0],
                 {1'b0,state}};
        endcase;
     end


    //switch controls
```

```
    //sw[0] is always reset
    //sw[1] is always used for state transition

    always @(posedge clk_65mhz) begin
        case(state)
            INITIALIZE: begin
                seg_display <= sw[14:13];
                use_rgb <= sw[12];
                show_thres <= sw[15];
                end
            SELECTED: begin
                seg_display <= sw[14:13];
                use_rgb <= sw[12];
                show_thres <= sw[15];
                //calibration related stuff too
                end
            CONFIRMED: begin
                {Kps[2:0],Kds,Kpt[2:0],Kdt,mode} <= sw[15:2];
                speed1_in[7:2] <= sw[15:10];
                speed2_in[7:2] <= sw[9:4];
                end
            PAUSE: begin
                {Kps[2:0],Kds,Kpt[2:0],Kdt,mode} <= sw[15:2];
                speed1_in[7:2] <= sw[15:10];
                speed2_in[7:2] <= sw[9:4];
                end
            default: begin
                {Kps[2:0],Kds,Kpt[2:0],Kdt,mode} <= sw[15:2];
                speed1_in[7:2] <= sw[15:10];
                speed2_in[7:2] <= sw[9:4];
                end
        endcase
    end






    //what to display
    reg b,hs,vs;

    always_ff @(posedge clk_65mhz) begin
        hs <= hsync;
        vs <= vsync;
        b <= blank;
        rgb <= pixel_out;
```

```
    end

    // the following lines are required for the Nexys4 VGA circuit - do not change
    assign vga_r = ~b ? rgb[11:8]: 0;
    assign vga_g = ~b ? rgb[7:4] : 0;
    assign vga_b = ~b ? rgb[3:0] : 0;

    assign vga_hs = ~hs;
    assign vga_vs = ~vs;

endmodule
```

tracker.sv

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:  MIT 6.111
// Engineer: Emmanuel HAVUGIMANA
//
// Create Date: 01.12.2019 19:50:24
// Design Name:
// Module Name: tracker
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module tracker(
    input clk,
    input use_rgb,
    input [11:0] cam,
    input [10:0] hcount,
    input [9:0] vcount,
    input [11:0]  goalpixel,
    input vsync,
    output logic [23:0] radius,
```

```
    output logic [31:0] x_center,
    output logic [31:0] y_center,
    output logic [11:0] thres
    );

logic[7:0] h_t,s_t,v_t;

logic [31:0] x_remainder;
logic [31:0] y_remainder;
logic [31:0] x_remainder1;
logic [31:0] y_remainder;
logic clk_65mhz;
logic [31:0] pos_y_d;
logic [31:0] pos_x_d;
logic [31:0] size;
logic [31:0] size_d;
logic [31:0] size_;
logic [31:0] size__;

logic [31:0] pos_y;
logic [31:0] pos_y_;
logic [31:0] pos_x;
logic [31:0] pos_x_;
logic [31:0] pos_x_d;
logic [7:0] h,s,v;
logic [3:0] red,green, blue;
logic [23:0] radius_;
logic [31:0] pos_y_;
logic [26:0] count_f=0;
logic [31:0] square_r;
logic ready4;
logic ready2;
logic ready3;
logic ready1;
logic ready;
logic threshold;

assign size__=size_*3'd7;
assign clk_65mhz=clk;

//changed any pixel from rgb to hsv
rgb2hsv  x(.clock(clk_65mhz),.reset(reset),.r({red,4'h0}), .
    g({green,4'h0}), .b({blue,4'h0}), .h(h), .s(s), .v(v));

// process goal pixel from rgb to hsv
rgb2hsv  goal_px(.clock(clk_65mhz),.reset(reset),.r({
    goalpixel[11:8],4'h0}),.g({goalpixel[7:4],4'h0}), .b({
    goalpixel[3:0],4'h0}), .h(h_t), .s(s_t), .v(v_t));

assign {red,green,blue}=cam; // get camera components
```

```
//always @(posedge clk) begin
//    if (count_f<6500000) begin
//        count_f<=count_f+1;
//    end
//    else begin
//        count_f<=0;
//        size_<=size_d;
//        pos_x_<=pos_x_d;
//        pos_y_<=pos_y_d;
//        radius_<=radius;
//    end
//end

always @(negedge vsync) begin
    count_f<=0;
    size_<=size_d;
    pos_x_<=pos_x_d;
    pos_y_<=pos_y_d;
    radius_<=radius;
end

// get the radius given radius squared
sqrt uut (.aclk(clk_65mhz),
                .s_axis_cartesian_tdata(square_r),
                .s_axis_cartesian_tvalid(1),
                .m_axis_dout_tdata(radius),
                .m_axis_dout_tvalid(ready4)
        );

// get  radius squared vien area(size__)
divider32 square_xx(.s_axis_divisor_tdata(32'd22),
            .s_axis_divisor_tvalid(1),
            .s_axis_dividend_tdata(size__),
            .s_axis_dividend_tvalid(1),
            .aclk(clk_65mhz),
            .m_axis_dout_tdata({square_r,x_remainder1}),
            .m_axis_dout_tvalid(ready3));

// get x_mean given sum of pixels and number of pixels
divider32 center_xx(.s_axis_divisor_tdata(size_),
            .s_axis_divisor_tvalid(1),
            .s_axis_dividend_tdata(pos_x_),
            .s_axis_dividend_tvalid(1),
            .aclk(clk_65mhz),
            .m_axis_dout_tdata({x_center,x_remainder}),
            .m_axis_dout_tvalid(ready2));

 // compute Y center given sum of y values of pixels and
```

```
     number of pixels ,
 divider32 center_yy(.s_axis_divisor_tdata(size_),
             .s_axis_divisor_tvalid(1),
             .s_axis_dividend_tdata(pos_y_),
             .s_axis_dividend_tvalid(1),
             .aclk(clk_65mhz),
             .m_axis_dout_tdata({y_center,y_remainder}),
             .m_axis_dout_tvalid(ready1));


always_comb begin
    if (use_rgb) threshold=(red>(green+4))&&(red>(blue+4));
    else threshold=(h<h_t+5)&&(s>s_t-20)&&(v>v_t-20);
 end

    parameter DELAY_SIZE=23;
    reg[10:0] hcount_delay  [DELAY_SIZE:0];
    reg [9:0] vcount_delay  [DELAY_SIZE:0];
    reg vsync_delay  [DELAY_SIZE:0];
    reg [4:0] i;
    parameter SEL_D=22;

    always@(posedge clk_65mhz) begin
    //delay the hcount and vcount signals 18 times
    hcount_delay[0]<=hcount;
    vcount_delay[0]<=vcount;
    vsync_delay[0]<=vsync;


//     pixel_out_delay<=pixel_out;
        for(i=1; i<DELAY_SIZE; i=i+1) begin
                    hcount_delay[i] <= hcount_delay[i-1];
                    vcount_delay[i] <= vcount_delay[i-1];
                    vsync_delay[i] <= vsync_delay[i-1];
            end
    end

always @(posedge clk_65mhz) begin
    if (threshold) begin
        thres<=cam;
        size<=size+1;
        pos_x<=pos_x+hcount_delay[SEL_D]; // to use the right values of hcount and vcoun given delay of rgb2hsv
        pos_y<=pos_y+vcount_delay[SEL_D];
       end
    else begin thres=12'b0;end
    if (vsync_delay[SEL_D]) begin
            size_d<=size;
            pos_x_d<=pos_x;
```

```
            pos_y_d<=pos_y;
          end
    else begin size<=0;
        pos_x<=0;
        pos_y<=0;
    end
end

endmodule
```

rgb2hsv.sv

```
`timescale 1ns / 1ps
//
    ////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer: Kevin Zheng Class of 2012
//           Dept of Electrical Engineering &  Computer
    Science
//
// Create Date:    18:45:01 11/10/2010
// Design Name:
// Module Name:    rgb2hsv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
    ////////////////////////////////////////////////////////////////////////////////

module rgb2hsv(clock, reset, r, g, b, h, s, v);
                input wire clock;
                input wire reset;
                input wire [7:0] r;
                input wire [7:0] g;
                input wire [7:0] b;
                output reg [7:0] h;
                output reg [7:0] s;
                output reg [7:0] v;
                reg [7:0] my_r_delay1, my_g_delay1,
                    my_b_delay1;
                reg [7:0] my_r_delay2, my_g_delay2,
                    my_b_delay2;
```

```
                reg [7:0] my_r, my_g, my_b;
                reg [7:0] min, max, delta;
                reg [15:0] s_top;
                reg [15:0] s_bottom;
                reg [15:0] h_top;
                reg [15:0] h_bottom;
                wire [15:0] s_quotient;
                wire [15:0] s_remainder;
                wire s_rfd;
                wire [15:0] h_quotient;
                wire [15:0] h_remainder;
                wire h_rfd;
                reg [7:0] v_delay [19:0];
                reg [18:0] h_negative;
                reg [15:0] h_add [18:0];
                reg [4:0] i;
                // Clocks 4-18: perform all the divisions
                //the s_divider (16/16) has delay 18
                //the hue_div (16/16) has delay 18


logic start;
parameter m=8;
reg[3:1] state=0;
assign start=1'b1;
reg[5:0] count=0;


div_gen_1 hue1(.aclk(clock),
    .s_axis_divisor_tvalid(1'b1),
    .s_axis_divisor_tdata(s_buttom),
    .s_axis_dividend_tvalid(1'b1),
    .s_axis_dividend_tdata(s_top),
    .m_axis_dout_tdata(s_quotient),
    .m_axis_dout_tvalid(s_rfd));

div_gen_1 hue2(.aclk(clock),
    .s_axis_divisor_tvalid(1'b1),
    .s_axis_divisor_tdata(h_buttom),
    .s_axis_dividend_tvalid(1'b1),
    .s_axis_dividend_tdata(h_top),
    .m_axis_dout_tdata(h_quotient),
    .m_axis_dout_tvalid(h_rfd));



always @ (posedge clock) begin

// Clock 1: latch the inputs (always positive)
{my_r, my_g, my_b} <= {r, g, b};
```

```
// Clock 2: compute min, max
{my_r_delay1, my_g_delay1, my_b_delay1} <= {my_r, my_g, my_b
    };
if((my_r >= my_g) && (my_r >= my_b)) //(B,S,S)
        max <= my_r;
else if((my_g >= my_r) && (my_g >= my_b)) //(S,B,S)
        max <= my_g;
else    max <= my_b;

if((my_r <= my_g) && (my_r <= my_b)) //(S,B,B)
        min <= my_r;
else if((my_g <= my_r) && (my_g <= my_b)) //(B,S,B)
        min <= my_g;
else
        min <= my_b;

// Clock 3: compute the delta
{my_r_delay2, my_g_delay2, my_b_delay2} <= {my_r_delay1,
    my_g_delay1, my_b_delay1};
v_delay[0] <= max;
delta <= max - min;

// Clock 4: compute the top and bottom of whatever divisions
     we need to do
s_top <= 8'd255 * delta;
s_bottom <= (v_delay[0]>0)?{8'd0, v_delay[0]}: 16'd1;


if(my_r_delay2 == v_delay[0]) begin
        h_top <= (my_g_delay2 >= my_b_delay2)?
        (my_g_delay2 - my_b_delay2) * 8'd255:
        (my_b_delay2 - my_g_delay2) * 8'd255;
        h_negative[0] <= (my_g_delay2 >= my_b_delay2)?0:1;
        h_add[0] <= 16'd0;
end
else if(my_g_delay2 == v_delay[0]) begin
        h_top <= (my_b_delay2 >= my_r_delay2)?
        (my_b_delay2 - my_r_delay2) * 8'd255:
        (my_r_delay2 - my_b_delay2) * 8'd255;
        h_negative[0] <= (my_b_delay2 >= my_r_delay2)?0:1;
        h_add[0] <= 16'd85;
end
else if(my_b_delay2 == v_delay[0]) begin
    h_top <= (my_r_delay2 >= my_g_delay2)?
    (my_r_delay2 - my_g_delay2) * 8'd255:
    (my_g_delay2 - my_r_delay2) * 8'd255;
    h_negative[0] <= (my_r_delay2 >= my_g_delay2)?0:1;
                            h_add[0] <= 16'd170;
                    end
```

```
                        h_bottom <= (delta > 0)?delta * 8'd6 :16'd6;


                        //delay the v and h_negative signals 18 times
                        for(i=1; i<19; i=i+1) begin
                                v_delay[i] <= v_delay[i-1];
                                h_negative[i] <= h_negative[i-1];
                                h_add[i] <= h_add[i-1];
                end

    v_delay[19] <= v_delay[18];
//Clock 22: compute the final value of h
//depending on the value of h_delay[18],
//we need to subtract 255 from it to make it come back around the circle
    if(h_negative[18] && (h_quotient > h_add[18])) begin
        h <= 8'd255 - h_quotient[7:0] + h_add[18];
    end
    else if(h_negative[18]) begin
            h <= h_add[18] - h_quotient[7:0];
    end
    else begin
        h <= h_quotient[7:0] + h_add[18];
    end

//pass out s and v straight
    s <= s_quotient;
    v <= v_delay[19];
end
endmodule
```

blob.sv

```
// ////////////////////////////////////////////////////////////////////
//
// blob: generate rectangle on screen
//
// ////////////////////////////////////////////////////////////////////
module blob
   #(parameter COLOR = 12'hFFF)  // default color: white
   (input [10:0] x_in,hcount_in,
    input [9:0] y_in,vcount_in,
    input [7:0] width,height,
```

```
     output logic [11:0] pixel_out);

   always_comb begin
      if ((hcount_in >= x_in && hcount_in < (x_in+width)) &&
         (vcount_in >= y_in && vcount_in < (y_in+height)))
        pixel_out = COLOR;
      else pixel_out = 0;
   end
endmodule


module box
   #(parameter THICKNESS = 2,
               COLOR = 12'hFFF)  // default color: white
   (input [10:0] x_in,hcount_in,
    input [9:0] y_in,vcount_in,
    input [7:0] radius_in,
    output logic [11:0] pixel_out);

    logic [6:0] inner;  //length to specify inside the box
    assign inner = radius_in - THICKNESS;

//assuming the ball is always fully inside the picture
   always_comb begin
    //if inside outside frame
      if ((hcount_in >= (x_in-radius_in) && hcount_in < (
          x_in+radius_in)) &&
         (vcount_in >= (y_in-radius_in) && vcount_in < (y_in
            +radius_in))) begin
           //if outside inside frame
               if (~(hcount_in >= (x_in-inner) && hcount_in
                  < (x_in+inner)) |
                   ~(vcount_in >= (y_in-inner) && vcount_in
                     < (y_in+inner))) begin
               pixel_out = COLOR;
           end
           else pixel_out = 0;
           end
      else pixel_out = 0;
   end
endmodule

module cursor
    #(parameter THICKNESS = 1,
                COLOR = 12'hFFF, // default color: white
                WIDTH = 320,     //display width
                HEIGHT = 240     //display height
                )
    (input [10:0] x_in,hcount_in,
    input [9:0] y_in,vcount_in,
```

```
    input sw2,
    output logic [11:0] pixel_out);

    always_comb begin
        if (~sw2) begin
            if ((hcount_in <= WIDTH && vcount_in == y_in) |
                (vcount_in <= HEIGHT && hcount_in == x_in))
                pixel_out = COLOR;
            else pixel_out = 0;
        end
        else begin
          if ((hcount_in <= WIDTH*2 && vcount_in == y_in) |
              (vcount_in <= HEIGHT*2 && hcount_in == x_in))
              pixel_out = COLOR;
          else pixel_out = 0;
        end
    end

endmodule


module colorpad
    #(parameter WIDTH = 30,    //pad width
                HEIGHT = 30,    //pad height
                X = 800,        //pad x
                Y = 600         //pad y
                )
    (input [10:0] hcount_in,
    input [9:0] vcount_in,
    input [11:0] pixel_in,
    output logic [11:0] pixel_out);

    always_comb begin
        if ((hcount_in >= X && hcount_in < (X+WIDTH)) && (
            vcount_in >= Y && vcount_in < (Y+HEIGHT)))
            pixel_out = pixel_in;
        else pixel_out = 0;
    end

endmodule


module speed_bar  //display a bar indicating speed of each
    motor
      #(parameter WIDTH=50, //length of the bar
                  HEIGHT =256, //height of the bar
                  X = 700,  //start pos
                  Y = 300,  //baseline pos
                  TOTAL = WIDTH*5,
                  COLOR = 12'hF00 // default color: red, the
```

```
                          baseline is white
      )
      (input [10:0] hcount_in,
      input [9:0] vcount_in,
      input signed [8:0] speed1,speed2,
      output logic [11:0] pixel_out);

      logic [10:0] x1,x2;
      logic [9:0] y1,y2;
      logic [11:0] motor1,motor2,bar;
      logic [7:0] abs1,abs2;

      assign x1 = X + WIDTH;
      assign x2 = X + WIDTH*3;
      assign y1 = speed1[8]?Y:Y-speed1[7:0];
      assign y2 = speed2[8]?Y:Y-speed2[7:0];
      assign abs1 = speed1[8]?~speed1[7:0]+8'b1:speed1[7:0];
      assign abs2 = speed2[8]?~speed2[7:0]+8'b1:speed2[7:0];

      assign pixel_out = &bar?bar:motor1 + motor2;
//      assign pixel_out = motor1;
      blob #(.COLOR(COLOR))  m1
            (.x_in(x1), .y_in(y1), .hcount_in(hcount_in), .vcount_in(vcount_in), .width(WIDTH), .height(abs1), .pixel_out(motor1));
      blob #(.COLOR(COLOR)) m2
            (.x_in(x2), .y_in(y2), .hcount_in(hcount_in), .vcount_in(vcount_in), .width(WIDTH), .height(abs2), .pixel_out(motor2));

      always_comb begin
          if ((hcount_in >= X && hcount_in < (X+TOTAL)) && (vcount_in >= Y && vcount_in < (Y+1))) bar = 12'hFFF;
          else bar = 0;
      end

endmodule

module arrow    //To be continued
    #(parameter WIDTH=50, //length of the bar
            HEIGHT =256, //height of the bar
            X = 800,  //start pos
            Y = 200,  //baseline pos
            TOTAL = WIDTH*5,
            COLOR = 12'hF00 // default color: red, the baseline is white
            )
      (input [10:0] hcount_in,
      input [9:0] vcount_in,
```

```
      input signed [8:0] speed1,speed2,
      output logic [11:0] pixel_out);

endmodule
```

camera_read.sv

```
module camera_read(
        input  p_clock_in,
        input  vsync_in,
        input  href_in,
        input  [7:0] p_data_in,
        output logic [15:0] pixel_data_out,
        output logic pixel_valid_out,
        output logic frame_done_out
    );

        rgb2hsv xx(.clock(), .reset(), .r(), .g(), .b(), .h
            (),.s(), .v());//
        logic [1:0] FSM_state = 0;
    logic pixel_half = 0;

        localparam WAIT_FRAME_START = 0;
        localparam ROW_CAPTURE = 1;


        always_ff@(posedge p_clock_in)
        begin
        case(FSM_state)

        WAIT_FRAME_START: begin //wait for VSYNC
           FSM_state <= (!vsync_in) ? ROW_CAPTURE :
               WAIT_FRAME_START;
           frame_done_out <= 0;
           pixel_half <= 0;
        end

        ROW_CAPTURE: begin
           FSM_state <= vsync_in ? WAIT_FRAME_START :
               ROW_CAPTURE;
           frame_done_out <= vsync_in ? 1 : 0;
           pixel_valid_out <= (href_in && pixel_half) ? 1 :
               0;
           if (href_in) begin
               pixel_half <= ~ pixel_half;
               if (pixel_half) pixel_data_out[7:0] <=
                   p_data_in;
               else pixel_data_out[15:8] <= p_data_in;
           end
        end
        endcase
```

```
        end

endmodule
```

control.sv

```
`timescale 1ns / 1ps


module control(
                input clk_in,
                input rst_in,
                input ready_in,
                input [8:0] cur_pos_x,
                input [8:0] cur_pos_y,
                input [6:0] cur_rad,
                input [6:0] goal_rad,
                input [15:0] params,
                output logic signed [8:0] speed,
                output logic signed [8:0] turn
);

//camera size
parameter HEIGHT = 240;
parameter WIDTH = 320;

//modes
parameter FORWARD = 0;
parameter DIRECT = 1;
parameter GOALKEEP = 3;
parameter CHASE = 2;

//assign params
logic signed [4:0] Ksp;
logic signed [3:0] Ksd;
logic signed [4:0] Ktp;
logic signed [3:0] Ktd;
logic [1:0] mode;

assign {Ksp[3:0],Ksd[2:0],Ktp[3:0],Ktd[2:0],mode} = params;

assign Ksp[4] = 0;
assign Ksd[3] = 0;
assign Ktp[4] = 0;
assign Ktd[3] = 0;

//desired x,r
logic [8:0] x_d;
logic [6:0] r_d;
assign x_d = WIDTH >> 1;
assign r_d = goal_rad;
```

```
//current x,rad,dx,dr
logic [8:0] x;
logic [6:0] r;
logic [8:0] dx;
logic [6:0] dr;

//previous x,rad,dx,dr
logic [8:0] pre_x;
logic [6:0] pre_r;
logic [8:0] pre_dx;
logic [6:0] pre_dr;


//errors
logic signed [8:0] e_x;
logic signed [8:0] e_r;
logic signed [8:0] e_dx;
logic signed [8:0] e_dr;

assign e_x = x_d - x;   //abs less than 240
assign e_r = r_d - r;


//raw speed,turn
logic signed [16:0] raw_speed;
logic signed [16:0] raw_turn;


//threshold the output
logic [7:0] pass1;
logic [7:0] pass2;

threshold_by_abs threshold_speed(.signed_in(raw_speed), .threshold(16'h00ff), .signed_out({speed[8],pass1,speed[7:0]}));
threshold_by_abs threshold_turn(.signed_in(raw_turn), .threshold(16'h00ff), .signed_out({turn[8],pass2,turn[7:0]}));



always_comb begin
    case(mode)
        GOALKEEP:begin
                    raw_speed = (Ksp * e_x) + (Ksd * e_dx);
                    raw_turn = 0;

                 end
        CHASE:  begin
```

```
                    raw_speed = (Ksp * e_r) + (Ksd * e_dr);
                    raw_turn = (Ktp * e_x) + (Ktd * e_dx);
                end
        DIRECT: begin
                    raw_speed = {params[15],8'd0,params[14:8],1'b0};
                    raw_turn = {params[7],8'd0,params[6:0],1'b0};
                end
        default:begin
                    raw_speed = 0;
                    raw_turn = 0;
                end
    endcase
end

always_ff @(posedge clk_in) begin
    if(rst_in) begin
        //initialize
        x <= 0;
        r <= 0;
        dx <= 0;
        dr <= 0;
        pre_x <= 0;
        pre_r <= 0;
        pre_dx <= 0;
        pre_dr <= 0;
        end

    else begin
        if(ready_in) begin
            x <= cur_pos_x;
            r <= cur_rad;
            e_dx <= x - cur_pos_x;
            e_dr <= r - cur_rad;
        end
    end
end
endmodule



module threshold_by_abs(
        input signed [16:0] signed_in,
        input [15:0] threshold,
        output signed [16:0] signed_out
);

logic sign;
logic [15:0] abs;
```

```
assign sign = signed_in[16];
assign signed_out[16] = sign;

assign abs = sign?~signed_in[15:0] + 16'h0001:signed_in[15:0];
assign signed_out[15:0] = (abs <= threshold)? signed_in[15:0]:(sign?~threshold+16'h0001:threshold);
endmodule
```

xvga.sv

```


// //////////////////////////////////////////////////////////////////////////////
// Update: 8/8/2019 GH
// Create Date: 10/02/2015 02:05:19 AM
// Module Name: xvga
//
// xvga: Generate VGA display signals (1024 x 768 @ 60Hz)
//
//                              ---- HORIZONTAL -----     ------VERTICAL -----
//                              Active                    Active
//                    Freq      Video   FP  Sync   BP     Video   FP  Sync  BP
//   640x480, 60Hz    25.175    640     16    96   48     480     11   2    31
//   800x600, 60Hz    40.000    800     40   128   88     600      1   4    23
//   1024x768, 60Hz   65.000    1024    24   136  160     768      3   6    29
//   1280x1024, 60Hz  108.00    1280    48   112  248     768      1   3    38
//   1280x720p 60Hz   75.25     1280    72    80  216     720      3   5    30
//   1920x1080 60Hz   148.5     1920    88    44  148     1080     4   5    36
//
// change the clock frequency, front porches, sync's, and back porches to create
// other screen resolutions
// //////////////////////////////////////////////////////////////////////////////

module xvga(input vclock_in,
            output reg [10:0] hcount_out,    // pixel number
```

```
                on current line
            output reg [9:0] vcount_out,     // line number
            output reg vsync_out, hsync_out,
            output reg blank_out);

    parameter DISPLAY_WIDTH  = 1024;       // display width
    parameter DISPLAY_HEIGHT = 768;        // number of lines

    parameter  H_FP = 24;                  // horizontal front porch
    parameter  H_SYNC_PULSE = 136;         // horizontal sync
    parameter  H_BP = 160;                 // horizontal back porch

    parameter  V_FP = 3;                   // vertical front porch
    parameter  V_SYNC_PULSE = 6;           // vertical sync
    parameter  V_BP = 29;                  // vertical back porch

    // horizontal: 1344 pixels total
    // display 1024 pixels per line
    reg hblank,vblank;
    wire hsyncon,hsyncoff,hreset,hblankon;
    assign hblankon = (hcount_out == (DISPLAY_WIDTH -1));
    assign hsyncon = (hcount_out == (DISPLAY_WIDTH + H_FP - 1));  //1047
    assign hsyncoff = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE - 1));  // 1183
    assign hreset = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE + H_BP - 1));  //1343

    // vertical: 806 lines total
    // display 768 lines
    wire vsyncon,vsyncoff,vreset,vblankon;
    assign vblankon = hreset & (vcount_out == (DISPLAY_HEIGHT - 1));   // 767
    assign vsyncon = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP - 1));  // 771
    assign vsyncoff = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE - 1));  // 777
    assign vreset = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE + V_BP - 1)); // 805

    // sync and blanking
    wire next_hblank,next_vblank;
    assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
    assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
    always_ff @(posedge vclock_in) begin
       hcount_out <= hreset ? 0 : hcount_out + 1;
```

```
      hblank <= next_hblank;
      hsync_out <= hsyncon ? 0 : hsyncoff ? 1 : hsync_out;  // active low

      vcount_out <= hreset ? (vreset ? 0 : vcount_out + 1) : vcount_out;
      vblank <= next_vblank;
      vsync_out <= vsyncon ? 0 : vsyncoff ? 1 : vsync_out;  // active low

      blank_out <= next_vblank | (next_hblank & ~hreset);
   end

endmodule
```

clk_wiz_lab3.v

```
// file: clk_wiz_lab3.v
//
// (c) Copyright 2008 - 2013 Xilinx, Inc. All rights reserved.
//
// This file contains confidential and proprietary information
// of Xilinx, Inc. and is protected under U.S. and
// international copyright and other intellectual property
// laws.
//
// DISCLAIMER
// This disclaimer is not a license and does not grant any
// rights to the materials distributed herewith. Except as
// otherwise provided in a valid license issued to you by
// Xilinx, and to the maximum extent permitted by applicable
// law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
// WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
// AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
// BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
// INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
// (2) Xilinx shall not be liable (whether in contract or tort,
// including negligence, or under any other theory of
// liability) for any loss or damage of any kind or nature
// related to, arising under or in connection with these
// materials, including for any direct, or any indirect,
// special, incidental, or consequential loss or damage
// (including loss of data, profits, goodwill, or any type of
// loss or damage suffered as a result of any action brought
// by a third party) even if such damage or loss was
// reasonably foreseeable or Xilinx had been advised of the
```

```
// possibility of the same.
//
// CRITICAL APPLICATIONS
// Xilinx products are not designed or intended to be fail-
// safe, or for use in any application requiring fail-safe
// performance, such as life-support or safety devices or
// systems, Class III medical devices, nuclear facilities,
// applications related to the deployment of airbags, or any
// other applications that could lead to death, personal
// injury, or severe property or environmental damage
// (individually and collectively, "Critical
// Applications"). Customer assumes the sole risk and
// liability of any use of Xilinx products in Critical
// Applications, subject only to applicable laws and
// regulations governing limitations on product liability.
//
// THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
// PART OF THIS FILE AT ALL TIMES.
//
// ----------------------------------------------------------------------------
// User entered comments
// ----------------------------------------------------------------------------
// None
//
// ----------------------------------------------------------------------------
//  Output     Output      Phase    Duty Cycle   Pk-to-Pk     Phase
//   Clock     Freq (MHz)  (degrees)    (%)     Jitter (ps)  Error (ps)
// ----------------------------------------------------------------------------
// CLK_OUT1____65.000______0.000______50.0______254.866____297.890
//
// ----------------------------------------------------------------------------
// Input Clock   Freq (MHz)    Input Jitter (UI)
// ----------------------------------------------------------------------------
// __primary_________100.000____________0.010

```

```
`timescale 1ps/1ps

module clk_wiz_lab3
 (// Clock in ports
  input         clk_in1,
  // Clock out ports
  output        clk_out1
 );

  // Input buffering
  //------------------------------------
  IBUF clkin1_ibufg
   (.O (clk_in1_clk_wiz_0),
    .I (clk_in1));



  // Clocking PRIMITIVE
  //------------------------------------

  // Instantiation of the MMCM PRIMITIVE
  //    * Unused inputs are tied off
  //    * Unused outputs are labeled unused
  wire [15:0] do_unused;
  wire        drdy_unused;
  wire        psdone_unused;
  wire        locked_int;
  wire        clkfbout_clk_wiz_0;
  wire        clkfbout_buf_clk_wiz_0;
  wire        clkfboutb_unused;
    wire clkout0b_unused;
   wire clkout1_unused;
   wire clkout1b_unused;
   wire clkout2_unused;
   wire clkout2b_unused;
   wire clkout3_unused;
   wire clkout3b_unused;
   wire clkout4_unused;
  wire        clkout5_unused;
  wire        clkout6_unused;
  wire        clkfbstopped_unused;
  wire        clkinstopped_unused;

  MMCME2_ADV
  #(.BANDWIDTH            ("OPTIMIZED"),
    .CLKOUT4_CASCADE      ("FALSE"),
    .COMPENSATION         ("ZHOLD"),
    .STARTUP_WAIT         ("FALSE"),
    .DIVCLK_DIVIDE        (5),
    .CLKFBOUT_MULT_F      (50.375),
```

```
    .CLKFBOUT_PHASE       (0.000),
    .CLKFBOUT_USE_FINE_PS ("FALSE"),
    .CLKOUT0_DIVIDE_F     (15.500),
    .CLKOUT0_PHASE        (0.000),
    .CLKOUT0_DUTY_CYCLE   (0.500),
    .CLKOUT0_USE_FINE_PS  ("FALSE"),
    .CLKIN1_PERIOD        (10.0))
  mmcm_adv_inst
    // Output clocks
   (
    .CLKFBOUT            (clkfbout_clk_wiz_0),
    .CLKFBOUTB           (clkfboutb_unused),
    .CLKOUT0             (clk_out1_clk_wiz_0),
    .CLKOUT0B            (clkout0b_unused),
    .CLKOUT1             (clkout1_unused),
    .CLKOUT1B            (clkout1b_unused),
    .CLKOUT2             (clkout2_unused),
    .CLKOUT2B            (clkout2b_unused),
    .CLKOUT3             (clkout3_unused),
    .CLKOUT3B            (clkout3b_unused),
    .CLKOUT4             (clkout4_unused),
    .CLKOUT5             (clkout5_unused),
    .CLKOUT6             (clkout6_unused),
     // Input clock control
    .CLKFBIN             (clkfbout_buf_clk_wiz_0),
    .CLKIN1              (clk_in1_clk_wiz_0),
    .CLKIN2              (1'b0),
     // Tied to always select the primary input clock
    .CLKINSEL            (1'b1),
    // Ports for dynamic reconfiguration
    .DADDR               (7'h0),
    .DCLK                (1'b0),
    .DEN                 (1'b0),
    .DI                  (16'h0),
    .DO                  (do_unused),
    .DRDY                (drdy_unused),
    .DWE                 (1'b0),
    // Ports for dynamic phase shift
    .PSCLK               (1'b0),
    .PSEN                (1'b0),
    .PSINCDEC            (1'b0),
    .PSDONE              (psdone_unused),
    // Other control and status signals
    .LOCKED              (locked_int),
    .CLKINSTOPPED        (clkinstopped_unused),
    .CLKFBSTOPPED        (clkfbstopped_unused),
    .PWRDWN              (1'b0),
    .RST                 (1'b0));
```

```
    // Output buffering
    //-----------------------------------

    BUFG clkf_buf
     (.O (clkfbout_buf_clk_wiz_0),
      .I (clkfbout_clk_wiz_0));



    BUFG clkout1_buf
     (.O   (clk_out1),
      .I   (clk_out1_clk_wiz_0));




endmodule
```

display_8hex.sv

```

// ////////////////////////////////////////////////////////////////////////////////
// Engineer:   g.p.hom
//
// Create Date:    18:18:59 04/21/2013
// Module Name:    display_8hex
// Description:  Display 8 hex numbers on 7 segment display
//
// ////////////////////////////////////////////////////////////////////////////////

module display_8hex(
    input clk_in,                  // system clock
    input [31:0] data_in,          // 8 hex numbers, msb first
    output reg [6:0] seg_out,      // seven segment display output
    output reg [7:0] strobe_out    // digit strobe
    );

    localparam bits = 13;

    reg [bits:0] counter = 0;  // clear on power up

    wire [6:0] segments[15:0]; // 16 7 bit memorys
    assign segments[0]  = 7'b100_0000;  // inverted logic
    assign segments[1]  = 7'b111_1001;  // gfedcba
```

```
    assign segments[2]  = 7'b010_0100;
    assign segments[3]  = 7'b011_0000;
    assign segments[4]  = 7'b001_1001;
    assign segments[5]  = 7'b001_0010;
    assign segments[6]  = 7'b000_0010;
    assign segments[7]  = 7'b111_1000;
    assign segments[8]  = 7'b000_0000;
    assign segments[9]  = 7'b001_1000;
    assign segments[10] = 7'b000_1000;
    assign segments[11] = 7'b000_0011;
    assign segments[12] = 7'b010_0111;
    assign segments[13] = 7'b010_0001;
    assign segments[14] = 7'b000_0110;
    assign segments[15] = 7'b000_1110;

    always_ff @(posedge clk_in) begin
      // Here I am using a counter and select 3 bits which
          provides
      // a reasonable refresh rate starting the left most
          digit
      // and moving left.
      counter <= counter + 1;
      case (counter[bits:bits-2])
          3'b000: begin  // use the MSB 4 bits
                  seg_out <= segments[data_in[31:28]];
                  strobe_out <= 8'b0111_1111 ;
                 end

          3'b001: begin
                  seg_out <= segments[data_in[27:24]];
                  strobe_out <= 8'b1011_1111 ;
                 end

          3'b010: begin
                   seg_out <= segments[data_in[23:20]];
                   strobe_out <= 8'b1101_1111 ;
                  end
          3'b011: begin
                  seg_out <= segments[data_in[19:16]];
                  strobe_out <= 8'b1110_1111;
                 end
          3'b100: begin
                  seg_out <= segments[data_in[15:12]];
                  strobe_out <= 8'b1111_0111;
                 end

          3'b101: begin
                  seg_out <= segments[data_in[11:8]];
                  strobe_out <= 8'b1111_1011;
                 end
```

```
            3'b110: begin
                      seg_out <= segments[data_in[7:4]];
                      strobe_out <= 8'b1111_1101;
                    end
            3'b111: begin
                    seg_out <= segments[data_in[3:0]];
                    strobe_out <= 8'b1111_1110;
                  end

         endcase
       end

endmodule
```

initialize.sv

```
`timescale 1ns / 1ps

//given the camera pixel, outputs everything to display
module initialize(
  input clk_65mhz,
  input reset,
  input [10:0] hcount,
  input [9:0] vcount,
  input vsync,
  input [3:0] directions, //up,down,left,right
  input confirm_in,
  input activate_in,
  input sw2, //whether to make the size twice
  input [11:0] cam,
  input [8:0] cur_pos_x,
  input [8:0] cur_pos_y,
  input [6:0] cur_rad,
  input signed [8:0] speed1,speed2,
  output logic [11:0] pixel_out,
  output logic [11:0] goal_pixel,
  output logic [6:0] goal_rad,
  output logic track,
  output logic move,
  //for debug
  output logic [2:0] state,
  output logic [10:0] cursor_x,
  output logic [9:0] cursor_y,
  output logic up,down,left,right
  );

  logic [8:0] height;
  logic [9:0] width;

  assign height = sw2?9'd480:9'd240;
```

```
  assign width = sw2?10'd640:10'd320;


  //generate the blobs
  logic [11:0] box,box_confirmed,cursor,pad,speed_bar,selected_pixel,selected_buff,goal_pad;
  //logic [10:0] cursor_x;
  //logic [9:0] cursor_y;

  box box_gen(.x_in({2'b00,cur_pos_x}), .y_in({1'b0,cur_pos_y}), .hcount_in(hcount), .vcount_in(vcount), .radius_in(cur_rad), .pixel_out(box));
  box #(.COLOR(12'hf00)) confirmed_box_gen (.x_in({2'b00,cur_pos_x}), .y_in({1'b0,cur_pos_y}), .hcount_in(hcount), .vcount_in(vcount), .radius_in(cur_rad), .pixel_out(box_confirmed));
  cursor cursor_gen(.x_in(cursor_x), .y_in(cursor_y), .hcount_in(hcount), .vcount_in(vcount), .sw2(sw2), .pixel_out(cursor));
  colorpad colorpad_gen(.pixel_in(selected_buff), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(pad));
  colorpad goal_colorpad_gen(.pixel_in(goal_pixel), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(goal_pad));
  speed_bar speed_bar_gen(.speed1(speed1), .speed2(speed2), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(speed_bar));

  //////////////////////////////////cursor control///////////////////////////////////////////////////////

  parameter CURSORSPEED = 3;
  parameter SAMPLESIZE = 1; // (0 -> 1, 1 -> 4, 2 -> 16)

//  logic up,down,left,right;
  logic [7:0] sum_r,sum_g,sum_b,sum_r_d,sum_g_d,sum_b_d;
  logic [3:0] shifted_r,shifted_g,shifted_b;

  assign shifted_r = sum_r >> 4;
  assign shifted_g = sum_g >> 4;
  assign shifted_b = sum_b >> 4;


  //single sample
  //assign selected_pixel = (vcount == cursor_y && hcount == cursor_x)? cam:selected_pixel;

  //assign selected_pixel = {shifted_r,shifted_g,shifted_b};
  logic confirm,confirm_serial,old_confirmed,activate;
  assign confirm = confirm_serial & ~old_confirmed; //pulse
```

```
  debounce db1(.reset_in(reset),.clock_in(clk_65mhz),.
      noisy_in(directions[3]),.clean_out(up));
  debounce db2(.reset_in(reset),.clock_in(clk_65mhz),.
      noisy_in(directions[2]),.clean_out(down));
  debounce db3(.reset_in(reset),.clock_in(clk_65mhz),.
      noisy_in(directions[1]),.clean_out(left));
  debounce db4(.reset_in(reset),.clock_in(clk_65mhz),.
      noisy_in(directions[0]),.clean_out(right));
  debounce db5(.reset_in(reset),.clock_in(clk_65mhz),.
      noisy_in(confirm_in),.clean_out(confirm_serial));
  debounce db6(.reset_in(reset),.clock_in(clk_65mhz),.
      noisy_in(activate_in),.clean_out(activate));

  always_ff @(posedge vsync) begin
    if(reset) begin
        cursor_x <= 11'h00f;
        cursor_y <= 10'h00f;


        //selected_buff <= 12'hff0;
    end else begin
        if (up) begin
            if(cursor_y < CURSORSPEED) cursor_y <= 0;
            else cursor_y <= cursor_y -  CURSORSPEED;

            end

        if (down) begin
            if (cursor_y > height - CURSORSPEED) cursor_y <=
                height;
            else cursor_y <= cursor_y + CURSORSPEED;
            end

        if (left) begin
            if(cursor_x < CURSORSPEED) cursor_x <= 0;
            else cursor_x <= cursor_x -  CURSORSPEED;
            end

        if (right) begin
            if (cursor_x > width - CURSORSPEED) cursor_x <=
               width;
            else cursor_x <= cursor_x + CURSORSPEED;
            end
        end
    end

//  always_ff @(posedge clk_65mhz) begin
//     //update pixel selected by cursor (average 4 or 16bits
     around)
```

```
//    if ((vcount >= cursor_y - SAMPLESIZE && vcount < cursor_y) && (hcount >= cursor_x - SAMPLESIZE && hcount < cursor_x + SAMPLESIZE)) begin
//              sum_r <= sum_r + cam[11:8];
//              sum_g <= sum_g + cam[7:4];
//              sum_b <= sum_b + cam[3:0];
//        end
//    if(vcount == 10'd550 && hcount ==11'd750) begin
//        sum_r <= 0;
//        sum_g <= 0;
//        sum_b <= 0;
//        selected_buff <= selected_buff + 1;
////        selected_buff <= 12'hf00;
//    end
//end //end always_ff


//////////////////////////////////////////////////end cursor control///////////////////////////////

/////////////////////////////////////////////////main FSM ///////////////////////////////////////////
parameter INITIALIZE = 0;
parameter SELECTED = 1;
parameter CONFIRMED = 2;
parameter MOVE = 3;
parameter PAUSE = 4;

//logic [1:0] state;
logic [2:0] old_state;
logic old_activate;
logic activated;
logic selected;
logic confirmed;

assign activated = ~old_activate & activate;     //if switched to activated
assign selected = (state==SELECTED && old_state==INITIALIZE);
assign confirmed = (state==CONFIRMED && old_state==SELECTED);

always_ff @(posedge clk_65mhz) begin
    if(reset) begin
        //initialize
        old_state <= 0;
        old_activate <= 0;
        state <= INITIALIZE;
        goal_rad <= 0;
        goal_pixel <= 0;
```

```
        pixel_out <= 0;
        old_confirmed <= 0;
        selected_buff <= 12'h000;
        sum_r <= 0;
        sum_g <= 0;
        sum_b <= 0;
    end else begin
    //pad
        if(vcount == cursor_y && hcount == cursor_x) selected_buff <= cam;
        if ((vcount >= cursor_y - SAMPLESIZE && vcount < cursor_y) && (hcount >= cursor_x - SAMPLESIZE && hcount < cursor_x + SAMPLESIZE)) begin
            sum_r <= sum_r + cam[11:8];
            sum_g <= sum_g + cam[7:4];
            sum_b <= sum_b + cam[3:0];
        end
        if(vcount == 10'd550 && hcount == 11'd750) begin
            sum_r <= 0;
            sum_g <= 0;
            sum_b <= 0;
//            selected_buff <= {sum_r[5:2],sum_g[5:2],sum_b[5:2]};
//            selected_buff <= 12'hf00;
        end

        old_state <= state;
        old_activate <= activate;
        old_confirmed <= confirm_serial;
        //switch to initialize every time activated
        if(activated) state <= INITIALIZE;
        //get the goal pixel
        if(selected) goal_pixel <= selected_buff;
        //get the goal radius
        if(confirmed) goal_rad <= cur_rad;

        case(state)
          INITIALIZE: begin
            track <= 0;
            move <= 0;
            if(confirm) state <= SELECTED;
            pixel_out <= &cursor?cursor:cam+pad;
            end
          SELECTED: begin
            track <= 1;
            move <= 0;
            if(up | down | left | right) state <= INITIALIZE;
            if(confirm) state <= CONFIRMED;
            pixel_out <= (&cursor | &box)?cursor+box:cam +
```

```
                    goal_pad;

            end
          CONFIRMED: begin
            track <= 1;
            move <= 0;
            if(~activate) state <= MOVE;
            pixel_out <= &box?box_confirmed:cam+speed_bar+
                goal_pad;
            end
          MOVE: begin
            track <= 1;
            move <= 1;
            pixel_out <= &box?box_confirmed:cam+goal_pad+
                speed_bar;
            if(confirm) state <= PAUSE;
            end
          PAUSE: begin
            track <= 0;
            move <= 0;
            pixel_out <= &box?box_confirmed:cam+goal_pad+
                speed_bar;
            if(confirm) state <= MOVE;
            end
        endcase
    end
end

endmodule
```

control.sv

```
`timescale 1ns / 1ps


module control(
                input clk_in,
                input rst_in,
                input ready_in,
                input [8:0] cur_pos_x,
                input [8:0] cur_pos_y,
                input [6:0] cur_rad,
                input [6:0] goal_rad,
                input [15:0] params,
                output logic signed [8:0] speed,
                output logic signed [8:0] turn
);

//camera size
parameter HEIGHT = 240;
parameter WIDTH = 320;
```

```
//modes
parameter FORWARD = 0;
parameter DIRECT = 1;
parameter GOALKEEP = 3;
parameter CHASE = 2;

//assign params
logic signed [4:0] Ksp;
logic signed [3:0] Ksd;
logic signed [4:0] Ktp;
logic signed [3:0] Ktd;
logic [1:0] mode;

assign {Ksp[3:0],Ksd[2:0],Ktp[3:0],Ktd[2:0],mode} = params;

assign Ksp[4] = 0;
assign Ksd[3] = 0;
assign Ktp[4] = 0;
assign Ktd[3] = 0;

//desired x,r
logic [8:0] x_d;
logic [6:0] r_d;
assign x_d = WIDTH >> 1;
assign r_d = goal_rad;

//current x,rad,dx,dr
logic [8:0] x;
logic [6:0] r;
logic [8:0] dx;
logic [6:0] dr;

//previous x,rad,dx,dr
logic [8:0] pre_x;
logic [6:0] pre_r;
logic [8:0] pre_dx;
logic [6:0] pre_dr;


//errors
logic signed [8:0] e_x;
logic signed [8:0] e_r;
logic signed [8:0] e_dx;
logic signed [8:0] e_dr;

assign e_x = x_d - x;    //abs less than 240
assign e_r = r_d - r;


```

```
//raw speed,turn
logic signed [16:0] raw_speed;
logic signed [16:0] raw_turn;


//threshold the output
logic [7:0] pass1;
logic [7:0] pass2;

threshold_by_abs threshold_speed(.signed_in(raw_speed), .threshold(16'h00ff), .signed_out({speed[8],pass1,speed[7:0]}));
threshold_by_abs threshold_turn(.signed_in(raw_turn), .threshold(16'h00ff), .signed_out({turn[8],pass2,turn[7:0]}));



always_comb begin
    case(mode)
        GOALKEEP:begin
                    raw_speed = (Ksp * e_x) + (Ksd * e_dx);
                    raw_turn = 0;

                end
        CHASE:  begin
                    raw_speed = (Ksp * e_r) + (Ksd * e_dr);
                    raw_turn = (Ktp * e_x) + (Ktd * e_dx);
                end
        DIRECT: begin
                   raw_speed = {params[15],8'd0,params[14:8],1'b0};
                   raw_turn = {params[7],8'd0,params[6:0],1'b0};
                end
        default:begin
                    raw_speed = 0;
                    raw_turn = 0;
                end
    endcase
end

always_ff @(posedge clk_in) begin
    if(rst_in) begin
        //initialize
        x <= 0;
        r <= 0;
        dx <= 0;
        dr <= 0;
        pre_x <= 0;
```

```
            pre_r <= 0;
            pre_dx <= 0;
            pre_dr <= 0;
            end

        else begin
            if(ready_in) begin
                x <= cur_pos_x;
                r <= cur_rad;
                e_dx <= x - cur_pos_x;
                e_dr <= r - cur_rad;
            end
        end
end
endmodule



module threshold_by_abs(
        input signed [16:0] signed_in,
        input [15:0] threshold,
        output signed [16:0] signed_out
);

logic sign;
logic [15:0] abs;

assign sign = signed_in[16];
assign signed_out[16] = sign;

assign abs = sign?~signed_in[15:0] + 16'h0001:signed_in[15:0
    ];
assign signed_out[15:0] = (abs <= threshold)? signed_in[15:0
    ]:(sign?~threshold+16'h0001:threshold);
endmodule
```

motor_out.sv

```
`timescale 1ns / 1ps

module motor_out(input clk_in,
                input rst_in,
                input[7:0] offset,  //minimum speed needed
                    to drive the car
                input signed [8:0] speed_in,
                input signed [8:0] turn_in,
                output logic [5:0] motor_out,    //en1,ina1,
                    inb1,ina2,inb2,en2
                output logic [7:0] speed_1,speed_2
    );

```

```
logic signed [9:0] raw_motor1;
logic signed [9:0] raw_motor2;
logic [8:0] expanded_1;
logic [8:0] expanded_2;
logic [7:0] speed_1_offset; //without offset
logic [7:0] speed_2_offset; //without offset

logic forward_1;
logic forward_2;

assign raw_motor1 = speed_in - (turn_in >>> 1);
assign raw_motor2 = speed_in + (turn_in >>> 1);

assign forward_1 = ~raw_motor1[9];
assign forward_2 = ~raw_motor2[9];

assign expanded_1 = forward_1?raw_motor1[8:0]:~raw_motor1[8:
    0] + 9'h001;
assign expanded_2 = forward_2?raw_motor2[8:0]:~raw_motor2[8:
    0] + 9'h001;

assign speed_1_offset = expanded_1[8]?8'hff:expanded_1[7:0];
assign speed_2_offset = expanded_2[8]?8'hff:expanded_2[7:0];

assign speed_1 = (speed_1_offset>8'hff-offset)?8'hff:
    speed_1_offset+offset;
assign speed_2 = (speed_2_offset>8'hff-offset)?8'hff:
    speed_2_offset+offset;

pwm en1(.clk_in(clk_in), .rst_in(rst_in), .level_in(speed_1)
    , .pwm_out(motor_out[5]));
pwm en2(.clk_in(clk_in), .rst_in(rst_in), .level_in(speed_2)
    , .pwm_out(motor_out[0]));

assign motor_out[4:1] = {forward_1,~forward_1,forward_2,~
    forward_2};



endmodule


module pwm (input clk_in, input rst_in, input [7:0] level_in
    , output logic pwm_out);
    logic [7:0] count;
    assign pwm_out = count<level_in;
    always_ff @(posedge clk_in)begin
        if (rst_in)begin
            count <= 8'b0;
```

```
        end else begin
            count <= count+8'b1;
        end
    end
endmodule
```

color_class.sv

```
`timescale 1ns / 1ps
//
    ////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer:
//
// Create Date: 12/09/2019 02:46:49 PM
// Design Name:
// Module Name: color_class
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
    ////////////////////////////////////////////////////////////////////////////////



module color_class(
        input [7:0] h,
        input [7:0] s,
        input [7:0] v,
        output reg [2:0] cls );
 always_comb begin
     if ((h<=8'd3) &&(s>8'd50)&&(v>8'd50)) cls=3'b001;
     else if ((h>8'd3)&&(h<8'd10)&&(s>8'd100)&&(v>8'd100))
         cls=3'b100;
     else begin
        if ((h<=8'd32)&& (h>8'd20) &&(s>8'd100)&&( v>8'd100)
            ) cls=3'b010; // yellow
     else begin
        if ((h<=8'd32) &&(s>8'd100)&&( v>8'd100)) cls=3'b011
            ;
      else cls=3'b111;
     end
```

```
    end
end

endmodule
```