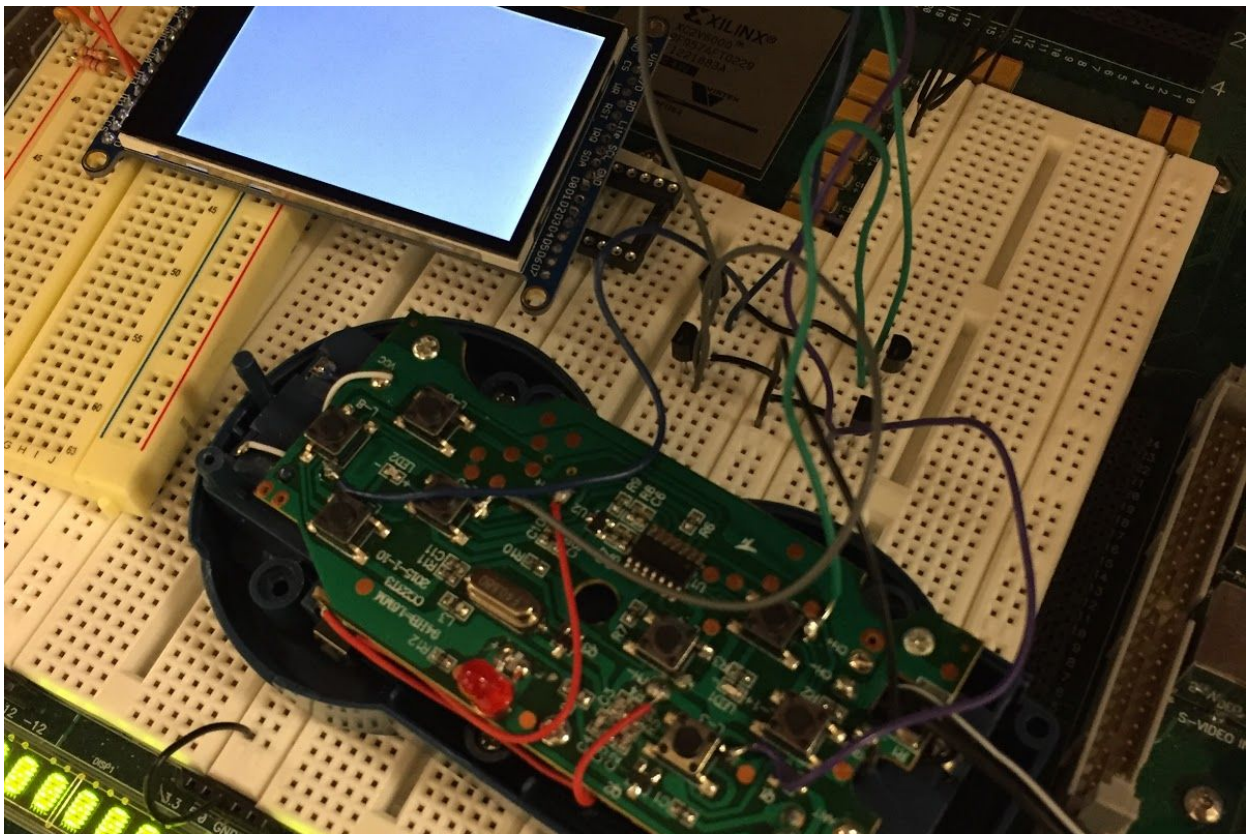


Touch & Go

Fall 2016

Natalie Mionis and Tianye Chen

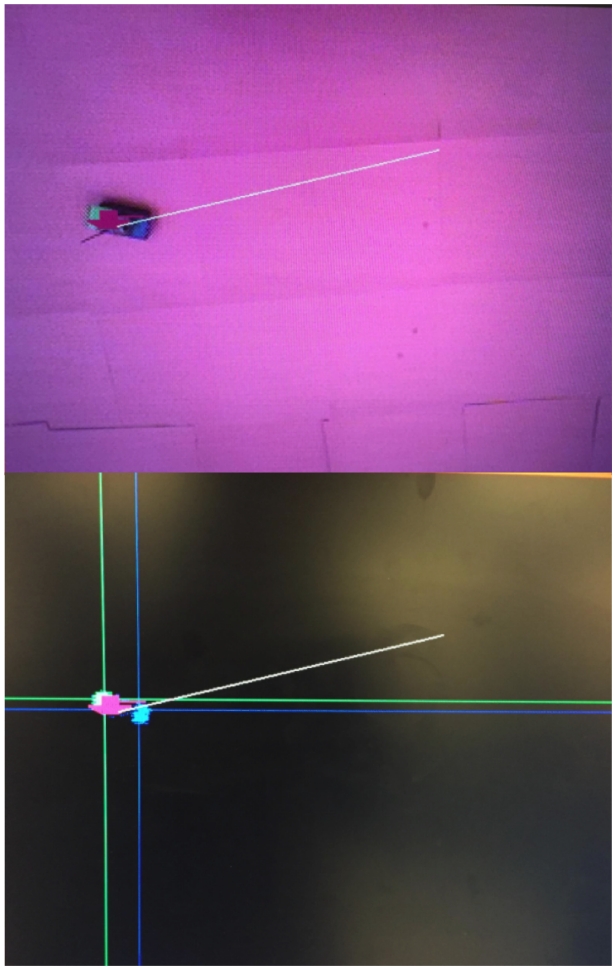
6.111 Final Project

Table of Contents

Overview

Touch & Go is an interactive interface through which a user can drag a remote controlled tank across the ground using a touch screen. We use an Adafruit capacitive touchscreen to serve as the interface between the user and a small Tiger remote controlled tank. A National Television System Committee Standard (NTSC) camera provides real time feedback of the tank's angle and position. We decided to use the labkit with a Xilinx FPGA to more easily work with the NTSC camera.

Based on the user's touch on the touchscreen, we direct the tank to drive to the user's touch on the touchscreen. Additionally, the NTSC camera provides constant feedback of the tank's current angle and position, so we can redirect the tank when the finger position on the touchscreen changes. The movements of the tank can be viewed on the VGA monitor, which has two modes of operation. The first displays the RGB image from the camera, as well as the path drawn. The second view displays the filtered camera view and center of mass calculations, as well as the path drawn on the touchscreen.

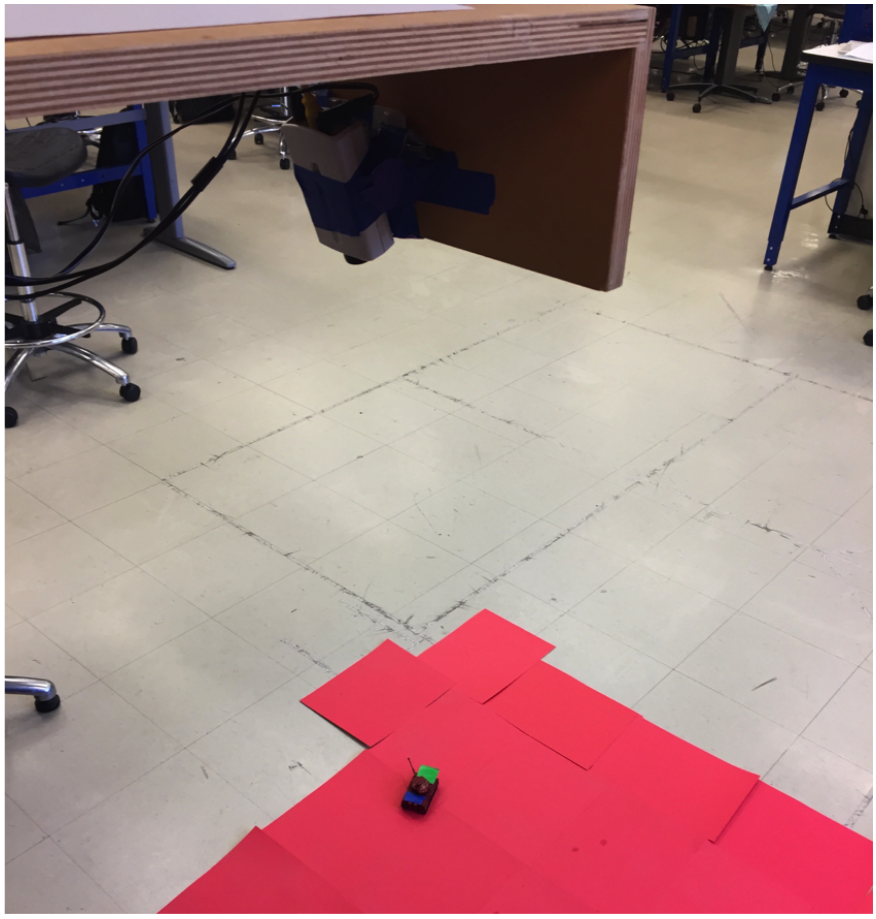


Design Decisions

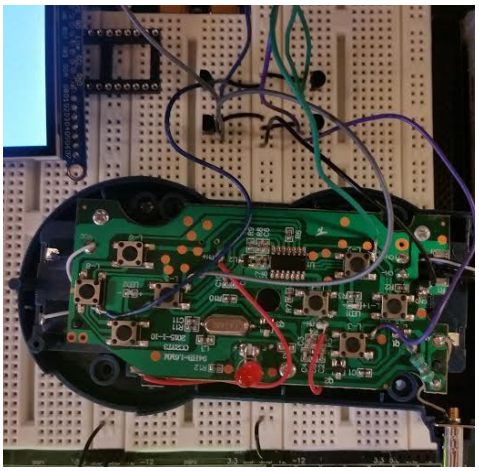
The camera is mounted approximately 4 feet and 8 inches above the ground, which gives the camera a field of vision that is 37 inches long and 46 inches wide. While writing the code for filtering and center of mass, we realized that the white linoleum floors of the lab were reflecting blue and green light from the fluorescent bulbs in the room. We also noticed additional lighting issues due to strange shadow patterns from a nearby window. To alleviate these issues, we covered the floor with matte red paper, so that no blue or green light would reflect, and the effects of strange shadows would be lessened.

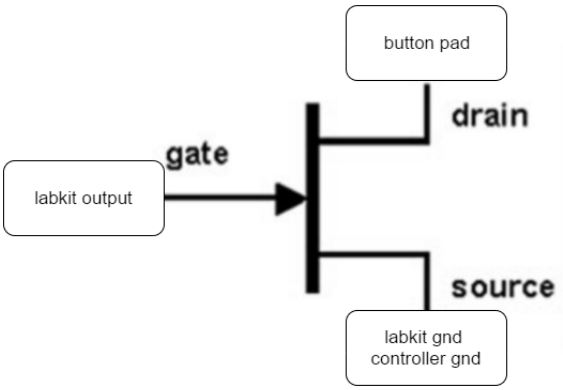
We chose to have two markers on the tank so that we could detect the direction of the tank. We chose to use green and blue as the markers, since RGB filtering showed us much more red noise in the room than green or blue noise. We also decided to filter in RGB as opposed to HSV, since RGB filtering seemed to create more dense images after filtering. To determine the values of the filter, we wrote a testing program that identifies the values of R, G, or B in a pixel and prints them on the labkit. Using this, we were able to effectively filter purely on RGB.

Our decision to use a tank was not our original plan. We originally purchased a small remote controlled car, but soon realized the importance of having a small turn radius and the ability to rotate locally. Since a tank can rotate locally, we decided instead to use a tank and have much more accurate path following. The basic physical set-up of the system is shown below:



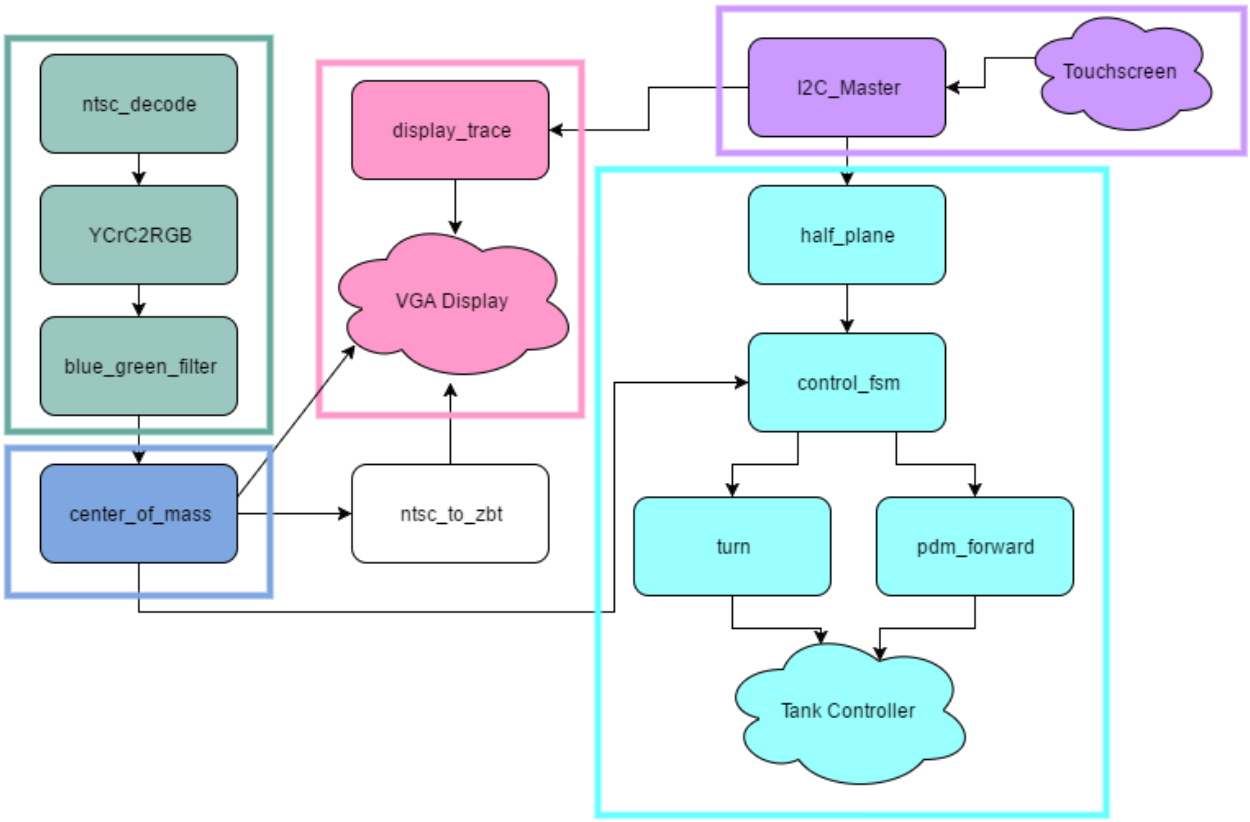
To control the tank from the labkit we used fets to act as the buttons on the remote controller. We tested the buttons and noticed that the buttons connected a certain pad to ground, which activated an action (right, left, forward, backward) depending on the button. Since the button was making connections to ground it made sense to use a NFET to control the remote controller. The labkit ground and the controller ground are tied together and the labkit output signals to control the gate of the mosfet to simulate a button press.





Module Overview

The project is divided into many modules that interact with each other to make a working product. The overall block diagram is shown in the figure below. The modules can be grouped into five main groups: color filtering, center of mass calculations, touch screen integration, vga display, and tank controls. An overview of the modules is provided in this section. A more detailed description of each module can be found in the Module Analysis section.



Color Filtering:

From the camera, we get a 30 bit YCrCb value, as well as f, v, h, (frame, vertical position, horizontal position) and data_valid signals. These YCrCb values are then converted into RGB in the YCrCb2RGB module, which outputs 24 bit values of RGB. Next, the blue_green_filter uses RGB thresholding to determine which pixels are the shades of green and blue that we have on our tank markers. For example, to threshold green, we check that the amount of green in a pixel is greater than all other colors, and greater than a certain threshold of green, which was obtained experimentally. This module also includes switch logic to set the RGB outputs of the filter to show a black background with green and blue markers, or simply pass through the RGB camera output.

Center of Mass:

This module finds the center of mass of the green and blue markers. Using the same thresholding as the filter, the center of mass determines which pixels should or should not be included in the calculations. The module keeps four running sums of the x and y coordinates of green pixels, and the x and y coordinates of blue pixels. The module also sums the number of pixels being counted. Four divider cores are implemented to carry out these divisions, which produce four center of mass values, x_g, y_g, x_b, and y_b.

Touch Screen Integration:

The touchscreen unit allows the user to input a path for the tank to follow. The touchscreen is a capacitive touchscreen unit from Adafruit that is 320 pixels by 240 pixels. The unit has an onboard touchscreen chip(FT6x06 CTPM) that processes finger inputs and stores information about the finger input, such as x, y position in its registers. The FT6x06 CTPM communicates through the I2C and external 4.7k resistors are used to pull up the SCL and SDA lines. The Labkit powers the touchscreen and it is used as an I2C master to communicate with FT6x06 CTPM (slave) to periodically poll it for the x,y position of the finger touch. Two of the labkit's in/out pins are connected to the FT6x06 CTPM's SCL and SDA lines.

VGA display:

The VGA display displays the feed from the camera, the center of mass markers, the current position of the finger touch, and the path drawn by the finger. The display has two settings, one where it displays the RGB feed from the camera, and one where it displays the filtered feed as well as the center of mass markers. The setting is controlled by switch 5. Both settings display the position of the last touch. The VGA display is set at 800x600 resolution and so the display to touchscreen ratio is 2:1. Every touchscreen x,y position is scaled by 2 before being displayed on the vga display.

Tank Controls:

The tank controls modules translate the x,y position information from the touchscreen and from the camera to physical tank movements. The tank control switches between three FSM states: idle, turn, and move forward. The tank is in the idle state when it is within range of the last registered finger touch, otherwise the tank will alternate between the turn and motion states until it arrives at the last registered finger touch. The turn state makes sure that the direction of the tank movement is lined up with the goal set by the finger touch. The output of the tank controls modules toggles four output pins on the Labkit that control the gate of four MOSFETs that then act as "buttons" on the tank controller to move the tank left, right, forward, and reverse.

Module Analysis

Camera Filtering

ntsc_decode

Overview:
This module's job is to convert the input signals from the camera, outputting a 30 bit YCrCb value. This module will also output f, v, and h, to communicate when the frame, vertical position, and horizontal position of a pixel has changed as the VGA screen is filled.

YCrCb2RGB

Overview:
This module converts a 30 bit YCrCb value to a 24 bit RGB value.

blue_green_filter

Overview:
This filter takes 24 bit RGB values from the camera, filters for green and blue, and outputs Rout, Gout, and Bout as modified RGB values. To filter RGB values, we check that the color of interest (ie green or blue) is greater than a set threshold, as well as all other RGB values in the pixel. For example, to filter for green, we check that a pixel has more green than any other color, and that green pigment is higher than a certain threshold between 0 and 255. Depending on the value of switch 5, this module either sets Rout, Gout, and Bout to show the black background with the filtered green and blue areas, or just passes the camera RGB image through the filter.
Implementation:
These thresholds were obtained experimentally through displaying RGB values of a pixel on the labkit display. We decided to filter in RGB rather than HSV because we saw more dense areas of pixels with RGB filtering, even though we also saw more noise. These dense blue and green filtered areas were important to calculating accurate center of mass values. The noise can be eliminated by using red paper as a background.

ntsc_to_zbt

Overview:
This module will take in the filtered RGB values from blue_green_filter to generate an address in ZBT memory. However, we only pass in the 6 most significant bits of each of the RGB values, so that we can store an 18 bit RGB value in memory. The ZBT memory can store 36 bits at each address, so we will store two RGB pixels at each address.
Implementation:
This module had to be modified from the original version to support RGB camera output.

Center of Mass

center_of_mass

Overview:
This module calculates the center of mass of the green and blue areas of the tank. It takes RGB values of each pixel as inputs, and outputs two (x,y) points. To calculate the center of mass, the module first determines whether the pixel is blue or green enough, using the same thresholding as the filter. Then, if a pixel is green or blue enough, the x and y values of that pixel are added to a running sum. After all pixels in the screen have been counted, the sum of x coordinate and y coordinate values are divided by the total number of green or blue pixels counted. We use four IP divider cores to carry out these divisions.

Implementation:

We did not want the dividers to constantly divide the running sum, as this would create inaccurate quotient outputs. In using the divider cores, we were careful to only update the numerator and denominator values when a full screen of pixels had been scanned, as signaled by frame_edge flagging high.

Touch Screen Integration

I2C_master

Overview:

The I2C_Master manipulates two in/out pins that act as the SDA and SCL lines of the I2C to read information from the FT6x06 CTPM's registers. This implementation of the labkit as an I2C master only supports reads from a slave. The project did not require any writes to the FT6x06 CTPM since we only needed to read the x and y position of the last finger touch. The module reads four consecutive 8 bit registers given a starting register address, and outputs the data to a 32 bit register. The x and y position is then extracted from the 32 bit number during post processing in the top level module.

Implementation:

The I2C master is a state machine that manipulates the SDA and SCL lines according to the I2C protocol for requesting information from the slave. The state machine is customizable for the number of consecutive registers to be read at once since each read is a separate state. The SCL line has to be toggled in sync with the SDA line and cannot always be left on because the SCL line of the I2C is only active during the period that the master is requesting information and when the slave is responding with the information. The SCL line is toggled at the beginning of each state except for the start and stop states to simplify the logic of the I2C module. The logic of toggling the SDA line is built off of the architecture that the SCL line is toggled at the beginning of every state.

200khz_clock

Overview:

This module generates the clock for the I2C Master. 200khz is halfway between the standard 100khz and the fast rate of 400khz. From testing 200khz works for our system.

Implementation:

A 27mhz clock is fed into the 200khz clock and a counter is used to count the number of 27mhz clock that has passed in order to determine the output of the 200khz clock. There are 135 27mhz cycles in one 200khz cycle.

VGA Display

Display_trace

Overview:

The display_trace module manages the pixel output to the VGA screen. The module draws the path that the user has traced and overlays it on top of the image from the camera. An assign statement determines what controls a single pixel on the screen. The trace that the player draws has priority over the image from the camera, so the white trace will always show on top of the screen and the pixel information from the camera is not displayed

Implementation:

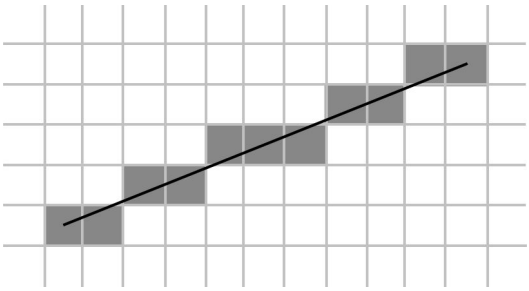
Crosshairs and camera image:

In center of mass mode, pixels that have the same x and y values as the blue and green center of masses are colored blue and green respectively. The camera image is constructed from vr_pixel. Vr_pixel is camera image that is in 6 bit RGB instead of the usual 8bit RGB, so each color representation in vr_pixel is padded with two zeros to make sure that all pixel values are 24 bits.

Displaying finger trace:

The state machine within the module implements the Bresenham drawing algorithm to draw the path that the user draws on the touch screen. The Bresenham drawing algorithm is shown in the figure. The

input to the algorithm are the coordinates of the start and ending point of the line straight, and it colors in the pixels between the two points. The state machine starts when the hcount and vcount reach the blanking period. There is a register that stores the previous finger touch during the last blanking period; the algorithm takes the coordinates of the previous finger touch and the current finger position as the start and end points. There is a block ram that is 1 bit wide and 480,000 lines deep (800 x 600), where each line represents a pixel. It stores a 1 if a pixel is part of the traced path, and a 0 otherwise. The block ram is constantly being read and only written to during the blanking period. If the clear button is engaged, a 0 is written to every line in the block ram.



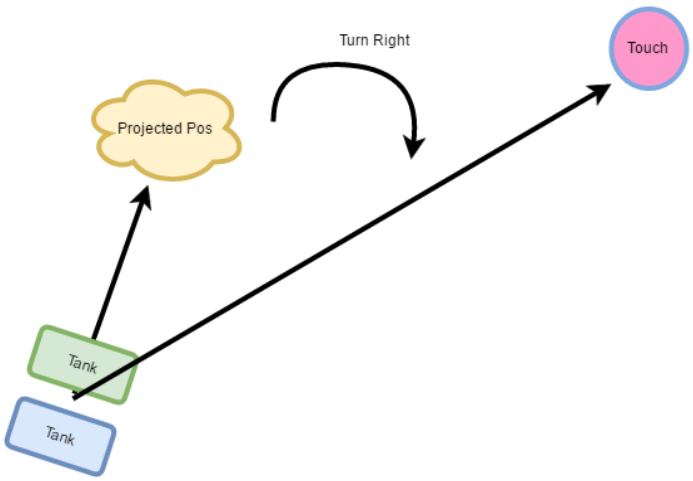
Displaying tank sprite:
A block ram is initialized with a coe file that has the image of a tank. The block ram is read when hcount and vcount is in range of the corresponding pixel on the VGA of where the last finger touch is. The tank sprite shows where the last finger touch is, which is synonymous with the goal of the tank.

Control FSM

half_plane

Overview:
The module takes in the current position of the tank, its projected position, and its goal, and determines if the tank should turn right or left to align itself towards the goal.

Implementation:
The module uses the point slope form of a line to generate an equation for a line defined by the current position of the tank and its goal. The x and y sides of the equation are stored in separate registers. The projected position of the tank is plugged into the equation, and the x and y sides of the equation is compared to determine if the tank should turn left or right.



turn

Overview:

The turn module toggles the output pin on the labkit and turns the tank in the smallest possible discrete steps when the "start" input to the module is high.

Implementation:

A counter determines the on and off times of the output labkit signal. The signal is high for 50 ms and low for 111ms to create discrete turning steps. We attempted to make the discrete turning steps smaller, but since the remote control was made for human finger presses, it was difficult to make the discrete turning steps any smaller. Each discrete turn is around 18 degrees.

pdm

Overview:

The module toggles the output pin on the labkit and moves the tank forward at a reduced speed when the "start" in put to the module is high.

Implementation:

A counter determines the on and off times of the output labkit signal. The signal is high for 50ms and off for 50 ms to allow the tank to move at a slower speed. The tank must be slowed down in order to avoid overshooting the target.

control_fsm

Overview:

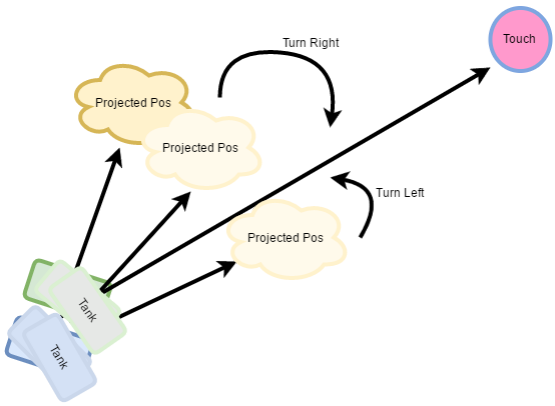
The module determines the motion of the tank, and the motion is determined by a FSM. The FSM has four states.

Implementation:

idle: The tank is in the idle state if its current position is within a 20 pixel square around the last finger state, and leaves the idle state to the half_plane_wait state if the current position is outside of the goal bounds. The tank does not move in the idle state.

half_plane_wait: It takes a few cycles for the half_plane module to calculate whether the tank should turn right or left based on the current position of the tank and its goal. To avoid the tank turning or moving forward because of invalid signals, the tank is held still while the half_plane_module finishes calculating whether the tank should turn right or left. After waiting for a certain number of cycles, it transitions to the rough_turn state. The tank does not move in the half_plane_wait state.

rough_turn: The tank turns in the direction determined by the half_plane module. The tank remains in this state until the turning direction changes, which indicates that the projected position of the tank has crossed the line that connects the tank position with the finger position. At this point the tank is facing its goal with a small error. When half_plane outputs a different direction than the previous timestep, it stops moving and moves to the motion state.



motion: The tank moves forward in this state and stays in the state for around 200ms before cycling back to the half_plane_wait state and then the rough_turn state to realign itself. The tank repositions its direction at a rate of 5HZ. This is to correct for over or under turning in the rough_turn state and the inherent physical fact that the tank does not move forward in a straight line. A recalculation rate of 5HZ was determined to be optimal after some testing.

In the beginning of every state the module checks if the tank position is within the goal window. If it is, the tank immediately enters the idle state and stops.

Future Work

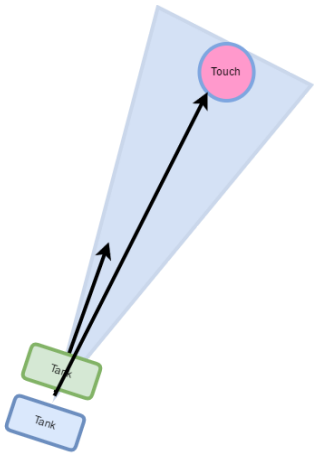
In the future, one goal is to add additional filtering methods to help alleviate the noise we saw reflected from the white linoleum tiles so that the system can function on a larger variety of surfaces. Additional filtering methods include creating a ring buffer to average the pixel values from the last few frames to eliminate random noise that shows up periodically. We could also implement regional filtering by adding a pixel to the center of mass if the pixels immediately around it are the same color as well.

In terms of controls we were limited by the tank controller. Since the tank controller was intended to be used on the timescale of finger presses and not on the timescale of digital logic, it was difficult to achieve fine control of the tank. For example, the shortest pulse can only turn the tank left or right about 18 degrees. We were limited by the controller. In order to implement a more sophisticated, accurate, and faster control algorithm, we can utilize the labkit to remotely control a microcontroller circuit that will directly tap into the H bridges of the the tank motors to finely control the motion of the tank. This will allow the tank to follow the path more closely.

Conclusion

The most challenging aspect of getting valid information from the NTSC camera was experimenting with different filtering methods, and seeing which created the most accurate center of mass values. We noticed a trade off between the density of the green and blue patches and the prevalence of noise . Filtering based on HSV values created less dense patches of blue and green, which made it harder to find accurate center of mass values. However, HSV filtering also reduced the prevalence of noise from the white linoleum tiles. Filtering with RGB produced much more dense patches of green and blue, but created more noise from the white light reflecting on the floor. This noise could be reduced by covering the floor with matte red paper.

Another challenge we faced was accurately controlling the tank. We discovered the tank controller had very coarse control of the tank, and had to work with that in mind to develop the control FSM. We experimented with different versions of the half-plane method, such as having a “cone” of acceptability (see below), where the tank would only move forward when it’s projected position lands in a certain window of acceptability. Although this took the tank to its destination it did not result in a straight path.



Since the turn granularity of of the tank was very coarse, we settled on a bang bang control to align the tank with it’s target and separating the movements of the tank into alignment and forward motion.

One lesson we both learned through doing this project was the efficiency of creating testing methods and modules rather than using guess and check methods. For example, to find accurate thresholding

values for the RGB filter, we started off by guessing and checking different threshold values between 0 and 255. However, this proved to be a very slow and inefficient method, as the code took about 15 minutes to compile each time we wanted to test a different value. A much more effective method was to create a module that used the labkit display to show the RGB value of the pixel at the intersection of a pair of cross-hairs. This allowed us to more quickly hone in on a range of accurate thresholding values. Another useful testing tool is ModelSim, which proved to be very useful in creating the I2C module. ModelSim was used to debug timing and state machine transition problems before the code was uploaded onto hardware. The rest of the module was debugged using the oscilloscope, but without the Modelsim simulations initially, the module would have taken much longer to create. ModelSim was also frequently used in creating the half_plane modules. We tested every single tank and finger position combination to make sure that the logic was correct before testing on the physical hardware.

To conclude this report, we would like to offer some advice to future students. We recommend creating test modules rather than wasting time with guessing and checking. Although creating testing modules seems like more work, it actually saves a lot of time. We also suggest that students try to front-load much of the work for the project, and get some big parts of the code written early in the semester. This will leave more time for working on extra bells and whistles towards the end, and will allow time to solve any unforeseen issues. Integration of different parts of the code often takes more time than expected. Lastly, we suggest that students who have physical components to their project purchase and test them as soon as possible. We purchased our tank very early, and noticed that it veered off to one direction when it was supposed to be driving in a straight line. Because we knew this early on, we could take this imperfection into account when developing our control logic.

Appendix

```
`timescale 1ns / 1ps
`default_nettype none
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:       Tianye Chen
//
// Create Date:    18:00:13 10/31/2016
// Design Name:
// Module Name:    I2C_master
//////////////////////////////////////////////////////////////////////////////////
module I2C_master(
  input wire clk,//system clock
  input wire reset,//active high reset
  input wire start, //initiate a data transfer
  input wire [6:0] slave_addr, //address of target slave 0x38 for touchscreen
  input wire [7:0] reg_addr, //register to read from or write to
  output reg [31:0] data_rd,
  output reg [7:0] state = 8'd0, //state of the I2C
  inout  wire sda, //serial data line of I2C bus ///if it is inout...it has to be a single wire..how to fix
that???
  inout wire scl //serial clock line of I2C bus
   );

  parameter STATE_IDLE = 8'd0;
  parameter STATE_DEVICE_ADDR1 = 8'd1;
  parameter STATE_ACK1 = 8'd2;
  parameter STATE_REG_ADDR = 8'd3;
  parameter STATE_ACK2 = 8'd4;
  parameter STATE_STOP1A = 8'd5;
  parameter STATE_STOP1B = 8'd6;
  parameter STATE_START2 = 8'd7;
  parameter STATE_DEVICE_ADDR2 = 8'd10;
  parameter STATE_ACK3 = 8'd11;
  parameter STATE_DATA1 = 8'd20;
  parameter STATE_DATA2 = 8'd21;
  parameter STATE_DATA3 = 8'd22;
  parameter STATE_DATA4 = 8'd23;
  parameter STATE_STOP2A = 8'd25;
  parameter STATE_STOP2B = 8'd26;
  parameter STATE_STOP2C = 8'd27;
```

```
reg sda_val = 1;
reg scl_val = 1;
assign sda = sda_val? 1'bz: 1'b0; //high Z unless pulled down
assign scl = scl_val? 1'bz: 1'b0; //high Z unless pulled down
reg[6:0] counter;
reg counter_done;

always @(posedge clk)
begin
  if (reset == 1) begin
    state <= STATE_IDLE;
  end
  else begin
    case (state)
    STATE_IDLE: begin
      if (start) begin
        state <= STATE_DEVICE_ADDR1;
        sda_val <= 0;
        counter <= 7'd7;
      end
      else begin
        sda_val <= 1;
        scl_val <= 1;
      end
    end
    STATE_DEVICE_ADDR1: begin
      scl_val <= ~scl_val;
      if (counter_done && scl_val)begin
        counter_done <= 0;
        sda_val <= 1;
        counter <= 7'd8;
        state <= STATE_ACK1;
      end
      else if (scl_val) begin
        if (counter == 0)begin
          counter_done <= 1;
          sda_val <= 0; //write
        end
        else begin
          sda_val <= slave_addr[counter-1];
          counter <= counter -1;
        end
      end
```

```
end
STATE_ACK1: begin
  scl_val <= ~scl_val;
  if (scl_val) begin
    if (sda) begin //no acknowledgement
      state <= STATE_IDLE;
    end
    else begin
      state <= STATE_REG_ADDR;
      sda_val <= reg_addr[counter-1]; //ms of register
      counter <= counter-1;
    end
  end
end
STATE_REG_ADDR: begin
  scl_val <= ~scl_val;

  if (scl_val) begin
    if (counter == 0)begin
      sda_val <= 1;
      state <= STATE_ACK2;
    end
    else begin
      sda_val <= reg_addr[counter-1];
      counter <= counter -1;
    end
  end
end
STATE_ACK2: begin
  scl_val <= ~scl_val;
  if (scl_val) begin
    if (sda) begin //no acknowledgement
      state <= STATE_IDLE;
    end
    else begin
      scl_val <= 0;
      sda_val <= 0;
      state <= STATE_STOP1A;
    end
  end
end
STATE_STOP1A: begin
  scl_val <= 1;
```

```
  state <= STATE_STOP1B;
end
STATE_STOP1B: begin
  sda_val <= 1;
  state <= STATE_START2;
end
STATE_START2: begin
  sda_val <= 0;
  counter <= 7'd7;
  state <= STATE_DEVICE_ADDR2;
end
STATE_DEVICE_ADDR2:begin
  scl_val <= ~scl_val;
  if (counter_done && scl_val)begin
    counter_done <= 0;
    sda_val <= 1; ///relinquish control to the slave
    counter <= 7'd31; ///set counter for data///
    state <= STATE_ACK3;
  end
  else if (scl_val) begin
    if (counter == 0)begin
      counter_done <= 1;
      sda_val <= 1; //read
    end
    else begin
      sda_val <= slave_addr[counter-1];
      counter <= counter -1;
    end
  end
end
STATE_ACK3: begin
  scl_val <= ~scl_val;
  if (scl_val) begin
    if (sda) begin //no acknowledgement
      state <= STATE_IDLE;
    end
    else begin
      state <= STATE_DATA1;
      sda_val <=1;
    end
  end
end
STATE_DATA1: begin //state 20
```

```
        scl_val <= ~scl_val;
        if (scl_val) begin
          if (counter == 7'd24)begin
            sda_val <= 0; ///ACKNOWLEDGE THE READ.....
            state <= STATE_DATA2;
            data_rd[counter]<=sda;
          end
          else begin
            sda_val <= 1;
            data_rd[counter] <= sda;
            counter <= counter -1;
          end
        end
      end

      STATE_DATA2: begin //21
        scl_val <= ~scl_val;
        if (scl_val) begin
          if (counter == 7'd24) begin  ///waiting for the ack cycle to be over before reading from
sda
            sda_val <=1;
            counter <= counter-1;
          end
          else if (counter == 7'd16)begin
            sda_val <= 0; ///ACKNOWLEDGE THE READ.....
            state <= STATE_DATA3;
            data_rd[counter]<=sda;
          end
          else begin
            sda_val <=1;
            data_rd[counter] <= sda;
            counter <= counter -1;
          end
        end

      end

      STATE_DATA3: begin //22
        scl_val <= ~scl_val;
        if (scl_val) begin
          if (counter == 7'd16)begin ///waiting for ack cycle to be over before reading from sda
            sda_val <= 1;
            counter <= counter-1;
```

```
      end
      else if (counter == 7'd8)begin
        sda_val <= 0; ///ACKNOWLEDGE THE READ.....
        state <= STATE_DATA4;
        data_rd[counter]<=sda;
      end
      else begin
        sda_val <=1;
        data_rd[counter]<= sda;
        counter <= counter -1;
      end
    end

end

STATE_DATA4: begin //23
    scl_val <= ~scl_val;
    if (scl_val) begin
      if (counter == 7'd8)begin
        sda_val <= 1;
        counter <= counter-1;
      end
      else if (counter == 7'd0)begin
        sda_val <= 1; ///NOT ACKNOWLEDGE THE READ.....
        state <= STATE_STOP2A;
        data_rd[counter]<=sda;
      end
      else begin
        sda_val <=1;
        data_rd[counter] <= sda;
        counter <= counter -1;
      end
    end
end
STATE_STOP2A: begin
    sda_val <= 0;
    state <= STATE_STOP2B;
end
STATE_STOP2B: begin
    scl_val <= 1;
    state <= STATE_STOP2C;
end
STATE_STOP2C: begin
```

```
          sda_val <= 1;
          state <= STATE_IDLE;
        end
      endcase
  end
end

endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:38:34 11/14/2016
// Design Name:
// Module Name:    display_trace_bram
//////////////////////////////////////////////////////////////////////////////////
module display_trace_bram(
  input vclock,  // 40MHz clock
  input reset,       // 1 to initialize module
  input [10:0] hcount,   // horizontal index of current pixel (0..799)
  input [9:0]  vcount, // vertical index of current pixel (0..599)
  input hsync,       // XVGA horizontal sync signal (active low)
  input vsync,       // XVGA vertical sync signal (active low)
  input blank,       // XVGA blanking (1 means output black pixel)
  output reg phsync,
  output reg pvsync,
  output reg pblank,
  input [11:0] data_x,
  input [11:0] data_y,
  input clear_button_pulse,
  output[23:0] pixel, // pong game's pixel  // r=23:16, g=15:8, b=7:0
  input [17:0] vr_pixel,
  input [1:0] switch,
  input switch5,
  input [10:0] x_g,
  input [9:0] y_g,
  input [10:0] x_b,
  input [9:0] y_b
   );

  reg [3:0] state = 4'b0;
  parameter idle = 4'd0;
  parameter data_buff_filter = 4'd1;
  parameter data_buff = 4'd2;
  parameter draw_setup = 4'd3;
  parameter draw = 4'd4;
  parameter clear_screen = 4'd5;

  wire read;
```

```
  wire dina;
  assign dina = (state == clear_screen)? 0:1; //if clear screen, write all zeros == black//
  wire [18:0] addra;
  wire [18:0] addrb;
  reg write_enab = 0;
  wire doutb;
  reg [18:0] write_addr;
  wire [18:0] read_addr;

  reg [18:0]addr_counter = 19'd480000;///counter for clearing the screen

  assign addra = write_enab?write_addr:0;

  assign read = (hcount <799 && vcount < 599)?1:0; ///read while on screen, otherwise do not
read
  assign addrb = read?(hcount + (vcount*800)):1; ///when reading, read the correct pixel from
memory

  parameter[10:0] screen_x_size = 11'd799;
  parameter[9:0] screen_y_size = 10'd599;
  parameter[10:0] tscreen_x_size = 11'd654;
  parameter[19:0] tscreen_y_size = 10'd490;
  parameter tscreen_origin_x = 10;
  parameter tscreen_origin_y = 10;

  parameter filter_threshold = 5;

  reg[11:0] data_x_new;
  reg[11:0] data_y_new;
  reg data_x_ok;
  reg data_y_ok;
  reg[11:0] data_x_buffer1;
  reg[11:0] data_y_buffer1;
  reg[11:0] data_x_buffer2;
  reg[11:0] data_y_buffer2;
  reg[11:0] working_x;
  reg[11:0] working_y;
  reg[11:0] numerator;
  reg[11:0] shortest;
  reg[11:0] longest;
  reg x_dir;
  reg y_dir;
  reg steep;
```

```
  reg [11:0] dx,dy,next_numerator; //temporary registers used for calculating Bresenham's
algorithm
  reg carry_over; //temperary register used for calculating Bresenham's algorithm

  /// colors! ///
  parameter[23:0] color_orange = 24'hFF9A34;
  parameter[23:0] color_lilac= 24'hE785EB;
  parameter[23:0] color_red = 24'hFF0000;
  parameter[23:0] color_white = 24'hFFFFFF;
  parameter[23:0] color_black = 24'h0;
  parameter[23:0] color_purple = 24'h9c0042;

  ///calculations for the tank blob///
  parameter tank_size = 4'd10;
  wire [11:0] data_x_tank;
  wire [11:0] data_y_tank;
  assign data_x_tank = tscreen_origin_x + 3*(data_x>>2);
  assign data_y_tank= tscreen_origin_y + tscreen_y_size - 3*(data_y>>2);
  wire [23:0] tank_pixel;
  reg [23:0] camera_pixel;


  reg [4:0] addra_tank = 0;
  wire [62:0]douta_tank;
  reg sprite;

  reg [10:0] hcount_buff;
  reg [10:0] vcount_buff;
  reg [5:0] data_counter;

  parameter sprite_width = 6'd63;
  parameter sprite_height = 5'd31;
  wire [11:0] sprite_startx;
  wire [11:0] sprite_endx;
  wire [11:0] sprite_starty;
  wire [11:0] sprite_endy;
  assign sprite_startx = data_x_buffer1 - (sprite_width>>1);
  assign sprite_endx = sprite_startx + sprite_width;
  assign sprite_starty = data_y_buffer1 - (sprite_height>>1);
  assign sprite_endy = sprite_starty + sprite_height;
  assign pixel = doutb?color_white:(
                sprite?color_purple:(
```

```
                (camera_pixel)));
 ///path takes priority, then sprite, then the camera image


 tank_sprite_bram tank_sprite(.clka(vclock),.addra(addra_tank), .douta(douta_tank));

 reg [11:0] data_x_sync;
 reg [11:0] data_y_sync;

 always @ (posedge vsync) begin
   data_x_sync <= data_x;
   data_y_sync <= data_y;
 end
 always @(posedge vclock) begin
   if (switch[0] && !switch5 && (hcount == x_g || vcount == y_g)) begin
     camera_pixel <= 24'b000000001111111100000000;
   end
   else if (switch[1] && !switch5 && (hcount == x_b || vcount == y_b)) begin
     camera_pixel <= 24'b000000000000000011111111;
   end
   else begin
     camera_pixel <= {vr_pixel[17:12],2'b0,vr_pixel[11:6],2'b0,vr_pixel[5:0],2'b0};
   end
   phsync <= hsync;
   pvsync <= vsync;
   pblank <= blank;

   hcount_buff <= hcount;
   vcount_buff <= vcount;
   if((vcount>= sprite_starty) && (vcount<=(sprite_endy))) begin
     if (vcount_buff != vcount) begin
       addra_tank <= addra_tank + 1;
     end
     if ((hcount >= sprite_startx) && (hcount <= (sprite_endx))) begin
       sprite <= douta_tank[sprite_width-data_counter];
       if (hcount_buff != hcount) begin
         data_counter <= data_counter +1;
       end
     end
   end
   else begin
     sprite <= 0;
//     if(vcount == (sprite_endy)) begin
```

```
//            addra_tank <= 0;
//       end
    end
    case (state)
      idle: begin
        if (clear_button_pulse) begin
          state <= clear_screen;
          write_enab <=1;
        end
        else if ((hcount > 800) && (vcount > 600)) begin
          state <= data_buff_filter;
        end
        else begin
          write_enab <= 0;
        end
      end
      data_buff_filter: begin
        data_x_new = tscreen_origin_x + 2*data_x_sync;
        data_y_new = tscreen_origin_y + tscreen_y_size - 2*data_y_sync;
        data_x_buffer1 <= data_x_buffer2;
        data_y_buffer1 <= data_y_buffer2;
        data_x_ok = 1;
        data_y_ok = 1;
        if (data_y_new == data_y_buffer2) begin
          if (data_x_new > data_x_buffer2)begin
            if ((data_x_new - data_x_buffer2) > filter_threshold) begin
              data_x_ok = 0;
            end
          else
            if ((data_x_buffer2 - data_x_new) > filter_threshold) begin
              data_x_ok = 0;
            end
          end
        end
        if (data_x_new == data_x_buffer2) begin
          if (data_y_new > data_y_buffer2)begin
            if ((data_y_new - data_y_buffer2) > filter_threshold) begin
              data_y_ok = 0;
            end
          else
            if ((data_y_buffer2 - data_y_new) > filter_threshold) begin
              data_y_ok = 0;
            end
```

```
      end
    end
    if (data_x_ok && data_y_ok) begin
      data_x_buffer2 <= data_x_new;
      data_y_buffer2 <= data_y_new;
      state <= draw_setup;
    end
    else begin
      state <= draw_setup;
    end
  end
  draw_setup: begin
    working_x <= data_x_buffer1; // working y = the y pixel value currently being drawn
    working_y <= data_y_buffer1; // working x = the x pixel value
    if (data_x_buffer2 > data_x_buffer1) begin
      dx = data_x_buffer2 - data_x_buffer1;
      x_dir <= 1;
    end
    else begin
      dx = data_x_buffer1 - data_x_buffer2;
      x_dir <= 0;
    end
    if (data_y_buffer2 > data_y_buffer1) begin
      dy = data_y_buffer2 - data_y_buffer1;
      y_dir <= 1;
    end
    else begin
      dy = data_y_buffer1 - data_y_buffer2;
      y_dir <= 0;
    end
    if (dx > dy) begin
      steep <= 0;
      longest <= dx;
      shortest <= dy;
      numerator <= dx >>1;
    end
    else begin
      steep <= 1;
      longest <= dy;
      shortest <= dx;
      numerator <= dy >>1;
    end
    state <= draw;
```

```
        write_enab<= 1;
      end
      draw: begin ///bresenham's drawing algorithm///
        if (working_y == data_y_buffer2 && working_x == data_x_buffer2) begin
          state <= idle;
        end
        else begin
          next_numerator = numerator + shortest;
          carry_over = next_numerator >= longest;
          numerator <= carry_over?next_numerator - longest:next_numerator;
          if (steep)begin
            working_y <= y_dir? working_y+1: working_y-1;
            if (carry_over) begin
              working_x <= x_dir? working_x+1: working_x-1;
            end
          end
          else begin
            working_x <= x_dir? working_x+1: working_x-1;
            if (carry_over) begin
              working_y <= y_dir? working_y+1: working_y-1;
            end
          end
        end
        write_addr <= (working_x) + (working_y*800);
      end
      ///write everything in the bram to be 0, so it is black///
      clear_screen: begin
        if (addr_counter > 0) begin
          write_addr <= addr_counter;
          addr_counter <= addr_counter -1;
        end
        else begin
          write_addr <= addr_counter;
          state <= idle;
          addr_counter <= 19'd480000;
        end
      end
    endcase
  end

  bram_dualport bram(.clka(vclock), .dina(dina), .addra(addra),
  .wea(write_enab), .clkb(vclock), .addrb(addrb), .doutb(doutb));
//
```

```
endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Tianye Chen
//
// Create Date:    13:42:37 11/27/2016
// Design Name:
// Module Name:    half_plane
//////////////////////////////////////////////////////////////////////////////////
module half_plane(
  input wire clock,
  input wire signed [11:0] test_x,
  input wire signed [11:0] test_y,
  input wire signed [11:0] start_x,
  input wire signed [11:0] start_y,
  input wire signed [11:0] end_x,
  input wire signed [11:0] end_y,
  output reg turn_right,
  output reg turn_left
   );

  ///normalizing everything onto the same coordinate system, (0,0) at right bottom of the camera
screen, height and width of camera screen///
  ///tank x values do not need to be transposed///
  reg [3:0] state = 0;
  parameter slope = 0;
  parameter equation =1;
  parameter equation1 =2;
  parameter comparison =3;

  reg signed [11:0] dx =0;
  reg signed [11:0] dy =0;

  reg signed [23:0] y_side =0;
  reg signed [23:0] x_side1 =0;
  reg signed [23:0] x_side =0;

///uses point slope form of lines to determine which half of the plane the point is in///
///depending on which half of the plane the point is in, the tank turns left or right to match the
direction of the finger touch///
  always@(posedge clock) begin
     case(state)
```

```
slope: begin
   dx <= end_x - start_x;
   dy <= end_y - start_y;
   state <= equation;
end
equation: begin
   y_side <= test_y *dx;
   x_side1 <= dy*(test_x -start_x);
   state <= equation1;
end
equation1: begin
   x_side <= x_side1+ (start_y*dx);
   state <= comparison;
end
comparison:begin
   if (dx >0 && dy > 0)begin
      if (y_side >= x_side) begin
         turn_left <= 0;
         turn_right <= 1;
      end
      else begin
         turn_left <= 1;
         turn_right <= 0;
      end
   end
   else if (dx <0 && dy <0) begin /////inequalities are flipped due to a negative sign
      if (y_side < x_side) begin
         turn_left <= 1;
         turn_right <= 0;
      end
      else begin
         turn_left <= 0;
         turn_right <= 1;
      end
   end
   else if (dx >0 && dy < 0)begin
      if (y_side <= x_side) begin
         turn_left <= 1;
         turn_right <= 0;
      end
      else begin
         turn_left <= 0;
         turn_right <= 1;
```

```
            end
          end
          else if (dx <0 && dy >0) begin
            if (y_side <= x_side) begin
              turn_left <= 1;
              turn_right <= 0;
            end
            else begin
              turn_left <= 0;
              turn_right <= 1;
            end
          end
          state <= slope;
        end
      endcase
  end
endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Tianye Chen
//
// Create Date:    22:13:23 11/21/2016
// Design Name:
// Module Name:    turn
//////////////////////////////////////////////////////////////////////////////////
module turn(
  input clock,
  input start,
  output reg signal,
  output reg done
   );
  parameter degree_18 = 1350000; ///least number of clock cycles for the tank to turn a minimal
amount///
  parameter wait_time = 3000000; ///least number of clock cycles it has to wait before
commanding another turn///
  reg [30:0] count;
  reg [35:0] wait_count;
  reg [3:0] state =0;
  parameter on= 4'd0;
  parameter off = 4'd1;
```

```
  always @(posedge clock) begin
    if(start) begin
      case (state)
        on: begin
          signal <= 1;
          if (count < degree_18 -1) begin
            count <= count +1 ;
          end
          else if (count == degree_18 -1) begin
            count <= 0;
            state <= off;
          end
        end
        off: begin
          signal <= 0;
          if (wait_count < wait_time -1) begin
            wait_count <= wait_count +1;
          end
          else begin
            wait_count <= 0;
            state <= on;
          end
        end
      endcase
    end
    else begin
      signal <= 0;
    end
  end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:      Tianye Chen
//
// Create Date:    20:34:22 11/21/2016
// Design Name:
// Module Name:    pdm
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps
`default_nettype none
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:02:07 11/20/2016
// Design Name:
// Module Name:    pwm
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module pdm(
  input clock,
```

```
  input start,
  input [3:0] off_time,
  output reg signal,
  output reg done
   );

  reg [3:0] state;
// assign state_debug = state;
  parameter on  = 4'd0;
  parameter off = 4'd1;

  parameter off_cycle_half = 1350000 >>1;
  parameter off_cycle_0_0 = 0;
  parameter off_cycle_1_0 = 1350000;
  parameter off_cycle_1_5 = off_cycle_1_0+off_cycle_half;
  parameter off_cycle_2_0 = 1350000*2;
  parameter off_cycle_2_5 = off_cycle_2_0+off_cycle_half;
  parameter off_cycle_3_0 = 1350000*3;
  parameter off_cycle_3_5 = off_cycle_3_0+off_cycle_half;
  parameter off_cycle_4_0 = 1350000*4;


  parameter on_cycles = 1350000; ///1/2 of 10 herz///
  reg [30:0] off_cycles;
  reg [7:0] cycle_counter = 0;
  reg [21:0] on_counter = 0;
  reg [30:0] off_counter = 0;

  always @(*) begin
    case (off_time)
      0: begin
        off_cycles <= off_cycle_0_0;
      end
      1: begin
        off_cycles <= off_cycle_1_0;
      end
      2: begin
        off_cycles <= off_cycle_1_5;
      end
      3: begin
        off_cycles <= off_cycle_2_0;
      end
      4: begin
```

```
        off_cycles <= off_cycle_2_5;
      end
      5: begin
        off_cycles <= off_cycle_3_0;
      end
      6: begin
        off_cycles <= off_cycle_3_5;
      end
      7: begin
        off_cycles <= off_cycle_4_0;
      end
    endcase
end

always @ (posedge clock) begin
  if (start) begin
    case(state)
      on: begin
        signal <= 1;
        if (on_counter < on_cycles-1)begin
          on_counter <= on_counter +1;
        end
        else if (off_cycles == 0) begin
          on_counter <= 0;
          state <= on;
        end
        else begin
          on_counter <= 0;
          state <= off;
        end
      end
      off: begin
        signal <= 0;
        if (off_counter < off_cycles-1)begin
          off_counter <= off_counter +1;
        end
        else begin
          off_counter <= 0;
          state <= on;
        end
      end
    endcase
  end
```

```
    else begin
      signal <= 0;
      state <= on;
    end
  end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Tianye Chen
//
// Create Date:    17:46:23 11/22/2016
// Design Name:
// Module Name:    control_FSM
//////////////////////////////////////////////////////////////////////////////////
module control_FSM(
  input clock,
  input drive,
  input [10:0]tank_front_x,
  input [9:0]tank_front_y,
  input [10:0]tank_back_x,
  input [9:0]tank_back_y,
  input [11:0]touch_x_raw,
  input [11:0]touch_y_raw,
  output reg forward,
  output reg reverse,
  output reg right,
  output reg left,
  output wire [3:0] debug_state,
  output reg same_goal = 1
   );
  reg [3:0]state = 0;
  assign debug_state = state;
  parameter idle = 4'd0;
  parameter half_plane_wait = 4'd1;
  parameter rough_turn = 4'd2;
  parameter motion = 4'd3;

  ///normalizing everything onto the same coordinate system, (0,0) at right bottom of the camera
screen, height and width of camera screen///
  parameter camera_ratio = 2;
  parameter camera_size_x = 654; //number of pixels
  parameter camera_size_y = 490; //number of pixels
  wire [11:0] touch_x;
  assign touch_x = touch_x_raw << 1;
  wire [11:0] touch_y;
  assign touch_y = touch_y_raw << 1;
  wire [11:0] tank_front_y_transposed;
  assign tank_front_y_transposed = camera_size_y - tank_front_y;
```

```
wire [11:0] tank_back_y_transposed;
assign tank_back_y_transposed = camera_size_y - tank_back_y;

///tank x values do not need to be transposed///
wire signed[11:0] tank_x;
assign tank_x = (tank_front_x>>1) + (tank_back_x>>1); ///averaging things
wire signed[11:0] tank_y;
assign tank_y = (tank_front_y_transposed>>1) + (tank_back_y_transposed>>1);
wire signed [11:0] tank_dx;
assign tank_dx = tank_front_x -tank_x;
wire signed [11:0] tank_dy;
assign tank_dy = tank_front_y_transposed - tank_y;

///projected position of the tank///
wire signed [11:0] proj_x;
assign proj_x = tank_x + (tank_dx<<3);
wire signed [11:0] proj_y;
assign proj_y = tank_y + (tank_dy<<3);

wire signed [12:0] dx = touch_x - tank_x;
wire signed [12:0] dy = touch_y - tank_y;

///waiting until halfplane is finished calculating turn right or turn left///
parameter wait_count = 5'd15;
parameter done_threshold = 20;
reg signed[11:0] prev_touch_x;
reg signed[11:0] prev_touch_y;
wire signed [11:0] touch_x_upper = prev_touch_x +done_threshold;
wire signed [11:0] touch_x_lower = prev_touch_x -done_threshold;
wire signed [11:0] touch_y_upper = prev_touch_y +done_threshold;
wire signed [11:0] touch_y_lower = prev_touch_y -done_threshold;
wire turn_right3;
wire turn_left3;
wire [24:0] pulse_divisor = 25'd1350000;
wire sample_pulse;
reg signed [11:0] cone_point1_x;
reg signed [11:0] cone_point1_y;
reg signed [11:0] cone_point2_x;
reg signed [11:0] cone_point2_y;
reg signed [11:0] cone_start1_x;
reg signed [11:0] cone_start1_y;
reg signed [11:0] cone_start2_x;
reg signed [11:0] cone_start2_y;
```

```
  reg [4:0] wait_counter = 0;
  reg done;
  reg prev_right = 0;
  reg prev_left = 0;
  ///clock cycles to stay in "on" before recalculating position///
  parameter forward_count = 5700000; ///5hz
  reg [30:0] forward_counter = 0;

  half_plane turn_half_plane(.clock(clock), .test_x(proj_x), .test_y(proj_y),
.start_x(tank_x),.start_y(tank_y),
                             .end_x(touch_x), .end_y(touch_y),
.turn_right(turn_right3),.turn_left(turn_left3));
  reg dx_match, dy_match;

  always @(posedge clock) begin
    prev_touch_x <= touch_x; //store the last touch
    prev_touch_y <= touch_y;
    ///check if tank is at the target///
    if (tank_x < touch_x_lower || tank_x > touch_x_upper
        || tank_y < touch_y_lower || tank_y > touch_y_upper) begin
        done <= 0;
    end
    else begin
        done <= 1;
    end
    ///if new touch is greater than the threshold, it will register as the new target///
    if (touch_x < touch_x_lower || touch_x > touch_x_upper
        || touch_y < touch_y_lower || touch_y > touch_y_upper) begin
        same_goal <= 0;
    end
    else begin
        same_goal <= 1;
    end
  end

  always @(posedge clock) begin
    case (state)
      idle: begin
        if(done) begin
          forward <= 0;
          right <=0;
          left <=0;
          state <= idle;
```

```
    end
    else begin
      forward <= 0;
      right <=0;
      left <=0;
      state <= half_plane_wait;
    end
  end
  half_plane_wait: begin
    if (done == 0) begin
      if (wait_counter < wait_count) begin
        wait_counter <= wait_counter +1;
      end
      else begin
        wait_counter <= 0;
        state <= rough_turn;
      end
    end
    else begin
      forward <= 0;
      right <=0;
      left <=0;
      state <= idle;
    end
  end
  rough_turn: begin
    if (done == 0) begin
      prev_right <= turn_right3;
      prev_left <= turn_left3;
      if (prev_right || prev_left) begin
        if ((turn_right3 != prev_right) || (turn_left3 != prev_left)) begin
          state <= motion;
        end
        else begin
          forward <=0;
          right <= turn_right3;
          left <= turn_left3;
        end
      end
      else begin
        state <= rough_turn;
      end
    end
```

```
        else begin
          forward <= 0;
          right <=0;
          left <=0;
          state <= idle;
        end
      end
      motion: begin
        if (done == 0) begin
          if (same_goal && (forward_counter < forward_count))begin
            forward_counter <= forward_counter +1;
            forward <=1;
            right <= 0;
            left <= 0;
          end
          else begin
            forward <=0;
            right <=0;
            left <=0;
            prev_right <= 0;
            prev_left <= 0;
            forward_counter <=0;
            state <= rough_turn;
          end
        end
        else begin
          forward <= 0;
          right <=0;
          state <= idle;
        end
      end
    endcase
  end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:     Natalie Mionis
//
// Create Date:    19:45:11 11/20/2016
// Design Name:
// Module Name:    center_of_mass2
```

```
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//     center_of_mass2 center(.vclk(tv_in_line_clock1), fvh, dv, Rout, Gout, Bout, x_g, y_g, x_b,
y_b);
//////////////////////////////////////////////////////////////////////////////////
module center_of_mass(vclk, fvh, dv, R, G, B, x_g, y_g, x_b, y_b); //was Rout, Gout, Bout

  input    vclk; // video clock from camera
  input [2:0]      fvh;
  input    dv;
  input [7:0] R;
  input [7:0] G;
  input [7:0] B;
  output reg [10:0] x_g;
  output reg [9:0] y_g;
  output reg [10:0] x_b;
  output reg [9:0] y_b;


  //green = front
  reg [31:0] x_increment_front = 0; //the x incrementor for green
  reg [31:0] y_increment_front = 0; //the y incrementor for green
  reg [31:0] count_front = 0; //the count of pixels that pass the filtering for green
  reg [31:0] x_dividend_front = 0; //the x dividend that goes into a divider for green
  wire [31:0] x_q_front; //the x quotient out of the divider for green
  wire [31:0] x_r_front; //the x remainder out of the divider for green
  wire x_rfd_front; //the ready signal from the divider
  reg [31:0] y_dividend_front = 0; //the y dividend is the numerator
  wire [31:0] y_q_front; //the answer for y in the green
  wire [31:0] y_r_front; //the y remainder out of the divider for green
  wire y_rfd_front; //the ready signal from the divider
  reg [31:0] divisor_front = 0;//the divisor for the green

  //blue = back
  reg [31:0] x_increment_back = 0; //the x incrementor for blue
```

```
 reg [31:0] y_increment_back = 0; //the y incrementor for blue
 reg [31:0] count_back = 0; //the count of pixels that pass the filtering for blue
 reg [31:0] x_dividend_back = 0; //the x dividend that goes into a divider for blue
 wire [31:0] x_q_back; //the x quotient out of the divider for blue
 wire [31:0] x_r_back; //the x remainder out of the divider for blue
 wire x_rfd_back; //the ready signal from the divider
 reg [31:0] y_dividend_back = 0; //the y dividend is the count (denominator)
 wire [31:0] y_q_back; //the y quotient out of the divider for blue
 wire [31:0] y_r_back; //the y remainder out of the divider for blue
 wire y_rfd_back; //the ready signal from the divider
 reg [31:0] divisor_back = 0; //the divisor for the blue

 parameter   COL_START = 11'd0;
 parameter    ROW_START = 10'd0;


 reg [10:0]  col = 0;
 reg [9:0]    row = 0;
 reg B_threshold = 130;
 reg G_threshold = 150;

 reg valid_condition;
 wire blue_condition;
 wire green_condition;
 assign blue_condition = (B>G && B>R && B>B_threshold);
 assign green_condition = (G>B && G>R && G>G_threshold);


 reg         old_frame;    // frames are even / odd interlaced

 wire     frame = fvh[2];
 wire     frame_edge = frame & ~old_frame;
 reg [10:0] diff_x = 0;
 reg [9:0] diff_y = 0;
 always @ (posedge vclk) //LLC1 is reference
   begin
 old_frame <= frame;

 //only want to check pixels in this range
 valid_condition <= ((col > 10) && (col < 664) && (row*2 > 10) && (row*2 < 500) && !fvh[2] &&
!fvh[1] && dv);
```

```
 if (!fvh[2])
   begin
     col <= fvh[0] ? COL_START :
       (!fvh[2] && !fvh[1] && dv && (col < 800)) ? col + 1 : col;
     row <= fvh[1] ? ROW_START :
       (!fvh[2] && fvh[0] && (row < 600)) ? row + 1 : row;
     //vdata <= (dv && !fvh[2]) ? din : vdata;
   end
    //if a pixel is either green enough or blue enough, include it in center of mass calculation
    //sum the x and y values of green and blue pixels, and divide by total pixels counted to get
the center of mass.
   x_increment_back <= frame_edge ? 0 : (valid_condition && (blue_condition)) ?
x_increment_back + col : x_increment_back;
   y_increment_back <= frame_edge ? 0 : (valid_condition && (blue_condition)) ?
y_increment_back + (row*2) : y_increment_back;
   count_back <= frame_edge ? 1 : (valid_condition && (blue_condition)) ? (count_back + 1) :
count_back;

   x_increment_front <= frame_edge ? 0 : (valid_condition && (green_condition)) ?
x_increment_front + col : x_increment_front;
   y_increment_front <= frame_edge ? 0 : (valid_condition && (green_condition)) ?
y_increment_front + (row*2) : y_increment_front;
   count_front <= frame_edge ? 1 : (valid_condition && (green_condition)) ? (count_front + 1) :
count_front;

   x_dividend_front <= frame_edge ? x_increment_front : x_dividend_front;
   y_dividend_front <= frame_edge ? y_increment_front : y_dividend_front;
   divisor_front <= frame_edge ? count_front : divisor_front;

   x_dividend_back <= frame_edge ? x_increment_back : x_dividend_back;
   y_dividend_back <= frame_edge ? y_increment_back : y_dividend_back;
   divisor_back <= frame_edge ? count_back : divisor_back;

 end

 always @ (posedge vclk) begin
 //when ready signals asserts, assign corresponding values to outputs
    x_g <= x_rfd_front? x_q_front[10:0] : x_g;
    y_g <= y_rfd_front? y_q_front[9:0] : y_g;
    x_b <= x_rfd_back? x_q_back[10:0] : x_b;
    y_b <= y_rfd_back? y_q_back[9:0] : y_b;
 end
```

```
//instantiate a divider for x coordinate of the green
center_divider x_divide_front(
.clk(vclk),
.dividend(x_dividend_front),
.divisor(divisor_front),
.quot(x_q_front),
.remd(x_r_front),
.rfd(x_rfd_front)
);
//instantiate a divider for y coordinate of the green
center_divider y_divide_front(
.clk(vclk),
.dividend(y_dividend_front),
.divisor(divisor_front),
.quot(y_q_front),
.remd(y_r_front),
.rfd(y_rfd_front)
);

//instantiate a divider for x coordinate of the blue
center_divider x_divide_back(
.clk(vclk),
.dividend(x_dividend_back), //numerator
.divisor(divisor_back), //denominator
.quot(x_q_back), //answer
.remd(x_r_back),
.rfd(x_rfd_back)
);
//instantiate a divider for y coordinate of the blue
center_divider y_divide_back(
.clk(vclk),
.dividend(y_dividend_back),
.divisor(divisor_back),
.quot(y_q_back),
.remd(y_r_back),
.rfd(y_rfd_back)
);
```

```
endmodule
```

```
module YCrCb2RGB (fvh, dv, R, G, B, clk, rst, Y, Cr, Cb, fvh_delayed, dv_delayed );

output reg [7:0]  R, G, B;

input clk,rst;
input[9:0] Y, Cr, Cb;

//wire [7:0] R,G,B;
reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
reg [9:0] const1,const2,const3,const4,const5;
reg[9:0] Y_reg, Cr_reg, Cb_reg;

input [2:0] fvh;
input dv;
reg [2:0] fvh_d1;
reg [2:0] fvh_d2;
reg [2:0] fvh_d3;
reg [2:0] fvh_d4;
output reg [2:0] fvh_delayed;
reg dv_d1;
reg dv_d2;
reg dv_d3;
reg dv_d4;
output reg dv_delayed;

always @ (posedge clk) begin //delaying by 5 clock cycles
  fvh_d1 <= fvh;
  fvh_d2 <= fvh_d1;
  fvh_d3 <= fvh_d2;
  fvh_d4 <= fvh_d3;
  fvh_delayed <= fvh_d4;
  dv_d1 <= dv;
  dv_d2 <= dv_d1;
  dv_d3 <= dv_d2;
  dv_d4 <= dv_d3;
  dv_delayed <= dv_d4;
  R <= (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
  G <= (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
  B <= (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;
end

//registering constants
always @ (posedge clk)
```

```
begin
const1 = 10'b 0100101010; //1.164 = 01.00101010
const2 = 10'b 0110011000; //1.596 = 01.10011000
const3 = 10'b 0011010000; //0.813 = 00.11010000
const4 = 10'b 0001100100; //0.392 = 00.01100100
const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
  if (rst)
     begin
     Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
     end
  else
     begin
    Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
     end

always @ (posedge clk or posedge rst)
  if (rst)
     begin
      A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
     end
  else
    begin
    X_int <= (const1 * (Y_reg - 'd64)) ;
    A_int <= (const2 * (Cr_reg - 'd512));
    B1_int <= (const3 * (Cr_reg - 'd512));
    B2_int <= (const4 * (Cb_reg - 'd512));
    C_int <= (const5 * (Cb_reg - 'd512));
    end

always @ (posedge clk or posedge rst)
  if (rst)
     begin
      R_int <= 0; G_int <= 0; B_int <= 0;
     end
  else
    begin
    R_int <= X_int + A_int;
    G_int <= X_int - B1_int - B2_int;
    B_int <= X_int + C_int;
    end
```

```
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:36:58 11/13/2016
// Design Name:
// Module Name:    xvga
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module xvga(vclock,hcount,vcount,hsync,vsync,blank);
  input vclock;
  output [10:0] hcount;
  output [9:0] vcount;
  output  vsync;
  output  hsync;
  output  blank;

  reg       hsync,vsync,hblank,vblank,blank;
  reg [10:0]   hcount;    // pixel number on current line
  reg [9:0] vcount;    // line number

  // horizontal: 1056 pixels total
  // display 800 pixels per line
  wire      hsyncon,hsyncoff,hreset,hblankon;
  assign    hblankon = (hcount == 799);
  assign    hsyncon = (hcount == 839);
  assign    hsyncoff = (hcount == 967);
```

```
  assign    hreset = (hcount == 1055);

  // vertical: 628 lines total
  // display 600 lines
  wire      vsyncon,vsyncoff,vreset,vblankon;
  assign    vblankon = hreset & (vcount == 599);
  assign    vsyncon = hreset & (vcount == 600);
  assign    vsyncoff = hreset & (vcount == 604);
  assign    vreset = hreset & (vcount == 627);

  // 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 @(posedge vclock) begin
    hcount <= hreset ? 0 : hcount + 1;
    hblank <= next_hblank;
    hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

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

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

module vram_display(reset,clk,hcount,vcount,vr_pixel,
        vram_addr,vram_read_data);
input [9:0]     vcount;
  output [17:0] vr_pixel;
  output [18:0] vram_addr;
  input [35:0]  vram_read_data;

  //forecast hcount & vcount 8 clock cycles ahead to get data from ZBT
  wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount + 8);
  input reset, clk;
  input [10:0] hcount;

  wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount + 1) : vcount;

  //wire [18:0]    vram_addr = {1'b0, vcount_f, hcount_f[9:2]};
  wire [18:0]     vram_addr = {vcount_f, hcount_f[9:1]};
```

```
 //wire [1:0]     hc4 = hcount[1:0];
 reg [17:0]   vr_pixel;
 reg [35:0]   vr_data_latched;
 reg [35:0]   last_vr_data;

 always @(posedge clk)
   last_vr_data <= (hcount[0]==1'b1) ? vr_data_latched : last_vr_data;

 always @(posedge clk)
   vr_data_latched <= (hcount[0]==1'b0) ? vram_read_data : vr_data_latched;

 always @(*)    // each 36-bit word from RAM is decoded to 4 bytes
   case (hcount[0])
      1'd1: vr_pixel = last_vr_data[17:0]; //{last_vr_data[17:12], 2'd0, last_vr_data[11:6], 2'd0,
last_vr_data[5:0], 2'd0};
      1'd0: vr_pixel = last_vr_data[35:18]; //{last_vr_data[18+17:18+12], 2'd0,
last_vr_data[18+11:18+6], 2'd0, last_vr_data[18+5:18+0],2'd0};
   endcase

endmodule // vram_display
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:     Natalie Mionis
//
// Create Date:    16:30:07 11/15/2016
// Design Name:
// Module Name:    reg_green_filter
//
//////////////////////////////////////////////////////////////////////////////////
module blue_green_filter(clock, switch, reset, fvh, dv, R, G, B, Rout, Gout, Bout);
   input wire clock;
   input wire switch;
   input wire reset;
   input [2:0] fvh;
   input dv;
   input wire [7:0] R;
   input wire [7:0] G;
   input wire [7:0] B;
   output reg [7:0] Rout;
   output reg [7:0] Gout;
   output reg [7:0] Bout;

   reg G_threshold = 150;
   reg B_high = 240;
   reg B_threshold = 130;


  parameter   COL_START = 11'd0;
  parameter    ROW_START = 10'd0;


  reg [10:0]  col = 0;
  reg [9:0]    row = 0;


         row <= fvh[1] ? ROW_START :
           (!fvh[2] && fvh[0] && (row < 600)) ? row + 1 : row;
         //vdata <= (dv && !fvh[2]) ? din : vdata;
     end
     //filtering green in RGB
     if (G>B && G>R && G>G_threshold && !switch) begin
          Rout <= 0;
```

```
        Gout <= 255;
        Bout <= 0;
    end
    //filtering blue in RGB
    else if (B>G && B>R && B>B_threshold && !switch) begin
      Rout <= 0;
      Gout <= 0;
      Bout <= 255;
    end
    //make the exterior red
    else if ((row*2 < 10) || (row*2 > 500) || (col < 10) || (col > 664)) begin
      Rout <= 255;
      Gout <= 0;
      Bout <= 0;
    end
    //display the camera view if switch on
    else if (switch) begin  reg         old_frame;    // frames are even / odd interlaced

wire     frame = fvh[2];
wire     frame_edge = frame & ~old_frame;

always @ (posedge clock) //LLC1 is reference
  begin
old_frame <= frame;

    if (!fvh[2]) begin
        col <= fvh[0] ? COL_START :
          (!fvh[2] && !fvh[1] && dv && (col < 800)) ? col + 1 : col;

      Rout <= R;
      Gout <= G;
      Bout <= B;
    end
    //make everything else black if switch off
    else begin
      Rout <= 0;
      Gout <= 0;
      Bout <= 0;
    end
end
```

Endmodule

```
//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09  // gph mod 11/3/2011
//
//
// Bug due to memory management will be fixed. It happens because
// the memory addressing protocol is off between ntsc2zbt.v and
// vram_display.v. There are 2 solutions:
// -. Fix the memory addressing in this module (neat addressing protocol)
//    and do memory forecast in vram_display module.
// -. Do nothing in this module and do memory forecast in vram_display
//    module (different forecast count) while cutting off reading from
//    address(0,0,0).
//
// Bug in this module causes 4 pixel on the rightmost side of the camera
// to be stored in the address that belongs to the leftmost side of the
// screen.
//
// In this example, the second method is used. NOTICE will be provided
// on the crucial source of the bug.
//
//////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we, sw);

  input    clk;  // system clock
  input    vclk; // video clock from camera
  input [2:0]      fvh;
  input    dv;
  input [17:0]     din;
```

```
output [18:0] ntsc_addr;
output [35:0] ntsc_data;
output   ntsc_we;  // write enable for NTSC data
input    sw;      // switch which determines mode (for debugging)

parameter    COL_START = 10'd0;//40;
parameter    ROW_START = 10'd0;//18;

// here put the luminance data from the ntsc decoder into the ram
// this is for 1024 * 788 XGA display

reg [9:0]    col = 0;
reg [9:0]    row = 0;
reg [17:0]   vdata = 0;
reg          vwe;
reg          old_dv;
reg          old_frame;    // frames are even / odd interlaced
reg          even_odd; // decode interlaced frame to this wire

wire     frame = fvh[2];
wire     frame_edge = frame & ~old_frame;

always @ (posedge vclk) //LLC1 is reference
  begin
old_dv <= dv;
vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it
old_frame <= frame;
even_odd = frame_edge ? ~even_odd : even_odd;

if (!fvh[2])
  begin
    col <= fvh[0] ? COL_START :
      (!fvh[2] && !fvh[1] && dv && (col < 800)) ? col + 1 : col;
    row <= fvh[1] ? ROW_START :
      (!fvh[2] && fvh[0] && (row < 600)) ? row + 1 : row;
    vdata <= (dv && !fvh[2]) ? din : vdata;
  end
  end

// synchronize with system clock

reg [9:0] x[1:0],y[1:0];
reg [17:0] data[1:0]; //was 29:0
```

```
reg       we[1:0];
reg         eo[1:0];

always @(posedge clk)
  begin
{x[1],x[0]} <= {x[0],col};
{y[1],y[0]} <= {y[0],row};
{data[1],data[0]} <= {data[0],vdata};
{we[1],we[0]} <= {we[0],vwe};
{eo[1],eo[0]} <= {eo[0],even_odd};
  end

// edge detection on write enable signal

reg old_we;
wire we_edge = we[1] & ~old_we;
always @(posedge clk) old_we <= we[1];


reg [35:0] mydata;
always @(posedge clk)
  if (we_edge)
    mydata <=  {mydata[17:0], data[1]};

// NOTICE : Here we have put 4 pixel delay on mydata. For example, when:
// (x[1], y[1]) = (60, 80) and eo[1] = 0, then:
// mydata[31:0] = ( pixel(56,160), pixel(57,160), pixel(58,160), pixel(59,160) )
// This is the root of the original addressing bug.


// NOTICE : Notice that we have decided to store mydata, which
//          contains pixel(56,160) to pixel(59,160) in address
//          (0, 160 (10 bits), 60 >> 2 = 15 (8 bits)).
//
//          This protocol is dangerous, because it means
//          pixel(0,0) to pixel(3,0) is NOT stored in address
//          (0, 0 (10 bits), 0 (8 bits)) but is rather stored
//          in address (0, 0 (10 bits), 4 >> 2 = 1 (8 bits)). This
//          calculation ignores COL_START & ROW_START.
//
//          4 pixels from the right side of the camera input will
//          be stored in address corresponding to x = 0.
//
```

```
//         To fix, delay col & row by 4 clock cycles.
//         Delay other signals as well.

reg [19:0] x_delay;
reg [19:0] y_delay;
reg [1:0] we_delay;
reg [1:0] eo_delay;

always @ (posedge clk)
begin
  x_delay <= {x_delay[9:0], x[1]};
  y_delay <= {y_delay[9:0], y[1]};
  we_delay <= {we_delay[0], we[1]};
  eo_delay <= {eo_delay[0], eo[1]};
end

// compute address to store data in
wire [8:0] y_addr = y_delay[18:10];
wire [9:0] x_addr = x_delay[19:10];


wire [18:0] myaddr = {y_addr, eo_delay[1], x_addr[9:1]};

// Now address (0,0,0) contains pixel data(0,0) etc.
//wire [18:0] myaddr_delay;

// alternate (256x192) image data and address
wire [31:0] mydata2 = {data[1][17:10],data[1][17:10],data[1][17:10],data[1][17:10]};
wire [18:0] myaddr2 = 0;//{1'b0, y_delay[38:30], eo_delay[3], x_delay[37:30]};

// update the output address and data only when four bytes ready

reg [18:0] ntsc_addr;
reg [35:0] ntsc_data;
//wire ntsc_we = sw ? we_edge:we_edge & (x_delay[30]==1'b0);//was 31
wire ntsc_we = sw ? we_edge:we_edge & (x_delay[10]==1'b0);//was 31

always @(posedge clk)
  if ( ntsc_we )
    begin
  ntsc_addr <= sw ? myaddr2 : myaddr;  // normal and expanded modes
  //ntsc_data <= sw ? mydata2 : mydata;
  ntsc_data <= sw ? 36'b0 : mydata;
```

```
    end

endmodule // ntsc_to_zbt
```

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

////////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
      ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

  input clk;        // system clock
  input cen;        // clock enable for gating ZBT cycles
  input we;         // write enable (active HIGH)
  input [18:0] addr;     // memory address
  input [35:0] write_data;    // data to write
  output [35:0] read_data;    // data read from memory
  output   ram_clk;  // physical line to ram clock
  output   ram_we_b; // physical line to ram we_b
  output [18:0] ram_address;  // physical line to ram address
  inout [35:0]  ram_data; // physical line to ram data
  output   ram_cen_b;    // physical line to ram clock enable

  // clock enable (should be synchronous and one cycle high at a time)
  wire     ram_cen_b = ~cen;

  // create delayed ram_we signal: note the delay is by two cycles!
  // ie we present the data to be written two cycles after we is raised
  // this means the bus is tri-stated two cycles after we is raised.
```

```
  reg [1:0]   we_delay;

  always @(posedge clk)
    we_delay <= cen ? {we_delay[0],we} : we_delay;

  // create two-stage pipeline for write data

  reg [35:0]  write_data_old1;
  reg [35:0]  write_data_old2;
  always @(posedge clk)
    if (cen)
      {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

  // wire to ZBT RAM signals

  assign      ram_we_b = ~we;
  assign      ram_clk = 1'b0;  // gph 2011-Nov-10
                               // set to zero as place holder

//   assign      ram_clk = ~clk;     // RAM is not happy with our data hold
                                   // times if its clk edges equal FPGA's
                                   // so we clock it on the falling edges
                                   // and thus let data stabilize longer
  assign      ram_address = addr;

  assign      ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
  assign      read_data = ram_data;

endmodule // zbt_6111
```

```
`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

divider_core hue_div1(
.clk(clock),
.dividend(s_top),
.divisor(s_bottom),
.quotient(s_quotient),
    // note: the "fractional" output was originally named "remainder" in this
// file -- it seems coregen will name this output "fractional" even if
// you didn't select the remainder type as fractional.
.fractional(s_remainder),
.rfd(s_rfd)
);
divider_core hue_div2(
.clk(clock),
.dividend(h_top),
.divisor(h_bottom),
.quotient(h_quotient),
.fractional(h_remainder),
.rfd(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

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:19:35 11/20/2016
// Design Name:
// Module Name:    clock_divider
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module clock_divider(clock_in, clock_out);
  input clock_in;
  output reg clock_out;

  reg [6:0] counter;

  always @ (posedge clock_in) begin
     counter <= counter + 1;
     if (counter == 3) begin
        counter <= 0;
        clock_out <= !clock_out;
     end
  end


endmodule
```

```
module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
        clock_feedback_in, clock_feedback_out, locked);

  input ref_clock;               // Reference clock input
  output fpga_clock;               // Output clock to drive FPGA logic
  output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
  input  clock_feedback_in;        // Output to feedback trace
  output clock_feedback_out;       // Input from feedback trace
  output locked;                 // Indicates that clock outputs are stable

  wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset, ram_clock; //added ram_clock
to get rid of errors

  //////////////////////////////////////////////////////////////////////////////

  //To force ISE to compile the ramclock, this line has to be removed.
  //IBUFG ref_buf (.O(ref_clk), .I(ref_clock));

  assign ref_clk = ref_clock;

  BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

  DCM int_dcm (.CLKFB(fpga_clock),
     .CLKIN(ref_clk),
     .RST(dcm_reset),
     .CLK0(fpga_clk),
     .LOCKED(lock1));
  // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
  // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
  // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
  // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
  // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
  // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
  // synthesis attribute PHASE_SHIFT of int_dcm is 0

  BUFG ext_buf (.O(ram_clock), .I(ram_clk));

  IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

  DCM ext_dcm (.CLKFB(fb_clk),
        .CLKIN(ref_clk),
        .RST(dcm_reset),
        .CLK0(ram_clk),
```

```
        .LOCKED(lock2));
  // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
  // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
  // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
  // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
  // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
  // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
  // synthesis attribute PHASE_SHIFT of ext_dcm is 0

  SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
        .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
  // synthesis attribute init of dcm_rst_sr is "000F";


  OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
         .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
  OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
         .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
  OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock), .C1(~ram_clock),
         .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

  assign locked = lock1 && lock2;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:30:07 11/15/2016
// Design Name:
// Module Name:    reg_green_filter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module blue_green_filter(clock, switch, reset, fvh, dv, R, G, B, Rout, Gout, Bout);
   input wire clock;
   input wire switch;
   input wire reset;
   input [2:0] fvh;
   input dv;
   input wire [7:0] R;
   input wire [7:0] G;
   input wire [7:0] B;
   output reg [7:0] Rout;
   output reg [7:0] Gout;
   output reg [7:0] Bout;

   reg G_threshold = 150;
   reg B_high = 240;
   reg B_threshold = 130;


  parameter   COL_START = 11'd0;
  parameter    ROW_START = 10'd0;


  reg [10:0]  col = 0;
```

```
reg [9:0]    row = 0;


reg         old_frame;    // frames are even / odd interlaced

wire     frame = fvh[2];
wire     frame_edge = frame & ~old_frame;

always @ (posedge clock) //LLC1 is reference
  begin
old_frame <= frame;

   if (!fvh[2]) begin
       col <= fvh[0] ? COL_START :
         (!fvh[2] && !fvh[1] && dv && (col < 800)) ? col + 1 : col;
       row <= fvh[1] ? ROW_START :
         (!fvh[2] && fvh[0] && (row < 600)) ? row + 1 : row;
       //vdata <= (dv && !fvh[2]) ? din : vdata;
   end
   //filtering green in RGB
   if (G>B && G>R && G>G_threshold && !switch) begin
        Rout <= 0;
        Gout <= 255;
        Bout <= 0;
   end
   //filtering blue in RGB
   else if (B>G && B>R && B>B_threshold && !switch) begin
      Rout <= 0;
      Gout <= 0;
      Bout <= 255;
   end
   //make the exterior red
   else if ((row*2 < 10) || (row*2 > 500) || (col < 10) || (col > 664)) begin
      Rout <= 255;
      Gout <= 0;
      Bout <= 0;
   end
   //display the camera view if switch on
   else if (switch) begin
      Rout <= R;
      Gout <= G;
      Bout <= B;
   end
```

```
    //make everything else black if switch off
    else begin
      Rout <= 0;
      Gout <= 0;
      Bout <= 0;
    end
  end



endmodule
```

```
`default_nettype none
module zbt_6111_sample(beep, audio_reset_b,
          ac97_sdata_out, ac97_sdata_in, ac97_synch,
       ac97_bit_clock,

       vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
       vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
       vga_out_vsync,

       tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
       tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
       tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

       tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
       tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
       tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
       tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

       ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
       ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

       ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
       ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

       clock_feedback_out, clock_feedback_in,

       flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
       flash_reset_b, flash_sts, flash_byte_b,

       rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

       mouse_clock, mouse_data, keyboard_clock, keyboard_data,

       clock_27mhz, clock1, clock2,

       disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
       disp_reset_b, disp_data_in,

       button0, button1, button2, button3, button_enter, button_right,
       button_left, button_down, button_up,

       switch,
```

```
        led,

        user1, user2, user3, user4,

        daughtercard,

        systemace_data, systemace_address, systemace_ce_b,
        systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

        analyzer1_data, analyzer1_clock,
        analyzer2_data, analyzer2_clock,
        analyzer3_data, analyzer3_clock,
        analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
  vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
  tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
  tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
  tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
  tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;
```

```
input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
  button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0]  systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
    analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////
```

```
  // Audio Input and Output
  assign beep= 1'b0;
  assign audio_reset_b = 1'b0;
  assign ac97_synch = 1'b0;
  assign ac97_sdata_out = 1'b0;
/*
*/
  // ac97_sdata_in is an input

  // Video Output
  assign tv_out_ycrcb = 10'h0;
  assign tv_out_reset_b = 1'b0;
  assign tv_out_clock = 1'b0;
  assign tv_out_i2c_clock = 1'b0;
  assign tv_out_i2c_data = 1'b0;
  assign tv_out_pal_ntsc = 1'b0;
  assign tv_out_hsync_b = 1'b1;
  assign tv_out_vsync_b = 1'b1;
  assign tv_out_blank_b = 1'b1;
  assign tv_out_subcar_reset = 1'b0;

  // Video Input
  //assign tv_in_i2c_clock = 1'b0;
  assign tv_in_fifo_read = 1'b1;
  assign tv_in_fifo_clock = 1'b0;
  assign tv_in_iso = 1'b1;
  //assign tv_in_reset_b = 1'b0;
  assign tv_in_clock = clock_27mhz;//1'b0;
  //assign tv_in_i2c_data = 1'bZ;
  // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
  // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

  // SRAMs

/* change lines below to enable ZBT RAM bank0 */

/*
  assign ram0_data = 36'hZ;
  assign ram0_address = 19'h0;
  assign ram0_clk = 1'b0;
  assign ram0_we_b = 1'b1;
  assign ram0_cen_b = 1'b0;   // clock enable
*/
```

```
/* enable RAM pins */

  assign ram0_ce_b = 1'b0;
  assign ram0_oe_b = 1'b0;
  assign ram0_adv_ld = 1'b0;
  assign ram0_bwe_b = 4'h0;

/**********/

  assign ram1_data = 36'hZ;
  assign ram1_address = 19'h0;
  assign ram1_adv_ld = 1'b0;
  assign ram1_clk = 1'b0;

  //These values has to be set to 0 like ram0 if ram1 is used.
  assign ram1_cen_b = 1'b1;
  assign ram1_ce_b = 1'b1;
  assign ram1_oe_b = 1'b1;
  assign ram1_we_b = 1'b1;
  assign ram1_bwe_b = 4'hF;

  // clock_feedback_out will be assigned by ramclock
  // assign clock_feedback_out = 1'b0;  //2011-Nov-10
  // clock_feedback_in is an input

  // Flash ROM
  assign flash_data = 16'hZ;
  assign flash_address = 24'h0;
  assign flash_ce_b = 1'b1;
  assign flash_oe_b = 1'b1;
  assign flash_we_b = 1'b1;
  assign flash_reset_b = 1'b0;
  assign flash_byte_b = 1'b1;
  // flash_sts is an input

  // RS-232 Interface
  assign rs232_txd = 1'b1;
  assign rs232_rts = 1'b1;
  // rs232_rxd and rs232_cts are inputs

  // PS/2 Ports
  // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs
```

```
  // LED Displays
/*
  assign disp_blank = 1'b1;
  assign disp_clock = 1'b0;
  assign disp_rs = 1'b0;
  assign disp_ce_b = 1'b1;
  assign disp_reset_b = 1'b0;
  assign disp_data_out = 1'b0;
*/
  // disp_data_in is an input

  // Buttons, Switches, and Individual LEDs
  //lab3 assign led = 8'hFF;
  // button0, button1, button2, button3, button_enter, button_right,
  // button_left, button_down, button_up, and switches are inputs

  // User I/Os
  assign user1 = 32'hZ;
  assign user2 = 32'hZ;
  assign user3 = 32'hZ;
  assign user4 = 32'hZ;

  // Daughtercard Connectors
  assign daughtercard = 44'hZ;

  // SystemACE Microprocessor Port
  assign systemace_data = 16'hZ;
  assign systemace_address = 7'h0;
  assign systemace_ce_b = 1'b1;
  assign systemace_we_b = 1'b1;
  assign systemace_oe_b = 1'b1;
  // systemace_irq and systemace_mpbrdy are inputs

  // Logic Analyzer
  assign analyzer1_data = 16'h0;
  assign analyzer1_clock = 1'b1;
  assign analyzer2_data = 16'h0;
  assign analyzer2_clock = 1'b1;
  assign analyzer3_data = 16'h0;
  assign analyzer3_clock = 1'b1;
  assign analyzer4_data = 16'h0;
  assign analyzer4_clock = 1'b1;
```

```
/*   ////////////////////////////////////////////////////////////////////////
  // Demonstration of ZBT RAM as video memory

  // use FPGA's digital clock manager to produce a
  // 65MHz clock (actually 64.8MHz)
  wire clock_65mhz_unbuf,clock_65mhz;
  DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
  // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
  // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
  // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
  // synthesis attribute CLKIN_PERIOD of vclk1 is 37
  BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

//   wire clk = clock_65mhz;  // gph 2011-Nov-10
*/
  ////////////////////////////////////////////////////////////////////////
  // Demonstration of ZBT RAM as video memory

  // use FPGA's digital clock manager to produce a
  // 40MHz clock (actually 40.5MHz)
  wire clock_40mhz_unbuf,clock_40mhz;
  DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
  // synthesis attribute CLKFX_DIVIDE of vclk1 is 2
  // synthesis attribute CLKFX_MULTIPLY of vclk1 is 3
  // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
  // synthesis attribute CLKIN_PERIOD of vclk1 is 37
  BUFG vclk2(.O(clock_40mhz),.I(clock_40mhz_unbuf));

  wire clk;// = clock_40mhz;

  wire locked;
  //assign clock_feedback_out = 0; // gph 2011-Nov-10

  ramclock rc(.ref_clock(clock_40mhz), .fpga_clock(clk),
              .ram0_clock(ram0_clk),
              //.ram1_clock(ram1_clk),   //uncomment if ram1 is used
              .clock_feedback_in(clock_feedback_in),
              .clock_feedback_out(clock_feedback_out), .locked(locked));


  // power-on reset generation
  wire power_on_reset;    // remain high for first 16 clocks
```

```
SRL16 reset_sr (.D(1'b0), .CLK(clk), .Q(power_on_reset),
     .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
defparam reset_sr.INIT = 16'hFFFF;

// ENTER button is user reset
wire reset,user_reset;
debounce db1(power_on_reset, clk, ~button_enter, user_reset);
assign reset = user_reset | power_on_reset;

// display module for debugging



// generate basic XVGA video signals
wire [10:0] hcount;
wire [9:0]  vcount;
wire hsync,vsync,blank;
xvga xvga1(clk,hcount,vcount,hsync,vsync,blank);

// wire up to ZBT ram

wire [35:0] vram_write_data;
wire [35:0] vram_read_data;
wire [18:0] vram_addr;
wire        vram_we;

wire ram0_clk_not_used;
zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
     vram_write_data, vram_read_data,
     ram0_clk_not_used,   //to get good timing, don't connect ram_clk to zbt_6111
     ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

// generate pixel value from reading ZBT memory
wire [17:0]     vr_pixel;
wire [18:0]     vram_addr1;

vram_display vd1(reset,clk,hcount,vcount,vr_pixel,
      vram_addr1,vram_read_data);

// ADV7185 NTSC decoder interface code
// adv7185 initialization module
adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz),
         .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
```

```
        .tv_in_i2c_clock(tv_in_i2c_clock),
        .tv_in_i2c_data(tv_in_i2c_data));

 wire [29:0] ycrcb;  // video data (luminance, chrominance)
 wire [2:0] fvh; // sync for field, vertical, horizontal
 wire       dv;  // data valid

 ntsc_decode decode (.clk(tv_in_line_clock1), .reset(reset),
        .tv_in_ycrcb(tv_in_ycrcb[19:10]),
        .ycrcb(ycrcb), .f(fvh[2]),
        .v(fvh[1]), .h(fvh[0]), .data_valid(dv));

 // code to write NTSC data to video memory

 wire [18:0] ntsc_addr;
 wire [35:0] ntsc_data;
 wire        ntsc_we;
 wire [7:0] R;
 wire [7:0] G;
 wire [7:0] B;
 wire [2:0] fvh_delayed;
 wire dv_delayed;


 YCrCb2RGB convert (fvh, dv, R, G, B, tv_in_line_clock1, 1'b0, ycrcb[29:20], ycrcb[19:10],
ycrcb[9:0], fvh_delayed, dv_delayed);

 wire [7:0] H;
 wire [7:0] S;
 wire [7:0] V;
 wire [7:0] Rout;
 wire [7:0] Gout;
 wire [7:0] Bout;
 wire [10:0] x_g;
 wire [10:0] x_b;
 wire [9:0] y_g;
 wire [9:0] y_b;
 //wire [7:0] Rout_ntsc;
 //wire [7:0] Gout_ntsc;
 //wire [7:0] Bout_ntsc;

 //first, we filter the RGB color image from the camera based on RGB values
 blue_green_filter filter(tv_in_line_clock1, switch[5],  1'b0, fvh, dv, R, G, B, Rout, Gout, Bout);
```

```
  //then we pass the output of the RGB filter to the center of mass calculations
  center_of_mass center(tv_in_line_clock1, fvh, dv, R, G, B, x_g, y_g, x_b, y_b);

  //store the filtered image in ZBT
  ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, {Rout[7:2],Gout[7:2],Bout[7:2]}, //was Rout,
Gout, Bout
        ntsc_addr, ntsc_data, ntsc_we, switch[6]); //fvh_delayed, dv_delayed

  // code to write pattern to ZBT memory
  reg [31:0]  count;
  always @(posedge clk) count <= reset ? 0 : count + 1;

  wire [18:0]     vram_addr2 = count[0+18:0];
  wire [35:0]     vpat = ( switch[1] ? {4{count[3+3:3],4'b0}}
        : {4{count[3+4:4],4'b0}} );

  // mux selecting read/write to memory based on which write-enable is chosen

  wire    sw_ntsc = ~switch[7];
  wire    my_we = sw_ntsc ? (hcount[0]==1'd1) : blank; //weird
  wire [18:0]     write_addr = sw_ntsc ? ntsc_addr : vram_addr2;
  wire [35:0]     write_data = sw_ntsc ? ntsc_data : vpat;

//   wire  write_enable = sw_ntsc ? (my_we & ntsc_we) : my_we;
//   assign    vram_addr = write_enable ? write_addr : vram_addr1;
//   assign    vram_we = write_enable;

  assign  vram_addr = my_we ? write_addr : vram_addr1;
  assign  vram_we = my_we;
  assign  vram_write_data = write_data;

  // select output pixel data

  wire [23:0] pixel;
  reg     b,hs,vs;
  wire phsync,pvsync,pblank;

  always @(posedge clk)
    begin
// //pixel <= switch[0] ? {hcount[8:6],5'b0} : vr_pixel;
// //pixel <= switch[0] ? (hcount == 512 || vcount == 384) ? 0 : vr_pixel : vr_pixel;
```

```
// //pixel <= switch[0] ? (hcount == x_g || vcount == y_g) ? 85 : (hcount == x_b || vcount == y_b)
? 170 : vr_pixel : vr_pixel : vr_pixel; //85 is green
// if (switch[0] && (hcount == x_g || vcount == y_g)) begin
//    pixel <= 18'b000000111111000000;
// end
// else if (switch[1] && (hcount == x_b || vcount == y_b)) begin
//    pixel <= 18'b000000000000111111;
// end
// else begin
//    pixel <= vr_pixel;
// end


  b <= pblank;
  hs <= phsync;
  vs <= pvsync;
    end

  // VGA Output.  In order to meet the setup and hold times of the
  // AD7125, we send it ~clk.
//   assign vga_out_red = {pixel[17:12],2'b0};
//   assign vga_out_green = {pixel[11:6],2'b0};
//   assign vga_out_blue = {pixel[5:0],2'b0};
// wire [23:0] pixel;
  assign vga_out_red = pixel[23:16];
  assign vga_out_green = pixel[15:8];
  assign vga_out_blue = pixel[7:0];

  assign vga_out_sync_b = 1'b1;    // not used
  assign vga_out_pixel_clock = ~clk;
  assign vga_out_blank_b = ~b;
  assign vga_out_hsync = hs;
  assign vga_out_vsync = vs;

  // debugging

//   assign led = ~{vram_addr[18:13],reset,switch[0]};

//   always @(posedge clk)
//     // dispdata <= {vram_read_data,9'b0,vram_addr};
//     dispdata <= {ntsc_data,9'b0,ntsc_addr};

  /////Tianye's stuff///
```

```
  ///I2C stuff///
  wire clock_200khz;
  wire start;
  wire [6:0] slave_addr = 7'h38; ///address of touchscreen chip
  wire [7:0] reg_addr = 8'h03;
  wire [31:0] data_rd; ///data read from the registers 3, 4, 5, 6
  wire [1:0] event_flag;
  assign event_flag = data_rd[31:30];
  wire [11:0] data_x; ///horizontal position of the touch
  wire [11:0] data_y; ///vertical position of the touch t
  assign data_y = data_rd[27:16]; ///12bits
  assign data_x = data_rd[11:0]; ///12bits
  wire [7:0] i2c_state;
  wire sda;
  wire scl;
  assign user1[31] = sda;
  assign user1[30] = scl;
  assign user1[29] = start;
  wire read;
  i2c_start debug(.clock(clock_200khz), .i2c_start(start));
  clock_200khz i2c_clock(.reset(reset), .clk(clock_27mhz), .clock_200khz(clock_200khz));
  I2C_master i2c_master(.clk(clock_200khz), .reset(reset), .start(start), .slave_addr(slave_addr),
.reg_addr(reg_addr),
                        .data_rd(data_rd), .state(i2c_state), .sda(sda), .scl(scl));

  ///drawing path///
  wire clear_button_pulse;
  wire clear_button;
  debounce
db_clear(.reset(power_on_reset),.clock(clock_40mhz),.noisy(~button0),.clean(clear_button));
  single_pulse_button clear_drawing(.clock(clock_40mhz), .button(clear_button),
.button_on(clear_button_pulse));

  display_trace_bram dt_bram(.vclock(clock_40mhz),.reset(reset),
          .hcount(hcount),.vcount(vcount),
          .hsync(hsync),.vsync(vsync),.blank(blank),
          .phsync(phsync),.pvsync(pvsync),.pblank(pblank),.data_x(data_x), .data_y(data_y),
.clear_button_pulse(clear_button_pulse),
          .pixel(pixel),
          .vr_pixel(vr_pixel), .switch(switch[1:0]), .switch5(switch[5]), .x_g(x_g), .y_g(y_g),
.x_b(x_b), .y_b(y_b));
  ///tank controller stuff///
  wire forward;
```

```
  wire forward_button;
  wire forward_pulse;
  wire reverse;
  wire reverse_button;
  wire reverse_pulse;
  wire left;
  wire left_button;
  wire left_pulse;
  wire right;
  wire right_button;
  wire right_pulse;
  assign user2[31] = forward;
  assign user2[30] = reverse;
  assign user2[29] = left;
  assign user2[28] = right;

// used to debug tank movement //
// debounce
db_forward(.reset(power_on_reset),.clock(clock_27mhz),.noisy(~button_up),.clean(forward_butt
on));
// debounce
db_reverse(.reset(power_on_reset),.clock(clock_27mhz),.noisy(~button_down),.clean(reverse_
button));
// debounce
db_right(.reset(power_on_reset),.clock(clock_27mhz),.noisy(~button_left),.clean(left_button));
// debounce
db_left(.reset(power_on_reset),.clock(clock_27mhz),.noisy(~button_right),.clean(right_button));

// single_pulse_button sp_forward(.clock(clock_27mhz), .button(forward_button),
.button_on(forward_pulse));
// single_pulse_button sp_reverse(.clock(clock_27mhz), .button(reverse_button),
.button_on(reverse_pulse));
// single_pulse_button sp_left(.clock(clock_27mhz), .button(left_button), .button_on(left_pulse));
// single_pulse_button sp_right(.clock(clock_27mhz), .button(right_button),
.button_on(right_pulse));

  assign user2[0] = forward;
  assign user2[1] = reverse;
  wire forward_done;
  wire reverse_done;
  wire left_done;
  wire right_done;
  wire fsm_forward;
```

```
  wire fsm_right;
  wire fsm_left;

  pdm pdmforward (.clock(clock_27mhz), .start(fsm_forward),
.signal(forward),.off_time(4'b0010), . done(forward_done));
// pdm pdmreverse (.clock(clock_27mhz), .start(reverse_button),
.signal(reverse),.off_time(off_time), . done(reverse_done));
  turn turnright (.clock(clock_27mhz), .start(fsm_right),.signal(right), .done(right_done));
  turn turnleft (.clock(clock_27mhz), .start(fsm_left),.signal(left), .done(left_done));

////testing halfplane by hardcoding values////
// wire clean_button1;
// debounce
db_1(.reset(power_on_reset),.clock(clock_40mhz),.noisy(~button1),.clean(clean_button1));
// wire clean_button2;
// debounce
db_2(.reset(power_on_reset),.clock(clock_40mhz),.noisy(~button2),.clean(clean_button2));
// wire signed [11:0] test_x;
// assign test_x = clean_button1?12'd50:12'd100;
// wire signed [11:0] test_y = 12'd220;
// wire signed [11:0] start_x = 12'd100;
// wire signed [11:0] start_y = 12'd175;
// wire signed [11:0] end_x = 12'd200;
// wire signed [11:0] end_y = 12'd200;
// wire turn_left;
// wire turn_right;
// assign led = ~{5'b00000, forward_button ,turn_left, turn_right};

// half_plane test_half_plane(.clock(clock_27mhz), .test_x(test_x), .test_y(test_y),
.start_x(start_x),.start_y(start_y),
//                          .end_x(end_x), .end_y(end_y), .turn_right(turn_right),.turn_left(turn_left));

///testing control fsm///
// wire [10:0] front_x;
// assign front_x = switch[2]?11'd100:11'd50;
// wire [9:0] front_y;
// assign front_y = switch[2]?10'd50:10'd100;
// wire [10:0] back_x;
// assign back_x = 11'd50;
// wire [9:0] back_y;
// assign back_y = 10'd50;
// wire [11:0] test_touch_x;
// assign test_touch_x = 12'd200;
```

```
// wire [11:0] test_touch_y;
// assign test_touch_y = 12'd50;
// wire same_goal;
// wire fsm_right;
// wire fsm_left;
// control_FSM test_fsm(.clock(clock_27mhz), .tank_front_x(front_x), .tank_front_y(front_y),
//                      .tank_back_x(back_x), .tank_back_y(back_y), .touch_x_raw(test_touch_x),
.touch_y_raw(test_touch_y),
//                      .is_touch (is_touch), .forward(forward),
//                      .reverse(reverse), .right(right), .left(left), .debug_state(fsm_state));

  assign led = {~8'b0000000};
  wire [3:0] control_fsm_state;
  wire same_goal;
  control_FSM control_fsm(.clock(clock_27mhz), .tank_front_x(x_g), .tank_front_y(y_g),
                      .tank_back_x(x_b), .tank_back_y(y_b), .touch_x_raw(data_x),
.touch_y_raw(data_y),
                      .forward(fsm_forward),
                      .reverse(reverse), .right(fsm_right), .left(fsm_left),
.debug_state(control_fsm_state), .same_goal(same_goal));

  display_16hex hexdisp1(reset, clk, {40'b0,3'b0, same_goal, 3'b0, control_fsm_state,
3'b0,fsm_forward, 3'b0,fsm_left, 3'b0,fsm_right},
          disp_blank, disp_clock, disp_rs, disp_ce_b,
          disp_reset_b, disp_data_out);

endmodule

////small helper modules////
////generates start signals for i2c module////
module i2c_start(
  input wire clock ,
  output reg i2c_start
  );

  reg [23:0] counter = 0;
  always@(posedge clock) begin
    if (counter < 24'd10000)begin
      i2c_start <= 0;
      counter <= counter+1;
    end
    else begin
      i2c_start <=1;
```

```
      counter <=0;
    end
  end
endmodule

///turns a button press into a single pulse///
module single_pulse_button (
  input wire clock,
  input wire button,
  output reg button_on
  );
  reg button_state = 0;

  always @ (posedge clock) begin
    button_state <= button;
    if (button == 1 && button_state == 0)begin
      button_on <= 1;
    end
    else begin
      button_on <= 0;
    end
  end
endmodule

///generates clock for i2c module///
module clock_200khz   (input reset, input clk, output reg clock_200khz);
   //assume incoming signal is 25 mhz, so we need to slow clock down by 12.5*100
   //every 250 posedges of clock we will toggle slow_clock state
   reg [19:0] count=0;
   always @(posedge clk)begin
      if (reset) begin
         count <=0;
         clock_200khz <=1;
      end else begin
         if (count == 67)begin
            count <=0;
            clock_200khz <= !clock_200khz;
         end else begin
            count <= count +1;
         end
      end
   end
endmodule
```

```
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
module debounce #(parameter DELAY=270000)   // .01 sec with a 27Mhz clock
          (input reset, clock, noisy,
           output reg clean);

  reg [18:0] count;
  reg new;

  always @(posedge clock)
    if (reset)
      begin
    count <= 0;
    new <= noisy;
    clean <= noisy;
      end
    else if (noisy != new)
      begin
    new <= noisy;
    count <= 0;
      end
    else if (count == DELAY)
      clean <= new;
    else
      count <= count+1;

endmodule


////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 28-Nov-06 CJT: fixed race condition between CE and RS (thanks Javier!)
```

```
//
// This verilog module drives the labkit hex dot matrix displays, and puts
// up 16 hexadecimal digits (8 bytes).  These are passed to the module
// through a 64 bit wire ("data"), asynchronously.
//
//////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data,
    disp_blank, disp_clock, disp_rs, disp_ce_b,
    disp_reset_b, disp_data_out);

 input reset, clock_27mhz;    // clock and reset (active high reset)
 input [63:0] data;     // 16 hex nibbles to display

 output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
  disp_reset_b;

 reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

 //////////////////////////////////////////////////////////////////////////////
 //
 // Display Clock
 //
 // Generate a 500kHz clock for driving the displays.
 //
 //////////////////////////////////////////////////////////////////////////////

 reg [4:0] count;
 reg [7:0] reset_count;
 reg clock;
 wire dreset;

 always @(posedge clock_27mhz)
  begin
 if (reset)
  begin
    count = 0;
    clock = 0;
  end
 else if (count == 26)
  begin
    clock = ~clock;
    count = 5'h00;
```

```
  end
else
  count = count+1;
  end

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign dreset = (reset_count != 0);

assign disp_clock = ~clock;

/////////////////////////////////////////////////////////////////////////////
//
// Display State Machine
//
/////////////////////////////////////////////////////////////////////////////

reg [7:0] state;       // FSM state
reg [9:0] dot_index;       // index to current dot being clocked out
reg [31:0] control;    // control register
reg [3:0] char_index;   // index of current character
reg [39:0] dots;       // dots for a single digit
reg [3:0] nibble;      // hex nibble of current character

assign disp_blank = 1'b0; // low <= not blanked

always @(posedge clock)
  if (dreset)
    begin
  state <= 0;
  dot_index <= 0;
  control <= 32'h7F7F7F7F;
    end
  else
    casex (state)
 8'h00:
   begin
      // Reset displays
      disp_data_out <= 1'b0;
```

```
    disp_rs <= 1'b0; // dot register
    disp_ce_b <= 1'b1;
    disp_reset_b <= 1'b0;
    dot_index <= 0;
    state <= state+1;
  end

8'h01:
  begin
    // End reset
    disp_reset_b <= 1'b1;
    state <= state+1;
  end

8'h02:
  begin
    // Initialize dot register (set all dots to zero)
    disp_ce_b <= 1'b0;
    disp_data_out <= 1'b0; // dot_index[0];
    if (dot_index == 639)
  state <= state+1;
    else
  dot_index <= dot_index+1;
  end

8'h03:
  begin
    // Latch dot data
    disp_ce_b <= 1'b1;
    dot_index <= 31;    // re-purpose to init ctrl reg
    disp_rs <= 1'b1; // Select the control register
    state <= state+1;
  end

8'h04:
  begin
    // Setup the control register
    disp_ce_b <= 1'b0;
    disp_data_out <= control[31];
    control <= {control[30:0], 1'b0};    // shift left
    if (dot_index == 0)
  state <= state+1;
    else
```

```
  dot_index <= dot_index-1;
  end

 8'h05:
  begin
    // Latch the control register data / dot data
    disp_ce_b <= 1'b1;
    dot_index <= 39;    // init for single char
    char_index <= 15;       // start with MS char
    state <= state+1;
    disp_rs <= 1'b0;    // Select the dot register
  end

 8'h06:
  begin
    // Load the user's dot data into the dot reg, char by char
    disp_ce_b <= 1'b0;
    disp_data_out <= dots[dot_index]; // dot data from msb
    if (dot_index == 0)
      if (char_index == 0)
        state <= 5;        // all done, latch data
  else
  begin
    char_index <= char_index - 1;    // goto next char
    dot_index <= 39;
  end
     else
  dot_index <= dot_index-1;  // else loop thru all dots
  end

   endcase

always @ (data or char_index)
  case (char_index)
    4'h0:      nibble <= data[3:0];
    4'h1:      nibble <= data[7:4];
    4'h2:      nibble <= data[11:8];
    4'h3:      nibble <= data[15:12];
    4'h4:      nibble <= data[19:16];
    4'h5:      nibble <= data[23:20];
    4'h6:      nibble <= data[27:24];
    4'h7:      nibble <= data[31:28];
    4'h8:      nibble <= data[35:32];
```

```
    4'h9:      nibble <= data[39:36];
    4'hA:      nibble <= data[43:40];
    4'hB:      nibble <= data[47:44];
    4'hC:      nibble <= data[51:48];
    4'hD:      nibble <= data[55:52];
    4'hE:      nibble <= data[59:56];
    4'hF:      nibble <= data[63:60];
  endcase

 always @(nibble)
  case (nibble)
    4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
    4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
    4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
    4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
    4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
    4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
    4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
    4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
    4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
    4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
    4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
    4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
    4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
    4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
    4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
    4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
  endcase

endmodule
```