Need for Speed: Hacker's Trail

Richard Chan, Calvin Chung, Fan Yang

December 15, 2007

ABSTRACT

"Hacker's Trail" is a racing simulation system that allows the user to create his own race track and subsequently race in it. The user can create his desired track simply by moving the mouse. With a built-in double buffer, the track is being displayed on the screen while it is being traced out. Instead of using a typical wired steering wheel, the user will have an "air wheel" that can detect his motion and adjust both the direction and the speed of car through the game logic. More specifically, the user will wear two colored gloves that will be used in conjunction with a video camera so that his hand motion can be detected.

TABLE OF CONTENTS

LIST OF FIGURES

OVERVIEW

“Hacker’s Trail” is an innovative racing system with a well-designed user interface. It allows user to create his own racing track and control his car by driving an “air wheel” in front of a video camera. Through the game logic the acceleration and direction of the car is adjusted, as the user aims to reach checkpoints 7 times in the shortest time interval possible.

The game starts off in edit mode with a circular track blob on top of the grassland. As the user drags the mouse, circular tracks are stored and displayed around every mouse position passed through, ultimately creating the desired track. Track data is continuously stored and updated, and by building a double buffer system, the track can be displayed on the screen real-time. In preparation for the play mode, at this stage the user also determines two checkpoints along the track by clicking the left and right mouse buttons.

As the user transits into play mode, his car shows up on the screen behind a ready screen. As the ready screen disappears and a countdown is started, the user should be ready in front of the video camera wearing a red glove on one hand and a blue glove on the other. The race officially starts as the timer displayed on the top right corner starts running. To control the car, user moves his hands as if he’s holding a wired wheel. The video camera detects his hand motion and sends in the signal to the game logic control, subsequently adjusting the direction and speed of the car. The higher he lifts his hands, the faster the car goes. Turning the wheel beyond 45 degrees to either side indicates a turn. On the top left hand corner is a mini-screen which displays to the user his hand positions as detected by the video camera. The user can also review his actions by observing the speed gauge on the screen and the rotation of his car. If the user happens to run his car out of the track, the car will decelerate substantially until it gets back to the track entirely.

The goal of the game is to alternately visit the two pre-determined checkpoints a total of 7 times in the shortest time possible. While practice makes perfect, the difficulty of the game can be increased simply by resetting a game and creating a different track. If I were you, I would never stop playing it.

<u>DESCRIPTION</u>

1. High Level Design

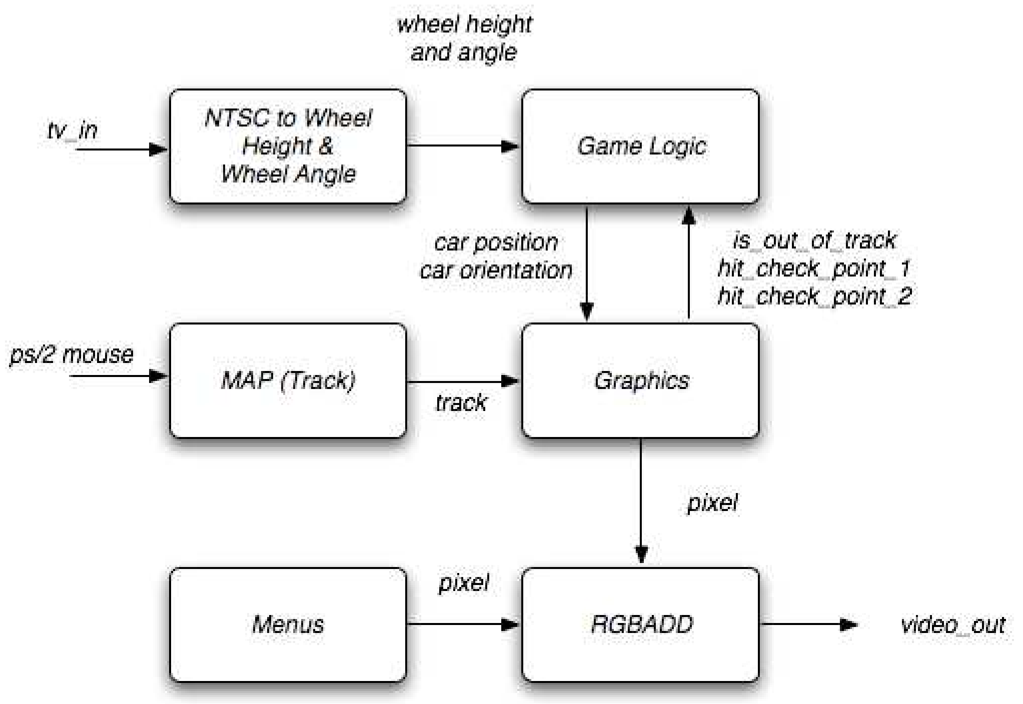

Figure 1. Top Level Block Diagram

The system takes in two inputs: the PS/2 mouse and the TV input, where the mouse cursor is used to draw the track and the TV input received from a camera is used to simulate the racing wheel.

The system can basically be divided into three major components: 1) the TV input, which is driven by a camera and the logic that translates the signal into wheel height and wheel angle; 2) the Game Logic and Graphics modules, which includes logics for updating the game state according to inputs every time frame; 3) the Map module, which updates according to signals from the PS/2 mouse and signals the Graphics module with whether certain pixels are covered by the track.

The following sections will describe the three modules in order. Lastly, section 5 will describe how the display is actually driven by the system.

2. Input Components

2.1 Camera Input Modules (Fan Yang and 6.111 staff)

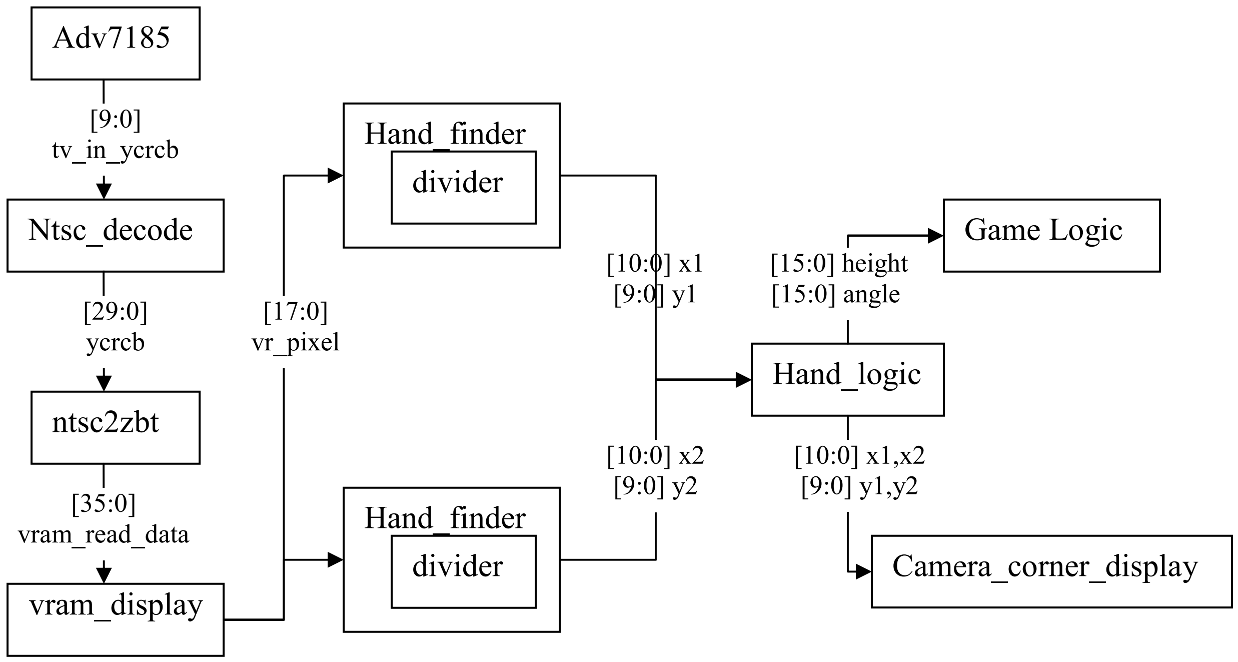

Figure 2. Camera Input Modules

From a high level standpoint, the input component encapsulates the logic of using the camera to detect user's hand positions, and then translates the hand positions into height and rotation signals that can be used to control the car. The camera input module consists of the adv7185, ntsc_decode, ntsc2zbt, vram_display, hand_finder, divider, hand_logic, and camera_corner_display modules. The adv7185 module takes the data from the camera and converts it to a stream of LLC data. The ntsc_decode module then takes in this stream of LLC data to generate the corresponding pixels into YCrCb format. The ntsc2zbt module is modified so that 18 bits of YCrCb data per pixel are buffered into the ZBT memory. Since the ZBT is 36 bits wide, YCrCb data for 2 pixels can be stored in each entry of ZBT. This provides enough bits to accurately detect objects with specific colors. Since we are detecting colored pixels, Cr and Cb components are much more important than the Y component. Therefore, to further enhance the color detection, the top 9 bits of Cr and the top 9 bits of Cb are buffered into ZBT memory while none of the Y bits is buffered. Several modifications were made to the ntsc2zbt module provided by the 6.111 staff. The major changes include having 18 bits of input for the YCrCb data and updating the output address and data only when two bytes are ready. The vram_display module then read the YCrCb values from the ZBT memory which can be then used by hand_finder module.

Three different methods of storing the pixel data has been experimented during the course of the project. For each of the methods, I stored 18 bits of data per pixel, but these methods differed in how the 18 bits of data is allocated and the quality of the color detection.

In the first method the most significant 6 bits were stored for each of the red, blue, and green. The YCrCb values were first converted to RGB values and then the RGB values were stored in the ZBT. The problem with this method lies in the color detection. Detecting color using RGB values works reasonably well when there's one color of interest. However, it becomes extremely challenging to detect two different colors simultaneously using RGB values because there's often overlap between the two colors and colors of the environment. This observation could be explained by the fact that in RGB, colors are a combination of red, blue, green. Having combinations of three colors to represent the color of interest is prone to have overlaps with the surrounding environment.

The second method I tried was storing the most significant 6 bits of each of the Y, Cr, and Cb values. This method worked better than the first method because it didn't have as much of an overlap with the colors of the surrounding environment. However, the Y value didn't contribute to the color detection as much as the Cr and Cb values. Therefore, this led me to the decision of choosing to store as many bits as possible for the Cr and Cb values, which helped me arrive at the third method that I experimented.

In the third method, the 18 bits were used to store only the Cr and Cb values. This method worked noticeably better than the second method. The background noise was reduced significantly. Red and Blue were chosen as the two colors for color detection because their Cr and Cb values are most different from the colors of the surroundings. Therefore, this is the method that I chose to use for this project.

2.2 Hand Finder Module (Fan Yang)

The hand_finder module takes in the YCrCb values from the Camera Input Modules and outputs the x, y values of the hand position after the given frame has passed. For each pixel, the module compares the YCrCb value of the pixel against a certain threshold of Cr and Cb values. If the threshold is met, then the pixel is counted as a hand pixel. The module also keeps a running sum of the hcount and vcount values of the hand pixels detected in a given frame. The horizontal and vertical center of mass of the hand pixels can be found by dividing the sum of hcount and vcount by the number of pixels. The division is done by using two instance of the divide module provided by Xilinx Core Generator. Due to the timing delays through the divide module, the divide operation starts only after the given frame has passed.

With modularity and reusability in mind, the module contains four parameters for specifying the appropriate range of Cr and Cb: CR_MAX, CR_MIN, CB_MAX, and CB_MIN. This design allows the user to specify the threshold for each instance of this module. Therefore, for calculating the position of two hands, two instances of the same module could be created, and the color ranges could be specified using the "defparam" method in Verilog. This design makes it very easy to provide support for more players. For instance, for creating a two player game, we can simply make four instances of the hand finder module and specify the color range for each module. This design makes the system more scalable as a whole.

2.3 Hand Logic Module (Fan Yang and Richard Chan)

The Hand Logic module takes in the (x, y) position of both hands and outputs the height and angle of the hands. The height and angle can then be used by the Game modules to compute the acceleration and rotation of the car. The height is calculated by taking the average of the y values of the two hands.

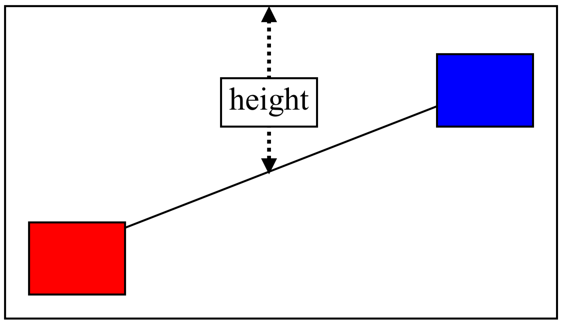

Figure 3. Height of the hands

This method of calculating the height of the two hands is reliable because the height of the hands will be independent of the angle between the two hands.

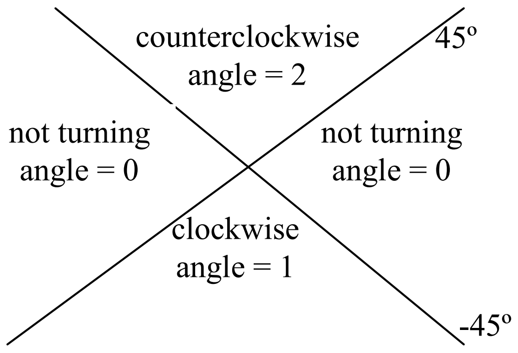

Figure 4. Orientation of hands

Since the orientation of the wheel will determine which direction the car is turning. The angle of the hands could have three values: 0, 1, or 2. If the hands are moving in a counterclockwise direction, the angle is 2; if the hands are moving in a clockwise direction, the angle is 1; if the hands are not moving, the angle is 0. The angle is determined by the angle formed by the line connecting the two hand positions and the horizontal line. If the angle is between 45 degrees and 135 degrees, then the hands are turning counterclockwise. If the angle is between -45 degrees and -135 degrees, then the hands are turning clockwise. If the angle is between -45 degrees and 45 degrees or between -135 degrees and 135 degrees, then the hands are recognized as not turning. In other words, if the angle formed by the hands is less than 45 degrees from the horizontal, then the hands are recognized as not turning. This decision was made based on the observation that the user tend to move hands inadvertently. On the other hand, when the user intentionally moves his hands to turn the car, the user usually moves his hands at least 45 degrees. Therefore, to avoid making the car overly sensitive to the movement of the user's hands, the decision was made to have 45 degrees as a threshold.

2.4 Camera Corner Display (Fan Yang)

Camera Corner Display provides the user with a view of the position of his hands so that he can see whether or not the camera is picking up the right angle and height. Since the camera input is very sensitive to light and the color of the environment, this module serves as a great debugging tool. This module takes in the (x, y) position of the two hands and outputs pixels that essentially displays the user's hands as two colored blobs on the top left part of the screen. The size of the display is 1/16 of the screen. The vertical height of the display is ¼ of the vertical height of the screen, and the horizontal length of the display is ¼ of the horizontal length of the screen. The ratio ¼ was chosen because it's a power of 2 so the division by 4 could be easily implemented in Verilog by a bit shift and would not result in any delays. Also, the ratio ¼ is very reasonable for the game—allowing the user to have enough screen area for the game playing and big enough view of his hand positions.

In order to display the relative hand positions correctly inside the display area on the top left of the screen, the x, y values of the blob is essentially the x, y values of the hand positions divided by 4. In each frame, this module compares the values of x/4 and y/4 against the hcount and vcount. If the hcount is between x/4-10 and x/4+10 and the vcount is between y/4-10 and y/4+10, then the pixel is colored with its corresponding color. The same logic is applied to the other set of (x, y) values.

3. Game Logic Modules (Richard Chan)

The Game Logic modules contain the logic behind how the game works, which is mainly separated into two main modules: the GameModule, which controls what happens to the game after each iteration (what happens in the next frame), and a Graphics module, which is responsible for not only drawing the background, the track and the car onto the screen, but also the collision detection of different objects in the game.

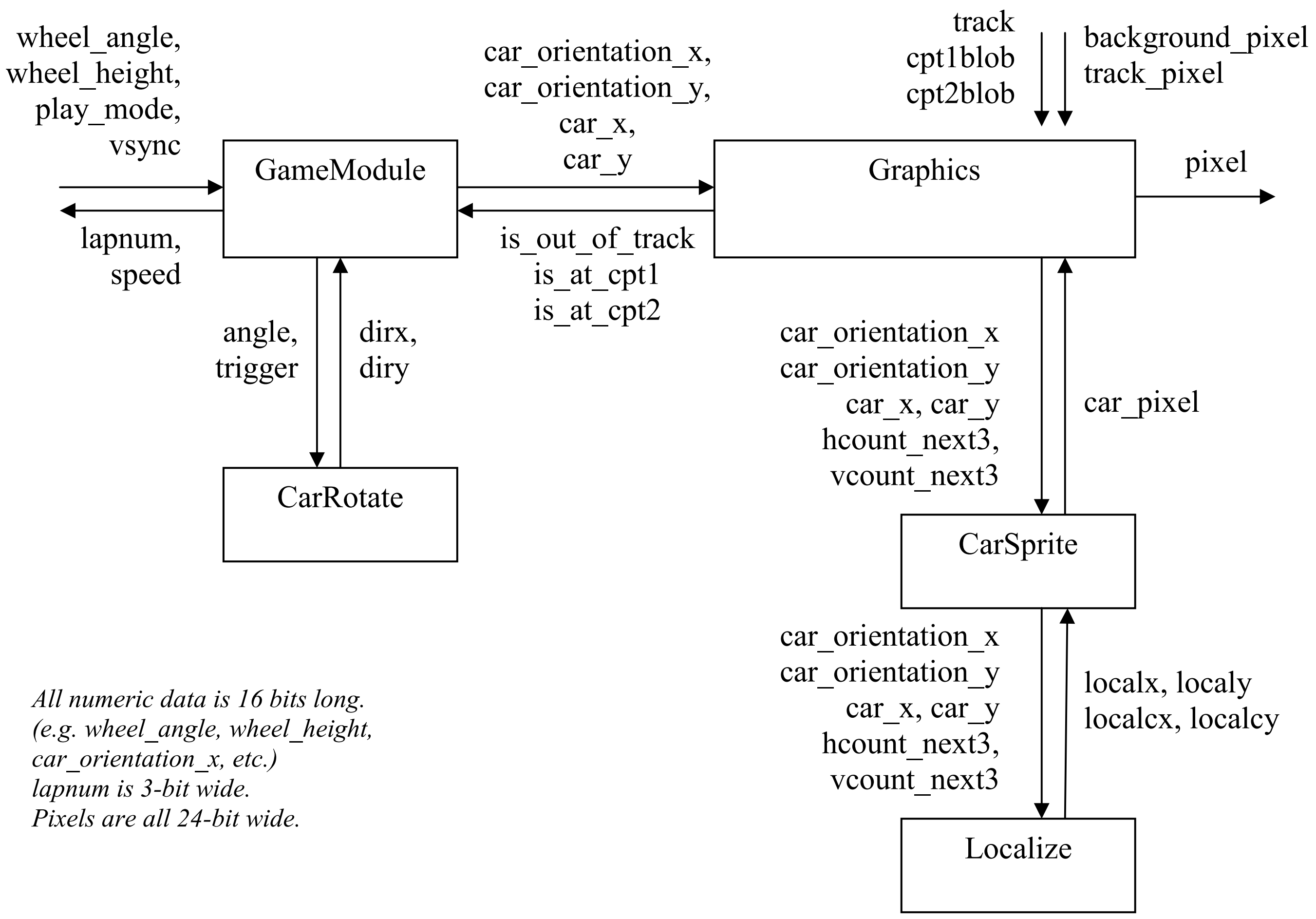

Figure 5: Game Logic Modules.

3.1 GameModule (Richard Chan)

The GameModule is the module that actually holds the state of the game: where the car is located, the direction of the car and the speed of the car. Its main responsibility is to take in the user inputs (wheel angle, wheel height) and computes new car positions and velocities based on those inputs as well as the current game state (e.g. whether the car is outside of the track) at every time frame. Each time frame in the system is basically one frame refresh, so, in other words, GameModule will update the car positions and velocities at every negative edge of the vsync, which signifies the end of a frame refresh.

The next car position is computed by multiplying the car's direction with its current speed, which essentially gives us the velocity of the car, and then adding the velocity onto the current position of the car.

The change in car speed depends on a few things: whether the user wishes to accelerate (which, in the case of using the *air wheel*, is lifting the wheel above the center of the screen), decelerate, or simply not apply gas, and if the car is out of track. If the car is out of track, it should not be able to go as fast as if it is on the track. To implement that, I basically allow users to accelerate (at higher rate for in-track, and at lower rate for out-of-track) until they hit some maximum speed limit (which is, again, higher for in-track and lower for out-of-track).

Note that the out-of-track signal is received from the Graphics module. Unlike some simple conventional game systems, collision detection is *not* computed directly using the location and directions of the car. Instead, we decided to detect collision *as the screen is being rendered*. Since, in the Graphics module, we will be rendering every pixel of the screen anyway, we can easily include logic to detect whether there is a pixel on the screen where both a pixel of the car and a pixel of the track overlap or do not overlap. The collision logic will be covered in more details in the next section.

The GameModule also keeps track of the lap number that the car is currently on. Starting from 0 whenever the system resets, the GameModule increments the counter whenever the car collides with the next checkpoint (the module keeps track of which checkpoint is next, starting with the 1st checkpoint and alternating whenever the lap number increases). Again the collision logic is done in the Graphics module, where the collisions with the checkpoints are represented with is_at_cpt1 and is_at_cpt2.

When the user turns the wheel, the GameModule must update the orientation of the car accordingly. As we briefly mentioned before, the GameModule keeps in memory the location of the car (x, y coordinates), the speed of the car, and the orientation of the car. To reduce complexity, the logic for rotating the car is separated into a sub-module: the CarRotate module.

3.2 CarRotate (Richard Chan)

The CarRotate module takes in the angle of the wheel and outputs the current orientation of the car (dir_x, dir_y). The orientation of the car changes basically every time the trigger input changes from negative to positive or vise versa. Note that the vector (dir_x , dir_y) is set to be normalized with length 256, which basically represents an 8-bit decimal point value for a unit vector.

Originally, CarRotate was very complicated, implemented by doing matrix multiplication with a rotation matrix (e.g. using [cos -sin ; sin cos]) to update the orientation. However, since there is always some error in multiplying with irrational numbers such as sine and cosine, the orientation vector (dir_x, dir_y) deviates a little every iteration. Because the orientation vector is used for mapping pixels of the car sprite to the screen (covered in more details in the Graphics module), the errors could accumulate to the point where the car could resize itself to a size bigger than the entire screen.

A quick hack was implemented to fix the problem by testing the length of the vector every time. If the length is too high, we will multiply dir_x and dir_y by a fraction slightly lower than 1 to "renormalize" the vectors. Note that we prefer not to truly normalize the vectors since normalizing requires taking the square root to find the length and dividing from numbers that are not powers of 2 (unable to do bit shift) – both of which requires many clock cycles to compute.

The hack was clearly not the best way to fix the problem. Eventually I realized a much simpler and more reliable solution to the problem – to simply pre-compute the values for each vector from 0 to 360 degrees and place them into a ROM. The current implementation uses this approach, where the values are stored in a ROM with 128 rows, 32 bits each. Each row contains a 16-bit number for dir_x and the 2nd 16-bit number represents dir_y. Turning left and right is simply implemented as lowering and increasing the address of the ROM to read from.

0	0.00 deg	0000000100000000 0000000000000000 = 256 | 0
1	2.85 deg	0000000011111111 0000000000001100 = 255 | 12
2	5.70 deg	0000000011111110 0000000000011001 = 254 | 25
3	8.55 deg	0000000011111101 0000000000100101 = 253 | 37

Figure 6: ROM Table for CarRotate

The above figure shows first 4 rows in the ROM storing the vectors at certain angles 0 to 360, normalized with length 256. To change the angle when triggered, we keep a register on the last trigger value. If that is different from the new trigger value, update the address appropriately.

3.3 Graphics Module as a Logic Module (Richard Chan)

The Graphics Module has two main responsibilities: to display the state of the game (using the inputs from the GameModule) into the screen and to record collisions between different objects as it is rendering the screen, making it essentially both a Logic and an Output module.

On the highest level, the Graphics Module is pretty simple. Basically, it takes in a "track" input, which is the signal of whether the user has drawn a track over that point of the screen (hcount, vcount) and the car positions and orientations signals from the Game Module. It sends the car positions and orientations signals into a Car Sprite module, which would output the RGB value of the car at (hcount, vcount) (for pixels the car sprite does not span over, the RGB value will be 0, signifying transparency).

Using those signals, the module can figure out whether the pixel should be a background pixel (if there is no track or car pixel there), a track pixel (if there is track but no car pixel) or a car pixel (if there is a car pixel at that point).

When outputting the RGB value of the pixel at (hcount, vcount), it can check if the pixel has a car pixel but no track on it. If so, that means the car is out of track and a register is_out_of_track, which is also an output to the game module, will be set to high. is_out_of_track remains high until the frame refreshes again – when there is a posedge on vsync, at which time is_out_of_track will be set to 0.

3.4 Car Sprite Module (Richard Chan)

The Car Sprite Module is what actually maps pixels of the car sprite onto the screen. The module basically has to do two things: 1) convert the (hcount, vcount) coordinate into a coordinate frame local to the orientation of the car; 2) using the localized coordinate, map the coordinate to an address on the ROM storing the actual image of the car and reads the RGB value from it.

To reduce the complexity of the module, the logic for localizing pixels are done in the LocalizePixel sub-module, where we basically pass it the center of the car, the direction of the car and the hcount, vcount coordinate. The LocalizePixel will convert the (hcount, vcount) coordinate into a frame local to the car and return the local coordinates as outputs in localx, localy, localcx, localcy. (relative to upper-left corner or center of car, respectively).

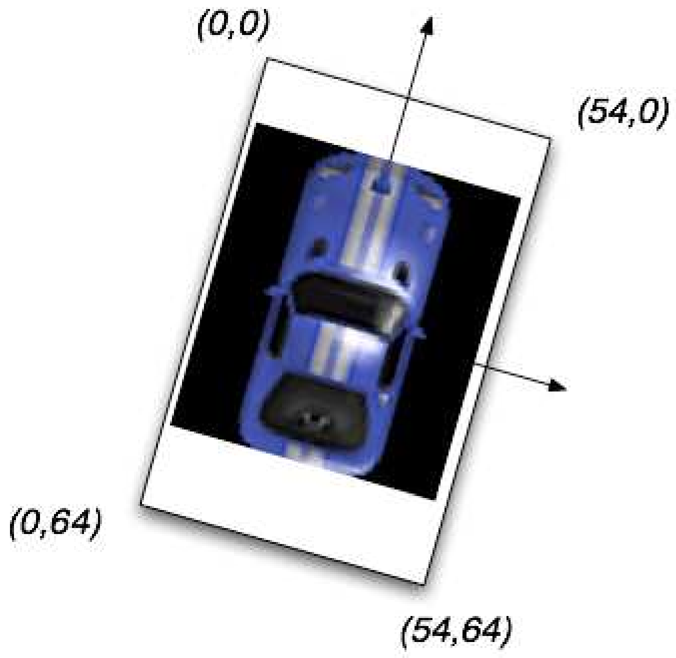

Figure 7: localized coordinates for mapping car sprite

If the local coordinates are outside boundaries of the car size (56x64), the output can simply be set 0, since the car does not span over those pixels.

Otherwise, if the local coordinates are indeed within range, using the localized coordinate, we can map the coordinate to a row in the ROM to get actual RGB data. For the image I used, I loaded a 54x64 bitmap onto the ROM, each row representing a 24-bit RGB value (using 54x64 = 3456 rows). The address is basically just:

*(localx + localy*56)*

where localx and localy are the local coordinates with respect to the top-left corner of the car.

3.5 Localize Pixel Module (Richard Chan)

The Localize Pixel Module is responsible for localizing (hcount, vcount) coordinates into a coordinate frame local to the car, using the car position and orientation.

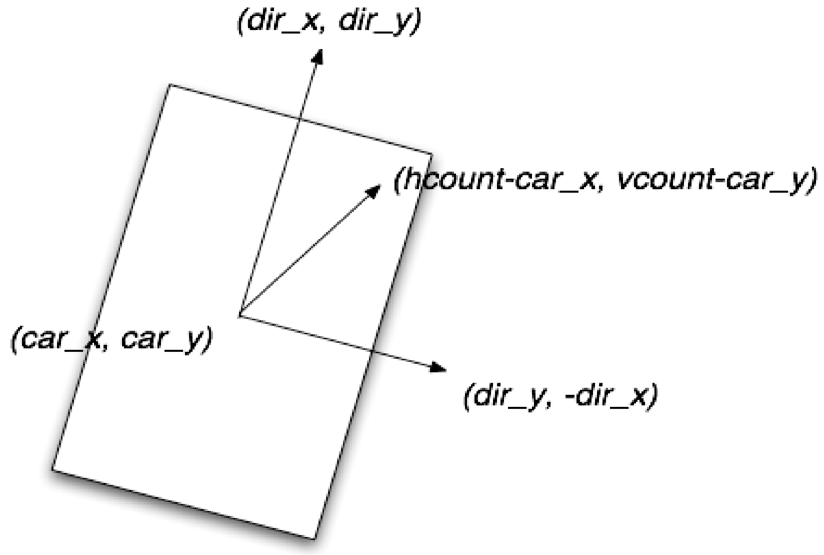

Figure 8: Localization of (hcount, vcount)

Basically, the Localize Pixel module takes the difference between (hcount, vcount) and the position of the car (car_x, car_y), giving us the vector from the center of the car to the pixel on the screen.

We know the orientation of the car (dir_x, dir_y), and a vector normal to it, which can be computed as (dir_y, -dir_x) (rotation by 90 degrees). The (dir_x, dir_y), as described in the previous section, always has length 256. Therefore, if we take the dot products of (hcount-car_x, vcount-car_y) with (dir_x, dir_y), it will return the component of the vector (hcount-car_x, vcount-car_y) in the direction of the car, multiplied by 256. Similarly, taking the dot products of (hcount-car_x, vcount-car_y) with (dir_y, -dir_x) will give us the component of the vector normal to the direction of the car multiplied by 256. Bit-shifting both components by 8 will give us the coordinates of (hcount, vcount) localized at (car_x, car_y) in the direction of the car.

While the algorithm is perfectly fine, since we are localizing one pixel per clock cycle under the 65MHz clock, it is also crucial to be able to finish computation within one clock cycle (in less than 15.38 ns). To achieve that, we can pipeline our operation. For example, my implementation would first store the products of (hcount-car_x) and (vcount_-car_y) with dir_x and dir_y into registers. Use another cycle to compute the dot product by summing up the products. And, finally one more cycle for shifting by 8 bits to output the correct coordinate.

In this setup, each input given to the Localize Pixel module will have the outputs ready after 3 clock cycles. In order to keep that synchronized with other parts of the system that depends on the current hcount, vcount, we can use as inputs in Localize Pixel the values of hcount and vcount 3 clock cycles in advance. To implement that, we have an xvga module that starts counting 3 clocks in advance. The signal from that will be sent to the Local Pixel to keep it in-sync.

4 Map Modules

4.1 Mouse Input Module (Fan Yang and Calvin Chung)

4.1.1 ps2_mouse Module (Fan Yang and 6.111 staff)

In our project, the ps2 mouse is used to draw the track. We used the ps2_mouse module provided by the 6.111 staff as a basis for implementing the mouse functionality. There are two major modifications to the ps2_mouse module. The first modification is that the speed of movement of the mouse is decreased by a factor of 4. The exact implementation is just a simple bit shift. This change was made because we want to guarantee that while drawing the track, the mouse doesn't move so fast relative to the screen refresh rate that the track becomes discontinuous. Various values were experimented, and a factor of 4 seems necessary and sufficient to guarantee that the track doesn't become discontinuous when the user moves the mouse fast. The second modification is that a 32.5 MHz clock is used instead of the 50MHz clock (discussed in detail later in the report).

4.1.2 Mouse_Div Module (by Calvin Chung)

One problem we encountered while using the mouse to draw out the track was that the mouse input was very jumpy. Turned out the clock speed of 65 MHz was too fast for the mouse. To solve the problem, a 32.5 MHz clock whose clock signal is simply inverted at every positive clock edge of the 65 MHz clock is built. A clock with double the period and half the frequency of the 65 MHz clock is created.

4.2 Map Module (by Calvin Chung)

4.2.1 Overview

The main function of the map module is to construct a double buffer with the ZBT SRAM chips so as to allow drawing of track while displaying it on the monitor. On top of that, the module is responsible for sharing ZBT memory as the memory is also used to store video camera data during play mode.

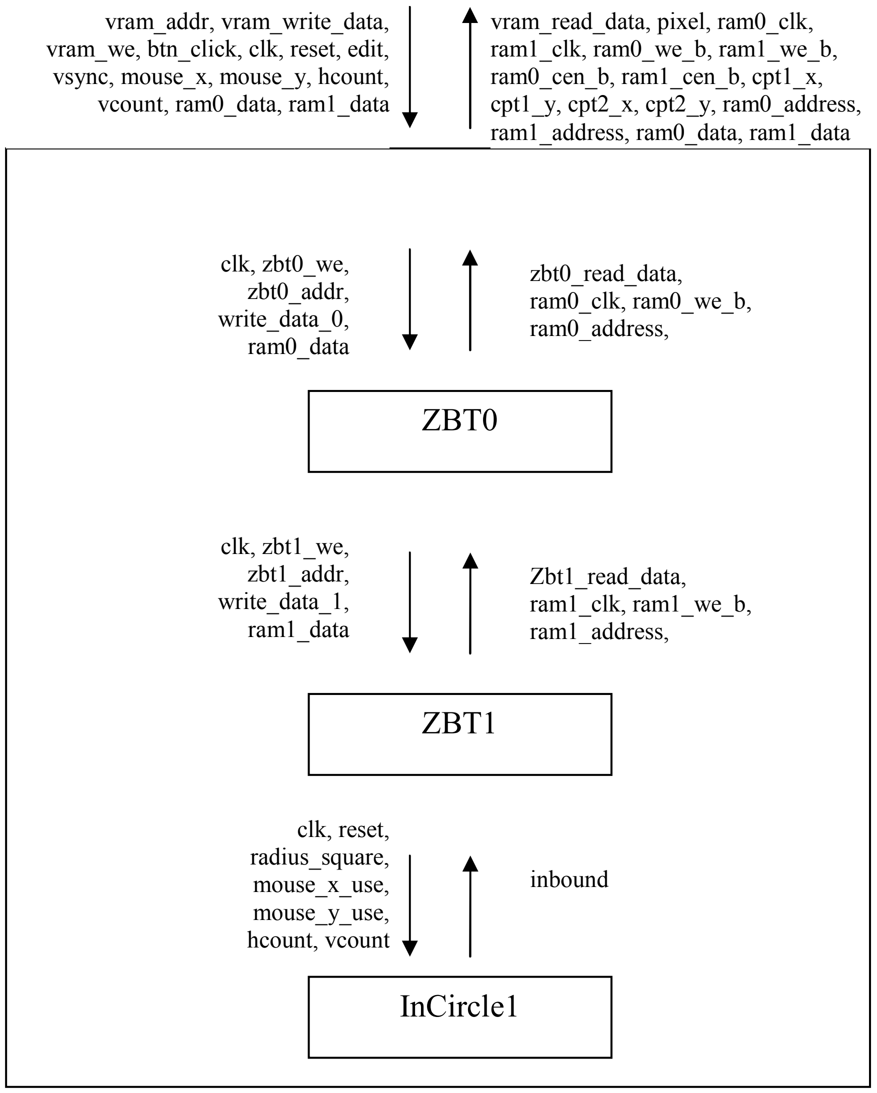
vram_addr, vram_write_data, vram_we, btn_click, clk, reset, edit, vsync, mouse_x, mouse_y, hcount, vcount, ram0_data, ram1_data
vram_read_data, pixel, ram0_clk, ram1_clk, ram0_we_b, ram1_we_b, ram0_cen_b, ram1_cen_b, cpt1_x, cpt1_y, cpt2_x, cpt2_y, ram0_address, ram1_address, ram0_data, ram1_data
clk, zbt0_we, zbt0_addr, write_data_0, ram0_data
zbt0_read_data, ram0_clk, ram0_we_b, ram0_address,
ZBT0
clk, zbt1_we, zbt1_addr, write_data_1, ram1_data
Zbt1_read_data, ram1_clk, ram1_we_b, ram1_address,
ZBT1
clk, reset, radius_square, mouse_x_use, mouse_y_use, hcount, vcount
inbound
InCircle1

Figure 9. Map Module Block Diagram

4.2.2 ZBT memory

A ZBT SRAM chip is a high-speed memory device (up to 167 MHz). On the backside of the PCB are two of these synchronized chips, with the memory size of each chip being 512k*36. While the ZBT memory has a high speed, it is very tricky to interface to. ZBT is a pipelined device, with its data bus being delayed by two cycles after address and control signals. That is, when a read cycle is being initiated at cycle n, data from address input will not be available until n+2 cycles. The same principle applies to writing data into ZBT memory. The pipelining complicates subsequent interfacing with the ZBT memory, as described in the following paragraphs. Also, ZBT memory doesn't have dual ports to read to and write from the same address concurrently. To solve this problem and to prevent glitches, a double buffer is built to allow simultaneous read-write processes. The 2-cycle delay property of a ZBT memory is illustrated in the following diagram:

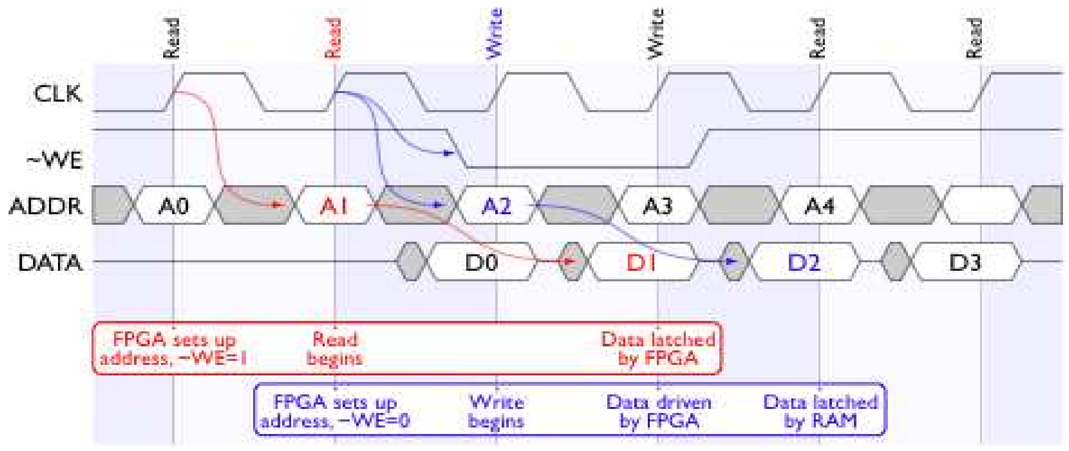

Figure 10. ZBT clock cycle diagram [1]

4.2.3 Double Buffer

The idea of a double buffer is to have two memories with identical structure to act as one memory device that supports read-write functionality during edit mode. In this system, the ZBT memory is used to store a 1-bit value for each pixel on the screen, indicating whether a track is drawn on that pixel. At any one point of time, one of the ZBT memories will be read and the other one will be written to. Let's say ZBT0 is being read and ZBT1 is being written to. As we provide ZBT0 with an address, we get the data corresponding to that address 2 clock cycles later. Data line corresponding to each address is 36 bits long, and in this module, we only use 32 bits of each line for reason described in the following address generator section. Each of these 32 bits corresponds to one certain pixel. If a track exists in that pixel, a value of "1" is stored in that bit.

At every rising edge of the clock, the module receives updated mouse-x and mouse-y positions from the mouse input. As described in the incircle module, the system places a circular blob on every mouse position input. This indicates where the new section of the track lies. Each bit of data from ZBT0 indicates whether there is a track at that pixel previously. From the incircle module we know whether a track is newly drawn at that pixel. Combing the information with an "OR" function, we know whether a track should exist in that pixel or not after updated mouse positions are being processed. The result is then stored into a write_data register, a register that stores the updated data to be written into memory. As described earlier, ZBT1 memory is being written into when ZBT0 is being read from. Every time a bit is updated and stored into the write_data register, it will be written into the ZBT1 memory. Thus after the pointer of the screen goes through the whole frame, ZBT1 contains the most updated positions of the whole track.

At frame refresh, the roles of ZBT0 and ZBT1 are swapped. With the most updated positions of the track, ZBT1 now acts as the memory being read. ZBT0 is now being written to. This swap occurs at every frame refresh, resulting in a continuous updating of track positions. As for the output of the track onto the screen, at every rising edge of the clock, the value of the bit being processed is displayed onto the screen. This value is obtained from the memory being read from, i.e. ZBT0 in our previous example. Through this process the read-write functionality of the double buffer is established.

4.2.4 Address Generator

Since each ZBT memory only has 512,000 addresses, it is not possible to allocate a unique address for each pixel. As a result, an address generator is built to match each address to multiple pixels. In this system, each address contains track values for 32 pixels to best utilize the memory while minimizing combinational delay.

Each pixel is first assigned a serial no according to the formula (vcount * 1344 + hcount). Each pixel thus has a unique serial no. This serial no is then divided by 32. The quotient equals the address line corresponding to that pixel, while the remainder represents the bit no in that address line. Originally a divider module was used to divide the serial no by 36 to maximize memory space, but it turned out that the combinational delay of the divider module exceeded our limitations. By dividing the serial no by 32, we can easily extract the quotient and remainder by shifting. This improves the system efficiency to a great extent and completes the task at the same time.

4.2.5 ZBT 2-cycle delay

As mentioned before, the ZBT memory has a 2-cycle delay in writing and reading data. Write/read addresses and data input/output have to be adjusted to account for this delay. The goal is to obtain old track data for a particular pixel at the same time when the new data for that pixel is being written into the memory. To prepare for this, the read address is being input two cycles earlier. The write data is also saved for 2 cycles before it is actually being read into the memory. Various pipelining issues further complicate the handling of the ZBT delay.

4.2.6 Checkpoints

As part of the design, users have to visit 2 checkpoints 7 times in total to finish the game. These checkpoints are created during the edit mode. While moving the mouse creates the track, clicking mouse buttons create the two checkpoints. Button click inputs from the mouse are sent to the map module every clock cycle. When a button is clicked, its value is set to “1”. Thus by detecting the rising edge of the button click signal, the corresponding mouse position can be recorded. In our design, the two checkpoints are set by clicking the left and right buttons of the mouse. Since the previous checkpoint is replaced if the mouse button is clicked again, users can freely move the checkpoints during edit mode. The x-y coordinates of the checkpoints are sent to the game logic module to keep track of the performance of the player during play mode.

4.2.7 Sharing ZBT memory

One major issue we encountered while implementing the design is the sharing of ZBT. In edit mode, both of the ZBT memories are being used as part of the double buffer. Each of the 2 memories alternates between write enabled and write disabled. In play mode, however, the video input modules need a ZBT memory to temporarily store the camera data to carry out the center of mass algorithm. Fortunately in play mode, the track is no longer being drawn and only one ZBT memory is needed to output the track.

In the map module, for each ZBT memory an instance of the zbt_6111 driver is created to interface the system with the ZBT by connecting data in registers with physical input/output to the ZBT memory chip. Originally another instance of the zbt_6111 is created in the top-level module to interface the camera data with the ZBT memory chip, resulting in two drivers for one of the two ZBT memories. A mux is then used to determine which driver to process data from based on whether the game is in play mode or in edit mode. This method turns out to be faulty most probably because ram_data is an inout port. In the end, the driver in the top-level module is eliminated. Instead, the addresses, write enable signal and data are passed from the video camera modules directly to the map module to be processed. By sharing the same driver, ZBT0 is used to store camera input data during play mode, while ZBT1 is used to output track data.

4.3 InCircle Module (Calvin Chung)

The incircle module is used to determine whether a point lies within the fixed radius of a given point. Within the system this module is used to determine whether a pixel is within a given radius of latest mouse position, in turn deciding whether the pixel is part of the track.

First the absolute difference between the respective x and y coordinates of the two points are being computed. This is done by first comparing the two numbers and then subtracting the smaller number from the larger one. The absolute differences in the x-axis and the y-axis are then squared and added up and compared with the square of the fixed radius. If the sum of squares is smaller than the square of the radius, the output signal will be set to 1. Otherwise it is set to 0.

4.4 Zbt_6111 Driver (6.111 Staff)

This module provides the interface to the ZBT memory chips. Taking in addresses, write enable signals, write data and physical data from ZBT, this module produces the necessary physical output to the ZBT and also data read from the memory. It takes care of the 2-cycle delay of writing into ZBT by storing data for 2 extra cycles within the module.

5. Output Components

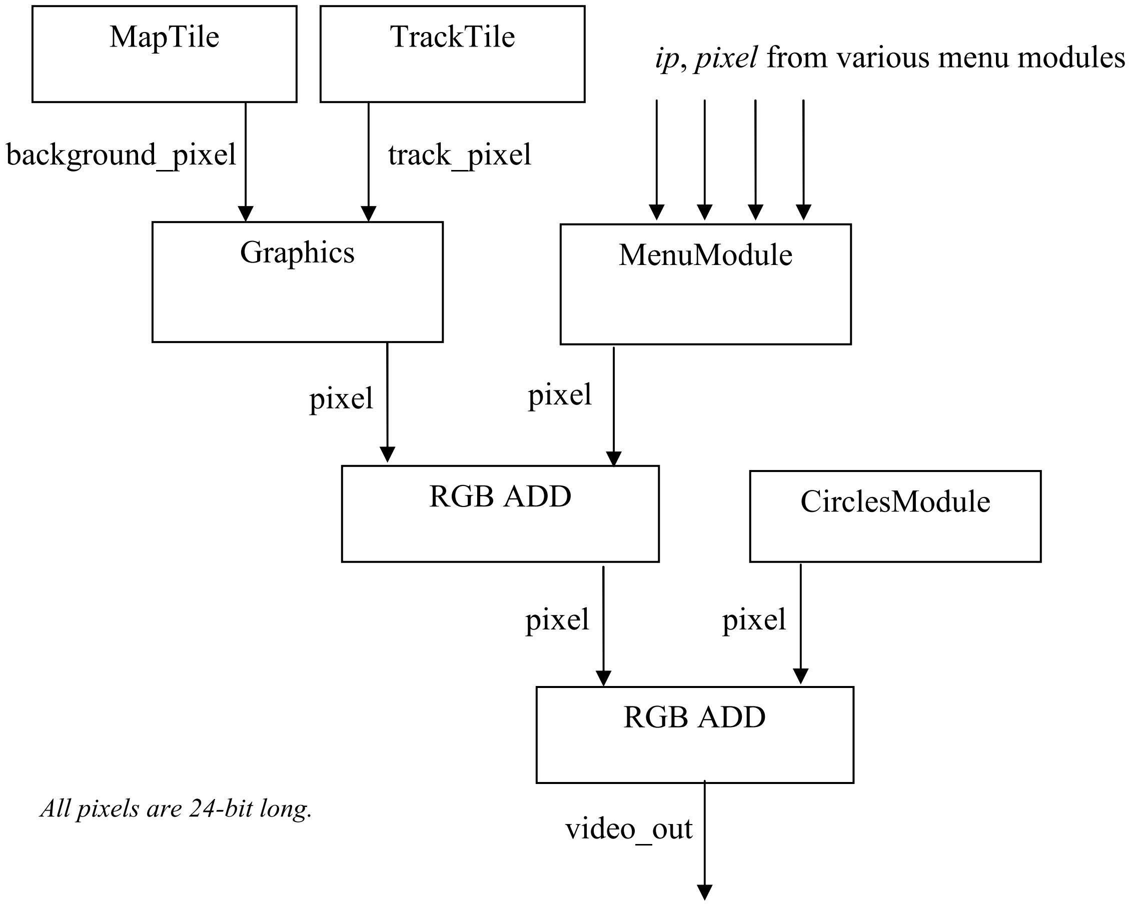

Figure 11: Output Modules: From Graphics to Video_Out.

The output modules are what actually generate the display to the screen. Basically, on the highest level, the Output Modules includes three parts: 1) The Graphics Module, which outputs to the entire screen according to the state of the game; 2) The Menu Module, which lays out various menus on top of the game graphics; 3) The Circles Module, which outputs where the checkpoints are located and to show the position of the cursor in *edit* mode.

5.1 Graphics Module as an Output Module (Richard Chan)

The functions of the Graphics Module as a Logic module has been described previously in Section 3.3. Other than the Car Sprite module, the Graphics Module must also produce the background and track pixels.

For aesthetical purposes, both the track and background will be generated using a ROM containing a bitmap of a track or background tile.

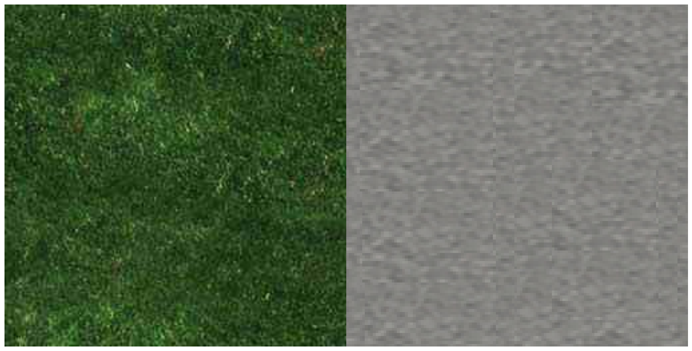

Figure 12: Tiles. Left: Background. Right: Track.

Both of these tiles are 128x128 in size, so the address for looking up pixels can simply depend on hcount and vcount. (e.g. the current implementation uses “hcount * 128 + vcount” as the address, which will simply tile the two images across the screen in a grid-like manner).

5.2 Menu Module

The Menu Module takes in a collection of different menu items (such as the speed gauge, ready screen, the timer, etc.). Each menu module has a “pixel” output providing the RGB value, and an “ip” output, which stands for “in-picture.” Modules that are in-picture for certain (hcount, vcount) coordinates means that those modules should be visible at those locations.

The Menu Module basically does a bit-wise OR on the outputs of the various RGB values coming out of the menu modules if any one of their in-picture signal is on. The system is designed such that no modules overlap, so when the ip values are high, the result of doing an OR over all the RGB values will give us the one RGB value generated by the menu that has ip set to high.

Note that we are doing a bit-wise OR for optimization purposes. The same result may be obtained using a series of Multiplexers over the ip values but will require significantly higher delay. Keeping the delay low is crucial for having a completely glitch-less display.

5.2.1 Cstringdisp Module (6.111 Staff and Calvin Chung)

This module is responsible for generating the characters to be displayed on the screen. Code used here is almost the same as the one provided on the website, except that RGB output values are changed from 3-bit to 24-bit.

5.2.2 Finish_Game module (Calvin Chung)

This module displays a finish screen showing "Good Job!" when the game is over. Once the input finish_game signal is set to "1", the finish screen is enabled and shows up on top of the track. The character display is done through the cstringdisp module.

5.2.3 Lap_Disp Module (Calvin Chung)

This module is responsible for displaying the number of checkpoints visited by the user. Receiving a 3-bit lap_num signal from the game logic module, this module displays the number together with the characters "CPT #" onto the screen through the cstringdisp module. The default lap_num is 0 and the maximum lap_num allowed is 7.

5.2.4 Lights Module (Calvin Chung)

This module controls the traffic lights at the left side of the interface, which consists of four circles in additional to a background. As ready_screen module sends in a ready_done signal, the module starts counting the number of frame refresh. For every 60 frame refresh (1 second) within the first 240 refresh (4 seconds), a different light is turned on. Three red lights from top of bottom are first turned on and turned off in series each with a period of 1 second, while a green light at the very bottom lights up after 3 seconds and stays on for the rest of the game. Incircle instances are used to determine whether a pixel is within the radius of a certain circle. As count register reaches 239, the green light is turned on for a second and the start_game signal is set to "1". This signal is output to the game logic module which in turns enables car motion.

5.2.5 Ready _Screen Module (Calvin Chung)

This module displays a ready screen as the player enters play mode to allow time for the player to get ready. Once the player hits the button to transit the game from edit mode into play mode, the module starts counting the number of frame refresh. Until count register reaches 180 (3 seconds), a ready screen showing "Get Ready Hacker" is displayed on top of the track through the cstringdisp module. After 3 seconds, the ready screen disappears and the "ready_done" signal is set to "1". This starts off the counting in the lights module and eventually the game.

5.2.6 Timer Modules (Calvin Chung)

The timer modules are responsible for generating the timer display on the top right hand corner of the display screen. There are two types of timers built to implement this function. The simpler type of timer is responsible for the two decimal places to the left of the decimal point. The module constantly keeps track of the number of frame refresh before the display number is incremented by one. E.g. to display the unit second requires an increment every 65 times the frame is refreshed. To accurate keep track of a tenth of a second, another type of timer which counts the no of clock cycles instead of frame refresh is needed. For example, to display one-tenth of a second requires an update of the display once every 6480000 clock cycles.

5.2.7 Title Module (Calvin Chung)

The title module displays the title "Hacker's Trail" on the top of the screen at all times. This is done through inputting the corresponding x-y positions and characters in binary ASCII mode to the characters display module (cstringdisp).

5.2.8 Speed Gauge (Richard Chan)

The speed gauge displays the speed in a vertical bar near the bottom left corner of the screen. The height of the bar depends on the speed of the car. The color will also change depending on how fast the car is going.

5.3 Circles Module (Richard Chan)

The Circles Module is basically the circles generated by the checkpoints and the current mouse position. In edit mode, a circular blob will be generated on the screen to show where the current mouse position is, along with the checkpoints previously set by clicking the left and right mouse buttons. In play mode, depending on which checkpoint is next, the Circles Module will highlight the next checkpoint on the screen to notify the user where to go to next.

The Circles Module takes in as input which blobs cover (hcount, vcount). It then chooses, according to the mode, what the output.

5.4 RGB ADD Module (Richard Chan)

The RGB ADD Module is similar to doing alpha composing on the RGB inputs, except it assumes that the RGB values the inputs provide have been pre-multiplied by their alpha values.

What the module does is basically summing up the red, green and blue components separately and outputting the result. If the sum goes over 255, the values will simply be bounded at 255.

The end result of using this module is the system we see now: with semi-transparent menus and highlighted checkpoints.

6. Testing and Debugging

The modules were tested individually by simulation to ensure that they work according to the specification integrating them into the system.

6.1 Input Components

For the modules in the input component such as the mouse module and camera module, testing was performed by actually testing against the inputs and observing the outputs.

6.1.1 Color Detection

Camera pixels in YCrCb format are converted to RGB to be displayed on the screen. This will display the camera pixels on the screen so that debugging could be done on the color detection. Also, in testing the color detection, I colored the pixels that I detected as "pixels of interest" with a different color so that I can see which pixels are recognized with some threshold of YCrCb.

6.1.2 Hand Finder

With the pixels of interests in a different color, it becomes clear where the noise is coming from, which false positives get picked up. In checking the center of mass calculation, I use crosshair to indicate the center of mass calculated by my hand finder module. Using crosshair in conjunction with the colored pixels, it makes the debugging process significantly more straightforward.

6.2 Game Logic

Since my responsibility includes building a Graphics module that displays the car onto the screen, which was what I created first. Even though, initially, the car is simply the rectangular Blob provided by Lab 5 in the Pong game, having video display to guide my debugging process definitely helped a lot.

6.2.1 Signed/Unsigned Error

Initially, the code I wrote for converting pixels into local coordinates has some signed/unsigned conversion errors. For instance, after a value has grown so large such that the Most Significant Bit becomes a 1, somewhere in the calculations the number gets treated as a 2-Complement, a values much larger than expected.

Since that error only happens sometimes, it was quite annoying to debug, but was eventually fixed by simply printing out each arithmetic calculation's result onto the hex display. (Displaying signed values onto the display will simply treat the numbers as unsigned values, which is perfectly fine for debugging as I would simply like to see what values are stored in the registers).

6.2.2 Sprite Resizing Error

As previously mentioned in Section 3.2, there was a strange resizing bug introduced by the small errors matrix multiplication over irrational numbers causes. Since the errors were so small, I did not expect them to have that much of an effect. However, it seems that, as long as the car stay in some orientation, it will continuously resize as the error grows exponentially, which makes sense since the localization of pixels method works by taking the dot products with the orientation vector and its normal.

As I also briefly mentioned in Section 3.2, I attempted a few methods to solve that bug and eventually settle on using a ROM to pre-compute all the values we need.

6.2.3 Pixel Localization Delay Error

The Pixel Localization has not always been pipelined. However, without the pipelining, I have noticed that, quite often, a number of vertical lines would randomly appear on the screen near the car sprite.

The VGA monitor is a great tool for debugging. When we see situations such as the one described above, it is not difficult to guess that excessive delay could be the cause. Considering the number of bits the module has to multiply and add to compute the dot products, it is not surprising that the delay may go over one clock cycle, therefore causing unexpected behaviors to occur.
In order to fix that problem, as what the current implementation is doing (as described by Section 3.5), we can pipeline the operation into a few cycles: e.g. 1st, take the products, 2nd, add the products up accordingly into the dot products, then, lastly, bit-shift and return. After the pipelining is added, no more glitches are detected, so it is safe to assume that the bug has been fixed.

6.3 Map Module

To start off I built the incircle module to test the delay of building a circular blob around a mouse position. The combinational delay turned out to be not that serious and everything worked fine.

To test out the double buffer built with ZBT memory chips, I sent in pre-set mouse x and mouse y values to test if a circle was created around the mouse position during edit mode and whether the circle stayed when I switched to play mode.

The first major problem I had was that I wasn't interfacing with the ZBT correctly. I started my coding based on the zbt_6111 driver provided. At one point the circle showed up when I inverted all the write enabled signals. I thought I got it working, but it turned out that the data passed through the ZBT without being stored into it. After series of elimination and isolating testing, I realized that I didn't assign the default ZBT physical data inputs/outputs in the labkit correctly.

The next step was to actually test if my design was working. At first I didn't consider the delay of ZBT, assuming that the delay wouldn't cause too much of a problem except for a few glitches. It turned out that the delay actually messed up the whole system, giving me blank screens within milliseconds after loading the program onto the labkit.

Reconsidering the delay problems, I finally got the circle to show up on the monitor. However, there are thin lines at fixed distances from each other. It turned out that my read and write cycles were still 1 cycle off.

Another problem I had was that my system had a non-negligible combinational delay, creating blurred lines on the left end of the monitor. I had been using a divider module to generate addresses and bit numbers for each pixel serial no, but clearly it was not the best idea. Instead, I should have just divided the serial no by a power of 2 and the division could be done by shifting. I didn't realize this until 10 days before check off.

After solving all these problems I used the mouse to test my integrated system. I had always been inputting to the mouse a 65 MHz clock signal, assuming that the frequency's within the range of a ps2 mouse. However, every time I used the mouse, it would work fine for the first 10 seconds, but after that the cursor started jumping and getting out of control. After a day of testing, I decided to divide up the clock to create a 32.5MHz clock. With this lower frequency the clock worked right away.

6.4 Integration Process

We started off integration process by combining the game logic module and map module to test out the graphics collision detection. We tested this just by observing the output on the monitor. Everything worked fine except we inverted the collision algorithm, but that was a minor issue that was easily fixed. We then spent a few more days working on rotation perfection.

The video input was the last thing we integrated into the system. While there were some minor integration issues between the game logic module and the video input, that was quickly resolved. One bigger problem we faced was that the combinational delay was more than we could take on. The video input seemed to be lagging as observed from the video output. After we pipelined the video input modules and the graphics output process, the system performed much better.

With the remaining time, we improved our project by adding a user interface. Creating different menu functions and loading images from the ROM as track and background, the project was a lot more presentable and was ready to go.

CONCLUSION

In summary, we have successfully built a racing game unlike any other. We can detect hand positions with relatively high accuracy, which in turn gives us pretty good control over maneuvering the vehicle. The user interface is decent if not superb, with a graphical background and track, along with semi-transparent menus.

The ZBT double buffer is working as expected without any noticeable flaw. The track can be displayed while being drawn concurrently. The ZBT memory is successfully sharing between the Double Buffer and the Video Input Cache.

The rotation algorithm is nearly perfect. The sprites are mapped perfectly onto the locations of the vehicles using the pre-computed angle method. Although it was originally intended to be able to compute angles perfectly in real-time using matrix multiplication, the performance gain form using the ROM clearly outweighs the original method.

There is however, clearly more room for improvement. Our system can easily be scaled to support more than one player – if there is no bottleneck on only being able to use one camera, since the viewing angle of a camera clearly limits the amount of people who can simultaneously participate.

Acknowledgments
We would like to thank Professor Terman, TA Roberto Carli, as well as Jim Hom for their unconditional help and support with our project.

REFERENCES

[1] Chris Terman, 6.111 Labkit, [Online Document], Dec. 2007, [2007 Dec 14],
Available HTTP:
http://web.mit.edu/6.111/www/f2007/index.html

APPENDIX A –Angle ROM Generator

```
public static void main(String[] args) {

        for (int i=0; i<128; i++) {
           double angle = ((double)i)*360*Math.PI/180/128;
           double nextx = 256 * Math.cos(angle);
           double nexty = 256 * Math.sin(angle);

           System.out.print(getBinary((int) (nextx*Math.pow(2, 16))).substring(0,16));
           System.out.print(getBinary((int) (nexty*Math.pow(2, 16))).substring(0,16));
           if (i<360) System.out.println(",");
        }
    }

    private static String getBinary(int x) {
        String str = Integer.toBinaryString(x);
        while (str.length() < 32)
           str = "0" + str;
        return str;
    }
```

APPENDIX B—Verilog Code

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//          value. (Previous versions of this file declared this port to
//          be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//          actually populated on the boards. (The boards support up to
//          72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
///////////////////////////////////////////////////////////////////////////////

module labkit (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;

inout  mouse_clock, mouse_data;
       input  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

// VGA Output
//assign vga_out_red = 8'h0;
//assign vga_out_green = 8'h0;
//assign vga_out_blue = 8'h0;
//assign vga_out_sync_b = 1'b1;
//assign vga_out_blank_b = 1'b1;
//assign vga_out_pixel_clock = 1'b0;
//assign vga_out_hsync = 1'b0;
//assign vga_out_vsync = 1'b0;

// 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;
//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
//assign ram0_data = map_data;
//assign ram0_address = map_address;
assign ram0_adv_ld = 1'b0;
//assign ram0_clk = map_clk;
// assign ram0_cen_b = map_cen_b;
assign ram0_ce_b = 1'b0;
assign ram0_oe_b = 1'b0;
        //assign ram0_we_b = map_we_b;
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;
//assign ram1_cen_b = 1'b0;
assign ram1_ce_b = 1'b0;
assign ram1_oe_b = 1'b0;
//assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'h0;
        assign clock_feedback_out = 1'b0;
//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;

// 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));

        ////////////////// ALIASING AND DEBOUNCING //////////////////

        wire clk,
                        play_mode,   // 1 if in play mode
                        is_out_of_track; // 1 if some part of car is out of track

assign clk = clock_65mhz;

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

```
SRL16 reset_sr (.D(1'b0), .CLK(clock_65mhz), .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, clock_65mhz, ~button_enter, user_reset);
assign reset = user_reset | power_on_reset;

// keeping states of the game
        wire finish_game;
        wire start_game;
        wire editbtn; // stops edit mode and goes into play mode
        reg edit;
debounce db2(power_on_reset, clock_65mhz, ~button3, editbtn);
// simple mux so the start button can only be used once
        always @(posedge clk)
        begin
                if (reset)
                        edit <= 1;
                else
                        if (editbtn)
                                edit <= 0;
        end

        // play mode is after game is started and before game finished
        assign play_mode = start_game && (~finish_game);

        wire [15:0] angle, height;

        ////////////////// VIDEO SIGNALS //////////////////

        // generate basic XVGA video signals
        // (and the future counts in case there is delay for pipelining)
wire [10:0] hcount, hcount_next1, hcount_next2, hcount_next3;
wire [9:0]  vcount, vcount_next1, vcount_next2, vcount_next3;
wire hsync,vsync,blank;
wire hsync1,vsync1,blank1,hsync2,vsync2,blank2,hsync3,vsync3,blank3;
xvga xvga0(clock_65mhz, hcount, vcount, hsync,vsync,blank);
xvga xvga1(clock_65mhz, hcount_next1, vcount_next1, hsync1,vsync1,blank1);
xvga xvga2(clock_65mhz, hcount_next2, vcount_next2, hsync2,vsync2,blank2);
xvga xvga3(clock_65mhz, hcount_next3, vcount_next3, hsync3,vsync3,blank3);
defparam xvga1.OFFSET = 1;
defparam xvga2.OFFSET = 2;
defparam xvga3.OFFSET = 3;

        ////////////////// MOUSE //////////////////

        wire clock_325mhz;
        mouse_div mouse_div1(clock_65mhz, clock_325mhz);

        wire [11:0] mouse_x,mouse_y;
wire [2:0]  btn_click;
ps2_mouse_xy m1(clock_325mhz, reset, mouse_clock, mouse_data, mouse_x, mouse_y, btn_click);

        ////////////////// GAME MODULE //////////////////
```

```
 // game module
  wire [15:0] car_x, car_y;
  wire signed [15:0] car_orientation_x, car_orientation_y;
  wire [7:0] speed;
         wire is_at_cpt1, is_at_cpt2;
         wire[2:0] lapnum;
         wire [15:0] simulated_angle;
         // using the angles as binary should be fine for now
         // camera is not accurate enough to have previse turning
         assign simulated_angle = (angle == 0) ? 1800 :
                                                                        (angle
== 1) ? 2700 : 900;
  gamemodule gamemodule1
   (clock_65mhz, reset, vsync,
   simulated_angle, height,
   play_mode,
   is_out_of_track, is_at_cpt1, is_at_cpt2,
   car_x, car_y, car_orientation_x, car_orientation_y,
          speed,
          lapnum);


         ////////////////// MENU ITEMS //////////////////
          // menu items
         wire ready_ip, ready_done;
         wire[23:0] ready_rgb;
  ready_screen ready_screen1(clock_65mhz, edit, hcount, vcount, vsync, reset, ready_ip, ready_rgb,
ready_done);

         wire [23:0] lap_rgb;
         wire lap_ip;
         lap_disp lap_disp1(clock_65mhz, reset, hcount, vcount, vsync, lapnum, lap_rgb, lap_ip);

         wire[23:0] lights_rgb;
         wire light_ip;
         lights lights1(reset, clock_65mhz, ready_done, vsync, hcount, vcount, lights_rgb, light_ip,
start_game);

         wire title_ip;
         wire[23:0] title_rgb;
         title title1(clock_65mhz, reset, hcount, vcount, vsync, title_ip, title_rgb);

         wire timer1_ip;
  wire[23:0] timer1_rgb;
  timer timer1(clock_65mhz, reset, hcount, vcount, 11'd900, 10'd80, 13'd599, vsync, play_mode, timer1_ip,
timer1_rgb);

  wire timer2_ip;
  wire[23:0] timer2_rgb;
  timer timer2(clock_65mhz, reset, hcount, vcount, 11'd920, 10'd80, 13'd59, vsync, play_mode, timer2_ip,
timer2_rgb);

  wire colon_ip;
  wire[23:0] colon_rgb;
  colon_display colon1(clock_65mhz, reset, hcount, vcount, 11'd940, 10'd80, colon_ip, colon_rgb);
```

```
   wire timer_lsb_ip;
   wire[23:0] timer_lsb_rgb;
   timer_lsb timer_lsb(clock_65mhz, reset, hcount, vcount, 11'd960, 10'd80, vsync, play_mode,
timer_lsb_ip, timer_lsb_rgb);

            wire [23:0] speedpixel;
            wire speed_ip;
            speedgauge speedgauge1(clock_65mhz, vsync, speed, hcount, vcount, speed_ip, speedpixel);

            wire finish_ip;
            assign finish_game = (lapnum == 7);
            wire[23:0] finish_rgb;
            finish_game finish1(clock_65mhz, reset,finish_game, hcount, vcount, vsync, finish_ip,
finish_rgb);
            // edit => ready_done => start_game => finish_game

            assign led[7:0] = speed[7:0];

            ////////////////// MAP + CIRCLES //////////////////

            wire zbt0_we;
            wire inbound;
            wire track;
            wire[11:0] cpt1_x;
            wire[11:0] cpt1_y;
            wire[11:0] cpt2_x;
            wire[11:0] cpt2_y;
            wire[35:0] vram_read_data;
            wire[35:0] vram_write_data;
            wire vram_we;
            wire[18:0] vram_addr;

            Map Map1(vram_addr, vram_write_data, vram_read_data, vram_we, btn_click, clock_65mhz,
reset, edit, vsync, mouse_x, mouse_y, hcount, vcount, track, ram0_clk, ram1_clk, ram0_we_b, ram1_we_b,
ram0_cen_b, ram1_cen_b, ram0_address, ram1_address, ram0_data, ram1_data, cpt1_x, cpt1_y, cpt2_x,
cpt2_y);

            wire cir_blob;
   incircle incircle2(clock_65mhz, reset, 21'd2500, mouse_x, mouse_y, hcount, vcount, cir_blob);

            wire cpt1blob;
            incircle cpt1circle(clock_65mhz, reset, 21'd2500, cpt1_x, cpt1_y, hcount, vcount, cpt1blob);
            wire cpt2blob;
            incircle cpt2circle(clock_65mhz, reset, 21'd2500, cpt2_x, cpt2_y, hcount, vcount, cpt2blob);

            ////////////////// GRAPHICS MODULE  //////////////////

             // GRAPHICS module
   wire [23:0] pixel;
   wire phsync,pvsync,pblank;
            // wire is_at_cpt1, is_at_cpt2;
   graphics graphics1(clock_65mhz, reset,
     car_x, car_y, car_orientation_x, car_orientation_y,
     track, cpt1blob, cpt2blob,
     hcount, hcount_next1, hcount_next2, hcount_next3,
 vcount, vcount_next1, vcount_next2, vcount_next3,
 hsync,vsync,blank,
 phsync,pvsync,pblank,pixel,
 is_out_of_track, is_at_cpt1, is_at_cpt2);

      wire [10:0] x_left, x_right;
      wire [9:0] y_left, y_right;

      reg old_is_out_of_track;
      reg old_vsync;

      ////////////////// ZBT AND CAMERA //////////////////

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

vram_display vd1(reset,clock_65mhz,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;
ntsc_to_zbt n2z (clock_65mhz, tv_in_line_clock1, fvh, dv, {ycrcb[19:11], ycrcb[9:1]},
                 ntsc_addr, ntsc_data, ntsc_we, switch[6]);

// code to write pattern to ZBT memory
reg [31:0]      count;
always @(posedge clock_65mhz) count <= reset ? 0 : count + 1;

wire [18:0]      vram_addr2 = count[0+18:0];
wire [35:0]      vpat = ( switch[4] ? {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[1:0]==2'd2) : blank;
```

```
  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

  //reg [17:0]      pixel;
  //wire  b,hs,vs;

  //delayN dn1(clock_65mhz,hsync,hs);       // delay by 3 cycles to sync with ZBT read
  //delayN dn2(clock_65mhz,vsync,vs);
  //delayN dn3(clock_65mhz,blank,b);

        hand_finder hand_xy_left(clock_65mhz, reset, hcount, vcount, vsync, vr_pixel, x_left, y_left);
        //blue is left
        defparam hand_xy_left.CR_MAX = 230;
        defparam hand_xy_left.CR_MIN = 160;
        defparam hand_xy_left.CB_MAX = 350;
        defparam hand_xy_left.CB_MIN = 260;


        hand_finder hand_xy_right(clock_65mhz, reset, hcount, vcount, vsync, vr_pixel, x_right, y_right);
        //red is right
        defparam hand_xy_right.CR_MAX = 500;
        defparam hand_xy_right.CR_MIN = 330;
        defparam hand_xy_right.CB_MAX = 260;
        defparam hand_xy_right.CB_MIN = 220;

        //wire [15:0] angle, height;
        hand_logic hand_logic1(clock_65mhz, x_left, y_left, x_right, y_right, angle, height);

        wire [23:0] camera_rgb;
        wire camera_ip;
       camera_corner_display camera_corner_disp1(clock_65mhz, hcount, vcount, x_left, y_left, x_right,
y_right, camera_ip, camera_rgb);

        ////////////////// UI //////////////////

        wire [23:0] menu_rgb;
        menuModules menuModules1(clock_65mhz, ready_ip, light_ip, lap_ip, title_ip, camera_ip,
timer1_ip, timer2_ip, colon_ip, timer_lsb_ip, finish_ip, speed_ip,menu_rgb);

        wire [23:0] circle_rgb;
        circlesModule circlesModule2(clock_65mhz, edit, cir_blob, cpt1blob, cpt2blob, ~lapnum[0],
circle_rgb);

        wire [23:0] rgb1, rgb2;
        rgbadd rgbadd1(clock_65mhz, pixel, menu_rgb, rgb1);
        rgbadd rgbadd2(clock_65mhz, rgb1, circle_rgb, rgb2);
```

```
reg [23:0] rgb;
reg b,hs,vs;
always @(posedge clock_65mhz) begin
                        if (vsync && ~old_vsync)
                                    old_is_out_of_track <= is_out_of_track;
  if (switch[1:0] == 2'b01) begin
  // 1 pixel outline of visible area (white)
  hs <= hsync;
  vs <= vsync;
  b <= blank;
  rgb <= (hcount==0 | hcount==1023 | vcount==0 | vcount==767) ? {24{1'b1}} : 0;
  end else if (switch[1:0] == 2'b10) begin
  // color bars
  hs <= hsync;
  vs <= vsync;
  b <= blank;
  rgb <= hcount;
  end else begin
    // default: game
    hs <= hsync;
    vs <= vsync;
    b <= blank;

                                    // normal
                                    rgb <= rgb2;
                                    old_vsync <= vsync;

  end
end

// VGA Output.  In order to meet the setup and hold times of the
// AD7125, we send it ~clock_65mhz.
assign vga_out_red = rgb[23:16];
assign vga_out_green = rgb[15:8];
assign vga_out_blue = rgb[7:0];
assign vga_out_sync_b = 1'b1;    // not used
assign vga_out_blank_b = ~b;
assign vga_out_pixel_clock = ~clock_65mhz;
assign vga_out_hsync = hs;
assign vga_out_vsync = vs;


            // display for debugging purposes:
// x y dirx diry
wire[63:0] datad;
assign datad[15:0] = car_x[15:0];
            assign datad[31:16] = car_y[15:0];
assign datad[47:32] = car_orientation_x;
assign datad[63:48] = car_orientation_y;
display_16hex disp(!button_left, clk,
                        datad,
                        disp_blank, disp_clock, disp_rs, disp_ce_b,
                        disp_reset_b, disp_data_out);
```

```
endmodule

module menuModules(clock_65mhz, ready_ip, light_ip, lap_ip, title_ip, camera_ip, timer1_ip, timer2_ip,
colon_ip, timer_lsb_ip, finish_ip, speed_ip,menu_rgb);
  input ready_ip, light_ip, lap_ip, title_ip, camera_ip, timer1_ip, timer2_ip, colon_ip, timer_lsb_ip,
finish_ip, speed_ip;
  input clock_65mhz;
  input[23:0] ready_rgb, lights_rgb, lap_rgb, title_rgb, camera_rgb, timer1_rgb, timer2_rgb, colon_rgb,
timer_lsb_rgb, finish_rgb, speedpixel;
  output[23:0] menu_rgb;
  reg[23:0] menu_rgb;

  always @(posedge clock_65mhz) begin
    if (ready_ip | light_ip | lap_ip | title_ip |camera_ip | timer1_ip | timer2_ip | colon_ip | timer_lsb_ip |
finish_ip | speed_ip)
              menu_rgb <= ready_rgb | lights_rgb | lap_rgb | title_rgb |camera_rgb|timer1_rgb | timer2_rgb |
colon_rgb | timer_lsb_rgb | finish_rgb | speedpixel;
    else
      menu_rgb <= 0;
  end

endmodule

module circlesModule(clock_65mhz, edit, cir_blob, cpt1blob, cpt2blob, nextcptisone, circle_rgb);
  input clock_65mhz, edit, cir_blob, cpt1blob, cpt2blob, nextcptisone;
  output[23:0] circle_rgb;
  reg[23:0] circle_rgb;

  always @(posedge clock_65mhz) begin
    if (edit)
         begin
      if (cir_blob)
        circle_rgb <= { {8{1'b1}}, {8{1'b1}}, {8{1'b0}} };
                else if (cpt1blob)
                        circle_rgb <= { {8{1'b0}}, {8{1'b0}}, {8{1'b1}} };
                else if (cpt2blob)
                        circle_rgb <= { {8{1'b1}}, {8{1'b0}}, {8{1'b0}} };
                else
                        circle_rgb <= 0;
         end
         else
         begin
                if (nextcptisone)
                        if (cpt1blob)
                                circle_rgb <= { {8{1'b0}}, {8{1'b0}}, {8{1'b1}} };
                        else
                                circle_rgb <= 0;
                else
                        if (cpt2blob)
                                circle_rgb <= { {8{1'b0}}, {8{1'b0}}, {8{1'b1}} };
                        else
                                circle_rgb <= 0;
         end
         end
```

```
endmodule

//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode and adv7185init modules
//
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

/////////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

  // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
  // reset - system reset
  // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
  // ycrcb - 24 bit luminance and chrominance (8 bits each)
  // f - field: 1 indicates an even field, 0 an odd field
  // v - vertical sync: 1 means vertical sync
  // h - horizontal sync: 1 means horizontal sync

  input clk;
  input reset;
  input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
  output [29:0] ycrcb;
  output f;
  output v;
  output h;
  output data_valid;
  // output [4:0] state;

  parameter        SYNC_1 = 0;
  parameter        SYNC_2 = 1;
  parameter        SYNC_3 = 2;
  parameter        SAV_f1_cb0 = 3;
  parameter        SAV_f1_y0 = 4;
  parameter        SAV_f1_cr1 = 5;
  parameter        SAV_f1_y1 = 6;
  parameter        EAV_f1 = 7;
  parameter        SAV_VBI_f1 = 8;
  parameter        EAV_VBI_f1 = 9;
  parameter        SAV_f2_cb0 = 10;
  parameter        SAV_f2_y0 = 11;
  parameter        SAV_f2_cr1 = 12;
  parameter        SAV_f2_y1 = 13;
  parameter        EAV_f2 = 14;
  parameter        SAV_VBI_f2 = 15;
  parameter        EAV_VBI_f2 = 16;




  // In the start state, the module doesn't know where
  // in the sequence of pixels, it is looking.

  // Once we determine where to start, the FSM goes through a normal
  // sequence of SAV process_YCrCb EAV... repeat

  // The data stream looks as follows
  // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV sequence
  // There are two things we need to do:
  //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
  //   2. Decode the subsequent data

  reg [4:0]          current_state = 5'h00;
  reg [9:0]          y = 10'h000;  // luminance
  reg [9:0]          cr = 10'h000; // chrominance
  reg [9:0]          cb = 10'h000; // more chrominance

  assign  state = current_state;

  always @ (posedge clk)
    begin
          if (reset)
            begin

            end
          else
            begin
              // these states don't do much except allow us to know where we are in the stream.
              // whenever the synchronization code is seen, go back to the sync_state before
              // transitioning to the new state
              case (current_state)
                SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
                SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
                SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                                          (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                                          (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                                          (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                                          (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                                          (tv_in_ycrcb == 10'h368) ? EAV_f2 :
                                          (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                                          (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

                SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y0;
                SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cr1;
                SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y1;
                SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cb0;
```

```
                SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y0;
                SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cr1;
                SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y1;
                SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cb0;

                // These states are here in the event that we want to cover these signals
                // in the future. For now, they just send the state machine back to SYNC_1
                EAV_f1: current_state <= SYNC_1;
                SAV_VBI_f1: current_state <= SYNC_1;
                EAV_VBI_f1: current_state <= SYNC_1;
                EAV_f2: current_state <= SYNC_1;
                SAV_VBI_f2: current_state <= SYNC_1;
                EAV_VBI_f2: current_state <= SYNC_1;

              endcase
            end
    end // always @ (posedge clk)

   // implement our decoding mechanism

   wire y_enable;
   wire cr_enable;
   wire cb_enable;

   // if y is coming in, enable the register
   // likewise for cr and cb
   assign y_enable = (current_state == SAV_f1_y0) ||
                     (current_state == SAV_f1_y1) ||
                     (current_state == SAV_f2_y0) ||
                     (current_state == SAV_f2_y1);
   assign cr_enable = (current_state == SAV_f1_cr1) ||
                      (current_state == SAV_f2_cr1);
   assign cb_enable = (current_state == SAV_f1_cb0) ||
                      (current_state == SAV_f2_cb0);

   // f, v, and h only go high when active
   assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

   // data is valid when we have all three values: y, cr, cb
   assign data_valid = y_enable;
   assign ycrcb = {y,cr,cb};

   reg        f = 0;

   always @ (posedge clk)
    begin
        y <= y_enable ? tv_in_ycrcb : y;
        cr <= cr_enable ? tv_in_ycrcb : cr;
        cb <= cb_enable ? tv_in_ycrcb : cb;
        f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
    end

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////
// Register 0
///////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                  4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

///////////////////////////////////////////////////////////////////////////////
// Register 1
///////////////////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY               2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE        1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT          1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING         1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                     1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE              1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM,
`FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT, `SQUARE_PIXEL_IN_MODE,
`VIDEO_QUALITY}

///////////////////////////////////////////////////////////////////////////////
// Register 2
///////////////////////////////////////////////////////////////////////////////

`define Y_PEAKING_FILTER                      3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

///////////////////////////////////////////////////////////////////////////////
// Register 3
///////////////////////////////////////////////////////////////////////////////

`define INTERFACE_SELECT                      2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                         4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS               1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                            1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions
```

```
`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS,
`OUTPUT_FORMAT, `INTERFACE_SELECT}

///////////////////////////////////////////////////////////////////////////////
// Register 4
///////////////////////////////////////////////////////////////////////////////

`define OUTPUT_DATA_RANGE                1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                       1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

///////////////////////////////////////////////////////////////////////////////
// Register 5
///////////////////////////////////////////////////////////////////////////////


`define GENERAL_PURPOSE_OUTPUTS          4'b0000
`define GPO_0_1_ENABLE                   1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                   1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI              1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                     1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI,
`GPO_2_3_ENABLE, `GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

///////////////////////////////////////////////////////////////////////////////
// Register 7
///////////////////////////////////////////////////////////////////////////////

`define FIFO_FLAG_MARGIN                 5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                       1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET             1'b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME              1'b1
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET,
`FIFO_RESET, `FIFO_FLAG_MARGIN}
```

```
////////////////////////////////////////////////////////////////////////////
// Register 8
////////////////////////////////////////////////////////////////////////////

`define INPUT_CONTRAST_ADJUST                    8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////

`define INPUT_SATURATION_ADJUST                  8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register A
////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                  8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register B
////////////////////////////////////////////////////////////////////////////

`define INPUT_HUE_ADJUST                         8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register C
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                     1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE           1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                          6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE,
`DEFAULT_VALUE_AUTOMATIC_ENABLE, `DEFAULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register D
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                         4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                         4'h8
```

```
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

///////////////////////////////////////////////////////////////////////////////
// Register E
///////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE            1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL           2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE              4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE,
`TEMPORAL_DECIMATION_CONTROL, `TEMPORAL_DECIMATION_ENABLE}

///////////////////////////////////////////////////////////////////////////////
// Register F
///////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                    2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY            1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE                  1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR              1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                       1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                      1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                            1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP,
`POWER_DOWN_LLC_GENERATOR, `POWER_DOWN_REFERENCE,
`POWER_DOWN_SOURCE_PRIORITY, `POWER_SAVE_CONTROL}

///////////////////////////////////////////////////////////////////////////////
// Register 33
```

```
///////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                    1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES            1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                          3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                           1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE,
`AVERAGE_BIRIGHTNESS_LINES, `PEAK_WHITE_UPDATE}

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
```

```
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80


module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                   tv_in_i2c_clock, tv_in_i2c_data);

  input reset;
  input clock_27mhz;
  output tv_in_reset_b; // Reset signal to ADV7185
  output tv_in_i2c_clock; // I2C clock output to ADV7185
  output tv_in_i2c_data; // I2C data line to ADV7185
  input source; // 0: composite, 1: s-video

  initial begin
    $display("ADV7185 Initialization values:");
    $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
    $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
    $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
    $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
    $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
    $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
    $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
    $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
    $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
    $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
    $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
    $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
    $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
    $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
    $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
    $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
  end

  //
  // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
  //

  reg [7:0] clk_div_count, reset_count;
  reg clock_slow;
  wire reset_slow;
```

```
initial
  begin
          clk_div_count <= 8'h00;
          // synthesis attribute init of clk_div_count is "00";
          clock_slow <= 1'b0;
          // synthesis attribute init of clock_slow is "0";
  end

always @(posedge clock_27mhz)
  if (clk_div_count == 26)
    begin
            clock_slow <= ~clock_slow;
            clk_div_count <= 0;
    end
  else
    clk_div_count <= clk_div_count+1;

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

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
              .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
              .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
   if (reset_slow)
          begin
             state <= 0;
             load <= 0;
             tv_in_reset_b <= 0;
             old_source <= 0;
          end
   else
          case (state)
            8'h00:
```

```
  begin
     // Assert reset
     load <= 1'b0;
     tv_in_reset_b <= 1'b0;
     if (!ack)
            state <= state+1;
  end
8'h01:
  state <= state+1;
8'h02:
  begin
     // Release reset
     tv_in_reset_b <= 1'b1;
     state <= state+1;
               end
8'h03:
  begin
     // Send ADV7185 address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
            state <= state+1;
  end
8'h04:
  begin
     // Send subaddress of first register
     data <= 8'h00;
     if (ack)
            state <= state+1;
  end
8'h05:
  begin
     // Write to register 0
     data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
     if (ack)
            state <= state+1;
  end
8'h06:
  begin
     // Write to register 1
     data <= `ADV7185_REGISTER_1;
     if (ack)
            state <= state+1;
  end
8'h07:
  begin
     // Write to register 2
     data <= `ADV7185_REGISTER_2;
     if (ack)
            state <= state+1;
  end
8'h08:
  begin
     // Write to register 3
     data <= `ADV7185_REGISTER_3;
     if (ack)
```

```
          state <= state+1;
  end
8'h09:
  begin
    // Write to register 4
    data <= `ADV7185_REGISTER_4;
    if (ack)
          state <= state+1;
  end
8'h0A:
  begin
    // Write to register 5
    data <= `ADV7185_REGISTER_5;
    if (ack)
          state <= state+1;
  end
8'h0B:
  begin
    // Write to register 6
    data <= 8'h00; // Reserved register, write all zeros
    if (ack)
          state <= state+1;
  end
8'h0C:
  begin
    // Write to register 7
    data <= `ADV7185_REGISTER_7;
    if (ack)
          state <= state+1;
  end
8'h0D:
  begin
    // Write to register 8
    data <= `ADV7185_REGISTER_8;
    if (ack)
          state <= state+1;
  end
8'h0E:
  begin
    // Write to register 9
    data <= `ADV7185_REGISTER_9;
    if (ack)
          state <= state+1;
  end
8'h0F: begin
   // Write to register A
   data <= `ADV7185_REGISTER_A;
 if (ack)
   state <= state+1;
end
8'h10:
  begin
    // Write to register B
    data <= `ADV7185_REGISTER_B;
    if (ack)
          state <= state+1;
  end
8'h11:
  begin
     // Write to register C
     data <= `ADV7185_REGISTER_C;
     if (ack)
             state <= state+1;
  end
8'h12:
  begin
     // Write to register D
     data <= `ADV7185_REGISTER_D;
     if (ack)
             state <= state+1;
  end
8'h13:
  begin
     // Write to register E
     data <= `ADV7185_REGISTER_E;
     if (ack)
             state <= state+1;
  end
8'h14:
  begin
     // Write to register F
     data <= `ADV7185_REGISTER_F;
     if (ack)
             state <= state+1;
  end
8'h15:
  begin
     // Wait for I2C transmitter to finish
     load <= 1'b0;
     if (idle)
             state <= state+1;
  end
8'h16:
  begin
     // Write address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
             state <= state+1;
  end
8'h17:
  begin
     data <= 8'h33;
     if (ack)
             state <= state+1;
  end
8'h18:
  begin
     data <= `ADV7185_REGISTER_33;
     if (ack)
             state <= state+1;
  end
```

```
8'h19:
  begin
     load <= 1'b0;
     if (idle)
             state <= state+1;
  end

8'h1A: begin
   data <= 8'h8A;
   load <= 1'b1;
   if (ack)
     state <= state+1;
end
8'h1B:
  begin
     data <= 8'h33;
     if (ack)
             state <= state+1;
  end
8'h1C:
  begin
     load <= 1'b0;
     if (idle)
             state <= state+1;
  end
8'h1D:
  begin
     load <= 1'b1;
     data <= 8'h8B;
     if (ack)
             state <= state+1;
  end
8'h1E:
  begin
     data <= 8'hFF;
     if (ack)
             state <= state+1;
  end
8'h1F:
  begin
     load <= 1'b0;
     if (idle)
             state <= state+1;
  end
8'h20:
  begin
     // Idle
     if (old_source != source) state <= state+1;
     old_source <= source;
  end
8'h21: begin
   // Send ADV7185 address
   data <= 8'h8A;
   load <= 1'b1;
   if (ack) state <= state+1;
end
```

```
                8'h22: begin
                   // Send subaddress of register 0
                   data <= 8'h00;
                   if (ack) state <= state+1;
                end
                8'h23: begin
                   // Write to register 0
                   data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
                   if (ack) state <= state+1;
                end
                8'h24: begin
                   // Wait for I2C transmitter to finish
                   load <= 1'b0;
                   if (idle) state <= 8'h20;
                end
         endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

   input reset;
   input clock4x;
   input [7:0] data;
   input load;
   output ack;
   output idle;
   output scl;
   output sda;

   reg [7:0] ldata;
   reg ack, idle;
   reg scl;
   reg sdai;

   reg [7:0] state;

   assign sda = sdai ? 1'bZ : 1'b0;

   always @(posedge clock4x)
     if (reset)
       begin
                state <= 0;
                ack <= 0;
       end
     else
       case (state)
               8'h00: // idle
                 begin
                    scl <= 1'b1;
                    sdai <= 1'b1;
                    ack <= 1'b0;
                    idle <= 1'b1;
                    if (load)
```

```
            begin
              ldata <= data;
              ack <= 1'b1;
              state <= state+1;
            end
  end
8'h01: // Start
  begin
    ack <= 1'b0;
    idle <= 1'b0;
    sdai <= 1'b0;
    state <= state+1;
  end
8'h02:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h03: // Send bit 7
  begin
    ack <= 1'b0;
    sdai <= ldata[7];
    state <= state+1;
  end
8'h04:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h05:
  begin
    state <= state+1;
  end
8'h06:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h07:
  begin
    sdai <= ldata[6];
    state <= state+1;
  end
8'h08:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h09:
  begin
    state <= state+1;
  end
8'h0A:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h0B:
  begin
    sdai <= ldata[5];
    state <= state+1;
  end
8'h0C:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h0D:
  begin
    state <= state+1;
  end
8'h0E:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h0F:
  begin
    sdai <= ldata[4];
    state <= state+1;
  end
8'h10:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h11:
  begin
    state <= state+1;
  end
8'h12:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h13:
  begin
    sdai <= ldata[3];
    state <= state+1;
  end
8'h14:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h15:
  begin
    state <= state+1;
  end
8'h16:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h17:
  begin
    sdai <= ldata[2];
    state <= state+1;
  end
8'h18:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h19:
  begin
    state <= state+1;
  end
8'h1A:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h1B:
  begin
    sdai <= ldata[1];
    state <= state+1;
  end
8'h1C:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h1D:
  begin
    state <= state+1;
  end
8'h1E:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h1F:
  begin
    sdai <= ldata[0];
    state <= state+1;
  end
8'h20:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h21:
  begin
    state <= state+1;
  end
8'h22:
  begin
```

```
            scl <= 1'b0;
            state <= state+1;
          end
        8'h23: // Acknowledge bit
          begin
            state <= state+1;
          end
        8'h24:
          begin
            scl <= 1'b1;
            state <= state+1;
          end
        8'h25:
          begin
            state <= state+1;
          end
        8'h26:
          begin
            scl <= 1'b0;
            if (load)
                    begin
                      ldata <= data;
                      ack <= 1'b1;
                      state <= 3;
                    end
            else
                    state <= state+1;
          end
        8'h27:
          begin
            sdai <= 1'b0;
            state <= state+1;
          end
        8'h28:
          begin
            scl <= 1'b1;
            state <= state+1;
          end
        8'h29:
          begin
            sdai <= 1'b1;
            state <= 0;
          end
    endcase

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.

///////////////////////////////////////////////////////////////////////////
// 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;  //Passing in 18 bits of data
   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'd30;
   parameter          ROW_START = 10'd30;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 1024 x 768 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 < 1024)) ? col + 1 : col;
               row <= fvh[1] ? ROW_START :
                         (!fvh[2] && fvh[0] && (row < 768)) ? 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];
   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];

   // shift each set of four bytes into a large register for the ZBT

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

   // compute address to store data in

   wire [18:0] myaddr = {y[1][8:0], eo[1], x[1][9:1]};

   // alternate (256x192) image data and address
   wire [35:0] mydata2 = {data[1],data[1],data[1],data[1]};
   wire [18:0] myaddr2 = {1'b0, y[1][8:0], eo[1], x[1][7:0]};

           // modification is made here!
   // update the output address and data only when TWO BYTES ARE READY

   reg [18:0] ntsc_addr;
   reg [35:0] ntsc_data;
   wire       ntsc_we = sw ? we_edge : (we_edge & (x[1][0]==1'b0));

   always @(posedge clk)
     if ( ntsc_we )
       begin
            ntsc_addr <= sw ? myaddr2 : myaddr;        // normal and expanded modes
            ntsc_data <= sw ? {4'b0,mydata2} : {4'b0,mydata};
       end

endmodule // ntsc_to_zbt

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:13:50 12/02/2007
// Design Name:
// Module Name:    hand_sprite
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

// hand_finder module: given the hcount, vcount, and the pixel_in, it outputs the
//                                                         center of mass of the hand pixels as (x, y) value
module hand_finder(clk, reset, hcount,vcount, vsync, pixel_in, x_out, y_out);
        //using parameters here to specify the maximum cr and cb and minimum cr and cb thresholds
        //that we want to use to detect the hand pixels
        //this allows us to create instances of the same class to detect different colors by simply
        //specifying the parameter values by calling defparams

        //max of cr
        parameter CR_MAX = 512;
        //min of cr
        parameter CR_MIN = 0;
        //max of cb
        parameter CB_MAX = 512;
        //min of cb
        parameter CB_MIN = 0;

        input vsync;
        input clk;
        input [10:0] hcount;
        input [9:0] vcount;
        input [17:0] pixel_in;
        output [10:0] x_out;
        output [9:0] y_out;
        input reset;

        //initializes the variables
        reg [29:0] hsum, vsum, last_hsum, last_vsum;
        reg [18:0] counter;
        reg [18:0] last_counter;
        reg old_vsync;
        wire [11:0] cr, cb;
        assign cr = {3'b0, pixel_in[17:9]};
        assign cb = {3'b0, pixel_in[8:0]};

        wire rfd_v, rfd_h;
        wire [18:0] r_v, r_h;
        wire [29:0] x, y;
        reg [10:0] last_x;
        reg [9:0] last_y;
```

```
	//creating an instance of the divider module to compute the vertical center of mass
	//y = last_vsum / last_counter
	divider vdiv(clk, last_vsum, last_counter, y, r_v, rdf_v);
	//creating an instance of the divider module to compute the horizontal center of mass
	//x = last_hsum / last_counter
	divider hdiv(clk, last_hsum, last_counter, x, r_h, rdf_h);

	assign x_out = last_x;
	assign y_out = last_y;

	always @ (posedge clk) begin
		old_vsync <= vsync;
		if (reset)
			begin
				//initializing the variables to 0
				last_counter <= 0;
				last_hsum <= 0;
				last_vsum <= 0;
				last_x <= 0;
				last_y <= 0;
				counter <= 0;
				hsum <= 0;
				vsum <= 0;
			end
		//if it's on the screen and the cr and cb values for the pixel satisfies the cr and cb
thresholds
		//then the hcount and vcount values are added to the running sum of hcounts and vcounts
		//the counter is incremented
		else if ((hcount < 1024) && (vcount < 768) && (cr > CR_MIN) && (cr < CR_MAX)
&& (cb > CB_MIN) && (cb < CB_MAX))
				begin
					hsum <= hsum + hcount;
					vsum <= vsum + vcount;
					counter <= counter + 1;
				end
				//if the current frame has passed, store sum and counter and do divide
		else if (old_vsync && ~vsync)
			begin
				last_counter <= counter;
				last_hsum <= hsum;
				last_vsum <= vsum;
				last_x <= x[10:0];
				last_y <= y[9:0];
				counter <= 0;
				hsum <= 0;
				vsum <= 0;
			end
	end
endmodule

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:14:49 12/09/2007
// Design Name:
// Module Name:    hand_logic
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

//hand_logic : this module takes in two hand positions in (x, y) form and
//                                        computes the angle and the height of the hand positions
module hand_logic(clk, x1, y1, x2, y2, angle, height);
          input clk;
          input [10:0] x1, x2;
          input [9:0] y1, y2;
          output reg [15:0] angle;
          output reg [15:0] height;
          wire [10:0] ndx;
          wire [9:0] ndy;
          wire sx, sy;

          //using parameters so that the users can easily specify the behavior and output
          //using the defparams method on the instances of the module
          parameter SENSITIVITY = 300;
          parameter LEFT = 2;
          parameter RIGHT = 1;
          parameter CENTER = 0;

          //using signed dx and dy so that we know the relative positions of x and y
          reg signed[10:0] dx;
          reg signed[10:0] dy;

          always @ (posedge clk)
          begin
          dx <= $signed({1'b0,x2}) - $signed({1'b0,x1});
          dy <= $signed({1'b0,y2}) - $signed({1'b0,y1});

          //height is simply the average of the y values of the two hands
          height <= (y1 + y2)/2;

          //angle is the angle between the hands
          //more formally, this is the angle between the line connecting the two hands and the horizontal
          //If the angle is between 45 degrees and 135 degrees, then the hands are turning counterclockwise.
          //If the angle is between -45 degrees and -135 degrees, then the hands are turning clockwise.
          //If the angle is between -45 degrees and 45 degrees or between -135 degrees and 135 degrees,
          //then the hands are recognized as not turning.
          //In other words, if the angle formed by the hands is less than 45 degrees from the horizontal,
          //then the hands are recognized as not turning.
          angle <= ((dy > dx) && (dy > -dx)) ? RIGHT :
```

```
                                                        ((-dy > dx) && (-dy > -dx)) ? LEFT :
CENTER;

        end
endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:17:53 12/10/2007
// Design Name:
// Module Name:    vram_display
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
// We take care of that by latching the data at an appropriate time.
//
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.

module vram_display(reset,clk,hcount,vcount,vr_pixel,
                    vram_addr,vram_read_data);

   input reset, clk;
   input [10:0] hcount;
   input [9:0]      vcount;
   output [17:0] vr_pixel;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;

        wire [18:0]         vram_addr = {vcount, hcount[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 <= (hc4==2'd3) ? vr_data_latched : last_vr_data;
```

```
  always @(posedge clk)
    vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;

          //modification is made here!
          //the vr_pixel is now 18 bits long. So instead of getting 8 bits 4 times
          // you get 18 bits of last_vr_data 2 times
  always @(*)                  // each 36-bit word from RAM is decoded to 4 bytes
    case (hc4)
      2'd3: vr_pixel = last_vr_data[17:0];//last_vr_data[7:0];
      2'd2: vr_pixel = last_vr_data[17:0];//last_vr_data[7+8:0+8];
      2'd1: vr_pixel = last_vr_data[35:18];//last_vr_data[7+16:0+16];
      2'd0: vr_pixel = last_vr_data[35:18];// last_vr_data[7+24:0+24];
    endcase

endmodule // vram_display

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:52:21 12/10/2007
// Design Name:
// Module Name:    camera_corner_display
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: logic for creating the display on the top left corner of the screen
//
////////////////////////////////////////////////////////////////////////////////

//camera_corner_display module : this module contains the logic for creating the display
//on the top left corner of the screen
module camera_corner_display(clk, hcount, vcount, x_left, y_left, x_right, y_right, camera_ip, pixel_out);
          input clk;
          input [10:0] hcount, x_left, x_right;
          input [9:0] vcount, y_left, y_right;
          output [23:0] pixel_out;
          output camera_ip;
          reg camera_ip;
          reg [23:0] pixel_reg;
          assign pixel_out = pixel_reg;
          always @ (posedge clk) begin
          //if the hcount is less than 255 and vcount is less than 192
          //then the pixel is in the top left corner of the screen
          //set the camera_ip to 1
                    if ((hcount <= 255) && (vcount < 192))
                              camera_ip <= 1;
                    else camera_ip <= 0;
                    if (camera_ip)
				//If the hcount is between x/4-10 and x/4+10 and the vcount is between y/4-10
and y/4+10,
				//then the pixel is colored with its corresponding color.
				//red gloves correspond to the red blob
				//blue gloves correspond to the blue blob
				pixel_reg <= ((hcount > x_left/4-10 & hcount < x_left/4+10) &
									(vcount > y_left/4-10 & vcount <
y_left/4+10)) ? 24'h0066FF :

		((hcount > x_right/4-10 & hcount < x_right/4+10) &

				(vcount > y_right/4-10 & vcount < y_right/4+10) ? 24'hFF0000 :


									24'h222222);
  else pixel_reg <= 24'b0;
	end
endmodule
```

```
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file divider.v when simulating
// the core, divider. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".
```

```
`timescale 1ns/1ps

module divider(
        clk,
        dividend,
        divisor,
        quotient,
        remainder,
        rfd);


input clk;
input [29 : 0] dividend;
input [18 : 0] divisor;
output [29 : 0] quotient;
output [18 : 0] remainder;
output rfd;

// synopsys translate_off

  DIV_GEN_V1_0 #(
            1,      // algorithm_type
            0,      // bias
            0,      // c_has_aclr
            0,      // c_has_ce
            0,      // c_has_sclr
            0,      // c_sync_enable
            1,      // divclk_sel
            30,     // dividend_width
            19,     // divisor_width
            8,      // exponent_width
            0,      // fractional_b
            19,     // fractional_width
            1,      // latency
            8,      // mantissa_width
            0)      // signed_b
        inst (
            .CLK(clk),
            .DIVIDEND(dividend),
            .DIVISOR(divisor),
            .QUOTIENT(quotient),
            .REMAINDER(remainder),
            .RFD(rfd),
            .CE(),
            .ACLR(),
            .SCLR(),
            .DIVIDEND_MANTISSA(),
            .DIVIDEND_SIGN(),
            .DIVIDEND_EXPONENT(),
            .DIVISOR_MANTISSA(),
            .DIVISOR_SIGN(),
            .DIVISOR_EXPONENT(),
            .QUOTIENT_MANTISSA(),
            .QUOTIENT_SIGN(),
            .QUOTIENT_EXPONENT(),
            .OVERFLOW(),
                        .UNDERFLOW());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of divider is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of divider is "black_box"

endmodule


// ps2_mouse_xy gives a high-level interface to the mouse, which
// keeps track of the "absolute" x,y position (within a parameterized
// range) and also returns button presses.

module ps2_mouse_xy(clk, reset, ps2_clk, ps2_data, mx, my, btn_click);

   input clk, reset;
   inout ps2_clk, ps2_data; // data to/from PS/2 mouse
   output [11:0] mx, my;     // current mouse position, 12 bits
   output [2:0]  btn_click;   // button click: Left-Middle-Right

   // module parameters
   parameter          MAX_X = 1023;
   parameter          MAX_Y = 767;


   // low level mouse driver

   wire [8:0]          dx, dy;
   wire [2:0]          btn_click;
   wire     data_ready;
   wire     error_no_ack;
   wire [1:0]          ovf_xy;
   wire     streaming;

   ps2_mouse m1(clk,reset,ps2_clk,ps2_data,dx,dy,ovf_xy, btn_click,
                        data_ready,streaming);

   // Update "absolute" position of mouse

   reg [11:0]  mx, my;
   wire   sx = dx[8];                        // signs
   wire   sy = dy[8];

           //MODIFICATION!!!
           //decreasing the speed by a factor of 4
           //this is for making sure that the mouse doesn't move too fast
           //that the track created by the mouse becomes discontinuous
   wire [8:0]  ndx = sx ? {3'b0,~dx[7:2]}+1 : {3'b0,dx[7:2]};
           wire [8:0]  ndy = sy ? {3'b0,~dy[7:2]}+1 : {3'b0,dy[7:2]};
   always @(posedge clk) begin
      mx <= reset ? 12'd300 :
              data_ready ? (sx ? (mx>ndx ? mx - ndx : 0)
                                 : (mx < MAX_X - ndx ? mx+ndx : MAX_X)) : mx;
      // note Y is flipped for video cursor use of mouse
      my <= reset ? 12'd300 :
              data_ready ? (sy ? (my < MAX_Y - ndy ? my+ndy : MAX_Y)
                                 : (my>ndy ? my - ndy : 0))  : my;
//            data_ready ? (sy ? (my>ndy ? my - ndy : 0)
//                               : (my < MAX_Y - ndy ? my+ndy : MAX_Y)) : my;
   end

endmodule

///////////////////////////////////////////////////////////////////////////////////////////
//  PS/2 MOUSE
//
//  6.111 Fall 2005
//
//  NOTE:  make sure to change the mouse ports (mouse_clock, mouse_data) to
//                  bi-directional 'inout' ports in the top-level module
//
//  specifically, labkit.v should have the line
//
//     inout  mouse_clock, mouse_data;
//
// This module interfaces to a mouse connected to the labkit's PS2 port.
// The outputs provided give dx and dy movement values (9 bits 2's comp)
// and three button click signals.
//
// NOTE: change the following parameters for a different system clock
//          (current configuration for 50 MHz clock)
//                  CLK_HOLD                               : 100 usec hold to bring PS2_CLK low
//                  RCV_WATCHDOG_TIMER_VALUE : (PS/2 RECEIVER) 2 msec count
//                  RCV_WATCHDOG_TIMER_BITS  :                      bits needed for timer
//                  INIT_TIMER_VALUE              : (INIT process) 0.5 sec count
//                  INIT_TIMER_BITS               :                      bits needed for timer
///////////////////////////////////////////////////////////////////////////////////////////
//
// Nov-8-2005: Registered the outputs (dout_dx, dout_dy, ovf_xy, btn_click) in [ps2_mouse]
//                          Added output "streaming"
// Nov-9-2005: synchronized ps2_clk to local clock for transmitter in [ps2_interface]
//                          Programmed watchdog_timer for [ps2] (receiver module)
//                          Programmed init_timer for [ps2_mouse] (resets initialization)
///////////////////////////////////////////////////////////////////////////////////////////

module ps2_mouse(clock, reset, ps2_clk, ps2_data, dout_dx,
                 dout_dy, ovf_xy, btn_click, ready, streaming);

  input clock, reset;
  inout ps2_clk, ps2_data;           //data to/from PS/2 mouse
  output [8:0] dout_dx, dout_dy;     //9-bit 2's compl, indicates movement of mouse
  output [1:0] ovf_xy;                      //==1 if overflow: dx, dy
  output [2:0] btn_click;            //button click: Left-Middle-Right
  output ready;                             //synchronous 1 cycle ready flag
  output streaming;                                    //==1 if mouse is in stream mode

  ////////////////////////////////////////////////////////////
  // PARAMETERS
  // # of cycles for clock=50 MHz
  parameter CLK_HOLD = 3250;                                    //100 usec hold to bring PS2_CLK
low
  parameter RCV_WATCHDOG_TIMER_VALUE = 65000;   // For PS/2 RECEIVER : # of sys_clks for
2msec.
  parameter RCV_WATCHDOG_TIMER_BITS  = 17;          //                                             : bits
needed for timer
  parameter INIT_TIMER_VALUE = 16250000;                 // For INIT process : sys_clks for 0.5
sec.(SELF-TEST phase takes several miliseconds)
  parameter INIT_TIMER_BITS  = 28;                              //                                    : bits needed
for timer
  ////////////////////////////////////////////////////////////
  wire reset_init_timer;


  ////////////////////////////////////////////////////////////
  //CONTROLLER:
  //-initialization process:
  //        Host:  FF  Reset command
  //        Mouse: FA  Acknowledge
  //        Mouse: AA  Self-test passed
  //        Mouse: 00  Mouse ID
  //        Host:  F4  Enable
  //        Mouse: FA  Acknowledge
  parameter SND_RESET = 0,  RCV_ACK1 = 1,  RCV_STEST = 2, RCV_ID = 3;
  parameter SND_ENABLE =4,  RCV_ACK2 = 5,  STREAM = 6;
  reg [2:0] state;

  wire send, ack;
  wire [7:0] packet;
  wire [7:0] curkey;
  wire key_ready;

  //NOTE: no support for scrolling wheel, extra buttons
  always @(posedge clock) begin
    if (reset || reset_init_timer) state <= SND_RESET;
    else case (state)
       SND_RESET:           state <= ack ? RCV_ACK1 : state;
       RCV_ACK1:state <= (key_ready && curkey==8'hFA) ? RCV_STEST : state;
       RCV_STEST:           state <= (key_ready && curkey==8'hAA) ? RCV_ID : state;
       RCV_ID:              state <= (key_ready) ? SND_ENABLE : state;                    //any device type
       SND_ENABLE:          state <= ack ? RCV_ACK2 : state;
       RCV_ACK2:state <= (key_ready && curkey==8'hFA) ? STREAM :state;
       STREAM:              state <= state;
              default:state <= SND_RESET;
    endcase
  end

  assign send = (state==SND_RESET) || (state==SND_ENABLE);
  assign packet = (state==SND_RESET) ? 8'hFF :
                                  (state==SND_ENABLE) ? 8'hF4 :
                                                                          8'h00;
```

```
assign streaming = (state==STREAM);


// Connect PS/2 interface module
ps2_interface ps2_mouse(.reset(reset), .clock(clock),
                                        .ps2c(ps2_clk), .ps2d(ps2_data),
                                        .send(send), .snd_packet(packet), .ack(ack),
                                        .rcv_packet(curkey), .key_ready(key_ready)  );
defparam ps2_mouse.CLK_HOLD                         = CLK_HOLD;
defparam ps2_mouse.WATCHDOG_TIMER_VALUE = RCV_WATCHDOG_TIMER_VALUE;
defparam ps2_mouse.WATCHDOG_TIMER_BITS  = RCV_WATCHDOG_TIMER_BITS;

////////////////////////////////////////////////////////////
// DECODER
//http://www.computer-engineering.org/ps2mouse/
//                        bit-7                           3                         bit-0
//Byte 1:  Y-ovf  X-ovf  Y-sign  X-sign  1  Btn-M  Btn-R  Btn-L
//Byte 2:  X movement
//Byte 3:  Y movement
reg [1:0] bindex, old_bindex;
reg [7:0] status, dx, dy;           //temporary storage of mouse status
reg [8:0] dout_dx, dout_dy;                 //Clock the outputs
reg [1:0] ovf_xy;
reg [2:0] btn_click;
wire ready;

always @(posedge clock) begin
        if (reset) begin
          bindex <= 0;
          status <= 0;
          dx <= 0;
          dy <= 0;
        end else if (key_ready && state==STREAM) begin
          case (bindex)
                2'b00:    status <= curkey;
                2'b01:    dx <= curkey;
                2'b10:    dy <= curkey;
                default:  status <= curkey;
          endcase

          bindex <= (bindex==2'b10) ? 0 : bindex + 1;
          if (bindex == 2'b10) begin                                    //Now, dy is ready
                dout_dx   <= {status[4], dx};                           //2's compl 9-bit
                dout_dy   <= {status[5], curkey};                 //2's compl 9-bit
                ovf_xy    <= {status[6], status[7]};              //overflow: x, y
                btn_click <= {status[0], status[2], status[1]};  //button click: Left-Middle-Right
    end
        end     //end else-if (key_ready)
end

always @(posedge clock)
  old_bindex <= bindex;

assign ready = (bindex==2'b00) && old_bindex==2'b10;

////////////////////////////////////////////////////////////
```

```
  // INITIALIZATION TIMER
  //          ==> RESET if processs hangs during initialization
  reg [INIT_TIMER_BITS-1:0] init_timer_count;
  assign reset_init_timer = (state != STREAM) && (init_timer_count==INIT_TIMER_VALUE-1);
  always @(posedge clock)
  begin
            init_timer_count <= (reset || reset_init_timer || state==STREAM) ?
                                                                        0 : init_timer_count + 1;
  end

endmodule



////////////////////////////////////////////////////////////////////////////
// PS/2 INTERFACE: transmit or receive data from ps/2

module ps2_interface(reset, clock, ps2c, ps2d, send, snd_packet, ack, rcv_packet, key_ready);
   input clock,reset;
   inout ps2c;                          // ps2 clock       (BI-DIRECTIONAL)
   inout ps2d;                          // ps2 data        (BI-DIRECTIONAL)
   input send;                          //flag: send packet                       _
   output ack;                          // end of transmission                 | for transmitting
   input [7:0] snd_packet;   // data packet to send to PS/2         _|
   output [7:0] rcv_packet;  //packet received from PS/2               _
   output key_ready;              // new data ready (rcv_packet)         _| for receiving

   ///////////////////////////////////////////////////////////////////////////////
   // MAIN CONTROL
   ///////////////////////////////////////////////////////////////////////////////
   parameter CLK_HOLD = 3250;                        //hold PS2_CLK low for 100 usec (50 Mhz)
   parameter WATCHDOG_TIMER_VALUE = 65000;           // Number of sys_clks for 2msec.    _
   parameter WATCHDOG_TIMER_BITS  = 17;        // Number of bits needed for timer   _| for
RECEIVER

   wire serial_dout;                    //output (to ps2d)
   wire rcv_ack;                        //ACK from ps/2 mouse after data transmission

   wire we_clk, we_data;

   assign ps2c = we_clk  ? 0 : 1'bZ;
   assign ps2d = we_data ? serial_dout : 1'bZ;

   assign ack = rcv_ack;


   ///////////////////////////////////////////////////////////////////////////////
   // TRANSMITTER MODULE
   ///////////////////////////////////////////////////////////////////////////////

   /////////////////////////////////////////////////////
   // COUNTER: 100 usec hold
   reg [15:0] counter;
   wire en_cnt;
   wire cnt_ready;
   always @(posedge clock) begin
    counter <= reset ? 0 :
                                   en_cnt ? counter+1 :
                                                    0;
   end
   assign cnt_ready = (counter>=CLK_HOLD);

   ////////////////////////////////////////////////////
   // SEND DATA
   //       hold CLK low for at least 100 usec
   //       DATA low
   //       Release CLK
   //       (on negedge of ps2_clock) - device brings clock LOW
   //                REPEAT:          SEND data
   //       Release DATA
   //       Wait for device to bring DATA low
   //       Wait for device to bring CLK low
   //       Wait for device to release CLK, DATA
   reg [3:0] index;

   // synchronize PS2 clock to local clock and look for falling edge
   reg [2:0] ps2c_sync;
   always @ (posedge clock) ps2c_sync <= {ps2c_sync[1:0],ps2c};
   wire falling_edge = ps2c_sync[2] & ~ps2c_sync[1];

   always @(posedge clock) begin
            if (reset) begin
                    index <= 0;
            end
            else if (falling_edge) begin                          //falling edge of ps2c
              if (send) begin                                             //transmission mode
                    if (index==0)
                            index <= cnt_ready ? 1 : 0;           //index=0: CLK low
                    else
                            index <= index + 1;                                   //index=1: snd_packet[0],
=8: snd_packet[7],
                                                                                //    9: odd parity,
=10: stop bit
                                                                                //          11:
wait for ack
              end else
                    index <= 0;
            end else
              index <= (send) ? index : 0;
   end

   assign en_cnt = (index==0 && ~reset && send);
   assign serial_dout = (index==0 && cnt_ready) ? 0 :                    //bring DATA low before
releasing CLK
                                        (index>=1 && index <=8) ? snd_packet[index-1] :
                                        (index==9) ? ~(^snd_packet) :
            //odd parity
                                                            1;
                    //including last '1' stop bit

   assign we_clk = (send && !cnt_ready && index==0);                            //Enable when
counter is counting up
```

```
  assign we_data = (index==0 && cnt_ready) || (index>=1 && index<=9);//Enable after 100usec CLK
hold
  assign rcv_ack = (index==11 && ps2d==0);                                              //use to
reset RECEIVER module


  ////////////////////////////////////////////////////////////////////////////////
  // RECEIVER MODULE
  ////////////////////////////////////////////////////////////////////////////////
  reg [7:0]   rcv_packet;               // current keycode
  reg                  key_ready;                     // new data
  wire        fifo_rd;                               // read request
  wire [7:0]  fifo_data;                // data from mouse
  wire        fifo_empty;               // flag: no data
  //wire        fifo_overflow;          // keyboard data overflow

  assign      fifo_rd = ~fifo_empty;  // continuous read

  always @(posedge clock)
  begin
        // get key if ready
        rcv_packet <= ~fifo_empty ? fifo_data : rcv_packet;
        key_ready  <= ~fifo_empty;
  end

  ///////////////////////////////////////////////////////
  // connect ps2 FIFO module
  reg [WATCHDOG_TIMER_BITS-1 : 0] watchdog_timer_count;
  wire [3:0] rcv_count;                              //count incoming data bits from ps/2 (0-11)

  wire watchdog_timer_done = watchdog_timer_count==(WATCHDOG_TIMER_VALUE-1);
  always @(posedge clock)
  begin
        if (reset || send || rcv_count==0) watchdog_timer_count <= 0;
        else if (~watchdog_timer_done)
          watchdog_timer_count <= watchdog_timer_count + 1;
  end

  ps2 ps2_receiver(.clock(clock), .reset(!send && (reset || rcv_ack) ),    //RESET on reset or End of
Transmission
                                          .ps2c(ps2c), .ps2d(ps2d),
                                          .fifo_rd(fifo_rd), .fifo_data(fifo_data),              //in1,
out8
                                          .fifo_empty(fifo_empty) , .fifo_overflow(),            //out1,
out1
                                          .watchdog(watchdog_timer_done), .count(rcv_count) );

endmodule



////////////////////////////////////////////////////////////////////////////////
// PS/2 FIFO receiver module (from 6.111 Fall 2004)

module ps2(reset, clock, ps2c, ps2d, fifo_rd, fifo_data, fifo_empty,fifo_overflow, watchdog, count);
 input clock,reset,watchdog,ps2c,ps2d;
 input fifo_rd;
 output [7:0] fifo_data;
 output fifo_empty;
 output fifo_overflow;
 output [3:0] count;

 reg [3:0] count;      // count incoming data bits
 reg [9:0] shift;      // accumulate incoming data bits

 reg [7:0] fifo[7:0];   // 8 element data fifo
 reg fifo_overflow;
 reg [2:0] wptr,rptr;   // fifo write and read pointers

 wire [2:0] wptr_inc = wptr + 1;

 assign fifo_empty = (wptr == rptr);
 assign fifo_data = fifo[rptr];

 // synchronize PS2 clock to local clock and look for falling edge
 reg [2:0] ps2c_sync;
 always @ (posedge clock) ps2c_sync <= {ps2c_sync[1:0],ps2c};
 wire sample = ps2c_sync[2] & ~ps2c_sync[1];

 reg timeout;
 always @ (posedge clock) begin
   if (reset) begin
     count <= 0;
     wptr <= 0;
     rptr <= 0;
     timeout <= 0;
     fifo_overflow <= 0;
   end else if (sample) begin
     // order of arrival: 0,8 bits of data (LSB first),odd parity,1
     if (count==10) begin
       // just received what should be the stop bit
       if (shift[0]==0 && ps2d==1 && (^shift[9:1])==1) begin
         fifo[wptr] <= shift[8:1];
         wptr <= wptr_inc;
          fifo_overflow <= fifo_overflow | (wptr_inc == rptr);
       end
       count <= 0;
       timeout <= 0;
     end else begin
       shift <= {ps2d,shift[9:1]};
       count <= count + 1;
     end
   end else if (watchdog && count!=0) begin
     if (timeout) begin
       // second tick of watchdog while trying to read PS2 data
        count <= 0;
       timeout <= 0;
     end else timeout <= 1;
   end

   // bump read pointer if we're done with current value.
   // Read also resets the overflow indicator
```

```
    if (fifo_rd && !fifo_empty) begin
      rptr <= rptr + 1;
      fifo_overflow <= 0;
    end
  end
endmodule




// The RGBADD module takes in two rgb values (rgb1 and rgb2)
// sums them up by components, and returns the sum or 255 if they went over.
module rgbadd(clk, rgb1, rgb2, rgb);

        input clk;
        input [23:0] rgb1, rgb2;
        output [23:0] rgb;

        wire [7:0] r1 = rgb1[23:16];
        wire [7:0] g1 = rgb1[15:8];
        wire [7:0] b1 = rgb1[7:0];

        wire [7:0] r2 = rgb2[23:16];
        wire [7:0] g2 = rgb2[15:8];
        wire [7:0] b2 = rgb2[7:0];

        reg [8:0] sr, sg, sb;

        // takes a cycle to complete, which should be fine
        always @ (posedge clk)
        begin
                sr <= r1 + r2;
                sg <= g1 + g2;
                sb <= b1 + b2;
        end

   // bound by 255 if they went over
        wire [7:0] r = sr[8] ? 8'hFF : sr[7:0];
        wire [7:0] g = sg[8] ? 8'hFF : sg[7:0];
        wire [7:0] b = sb[8] ? 8'hFF : sb[7:0];


        assign rgb = {r[7:0],g[7:0],b[7:0]};

endmodule



// carsprite takes in the position of the car, the orientation of the car
// and sets the pixel appropriately if the car spans over (hcount, vcount)

module carsprite(
      clk,orientation_x,orientation_y,x,y,
      hcount, hcount_next1, hcount_next2, hcount_next3,
      vcount, vcount_next1, vcount_next2, vcount_next3,
      pixel);
   input clk;
   input signed [15:0] orientation_x, orientation_y;
   input [10:0] x, hcount, hcount_next1, hcount_next2, hcount_next3;
   input [10:0] y, vcount, vcount_next1, vcount_next2, vcount_next3;
   output [23:0] pixel;
   reg [23:0] pixel;

   // rom for car sprite
   wire [11:0] addr;
   wire [23:0] dout;
   rom3584x24 rom(addr,clk,dout);

   // localize coordinates
   wire signed [26:0] localx, localy, localcx, localcy;
        localizepixel localize(clk, x, y,
      hcount, hcount_next1, hcount_next2, hcount_next3,
      vcount, vcount_next1, vcount_next2, vcount_next3,
              orientation_x, orientation_y, localx, localy, localcx, localcy);

        // using the localized coordinates, if they did not go over the boundary
        // of the rectangular car, change the address so it maps to the appropriate
        // row
   wire [26:0] addr2;
   assign addr2 = (localcx >= $signed(0)) ?
                 ((localx < $signed(54)) ?
                   ((localcy >= $signed(0)) ?
                     ((localy < $signed(64)) ? // rowid + colid * 56
                       $unsigned(localx) + $unsigned(localy)*56 : 0) :
                     0) :
                   0) :
                 0;
   assign addr = addr2[11:0];

   // whenever anything changes, reset pixel if addr is not 0.
   // NOTE that we assumes that addr 0 in the rom is transparent
   always @ (x or y or orientation_x or orientation_y or hcount or vcount or dout)
   begin
      pixel = (addr == 0) ? 0 : dout;
   end

endmodule

// localize a pixel relative to the upper-left corner of car (x-halfwidth,y-halfheight)
// and also relative to the center
module localizepixel(clk, x, y,
      hcount, hcount_next1, hcount_next2, hcount_next3,
      vcount, vcount_next1, vcount_next2, vcount_next3,
      dirx, diry, localx, localy, localcx, localcy);
   input clk;
   input [10:0] x, y, hcount, hcount_next1, hcount_next2, hcount_next3, vcount, vcount_next1,
vcount_next2, vcount_next3;
   input signed [15:0] dirx, diry;
   output reg [26:0] localx, localy, localcx, localcy;

   parameter HALFWIDTH = 6912;      // default width: 54 pixels * 256 / 2
   parameter HALFHEIGHT = 8192;     // default height: 64 pixels * 256 / 2
   wire signed [10:0] dist_x, dist_y;
   assign dist_x = hcount_next3 - $signed(x);
   assign dist_y = vcount_next3 - $signed(y);

   reg signed [26:0] xdx, ydy, xdy, ydx, localcxtemp, localcytemp;

   always @ (posedge clk)
   begin
      // FIRST CYCLE
      xdx <= dist_x * dirx;
      ydy <= dist_y * diry;
      xdy <= dist_x * diry;
      ydx <= dist_y * dirx;

      // SECOND CYCLE
      // dot products with orientation of car to find x and y
      localcytemp <= xdx - ydy; // local x from center
      // flip x and y for normal
      localcxtemp <= xdy + ydx; // local y from center

      // THRID CYCLE
      localcx <= localcxtemp / $signed(256);
      localcy <= localcytemp / $signed(256);
      localx <= (localcxtemp + HALFWIDTH) / $signed(256);
      localy <= (localcytemp + HALFHEIGHT) / $signed(256);
   end

endmodule

`timescale 1ns / 1ps

// downloaded from 6.111
// used for debouncing noisy signals
// please refer to the 6.111 website.
module debounce (reset, clock_65mhz, noisy, clean);
   input reset, clock_65mhz, noisy;
   output clean;

   reg [19:0] count;
   reg new, clean;

   always @(posedge clock_65mhz)
      if (reset) begin new <= noisy; clean <= noisy; count <= 0; end
      else if (noisy != new) begin new <= noisy; count <= 0; end
      else if (count == 650000) clean <= new;
      else count <= count+1;

endmodule



// the gamemodule is what actually modifies and keeps track of the game state
// game-modifying inputs include wheel_angle and wheel_height, play_mode,
// and is_out_of_track, is_at_cpt1, is_at_cpt2, which come from the Graphics
// module.
```

```
module gamemodule(clk, reset, vsync,
            wheel_angle, wheel_height,
            play_mode,
            is_out_of_track, is_at_cpt1, is_at_cpt2,
            car_x, car_y, car_orientation_x, car_orientation_y,
                                                speed,
                                                lapnum);
  input clk, reset,
      vsync,      // active low (high when refreshing frame)
      play_mode,   // 1 if in play mode
      is_out_of_track, // 1 if out of track
                        is_at_cpt1, is_at_cpt2;
  input [15:0] wheel_angle, // angle = (clockwise degrees + 180) * 10:
                   // -180 deg => 0; 0 deg => 1800; 180 deg => 3600
          wheel_height; // 0 to 800

  output [15:0] car_x, car_y;
  // <car_orientation_x, car_orientation_y> is the direction vector of the car.
  // they should be normalized with length 9'b100000000 (256)
  output signed [15:0] car_orientation_x, car_orientation_y;
        output [15:0] speed; // 0 to 50
        output reg [2:0] lapnum;

        // rotater keeps track of the orientation of the car and rotates according
        // to wheel angle. Whenever rotate_trigger CHANGES, rotater will compute for
        // new value of the car.
        reg rotate_trigger;
        rotatecar rotater(clk, reset, rotate_trigger, wheel_angle, car_orientation_x, car_orientation_y);

  reg [15:0] car_x, car_y, speed;

  parameter BRAKE = 0; // CUTOFF at 500
  parameter NEUTRAL = 1; // CUTOFF at 300
  parameter GAS = 2;
  wire [1:0] gaspedal; // 0: brake, 1: neutral, 2: gas
  // we could potentially make it more analog by using the wheel_height
  // directly. But due to the noises in hand-finding, that doesn't seem
  // to be a good idea.
  assign gaspedal = (wheel_height > 500) ?
              BRAKE :
              (wheel_height > 300) ?
                NEUTRAL : GAS;

  // need to rescale x after multiplying by speed because orientation are
  // normalized to 256 in length and updating by speed number of pixels
  // will be too quick.
        wire signed [31:0] scaled_vel_x, scaled_vel_y;
        assign scaled_vel_x = car_orientation_x * $signed({1'b0,speed});
        assign scaled_vel_y = car_orientation_y * $signed({1'b0,speed});
        wire signed [20:0] rescaled_vel_x, rescaled_vel_y;
        assign rescaled_vel_x = (scaled_vel_x >>> 11);
        assign rescaled_vel_y = (scaled_vel_y >>> 11);

        wire signed [15:0] nextcar_x, nextcar_y;
        assign nextcar_x = $signed({1'b0,car_x}) + rescaled_vel_x;
        assign nextcar_y = $signed({1'b0,car_y}) - rescaled_vel_y ;

        reg old_vsync;
        reg next_cpt; // 0 for 1st 1 for 2nd.
        // This keeps track of which checkpoint we are interested in next.

  always @ (posedge clk)
  begin
    if (reset)
    begin
      car_x <= 300;
      car_y <= 300;
      speed <= 0;
                        rotate_trigger <= 0;
                        lapnum <= 0;
                        next_cpt <= 0;
    end
    else
                if (play_mode && (~vsync && old_vsync)) // after frame refresh
                        begin
                                // next position 1023 767
                                // force it to not go off-screen
                                car_x <= (nextcar_x < 1023) ?
                                                                ((nextcar_x >= 0) ?
                                                                        nextcar_x :
                                                                        1022) :
                                                                0;
                                car_y <= (nextcar_y < 767) ?
                                                                ((nextcar_y >= 0) ?
                                                                        nextcar_y :
                                                                        766) :
                                                                0;
                                // speed
                                if (is_out_of_track) // when outside the track, lower the maxspeed (15).
Also make braking faster.
                                begin
                                        case (gaspedal)
                                                BRAKE:
                                                        if (speed >= 10)
                                                                speed <= speed - 10;
                                                        else
                                                                speed <= 0;
                                                NEUTRAL:
                                                        if (speed > 5)
                                                                speed <= speed - 5;
                                                        else
                                                                speed <= 0;
                                                GAS: // limit to 15
                                                        if (speed < 15)
                                                                speed <= speed + 1;
                                                        else
                                                                if (speed >= 20)
                                                                        speed <= speed - 5;
                                                                else
                                                                        speed <= 15;
                                                default:
```

```
                                        speed <= 0;
                        endcase
                    end
                    else // otherwise, use normal configs: max speed = 30
                    begin
                        case (gaspedal)
                            BRAKE:
                                if (speed >= 5)
                                    speed <= speed - 5;
                                else
                                    speed <= 0;
                            NEUTRAL:
                                if (speed >= 1)
                                    speed <= speed - 1;
                            GAS:
                                if (speed < 27)
                                    speed <= speed + 3;
                                else
                                    speed <= 30;
                            default:
                                speed <= 0;
                        endcase
                    end

                    // depending on which checkpoint to look for, if that checkpoint
                    // is reached, sets to look for the other checkpoint and increments
                    // lapnum.
                    if (next_cpt)
                    begin
                        if (is_at_cpt2)
                        begin
                            next_cpt <= ~next_cpt;
                            lapnum <= lapnum + 1;
                        end
                    end
                    else
                    begin
                        if (is_at_cpt1)
                        begin
                            next_cpt <= ~next_cpt;
                            lapnum <= lapnum + 1;
                        end
                    end

                    // changing rotate_trigger will cause direction to change to a new
                    // direction according to the wheel angle.
                    rotate_trigger <= ~rotate_trigger;
                end

            old_vsync <= vsync;
  end

endmodule

// rotatecar keeps track of the direction of the car and updates according to
// the wheel angle and when trigger changes.
```

```
module rotatecar(clk, reset, trigger, angle, dir_x, dir_y);
  input clk, reset, trigger;
         input [15:0] angle;
  output signed [15:0] dir_x, dir_y;

  // the rom storing vector components for 128 angles
         reg [7:0] addr;
  wire [31:0] dout;
         angle_rom angles(addr, clk, dout);

         // derive direction from a row (dir_x in first 16 bits and dir_y in last 16)
         assign dir_x = $signed(dout[31:16]);
         assign dir_y = $signed(dout[15:0]);

         reg old_trigger;
         always @ (posedge clk)
         begin
                 if (reset)
                 begin
                         addr <= 0;
                         old_trigger <=0;
                 end
                 else
                 begin
                         // update angle on change
                         if (~old_trigger & trigger)
                         begin

                                 if (angle < 1500) // left
                                         addr <= (addr > 0) ? (addr - 1) : 127;
                                 else
                                         if (angle > 2100) // right
                                                 addr <= (addr < 127) ? (addr + 1) : 0;

                         end

                         old_trigger <= trigger;
                 end
         end

         // orientation

         // matrix multiplication with [ cos -sin ; sin cos ] of 30 or -30
         // cos 5 * 1024 = 1020, sin 5 * 4096 = 357

         /*
         // ALTERNATE IMPLEMENTATION, using MATRIX MULTIPLICATION

  reg [3:0] stage;
         reg oldtrigger;

  always @ (posedge clk)
  begin
                 oldtrigger <= trigger;
    if (reset)
    begin
```

```
 dir_x <= 0;
 dir_y <= 256;
            stage <= 10;
end
        else
            if (trigger != oldtrigger)
            begin
                stage <= 0;
            end
            else
            begin
                case (stage)
                    0:
                    begin
                        xcos <= dir_x * $signed(12'h3FC);
                        xtmp <= dir_x * $signed(12'h165);
                        ycos <= dir_y * $signed(12'h3FC);
                        ytmp <= dir_y * $signed(12'h165);
                        stage <= 1;
                    end
                    1:
                    begin
                        xsin <= xtmp / $signed(4);
                        ysin <= ytmp / $signed(4);
                        stage <= 2;
                    end
                    2:
                    begin
                        leftx <= xcos - ysin;
                        lefty <= xsin + ycos;
                        rightx <= xcos + ysin;
                        righty <= ycos - xsin;
                        stage <= 3;
                    end
                    3:
                    begin
                        newleftdir_x <= leftx / $signed(1024);
                        newleftdir_y <= lefty / $signed(1024);
                        newrightdir_x <= rightx / $signed(1024);
                        newrightdir_y <= righty / $signed(1024);
                        stage <= 4;
                    end
                    4:
                    begin
                        leftxx <= newleftdir_x * newleftdir_x;
                        leftyy <= newleftdir_y * newleftdir_y;
                        rightxx <= newrightdir_x * newrightdir_x;
                        rightyy <= newrightdir_y * newrightdir_y;
                        stage <= 5;
                    end
                    5:
                    begin
                        leftdist2 <= leftxx + leftyy;
                        rightdist2 <= rightxx + rightyy;
                        stage <= 6;
                    end
```

```
					6:
					begin
						if (leftdist2 > $signed(67600)) // 260^2
						begin
							newleftdir_x1 <= newleftdir_x *
$signed(12'h0FC); // approximation: 252 / 256 = 256 / 260
							newleftdir_y1 <= newleftdir_y *
$signed(12'h0FC);
						end
						else
							if (leftdist2 < $signed(63504)) // 252^2
							begin
								newleftdir_x1 <= newleftdir_x *
$signed(12'h104);
								newleftdir_y1 <= newleftdir_y *
$signed(12'h104);
							end
							else
							begin
								newleftdir_x1 <= newleftdir_x *
$signed(12'h100);
								newleftdir_y1 <= newleftdir_y *
$signed(12'h100);
							end

						if (rightdist2 > $signed(67600)) // 260^2
						begin
							newrightdir_x1 <= newrightdir_x *
$signed(12'h0FC); // approximation: 252 / 256 = 256 / 260
							newrightdir_y1 <= newrightdir_y *
$signed(12'h0FC);
						end
						else
							if (rightdist2 < $signed(63504)) // 252^2
							begin
								newrightdir_x1 <= newrightdir_x *
$signed(12'h0FC);
								newrightdir_y1 <= newrightdir_y *
$signed(12'h104);
							end
							else
							begin
								newrightdir_x1 <= newrightdir_x *
$signed(12'h100);
								newrightdir_y1 <= newrightdir_y *
$signed(12'h100);
							end
						stage <= 7;
					end
					7:
					begin
						newleftdir_x <= newleftdir_x1 / $signed(256);
						newleftdir_y <= newleftdir_y1 / $signed(256);
						newrightdir_x <= newrightdir_x1 / $signed(256);
						newrightdir_y <= newrightdir_y1 / $signed(256);
						stage <= 8;
```

```
                                        end
                                        8:
                                        begin
                                                if (angle < 1500) // left
                                                begin
                                                        dir_x <= newleftdir_x;
                                                        dir_y <= newleftdir_y;
                                                end
                                                else
                                                        if (angle > 2100) // right
                                                        begin
                                                                dir_x <= newrightdir_x;
                                                                dir_y <= newrightdir_y;
                                                        end
                                                stage <= 9;
                                        end
                                        default:
                                                stage <= 10; // do nothing
                                endcase
                        end
  end
  */
endmodule



// the graphics module outputs game-related display: the car,
// the track and the background.
// it also does collision detection and outputs the result in
// is_out_of_track, is_at_apt2 and is_at_cpt2.

module graphics (vclock, reset,
       car_x, car_y, car_orientation_x, car_orientation_y,
       track, cpt1blob, cpt2blob,
       hcount, hcount_next1, hcount_next2, hcount_next3,
       vcount, vcount_next1, vcount_next2, vcount_next3,
       hsync,vsync,blank,
       phsync,pvsync,pblank,pixel,
                    is_out_of_track, is_at_cpt1, is_at_cpt2);
  input vclock;   // 65MHz clock
  input reset;    // 1 to initialize module
  input [10:0] hcount, hcount_next1, hcount_next2, hcount_next3;
  // horizontal index of current pixel (0..1023) and the next 3 pixels
  input [9:0] vcount, vcount_next1, vcount_next2, vcount_next3;
  // vertical index of current pixel (0..767) and the next 3 pixels
  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)

  // car position and orientation
  input [15:0] car_x, car_y;
  input signed [15:0] car_orientation_x, car_orientation_y;

  input track; // there is a track at hcount vcount
  output reg is_out_of_track; // output 1 if car pixel has gone out of track pixels
```

```
        input cpt1blob, cpt2blob; // the check point 1/2 has a pixel at that point
        output reg is_at_cpt1, is_at_cpt2;

  output phsync;   // horizontal sync
  output pvsync;   // vertical sync
  output pblank;   // blank
  output [23:0] pixel;   // pixel's rgb

  assign phsync = hsync;
  assign pvsync = vsync;
  assign pblank = blank;

  wire [23:0] pixel1;

        wire is_off_track; // whether current pixel indicates car is off track
        reg old_vsync;
  always @ (posedge vclock)
  begin
                if (reset)
                begin
                        is_out_of_track <= 0;
                        is_at_cpt1 <= 0;
                        is_at_cpt2 <= 0;
                end
                else
                        if (~old_vsync && vsync) // posedge of vsync, reset and begin checking
                        begin
                                is_out_of_track <= 0;
                                is_at_cpt1 <= 0;
                                is_at_cpt2 <= 0;
                        end
                        else
                        begin
                          // sets is_out_of_track if some is_off_track
                          // sets is_at_cpt1 if there is a checkpoint and a car pixel at the same point
                          // sets is_at_cpt2 if there a pixel on checkpoint 2 and a car pixel are at the
same point
                          is_out_of_track <= is_out_of_track | is_off_track;
                                is_at_cpt1 <= is_at_cpt1 | (cpt1blob && (pixel1 != 0));
                                is_at_cpt2 <= is_at_cpt2 | (cpt2blob && (pixel1 != 0));
                        end
                old_vsync = vsync;
        end

        // the car sprite placed at the car position and orientation as specified.
  carsprite cs1 (
    vclock,car_orientation_x,{1'b0,car_orientation_y},car_x,car_y,
    hcount, hcount_next1, hcount_next2, hcount_next3,
    vcount, vcount_next1, vcount_next2, vcount_next3,
    pixel1);

        // get rgb data from tiles
        wire [23:0] track_pixel;
        tracktile tt1 (vclock,hcount,vcount,track,track_pixel);
        wire [23:0] background_pixel;
        maptile mt1 (vclock,hcount,vcount,background_pixel);
```

```
   // if the car is there, print a car pixel, otherwise print the track
   // pixel if there is track on it
            assign pixel = (pixel1 == 0) ?
                                                                                                    ((track) ?
                                                                                                             track_pixel :
background_pixel) :
                                                                                                    pixel1;

   // if there a car pixel there but no track, set to 1
   assign is_off_track = (pixel1 != 0) & (~track);

endmodule

// tracktile basically queries a rom storing a tile for the track
// by changing using hcount and vcount. If there is no tile there,
// however, do not display anything.
module tracktile(clk, hcount, vcount, track, track_pixel);
            input clk;
            input [10:0] hcount;
            input [9:0] vcount;
            input track; // has track at hcount vcount
            output [23:0] track_pixel;

            parameter WIDTH = 128;
            parameter HEIGHT = 128;

            wire[13:0] addr;
            wire[23:0] dout;
            assign addr = hcount * HEIGHT + vcount;
            tracktile_rom trackrom(addr,clk,dout);
            assign track_pixel = (track) ? dout : 0;

endmodule

// maptile basically queries a rom storing a tile for the track
// by changing using hcount and vcount
module maptile(clk, hcount, vcount, map_pixel);
            input clk;
            input [10:0] hcount;
            input [9:0] vcount;
            output [23:0] map_pixel;

            parameter WIDTH = 128;
            parameter HEIGHT = 128;

            wire[13:0] addr;
            assign addr = hcount * HEIGHT + vcount;
            maptile_rom maptilerom(addr,clk,map_pixel);

endmodule

`timescale 1ns / 1ps

// Displays speed gauge showing the specified speed.
module speedgauge(clk, vsync, speed, hcount, vcount, speed_ip, pixel);
```

```
            input clk;
            input vsync;
            input [15:0] speed;
            input [10:0] hcount, vcount;
            output [23:0] pixel;
            output reg speed_ip;

            // specify dimensions
            parameter LEFT = 12;
            parameter WIDTH = 30;
  parameter BOTTOM = 620;
            parameter TOP = 420;

            // top line is determined by speed
            wire [10:0] top_line;
            assign top_line = BOTTOM - (speed * 2);

            // fills up to top line.
            reg [23:0] pixel;
  always @ (posedge clk) begin
                        if ((hcount >= LEFT && hcount < (LEFT+WIDTH)) &&
                                    ((vcount >= TOP) && vcount < BOTTOM))
                              begin
                                    if (vcount >= top_line)
                                          pixel <= {speed[5:0],18'b0};
                                    else
                                          pixel <= 0;
                                    speed_ip <= 1;
                              end
    else
                        begin
                              pixel <= 0;
                              speed_ip <= 0;
                        end
  end
endmodule

module colon_display(clk, reset, hcount, vcount, top_left_x, top_left_y, colon_ip, colon_rgb);

input clk;
input reset;
input[10:0] hcount;
input[9:0] vcount;
input[10:0] top_left_x; //top left corner position
input[9:0] top_left_y;
output colon_ip;
output[23:0] colon_rgb;

reg colon_ip;
reg[23:0] colon_rgb;
wire[23:0] line;
//display colons
char_string_display char6(clk, hcount, vcount, line, 8'b00101110, top_left_x, top_left_y);
defparam char6.NCHAR = 1;
defparam char6.NCHAR_BITS = 1;
```

```
always @ (posedge clk)
begin
//within boundaries
if ((hcount >= top_left_x) && (hcount <= top_left_x + 19) && (vcount >= top_left_y) && (vcount <=
top_left_y + 24))
            colon_ip <= 1;
else colon_ip <= 0;
//displays characters if within boundaries
if (colon_ip)
            colon_rgb <= line;
else colon_rgb <= 24'b0;

end
endmodule


//
// File:   cstringdisp.v
// Date:   24-Oct-05
// Author: I. Chuang, C. Terman
//
// Display an ASCII encoded character string in a video window at some
// specified x,y pixel location.
//
// INPUTS:
//
//   vclock       - video pixel clock
//   hcount       - horizontal (x) location of current pixel
//   vcount       - vertical (y) location of current pixel
//   cstring      - character string to display (8 bit ASCII for each char)
//   cx,cy        - pixel location (upper left corner) to display string at
//
// OUTPUT:
//
//   pixel        - video pixel value to display at current location
//
// PARAMETERS:
//
//   NCHAR        - number of characters in string to display
//   NCHAR_BITS   - number of bits to specify NCHAR
//
// pixel should be OR'ed (or XOR'ed) to your video data for display.
//
// Each character is 8x12, but pixels are doubled horizontally and vertically
// so fonts are magnified 2x.  On an XGA screen (1024x768) you can fit
// 64 x 32 such characters.
//
// Needs font_rom.v and font_rom.ngo
//
// For different fonts, you can change font_rom.  For different string
// display colors, change the assignment to cpixel.


//////////////////////////////////////////////////////////////////////////////
//
// video character string display
```

```
//
////////////////////////////////////////////////////////////////////////////

module char_string_display (vclock,hcount,vcount,pixel,cstring,cx,cy);

  parameter NCHAR = 8;  // number of 8-bit characters in cstring
  parameter NCHAR_BITS = 3; // number of bits in NCHAR

  input vclock;    // 65MHz clock
  input [10:0] hcount;       // horizontal index of current pixel (0..1023)
  input [9:0]       vcount; // vertical index of current pixel (0..767)
  output [23:0] pixel;       // char display's pixel //changed to 24-bits for rgb output
  input [NCHAR*8-1:0] cstring;    // character string to display
  input [10:0] cx;
  input [9:0]       cy;

  // 1 line x 8 character display (8 x 12 pixel-sized characters)

  wire [10:0]       hoff = hcount-1-cx;
  wire [9:0]        voff = vcount-cy;
  wire [NCHAR_BITS-1:0] column = NCHAR-1-hoff[NCHAR_BITS-1+4:4];  // < NCHAR
  wire [2:0]        h = hoff[3:1];           // 0 .. 7
  wire [3:0]        v = voff[4:1];                    // 0 .. 11

  // look up character to display (from character string)
  reg [7:0]  char;
  integer  n;
  always @(*)
   for (n=0 ; n<8 ; n = n+1 )                      // 8 bits per character (ASCII)
     char[n] <= cstring[column*8+n];

  // look up raster row from font rom
  wire reverse = char[7];
  wire [10:0] font_addr = char[6:0]*12 + v;    // 12 bytes per character
  wire [7:0]  font_byte;
  font_rom f(font_addr,vclock,font_byte);

  // generate character pixel if we're in the right h,v area
  wire [23:0] cpixel = (font_byte[7 - h] ^ reverse) ? 24'b111111111111111111111111 : 0; //changed to 24
bits for output
  wire dispflag = ((hcount > cx) & (vcount >= cy) & (hcount <= cx+NCHAR*16)
                              & (vcount < cy + 24));
  wire [23:0] pixel = dispflag ? cpixel : 0;

endmodule


module finish_game(clk, reset, finish_game, hcount, vcount, vsync, finish_ip, finish_rgb);

input clk;
input finish_game; //whether game is finished or not
input[10:0] hcount;
input[9:0] vcount;
input reset;
output[23:0] finish_rgb;
output finish_ip;
```

```
input vsync;

reg finish_ip;
reg[23:0] finish_rgb;

//determines what to display
wire[23:0] line;
char_string_display char3(clk, hcount, vcount, line,
72'b010001110100111101001111010001000010000001001010010011110100001000100001 , 11'd400,
10'd350);
defparam char3.NCHAR = 9;
defparam char3.NCHAR_BITS = 4;

always @(posedge clk)
begin
if (reset)
            finish_ip <= 0; //set to 0 if reset so as to not appear after reset
//if within boundaries
if (finish_game && (hcount >= 300) && (hcount <= 650) && (vcount >= 300) && (vcount <= 450))
            finish_ip <= 1;
else finish_ip <= 0;
if (finish_ip)
finish_rgb <= line; //displays box with lines
else finish_rgb <= 0;
end


endmodule

module incircle(clk, reset, radius_square, mouse_x, mouse_y, hcount, vcount, inbound);

input clk;
input reset;
input [20:0] radius_square; //radius square to be compared with
input[11:0] mouse_x; //positions of mouse input
input[11:0] mouse_y;
input[10:0] hcount;
input[9:0] vcount;
output inbound;

reg[10:0] hdiff;
reg[9:0] vdiff;
wire inbound;
assign inbound = (hdiff*hdiff + vdiff*vdiff <= radius_square)? 1'd1: 1'd0;  //determines whether pixel lies
within circle

always @(posedge clk)
begin
if (hcount >= mouse_x) //find the absolute difference between the 2 x-coordinates
            hdiff <= hcount - mouse_x;
else hdiff <= mouse_x- hcount;
if (vcount >= mouse_y)//find the absolute difference between the 2 y-coordinates
            vdiff <= vcount - mouse_y;
else vdiff <= mouse_y- vcount;
end
```

```
endmodule

module lap_disp(clk, reset, hcount, vcount, vsync, lap_no, lap_rgb, lap_ip);

input clk;
input reset;
input[10:0] hcount;
input[9:0] vcount;
input vsync;
input[2:0] lap_no; //no of checkpoints visited
output[23:0] lap_rgb; //rgb to be output
output lap_ip;

reg lap_ip;
reg[47:0] lap_b;
reg[23:0] lap_rgb;
wire[23:0] line;
//char_string_display instance to display charcter
char_string_display char2(clk, hcount, vcount, line, lap_b , 11'd850, 10'd40);
defparam char2.NCHAR = 6;
defparam char2.NCHAR_BITS = 3;


always @(posedge clk)
begin
//determine character to display based on lap no
case (lap_no)
3'd0: lap_b <= 48'b010000110101000001010100001000000010001100110000;
3'd1: lap_b <= 48'b010000110101000001010100001000000010001100110001;
3'd2: lap_b <= 48'b010000110101000001010100001000000010001100110010;
3'd3: lap_b <= 48'b010000110101000001010100001000000010001100110011;
3'd4: lap_b <= 48'b010000110101000001010100001000000010001100110100;
3'd5: lap_b <= 48'b010000110101000001010100001000000010001100110101;
3'd6: lap_b <= 48'b010000110101000001010100001000000010001100110110;
3'd7: lap_b <= 48'b010000110101000001010100001000000010001100110111;\
//default: not needed
endcase
//determines whether hcount vcount is within boundaries
if ((hcount >= 850) && (hcount <= 980) && (vcount <= 60) && (vcount >= 40))
            lap_ip <= 1'b1;
else lap_ip <= 1'b0;

if (lap_ip) //output character if within boundaries
            lap_rgb <= line;
else lap_rgb <= 24'b0;

end


endmodule


module lights(reset, clk, ready_done, vsync, hcount, vcount, lights_rgb, light_ip, start_game);

input reset;
input clk;
input ready_done; //input from ready_screen module
input vsync;
input[10:0] hcount;
input[9:0] vcount;
output[23:0] lights_rgb;
output light_ip;
output start_game;

reg[7:0] count;
reg oldvsync;
reg[23:0] lights_rgb;
reg light_ip;
reg start_game;

//locations of circles
parameter radius_square = 289;
parameter circlex = 30;
parameter circley1 = 220;
parameter circley2 = 270;
parameter circley3 = 320;
parameter circley4 = 370;

//determines whether pixel lies in any circle
wire circle1;
wire circle2;
wire circle3;
wire circle4;
incircle lights_incircle1(clk, reset, radius_square, circlex, circley1, hcount, vcount, circle1);
incircle lights_incircle2(clk, reset, radius_square, circlex, circley2, hcount, vcount, circle2);
incircle lights_incircle3(clk, reset, radius_square, circlex, circley3, hcount, vcount, circle3);
incircle lights_incircle4(clk, reset, radius_square, circlex, circley4, hcount, vcount, circle4);

always @(posedge clk)
begin
if (reset)
        begin
        lights_rgb <= 24'b0;
        count <= 0;//resets count
        start_game <= 0;//resets start_game
        end
oldvsync <= vsync;
if (ready_done && oldvsync && ~vsync) //when ready_done is 1 and frame refresh
        begin
                if (count == 8'd239) //sets start_game signal when count is over
                        start_game <= 1;
                else
                        begin
                                count <= count + 1; //increment counts if count is not over
                                start_game <= 0;
                        end
        end
//if pixel within boundaries
if ((hcount <= 51) && (hcount >= 12) && (vcount >= 200) && (vcount <= 390))
        light_ip <= 1;
else light_ip <= 0;
if ((count <= 8'd59) && circle1)  //1st second
```

```
        lights_rgb <= 24'b111111110000000000000000;
else if ((count >= 8'd60) && (count <= 8'd119) && circle2) //2nd second
        lights_rgb <= 24'b111111110000000000000000;
else if ((count >= 8'd120) && (count <= 8'd179) && circle3)//3rd second
        lights_rgb <= 24'b111111110000000000000000;
else if ((count >= 8'd180) && (count <= 8'd239) && circle4)//4th second
        lights_rgb <= 24'b000000001111111100000000;
else lights_rgb <=  24'h222222; //otherwise
end



endmodule


module Map(vram_addr, vram_write_data, vram_read_data, vram_we, btn_click, clk, reset, edit, vsync,
mouse_x, mouse_y, hcount, vcount, pixel, ram0_clk, ram1_clk, ram0_we_b, ram1_we_b, ram0_cen_b,
ram1_cen_b, ram0_address, ram1_address, ram0_data, ram1_data, cpt1_x, cpt1_y, cpt2_x, cpt2_y);

input[18:0] vram_addr; //zvt address from video input modules
input[35:0] vram_write_data;//data from video input modules
input vram_we;//write enabled signal from video input modules
output[35:0] vram_read_data;//data read to output to video input modules
input clk;
input reset;
input vsync;
input edit;
input[11:0] mouse_x;
input[11:0] mouse_y;
input[10:0] hcount;
input[9:0] vcount;
input[2:0] btn_click;//button-click of mouse

output pixel;
output[11:0] cpt1_x;//x-position of check-pt 1
output[11:0] cpt1_y;//y-position of check-pt 2
output[11:0] cpt2_x;
output[11:0] cpt2_y;

reg[35:0] vram_read_data;
reg[11:0] mouse_x_use;//register to hold fixed mouse position for each frame
reg[11:0] mouse_y_use;
reg[2:0] old_btn_click;
reg[11:0] cpt1_x;
reg[11:0] cpt1_y;
reg[11:0] cpt2_x;
reg[11:0] cpt2_y;

reg[35:0] write_data_0; //register to hold data to write to zbt0
reg[35:0] write_data_1;//register to hold data to write to zbt1
reg zbt0_we; //write enabled signal for zbt0
reg zbt1_we; //write enabled signal for zbt1
reg oldvsync;
reg pixel;

reg[31:0] pixel_r;
```

```
reg[31:0] pixel_w;
reg[31:0] pixel_c;
reg[31:0] vcount_1;
reg[18:0] zbt0_addr;
reg[18:0] zbt1_addr;
reg[6:0] bit_no_c;



wire[35:0] zbt0_read_data; //data read from zbt0
wire[35:0] zbt1_read_data; //data read from zbt1


output ram0_clk, ram1_clk, ram0_we_b, ram1_we_b, ram0_cen_b, ram1_cen_b; //physical output to zbt
output[18:0] ram0_address, ram1_address; //physical output to zbt
inout[35:0] ram0_data; //physical in/out with zbt
inout[35:0] ram1_data; // physical in/out with zbt

//zbt drivers
zbt_6111 zbt0(clk, 1'b1, zbt0_we, zbt0_addr, write_data_0, zbt0_read_data, ram0_clk, ram0_we_b,
ram0_address, ram0_data, ram0_cen_b);
zbt_6111 zbt1(clk, 1'b1, zbt1_we, zbt1_addr, write_data_1, zbt1_read_data, ram1_clk, ram1_we_b,
ram1_address, ram1_data, ram1_cen_b);

wire inbound; //determine whether pixel is in circle
reg inbound_old_1; //stores old  inbound data to deal with delays
reg inbound_old_2;
incircle incircle1(clk, reset, 21'd2500, mouse_x_use, mouse_y_use, hcount, vcount, inbound);


parameter hpixel = 1344;


always @(posedge clk)
begin
{inbound_old_2, inbound_old_1} <= {inbound_old_1, inbound}; //store old inbound data
vcount_1<= vcount * hpixel; //pipelined method to find out serial no
pixel_r <= vcount_1 + hcount - 1;//read address has to be input 2 cycles ahead
pixel_c <= vcount_1 + hcount - 3;
pixel_w <= vcount_1 + hcount - 4;//extra cycle due to pipelining
oldvsync <= vsync;
old_btn_click <= btn_click; //saves old mouse button-click

if (~old_btn_click[2] && btn_click[2]) //stores mouse x,y position for left button click
        begin
                cpt1_x <= mouse_x;
                cpt1_y <= mouse_y;
        end
if (~old_btn_click[0] && btn_click[0])
        begin
        cpt2_x <= mouse_x;
        cpt2_y <= mouse_y;
        end //stores mouse x,y position for right button click
if (reset)
        begin
                zbt0_we <= 1'b1; //clear memories of zbt 0 and zbt1
```

```
                    zbt1_we <= 1'b1;
                    write_data_0 <= 36'b0;
                    write_data_1 <= 36'b0;
                    mouse_x_use<= mouse_x;

                    mouse_y_use <= mouse_y;
                    zbt0_addr <= pixel_w[23:5]; //address for pixel
                    zbt1_addr <= pixel_w[23:5];
                    pixel <= 1'b0; //output blank screen
            end
else
            if(edit)  //in edit mode
                    begin
                      if (oldvsync && ~vsync)
                                    begin
                                            zbt1_we <= zbt0_we; //flip zbt every frame refresh
                                            zbt0_we <= ~zbt0_we;
                                            mouse_x_use <= mouse_x;//update mouse position
                                            mouse_y_use <= mouse_y;
                                    end
                            if (~zbt0_we) //when zbt0 is being read
                                    begin
                                            zbt1_addr <= pixel_w[23:5]; //address to write to zbt1
                                            zbt0_addr <= pixel_r[23:5]; //address to read from zbt0
                                            bit_no_c <= {2'b0, pixel_c[4:0]}; //corresponding bit no

                                            //updates write_data_1 register every cycle
                                            //writes 1 when track exists or new track is being drawn


                                            case (bit_no_c)
                                            7'd0: begin
                                                                    write_data_1 <=
{zbt0_read_data[35:1], zbt0_read_data[0] | inbound_old_2};
                                                                    pixel<= zbt0_read_data[0];
                                                            end
                                            7'd1: begin
                                                                    write_data_1 <=
{zbt0_read_data[35:2], zbt0_read_data[1] | inbound_old_2, write_data_1[0]};
                                                                    pixel<= zbt0_read_data[1];
                                                            end
                                            7'd2: begin
                                                                    write_data_1 <=
{zbt0_read_data[35:3], zbt0_read_data[2] | inbound_old_2, write_data_1[1:0]};
                                                                    pixel<= zbt0_read_data[2];
                                                            end
                                            7'd3: begin
                                                                    write_data_1 <=
{zbt0_read_data[35:4], zbt0_read_data[3] | inbound_old_2, write_data_1[2:0]};
                                                                    pixel<= zbt0_read_data[3];
                                                            end
                                            7'd4: begin
                                                                    write_data_1 <=
{zbt0_read_data[35:5], zbt0_read_data[4] | inbound_old_2, write_data_1[3:0]};
                                                                    pixel<= zbt0_read_data[4];
                                                            end
```

```
                                        7'd5: begin
                                                        write_data_1 <=
{zbt0_read_data[35:6], zbt0_read_data[5] | inbound_old_2, write_data_1[4:0]};
                                                        pixel<= zbt0_read_data[5];
                                                end
                                        7'd6: begin
                                                        write_data_1 <=
{zbt0_read_data[35:7], zbt0_read_data[6] | inbound_old_2, write_data_1[5:0]};
                                                        pixel<= zbt0_read_data[6];
                                                end
                                        7'd7: begin
                                                        write_data_1 <=
{zbt0_read_data[35:8], zbt0_read_data[7] | inbound_old_2, write_data_1[6:0]};
                                                        pixel<= zbt0_read_data[7];
                                                end
                                        7'd8: begin
                                                        write_data_1 <=
{zbt0_read_data[35:9], zbt0_read_data[8] | inbound_old_2, write_data_1[7:0]};
                                                        pixel<= zbt0_read_data[8];
                                                end
                                        7'd9: begin
                                                        write_data_1 <=
{zbt0_read_data[35:10], zbt0_read_data[9] | inbound_old_2, write_data_1[8:0]};
                                                        pixel<= zbt0_read_data[9];
                                                end
                                        7'd10: begin
                                                        write_data_1 <=
{zbt0_read_data[35:11], zbt0_read_data[10] | inbound_old_2, write_data_1[9:0]};
                                                        pixel<= zbt0_read_data[10];
                                                end
                                        7'd11: begin
                                                        write_data_1 <=
{zbt0_read_data[35:12], zbt0_read_data[11] | inbound_old_2, write_data_1[10:0]};
                                                        pixel<= zbt0_read_data[11];
                                                end
                                        7'd12: begin
                                                        write_data_1 <=
{zbt0_read_data[35:13], zbt0_read_data[12] | inbound_old_2, write_data_1[11:0]};
                                                        pixel<= zbt0_read_data[12];
                                                end
                                        7'd13: begin
                                                        write_data_1 <=
{zbt0_read_data[35:14], zbt0_read_data[13] | inbound_old_2, write_data_1[12:0]};
                                                        pixel<= zbt0_read_data[13];
                                                end
                                        7'd14: begin
                                                        write_data_1 <=
{zbt0_read_data[35:15], zbt0_read_data[14] | inbound_old_2, write_data_1[13:0]};
                                                        pixel<= zbt0_read_data[14];
                                                end
                                        7'd15: begin
                                                        write_data_1 <=
{zbt0_read_data[35:16], zbt0_read_data[15] | inbound_old_2, write_data_1[14:0]};
                                                        pixel<= zbt0_read_data[15];
                                                end
                                        7'd16: begin
                                                        write_data_1 <=
{zbt0_read_data[35:17], zbt0_read_data[16] | inbound_old_2, write_data_1[15:0]};
                                                        pixel<= zbt0_read_data[16];
                                                end
                                        7'd17: begin
                                                        write_data_1 <=
{zbt0_read_data[35:18], zbt0_read_data[17] | inbound_old_2, write_data_1[16:0]};
                                                        pixel<= zbt0_read_data[17];
                                                end
                                        7'd18: begin
                                                        write_data_1 <=
{zbt0_read_data[35:19], zbt0_read_data[18] | inbound_old_2, write_data_1[17:0]};
                                                        pixel<= zbt0_read_data[18];
                                                end
                                        7'd19: begin
                                                        write_data_1 <=
{zbt0_read_data[35:20], zbt0_read_data[19] | inbound_old_2, write_data_1[18:0]};
                                                        pixel<= zbt0_read_data[19];
                                                end
                                        7'd20: begin
                                                        write_data_1 <=
{zbt0_read_data[35:21], zbt0_read_data[20] | inbound_old_2, write_data_1[19:0]};
                                                        pixel<= zbt0_read_data[20];
                                                end
                                        7'd21: begin
                                                        write_data_1 <=
{zbt0_read_data[35:22], zbt0_read_data[21] | inbound_old_2, write_data_1[20:0]};
                                                        pixel<= zbt0_read_data[21];
                                                end
                                        7'd22: begin
                                                        write_data_1 <=
{zbt0_read_data[35:23], zbt0_read_data[22] | inbound_old_2, write_data_1[21:0]};
                                                        pixel<= zbt0_read_data[22];
                                                end
                                        7'd23: begin
                                                        write_data_1 <=
{zbt0_read_data[35:24], zbt0_read_data[23] | inbound_old_2, write_data_1[22:0]};
                                                        pixel<= zbt0_read_data[23];
                                                end
                                        7'd24: begin
                                                        write_data_1 <=
{zbt0_read_data[35:25], zbt0_read_data[24] | inbound_old_2, write_data_1[23:0]};
                                                        pixel<= zbt0_read_data[24];
                                                end
                                        7'd25: begin
                                                        write_data_1 <=
{zbt0_read_data[35:26], zbt0_read_data[25] | inbound_old_2, write_data_1[24:0]};
                                                        pixel<= zbt0_read_data[25];
                                                end
                                        7'd26: begin
                                                        write_data_1 <=
{zbt0_read_data[35:27], zbt0_read_data[26] | inbound_old_2, write_data_1[25:0]};
                                                        pixel<= zbt0_read_data[26];
                                                end
                                        7'd27: begin
```

```
                                                write_data_1 <=
{zbt0_read_data[35:28], zbt0_read_data[27] | inbound_old_2, write_data_1[26:0]};
                                                pixel<= zbt0_read_data[27];
                                        end
                                7'd28: begin
                                                write_data_1 <=
{zbt0_read_data[35:29], zbt0_read_data[28] | inbound_old_2, write_data_1[27:0]};
                                                pixel<= zbt0_read_data[28];
                                        end
                                7'd29: begin
                                                write_data_1 <=
{zbt0_read_data[35:30], zbt0_read_data[29] | inbound_old_2, write_data_1[28:0]};
                                                pixel<= zbt0_read_data[29];
                                        end
                                7'd30: begin
                                                write_data_1 <=
{zbt0_read_data[35:31], zbt0_read_data[30] | inbound_old_2, write_data_1[29:0]};
                                                pixel<= zbt0_read_data[30];
                                        end
                                7'd31: begin
                                                write_data_1 <=
{zbt0_read_data[35:32], zbt0_read_data[31] | inbound_old_2, write_data_1[30:0]};
                                                pixel<= zbt0_read_data[31];
                                        end
                                7'd32: begin
                                                write_data_1 <=
{zbt0_read_data[35:33], zbt0_read_data[32] | inbound_old_2, write_data_1[31:0]};
                                                pixel<= zbt0_read_data[32];
                                        end
                                7'd33: begin
                                                write_data_1 <=
{zbt0_read_data[35:34], zbt0_read_data[33] | inbound_old_2, write_data_1[32:0]};
                                                pixel<= zbt0_read_data[33];
                                        end
                                7'd34: begin
                                                write_data_1 <=
{zbt0_read_data[35], zbt0_read_data[34] | inbound_old_2, write_data_1[33:0]};
                                                pixel<= zbt0_read_data[34];

                                        end
                                7'd35: begin
                                                write_data_1 <=
{zbt0_read_data[35] | inbound_old_2, write_data_1[34:0]};
                                                pixel<= zbt0_read_data[35];
                                        end
                                default: pixel <= 1'b1;
                                endcase
                        end
                else
                        begin
                                zbt1_addr <= pixel_r[23:5]; //zbt1 is now being read
                                zbt0_addr <= pixel_w[23:5];
                                bit_no_c <= {2'b0, pixel_c[4:0]}; //corresponding bit_no
                                case (bit_no_c)
                                7'd0: begin
```

```
                                                                        write_data_0 <=
{zbt1_read_data[35:1], zbt1_read_data[0] | inbound_old_2};
                                                                        pixel<= zbt1_read_data[0];
                                                                end
                                                7'd1: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:2], zbt1_read_data[1] | inbound_old_2, write_data_0[0]};
                                                                        pixel<= zbt1_read_data[1];
                                                                end
                                                7'd2: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:3], zbt1_read_data[2] | inbound_old_2, write_data_0[1:0]};
                                                                        pixel<= zbt1_read_data[2];
                                                                end
                                                7'd3: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:4], zbt1_read_data[3] | inbound_old_2, write_data_0[2:0]};
                                                                        pixel<= zbt1_read_data[3];
                                                                end
                                                7'd4: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:5], zbt1_read_data[4] | inbound_old_2, write_data_0[3:0]};
                                                                        pixel<= zbt1_read_data[4];
                                                                end
                                                7'd5: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:6], zbt1_read_data[5] | inbound_old_2, write_data_0[4:0]};
                                                                        pixel<= zbt1_read_data[5];
                                                                end
                                                7'd6: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:7], zbt1_read_data[6] | inbound_old_2, write_data_0[5:0]};
                                                                        pixel<= zbt1_read_data[6];
                                                                end
                                                7'd7: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:8], zbt1_read_data[7] | inbound_old_2, write_data_0[6:0]};
                                                                        pixel<= zbt1_read_data[7];
                                                                end
                                                7'd8: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:9], zbt1_read_data[8] | inbound_old_2, write_data_0[7:0]};
                                                                        pixel<= zbt1_read_data[8];
                                                                end
                                                7'd9: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:10], zbt1_read_data[9] | inbound_old_2, write_data_0[8:0]};
                                                                        pixel<= zbt1_read_data[9];
                                                                end
                                                7'd10: begin
                                                                        write_data_0 <=
{zbt1_read_data[35:11], zbt1_read_data[10] | inbound_old_2, write_data_0[9:0]};
                                                                        pixel<= zbt1_read_data[10];
                                                                end
                                                7'd11: begin
```

```
                                        write_data_0 <=
{zbt1_read_data[35:12], zbt1_read_data[11] | inbound_old_2, write_data_0[10:0]};
                                        pixel<= zbt1_read_data[11];
                                    end
                            7'd12: begin
                                        write_data_0 <=
{zbt1_read_data[35:13], zbt1_read_data[12] | inbound_old_2, write_data_0[11:0]};
                                        pixel<= zbt1_read_data[12];
                                    end
                            7'd13: begin
                                        write_data_0 <=
{zbt1_read_data[35:14], zbt1_read_data[13] | inbound_old_2, write_data_0[12:0]};
                                        pixel<= zbt1_read_data[13];
                                    end
                            7'd14: begin
                                        write_data_0 <=
{zbt1_read_data[35:15], zbt1_read_data[14] | inbound_old_2, write_data_0[13:0]};
                                        pixel<= zbt1_read_data[14];
                                    end
                            7'd15: begin
                                        write_data_0 <=
{zbt1_read_data[35:16], zbt1_read_data[15] | inbound_old_2, write_data_0[14:0]};
                                        pixel<= zbt1_read_data[15];
                                    end
                            7'd16: begin
                                        write_data_0 <=
{zbt1_read_data[35:17], zbt1_read_data[16] | inbound_old_2, write_data_0[15:0]};
                                        pixel<= zbt1_read_data[16];
                                    end
                            7'd17: begin
                                        write_data_0 <=
{zbt1_read_data[35:18], zbt1_read_data[17] | inbound_old_2, write_data_0[16:0]};
                                        pixel<= zbt1_read_data[17];
                                    end
                            7'd18: begin
                                        write_data_0 <=
{zbt1_read_data[35:19], zbt1_read_data[18] | inbound_old_2, write_data_0[17:0]};
                                        pixel<= zbt1_read_data[18];
                                    end
                            7'd19: begin
                                        write_data_0 <=
{zbt1_read_data[35:20], zbt1_read_data[19] | inbound_old_2, write_data_0[18:0]};
                                        pixel<= zbt1_read_data[19];
                                    end
                            7'd20: begin
                                        write_data_0 <=
{zbt1_read_data[35:21], zbt1_read_data[20] | inbound_old_2, write_data_0[19:0]};
                                        pixel<= zbt1_read_data[20];
                                    end
                            7'd21: begin
                                        write_data_0 <=
{zbt1_read_data[35:22], zbt1_read_data[21] | inbound_old_2, write_data_0[20:0]};
                                        pixel<= zbt1_read_data[21];
                                    end
                            7'd22: begin
```

```
                                                                write_data_0 <=
{zbt1_read_data[35:23], zbt1_read_data[22] | inbound_old_2, write_data_0[21:0]};
                                                                pixel<= zbt1_read_data[22];
                                                        end
                                            7'd23: begin
                                                                write_data_0 <=
{zbt1_read_data[35:24], zbt1_read_data[23] | inbound_old_2, write_data_0[22:0]};
                                                                pixel<= zbt1_read_data[23];
                                                        end
                                            7'd24: begin
                                                                write_data_0 <=
{zbt1_read_data[35:25], zbt1_read_data[24] | inbound_old_2, write_data_0[23:0]};
                                                                pixel<= zbt1_read_data[24];
                                                        end
                                            7'd25: begin
                                                                write_data_0 <=
{zbt1_read_data[35:26], zbt1_read_data[25] | inbound_old_2, write_data_0[24:0]};
                                                                pixel<= zbt1_read_data[25];
                                                        end
                                            7'd26: begin
                                                                write_data_0 <=
{zbt1_read_data[35:27], zbt1_read_data[26] | inbound_old_2, write_data_0[25:0]};
                                                                pixel<= zbt1_read_data[26];
                                                        end
                                            7'd27: begin
                                                                write_data_0 <=
{zbt1_read_data[35:28], zbt1_read_data[27] | inbound_old_2, write_data_0[26:0]};
                                                                pixel<= zbt1_read_data[27];
                                                        end
                                            7'd28: begin
                                                                write_data_0 <=
{zbt1_read_data[35:29], zbt1_read_data[28] | inbound_old_2, write_data_0[27:0]};
                                                                pixel<= zbt1_read_data[28];
                                                        end
                                            7'd29: begin
                                                                write_data_0 <=
{zbt1_read_data[35:30], zbt1_read_data[29] | inbound_old_2, write_data_0[28:0]};
                                                                pixel<= zbt1_read_data[29];
                                                        end
                                            7'd30: begin
                                                                write_data_0 <=
{zbt1_read_data[35:31], zbt1_read_data[30] | inbound_old_2, write_data_0[29:0]};
                                                                pixel<= zbt1_read_data[30];
                                                        end
                                            7'd31: begin
                                                                write_data_0 <=
{zbt1_read_data[35:32], zbt1_read_data[31] | inbound_old_2, write_data_0[30:0]};
                                                                pixel<= zbt1_read_data[31];
                                                        end
                                            7'd32: begin
                                                                write_data_0 <=
{zbt1_read_data[35:33], zbt1_read_data[32] | inbound_old_2, write_data_0[31:0]};
                                                                pixel<= zbt1_read_data[32];
                                                        end
                                            7'd33: begin
```

```
                                                                        write_data_0 <=
{zbt1_read_data_store[35:34], zbt1_read_data[33] | inbound_old_2, write_data_0[32:0]};
                                                                        pixel<= zbt1_read_data[33];
                                                                end
                                                        7'd34: begin
                                                                        write_data_0 <=
{zbt1_read_data[35], zbt1_read_data[34] | inbound_old_2, write_data_0[33:0]};
                                                                        pixel<= zbt1_read_data[34];
                                                                end
                                                        7'd35: begin
                                                                        write_data_0 <=
{zbt1_read_data_store[35] | inbound_old_2, write_data_0[34:0]};
                                                                        pixel<= zbt1_read_data[35];
                                                                end
                                                        default: pixel <= 1'b1;
                                                        endcase
                                                end
                                end
else    //in play mode
        begin
                zbt0_addr <= vram_addr; //zbt0 is used by video camera
                write_data_0 <= vram_write_data;
                vram_read_data <= zbt0_read_data;//output data from zbt0 to camera module
                zbt0_we <= vram_we;
                zbt1_addr <= pixel_r[23:5]; //zbt1 used to display track
                bit_no_c <= {2'b0, pixel_c[4:0]};//
                zbt1_we<= 1'b0;//write disabled
                case (bit_no_c) //read corresponding track pixel
                        7'd0: pixel <= zbt1_read_data[0];
                        7'd1: pixel <= zbt1_read_data[1];
                        7'd2: pixel <= zbt1_read_data[2];
                        7'd3: pixel <= zbt1_read_data[3];
                        7'd4: pixel <= zbt1_read_data[4];
                        7'd5: pixel <= zbt1_read_data[5];
                        7'd6: pixel <= zbt1_read_data[6];
                        7'd7: pixel <= zbt1_read_data[7];
                        7'd8: pixel <= zbt1_read_data[8];
                        7'd9: pixel <= zbt1_read_data[9];
                        7'd10: pixel <= zbt1_read_data[10];
                        7'd11: pixel <= zbt1_read_data[11];
                        7'd12: pixel <= zbt1_read_data[12];
                        7'd13: pixel <= zbt1_read_data[13];
                        7'd14: pixel <= zbt1_read_data[14];
                        7'd15: pixel <= zbt1_read_data[15];
                        7'd16: pixel <= zbt1_read_data[16];
                        7'd17: pixel <= zbt1_read_data[17];
                        7'd18: pixel <= zbt1_read_data[18];
                        7'd19: pixel <= zbt1_read_data[19];
                        7'd20: pixel <= zbt1_read_data[20];
                        7'd21: pixel <= zbt1_read_data[21];
                        7'd22: pixel <= zbt1_read_data[22];
                        7'd23: pixel <= zbt1_read_data[23];
                        7'd24: pixel <= zbt1_read_data[24];
                        7'd25: pixel <= zbt1_read_data[25];
                        7'd26: pixel <= zbt1_read_data[26];
                        7'd27: pixel <= zbt1_read_data[27];
```

```
                    7'd28: pixel <= zbt1_read_data[28];
                    7'd29: pixel <= zbt1_read_data[29];
                    7'd30: pixel <= zbt1_read_data[30];
                    7'd31: pixel <= zbt1_read_data[31];
                    7'd32: pixel <= zbt1_read_data[32];
                    7'd33: pixel <= zbt1_read_data[33];
                    7'd34: pixel <= zbt1_read_data[34];
                    7'd35: pixel <= zbt1_read_data[35];
                    default: pixel <= 1;
            endcase
        end
end

endmodule


module mouse_div(clk_in, clk_out);
input clk_in; //65mhz clk in
output clk_out;//32.5 mhz clk out

reg clk_out;

always @(posedge clk_in)
begin
clk_out <= ~clk_out; //inverts 32.5 Mhz signal at positive edge of 65 mhz clk
end


endmodule



module ready_screen(clk, edit, hcount, vcount, vsync, reset, ready_ip, ready_rgb, ready_done);

input clk;
input edit;
input[10:0] hcount;
input[9:0] vcount;
input reset;
output[23:0] ready_rgb;
output ready_done;
output ready_ip;
input vsync;

reg[7:0] count;
reg ready_ip;
reg[23:0] ready_rgb;
reg ready_done;
reg oldvsync;

wire[23:0] line;
//determines what to output
char_string_display char1(clk, hcount, vcount, line,
128'b0100011101000101010101000010000001010010010001010100000101000100010110010010000001
0010000100000101000011010010110100010101010010 , 11'd400, 10'd350);
defparam char1.NCHAR = 16;
defparam char1.NCHAR_BITS = 4;

always @(posedge clk)
begin
oldvsync <= vsync;
if (reset)
		begin
		count <= 8'd0;
		ready_done <= 0;
		end
if (~edit && oldvsync && ~vsync)  //starts counting when edit is done and frame refresh
		begin
				if (count == 8'd180) //sets ready_done signal to 1 when count is over
						begin
								count <= 8'd180;
								ready_done <= 1;
						end
				else count <= count + 1; //increment count if count is not over
		end
//if within boundaries
if ((count >= 8'd1) && (count <= 8'd179) && (hcount >= 300) && (hcount <= 800) && (vcount >= 300)
&& (vcount <= 450))
		ready_ip <= 1;
else ready_ip <= 0;
if (ready_ip) //outputs line from character display if pixel within boundaries
ready_rgb <= line;
else ready_rgb<= 0;
end


endmodule


module timer_lsb(clk, reset, hcount, vcount, top_left_x, top_left_y, vsync, start_game, timer_ip,
timer_rgb);

input clk;
input reset;
input[10:0] hcount;
input[9:0] vcount;
input[10:0] top_left_x;
input[9:0] top_left_y;
input vsync;
input start_game;
output timer_ip;
output[23:0] timer_rgb;

reg[23:0] timer_rgb;
reg timer_ip;
reg old_vsync;
reg[22:0] count;
reg[3:0] display_no;
reg[7:0] display_no_ascii;

wire[23:0] line;
char_string_display char4(clk, hcount, vcount, line, display_no_ascii, top_left_x, top_left_y);
```

```
defparam char4.NCHAR = 1;
defparam char4.NCHAR_BITS = 1;

always @(posedge clk)
begin
if (reset)
        begin
        count <= 0; //set count to 0
        display_no <= 0;
        timer_ip <= 0;
        timer_rgb <= 0;
        end
old_vsync <= vsync;

if (start_game) //if game has started do this every cycle
        begin
                if (count == 23'd6480000)  //if 0.1 second has passed
                        begin
                                count <= 0;
                                if (display_no == 4'd9) //reset no to 0
                                        display_no <= 4'd0;
                                else display_no <= display_no + 1; //else increments
                        end
                else count <= count +1;
        end
//if within boundaries
if ((hcount >= top_left_x) && (hcount <= top_left_x + 19) && (vcount >= top_left_y) &&  (vcount <=
top_left_y + 24))
        timer_ip <= 1;
else timer_ip <= 0;

if (timer_ip) //determines what to display
        begin
                case (display_no)
                        4'd0: display_no_ascii <= 8'b00110000;
                        4'd1: display_no_ascii <= 8'b00110001;
                        4'd2: display_no_ascii <= 8'b00110010;
                        4'd3: display_no_ascii <= 8'b00110011;
                        4'd4: display_no_ascii <= 8'b00110100;
                        4'd5: display_no_ascii <= 8'b00110101;
                        4'd6: display_no_ascii <= 8'b00110110;
                        4'd7: display_no_ascii <= 8'b00110111;
                        4'd8: display_no_ascii <= 8'b00111000;
                        4'd9: display_no_ascii <= 8'b00111001;
                        default: display_no_ascii <= 8'b11111111; //default case
                endcase
        timer_rgb<= line;
        end
else timer_rgb <= 24'b0;



end
```

```
Endmodule



module timer(clk, reset, hcount, vcount, top_left_x, top_left_y, limit, vsync, start_game, timer_ip,
timer_rgb);

input clk;
input reset;
input[10:0] hcount;
input[9:0] vcount;
input[10:0] top_left_x; //position of left top corner of box
input[9:0] top_left_y;
input[12:0] limit; //limit to count towards
input vsync;
input start_game; //if game has started
output timer_ip;
output[23:0] timer_rgb;

reg[23:0] timer_rgb;
reg timer_ip;
reg old_vsync;
reg[12:0] count;
reg[3:0] display_no;
reg[7:0] display_no_ascii;

determines what to display
wire[23:0] line;
char_string_display char5(clk, hcount, vcount, line, display_no_ascii, top_left_x, top_left_y);
defparam char5.NCHAR = 1;
defparam char5.NCHAR_BITS = 1;

always @(posedge clk)
begin
if (reset)
        begin
        count <= 0; //set count to 0
        display_no <= 0;
        timer_ip <= 0;
        timer_rgb <= 0;
        end
old_vsync <= vsync;

if (start_game && old_vsync && ~vsync) //if game has started and frame refresh
        begin
                if (count == limit)  //if count has reached limit
                        begin
                        count <= 0; //reset count to 0
                        if (display_no == 4'd9)
                                display_no <= 4'd0;
                        else display_no <= display_no + 1;
                        end
                else count <= count +1; //else increments
        end
//if within boundaries
if ((hcount >= top_left_x) && (hcount <= top_left_x + 19) && (vcount >= top_left_y) &&  (vcount <=
top_left_y + 24))
        timer_ip <= 1;
else timer_ip <= 0;

if (timer_ip)
        begin
                //determines what to display
                case (display_no)
                        4'd0: display_no_ascii <= 8'b00110000;
                        4'd1: display_no_ascii <= 8'b00110001;
                        4'd2: display_no_ascii <= 8'b00110010;
                        4'd3: display_no_ascii <= 8'b00110011;
                        4'd4: display_no_ascii <= 8'b00110100;
                        4'd5: display_no_ascii <= 8'b00110101;
                        4'd6: display_no_ascii <= 8'b00110110;
                        4'd7: display_no_ascii <= 8'b00110111;
                        4'd8: display_no_ascii <= 8'b00111000;
                        4'd9: display_no_ascii <= 8'b00111001;
                        default: display_no_ascii <= 8'b11111111;
                endcase
        timer_rgb<= line;
        end
else timer_rgb <= 24'b0;



end




Endmodule


module title(clk, reset, hcount, vcount, vsync, title_ip, title_rgb);

input clk;
input reset;
input[10:0] hcount;
input[9:0] vcount;
input vsync;
output title_ip;
output[23:0] title_rgb;

reg[23:0] title_rgb;
reg title_ip;

wire[23:0] line;
char_string_display char3(clk, hcount, vcount, line,
112'b0100100001000001010000110100101101000101010100100010011101010011001000000101010001
0100100100000101001001010011 00 , 11'd350, 10'd40);
```

```
defparam char3.NCHAR = 14;
defparam char3.NCHAR_BITS = 4;


always @(posedge clk)
begin
//if within boundaries
if ((hcount >= 340) && (hcount <= 590) && (vcount <= 70) && (vcount >= 30))
		title_ip <= 1'b1;
else title_ip <= 1'b0;
//display line or white background
if (title_ip)
		title_rgb <= (line == 0) ? 24'h222222 : line;
else title_rgb <= 24'h222222;

end


endmodule


//
// 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 = ~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
```