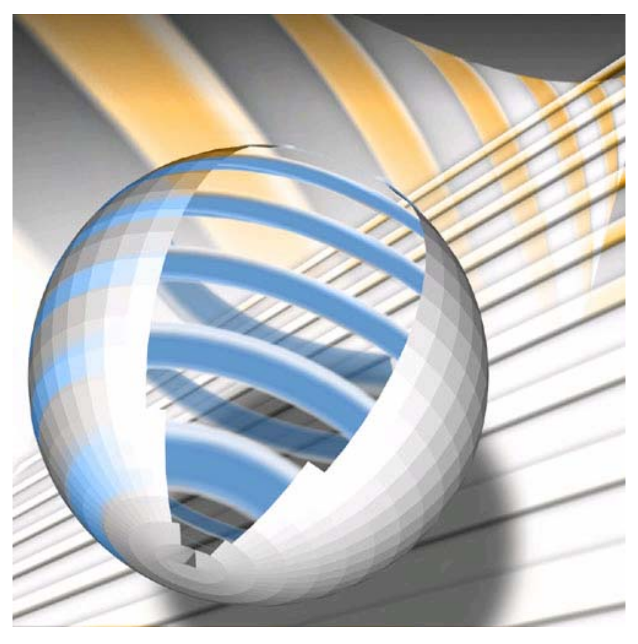
Object Orientation and Position Tracker



Andrew Lee & Tiffany Wang

Fall 2005 – 6.111 Final Project

Abstract

The "Object Orientation and Position Tracker" is a digital tracking system that determines an object's exact position and orientation within a predefined space. Applications of this system involve mostly object location. For example, the tracking system could be used to determine the location of a person within a room and distinguish what the person is looking at or doing based on the orientation of the body. Other applications include boundary detection, such as indications of when an object is about to collide with a wall or barrier, and area detection, such as indications of when the object has entered a particular region in the predefined area.

For this project, a small remote controlled car constrained within a set area is used to simulate the tracking system model. The position of the car is determined using video processing, mapped onto a coordinate system, and displayed on a monitor. In addition to tracking the movement of the car as it traverses the empty space, there are a few predefined maps for the car to navigate through. These maps are all virtually defined and collisions with these virtual boundaries are reflected in the motion and feedback to the controls of the car.

Table of Contents

Overview 6

Module Description/Implementation 13

- Maps 13
  - *Inbox* 13
- Carstate 14
- Object Tracker 16
  - *Minefield* 18
  - *Maze* 18
  - *Racetrack* 19
  - *Open* 19
  - *Vert_Line* 19
  - *Horz_Line* 20
  - *Vga_Sync* 21
  - *Point* 21
- Hexdisplay 21
  - *Divider* 22
  - *Clock* 23
  - *Display_String* 23
- Synchronize 24
- Movement 25
- Position_Calculator 26
  - *XVGA* 26
  - *ZBT_6111* 26
  - *Coordinates* 27
  - *Adv7185init* 27
  - *NTSC_Decode* 27
  - *Gen_Model* 28
  - *NTSC_To_ZBT* 28
  - *DelayN* 28
- Labkit 29

Testing and Debugging 30

Conclusion 34

Appendix 35

maps.v 34
inbox.v 35
carstate.v 36
object_tracker.v 42
open.v 46
minefield.v 48
maze.v 52

racetrack.v……………………………………………………………………….……...…55
horz_line.v……………………………………………………………….…..…….58
vert_line.v……………………………………………………………..………….61
vga_sync.v…………………………………………………………..…………….64
point.v…………………………………………………………….……………….65
hexdisplay.v……………………………………………………………………….66
divider.v…………………………………………..………………………………69
clock.v…………………………………………….………………………………70
display_string.v………………………………………………………..……….72
synchronize.v…………………………………………………………………..…80
coordinates.v……………………………………………………………...……….81
movement.v…………………………………………………….…………….83
ntsc_to_zbt.v……………………………………………………………………84
xvga.v……………………………………………………………..……………86
zbt_6111.v………………………………………..………………………87
YCrCb2RGB.v…………………………………………………………………88
labkit.v (excerpt)……………………………………………………………..…90

Figures

Figure 1:Car on the playing field 6
Figure 2: Car display........ 7
Figure 3: Map change controls. 8
Figure 4: “Start” bounding box........ 8
Figure 5: Open-field terrain. 9
Figure 6: Minefield terrain........ 9
Figure 7: Boom!........ 10
Figure 8: Maze terrain........ 10
Figure 9: Timer controls. 11
Figure 10: Timer display........ 11
Figure 11: MIT 6.111 Racetrack terrain. 11
Figure 12: Object Tracker Block Diagram. 13
Figure 13: Maps module. 14
Figure 14: Carstate module........ 14
Figure 15: Boundary checkpoints. 16
Figure 16: Object_Tracker module. 17
Figure 17: Vert_Line........ 20
Figure 18: Horz_Line........ 20
Figure 19: Hexdisplay module........ 22
Figure 20: Synchronize module. 24
Figure 21: Movement module........ 25
Figure 22: Position_Calculator module 26

Tables

Table 1: Map parameter encoding. 8
Table 2: Car states........ 15

Overview

This “Object Orientation and Position Tracker” final project involves the design and implementation of a digital tracking system used to determine the position and orientation of an object within an enclosed area. The object being tracked is a small remote controlled car. The goal is to track the car and accurately display its position and orientation on a monitor in real time, have it interact with virtually defined boundaries, and provide feedback to the remote control in the case of collisions.

The position of the car is determined by using a video camera to capture the location of the car in respect to the background playing field as shown in Figure 1. To help the video camera pick out the car from the background, the four corners of the car are indicated by colored dots. The image captured by the video camera then filters the image for the colored dots. Using the filtered image, the position of the car is calculated in respect to the playing field.

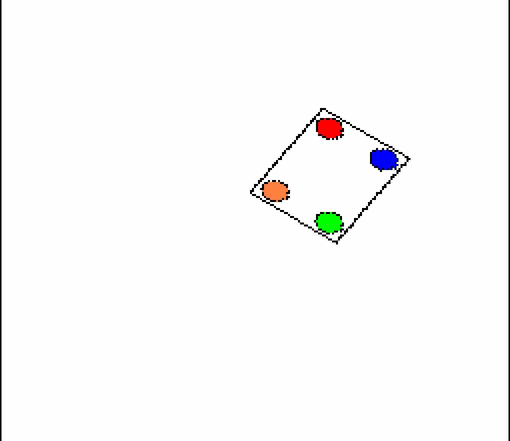


Figure 1:Car on the playing field

The corners of the car are indicated by different colors. Red is front left, blue is front right, orange is back left, and green is back right.

....

The car is displayed on the monitor as four points, as seen in Figure 2, corresponding to the front left, front right, back left and back right corners. The front points of the car are in yellow to represent headlights while the back points of the car are in red to represent brake lights.

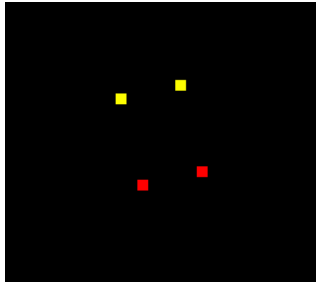


Figure 2: Car display.

The four corners of the car are displayed on the monitor as small dots; yellow representing the front headlights and red representing the back taillights.

The tracking system provides feedback to the car's remote through direction halting signals. By checking the car's coordinates against the boundaries of each map terrain element, such as walls, collisions are reported and proper halting signals for the car's motion are asserted. For example, if the car is facing a wall, feedback to the remote control will not allow the car to move forward. Similar signals are also asserted for right, left, and backward directions.

The remote control of the car is connected to the directional buttons of the Labkit. When the respective button on the Labkit is depressed, the car will move in that direction unless a halting signal is received. Thus, if the remote control feedback reports a forward halting signal, the car will not move forward even when the forward button is depressed.

There are four predefined maps that the car is placed in: an open-field, a minefield, a maze, and a racetrack. The user selects between each map using the lower two switches and the enter button on the Labkit, as seen in Figure 3. The switches are used to select the parameter encoding of each map and the enter button is used to assert the change. Map changes can only occur if the car is within the green "Start" bounding box, shown in Figure 4. This constraint prevents the car from becoming trapped within any boundaries of the new map.

Figure 3: Map change controls.

Switch[1] and switch[0] are used to specify the parameter encoding of each terrain. The Enter button asserts the switch between maps. Switch[7] is used to put the system in reset mode.

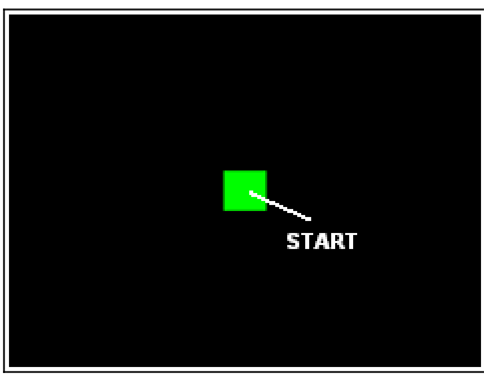


Figure 4: "Start" bounding box.

The car must be within this box in order to switch between maps to avoid getting trapped within walls.

To make repositioning the car a little easier, the user can set the system into "reset" mode using another of the switches on the Labkit. This mode clears all walls and boundaries so the user does not need to backtrack through the terrain and can navigate directly to the "Start" box.

Each map terrain has a different parameter encoding as specified in Table 1.

Table 1: Map parameter encoding.

Each map has a 2-bit binary encoding which is selected by the user through the Labkit switches.

Map Name	Parameter Number
open-field	00
minefield	01
maze	10
racetrack	11

The first map is an open-field, as seen in Figure 5. This terrain is bounded only by the borders of the predefined area. This map allows the car to roam freely within the enclosed space.

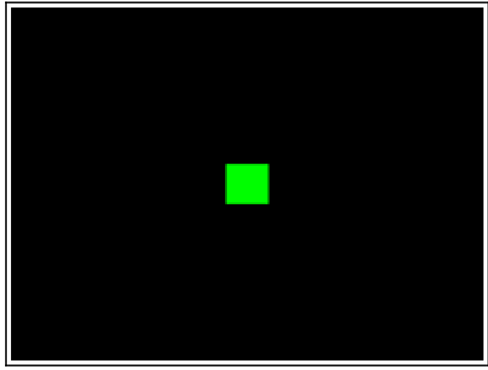


Figure 5: Open-field terrain.

This map is bounded only by the borders of the predefined area.

The second map, as seen in Figure 6, simulates a minefield. There are explosive mines scattered throughout the area in which the car must navigate through safely. If the car comes in contact with any of the mines, the mine is detonated as demonstrated by the car being halted and the "BOOM!" alert message being displayed on the Labkit's LED hex display, shown in Figure 7.

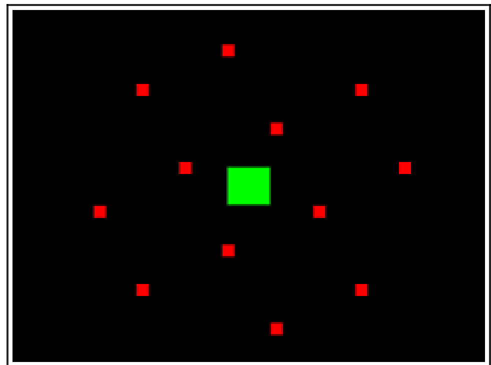


Figure 6: Minefield terrain.

This map contains mines that the car must avoid running into while navigating the area.

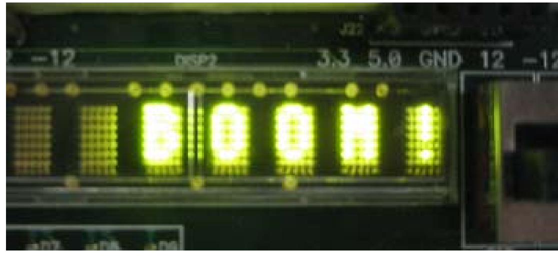


Figure 7: Boom!

This message is displayed on the hex display when the car runs into a mine.

The third map is a maze, as seen in Figure 8. Bounded by the walls in the terrain, the car must find its way from the green "Start" box to the cyan "Finish" box. By turning on the timer switch, as seen in Figure 9, users have the option of timing their performance. The timer, displayed on the Labkit's LED hex display, starts as soon as the car is outside the "Start" box and halts once it is completely inside the "Finish" box. The timer can be reset at anytime by pushing Button3 on the Labkit.

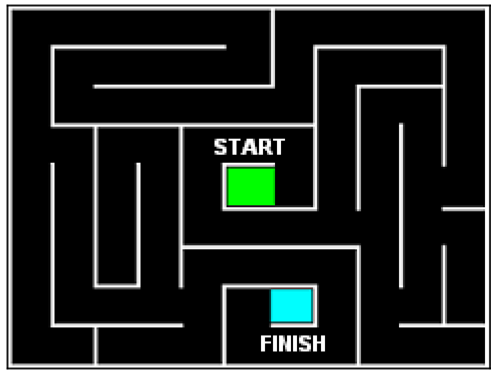


Figure 8: Maze terrain.

This map contains walls placed in a maze arrangement. The car must navigate its way through the area from the start to finish box.

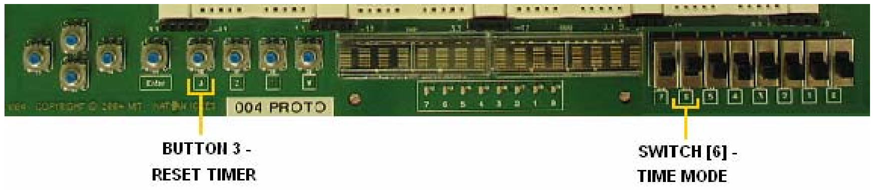


Figure 9: Timer controls.

The user has the option of turning on the timer by using switch[6] and resetting the timer with button[3].

Figure 10: Timer display.

The timer is displayed on the hex display in the form minutes : seconds.

The last map, as seen in Figure 11, is the MIT 6.111 racetrack. Bounded by the walls of the track, the car is allowed to run laps within the terrain. As in the maze terrain, users have the option of turning on the timer to clock their lap times. The timer runs as long as the car is outside the "Start" box and may be reset at anytime.

Figure 11: MIT 6.111 Racetrack terrain.

This map contains walls placed in a racetrack arrangement. The car is allowed to run laps within the terrain.

After laying out the functionality and specifications of the project, the overall system was divided into smaller modules, each with a separate role. This approach made it easier to plan the design and implementation of the system as well as divide up the tasks. The first half involved setting up the remote controlled car to interface with the Labkit and to set up the video processing for tracking the car's position and orientation. The second half involved creating and implementing the map terrains and setting up the boundary checking system to provide feedback to the car remote. Once each part was properly implemented and tested, they were integrated under a single top level module to complete the system.

Module Description/Implementation

The object tracking system is broken up into several smaller modules, each with its own functionality that contributes to the overall system. See Figure 12 for the overall schematics of the system.

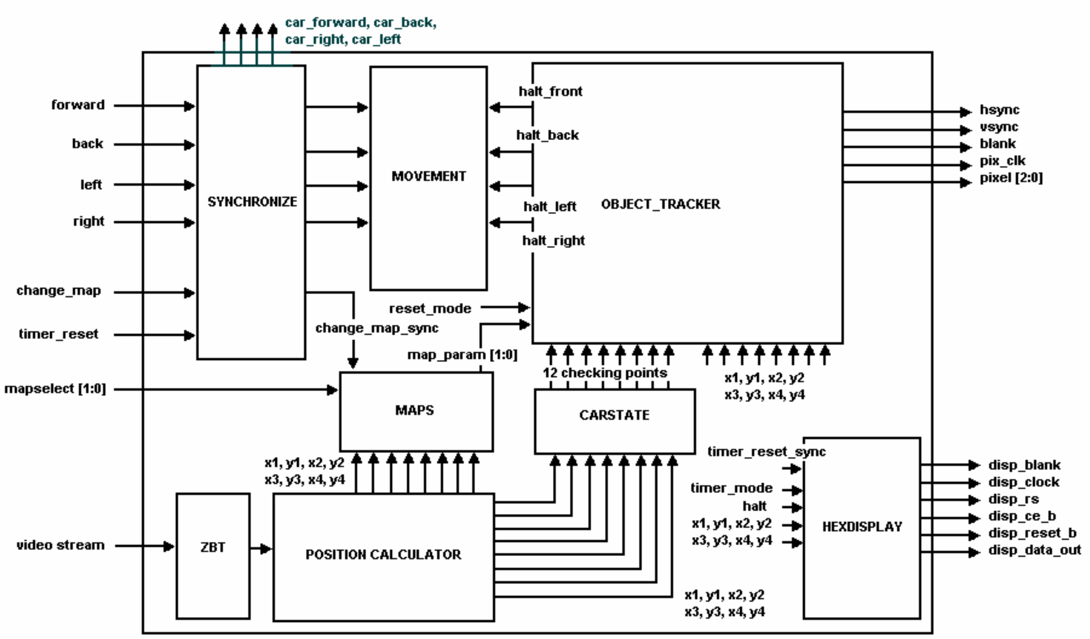


Figure 12: Object Tracker Block Diagram.

The object tracking system is broken down into several submodules.

- **Maps**

The **maps** module handles the map switch selections asserted by the user through the Labkit switch and button values. The value of `<mapselect>` specifies the parameter encoding of the selected terrain, and outputs this parameter value to the object tracker module when `<map_change>` is asserted.

- ***Inbox***

  The car coordinates from the position calculator module are used as inputs for the **inbox** submodule, which checks if all four points are within the specified bounds of the "Start" box. A map change occurs only if this submodule returns a high value.

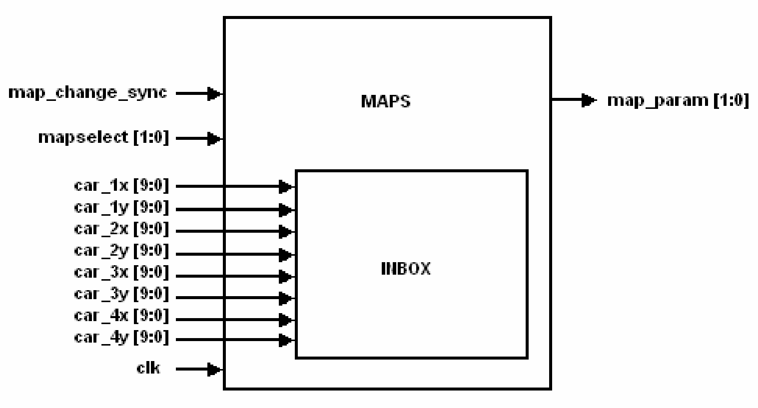


Figure 13: Maps module.

This module handles transitions between maps by taking in user inputs from the Labkit and checking that the car is within the "Start" box.

- **Carstate**

The **carstate** module is the FSM (finite state machine) of the tracking system. The states of the system are the car's orientation, or direction, since collision checks and corresponding directional halting feedback to the controller will depend on which direction the car is oriented. As seen in Figure 14, the module takes in the coordinates of the car corners from the position calculator module and outputs another set of twelve coordinates points used in the object_tracker module for collision checking.

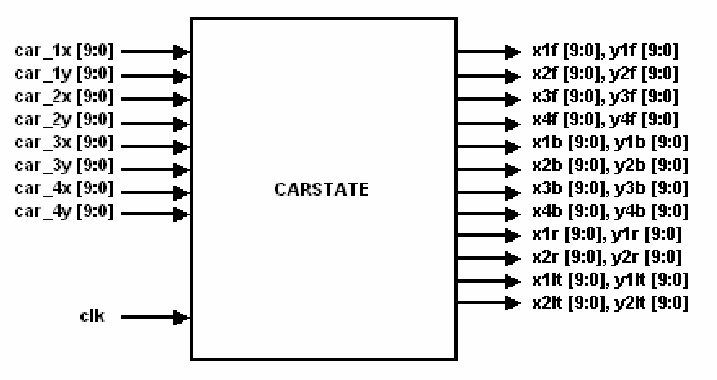


Figure 14: Carstate module.

This module determines the state, or direction, of the car based on the car's coordinate points and outputs a set of points for collision checking.

Since the car coordinates do not change in relative position to each other, the state of the car is determined by comparing the vertical position (y-coordinates) of each of the four car coordinates. There are a total of eight states, as seen in Table 2.

Table 2: Car states.

States correspond to the orientation of the car and determined by the position of the car coordinates in relation to each other.

State	Direction	Logic	Example
0	north	car_1y == car_2y && car_1y < car_4y	
1	northeast	car_1y < car_2y, car_3y, car_4y	
2	east	car_1y == car_4y && car_1y < car_2y	
3	southeast	car_4y < car_1y, car_2y, car_3y	
4	south	car_1y == car_2y && car_1y > car_4y	
5	southwest	car_3y < car_1y, car_2y, car_4y	
6	west	car_1y == car_4y && car_1y > car_2y	
7	northwest	car_2y < car_1y, car_3y, car_4y	

The module outputs twelve boundary points around the car, based on the state of the car, which are used for collision checking: four points in front, four points in the back, two points on the

right, and two points on the left. These points are calculated by adding an offset (defaulted to 5 pixels) to the actual car coordinates in certain directions, depending on the state. This creates a sort of buffer zone around the car to check for collisions. Calculations for checkpoints for state 7 (northwest) is shown in Figure 15 (see Verilog code in the Appendix for the remaining states' points calculations).

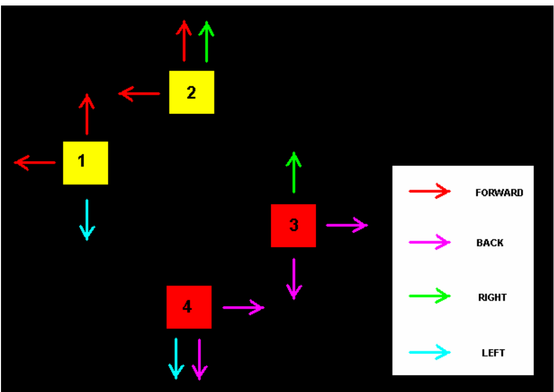


Figure 15: Boundary checkpoints.

Boundary checkpoint, calculated by an offset from the actual car points in state-determined directions, are used for collision checking. This figure shows the points used in boundary checking for state7 (northwest direction).

- **Object Tracker**

The **object tracker** module acts as a sort of central processing module and serves numerous functions including producing display signals for the monitor, holding terrain element information for each map, performing boundary checking between the map elements and car coordinates, and producing proper feedback to the car controller through halting signals to the movement module.

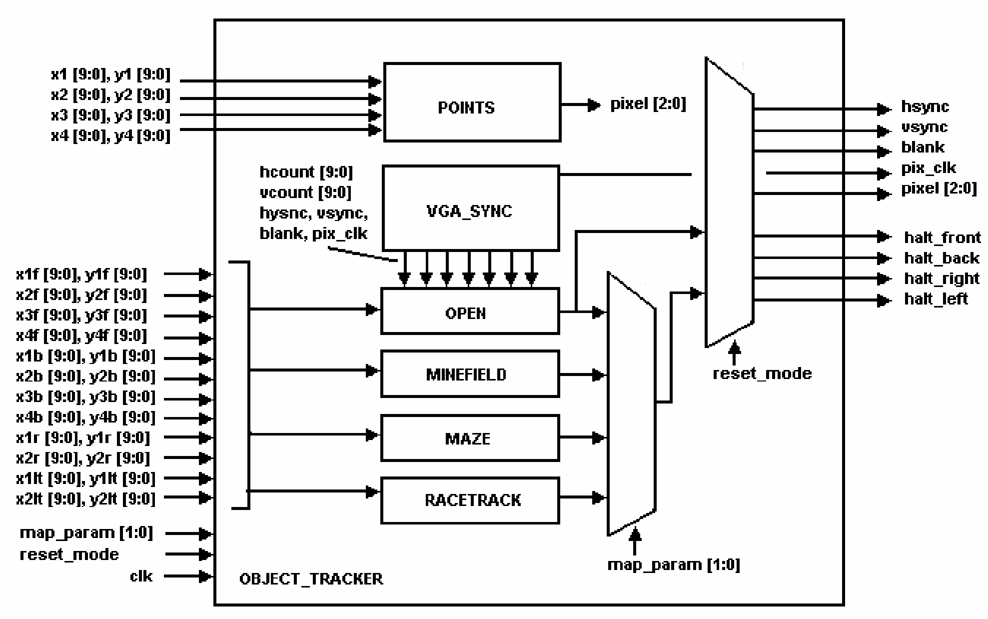


Figure 16: Object_Tracker module.

This module produces display signals for the monitor, holds terrain element information for each map, performs boundary checking, and produces feedback signals to the controller.

Each of the map terrains is defined as a separate submodule within the object_tracker module and outputs its own pixel and halting values. Which module's outputs to use in the system is determined within object_tracker based on the map selected by the user. With the exception of the open field, the map terrains images are loaded from an image ROM. This allows for the flexibility of having curved lines and slightly more complex graphics than straight lines, as seen in the racetrack map. Collision checking is achieved through pixel checking. The set of check points are used to extract the corresponding pixel from the image ROM. If the pixel is not black, signifying a wall, mine, or other non-track coordinate, the halt signal is asserted.

The map images were first created using the Paint application and saved as a bitmap file. Using the Athena graphics file conversion tool, each bitmap file was converted to a .pgm file. The .pgm file was then run through the provided pgm2coe file converter to produce a .coe file, which is the format used for preloading ROM values. However, the given converter was used for loading 8-bit graymap pixels instead of the 3-bit RGB encoding used for the display. A simple color mapping and a find and replace scan was used to replace each 8-bit value in the .coe file with a corresponding 3-bit RGB value.

Due to the size constraints of the Labkit block and the total size of ROMs that can be instantiated, it is not possible to output 3-bit RGB pixel values for each terrain directly to the display. Therefore ROMs of 1-bit and 2-bit widths are used and their values mapped to a color lookup table to determine what color pixel to output.

- ***Minefield***

The mine1 (one port) and mine2 (dual ports) image ROMs are instantiated within the **minefield** module. Since the created BRAMs have a maximum of two read ports, a total of seven ROMs are required for displaying and collision checking. The module takes in `<hcount>`, `<vcount>`, and the set of checkpoints and converts these coordinates into ROM memory addresses. Since the ROM image has 320x240 resolution while the screen is at 640x480 resolution, each memory location contains the value for four pixels. The memory address is thus calculated by dividing the x- and y-coordinates by two, multiplying the new y-value by 320 and adding this to the new x-value: x/2 + (y/2 * 320). The ROMs output a 1-bit value, indicating a black or non-black pixel. The high values are converted to red pixels (3'b100) and assert a halt signal while low values are converted to black pixels (3'b000) and do not assert a halt signal. The borders of the terrain were instants of horizontal and vertical lines described below.

- ***Maze***

The maze1 (one port) and maze2 (dual ports) image ROMs are instantiated within the **maze** module. The memory address calculation, pixel extrapolation, and collision checking processes are identical to the minefield module. High values from the ROM are converted to white pixels (3'b111) collision and assert a halt signal while low values are converted to black pixels (3'b000) and do not assert a halt signal.

- ***Racetrack***

The racetrack1 (one port) and racetrack2 (dual ports) image ROMs are instantiated within the **racetrack** module. Again, the memory address calculation, pixel extrapolation, and collision checking processes are done in the same manner as the previous two terrain modules. However, instead of a 1-bit wide ROM, this module utilizes 2-bit wide ROMs since it uses more colors. The ROM outputs are mapped to a lookup table to produce white, green, blue and black pixels. A halting signal is returned high if any of the pixels return a non-black point.

- ***Open***

The **open** module for the display of the open field terrain is not loaded from the ROM (due to space constraints), but instead contains multiple instances of terrain elements such as vertical lines and horizontal lines. The pixel output and collision checking is handled by individual terrain elements, so the main function of the open terrain module is to instantiate these line elements and to gather and process each of their pixel and halting outputs into a single value to return to the object_tracker module.

Each of these terrain element modules takes in `<hcount>` and `<vcount>` and checks them against the bounds of the element in order to determine whether to output a color pixel if the point is within the element or black pixel if it is not. Collision checking is done in an identical manner, checking the coordinate points to the bounds of the element in order to determine whether to assert a halt if the point is within the element boundary.

- ***Vert_Line***

A **vertical line** instance may be drawn in three segments, as seen in Figure 17, and has adjustable width, heights, and color as ascribed by its parameter values. The `<x>` input value determines where the left edge of the line is located and the `<y1, y2,`

y3> inputs values indicate the upper edge of each segment. The length of each segment has a separate adjustable parameter.

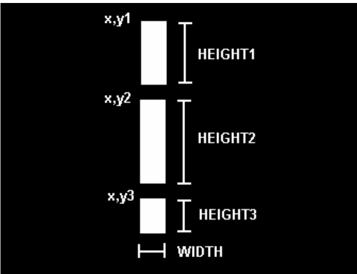


Figure 17: Vert_Line.

Vertical lines have adjustable width, heights, and color. Each segment is placed according to the specified x and y values and have adjustable heights.

- ***Horz_Line***

A **horizontal line** instance may be drawn in three segments, as seen in Figure 18, and has adjustable widths, height, and color as ascribed by its parameter values. The <y> input value determines where the top edge of the line is located and the <x1, x2, x3> inputs values indicate the left edge of each segment. The width of each segment has a separate adjustable parameter.

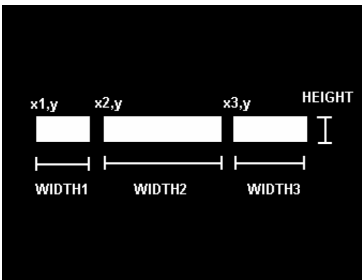


Figure 18: Horz_Line.

Horizontal lines have adjustable widths, height, and color. Each segment is placed according to the specified x and y values and have adjustable widths.

The object_tracker module also contains modules used to produce VGA signals such as syncs and blanks needed to properly display images on the monitor.

- ***Vga_Sync***

  The **vga_sync** submodule is used to produce VGA sync and blanking signals to support a 640x480 resolution screen and runs on a 50-MHz clock. The `<hcount>` and `<vcount>` outputs correspond to the pixel number and line number on the screen and are incremented at every positive clock edge. When the counts reach their max values (639 and 479 for this resolution) the `<hsync>` and `<vsync>` signals trigger the start of a new pixel line and a screen refresh, respectively. The `<blank>` signal is triggered at the end of each pixel line and produces the empty (black) pixels observed at the edges of the image when no image is displayed. The `<hcount>` and `<vcount>` outputs are utilized by the other map terrain modules to display the proper pixel. The sync and blank signals are outputted to the Labkit to produce the monitor image.

The coordinate images representing the car are also instantiated within object_tracker.

- ***Point***

  The **point** module is used to display the points on the car given their coordinate points. A point instance has adjustable width, height, and color as ascribed by its parameter values. The `<x>` and `<y>` input values determine where the upper left corner of the point is located.

- **Hexdisplay**

The **hexdisplay** module employs several submodules to output display signals for the Labkit's 16-Digit HEX display. The system displays a different message on the HEX display depending on which map terrain is currently being used. As seen in Figure 19, the module takes in the input `<map_param>` from the maps module as the control value to select which data stream to output. The openfield terrain simply prints "SELECT A MAP." The minefield terrain displays "BOOM!" when the car comes in contact with a mine. This is implemented by displaying the characters

based on the inputted <halt> signal, which is high if any of the halting feedback produced by the object_tracker module is asserted. Otherwise, the display is normally blank if the signal is low. Lastly, both the maze and racetrack terrains display a timer, produced by the clock submodule.

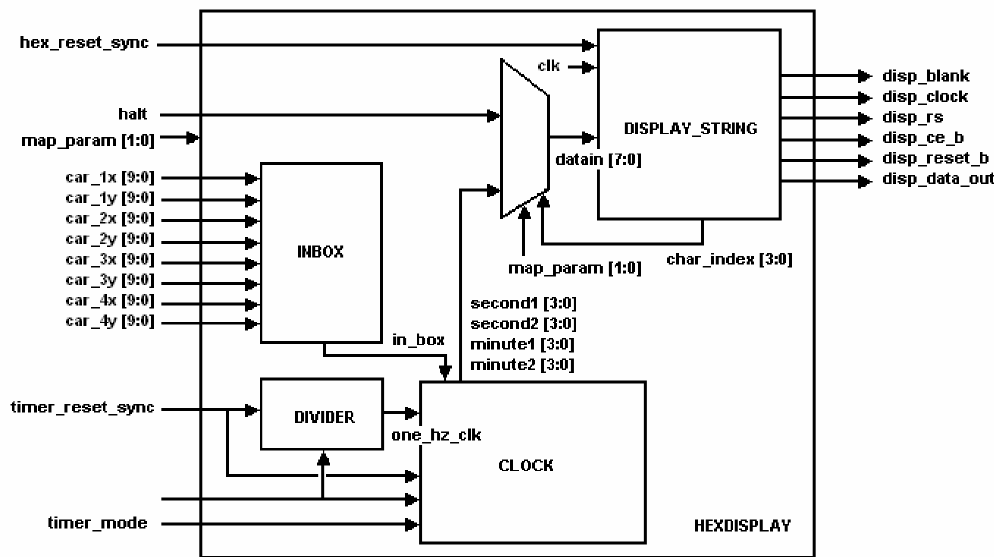


Figure 19: Hexdisplay module.

This module consists of several submodules used to output display signals and data values for the Labkit HEX display.

- ***Divider***

In order to properly time the user's progress on the maze and racetrack terrain, a system of measuring time in minute sand seconds is required. The frequency of the system clock used in the tracking system is 27-MHz, but a 1-Hz signal is needed to generate a signal that is high for one clock cycle at every second. The **divider** submodule is essentially used to slow down the system clock by dividing its frequency by a specific factor (2.7x10^7 in this case). The divider increments a counter at every rising edge of the system clock edge. Once the counter reaches 2.7x10^7, the output <one_hz_clk> is set to high for that clock cycle, thus producing a 1 Hz signal. The counter is then reset to zero as the cycle starts anew. If

the <timer_reset_sync> input is high (timer reset button pushed), the counter and output signal are both reset to zero.

- ***Clock***

The divider module's <one_hz_clk> is outputted to the **clock** submodule which uses the signal as a sort of local clock to keep track of how many minutes and seconds have gone by. The time value is divided into two digits for minutes and two digits for seconds. At each positive edge of this local clock, the seconds and minutes values are incremented according to clock timing standards (i.e. when the lower seconds digit equals 9, reset to 0 and increment the higher seconds digit; when the higher seconds digit is equal to 5, reset to zero and increment the lower minutes digit).

The timer runs only if the system is in timer mode and the car is outside of the "Start" and "Finish" boxes on the map. Thus the module takes in <timer_mode> and <in_box> inputs and increments the timer only if these signals are high and low, respectively. The <in_box> signal is produced combinationally by checking the car's coordinates against the bounds of the "Start" and "Finish" using the inbox submodules. If the <timer_reset_sync> input is high (timer reset button pushed), the timer is reset and all digit values set to zero.

- ***Display_String***

The **display_string** submodule produces proper signals to run the Labkit's 16-character LED hex display and show the characters of each message. Every character on the display is made up of 40 LEDs (dots) and is displayed on the Labkit one at a time, as specified by the character index. Every number, letter, and symbol that can be displayed is mapped as a stream of dot values and looked up in a table according to ASCII encoding.

The submodule's Verilog code is a modified version of the given display_string module. Instead of the original 128-bit data stream that was passed in all at once and broken down into smaller ASCII character encoding, the index of the character being displayed `<char_index>` is outputted to the hexdisplay module. The 8-bit ASCII encoding corresponding to the index value and current map terrain is returned back to the submodule and used to look up the dot values for the display. This modification was made to make the implementation design of the system more efficient in size by reducing the large 128-bit data input bus to a 4- and 8-bit data exchange.

- **Synchronize**

In this system, all user inputs (map change, timer reset, and car directional control buttons) can be changed at any point in time and are thus asynchronous with the system clock. The **synchronize** module ensures that all user inputs are synchronized to the system clock.

The module accepts the system clock and user interface signals as inputs and outputs a synchronized version of the input to other modules in the object tracking system. As seen in Figure 20, the user inputs placed through the synchronizer are `<map_change, timer_reset, forward, back, right, left>`. The synchronizer essentially checks the value of the input at each rising clock edge and maps that value to a signal that is synchronous to the system clock. This module provides a safety check against the risk of metastable signals entering the system.

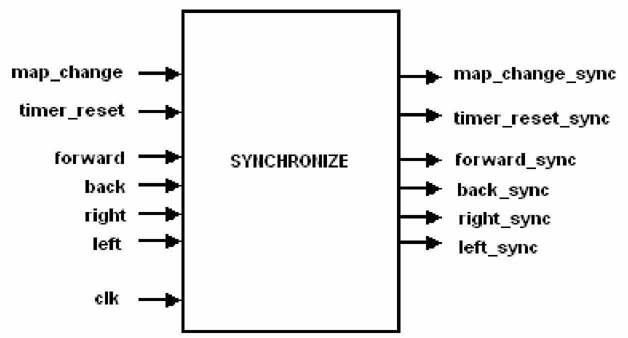


Figure 20: Synchronize module.

This module takes in user inputs from the Labkit and synchronizes them to the system clock.

- **Movement**

The **movement** module receives signals from the directional buttons as well as halting signals from the object tracker module. Using these input signals, the module determines the signal to send to the car to elicit movement. To connect the module to the remote control for the car, the signals are sent to the user I/O interface that is then wired to the remote control as seen in the following figure.

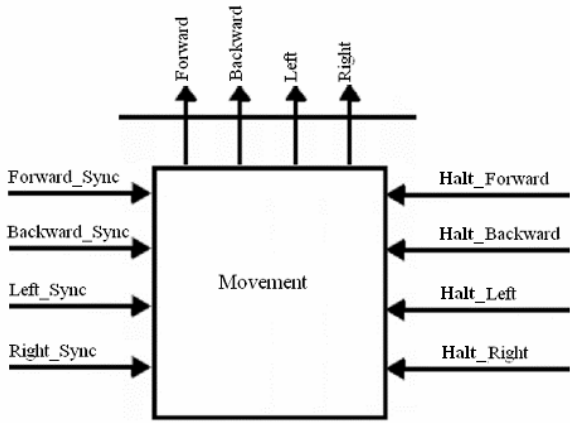


Figure 21: Movement module

This module takes button signals and halt signals and sends them to the remote control to move the car.

The car itself moves very quickly and picks up a lot of momentum from the movement. To prevent the car from quickly running off of the playing field, the signal sent to the remote control for `<forward>` and `<backward>` movement was pulsed such that the car would not have time to gain momentum. To pulse the `<forward>` and `<backward>` signal, the movement module would allow the car to move for 20 clock cycles, then would force the signal to break for 180 clock cycles before continuing. While this caused a jerky motion for the car, it effectively stopped the car from gaining too much momentum.

- **Position_Calculator**

To **position calculator** module reads and stores a stream of data from the video camera. Once the data is stored, the image is filtered and the coordinates of the car are determined from the filtered image. The module itself is composed of several submodules that captures the image from a stream, stores the image in a ZBT RAM, filters the image, and calculates the position from the filtered image as shown in the following figure.

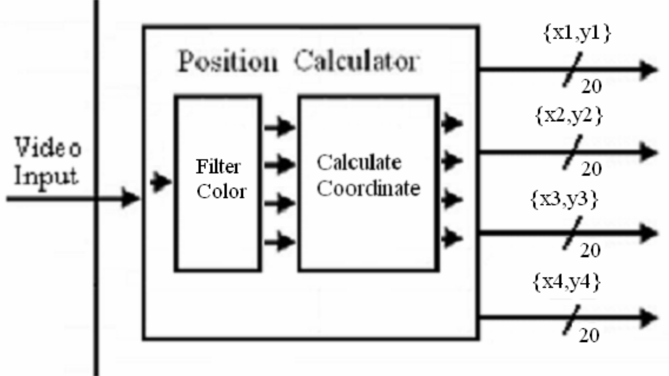


Figure 22: Position_Calculator module

This module takes in video data and outputs the coordinates of the car.

- ***XVGA***

  The **xvga** submodule generates xvga display signals for a 680x480 display so that the coordinates from the video image can be mapped to a display similar to the maps module. `<hcount>`, `<vcount>`, `<hsync>`, `<vsync>`, and `<blank>` signals are generated from an inputted 65mhz clock signal and are outputted for the coordinates submodule to use to determine car location.

- ***ZBT_6111***

  The **zbt_6111** submodule generates a ZBT RAM to store the video data that is streamed from the video camera and pipelines the data in. The data is not written to the RAM immediately but rather delayed by two cycles to ensure a stable signal. Thus, the data cannot be read immediately. Once the data is in the ZBT RAM, other modules and submodules can read it.

- ***Coordinates***

  The **coordinates** submodule determines whether or not a given point is one of the four corners of the car by filtering using the predefined coloring scheme. The colors used on the car were red, blue, green, and orange. Since the detected color intensity was dependant on such things as light intensity, which could be affected by a passing hand, the filtering was giving a broad range rather than a specific value to filter for.

  Once a given point was determined to be a corner, the coordinate of that point was passed to the maps module as `<x1[9:0], y1[9:0]>` (front left), `<x2[9:0], y2[9:0]>` (front right), `<x3[9:0], y3[9:0]>` (back right), or `<x4[9:0], y4[9:0]>` (back, left).

- ***Adv7185init***

  The **adv7185init** submodule initializes the raw video data `<source>` that is inputted from the Labkit's RCA input jack and sends it to the i2c submodule that converts it to a readable format. The camera itself is connected to the lower RCA phono jack located on the right side of the labkit.

  - ***I2C***

    The data from the video camera is not immediately usable and must first be converted. The raw data is first initialized and then passed from the adv7185init submodule to the **i2c** submodule. The i2c submodule then converts the raw streaming data from the video camera into a video format that can then be interpreted and decoded by the ntsc_decode submodule into NTSC video format.

- ***NTSC_Decode***

  The video data that is read in from the adv7185init submodule can be converted to NTSC (National Television Systems Committee) format that can then be readily used

to display a graphic or do conduct color analysis. Thus, the **ntsc_decode** module does this conversion and stores the data `<ycrcb[29:0]>` in a YCrCb format. NTSC video format separates the pixel signal into a luminescence component (Y) and a color difference (Cr and Cb) component that when combined, produce the color signal for a point.

- ***Gen_Model***
  The NTSC video data is a different form from the VGA form that is used to display on the computer screen. The VGA display uses a RGB format that splits the signal into a red, a blue, and a green component. The relationship between the NTSC and the RGB format is complicated and so the **gen_model** does the math. To covert from NTSC to RGB, the **gen_model** module takes a `<y[9:0],cr[9:0],cb[9:0]>` input and outputs a `<r[7:0],g[7:0],b[7:0]>` color data for a point.

- ***NTSC_To_ZBT***
  The ZBT memory is 36 bits wide while the NTSC video data passed in is 30 bits wide `<din[29:0]>`. The **ntsc_to_zbt** submodule takes the video data passed from the ntsc_decode submodule and generates the ZBT address to store the data in `<ntsc_addr[18:0]>`, as well as the 36 bit data to be stored in the ZBT `<ntsc_data[35:0]>`.

- ***DelayN***
  Because the data written into the ZBT is not immediately readable (as described in the **ZBT_6111** submodule), the clocking must be delayed such that the main module is synchronized with the ZBT.

Using these submodules, the **position_calculator** produces a set of coordinates to be passed to the maps module. The module uses an implementation of the ZBT RAM as instantiated in the zbt_6111 submodule. Video data is read in through the Labkit's RCA input jack and interpreted by the adv7185init and i2c submodules. The video data is then decoded by the ntsc_decode

submodule and stored in the ZBT by the ntsc_to_zbt submodule. The xvga submodule creates `<hcount>` and `<vcount>` signals to map the video data to computer x,y coordinates. Using these signals, the coordinates submodule filters the image and returns the address of the corners of the car. Finally, the delayN submodule synchronizes the ZBT read with the rest of the Labkit.

- **Labkit**

The labkit module is the top level module that connects all the individual modules together to form the complete object tracking system. This module is also where the values from buttons, switches, and LEDs on the Labkit for the user I/O interface are assigned.

Testing and Debugging

The **maps** module was tested simply by connecting the `<map_param>` output to LEDs and checking that changes were only observed when the `<change_map>` button was asserted and when the car was located within the "Start" bounding box.

The **carstate** module was tested by placing dummy car coordinates at different y-coordinates and checking that the state values (wired to Labkit LEDs) changed accordingly. The outputted check points were checked once the collision reporting system was set up. These were tested by running each of the points against a boundary and verifying that its motion was halted in the specified direction.

The **object_tracker** module was tested in several stages. The first step was to design each map terrain, which was a bit time consuming since each map had to be carefully laid out by pixel coordinates and drawn up in Paint. The next step was to correctly display the terrain elements and ROM images. Due to the once clock cycle delay between addressing and reading pixels from the ROM, there was a one pixel offset on the display screen (seen as a line of garbage pixels along the left edge of the screen). This problem was fixed by adding additional registers for the sync and blank values to delay each by one cycle to match the pixels.

The collision checking logic was then added to the map element modules and tested with the dummy car coordinates arranged in different states. This also required a bit of time to decide what the best manner of doing boundary checking would be, distinguishing between states, and determining which points to check for each state. The original number of checkpoints had to be reduced strategically in order to make computation and wiring a little more efficient. The actually checking did not work successfully at first due to an incorrect ROM memory address calculation. This bug was discovered by outputting the coordinate points and corresponding memory address to the Labkit HEX display.

The biggest problem with implementing the object_tracker module was finding a good balance between computation time and physical block space. As described in the module

implementation section of this report, different approaches in different combinations were attempted before using mostly ROM loaded images.

The **hexdisplay** module was added first as a debug tool for the object_tracker module, then as a project extension to implement message display and timer functionality for each of the map terrains. This module was also set up and tested in a series of stages. The first step was just to get the HEX display to work. The clock module for the timer was implemented next and tested to make sure the number of minutes and seconds were incremented and displayed correctly. Since the given display_16hex module only supported hex digits (0-F), the module was modified to support customized characters by expanding the bit size of the encoding. However, this modified module was soon scrapped when it was discovered that there was a provided module (string_display) that supported ASCII character displays.

The addition of the string_display module created wiring issues, causing the top half of the hex display to be blank. The collision checking part of the tracking system involves a lot of logic computation and large bus values being passed in between modules. The string_display module itself takes in a 128-bit data stream for the hex character message display. To alleviate some of the wiring issues, the string_dispaly module was modified to take in one 8-bit character encoding at a time. The system still remains temperamental to slight changes in the wiring. This was the most difficult bug to deal with since it was a hardware implementation issue as opposed to a logic and timing error.

The **synchronize** module was the same module provided from the Lab 2 project. Nevertheless the module was tested using the test bench waveform generator to verify that inputs that were out of sync with the system clock were mapped as a synchronized output signal.

The **movement** module was tested by connecting the remote control to the labkit and checking to see that the car received the correct signals and moved accordingly when the corresponding button was depressed. To check that the halting signals were processed correctly, the halting inputs were connected to the labkit switches and verified that the car would not turn or move in that direction.

One major bug was discovered when the remote control was first connected to the labkit. The remote control was originally built such that signals were sent when the directional button was depressed. When taking apart the remote control, it was discovered that the depressed buttons completed a circuit that caused the signal to be sent. It was originally assumed that this signal was a voltage difference, but was later discovered that the circuit was completed by current flow. This caused many problems in that the integrated circuits (MOSFETs) that were originally used completed the circuit regardless of the voltage applied. To solve the problem, Zener diodes were employed that have a property that inhibits current flow until a certain voltage level was achieved. Placed in series, the Zener diode made it such that an applied voltage from the user I/O ports allowed or blocked current flow, effectively opening or closing the circuit for the remote control.

The **position_calculator** module was tested in many stages. The first stage was to test to make sure that the video camera was recording properly. The pre-written zbt_6111_sample modules was downloaded and run to show that the camera was working correctly. These modules stored the black and white video data in a ZBT RAM then displayed it on the VGA display.

Once the video camera was shown to work in black and white, the next stage in the testing was to show that it could display in color. The code was modified and the gen_model submodule was added to convert the ntsc video format into the computer compatible rgb video format. Tests showed that the video camera captured the image, stored it in the ZBT RAM, and displayed the color image on the computer VGA display.

Once color was successfully implemented, the next stage was to filter the image for the color indicators located on the car. Here, several bugs were discovered and dealt with. Since the color detected by the video camera was dependant on the lighting conditions (due to luminescence), the color filtering had to be implemented as a range of color values rather than set values. Because of this, the video camera had a tendency of picking up background color such as the blue mat the labkit rested on, the yellow tinted floor, and the blue walls. To deal with this issue, a white playing field was created for the car to run on. While this eliminated much of the

background noise, it could not remove all of the background color and so errant background color still seeped through.

Once the filtering was in place, the next stage was to determine the coordinates of the car from the filtered image. Since each corner was designated with a different color indicator, the different corners were differentiated using 4 different color filters. After filtering, several bugs were uncovered. The first was that depending on the distance between the video camera and the car, the video camera would pick up many points within the color filtering range. Only picking one point and returning those coordinates dealt this with bug. This caused another bug to show up. Due to the background color picked up by the video camera, the point selected was sometimes not one that corresponded with a car corner. Stricter color filtering decreased the frequency that this happened, but could not completely remove this problem.

Conclusion

Although each separate part functioned correctly, they unfortunately were not successfully integrated by the end of the project. The car controls and map terrains worked properly, but the car coordinates would not register correctly on screen. The initial suspicion was the conflict between different rate clocks that the video camera and monitor display were running at. Attempts to latch the coordinate values and synchronize them with the display clock provided no resolution to the problem.

This could also have been a problem with scale mapping from the larger resolution field of the camera to the smaller resolution of the VGA display. One possible solution to test would be converting the display to XVGA, which would have required changing the parameters of the image but would have solved the issue of having separate clocks.

Another issue that arose involved the accurate and stable calculation of the car coordinates. Due to the sporadic nature of the position calculator, the coordinates passed to the object tracker was not stable. Also, the video camera often picked up background color and often caused the position calculator to return incorrect coordinates. If the coordinates had been displayed correctly, it would have been difficult to enforce the boundary checking since points may not have been stationary long enough to produce accurate halting signals. One possible solution to this problem could have been enlarging the field terrain to have wider pathways and wall boundaries.

A lot has been learned about optimizing modules to address timing or space constraints, dealing with the problems and bugs that can arise from the imperfections of the analog world, and the difficulty of mapping these imperfections to a digital interface. Although disappointing that the project did not work out as hoped, the project has been definitely been good exposure to and practice for designing and implementing complex digital systems.

```
////////////////////////////////////////////////////////////////////////////////
//
// maps: handles map selection
//
////////////////////////////////////////////////////////////////////////////////

module maps (clk, mapchange, mapselect, car_1x, car_1y, car_2x, car_2y, car_3x,
car_3y, car_4x, car_4y, map_param);
	parameter START_UPX = 349;
	parameter START_LOWX = 290;
	parameter START_UPY = 266;
	parameter START_LOWY = 213;

	input clk;
	input mapchange;
	input [1:0] mapselect;
	input [9:0] car_1x;
	input [9:0] car_1y;
	input [9:0] car_2x;
	input [9:0] car_2y;
	input [9:0] car_3x;
	input [9:0] car_3y;
	input [9:0] car_4x;
	input [9:0] car_4y;

	output [1:0] map_param;

	reg [1:0] map_param = 2'b00;
	wire canchange;

	// is car in START box?
	inbox check(car_1x, car_1y, car_2x, car_2y, car_3x, car_3y, car_4x, car_4y,
canchange);
	defparam check.START_UPX   = 349;
	defparam check.START_LOWX = 290;
	defparam check.START_UPY = 266;
	defparam check.START_LOWY = 213;

	always @ (posedge clk)
	begin
		// must be witin START box to change map terrains
		if (canchange && mapchange)
			map_param <= mapselect;
		end

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// inbox: checks if car is within given bounds
//
////////////////////////////////////////////////////////////////////////////////

module inbox(car_1x, car_1y, car_2x, car_2y, car_3x, car_3y, car_4x, car_4y, inside);

	parameter START_UPX = 100;
	parameter START_LOWX = 0;
	parameter START_UPY = 100;
	parameter START_LOWY = 0;

	input [9:0] car_1x;
	input [9:0] car_1y;
	input [9:0] car_2x;
	input [9:0] car_2y;
	input [9:0] car_3x;
	input [9:0] car_3y;
	input [9:0] car_4x;
	input [9:0] car_4y;

	output inside;

	assign inside = (car_1x > START_LOWX && car_2x > START_LOWX && car_3x >
START_LOWX && car_4x > START_LOWX &&
				car_1x < START_UPX && car_2x < START_UPX && car_3x <
START_UPX && car_4x < START_UPX &&
				car_1y > START_LOWY && car_2y > START_LOWY && car_3y >
START_LOWY && car_4y > START_LOWY &&
				car_1y < START_UPY && car_2y < START_UPY && car_3y <
START_UPY && car_4y < START_UPY) ? 1 : 0;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// carstate: determines state (direction) of car and boundary points to check
//
////////////////////////////////////////////////////////////////////////////////


module carstate(clk, car_1x, car_1y, car_2x, car_2y,
                     car_3x, car_3y, car_4x, car_4y,
                     x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f,
                     x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b,
                     x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt);

	parameter OFFSET = 10'd5;

	input clk;
	input [9:0] car_1x;
	input [9:0] car_1y;
	input [9:0] car_2x;
	input [9:0] car_2y;
	input [9:0] car_3x;
	input [9:0] car_3y;
	input [9:0] car_4x;
	input [9:0] car_4y;

	output [9:0] x1f;
	output [9:0] y1f;
	output [9:0] x2f;
	output [9:0] y2f;
	output [9:0] x3f;
	output [9:0] y3f;
	output [9:0] x4f;
	output [9:0] y4f;
	output [9:0] x1b;
	output [9:0] y1b;
	output [9:0] x2b;
	output [9:0] y2b;
	output [9:0] x3b;
	output [9:0] y3b;
	output [9:0] x4b;
	output [9:0] y4b;
	output [9:0] x1r;
	output [9:0] y1r;
	output [9:0] x2r;
	output [9:0] y2r;
	output [9:0] x1lt;
	output [9:0] y1lt;
	output [9:0] x2lt;
	output [9:0] y2lt;

	reg [2:0] state = 0;
	reg [9:0] x1f;
	reg [9:0] y1f;
	reg [9:0] x2f;
	reg [9:0] y2f;
	reg [9:0] x3f;
	reg [9:0] y3f;
	reg [9:0] x4f;
	reg [9:0] y4f;
	reg [9:0] x1b;
	reg [9:0] y1b;
	reg [9:0] x2b;
	reg [9:0] y2b;
```

```
	reg [9:0] x3b;
	reg [9:0] y3b;
	reg [9:0] x4b;
	reg [9:0] y4b;
	reg [9:0] x1r;
	reg [9:0] y1r;
	reg [9:0] x2r;
	reg [9:0] y2r;
	reg [9:0] x1lt;
	reg [9:0] y1lt;
	reg [9:0] x2lt;
	reg [9:0] y2lt;

	always @ (posedge clk)
		begin
			// north
			if ((car_1y == car_2y) && (car_1y < car_4y))
				state <= 3'b000;
			// northeast
			else if ((car_1y < car_2y) && (car_1y < car_3y) && (car_1y <
car_4y))
				state <= 3'b001;
			// east
			else if ((car_1y == car_4y) && (car_1y < car_2y))
				state <= 3'b010;
			// southeast
			else if ((car_4y < car_1y) && (car_4y < car_2y) && (car_4y <
car_3y))
				state <= 3'b011;
			// south
			else if ((car_1y == car_2y) && (car_1y > car_4y))
				state <= 3'b100;
			// southwest
			else if ((car_3y < car_1y) && (car_3y < car_2y) && (car_3y <
car_4y))
				state <= 3'b101;
			// west
			else if ((car_1y == car_4y) && (car_1y > car_2y))
				state <= 3'b110;
			// northwest
			else if ((car_2y < car_1y) && (car_2y < car_3y) && (car_2y <
car_4y))
				state <= 3'b111;
		end

	always @ (state or car_1x or car_1y or car_2x or car_2y or car_3x or car_3y or
car_4x or car_4y)
		begin
			case (state)
				// north
				3'b000:
				begin
					x1f = car_1x;
					y1f = car_1y - OFFSET;
					x2f = car_1x;
					y2f = car_1y - OFFSET;
					x3f = car_2x;
					y3f = car_2y - OFFSET;
					x4f = car_2x;
					y4f = car_2y - OFFSET;
					x1b = car_3x;
					y1b = car_3y + OFFSET;
					x2b = car_3x;
```

```
        y2b = car_3y + OFFSET;
        x3b = car_4x;
        y3b = car_4y + OFFSET;
        x4b = car_4x;
        y4b = car_4y + OFFSET;
        x1r = car_2x + OFFSET;
        y1r = car_2y;
        x2r = car_3x + OFFSET;
        y2r = car_3y;
        x1lt = car_1x - OFFSET;
        y1lt = car_1y;
        x2lt = car_4x - OFFSET;
        y2lt = car_4y;
end
// northeast
3'b001:
begin
        x1f = car_1x;
        y1f = car_1y - OFFSET;
        x2f = car_1x + OFFSET;
        y2f = car_1y;
        x3f = car_2x;
        y3f = car_2y - OFFSET;
        x4f = car_2x + OFFSET;
        y4f = car_2y;
        x1b = car_3x;
        y1b = car_3y + OFFSET;
        x2b = car_3x - OFFSET;
        y2b = car_3y;
        x3b = car_4x;
        y3b = car_4y + OFFSET;
        x4b = car_4x - OFFSET;
        y4b = car_4y;
        x1r = car_2x + OFFSET;
        y1r = car_2y;
        x2r = car_3x + OFFSET;
        y2r = car_3y;
        x1lt = car_1x - OFFSET;
        y1lt = car_1y;
        x2lt = car_4x - OFFSET;
        y2lt = car_4y;
end
// east
3'b010:
begin
        x1f = car_1x + OFFSET;
        y1f = car_1y;
        x2f = car_1x + OFFSET;
        y2f = car_1y;
        x3f = car_2x + OFFSET;
        y3f = car_2y;
        x4f = car_2x + OFFSET;
        y4f = car_2y;
        x1b = car_3x - OFFSET;
        y1b = car_3y;
        x2b = car_3x - OFFSET;
        y2b = car_3y;
        x3b = car_4x - OFFSET;
        y3b = car_4y;
        x4b = car_4x - OFFSET;
        y4b = car_4y;
        x1r = car_2x;
        y1r = car_2y + OFFSET;
```

```
        x2r = car_3x;
        y2r = car_3y + OFFSET;
        x1lt = car_1x;
        y1lt = car_1y - OFFSET;
        x2lt = car_4x;
        y2lt = car_4y - OFFSET;
end
// southeast
3'b011:
begin
        x1f = car_1x + OFFSET;
        y1f = car_1y;
        x2f = car_1x;
        y2f = car_1y + OFFSET;
        x3f = car_2x + OFFSET;
        y3f = car_2y;
        x4f = car_2x;
        y4f = car_2y + OFFSET;
        x1b = car_3x - OFFSET;
        y1b = car_3y;
        x2b = car_3x;
        y2b = car_3y - OFFSET;
        x3b = car_4x - OFFSET;
        y3b = car_4y;
        x4b = car_4x;
        y4b = car_4y - OFFSET;
        x1r = car_2x;
        y1r = car_2y - OFFSET;
        x2r = car_3x;
        y2r = car_3y - OFFSET;
        x1lt = car_1x;
        y1lt = car_1y + OFFSET;
        x2lt = car_4x;
        y2lt = car_4y + OFFSET;
end
// south
3'b100:
begin
        x1f = car_1x;
        y1f = car_1y + OFFSET;
        x2f = car_1x;
        y2f = car_1y + OFFSET;
        x3f = car_2x;
        y3f = car_2y + OFFSET;
        x4f = car_2x;
        y4f = car_2y + OFFSET;
        x1b = car_3x;
        y1b = car_3y - OFFSET;
        x2b = car_3x;
        y2b = car_3y - OFFSET;
        x3b = car_4x;
        y3b = car_4y - OFFSET;
        x4b = car_4x;
        y4b = car_4y - OFFSET;
        x1r = car_2x - OFFSET;
        y1r = car_2y;
        x2r = car_3x - OFFSET;
        y2r = car_3y;
        x1lt = car_1x + OFFSET;
        y1lt = car_1y;
        x2lt = car_4x + OFFSET;
        y2lt = car_4y;
end
```

```
// southwest
3'b101:
begin
	x1f = car_1x;
	y1f = car_1y + OFFSET;
	x2f = car_1x - OFFSET;
	y2f = car_1y;
	x3f = car_2x;
	y3f = car_2y + OFFSET;
	x4f = car_2x - OFFSET;
	y4f = car_2y;
	x1b = car_3x;
	y1b = car_3y - OFFSET;
	x2b = car_3x + OFFSET;
	y2b = car_3y;
	x3b = car_4x;
	y3b = car_4y - OFFSET;
	x4b = car_4x + OFFSET;
	y4b = car_4y;
	x1r = car_2x - OFFSET;
	y1r = car_2y;
	x2r = car_3x - OFFSET;
	y2r = car_3y;
	x1lt = car_1x + OFFSET;
	y1lt = car_1y;
	x2lt = car_4x + OFFSET;
	y2lt = car_4y;
end
// west
3'b110:
begin
	x1f = car_1x - OFFSET;
	y1f = car_1y;
	x2f = car_1x - OFFSET;
	y2f = car_1y;
	x3f = car_2x - OFFSET;
	y3f = car_2y;
	x4f = car_2x - OFFSET;
	y4f = car_2y;
	x1b = car_3x + OFFSET;
	y1b = car_3y;
	x2b = car_3x + OFFSET;
	y2b = car_3y;
	x3b = car_4x + OFFSET;
	y3b = car_4y;
	x4b = car_4x + OFFSET;
	y4b = car_4y;
	x1r = car_2x;
	y1r = car_2y - OFFSET;
	x2r = car_3x;
	y2r = car_3y - OFFSET;
	x1lt = car_1x;
	y1lt = car_1y + OFFSET;
	x2lt = car_4x;
	y2lt = car_4y + OFFSET;
end
// northwest
3'b111:
begin
	x1f = car_1x - OFFSET;
	y1f = car_1y;
	x2f = car_1x;
	y2f = car_1y - OFFSET;
```

```
                                        x3f = car_2x - OFFSET;
                                        y3f = car_2y;
                                        x4f = car_2x;
                                        y4f = car_2y - OFFSET;
                                        x1b = car_3x + OFFSET;
                                        y1b = car_3y;
                                        x2b = car_3x;
                                        y2b = car_3y + OFFSET;
                                        x3b = car_4x + OFFSET;
                                        y3b = car_4y;
                                        x4b = car_4x;
                                        y4b = car_4y + OFFSET;
                                        x1r = car_2x;
                                        y1r = car_2y - OFFSET;
                                        x2r = car_3x;
                                        y2r = car_3y - OFFSET;
                                        x1lt = car_1x;
                                        y1lt = car_1y + OFFSET;
                                        x2lt = car_4x;
                                        y2lt = car_4y + OFFSET;
                                end

                        endcase
                end //always block


        endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// object tracker: source of all power!!!
//
////////////////////////////////////////////////////////////////////////////////


module object_tracker(clk, map_param, reset,
                             car_1x, car_1y, car_2x, car_2y,
                             car_3x, car_3y, car_4x, car_4y,
                             x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f,
                             x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b,
                             x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt,
                             hsync, vsync, blank, pix_clk, pixel,
                             halt_front, halt_back, halt_right, halt_left);

       parameter STARTX = 10'd292;
       parameter    STARTY = 10'd215;

       input clk;
       input [1:0] map_param;
       input reset;

       input [9:0] car_1x;
       input [9:0] car_1y;
       input [9:0] car_2x;
       input [9:0] car_2y;
       input [9:0] car_3x;
       input [9:0] car_3y;
       input [9:0] car_4x;
       input [9:0] car_4y;

     input [9:0] x1f;             // front boundary points
     input [9:0] y1f;
     input [9:0] x2f;
     input [9:0] y2f;
     input [9:0] x3f;
     input [9:0] y3f;
     input [9:0] x4f;
     input [9:0] y4f;
     input [9:0] x1b;             // back boundary points
     input [9:0] y1b;
     input [9:0] x2b;
     input [9:0] y2b;
     input [9:0] x3b;
     input [9:0] y3b;
     input [9:0] x4b;
     input [9:0] y4b;
     input [9:0] x1r;             // right boundary points
     input [9:0] y1r;
     input [9:0] x2r;
     input [9:0] y2r;
     input [9:0] x1lt;            // left boundary points
     input [9:0] y1lt;
     input [9:0] x2lt;
     input [9:0] y2lt;

       output hsync;
       output vsync;
       output blank;
       output pix_clk;
       output [2:0] pixel;
       output halt_front;
```

```
	output halt_back;
	output halt_right;
	output halt_left;

	reg [2:0] i_pixel = 0;
	reg halt_front = 0;
	reg halt_back = 0;
	reg halt_right = 0;
	reg halt_left = 0;
	reg hsync = 0;
	reg vsync = 0;
	reg blank = 0;
	reg delay_hsync = 0;
	reg delay_vsync = 0;
	reg delay_blank = 0;

	// vga module
	wire [9:0] hcount;
	wire [9:0] vcount;
	wire vga_hsync;
	wire vga_vsync;
	wire vga_blank;
      vga_sync vga_signals(clk, vga_hsync, vga_vsync, hcount, vcount, pix_clk,
vga_blank);

	// "car"
	wire [2:0] c1_pixel;
	wire [2:0] c2_pixel;
	wire [2:0] c3_pixel;
	wire [2:0] c4_pixel;
	blob car_flt(car_1x, car_1y, hcount, vcount, c1_pixel);     // 1
	defparam car_flt.COLOR = 3'b110;
	blob car_frt(car_2x, car_2y, hcount, vcount, c2_pixel);     // 2
	defparam car_frt.COLOR = 3'b110;
	blob car_brt(car_3x, car_3y, hcount, vcount, c3_pixel);     // 3
	defparam car_brt.COLOR = 3'b100;
	blob car_blt(car_4x, car_4y, hcount, vcount, c4_pixel);     // 4
	defparam car_blt.COLOR = 3'b100;

	// map change box
	wire [2:0] start_pixel;
	blob start(STARTX, STARTY, hcount, vcount, start_pixel);
	defparam start.COLOR = 3'b010;
	defparam start.WIDTH = 57;
	defparam start.HEIGHT = 49;

	// open map terrain
	wire [2:0] open_pixel;
	wire open_front;
	wire open_back;
	wire open_right;
	wire open_left;
	wire open_hsync;
	wire open_vsync;
	wire open_blank;
	open map00_image(vga_hsync, vga_vsync, vga_blank, hcount, vcount, x1f, y1f,
x2f, y2f, x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r,
y2r, x1lt, y1lt, x2lt, y2lt, open_hsync, open_vsync, open_blank, open_pixel,
open_front, open_back, open_right, open_left);

	// minefield map terrain
	wire [2:0] mine_pixel;
	wire mine_front;
```

```
	wire mine_back;
	wire mine_right;
	wire mine_left;
	wire mine_hsync;
	wire mine_vsync;
	wire mine_blank;
	minefield map01_image(clk, vga_hsync, vga_vsync, vga_blank, hcount, vcount,
pix_clk, x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b,
y4b, x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt, mine_hsync, mine_vsync, mine_blank,
mine_pixel, mine_front, mine_back, mine_right, mine_left);

	// maze map terrain
	wire [2:0] maze_pixel;
	wire maze_front;
	wire maze_back;
	wire maze_right;
	wire maze_left;
	wire maze_hsync;
	wire maze_vsync;
	wire maze_blank;
	maze map10_image(clk, vga_hsync, vga_vsync, vga_blank, hcount, vcount, pix_clk,
x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r,
y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt, maze_hsync, maze_vsync, maze_blank, maze_pixel,
maze_front, maze_back, maze_right, maze_left);

	// race map terrain
	wire [2:0] race_pixel;
	wire race_front;
	wire race_back;
	wire race_right;
	wire race_left;
	wire race_hsync;
	wire race_vsync;
	wire race_blank;
	racetrack map11_image(clk, vga_hsync, vga_vsync, vga_blank, hcount, vcount,
pix_clk, x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b,
y4b, x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt, race_hsync, race_vsync, race_blank,
race_pixel, race_front, race_back, race_right, race_left);

	always @(posedge clk)
	begin

		if (reset)
			begin
				i_pixel <= open_pixel;
				halt_front <= open_front;
				halt_back <= open_back;
				halt_right <= open_right;
				halt_left <= open_left;
				hsync <= open_hsync;
				vsync <= open_vsync;
				blank <= open_blank;
			end

		else
			begin

				case	(map_param) // map_param from MAPS
					// output proper pixel to display
					// open field
					2'b00:
						begin
							i_pixel <= open_pixel;
```

```
                                        halt_front <= open_front;
                                        halt_back <= open_back;
                                        halt_right <= open_right;
                                        halt_left <= open_left;
                                        hsync <= open_hsync;
                                        vsync <= open_vsync;
                                        blank <= open_blank;
                                end
                        // mine field
                        2'b01:
                                begin
                                        i_pixel <= mine_pixel;
                                        halt_front <= mine_front;
                                        halt_back <= mine_back;
                                        halt_right <= mine_right;
                                        halt_left <= mine_left;
                                        delay_hsync <= mine_hsync;
                                        delay_vsync <= mine_vsync;
                                        delay_blank <= mine_blank;
                                        hsync <= delay_hsync;
                                        vsync <= delay_vsync;
                                        blank <= delay_blank;
                                end
                        // maze
                        2'b10:
                                begin
                                        i_pixel <= maze_pixel;
                                        halt_front <= maze_front;
                                        halt_back <= maze_back;
                                        halt_right <= maze_right;
                                        halt_left <= maze_left;
                                        delay_hsync <= maze_hsync;
                                        delay_vsync <= maze_vsync;
                                        delay_blank <= maze_blank;
                                        hsync <= delay_hsync;
                                        vsync <= delay_vsync;
                                        blank <= delay_blank;
                                end
                        // racetrack
                        2'b11:
                                begin
                                        i_pixel <= race_pixel;
                                        halt_front <= race_front;
                                        halt_back <= race_back;
                                        halt_right <= race_right;
                                        halt_left <= race_left;
                                        delay_hsync <= race_hsync;
                                        delay_vsync <= race_vsync;
                                        delay_blank <= race_blank;
                                        hsync <= delay_hsync;
                                        vsync <= delay_vsync;
                                        blank <= delay_blank;
                                end
                endcase
            end // else

    end    // always

   assign pixel = c1_pixel | c2_pixel | c3_pixel | c4_pixel | start_pixel | i_pixel;

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// open: open map terrain
//
///////////////////////////////////////////////////////////////////////////////

module open (in_hsync, in_vsync, in_blank, hcount, vcount, x1f, y1f, x2f, y2f, x3f,
y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt, y1lt,
x2lt, y2lt, out_hsync, out_vsync, out_blank, pixel, halt_f, halt_b, halt_r, halt_lt);

   parameter vert_1x = 10'd0;
   parameter vert_1y = 10'd0;
   parameter vert_2x = 10'd634;
   parameter vert_2y = 10'd0;
   parameter horz_1x = 10'd0;
   parameter horz_1y = 10'd0;
   parameter horz_2x = 10'd0;
   parameter horz_2y = 10'd474;
   parameter zero = 10'd0;


   input [9:0] hcount;     // horizontal index of current pixel
   input [9:0] vcount;   // vertical index of current pixel
   input [9:0] x1f;        // front boundary points
   input [9:0] y1f;
   input [9:0] x2f;
   input [9:0] y2f;
   input [9:0] x3f;
   input [9:0] y3f;
   input [9:0] x4f;
   input [9:0] y4f;
   input [9:0] x1b;        // back boundary points
   input [9:0] y1b;
   input [9:0] x2b;
   input [9:0] y2b;
   input [9:0] x3b;
   input [9:0] y3b;
   input [9:0] x4b;
   input [9:0] y4b;
   input [9:0] x1r;        // right boundary points
   input [9:0] y1r;
   input [9:0] x2r;
   input [9:0] y2r;
   input [9:0] x1lt;       // left boundary points
   input [9:0] y1lt;
   input [9:0] x2lt;
   input [9:0] y2lt;

   input in_hsync;  // hsync signal
   input in_vsync;  // vsync signal
   input in_blank;  // blank signal

   output [2:0] pixel;     // open field's pixel
   output halt_f;   // halt move forward?
   output halt_b;   // halt move backward?
   output halt_r;   // halt move right?
   output halt_lt;  // halt move left?
   output out_hsync;  // hsync signal
   output out_vsync;  // vsync signal
   output out_blank;  // blank signal

   // boundary lines
   wire [2:0] vert1_pixel;
```

```
   wire [2:0] vert2_pixel;
   wire [2:0] horz1_pixel;
   wire [2:0] horz2_pixel;
   vert_line vert1(vert_1x, vert_1y, zero, zero, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, vert1_pixel, vert1_f, vert1_b, vert1_r, vert1_lt);
       defparam vert1.HEIGHT1 = 479;
   vert_line vert2(vert_2x, vert_2y, zero, zero, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, vert2_pixel, vert2_f, vert2_b, vert2_r, vert2_lt);
       defparam vert2.HEIGHT1 = 479;
   horz_line horz1(horz_1x, zero, zero, horz_1y, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, horz1_pixel, horz1_f, horz1_b, horz1_r, horz1_lt);
       defparam horz1.WIDTH1 = 639;
   horz_line horz2(horz_2x, zero, zero, horz_2y, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, horz2_pixel, horz2_f, horz2_b, horz2_r, horz2_lt);
       defparam horz2.WIDTH1 = 639;

   // display signals
   assign out_hsync = in_hsync;
   assign out_vsync = in_vsync;
   assign out_blank = in_blank;
   // display pixel
   assign pixel = vert1_pixel | vert2_pixel | horz1_pixel | horz2_pixel;
   // boundary check: if points within line bounds, HALT!
   assign halt_f = vert1_f | vert2_f | horz1_f | horz2_f;
   assign halt_b = vert1_b | vert2_b | horz1_b | horz2_b;
   assign halt_r = vert1_r | vert2_r | horz1_r | horz2_r;
   assign halt_lt = vert1_lt | vert2_lt | horz1_lt | horz2_lt;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// minefield: minefield map terrain
//
////////////////////////////////////////////////////////////////////////////////

module minefield (clk, in_hsync, in_vsync, in_blank, hcount, vcount, pix_clk,
                       x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f,
                       x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b,
                       x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt,
                       out_hsync, out_vsync, out_blank, pixel, halt_f, halt_b, halt_r,
halt_lt);

   // boundary lines
   parameter zero = 10'd0;
   parameter vert_1x = 10'd0;
   parameter vert_1y = 10'd0;
   parameter vert_2x = 10'd634;
   parameter vert_2y = 10'd0;
   parameter horz_1x = 10'd0;
   parameter horz_1y = 10'd0;
   parameter horz_2x = 10'd0;
   parameter horz_2y = 10'd474;

   input clk;       // video clock
   input in_hsync;  // hsync signal
   input in_vsync;  // vsync signal
   input in_blank;  // blank signal
   input [9:0] hcount;     // current x,y location of pixel
   input [9:0] vcount;
   input pix_clk;   // pixel clock

   input [9:0] x1f;  // front boundary points
   input [9:0] y1f;
   input [9:0] x2f;
   input [9:0] y2f;
   input [9:0] x3f;
   input [9:0] y3f;
   input [9:0] x4f;
   input [9:0] y4f;
   input [9:0] x1b;  // back boundary points
   input [9:0] y1b;
   input [9:0] x2b;
   input [9:0] y2b;
   input [9:0] x3b;
   input [9:0] y3b;
   input [9:0] x4b;
   input [9:0] y4b;
   input [9:0] x1r;  // right boundary points
   input [9:0] y1r;
   input [9:0] x2r;
   input [9:0] y2r;
   input [9:0] x1lt; // left boundary points
   input [9:0] y1lt;
   input [9:0] x2lt;
   input [9:0] y2lt;

   output [2:0] pixel;     // pixel value output
   output halt_f;   // halt move forward?
   output halt_b;   // halt move backward?
   output halt_r;   // halt move right?
   output halt_lt;  // halt move left?
   output out_hsync;  // hsync signal
```

```
   output out_vsync;  // vsync signal
   output out_blank;  // blank signal

   reg out_hsync = 0;
   reg out_vsync = 0;
   reg out_blank = 0;
   reg wait_hsync = 0;  // latch syncs and blanks for one clk cycle: match pixel
offset
   reg wait_vsync = 0;
   reg wait_blank = 0;

   // CALCULATE ROM ADDRESS OF EACH PIXEL/POINT
   // the memory address is x/2 + y/2 * 320
   // (4 pixels per memory location, since image is 320x240, and
   // display is 640x480).

   reg [16:0]       pixaddr;
   reg [16:0]       f1addr;
   reg [16:0]       f2addr;
   reg [16:0]       f3addr;
   reg [16:0]       f4addr;
   reg [16:0]       b1addr;
   reg [16:0]       b2addr;
   reg [16:0]       b3addr;
   reg [16:0]       b4addr;
   reg [16:0]       r1addr;
   reg [16:0]       r2addr;
   reg [16:0]       lt1addr;
   reg [16:0]       lt2addr;

   reg [16:0]       pixaddr_reg;
   reg [16:0]       f1addr_reg;
   reg [16:0]       f2addr_reg;
   reg [16:0]       f3addr_reg;
   reg [16:0]       f4addr_reg;
   reg [16:0]       b1addr_reg;
   reg [16:0]       b2addr_reg;
   reg [16:0]       b3addr_reg;
   reg [16:0]       b4addr_reg;
   reg [16:0]       r1addr_reg;
   reg [16:0]       r2addr_reg;
   reg [16:0]       lt1addr_reg;
   reg [16:0]       lt2addr_reg;

       // y/2 * 320
   always @(posedge clk)
   begin
     pixaddr <= (hcount==0 & vcount==0) ? 0
             : (hcount==0 & pix_clk & ~vcount[0]) ? pixaddr + 320 : pixaddr;
     f1addr = y1f[9:1] * 320;
     f2addr = y2f[9:1] * 320;
     f3addr = y3f[9:1] * 320;
     f4addr = y4f[9:1] * 320;
     b1addr = y1b[9:1] * 320;
     b2addr = y2b[9:1] * 320;
     b3addr = y3b[9:1] * 320;
     b4addr = y4b[9:1] * 320;
     r1addr = y1r[9:1] * 320;
     r2addr = y2r[9:1] * 320;
     lt1addr = y1lt[9:1] * 320;
     lt2addr = y2lt[9:1] * 320;
   end
```

```
    // x/2 + (y/2*320)
   always @(posedge clk)
   begin
      pixaddr_reg <= {8'b0,hcount[9:1]} + pixaddr[16:0];
      f1addr_reg <= {8'b0,x1f[9:1]} + f1addr[16:0];
      f2addr_reg <= {8'b0,x2f[9:1]} + f2addr[16:0];
      f3addr_reg <= {8'b0,x3f[9:1]} + f3addr[16:0];
      f4addr_reg <= {8'b0,x4f[9:1]} + f4addr[16:0];
      b1addr_reg <= {8'b0,x1b[9:1]} + b1addr[16:0];
      b2addr_reg <= {8'b0,x2b[9:1]} + b2addr[16:0];
      b3addr_reg <= {8'b0,x3b[9:1]} + b3addr[16:0];
      b4addr_reg <= {8'b0,x4b[9:1]} + b4addr[16:0];
      r1addr_reg <= {8'b0,x1r[9:1]} + r1addr[16:0];
      r2addr_reg <= {8'b0,x2r[9:1]} + r2addr[16:0];
      lt1addr_reg <= {8'b0,x1lt[9:1]} + lt1addr[16:0];
      lt2addr_reg <= {8'b0,x2lt[9:1]} + lt2addr[16:0];
   end

    // latch sync and blank
   always @(posedge clk)
   begin
      wait_hsync <= in_hsync;
      wait_vsync <= in_vsync;
      wait_blank <= in_blank;
      out_hsync <= wait_hsync;
      out_vsync <= wait_vsync;
      out_blank <= wait_blank;
   end


   // instantiate the image roms
   wire rom_pixel;
   wire halt1_f;
   wire halt2_f;
   wire halt3_f;
   wire halt4_f;
   wire halt1_b;
   wire halt2_b;
   wire halt3_b;
   wire halt4_b;
   wire halt1_r;
   wire halt2_r;
   wire halt1_lt;
   wire halt2_lt;
   mine1 rom1(pixaddr_reg, clk, rom_pixel);
   mine2 rom2(f1addr_reg, f2addr_reg, clk, clk, halt1_f, halt2_f);
   mine2 rom3(f3addr_reg, f4addr_reg, clk, clk, halt3_f, halt4_f);
   mine2 rom4(b1addr_reg, b2addr_reg, clk, clk, halt1_b, halt2_b);
   mine2 rom5(b3addr_reg, b4addr_reg, clk, clk, halt3_b, halt4_b);
   mine2 rom6(r1addr_reg, r2addr_reg, clk, clk, halt1_r, halt2_r);
   mine2 rom7(lt1addr_reg, lt2addr_reg, clk, clk, halt1_lt, halt2_lt);

   // instantiate boundary lines
   wire [2:0] vert1_pixel;
   wire [2:0] vert2_pixel;
   wire [2:0] horz1_pixel;
   wire [2:0] horz2_pixel;

   vert_line vert1(vert_1x, vert_1y, zero, zero, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, vert1_pixel, vert1_f, vert1_b, vert1_r, vert1_lt);
      defparam vert1.HEIGHT1 = 479;
```

```
   vert_line vert2(vert_2x, vert_2y, zero, zero, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, vert2_pixel, vert2_f, vert2_b, vert2_r, vert2_lt);
       defparam vert2.HEIGHT1 = 479;
   horz_line horz1(horz_1x, zero, zero, horz_1y, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, horz1_pixel, horz1_f, horz1_b, horz1_r, horz1_lt);
       defparam horz1.WIDTH1 = 639;
   horz_line horz2(horz_2x, zero, zero, horz_2y, hcount, vcount, x1f, y1f, x2f, y2f,
x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt, horz2_pixel, horz2_f, horz2_b, horz2_r, horz2_lt);
       defparam horz2.WIDTH1 = 639;

   wire [2:0] b_pixel = vert1_pixel | vert2_pixel | horz1_pixel | horz2_pixel;
   wire b_halt_f = vert1_f | vert2_f | horz1_f | horz2_f;
   wire b_halt_b = vert1_b | vert2_b | horz1_b | horz2_b;
   wire b_halt_r = vert1_r | vert2_r | horz1_r | horz2_r;
   wire b_halt_lt = vert1_lt | vert2_lt | horz1_lt | horz2_lt;

   // pixel output
   assign pixel = (rom_pixel == 1) ? (3'b100 | b_pixel) : (3'b000 | b_pixel);
   // check boundary points: if not black (1'b0) pixel, HALT!
   assign halt_f = ((halt1_f != 0) | (halt2_f != 0) | (halt3_f != 0) | (halt4_f != 0)
| b_halt_f) ? 1 : 0;
   assign halt_b = ((halt1_b != 0) | (halt2_b != 0) | (halt3_b != 0) | (halt4_b != 0)
| b_halt_b) ? 1 : 0;
   assign halt_r = ((halt1_r != 0) | (halt2_r != 0) | b_halt_r) ? 1 : 0;
   assign halt_lt = ((halt1_lt != 0) | (halt2_lt != 0) | b_halt_lt) ? 1 : 0;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// maze: maze map terrain
//
////////////////////////////////////////////////////////////////////////////////

module maze (clk, in_hsync, in_vsync, in_blank, hcount, vcount, pix_clk,
                       x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f,
                       x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b,
                       x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt,
                       out_hsync, out_vsync, out_blank, pixel, halt_f, halt_b, halt_r,
halt_lt);


   // finish box
   parameter FINISHX = 10'd349;
   parameter FINISHY = 10'd377;

   input clk;       // video clock
   input in_hsync;  // hsync signal
   input in_vsync;  // vsync signal
   input in_blank;  // blank signal
   input [9:0] hcount;     // current x,y location of pixel
   input [9:0] vcount;
   input pix_clk;   // pixel clock
   input [9:0] x1f;  // front boundary points
   input [9:0] y1f;
   input [9:0] x2f;
   input [9:0] y2f;
   input [9:0] x3f;
   input [9:0] y3f;
   input [9:0] x4f;
   input [9:0] y4f;
   input [9:0] x1b;  // back boundary points
   input [9:0] y1b;
   input [9:0] x2b;
   input [9:0] y2b;
   input [9:0] x3b;
   input [9:0] y3b;
   input [9:0] x4b;
   input [9:0] y4b;
   input [9:0] x1r;  // right boundary points
   input [9:0] y1r;
   input [9:0] x2r;
   input [9:0] y2r;
   input [9:0] x1lt; // left boundary points
   input [9:0] y1lt;
   input [9:0] x2lt;
   input [9:0] y2lt;

   output [2:0] pixel;     // pixel value output
   output halt_f;   // halt move forward?
   output halt_b;   // halt move backward?
   output halt_r;   // halt move right?
   output halt_lt;  // halt move left?
   output out_hsync;  // hsync signal
   output out_vsync;  // vsync signal
   output out_blank;  // blank signal

   reg out_hsync = 0;
   reg out_vsync = 0;
   reg out_blank = 0;
```

```
   reg wait_hsync = 0;  // latch syncs and blanks for one clk cycle: match pixel
offset
   reg wait_vsync = 0;
   reg wait_blank = 0;

   // CALCULATE ROM ADDRESS OF EACH PIXEL/POINT
   // the memory address is x/2 + y/2 * 320
   // (4 pixels per memory location, since image is 320x240, and display is 640x480).

   reg [16:0]       pixaddr;
   reg [16:0]       f1addr;
   reg [16:0]       f2addr;
   reg [16:0]       f3addr;
   reg [16:0]       f4addr;
   reg [16:0]       b1addr;
   reg [16:0]       b2addr;
   reg [16:0]       b3addr;
   reg [16:0]       b4addr;
   reg [16:0]       r1addr;
   reg [16:0]       r2addr;
   reg [16:0]       lt1addr;
   reg [16:0]       lt2addr;

   reg [16:0]       pixaddr_reg;
   reg [16:0]       f1addr_reg;
   reg [16:0]       f2addr_reg;
   reg [16:0]       f3addr_reg;
   reg [16:0]       f4addr_reg;
   reg [16:0]       b1addr_reg;
   reg [16:0]       b2addr_reg;
   reg [16:0]       b3addr_reg;
   reg [16:0]       b4addr_reg;
   reg [16:0]       r1addr_reg;
   reg [16:0]       r2addr_reg;
   reg [16:0]       lt1addr_reg;
   reg [16:0]       lt2addr_reg;

       // y/2 * 320
   always @(posedge clk)
   begin
     pixaddr <= (hcount==0 & vcount==0) ? 0
            : (hcount==0 & pix_clk & ~vcount[0]) ? pixaddr + 320 : pixaddr;
     f1addr = y1f[9:1] * 320;
     f2addr = y2f[9:1] * 320;
     f3addr = y3f[9:1] * 320;
     f4addr = y4f[9:1] * 320;
     b1addr = y1b[9:1] * 320;
     b2addr = y2b[9:1] * 320;
     b3addr = y3b[9:1] * 320;
     b4addr = y4b[9:1] * 320;
     r1addr = y1r[9:1] * 320;
     r2addr = y2r[9:1] * 320;
     lt1addr = y1lt[9:1] * 320;
     lt2addr = y2lt[9:1] * 320;
   end

    // x/2 + (y/2*320)
   always @(posedge clk)
   begin
      pixaddr_reg <= {8'b0,hcount[9:1]} + pixaddr[16:0];
      f1addr_reg <= {8'b0,x1f[9:1]} + f1addr[16:0];
      f2addr_reg <= {8'b0,x2f[9:1]} + f2addr[16:0];
      f3addr_reg <= {8'b0,x3f[9:1]} + f3addr[16:0];
```

```
        f4addr_reg <= {8'b0,x4f[9:1]} + f4addr[16:0];
        b1addr_reg <= {8'b0,x1b[9:1]} + b1addr[16:0];
        b2addr_reg <= {8'b0,x2b[9:1]} + b2addr[16:0];
        b3addr_reg <= {8'b0,x3b[9:1]} + b3addr[16:0];
        b4addr_reg <= {8'b0,x4b[9:1]} + b4addr[16:0];
        r1addr_reg <= {8'b0,x1r[9:1]} + r1addr[16:0];
        r2addr_reg <= {8'b0,x2r[9:1]} + r2addr[16:0];
        lt1addr_reg <= {8'b0,x1lt[9:1]} + lt1addr[16:0];
        lt2addr_reg <= {8'b0,x2lt[9:1]} + lt2addr[16:0];
    end

     // latch sync and blank
    always @(posedge clk)
    begin
        wait_hsync <= in_hsync;
        wait_vsync <= in_vsync;
        wait_blank <= in_blank;
        out_hsync <= wait_hsync;
        out_vsync <= wait_vsync;
        out_blank <= wait_blank;
    end

    // instantiate the image roms
    wire rom_pixel;
    wire halt1_f;
    wire halt2_f;
    wire halt3_f;
    wire halt4_f;
    wire halt1_b;
    wire halt2_b;
    wire halt3_b;
    wire halt4_b;
    wire halt1_r;
    wire halt2_r;
    wire halt1_lt;
    wire halt2_lt;
    maze1 rom1(pixaddr_reg, clk, rom_pixel);
    maze2 rom2(f1addr_reg, f2addr_reg, clk, clk, halt1_f, halt2_f);
    maze2 rom3(f3addr_reg, f4addr_reg, clk, clk, halt3_f, halt4_f);
    maze2 rom4(b1addr_reg, b2addr_reg, clk, clk, halt1_b, halt2_b);
    maze2 rom5(b3addr_reg, b4addr_reg, clk, clk, halt3_b, halt4_b);
    maze2 rom6(r1addr_reg, r2addr_reg, clk, clk, halt1_r, halt2_r);
    maze2 rom7(lt1addr_reg, lt2addr_reg, clk, clk, halt1_lt, halt2_lt);

    // finish box
    wire [2:0] finish_pixel;
    blob finish(FINISHX, FINISHY, hcount, vcount, finish_pixel);
    defparam finish.COLOR = 3'b011;
    defparam finish.WIDTH = 55;
    defparam finish.HEIGHT = 43;

    // pixel outputs
    assign pixel = (rom_pixel == 1) ? (3'b111 | finish_pixel) : (3'b000 |
finish_pixel);
    // check boundary points: if not black (1'b0) pixel, HALT!
    assign halt_f = ((halt1_f != 0) | (halt2_f != 0) | (halt3_f != 0) | (halt4_f != 0))
? 1 : 0;
    assign halt_b = ((halt1_b != 0) | (halt2_b != 0) | (halt3_b != 0) | (halt4_b != 0))
? 1 : 0;
    assign halt_r = ((halt1_r != 0) | (halt2_r != 0)) ? 1 : 0;
    assign halt_lt = ((halt1_lt != 0) | (halt2_lt != 0)) ? 1 : 0;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// racetrack: racetrack map terrain
//
////////////////////////////////////////////////////////////////////////////////

module racetrack (clk, in_hsync, in_vsync, in_blank, hcount, vcount, pix_clk,
                  x1f, y1f, x2f, y2f, x3f, y3f, x4f, y4f,
                  x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b,
                  x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt, y2lt,
                  out_hsync, out_vsync, out_blank, pixel, halt_f, halt_b, halt_r,
halt_lt);

   input clk;       // video clock
   input in_hsync;  // hsync signal
   input in_vsync;  // vsync signal
   input in_blank;  // blank signal
   input [9:0] hcount;     // current x,y location of pixel
   input [9:0] vcount;
   input pix_clk;   // pixel clock

   input [9:0] x1f;  // front boundary points
   input [9:0] y1f;
   input [9:0] x2f;
   input [9:0] y2f;
   input [9:0] x3f;
   input [9:0] y3f;
   input [9:0] x4f;
   input [9:0] y4f;
   input [9:0] x1b;  // back boundary points
   input [9:0] y1b;
   input [9:0] x2b;
   input [9:0] y2b;
   input [9:0] x3b;
   input [9:0] y3b;
   input [9:0] x4b;
   input [9:0] y4b;
   input [9:0] x1r;  // right boundary points
   input [9:0] y1r;
   input [9:0] x2r;
   input [9:0] y2r;
   input [9:0] x1lt; // left boundary points
   input [9:0] y1lt;
   input [9:0] x2lt;
   input [9:0] y2lt;

   output [2:0] pixel;     // pixel value output
   output halt_f;   // halt move forward?
   output halt_b;   // halt move backward?
   output halt_r;   // halt move right?
   output halt_lt;  // halt move left?
   output out_hsync;  // hsync signal
   output out_vsync;  // vsync signal
   output out_blank;  // blank signal

   reg out_hsync = 0;
   reg out_vsync = 0;
   reg out_blank = 0;
   reg wait_hsync = 0;  // latch syncs and blanks for one clk cycle: match pixel
offset
   reg wait_vsync = 0;
   reg wait_blank = 0;
```

```
// CALCULATE ROM ADDRESS OF EACH PIXEL/POINT
// the memory address is x/2 + y/2 * 320
// (4 pixels per memory location, since image is 320x240, and
// display is 640x480).

reg [16:0]       pixaddr;
reg [16:0]       f1addr;
reg [16:0]       f2addr;
reg [16:0]       f3addr;
reg [16:0]       f4addr;
reg [16:0]       b1addr;
reg [16:0]       b2addr;
reg [16:0]       b3addr;
reg [16:0]       b4addr;
reg [16:0]       r1addr;
reg [16:0]       r2addr;
reg [16:0]       lt1addr;
reg [16:0]       lt2addr;

reg [16:0]       pixaddr_reg;
reg [16:0]       f1addr_reg;
reg [16:0]       f2addr_reg;
reg [16:0]       f3addr_reg;
reg [16:0]       f4addr_reg;
reg [16:0]       b1addr_reg;
reg [16:0]       b2addr_reg;
reg [16:0]       b3addr_reg;
reg [16:0]       b4addr_reg;
reg [16:0]       r1addr_reg;
reg [16:0]       r2addr_reg;
reg [16:0]       lt1addr_reg;
reg [16:0]       lt2addr_reg;

    // y/2 * 320
always @(posedge clk)
begin
  pixaddr <= (hcount==0 & vcount==0) ? 0
          : (hcount==0 & pix_clk & ~vcount[0]) ? pixaddr + 320 : pixaddr;
  f1addr = y1f[9:1] * 320;
  f2addr = y2f[9:1] * 320;
  f3addr = y3f[9:1] * 320;
  f4addr = y4f[9:1] * 320;
  b1addr = y1b[9:1] * 320;
  b2addr = y2b[9:1] * 320;
  b3addr = y3b[9:1] * 320;
  b4addr = y4b[9:1] * 320;
  r1addr = y1r[9:1] * 320;
  r2addr = y2r[9:1] * 320;
  lt1addr = y1lt[9:1] * 320;
  lt2addr = y2lt[9:1] * 320;
end

    // x/2 + (y/2 * 320)
always @(posedge clk)
begin
    pixaddr_reg <= {8'b0,hcount[9:1]} + pixaddr[16:0];
    f1addr_reg <= {8'b0,x1f[9:1]} + f1addr[16:0];
    f2addr_reg <= {8'b0,x2f[9:1]} + f2addr[16:0];
    f3addr_reg <= {8'b0,x3f[9:1]} + f3addr[16:0];
    f4addr_reg <= {8'b0,x4f[9:1]} + f4addr[16:0];
    b1addr_reg <= {8'b0,x1b[9:1]} + b1addr[16:0];
    b2addr_reg <= {8'b0,x2b[9:1]} + b2addr[16:0];
    b3addr_reg <= {8'b0,x3b[9:1]} + b3addr[16:0];
```

```
      b4addr_reg <= {8'b0,x4b[9:1]} + b4addr[16:0];
      r1addr_reg <= {8'b0,x1r[9:1]} + r1addr[16:0];
      r2addr_reg <= {8'b0,x2r[9:1]} + r2addr[16:0];
      lt1addr_reg <= {8'b0,x1lt[9:1]} + lt1addr[16:0];
      lt2addr_reg <= {8'b0,x2lt[9:1]} + lt2addr[16:0];
   end

    // latch sync and blank
   always @(posedge clk)
   begin
      wait_hsync <= in_hsync;
      wait_vsync <= in_vsync;
      wait_blank <= in_blank;
      out_hsync <= wait_hsync;
      out_vsync <= wait_vsync;
      out_blank <= wait_blank;
   end


   // instantiate the image roms
   wire [1:0] rom_pixel;
   wire [1:0] halt1_f;
   wire [1:0] halt2_f;
   wire [1:0] halt3_f;
   wire [1:0] halt4_f;
   wire [1:0] halt1_b;
   wire [1:0] halt2_b;
   wire [1:0] halt3_b;
   wire [1:0] halt4_b;
   wire [1:0] halt1_r;
   wire [1:0] halt2_r;
   wire [1:0] halt1_lt;
   wire [1:0] halt2_lt;
   racetrack1 rom1(pixaddr_reg, clk, rom_pixel);
   racetrack2 rom2(f1addr_reg, f2addr_reg, clk, clk, halt1_f, halt2_f);
   racetrack2 rom3(f3addr_reg, f4addr_reg, clk, clk, halt3_f, halt4_f);
   racetrack2 rom4(b1addr_reg, b2addr_reg, clk, clk, halt1_b, halt2_b);
   racetrack2 rom5(b3addr_reg, b4addr_reg, clk, clk, halt3_b, halt4_b);
   racetrack2 rom6(r1addr_reg, r2addr_reg, clk, clk, halt1_r, halt2_r);
   racetrack2 rom7(lt1addr_reg, lt2addr_reg, clk, clk, halt1_lt, halt2_lt);

   // pixel outputs
   assign pixel = (rom_pixel == 2'b11) ? 3'b111 : {1'b0,rom_pixel};
   // check boundary points: if not black (2'b00) pixel, HALT!
   assign halt_f = ((halt1_f != 2'b00) | (halt2_f != 2'b00) | (halt3_f != 2'b00) |
(halt4_f != 2'b00)) ? 1 : 0;
   assign halt_b = ((halt1_b != 2'b00) | (halt2_b != 2'b00) | (halt3_b != 2'b00) |
(halt4_b != 2'b00)) ? 1 : 0;
   assign halt_r = ((halt1_r != 2'b00) | (halt2_r != 2'b00)) ? 1 : 0;
   assign halt_lt = ((halt1_lt != 2'b00) | (halt2_lt != 2'b00)) ? 1 : 0;

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// horz_line: horizontal line sprites and boundary checking
//
///////////////////////////////////////////////////////////////////////////////

module horz_line(x1, x2, x3, y, hcount, vcount, x1f, y1f, x2f, y2f, x3f, y3f, x4f,
y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt,
y2lt, pix, halt_f, halt_b, halt_r, halt_lt);

	parameter WIDTH1 = 0; // default segment 1 width: 0 pixels
	parameter WIDTH2 = 0; // default segment 1 width: 0 pixels
	parameter WIDTH3 = 0; // default segment 1 width: 0 pixels
	parameter HEIGHT = 5; // default height: 5 pixels
	parameter COLOR = 3'b111; // default color: white

	input [9:0] x1;      // segment 1 start
	input [9:0] x2;      // segment 2 start
	input [9:0] x3;      // segment 3 start
	input [9:0] y;
	input [9:0] hcount;
	input [9:0] vcount;

      input [9:0] x1f;      // front boundary points
      input [9:0] y1f;
      input [9:0] x2f;
      input [9:0] y2f;
      input [9:0] x3f;
      input [9:0] y3f;
      input [9:0] x4f;
      input [9:0] y4f;
      input [9:0] x1b;      // back boundary points
      input [9:0] y1b;
      input [9:0] x2b;
      input [9:0] y2b;
      input [9:0] x3b;
      input [9:0] y3b;
      input [9:0] x4b;
      input [9:0] y4b;
      input [9:0] x1r;      // right boundary points
      input [9:0] y1r;
      input [9:0] x2r;
      input [9:0] y2r;
      input [9:0] x1lt;     // left boundary points
      input [9:0] y1lt;
      input [9:0] x2lt;
      input [9:0] y2lt;

      output [2:0] pix;     // mine pixel
      output halt_f; // halt move forward?
      output halt_b; // halt move backward?
      output halt_r; // halt move right?
      output halt_lt;       // halt move left?

	reg [2:0] pix = 0;
	reg halt1_f = 0;
	reg halt2_f = 0;
	reg halt3_f = 0;
	reg halt4_f = 0;
	reg halt1_b = 0;
	reg halt2_b = 0;
	reg halt3_b = 0;
	reg halt4_b = 0;
```

```
	reg halt1_r = 0;
	reg halt2_r = 0;
	reg halt1_lt = 0;
	reg halt2_lt = 0;

	always @ (x1 or x2 or x3 or y or x1f or y1f or x2f or y2f or x3f or y3f or x4f
or y4f or x1b or y1b or x2b or y2b or x3b or y3b or x4b or y4b or x1r or y1r or x2r or
y2r or x1lt or y1lt or x2lt or y2lt or hcount or vcount)
		begin

		// mine pixel display

			if (((hcount >= x1 && hcount < (x1+WIDTH1)) | (hcount >= x2 &&
hcount < (x2+WIDTH2)) | (hcount >= x3 && hcount < (x3+WIDTH3))) && (vcount >= y &&
vcount < (y+HEIGHT)))
				pix = COLOR;
			else
				pix = 0;

		// check control forward

			if (((x1f >= x1 && x1f < (x1+WIDTH1)) | (x1f >= x2 && x1f <
(x2+WIDTH2)) | (x1f >= x3 && x1f < (x3+WIDTH3))) && (y1f >= y && y1f < (y+HEIGHT)))
				halt1_f = 1;
			else
				halt1_f = 0;
			if (((x2f >= x1 && x2f < (x1+WIDTH1)) | (x2f >= x2 && x2f <
(x2+WIDTH2)) | (x2f >= x3 && x2f < (x3+WIDTH3))) && (y2f >= y && y2f < (y+HEIGHT)))
				halt2_f = 1;
			else
				halt2_f = 0;
			if (((x3f >= x1 && x3f < (x1+WIDTH1)) | (x3f >= x2 && x3f <
(x2+WIDTH2)) | (x3f >= x3 && x3f < (x3+WIDTH3))) && (y3f >= y && y3f < (y+HEIGHT)))
				halt3_f = 1;
			else
				halt3_f = 0;
			if (((x4f >= x1 && x4f < (x1+WIDTH1)) | (x4f >= x2 && x4f <
(x2+WIDTH2)) | (x4f >= x3 && x4f < (x3+WIDTH3))) && (y4f >= y && y4f < (y+HEIGHT)))
				halt4_f = 1;
			else
				halt4_f = 0;

		// check  control back

			if (((x1b >= x1 && x1b < (x1+WIDTH1)) | (x1b >= x2 && x1b <
(x2+WIDTH2)) | (x1b >= x3 && x1b < (x3+WIDTH3))) && (y1b >= y && y1b < (y+HEIGHT)))
				halt1_b = 1;
			else
				halt1_b = 0;
			if (((x2b >= x1 && x2b < (x1+WIDTH1)) | (x2b >= x2 && x2b <
(x2+WIDTH2)) | (x2b >= x3 && x2b < (x3+WIDTH3))) && (y2b >= y && y2b < (y+HEIGHT)))
				halt2_b = 1;
			else
				halt2_b = 0;
			if (((x3b >= x1 && x3b < (x1+WIDTH1)) | (x3b >= x2 && x3b <
(x2+WIDTH2)) | (x3b >= x3 && x3b < (x3+WIDTH3))) && (y3b >= y && y3b < (y+HEIGHT)))
				halt3_b = 1;
			else
				halt3_b = 0;
			if (((x4b >= x1 && x4b < (x1+WIDTH1)) | (x4b >= x2 && x4b <
(x2+WIDTH2)) | (x4b >= x3 && x4b < (x3+WIDTH3))) && (y4b >= y && y4b < (y+HEIGHT)))
				halt4_b = 1;
			else
```

```
                        halt4_b = 0;

            // check  control right

                  if (((x1r >= x1 && x1r < (x1+WIDTH1)) | (x1r >= x2 && x1r <
(x2+WIDTH2)) | (x1r >= x3 && x1r < (x3+WIDTH3))) && (y1r >= y && y1r < (y+HEIGHT)))
                        halt1_r = 1;
                  else
                        halt1_r = 0;
                  if (((x2r >= x1 && x2r < (x1+WIDTH1)) | (x2r >= x2 && x2r <
(x2+WIDTH2)) | (x2r >= x3 && x2r < (x3+WIDTH3))) && (y2r >= y && y2r < (y+HEIGHT)))
                        halt2_r = 1;
                  else
                        halt2_r = 0;

            // check  control left

                  if (((x1lt >= x1 && x1lt < (x1+WIDTH1)) | (x1lt >= x2 && x1lt <
(x2+WIDTH2)) | (x1lt >= x3 && x1lt < (x3+WIDTH3))) && (y1lt >= y && y1lt <
(y+HEIGHT)))
                        halt1_lt = 1;
                  else
                        halt1_lt = 0;
                  if (((x2lt >= x1 && x2lt < (x1+WIDTH1)) | (x2lt >= x2 && x2lt <
(x2+WIDTH2)) | (x2lt >= x3 && x2lt < (x3+WIDTH3))) && (y2lt >= y && y2lt <
(y+HEIGHT)))
                        halt2_lt = 1;
                  else
                        halt2_lt = 0;

            end

      assign halt_f = halt1_f | halt2_f | halt3_f | halt4_f;
      assign halt_b = halt1_b | halt2_b | halt3_b | halt4_b;
      assign halt_r = halt1_r | halt2_r;
      assign halt_lt = halt1_lt | halt2_lt;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// vert_line: vertical line sprites and boundary checking
//
////////////////////////////////////////////////////////////////////////////////

module vert_line(x, y1, y2, y3, hcount, vcount, x1f, y1f, x2f, y2f, x3f, y3f, x4f,
y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt, y1lt, x2lt,
y2lt, pix, halt_f, halt_b, halt_r, halt_lt);

	parameter WIDTH = 5; // default width: 5pixels
	parameter HEIGHT1 = 0; // default segment 1 height: 0 pixels
	parameter HEIGHT2 = 0; // default segment 2 height: 0 pixels
	parameter HEIGHT3 = 0; // default segment 3 height: 0 pixels
	parameter COLOR = 3'b111; // default color: white

	input [9:0] x;
	input [9:0] y1;     // segment 1 start
	input [9:0] y2;     // segment 2 start
	input [9:0] y3;     // segment 3 start
	input [9:0] hcount;
	input [9:0] vcount;

      input [9:0] x1f;            // front boundary points
      input [9:0] y1f;
      input [9:0] x2f;
      input [9:0] y2f;
      input [9:0] x3f;
      input [9:0] y3f;
      input [9:0] x4f;
      input [9:0] y4f;
      input [9:0] x1b;            // back boundary points
      input [9:0] y1b;
      input [9:0] x2b;
      input [9:0] y2b;
      input [9:0] x3b;
      input [9:0] y3b;
      input [9:0] x4b;
      input [9:0] y4b;
      input [9:0] x1r;            // right boundary points
      input [9:0] y1r;
      input [9:0] x2r;
      input [9:0] y2r;
      input [9:0] x1lt;           // left boundary points
      input [9:0] y1lt;
      input [9:0] x2lt;
      input [9:0] y2lt;

      output [2:0] pix;     // mine pixel
      output halt_f; // halt move forward?
      output halt_b; // halt move backward?
      output halt_r; // halt move right?
      output halt_lt;       // halt move left?

	reg [2:0] pix = 0;
	reg halt1_f = 0;
	reg halt2_f = 0;
	reg halt3_f = 0;
	reg halt4_f = 0;
	reg halt1_b = 0;
	reg halt2_b = 0;
	reg halt3_b = 0;
	reg halt4_b = 0;
```

```
	reg halt1_r = 0;
	reg halt2_r = 0;
	reg halt1_lt = 0;
	reg halt2_lt = 0;

	always @ (x or y1 or y2 or y3 or x1f or y1f or x2f or y2f or x3f or y3f or x4f
or y4f or x1b or y1b or x2b or y2b or x3b or y3b or x4b or y4b or x1r or y1r or x2r or
y2r or x1lt or y1lt or x2lt or y2lt or hcount or vcount)
		begin

		// mine pixel display

			if ((hcount >= x && hcount < (x+WIDTH)) && ((vcount >= y1 &&
vcount < (y1+HEIGHT1)) | (vcount >= y2 && vcount < (y2+HEIGHT2)) | (vcount >= y3 &&
vcount < (y3+HEIGHT3))))
				pix = COLOR;
			else
				pix = 0;

		// check control forward

			if ((x1f >= x && x1f < (x+WIDTH)) && ((y1f >= y1 && y1f <
(y1+HEIGHT1)) | (y1f >= y2 && y1f < (y2+HEIGHT2)) | (y1f >= y3 && y1f <
(y3+HEIGHT3))))
				halt1_f = 1;
			else
				halt1_f = 0;
			if ((x2f >= x && x2f < (x+WIDTH)) && ((y2f >= y1 && y2f <
(y1+HEIGHT1)) | (y2f >= y2 && y2f < (y2+HEIGHT2)) | (y2f >= y3 && y2f <
(y3+HEIGHT3))))
				halt2_f = 1;
			else
				halt2_f = 0;
			if ((x3f >= x && x3f < (x+WIDTH)) && ((y3f >= y1 && y3f <
(y1+HEIGHT1)) | (y3f >= y2 && y3f < (y2+HEIGHT2)) | (y3f >= y3 && y3f <
(y3+HEIGHT3))))
				halt3_f = 1;
			else
				halt3_f = 0;
			if ((x4f >= x && x4f < (x+WIDTH)) && ((y4f >= y1 && y4f <
(y1+HEIGHT1)) | (y4f >= y2 && y4f < (y2+HEIGHT2)) | (y4f >= y3 && y4f <
(y3+HEIGHT3))))
				halt4_f = 1;
			else
				halt4_f = 0;

		// check  control back

			if ((x1b >= x && x1b < (x+WIDTH)) && ((y1b >= y1 && y1b <
(y1+HEIGHT1)) | (y1b >= y2 && y1b < (y2+HEIGHT2)) | (y1b >= y3 && y1b <
(y3+HEIGHT3))))
				halt1_b = 1;
			else
				halt1_b = 0;
			if ((x2b >= x && x2b < (x+WIDTH)) && ((y2b >= y1 && y2b <
(y1+HEIGHT1)) | (y2b >= y2 && y2b < (y2+HEIGHT2)) | (y2b >= y3 && y2b <
(y3+HEIGHT3))))
				halt2_b = 1;
			else
				halt2_b = 0;
			if ((x3b >= x && x3b < (x+WIDTH)) && ((y3b >= y1 && y3b <
(y1+HEIGHT1)) | (y3b >= y2 && y3b < (y2+HEIGHT2)) | (y3b >= y3 && y3b <
(y3+HEIGHT3))))
```

```
                        halt3_b = 1;
                else
                        halt3_b = 0;
                if ((x4b >= x && x4b < (x+WIDTH)) && ((y4b >= y1 && y4b <
(y1+HEIGHT1)) | (y4b >= y2 && y4b < (y2+HEIGHT2)) | (y4b >= y3 && y4b <
(y3+HEIGHT3))))
                        halt4_b = 1;
                else
                        halt4_b = 0;

        // check  control right

                if ((x1r >= x && x1r < (x+WIDTH)) && ((y1r >= y1 && y1r <
(y1+HEIGHT1)) | (y1r >= y2 && y1r < (y2+HEIGHT2)) | (y1r >= y3 && y1r <
(y3+HEIGHT3))))
                        halt1_r = 1;
                else
                        halt1_r = 0;
                if ((x2r >= x && x2r < (x+WIDTH)) && ((y2r >= y1 && y2r <
(y1+HEIGHT1)) | (y2r >= y2 && y2r < (y2+HEIGHT2)) | (y2r >= y3 && y2r <
(y3+HEIGHT3))))
                        halt2_r = 1;
                else
                        halt2_r = 0;

        // check  control left

                if ((x1lt >= x && x1lt < (x+WIDTH)) && ((y1lt >= y1 && y1lt <
(y1+HEIGHT1)) | (y1lt >= y2 && y1lt < (y2+HEIGHT2)) | (y1lt >= y3 && y1lt <
(y3+HEIGHT3))))
                        halt1_lt = 1;
                else
                        halt1_lt = 0;
                if ((x2lt >= x && x2lt < (x+WIDTH)) && ((y2lt >= y1 && y2lt <
(y1+HEIGHT1)) | (y2lt >= y2 && y2lt < (y2+HEIGHT2)) | (y2lt >= y3 && y2lt <
(y3+HEIGHT3))))
                        halt2_lt = 1;
                else
                        halt2_lt = 0;

        end

    assign halt_f = halt1_f | halt2_f | halt3_f | halt4_f;
    assign halt_b = halt1_b | halt2_b | halt3_b | halt4_b;
    assign halt_r = halt1_r | halt2_r;
    assign halt_lt = halt1_lt | halt2_lt;

endmodule
```

```
//
// File:   vga_sync.v
// Date:   04-Nov-05
// Author: C. Terman / I. Chuang
//
// MIT 6.111 Fall 2005
//
// Verilog code to produce VGA sync signals (and blanking) for 640x480 screen
//

module vga_sync(clk,hsync,vsync,hcount,vcount,pix_clk,blank);

   input clk;     // 50Mhz
   output hsync;
   output vsync;
   output [9:0] hcount, vcount;
   output    pix_clk;
   output    blank;

   // pixel clock: 25Mhz = 40ns (clk/2)
   reg             pcount;             // used to generate pixel clock
   wire      en = (pcount == 0);
   always @ (posedge clk) pcount <= ~pcount;
   wire      pix_clk = ~en;

   //****************************************************************
   //***  Sync and Blanking Signals
   //****************************************************************

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

   // horizontal: 794 pixels = 31.76us
   // display 640 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = en & (hcount == 639);
   assign    hsyncon = en & (hcount == 652);
   assign    hsyncoff = en & (hcount == 746);
   assign    hreset = en & (hcount == 793);

   wire      blank =  (vblank | (hblank & ~hreset));    // blanking => black

   // vertical: 528 lines = 16.77us
   // display 480 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 479);
   assign    vsyncon = hreset & (vcount == 492);
   assign    vsyncoff = hreset & (vcount == 494);
   assign    vreset = hreset & (vcount == 527);

   // sync and blanking
   always @(posedge clk) begin
      hcount <= en ? (hreset ? 0 : hcount + 1) : hcount;
      hblank <= hreset ? 0 : hblankon ? 1 : hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;   // hsync is active low

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

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// point: draw filled rectangle
//
////////////////////////////////////////////////////////////////////////////////


module point(x,y,hcount,vcount,pix);

	parameter WIDTH = 2; // default width: 2 pixels
	parameter HEIGHT = 2; // default height: 2 pixels
	parameter COLOR = 3'b111; // default color: white

	input [9:0] x,y,hcount;
	input [9:0] vcount;
	output [2:0] pix;

	reg [2:0] pix;

	always @ (x or y or hcount or vcount)
		begin
			if ((hcount >= x && hcount < (x+WIDTH)) &&
			(vcount >= y && vcount < (y+HEIGHT)))
			pix = COLOR;
			else pix = 0;
		end
endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// hexdisplay: produces hex display signals and outputs
//
///////////////////////////////////////////////////////////////////////////////

module hexdisplay(clk, clk_reset, hex_reset, map_param,
                        car_1x, car_1y, car_2x, car_2y, car_3x, car_3y, car_4x, car_4y,
timer_on, halt,
                        disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b,
disp_data_out);

   input clk;
   input clk_reset;
   input hex_reset;
   input [1:0] map_param;
   input [9:0] car_1x;
   input [9:0] car_1y;
   input [9:0] car_2x;
   input [9:0] car_2y;
   input [9:0] car_3x;
   input [9:0] car_3y;
   input [9:0] car_4x;
   input [9:0] car_4y;
   input timer_on;
   input halt;

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

   reg [7:0] datain;

   // TIMER: maze & racetrack
          // timer off if car in start box
          wire instart;
          inbox check_start(car_1x, car_1y, car_2x, car_2y, car_3x, car_3y, car_4x,
car_4y, instart);
          defparam check_start.START_UPX= 349;
          defparam check_start.START_LOWX = 290;
          defparam check_start.START_UPY = 266;
          defparam check_start.START_LOWY = 213;
          // timer off if car in finish box
          wire infinish;
          inbox check_finish(car_1x, car_1y, car_2x, car_2y, car_3x, car_3y, car_4x,
car_4y, infinish);
          defparam check_finish.START_UPX      = 402;
          defparam check_finish.START_LOWX = 349;
          defparam check_finish.START_UPY = 420;
          defparam check_finish.START_LOWY = 377;

          wire in_box = instart | infinish;

          // divide system clock into seconds clock
          wire seconds_clk;
          divider second(clk, clk_reset, seconds_clk);

          // clock incrementer
          wire [3:0]minute2;
          wire [3:0]minute1;
```

```
          wire [3:0]second2;
          wire [3:0]second1;
          clock clock1(clk, seconds_clk, clk_reset, timer_on, in_box, minute2,
minute1, second2, second1);


   // HEX DISPLAY OUTPUT

   wire [3:0] hex_char;
   always @ (hex_char or map_param or halt or minute1 or minute2 or second1 or
second2)
   begin
       case (map_param)
             2'b00: // open: "SELECT A MAP"
             begin
                    case (hex_char)
                     4'hF: datain = 8'h20;      // BLANK
                     4'hE: datain = 8'h20;      // BLANK
                     4'hD: datain = 8'h20;      // BLANK
                     4'hC: datain = 8'h20;      // BLANK
                     4'hB: datain = 8'h53;      // S
                     4'hA: datain = 8'h45;      // E
                     4'h9: datain = 8'h4C;      // L
                     4'h8: datain = 8'h45;      // E
                     4'h7: datain = 8'h43;      // C
                     4'h6: datain = 8'h54;      // T
                     4'h5: datain = 8'h20;      // BLANK
                     4'h4: datain = 8'h41;      // A
                     4'h3: datain = 8'h20;      // BLANK
                     4'h2: datain = 8'h4D;      // M
                     4'h1: datain = 8'h41;      // A
                     4'h0: datain = 8'h50;      // P
                  endcase
             end

             2'b01: // minefield: "BOOM!"
                         // if halt, display text; else, blank
             begin
                    if (halt)
                           case (hex_char)
                            4'hF: datain = 8'h20;      // BLANK
                            4'hE: datain = 8'h20;      // BLANK
                            4'hD: datain = 8'h20;      // BLANK
                            4'hC: datain = 8'h20;      // BLANK
                            4'hB: datain = 8'h20;      // BLANK
                            4'hA: datain = 8'h20;      // BLANK
                            4'h9: datain = 8'h20;      // BLANK
                            4'h8: datain = 8'h20;      // BLANK
                            4'h7: datain = 8'h20;      // BLANK
                            4'h6: datain = 8'h20;      // BLANK
                            4'h5: datain = 8'h20;      // BLANK
                            4'h4: datain = 8'h42;      // B
                            4'h3: datain = 8'h4F;      // O
                            4'h2: datain = 8'h4F;      // O
                            4'h1: datain = 8'h4D;      // M
                            4'h0: datain = 8'h21;      // !
                         endcase
                    else
                           datain = 8'h20;     // BLANK
             end

             2'b10: // maze: "TIMER --:--"
             begin
```

```
            case (hex_char)
             4'hF: datain = 8'h20;                    // BLANK
             4'hE: datain = 8'h20;                    // BLANK
             4'hD: datain = 8'h20;                    // BLANK
             4'hC: datain = 8'h20;                    // BLANK
             4'hB: datain = 8'h20;                    // BLANK
             4'hA: datain = 8'h20;                    // BLANK
             4'h9: datain = 8'h54;                    // T
             4'h8: datain = 8'h49;                    // I
             4'h7: datain = 8'h4D;                    // M
             4'h6: datain = 8'h45;                    // E
             4'h5: datain = 8'h20;                    // BLANK
             4'h4: datain = {4'h3,minute2};   // minute2
             4'h3: datain = {4'h3,minute1};   // minute1
             4'h2: datain = 8'h3A;     // :
             4'h1: datain = {4'h3,second2};   // second2
             4'h0: datain = {4'h3,second1};   // second1
           endcase
        end

        2'b11: // racetrack: "TIMER --:--"
        begin
            case (hex_char)
             4'hF: datain = 8'h20;                    // BLANK
             4'hE: datain = 8'h20;                    // BLANK
             4'hD: datain = 8'h20;                    // BLANK
             4'hC: datain = 8'h20;                    // BLANK
             4'hB: datain = 8'h20;                    // BLANK
             4'hA: datain = 8'h20;                    // BLANK
             4'h9: datain = 8'h54;                    // T
             4'h8: datain = 8'h49;                    // I
             4'h7: datain = 8'h4D;                    // M
             4'h6: datain = 8'h45;                    // E
             4'h5: datain = 8'h20;                    // BLANK
             4'h4: datain = {4'h3,minute2};   // minute2
             4'h3: datain = {4'h3,minute1};   // minute1
             4'h2: datain = 8'h3A;     // :
             4'h1: datain = {4'h3,second2};   // second2
             4'h0: datain = {4'h3,second1};   // second1
           endcase
        end

     endcase // map_param

  end // always

  // instantiate hex display signals
  display_string hexcode(hex_reset, clk, datain, disp_blank, disp_clock, disp_rs,
disp_ce_b, disp_reset_b, disp_data_out, hex_char);

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// divider: divides input clock by certain factor
//
////////////////////////////////////////////////////////////////////////////////

module divider(clk, reset, enable);

         parameter divideFactor = 27000000;
         input clk, reset;
         output enable;

         reg [24:0] counter = 0;

         always @ (posedge clk)
         begin
              if(reset)
                     begin
                            counter <= 0;
                     end
              else
                     begin
                            counter <= counter+1;
                     end

              if(divideFactor == (counter-1) )

                            counter <= 0;
         end

         assign enable = (divideFactor == counter) ? 1 : 0;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// clock: produces minute and second values for timer
//
////////////////////////////////////////////////////////////////////////////////

module clock(clk, count, clr, timer_on, in_box, minute2, minute1, second2, second1);
	input clk;
	input count;
	input clr;
	input timer_on;
	input in_box;

	output [3:0] second1;
	output [3:0] second2;
	output [3:0] minute1;
	output [3:0] minute2;

	reg [3:0]second1 = 4'b0000;
	reg [3:0]second2 = 4'b0000;
	reg [3:0]minute1 = 4'b0000;
	reg [3:0]minute2 = 4'b0000;

	always @ (posedge clk)
	begin
		if(clr | ~timer_on)
			begin
				second1 <= 0;
				second2 <= 0;
				minute1 <= 0;
				minute2 <= 0;
			end
		else if (count && timer_on && ~in_box)
			begin
				if(minute2 == 5 && minute1 == 9 && second2 == 5 && second1 
== 9) // 59:59
				begin
					minute2 <= 0;
					minute1 <= 0;
					second2 <= 0;
					second1 <= 0;
				end

				else if(minute1 == 9 && second2 == 5 && second1 == 9) // -
9:59
				begin
					minute2 <= minute2 + 1;
					minute1 <= 0;
					second2 <= 0;
					second1 <= 0;
				end

				else if(second2 == 5 && second1 == 9) // --:59
				begin
					minute1 <= minute1 + 1;
					second2 <= 0;
					second1 <= 0;
				end

				else if(second1 == 9) // --:-9
				begin
					second2 <= second2 + 1;
					second1 <= 0;
```

```
                    end

                    else
                        second1 <= second1 + 1; // increment seconds
                end

        end

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- 16 characer ASCII string display
//
//
// File:   display_string.v
// Date:   24-Sep-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Based on Nathan Ickes' hex display code
//
// This module drives the labkit hex displays and shows the value of
// 8 ascii bytes as characters on the displays.
//
// Uses the Jae's ascii2dots module
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   string_data - 128 bits; each 8 bits gives an ASCII coded character
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 & 004)
//
////////////////////////////////////////////////////////////////////////////////

module display_string (reset, clock_27mhz, string_data,
             disp_blank, disp_clock, disp_rs, disp_ce_b,
             disp_reset_b, disp_data_out, char_index);

   input reset, clock_27mhz;     // clock and reset (active high reset)
   input [7:0] string_data;             // ascii byte to display

   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
         disp_reset_b;
   output [3:0] char_index;             // index of character to display

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

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

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

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

```
        clock = ~clock;
        count = 5'h00;
      end
    else
      count = count+1;
  end

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

assign dreset = (reset_count != 0);

assign disp_clock = ~clock;

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

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

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

always @(posedge clock)
  if (dreset)
    begin
      state <= 0;
      dot_index <= 0;
      control <= 32'h7F7F7F7F;
    end
  else
    casex (state)
     8'h00:
       begin
          // Reset displays
          disp_data_out <= 1'b0;
          disp_rs <= 1'b0; // dot register
          disp_ce_b <= 1'b1;
          disp_reset_b <= 1'b0;
          dot_index <= 0;
          state <= state+1;
       end

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

     8'h02:
       begin
          // Initialize dot register (set all dots to zero)
```

```
          disp_ce_b <= 1'b0;
          disp_data_out <= 1'b0; // dot_index[0];
          if (dot_index == 639)
           state <= state+1;
          else
           dot_index <= dot_index+1;
       end

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

     8'h04:
       begin
          // Setup the control register
          disp_rs <= 1'b1; // Select the control register
          disp_ce_b <= 1'b0;
          disp_data_out <= control[31];
          control <= {control[30:0], 1'b0}; // shift left
          if (dot_index == 0)
           state <= state+1;
          else
           dot_index <= dot_index-1;
          char_index <= 15;           // set this up early for pipeline
       end

     8'h05:
       begin
          // Latch the control register data / dot data
          disp_ce_b <= 1'b1;
          dot_index <= 39;            // init for single char
          rdots <= dots;              // store dots of char 15
          char_index <= 14;           // ready for next char
          state <= state+1;
       end

     8'h06:
       begin
          // Load the user's dot data into the dot reg, char by char
          disp_rs <= 1'b0;                  // Select the dot register
          disp_ce_b <= 1'b0;
          disp_data_out <= rdots[dot_index]; // dot data from msb
          if (dot_index == 0)
            if (char_index == 15)
              state <= 5;                   // all done, latch data
           else
           begin
             char_index <= char_index - 1;  // goto next char
             dot_index <= 39;
             rdots <= dots;           // latch in next char dots
           end
          else
           dot_index <= dot_index-1;  // else loop thru all dots
       end

    endcase

ascii2dots a2d(string_data,dots);
```

```
endmodule

///////////////////////////////////////////////////////////////////////////////
// Display font dots generation from ASCII code

module ascii2dots(ascii_in,char_dots);

input [7:0] ascii_in;
output [39:0] char_dots;

  ///////////////////////////////////////////////////////////////////////////
  // ROM: ASCII-->DOTS conversion
  ///////////////////////////////////////////////////////////////////////////
	reg [39:0] char_dots;

	always @(ascii_in)
	 case(ascii_in)
		8'h20: char_dots = 40'b00000000_00000000_00000000_00000000_00000000; //
32	' '
		8'h21: char_dots = 40'b00000000_00000000_00101111_00000000_00000000; //
33	 !
		8'h22: char_dots = 40'b00000000_00000111_00000000_00000111_00000000; //
34	"
		8'h23: char_dots = 40'b00010100_00111110_00010100_00111110_00010100; //
35	 #
		8'h24: char_dots = 40'b00000100_00101010_00111110_00101010_00010000; //
36	 $
		8'h25: char_dots = 40'b00010011_00001000_00000100_00110010_00000000; //
37	 %
		8'h26: char_dots = 40'b00010100_00101010_00010100_00100000_00000000; //
38	 &
		8'h27: char_dots = 40'b00000000_00000000_00000111_00000000_00000000; //
39	'
		8'h28: char_dots = 40'b00000000_00011110_00100001_00000000_00000000;//
40	 (
		8'h29: char_dots = 40'b00000000_00100001_00011110_00000000_00000000; //
41	 )
		8'h2A: char_dots = 40'b00000000_00101010_00011100_00101010_00000000; //
42	 *
		8'h2B: char_dots = 40'b00001000_00001000_00111110_00001000_00001000; //
43	  +
		8'h2C: char_dots = 40'b00000000_01000000_00110000_00010000_00000000; //
44	,
		8'h2D: char_dots = 40'b00001000_00001000_00001000_00001000_00000000; //
45	 -
		8'h2E: char_dots = 40'b00000000_00110000_00110000_00000000_00000000; //
46	 .
		8'h2F: char_dots = 40'b00010000_00001000_00000100_00000010_00000000; //
47	 /
		8'h30: char_dots = 40'b00000000_00011110_00100001_00011110_00000000; //
48	 0	--> 17
		8'h31: char_dots = 40'b00000000_00100010_00111111_00100000_00000000; //
49	 1
		8'h32: char_dots = 40'b00100010_00110001_00101001_00100110_00000000; //
50	 2
		8'h33: char_dots = 40'b00010001_00100101_00100101_00011011_00000000; //
51	 3
		8'h34: char_dots = 40'b00001100_00001010_00111111_00001000_00000000; //
52	 4
		8'h35: char_dots = 40'b00010111_00100101_00100101_00011001_00000000; //
53	 5
		8'h36: char_dots = 40'b00011110_00100101_00100101_00011000_00000000; //
54	 6
```

```
            8'h37: char_dots = 40'b00000001_00110001_00001101_00000011_00000000; //
55      7
            8'h38: char_dots = 40'b00011010_00100101_00100101_00011010_00000000; //
56      8
            8'h39: char_dots = 40'b00000110_00101001_00101001_00011110_00000000; //
57      9
            8'h3A: char_dots = 40'b00000000_00110110_00110110_00000000_00000000; //
58      :          --> 27
            8'h3B: char_dots = 40'b01000000_00110110_00010110_00000000_00000000; //
59      ;
            8'h3C: char_dots = 40'b00000000_00001000_00010100_00100010_00000000; //
60      <
            8'h3D: char_dots = 40'b00010100_00010100_00010100_00010100_00000000; //
61      =
            8'h3E: char_dots = 40'b00000000_00100010_00010100_00001000_00000000; //
62      >
            8'h3F: char_dots = 40'b00000000_00000010_00101001_00000110_00000000; //
63      ?
            8'h40: char_dots = 40'b00011110_00100001_00101101_00001110_00000000; //
64      @
            8'h41: char_dots = 40'b00111110_00001001_00001001_00111110_00000000; //
65      A          --> 34
            8'h42: char_dots = 40'b00111111_00100101_00100101_00011010_00000000; //
66      B
            8'h43: char_dots = 40'b00011110_00100001_00100001_00010010_00000000; //
67      C
            8'h44: char_dots = 40'b00111111_00100001_00100001_00011110_00000000; //
68      D
            8'h45: char_dots = 40'b00111111_00100101_00100101_00100001_00000000; //
69      E
            8'h46: char_dots = 40'b00111111_00000101_00000101_00000001_00000000; //
70      F
            8'h47: char_dots = 40'b00011110_00100001_00101001_00111010_00000000; //
71      G
            8'h48: char_dots = 40'b00111111_00000100_00000100_00111111_00000000; //
72      H
            8'h49: char_dots = 40'b00000000_00100001_00111111_00100001_00000000; //
73      I
            8'h4A: char_dots = 40'b00010000_00100000_00100000_00011111_00000000; //
74      J
            8'h4B: char_dots = 40'b00111111_00001100_00010010_00100001_00000000; //
75      K
            8'h4C: char_dots = 40'b00111111_00100000_00100000_00100000_00000000; //
76      L
            8'h4D: char_dots = 40'b00111111_00000110_00000110_00111111_00000000; //
77      M
            8'h4E: char_dots = 40'b00111111_00000110_00011000_00111111_00000000; //
78      N
            8'h4F: char_dots = 40'b00011110_00100001_00100001_00011110_00000000; //
79      O
            8'h50: char_dots = 40'b00111111_00001001_00001001_00000110_00000000; //
80      P
            8'h51: char_dots = 40'b00011110_00110001_00100001_01011110_00000000; //
81      Q
            8'h52: char_dots = 40'b00111111_00001001_00011001_00100110_00000000; //
82      R
            8'h53: char_dots = 40'b00010010_00100101_00101001_00010010_00000000; //
83      S
            8'h54: char_dots = 40'b00000000_00000001_00111111_00000001_00000000; //
84      T
            8'h55: char_dots = 40'b00011111_00100000_00100000_00011111_00000000; //
85      U
```

```
         8'h56: char_dots = 40'b00001111_00110000_00110000_00001111_00000000; //
86    V
         8'h57: char_dots = 40'b00111111_00011000_00011000_00111111_00000000; //
87    W
         8'h58: char_dots = 40'b00110011_00001100_00001100_00110011_00000000; //
88    X
         8'h59: char_dots = 40'b00000000_00000111_00111000_00000111_00000000; //
89    Y
         8'h5A: char_dots = 40'b00110001_00101001_00100101_00100011_00000000; //
90    Z         --> 59
         8'h5B: char_dots = 40'b00000000_00111111_00100001_00100001_00000000; //
91    [
         8'h5C: char_dots = 40'b00000010_00000100_00001000_00010000_00000000; //
92    \
         8'h5D: char_dots = 40'b00000000_00100001_00100001_00111111_00000000; //
93    ]
         8'h5E: char_dots = 40'b00000000_00000010_00000001_00000010_00000000; //
94    ^
         8'h5F: char_dots = 40'b00100000_00100000_00100000_00100000_00000000; //
95    _
         8'h60: char_dots = 40'b00000000_00000001_00000010_00000000_00000000; //
96    '
         8'h61: char_dots = 40'b00011000_00100100_00010100_00111100_00000000; //
97    a         --> 66
         8'h62: char_dots = 40'b00111111_00100100_00100100_00011000_00000000; //
98    b
         8'h63: char_dots = 40'b00011000_00100100_00100100_00000000_00000000; //
99    c
         8'h64: char_dots = 40'b00011000_00100100_00100100_00111111_00000000; //
100   d
         8'h65: char_dots = 40'b00011000_00110100_00101100_00001000_00000000; //
101   e
         8'h66: char_dots = 40'b00001000_00111110_00001001_00000010_00000000; //
102   f
         8'h67: char_dots = 40'b00101000_01010100_01010100_01001100_00000000; //
103   g
         8'h68: char_dots = 40'b00111111_00000100_00000100_00111000_00000000; //
104   h
         8'h69: char_dots = 40'b00000000_00100100_00111101_00100000_00000000; //
105   i
         8'h6A: char_dots = 40'b00000000_00100000_01000000_00111101_00000000; //
106   j
         8'h6B: char_dots = 40'b00111111_00001000_00010100_00100000_00000000; //
107   k
         8'h6C: char_dots = 40'b00000000_00100001_00111111_00100000_00000000; //
108   l
         8'h6D: char_dots = 40'b00111100_00001000_00001100_00111000_00000000; //
109   m
         8'h6E: char_dots = 40'b00111100_00000100_00000100_00111000_00000000; //
110   n
         8'h6F: char_dots = 40'b00011000_00100100_00100100_00011000_00000000; //
111   o
         8'h70: char_dots = 40'b01111100_00100100_00100100_00011000_00000000; //
112   p
         8'h71: char_dots = 40'b00011000_00100100_00100100_01111100_00000000; //
113   q
         8'h72: char_dots = 40'b00111100_00000100_00000100_00001000_00000000; //
114   r
         8'h73: char_dots = 40'b00101000_00101100_00110100_00010100_00000000; //
115   s
         8'h74: char_dots = 40'b00000100_00011111_00100100_00100000_00000000; //
116   t
```

```
            8'h75: char_dots = 40'b00011100_00100000_00100000_00111100_00000000; //
117     u
            8'h76: char_dots = 40'b00000000_00011100_00100000_00011100_00000000; //
118     v
            8'h77: char_dots = 40'b00111100_00110000_00110000_00111100_00000000; //
119     w
            8'h78: char_dots = 40'b00100100_00011000_00011000_00100100_00000000; //
120     x
            8'h79: char_dots = 40'b00001100_01010000_00100000_00011100_00000000; //
121     y
            8'h7A: char_dots = 40'b00100100_00110100_00101100_00100100_00000000; //
122     z          --> 91
            8'h7B: char_dots = 40'b00000000_00000100_00011110_00100001_00000000; //
123     {
            8'h7C: char_dots = 40'b00000000_00000000_00111111_00000000_00000000; //
124     |
            8'h7D: char_dots = 40'b00000000_00100001_00011110_00000100_00000000; //
125     }
            8'h7E: char_dots = 40'b00000010_00000001_00000010_00000001_00000000; //
126     ~          --> 95
            default:     char_dots =
40'b01000001_01000001_01000001_01000001_01000001;
    endcase

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// synchronize: synchronizes asynchronous inputs to specified clock
//
////////////////////////////////////////////////////////////////////////////////

module synchronize(clk, in, in_sync);

  parameter NSYNC = 2;  // number of sync flops.  must be >= 2

  input clk;
  input in;

  output in_sync;

  reg [NSYNC-2:0] sync;
  reg in_sync;

  always @ (posedge clk)
  begin
    {in_sync,sync} <= {sync[NSYNC-2:0],in};
  end

endmodule
```

```
module coordinates(reset,clk,hcount,vcount,
                    x1,y1,x2,y2,x3,y3,x4,y4,
                 vram_addr,vram_read_data);

// 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.

/*           front
               ^
           _______
          |1     2|
          |       |
          |       |
          |       |
          |4_____3|

          1=red
          2=blue
          3=green
          4=orange
*/
   input reset, clk;
   input [10:0] hcount;
   input [9:0]      vcount;
   output [9:0] x1;
   output [9:0] y1;
   output [9:0] x2;
   output [9:0] y2;
   output [9:0] x3;
   output [9:0] y3;
   output [9:0] x4;
   output [9:0] y4;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;

   wire [18:0]       vram_addr = {1'b0, vcount, hcount[9:2]};

   wire [1:0] hc4 = hcount[1:0];
   reg [35:0]        vr_data_latched;
   reg [35:0]        last_vr_data;
   reg [9:0] x1;
   reg [9:0] y1;
   reg [9:0] x2;
   reg [9:0] y2;
   reg [9:0] x3;
   reg [9:0] y3;
   reg [9:0] x4;
   reg [9:0] y4;

// latch new data every few lines
   always @(posedge clk) begin
     last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;
       end

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


// filter for each corner
   always @(posedge clk) begin
```

```
	if (last_vr_data[23:16]>140 && last_vr_data[15:8]<80 && last_vr_data[7:0]<80)
begin  //red
		x1<=hcount[9:0];
		y1<=vcount[9:0];
	end
	else begin
		x1<=800;
		y1<=800;
	end
	if (last_vr_data[23:16]<90 && last_vr_data[15:8]<80 && last_vr_data[7:0]>90)
begin  //blue
		x2<=hcount[9:0];
		y2<=vcount[9:0];
	end
	else begin
		x2<=800;
		y2<=800;
	end
	if (last_vr_data[23:16]<90 && last_vr_data[15:8]>90 && last_vr_data[7:0]<100)
begin  //green
		x3<=hcount[9:0];
		y3<=vcount[9:0];
	end
	else begin
		x3<=800;
		y3<=800;
	end
	if (last_vr_data[23:16]>160 && last_vr_data[15:8]>90 && last_vr_data[7:0]<80)
begin  //orange
		x4<=hcount[9:0];
		y4<=vcount[9:0];
	end
	else begin
		x4<=800;
		y4<=800;
	end
	end

endmodule
```

```
module movement(clk,inhibit_left,inhibit_right,inhibit_forward,
       inhibit_back,left,right,forward,back,
       left_out,right_out,forward_out,back_out);

       input clk;
       input inhibit_left,inhibit_right,inhibit_forward,inhibit_back;
       input left,right,forward,back;
       output left_out,right_out,forward_out,back_out;

       reg left_out,right_out,forward_out,back_out;


       always @(posedge clk) begin
       if (left && !inhibit_left) begin
              left_out <= 1'h1;
              right_out <= 1'h0;
       end
       else if (right && !inhibit_right) begin
              right_out <=1'h1;
              left_out <= 1'h0;
       end
       else begin
              right_out <=1'h0;
              left_out <=1'h0;
       end
      if (forward && !inhibit_forward) begin
              forward_out <=1'h1;
              back_out <=1'h0;
       end
       else if (back && !inhibit_back) begin
              back_out <=1'h1;
              forward_out <=1'h0;
       end
       else begin
              back_out <=1'h0;
              forward_out <=1'h0;
       end
       end

endmodule
```

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

   input       clk;  // system clock
   input       vclk; // video clock from camera
   input [2:0]       fvh;
   input       dv;
   input [29:0]      din;
   output [18:0] ntsc_addr;
   output [35:0] ntsc_data;
   output      ntsc_we;    // write enable for NTSC data
   input reset;

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

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 640 x 480 XGA display

   reg [9:0]   col = 0;
   reg [9:0]   row = 0;
   reg [29:0] vdata = 0;
   wire [7:0] vdatar;
   wire [7:0] vdatag;
   wire [7:0] vdatab;
   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[29:0] : vdata;
         end
     end

       YCrCb2RGB ntsc_to_rgb ( vdatar, vdatag, vdatab, clk, reset,
                          vdata[29:20], vdata[19:10], vdata[9:0]);

   // synchronize with system clock

   reg [9:0] x[1:0],y[1:0];
   reg [7:0] data[5:0];
   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],vdatar};
        {data[3],data[2]} <= {data[2],vdatag};
        {data[5],data[4]} <= {data[4],vdatab};
        {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 [31:0] mydata;
   always @(posedge clk)
     if (we_edge)
 //      mydata <= { mydata[23:0], data[1] };
       mydata <= { mydata[31:24],data[1],data[3],data[5] };

   // compute address to store data in

   wire [18:0] myaddr = {1'b0, y[1][8:0], eo[1], x[1][9:2]};

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

   reg [18:0] ntsc_addr;
   reg [35:0] ntsc_data;
   wire       ntsc_we = we_edge;

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

endmodule // ntsc_to_zbt
```

```
module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
   output    vsync;
   output    hsync;
   output    blank;

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

   // horizontal: 1344 pixels total         640x480
   // display 640 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 639);
   assign    hsyncon = (hcount == 1047);
   assign    hsyncoff = (hcount == 1183);
   assign    hreset = (hcount == 1343);

   // vertical: 806 lines total
   // display 480 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 479);
   assign    vsyncon = hreset & (vcount == 776);
   assign    vsyncoff = hreset & (vcount == 782);
   assign    vreset = hreset & (vcount == 805);

   // sync and blanking
   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

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

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

```
module 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
```

```
module YCrCb2RGB ( R, G, B, clk, rst, Y, Cr, Cb );

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

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

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

//registering constants
always @ (posedge clk)
begin
 const1 = 10'b 0100101010; //1.164 = 01.00101010
 const2 = 10'b 0110011000; //1.596 = 01.10011000
 const3 = 10'b 0011010000; //0.813 = 00.11010000
 const4 = 10'b 0001100100; //0.392 = 00.01100100
 const5 = 10'b 1000000100; //2.017 = 10.00000100
end

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

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

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



/*always @ (posedge clk or posedge rst)
   if (rst)
      begin
```

```
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     R_int <= X_int + (const2 * (Cr_reg - 'd512));
     G_int <= X_int - (const3 * (Cr_reg - 'd512)) - (const4 * (Cb_reg - 'd512));
     B_int <= X_int + (const5 * (Cb_reg - 'd512));
     end

*/
/* limit output to 0 - 4095, <0 equals o and >4095 equals 4095 */
assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

endmodule
```

```
///////////////////////////////////////////////////////////////////////////
//
// labkit: object position/orientation tracker & maps (excerpt)
//
///////////////////////////////////////////////////////////////////////////

   // use FPGA's digital clock manager to produce a
   // 50MHz clock (actually 49.85MHz)
   wire clock_50mhz_unbuf,clock_50MHz, clock_27mhz;
   DCM vclk1(.CLKIN(xclock_27mhz),.CLKFX(clock_50mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 13
   // 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_50MHz),.I(clock_50mhz_unbuf));

   BUFG vclk5(.O(clock_27mhz),.I(xclock_27mhz));
   // power-on reset generation
   wire power_on_reset2;
   SRL16 reset_sr2 (.D(1'b0), .CLK(clock_50MHz), .Q(power_on_reset2),
               .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr2.INIT = 16'hFFFF;


//    Coordinates calculation from NTSC (video stored in ZBT)
// ****************************START*****************************
   // use FPGA's digital clock manager to produce a
   // 65MHz clock (actually 64.8MHz)
   wire clock_65mhz_unbuf,clock_65mhz;
   DCM vclk3(.CLKIN(xclock_27mhz),.CLKFX(clock_65mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk3 is 10
   // synthesis attribute CLKFX_MULTIPLY of vclk3 is 24
   // synthesis attribute CLK_FEEDBACK of vclk3 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk3 is 37
   BUFG vclk4(.O(clock_65mhz),.I(clock_65mhz_unbuf));
   wire clk = clock_65mhz;

   // power-on reset generation
   wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clk), .Q(power_on_reset),
               .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;

   // generate basic XVGA video signals
   wire [10:0] zbt_hcount;
   wire [9:0]  zbt_vcount;
   wire zbt_hsync,zbt_vsync,zbt_blank;
   xvga xvga1(clk,zbt_hcount,zbt_vcount,zbt_hsync,zbt_vsync,zbt_blank);

   // wire up to ZBT ram
   wire [35:0] vram_write_data;
   wire [35:0] vram_read_data;
   wire [18:0] vram_addr;
   wire        vram_we;
   zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
               vram_write_data, vram_read_data,
```

```
            ram0_clk, ram0_we_b, ram0_address, ram0_data, ram0_cen_b);


   // Coordinates interface code
   // ***************************
   wire [1:0] reset_coords;
   wire [9:0]     x1,y1,x2,y2,x3,y3,x4,y4;
   wire [18:0] vram_addr1;
   wire [9:0] set1, set2, set3, set4;
   wire [9:0]
latched_x1,latched_y1,latched_x2,latched_y2,latched_x3,latched_y3,latched_x4,
latched_y4;
   wire [9:0] in_x1,in_y1,in_x2,in_y2,in_x3,in_y3,in_x4,in_y4;
   coordinates
coord(reset,clk,hcount,vcount,in_x1,in_y1,in_x2,in_y2,in_x3,in_y3,in_x4,in_y4
,
                vram_addr1,vram_read_data);

  reg [79:0] second_point;
  reg [79:0] first_point;

   assign reset_coords = ((hcount == 0) && (vcount == 0)) ? reset_coords +1:
reset_coords;

   assign latched_x1 = (in_x1 != 800) ? in_x1 : 800;
   assign latched_y1 = (in_y1 != 800) ? in_y1 : 800;
   assign latched_x2 = (in_x2 != 800) ? in_x2 : 800;
   assign latched_y2 = (in_y2 != 800) ? in_y2 : 800;
   assign latched_x3 = (in_x3 != 800) ? in_x3 : 800;
   assign latched_y3 = (in_y3 != 800) ? in_y3 : 800;
   assign latched_x4 = (in_x4 != 800) ? in_x4 : 800;
   assign latched_y4 = (in_y4 != 800) ? in_y4 : 800;

  always @ (posedge clk)
  begin
    {second_point,first_point} <=
{first_point,latched_x1,latched_y1,latched_x2,latched_y2,latched_x3,latched_y
3,latched_x4,latched_y4};
  end

  assign x1 = (reset_coords ==0) ? 800 : (((second_point[79:70]==in_x1+2) &&
(set1 == 5)) ? second_point[79:70] : x1);
  assign y1 = (reset_coords ==0) ? 800 : (((second_point[69:60]==in_y1) &&
(set1 == 5)) ? second_point[69:60] : y1);
  assign x2 = (reset_coords ==0) ? 800 : (((second_point[59:50]==in_x2+2) &&
(set2 == 5)) ? second_point[59:50] : x2);
  assign y2 = (reset_coords ==0) ? 800 : (((second_point[49:40]==in_y2) &&
(set2 == 5)) ? second_point[49:40] : y2);
  assign x3 = (reset_coords ==0) ? 800 : (((second_point[39:30]==in_x3+2) &&
(set3 == 5)) ? second_point[39:30] : x3);
  assign y3 = (reset_coords ==0) ? 800 : (((second_point[29:20]==in_y3) &&
(set3 == 5)) ? second_point[29:20] : y3);
  assign x4 = (reset_coords ==0) ? 800 : (((second_point[19:10]==in_x4+2) &&
(set4 == 5)) ? second_point[19:10] : x4);
  assign y4 = (reset_coords ==0) ? 800 : (((second_point[9:0]==in_y4) &&
(set4 == 5)) ? second_point[9:0] : y4);
```

```
  assign set1 = (reset_coords ==0) ? 0 : (((second_point[79:70]==in_x1+2) &&
(second_point[69:60]==in_y1)) ? set1+1 : set1);
  assign set2 = (reset_coords ==0) ? 0 : (((second_point[59:50]==in_x2+2) &&
(second_point[49:40]==in_y2)) ? set2+1 : set2);
  assign set3 = (reset_coords ==0) ? 0 : (((second_point[39:30]==in_x3+2) &&
(second_point[29:20]==in_y3)) ? set3+1 : set3);
  assign set4 = (reset_coords ==0) ? 0 : (((second_point[19:10]==in_x4+2) &&
(second_point[9:0]==in_y4)) ? set4+1 : set4);
   //***********************************


   // 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 (clk, tv_in_line_clock1, fvh, dv, ycrcb[29:0],
                ntsc_addr, ntsc_data, ntsc_we, reset);

   // mux selecting read/write to memory based on which write-enable is
chosen
   wire     my_we = (zbt_hcount[1:0]==2'd2);
   wire [18:0]    write_addr = ntsc_addr;
   wire [35:0]    write_data = ntsc_data;
   assign   vram_addr = my_we ? write_addr : vram_addr1;
   assign   vram_we = my_we;
   assign   vram_write_data = write_data;

   wire     b,hs,vs;
   delayN dn1(clk,zbt_hsync,hs);    // delay by 3 cycles to sync with ZBT
read
   delayN dn2(clk,zbt_vsync,vs);
   delayN dn3(clk,zbt_blank,b);

//***************END*****************


   ////// BUTTONS //////

         // BUTTON_ENTER is changemap
         wire changemap;
         wire changemap_sync;
         assign changemap = ~button_enter;
```

```
        synchronize changemap_synchronizer (clock_27mhz, changemap,
changemap_sync);

        // BUTTON0 is reset for hex display
        wire hex_reset;
        wire hex_reset_sync;
        assign hex_reset = ~button0;
        synchronize hexreset_synchronizer (clock_27mhz, hex_reset,
hex_reset_sync);

        // BUTTON3 is reset for timer
        wire timer_reset;
        wire timer_reset_sync;
        assign timer_reset = ~button3;
        synchronize timerreset_synchronizer (clock_27mhz, timer_reset,
timer_reset_sync);

   ////// SWITCHES //////

        // switch[0] and switch[1] for map selection
        wire [1:0] mapselect;
        assign mapselect[0] = switch[0];
        assign mapselect[1] = switch[1];

        // switch[6] for timer mode
        wire timer_mode;
        assign timer_mode = switch[6];

        // switch[7] for reset mode
        wire reset_mode;
        assign reset_mode = switch[7];

        ////// MAPS //////
        wire [1:0] map_param;
       ////// OBJECT TRACKER //////
        wire vgahsync;
        wire vgavsync;
        wire vgablank;
        wire pix_clk;
        wire [2:0] pixel;
        wire halt_front;
        wire halt_back;
        wire halt_right;
        wire halt_left;
        ////// TEST CAR  //////

           wire [9:0] x1f;
           wire [9:0] y1f;
           wire [9:0] x2f;
           wire [9:0] y2f;
           wire [9:0] x3f;
           wire [9:0] y3f;
           wire [9:0] x4f;
           wire [9:0] y4f;
           wire [9:0] x1b;
           wire [9:0] y1b;
           wire [9:0] x2b;
```

```
            wire [9:0] y2b;
            wire [9:0] x3b;
            wire [9:0] y3b;
            wire [9:0] x4b;
            wire [9:0] y4b;
            wire [9:0] x1r;
            wire [9:0] y1r;
            wire [9:0] x2r;
            wire [9:0] y2r;
            wire [9:0] x1lt;
            wire [9:0] y1lt;
            wire [9:0] x2lt;
            wire [9:0] y2lt;

            wire [9:0] x1_sync, y1_sync, x2_sync, y2_sync, x3_sync, y3_sync,
x4_sync, y4_sync;

      point_sync sync1(clock_50MHz, {x1,y1}, {x1_sync,y1_sync});
      point_sync sync2(clock_50MHz, {x2,y2}, {x2_sync,y2_sync});
      point_sync sync3(clock_50MHz, {x3,y3}, {x3_sync,y3_sync});
      point_sync sync4(clock_50MHz, {x4,y4}, {x4_sync,y4_sync});

   ////// MAPS //////
   // generate map_param based on user selection

         maps maphandler(clock_27mhz, changemap_sync, mapselect, x1_sync,
y1_sync, x2_sync, y2_sync, x3_sync, y3_sync, x4_sync, y4_sync, map_param);

   ////// CAR STATE //////
   // determines state (direction) of car and proper points to use for
boundary checking
         carstate car_direction(clock_27mhz, x1_sync, y1_sync, x2_sync,
y2_sync, x3_sync, y3_sync, x4_sync, y4_sync, x1f, y1f, x2f, y2f, x3f, y3f,
x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r, x2r, y2r, x1lt,
y1lt, x2lt, y2lt);

   ////// OBJECT TRACKER //////
   // produce proper control feedback and display signals

         object_tracker objtrack(clock_50MHz, map_param, reset_mode, x1_sync,
y1_sync, x2_sync, y2_sync, x3_sync, y3_sync, x4_sync, y4_sync, x1f, y1f, x2f,
y2f, x3f, y3f, x4f, y4f, x1b, y1b, x2b, y2b, x3b, y3b, x4b, y4b, x1r, y1r,
x2r, y2r, x1lt, y1lt, x2lt, y2lt, vgahsync, vgavsync, vgablank, pix_clk,
pixel, halt_front, halt_back, halt_right, halt_left);

   ////// TEST CAR  //////
/*
         car testcar1(vsync,  forward_sync, back_sync, right_sync, left_sync,
halt_front, halt_back, halt_right, halt_left, car_1x, car_1y);
         defparam testcar1.X_COORD = 335;
         defparam testcar1.Y_COORD = 245;
         car testcar2(vsync,  forward_sync, back_sync, right_sync, left_sync,
halt_front, halt_back, halt_right, halt_left, car_2x, car_2y);
         defparam testcar2.X_COORD = 340;
         defparam testcar2.Y_COORD = 240;
         car testcar3(vsync,  forward_sync, back_sync, right_sync, left_sync,
halt_front, halt_back, halt_right, halt_left, car_3x, car_3y);
```

```
         defparam testcar3.X_COORD = 350;
         defparam testcar3.Y_COORD = 250;
         car testcar4(vsync,  forward_sync, back_sync, right_sync, left_sync,
halt_front, halt_back, halt_right, halt_left, car_4x, car_4y);
         defparam testcar4.X_COORD = 345;
         defparam testcar4.Y_COORD = 255;
*/

   ////// HEX DISPLAY  //////
   // produce proper hex display signals
          wire halt = halt_front | halt_back | halt_right | halt_left;

          hexdisplay textdisp(clock_27mhz, timer_reset_sync, hex_reset_sync,
map_param, x1_sync, y1_sync, x2_sync, y2_sync, x3_sync, y3_sync, x4_sync,
y4_sync, timer_mode, halt, disp_blank, disp_clock, disp_rs, disp_ce_b,
disp_reset_b, disp_data_out);

   ////// DISPLAY SIGNALS  /////
   // produce proper hex display signals

   assign     vga_out_red = {8{pixel[2]}};
   assign     vga_out_green = {8{pixel[1]}};
   assign     vga_out_blue = {8{pixel[0]}};
   assign     vga_out_sync_b = 1'b1;
   assign     vga_out_blank_b = ~vgablank;
   assign     vga_out_pixel_clock = pix_clk;
   assign     vga_out_hsync = vgahsync;
   assign     vga_out_vsync = vgavsync;


// movement module
//***************start**************
debounce db_left(reset,clk,~button_left,left);
debounce db_right(reset,clk,~button_right,right);
debounce db_up(reset,clk,~button_up,forward);
debounce db_down(reset,clk,~button_down,back);

movement mvt(clk,
halt_left,halt_right,halt_front,halt_back,left,right,forward,back,
      left_out, right_out,forward_out,back_out);

reg [9:0] move_count;
always @ (posedge clk) begin
      if (move_count>200) move_count <=0;
      else move_count<=move_count+1;
end

assign user1 = {31'hZ,~forward_out || (move_count>20)};
assign user2 = {31'hZ,~back_out || (move_count>20)};
assign user3 = {31'hZ,~left_out};
assign user4 = {31'hZ,~right_out};
//***************end***************

endmodule
```