MASSACHUSETTS INSTITUTE OF TECHNOLOGY

POSITION CONTROL FOR A WIRELESS, AUTONOMOUS VEHICLE

KRISHNA SETTALURI,
KEVIN ZHENG,
JORGE SIMOSA

6.111 FINAL PROJECT
REPORT

ABSTRACT

Modern Global Positioning Systems utilize a complex array of satellite data in order to triangulate the position of a target to within a few feet. However, such systems are limited by factors such as interference when positioning objects that are located inside of buildings. The aim of this project is to fill in this blind zone by emulating the GPS model in order to control the position of an autonomous, miniature, and wireless vehicle placed in an indoor field. By relying on hardware rather than software programs, we can develop a system that is cheap, fast, and reliable in order to provide indoor positioning.

TABLE OF CONTENTS

Table of Figures

OVERVIEW

This report details the creation and testing of a camera-based, obstacle-avoiding, position-control system for a miniature, toy tank. A National Television System Committee-standard (NTSC) camera was utilized in capturing the visual data of the tank and its surroundings. A small, Flashing© remote-controlled, toy tank was the subject of position-control and a Xilinx FPGA was also utilized in running the necessary calculations as well as controlling the inputs and outputs of the system. Based on a desired-position input from a user through the GUI, appropriate signals were sent to the remote-control of the tank. This, in turn, led to the motion of the tank in either the straight, left, or right directions. The field, tank, and camera are displayed clearly in **Figure 1**.

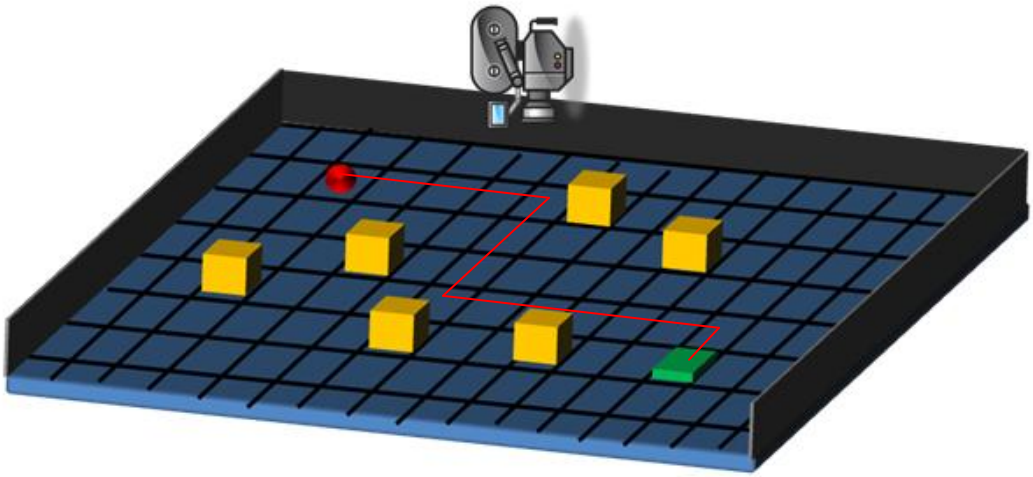

Figure 1Diagram of system. Red ball indicates robot, yellow cubes indicate obstacles, and green selection represents destination.

In addition, throughout this motion, the NTSC camera tracked the position of the tank and fed it back into the control system on the FPGA. As a result, a feedback loop was created which was composed of a set of continuous inputs from the camera for the position as well as from the GUI, the FPGA which outputted signals to the remote-control, and the actual motion of the tank. This yielded stable and reliable performance which, upon further modifications, produced a state-of-the-art position-control system for the toy tank. The plans, modifications, tests, and results are detailed in this report.

MOTIVATION

The need for a wireless, camera-based position-controlled, autonomous vehicle is seen in a wide array of applications and fields. Tasks such as the quick, reliable, and effective transportation of medical equipment from room to room may be easily accomplished with such an autonomous vehicle. More specifically, a doctor's request for

a particular tool or equipment may be inputted into a GUI located in every operating room and office. When such a request is made, the appropriate tool would be placed on a trolley which contained the aforementioned system. From here, the trolley would maneuver through the maze of hallways and elevators, all while avoiding obstacles, based on the data from surveillance cameras located throughout the hospital. Not only would this save time and effort on the part of the hospital's staff, but the effective, autonomous transportation of such equipment would allow doctors and staff to focus on more important issues rather than being burdened with such simple, mundane tasks.

This concept of wireless, traversing vehicles may also be easily integrated into the corporate world. For example, in cases where individuals require the transportation of packages within a large office building, one simply places the package on a position-controlled, moving platform or trolley, inputs a desired room or locations, and the control system safely and effective transports the item to the destination. This would, again, be beneficial to the business partly by saving costs, but also by producing a more efficient working environment for the employees.

The scope of such a design may be also expanded to focus on military transportation. More specifically, such a design is optimal for unmanned missions in deserted locals, where a particular destination is inputted, and the vehicle traverses the distance by means of satellite data for position control. Here, however, rather than relying on indoor cameras for position, satellite data would be received by, say, an onboard system located within the vehicle itself. The data would indicate the current position of the vehicle, as well as constantly monitor obstacle data impeding the vehicle's motion, and also modifications in user input of the destination. Based on the control mechanisms detailed in this report, the vehicle would travel to the desired position, while constantly avoiding obstacles within its near surroundings. This would not only allow for a safer operating environment for military personnel in charge of moving equipment, but the low cost of such a system would allow for easy integration into the current military's infrastructure.

DESIGN CHARACTERISTICS AND SYSTEM REQUIREMENTS

As mentioned, reliable position-control was implemented in a toy tank. However, particular design choices were made in order to not only simplify the system and increase its reliability, but to also provide as realistic as possible alternatives to the environment within which the vehicle may traverse. For example, one of the biggest design choices made was to position the camera such that it was located six feet from the ground, and looking down towards the field. Not only does this provide a large space for motion for the toy tank, but resolution limitations and lighting issues from the camera constrained the maximum height to six feet. Apart from this, constraints on the motion of the vehicle were also placed. More specifically, the tank was allowed to move in either the parallel or perpendicular directions to the field's rectangular boundaries. More specifically, when turning, the tank may only rotate in 90 degree increments, allowing for it to move in either the up, left, or right directions from its current location. Additionally, for easy recognition by the camera, the tank's front and back were colored red and yellow, respectively, and obstacles within the field were constrained to be black in color. Upon testing, the colors red, yellow, and black seemed to be the most responsive to recognition by the camera. Thus, these colors were chosen amongst the dozens tested. Additionally,

again due to the camera's limitations, the field was made white in color in order to facilitate a higher color divide as well as minimize the reflective components of room lights on the field. Materials and components were chosen so as to alleviate image processing issues and produce more reliable measurements and behavior.

The following table details the required components and equipment utilized for the effective implementation of this concept as well as the component's function within the system.

Table 1Components list and function

Component	*Function within system*
FPGA	Control mechanism monitoring inputs and producing accurate outputs
Remote-controlled toy vehicle	Required for the implementation of position-control
Mouse, monitor	GUI
Relays, resistors, BJTs, wire	Allow for FPGA to effectively communicate with toy vehicle
Square, black-painted objects	Obstacles
NTSC standard video camera	Overlooks the field and Video Input

Firstly, a Xilinx FPGA was utilized for running the system code which resulted in the proper behavior of the system. Code was written in Verilog and burnt onto the FPGA. The toy tank for this project was designed by Flashing© and is part of the SQS series. The remote control of the tank was taken apart and analyzed. Modifications were made to allow for the FPGA to control the position of the tank by sending appropriate signals to the output pins, which were connected to relay circuits, which in turn were connected to the tank's remote control. This is shown in **Figure 2**. The relay circuits consisted of taking inputs from the output pins of the FPGA, running them through bipolar-junction transistors (BJTs) and relays, and running the outputs across the relays into the remote control itself. This, in turn, led to the control of the motion of the tank. Next, the GUI for this system allowed for the user to select the destination by clicking the computer mouse. Information such as the tank's position, the obstacle's positions, and the desired position were displayed on the monitor. Lastly, small, square blocks were utilized as obstacles. As mentioned, due to limitations from the camera, all obstacles were painted black in color so as to ease the image processing. **Figure 3** shows the various components of the system.

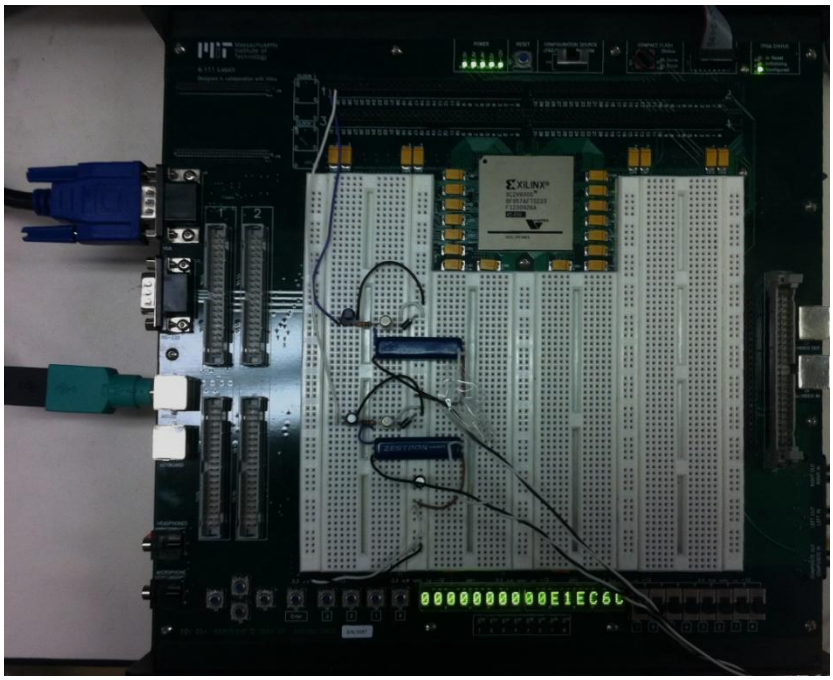

Figure 2 FPGA with relay circuitry

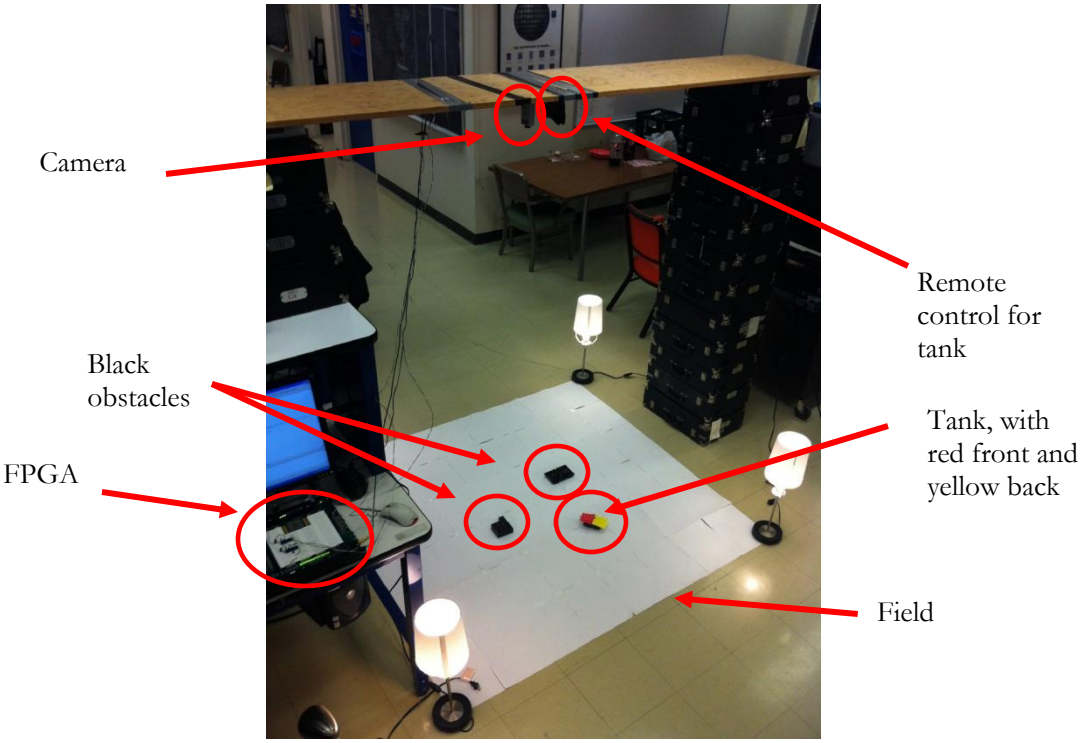

Figure 3 System environment, with key components highlighted.

Due to the relative independence between the various units of this project, the system was easily divided into three large modules. These modules, which will be detailed in a later section, are overviewed below:

1. **Image Processing Module:** Data from the camera needs to be filtered and modified so as to detect obstacles, and the tank. Due to constraints from the camera, only the colors red, yellow, and black were detectable. This module takes as input the data from the camera and outputs x and y positions for the front and back of the tank (indicated with red and yellow colors) and a three bit value indicating if an obstacle is present either in the front or sides of the tank.

2. **Graphics User Interface:** The aim of the GUI for this system was to ensure that a user's inputs were accurately detected and transmitted. More specifically, the user utilizes a computer mouse to click a point on a gridded field, which was displayed on the monitor. The x and y position of the mouse click were outputted from this module. In addition, apart from just clicking a desired position, the user also has the ability to select multiple desired positions (up to eight). Also, three boxes are displayed above the gridded field. A click in one of these boxes either displays a grid view of the field, what the camera sees, or both.

3. **Control FSM Module:** The aim of the control FSM is to ensure desirable and correct motion of the tank. More specifically, this module takes as inputs data from the GUI as well as the Image Processing module. This data indicates the x and y positions of the front and back of the tank, as well as a three-bit value indicating if an obstacle is present in either the front, left, or right of the current position of the tank. In addition, this module was designed so as to continuously monitor and modify the output signals. This translates the ability of the module to modify the motion of the tank instantaneously based on either a change in the destination position, current position of the tank, or an obstacle being detected in front of the tank.

Figure 2 shows the various elements of the block diagram, highlighting the three main independent units of the project, listed above.

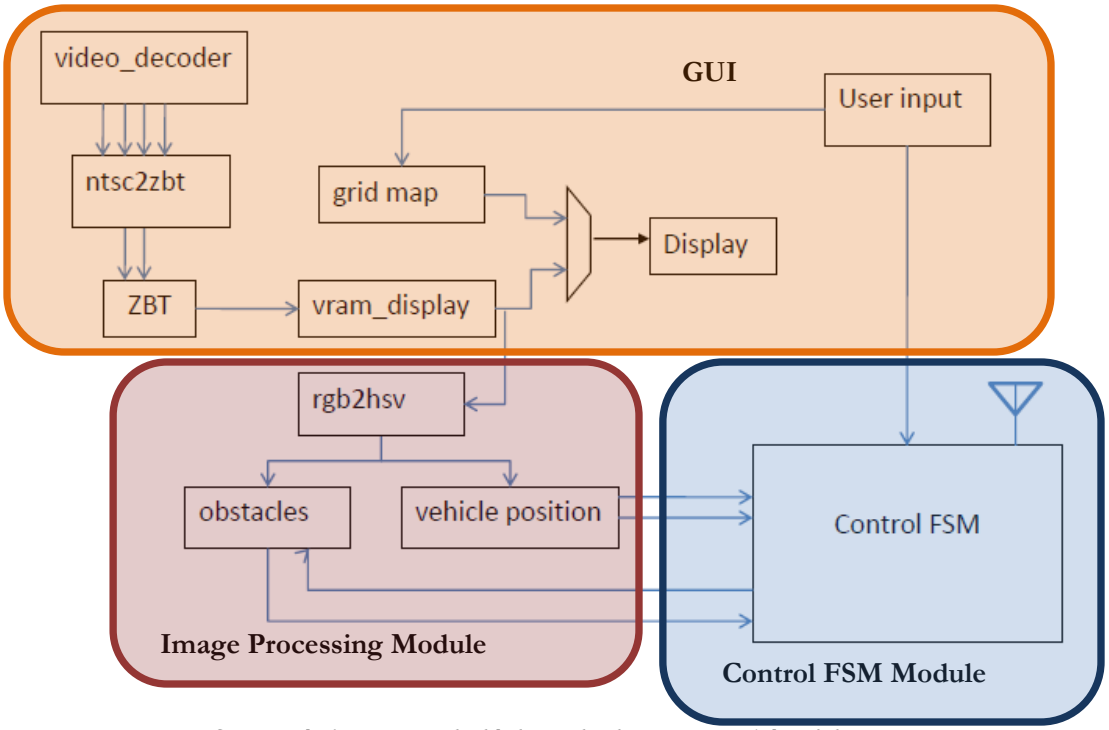

Figure 4 System Block Diagram, highlighting the three main modules of the project.

MODULE ANALYSIS

The following section details each module independently, focusing primarily on the means of communication between the various modules as well as the defining helper function within each main module.

IMAGE PROCESSING MODULES

The image processing modules involve color value conversion, vehicle position filtering, as well as obstacle detection. The conversion module converts RGB value to HSV value in order to filter color more efficiently. The vehicle position finding module uses center of masses of the front and back markers on top of the vehicle. An average buffer is implemented to eliminate undesired effects from noise. The obstacle detection module uses a look up table to store whether a grid on the map has obstacle. The processing unit processes video pixel by pixel.

1. ***RGB2HSV:*** The RGB to HSV converter module converts a 24-bit RGB value into a 24-bit HSV value, with each parameter ranging from 0 to 255. According to several experiments conducted, a simple RGB threshold filtering is not sufficient enough in order to precisely locate our objects of interest due to its highly volatile and sensitive nature to outside conditions, such as lighting and glare. A hue filtering performs a lot better in these cases in non-ideal environments. In order to

reduce noise from the environment to minimum, we chose red and yellow (of which the field noise turns out to contain less) and filter with saturation and luminance threshold as well. This module is pipelined and contains several divider cores generated by coregent. There is a 22 clock cycle delay in this module, which has to be taken into account when the hsv value is passed onto the next module.

2. ***vehicle_filter***: This module takes in the hsv value of a given pixel, along with its corresponding hcount and vcount (which both have to be delayed by 22 clock cycles in order to compensate for the clock delay from the rgb2hsv converter module), and outputs the center of mass of the two color blocks put on top of the vehicle. The outputs are two 21 bit values, with their first 11 bits to be the x (in pixels) and the last 10 bits to be y. The center of mass is found by accumulating the hcount and vcount of pixels whose hsv values pass the threshold and keep a count of the number of pixels which passed the test. When the scans of pixels of interest complete, we send the accumulated value and the count to a divider to calculate the average value, which is by definition the center of mass. At the beginning of each frame, we clear both the count and accumulator to calculate the new center of mass.

3. ***average_ buffer***: In order to deal with noise better, we decide to implement this average_buffer module to average the center of mass of the last 16 frames. The module contains a circular buffer that stores the most recent 16 frames' center of masses coming from the vehicle filter module. We chose 16 since it will translate to a simple right shift operation which doesn't require much computation, as well as the fact that it is not significant amount of delay in terms of updating the current position of the vehicle and passing it to the control module. At each frame, the module increments the offset pointer, stores the new center of mass at the pointer position in the buffer, and updates the new average. Results have shown that it significantly stabilizes the position tracking of the vehicle.

4. ***obstacle_detection***: This module detects obstacles present in the field and provides information necessary for both display as well as vehicle control. The module uses the same information as the vehicle filtering module in order to detect obstacles, except now we make assumptions that our obstacles will all be dark colors. The mechanism in which the module detects obstacles and interfaces with other module is to obtain information in two maps. Each map has location IDs as the keys (there are totally 70 grids in the map, therefore 70 entries in each map), and they map to the count of pixels that pass the threshold test for

obstacles. If the count of pixels in a location exceeds a COUNT_THRESHOLD, then the module will say that there is an obstacle in that location. The threshold checking mechanism ensures that speckles that may accidentally be treated as an obstacle pixel will not have effect on obstacle detections.

location	count
0	30
1	20
2	10
3	0
4	0

⇨

location	count
0	35
1	20
2	15
3	2
4	0

The map on the left is a temporary map whose count values will be cleared at each frame in order to detect obstacle in the most recent field image. At each frame the temporary map will be copied to the map on the right, which will be the one interfacing with other modules. This mechanism allows dynamic obstacle detection since the module redetects by clearing the temporary map at each frame, while making sure there is no instance when the other modules sees no obstacles (with all count 0 in the map entries). However, this mechanism will cause one frame delay in obstacle detection, but it is tolerable since this delay is negligible. The module implements the map as two 70 location register arrays, which have the advantages of simultaneous and parallel queries from different modules (mainly the display and vehicle control whose query locations are different from time to time).

Figure 5 shows the block diagram of image processing unit of the system

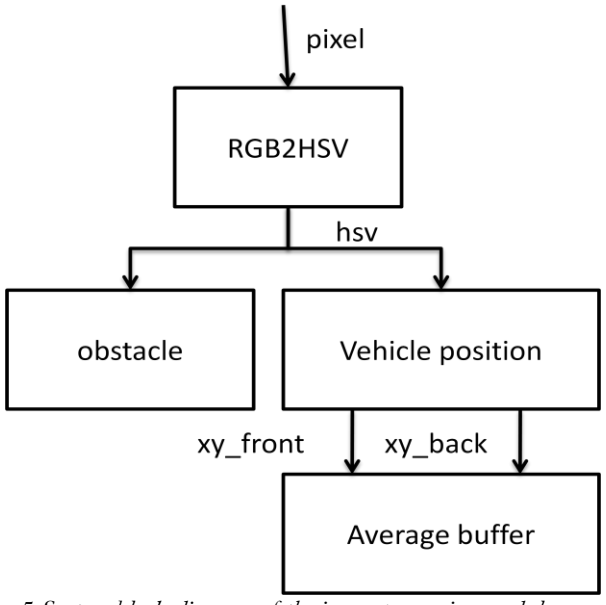

Figure 5 System block diagram of the image processing modules

USER INTERFACE MODULES

The User Interface Module encompasses several sub-modules that are involved in displaying the video data, displaying information about the current situation, and allow the user to select desired positions for the autonomous vehicle. The video data gathered from the central NTSC camera is stored on the on-board memory ZBT RAM to provide inputs for both the Display and Video Processing modules. The user interface has three different viewing modes, namely the Video view, Virtual view, and Both (virtual overlapped on the video) view. The User Interface module also contains the functionality of tracking the mouse position and detecting/maintaining the desired positions.

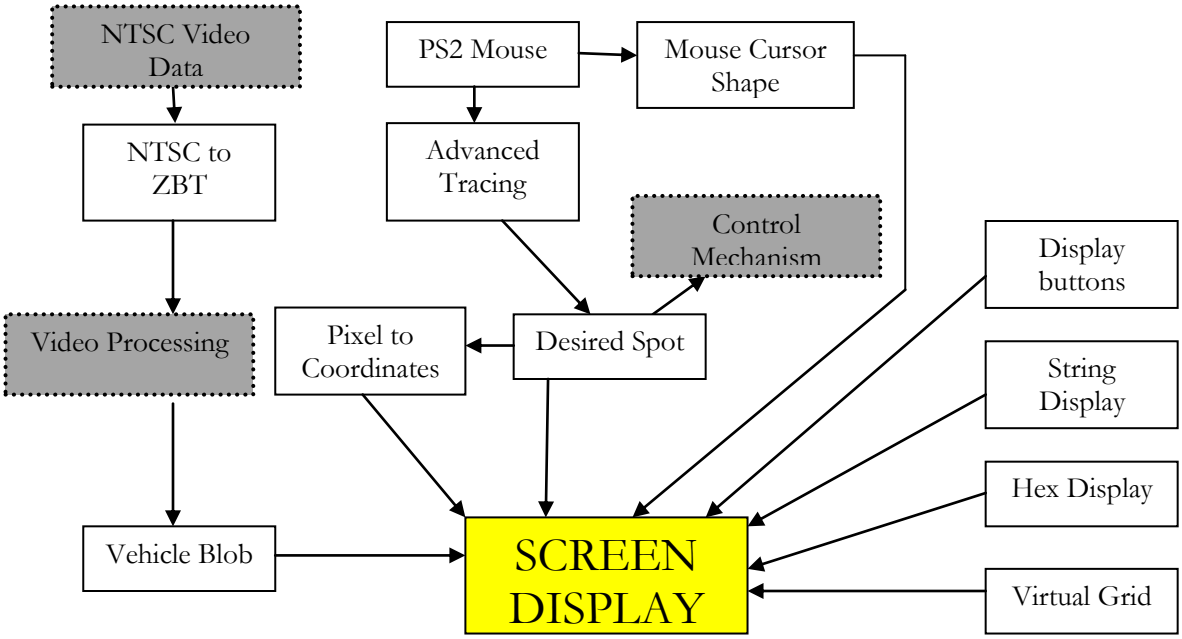

Figure 6 System block diagram of the user interface modules

Figure 6 shows an overall system block diagram showing dependencies between modules, where the shaded gray modules involve outside information. The following sections describe the modules in detail:

1. ***virtual_grid***: The virtual_grid module creates the grid lines which are displayed on the Virtual View based on the parameters of the map size and the desired screen location. The size of the virtual map corresponds to the size of the video data display in order to ease the overlapped feature in the Both view. The grid size is designed to be $O(2^n)$ to simplify later computations such as a coordinate system.
2. ***ntsc_decoder*:** This module's job is to convert the input signals from the camera and output a 30-bit YCrCb value along with other signals such as field, vsync and hsync.

3. ***ntsc_to_zbt***: This module takes YCrCb values from the ntsc_decoder and corresponding field, hcount, vcount signals to generate an address in the ZBT memory. Modification is made to this module so that it stores 2 pixels of RGB values (by instantiation of an YCrCb2RGB converter module found on Xilinx, and truncated the 24 bit value down to an 18 bit value) in each ZBT address.

4. ***vram_display***: The vram_display module generates address to read from the ZBT memory, and output a 24 bit RGB value as the pixel value to be displayed on screen or processed. Due to the 2 clock cycle delay in ZBT memory, addresses are generated such that we look ahead in the memory, and data is also latched to compensate for an extra delay.

5. ***ps2_mouse***: The ps2_mouse module takes care of the communication between the mouse and the labkit to easily extract information of the mouse such as position and button status. The module ranges from the low-level of the PS2 interface to the high-level of extracting mouse coordinates. The parameters were adapted to use the 40 MHz system clock that is universal to other modules in this system.

6. ***adv_tracing***: The adv_tracing module is in charge of tracking the mouse position and detecting valid mouse clicks that are translated into desired positions. The module checks to see if the mouse click occurs within the dimensions of the map, otherwise it will not be detected as a desired position. In order to allow the user to select multiple desired locations, we have also included a circular buffer which contains the pending desired positions. The buffer fans out to eight separate desired position instances in order to display the pending desired positions simultaneously on the screen.

7. ***mouse_cursor_shape*:** The mouse_cursor_shape module contains the mathematical model used to create the mouse cursor to be displayed. In the current design, the mouse cursor is designed as a red cross of a designated height, width, and thickness. By displaying the mouse cursor, the user can easily select desired positions as well as change between views.

8. ***desired_spot***: The desired_spot module is similar to the mouse_cursor_shape module in that the desired_spot are also in the form of a cross. However to differentiate the desired positions from the mouse cursor, we have assigned a green color to the desired spots. Additionally, we check to make sure the coordinates of the desired position are within the map boundaries, thus indicating a valid desired position.

9. ***vehicle_blob***: The vehicle_blob module uses a template similar to the desired_spot and mouse_cursor_shape modules in order to display the current position of the vehicle when in the Virtual view mode. The current design of the vehicle_blob involves a rectangular shape which has the colors red, white, and blue mimicking the United States flag. This module also ensures that the vehicle_blob will never be located outside of the map.

10. ***display_buttons***: The display_buttons module takes care of displaying the different view mode buttons and tracking possible mouse clicks within the area of the buttons. The buttons are designed using a basic rectangular shape of a predetermined width and height and are centered at the top of the display screen. The module also contains logic which tracks the mouse events and checks to see if the mouse click is within the area of one of the display buttons. By detecting the mouse release, we can then change the index of the display mode.

11. ***pixel_to_coordinates***: The pixel_to_coordinates module is in charge of converting the pixel hcount and vcount into a coordinate system for the map. Since the virtual grid is broken up into grids with a size of 64, we can easily use the offset of the map and divide by the grid size to create a coordinate system that ranges from (0,0) to (9,6). This module provides the coordinates that are located at the bottom of the screen (actual and desired positions).

12. ***cstringdisp***: The cstringdisp module allows for the display of characters on the screen, such as the labels for the display buttons and the grid coordinates. The module makes use of a font rom to obtain the correct pixel values based on the calculated font address. The size of the characters is currently 8 by 12 pixels. We can specify the position of the characters by providing an input of the upper-leftmost corner of the display area.

13. ***big_vga_hexdisp4***: The big_vga_hexdisp4 module is similar to the cstringdisp module in allowing us to display hex values on the screen. This module uses a different font rom which contains the pixel values for the hex numbers. The size of the hex digits is 8 by 12 pixels. The hex digits that are displayed on the screen correspond to the actual position of the vehicle and the next desired position, if available.

CONTROL FSM MODULE

The following shows the finite-state machine for the Control FSM module:

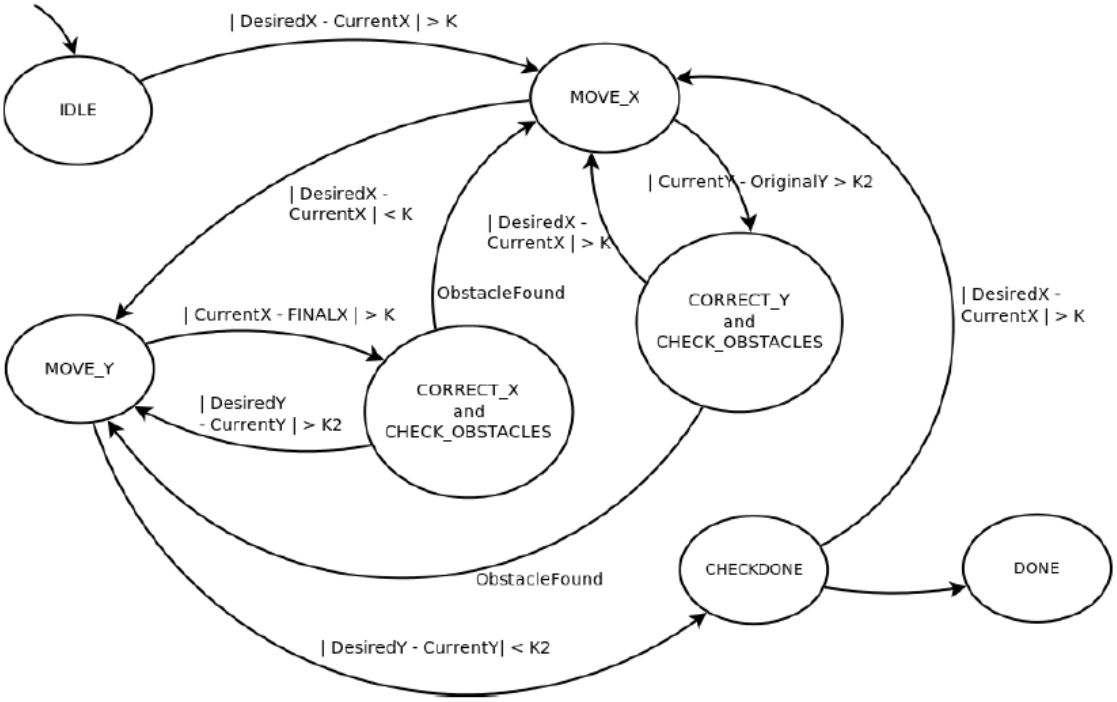

Figure 7 Control FSM

The inputs and outputs for the system are listed below:

1. **Inputs**:
    a. **desiredX and desiredY** from the GUI module
    b. **frontX and frontY** from the Image Processing module. This value indicates the center of mass of the red part on the front of the tank.
    c. **backX and backY** from the Image Processing module. This value indicates the center of mass of the yellow part on the back of the tank.
    d. **Obstacles** is a three bit value indicating if an obstacle is present to the left, right, or in front of the tank.
2. **Outputs:**
    a. **Left_wheel** is a single bit signal on the USER1[31] pin on the FPGA. This value, when 1, goes through the relay circuitry and controls if the left wheel of the tank is rotating or not.

b. **Right_wheel** is a single bit signal on the USER1[30] pin on the FPGA. This value controls the right wheel of the tank. With the Left_wheel and Right_wheel values both 1, for example, the tank moves forward.

Figure 7 shows the various states associated with the Control FSM. A total of seven states were utilized for controlling the motion. In addition, some helper states were also utilized in order to ensure steady, consistent motion. These helper states primarily focused on ensuring that the tank rotates and moves forward in a controlled manner. The following summarizes the various states and components:

1. **IDLE:** Upon resetting the system, or turning on the system for the first time, the initial state is IDLE. In this state, two checks are made:

    a. **Check 1:** Is the desired x position different from the current x (current = abs[frontX – backX]) position of the tank? If so, then go to the MOVE_X state.

    b. **Check 2:** Is the desired y position different from the current y of the robot? If so, then go to the MOVE_Y state. If this is not true, then the tank is in the desired position and the state remains unchanged.

2. **MOVE_X:** The aim of this state is to ensure that the tank aligns itself in the + x or – x directions and traverse to the desired x position. Upon aligning in the appropriate x direction (depending on the relative magnitude of the desiredX compared with currentX), two scenarios may arise. These two situations are handled in the CORRECT_Y_CHECK_OBSTACLES state:

3. **CORRECT_Y_CHECK_OBSTACLES:** This state checks two different conditions and jumps back to either the MOVE_X state or the MOVE_Y state depending on the conditions, listed below:

    a. **Check for obstacles:** The MSB of the three bit input into the Control FSM goes high if an obstacle impedes the current path of the tank. If this value goes high, then the following two scenarios arise:

        i. **Is the tank aligned in the y direction?** If so, then deviate off of the desired path and realign in the y direction, while simultaneously checking the LSB and 2^{nd} bit of the Obstacles value. Effectively, the tank moves around the obstacle towards the desired destination position.

        ii. **If the tank is not aligned in the y direction**, then adjust in the y direction by transitioning to the MOVE_Y state.

    b. **Rotate the tank either left or right if it deviates from the straight line path:** If the frontY of the tank passes beyond either the backY + ydeviation or backY – ydeviation, then adjust in order to ensure

straight motion. The state transitions to a mini-state, which steps through the turning of the robot, so as to ensure minimal overshooting when aligning in either the left or right directions.

4. **MOVE_Y:** This state, similar to the MOVE_X state, tries to align with the desiredY position. The same conditions hold; however, rather than jumping to the CORRECT_Y_CHECK_OBSTACLES state, the state transitions to the CORRECT_X_CHECK_OBSTACLES state.

5. **CORRECT_X_CHECK_OBSTACLES:** Similar to its counterpart in the y direction, this state either checks for obstacles impeding the tank's path, or if the frontX passes a set deviation when comparing with the backX. The same conditions hold. However, the state transitions to the mini-state associated with turning the robot so as to orient itself in either the up or down directions.

6. **CHECK_DONE:** This state constantly checks the initial condition indicating if the tank is in the desired position in the x and y directions. From here, the state either changes to the MOVE_X or MOVE_Y state depending on the difference of the desired position with respect to the current position of the robot. Additionally, if the desired position and current position of the tank are relatively close together, then the state changes to the DONE state.

7. **DONE:** Indicates that the tank is in the desired position. Here, appropriate signals are sent to the GUI module in order to indicate that the desired position has been attained. This signal is used when multiple destinations are selected by the user.

FUTURE WORK

The system performs according to the aforementioned requirements. In order to further this project in the future, the following additions/modifications may be made to enhance the performance or increase the scope of this concept:

1. **Integrate sensors onboard the vehicle**: Sensors such as proximity detectors on the tank may be utilized to double check obstacle avoidance. With the camera data as well as the wireless transmission of sensor data, the tank more reliably avoids obstacles and traverses to the destination.

2. **Utilize multiple cameras**: In order to increase the size of the field, multiple camera data may be used. This ensures a more reliable set of obstacle and tank position data.

3. **Effectively translate project so as to cover lab area**: Strategically place cameras around the lab area (room 34-600), create a grid of the room to display on the monitor, and have the robot find the optimal path from its current position to a selected desired position.

4. **Increase functions of vehicle**: More specifically, input not only the desired position of the vehicle, but also input a desired action, such a picking up an object, dropping off an object, etc. From these inputs, the vehicle should traverse to the destination and perform the desired action. Sensors onboard the vehicle will ensure correct operation during the performance of the action.

CONCLUSION

A novel, position-control system was built and tested on a small, toy tank. The system performs reliably and consistently on the 8 foot by 8 foot field. A camera was utilized in order to capture elements on the field. An Image Processing module was built in order to acquire the appropriate current position of the tank. Additionally, a GUI was designed so as to not only input multiple desired destinations for the tank, but also switch between the camera view, the grid view, and an overlap of the two views. Lastly, the Control FSM utilized the various outputs from the GUI module and the Image Processing module and translates these inputs into correct signals on the USER1[31] and USER1[30] pins on the FPGA. From here, the signals are fed into the relay circuitry, which in turn controls the motion of the tank from the remote-control. This led to the effective motion of the tank from its current position to the user's desired position.

ACKNOWLEDGMENTS

The group would like to thank the following individuals for their advice and assistance: Jacky Chang, Hossein Fariborzi, Professor Stojanovic, and, of course, Gim Hom.

```
//////////////////////////////////////////////////////////////////////////
////////
//
// Pushbutton Debounce Module
//
//////////////////////////////////////////////////////////////////////////
////////

module debounce (reset, clk, noisy, clean);
  input reset, clk, noisy;
  output clean;

  parameter NDELAY = 650000;
  parameter NBITS = 20;

  reg [NBITS-1:0] count;
  reg xnew, clean;

  always @(posedge clk)
    if (reset) begin xnew <= noisy; clean <= noisy; count <= 0; end
    else if (noisy != xnew) begin xnew <= noisy; count <= 0; end
    else if (count == NDELAY) clean <= xnew;
    else count <= count+1;

endmodule

//////////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 02-Nov-05 Ike: updated to make it completely synchronous
//
// Inputs:
//
//   reset      - active high
//   clock_27mhz - the synchronous clock
```

```
//   data       - 64 bits; each 4 bits gives a hex digit
//
// Outputs:
//
//    disp_*    - display lines used in the 6.111 labkit (rev 003 & 004)
//
//////////////////////////////////////////////////////////////////////
////////

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

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

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

  reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;


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

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

  reg [5:0] count;
  reg [7:0] reset_count;
//   reg           old_clock;
  wire     dreset;
  wire     clock = (count<27) ? 0 : 1;

  always @(posedge clock_27mhz)
    begin
      count <= reset ? 0 : (count==53 ? 0 : count+1);
      reset_count <= reset ? 100 : ((reset_count==0) ? 0 : reset_count-1);
//   old_clock <= clock;
    end

  assign dreset = (reset_count != 0);
  assign disp_clock = ~clock;
  wire   clock_tick = ((count==27) ? 1 : 0);
//   wire   clock_tick = clock & ~old_clock;
```

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

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

  reg [7:0] state;       // FSM state
  reg [9:0] dot_index;          // index to current dot being clocked out
  reg [31:0] control;       // control register
  reg [3:0] char_index;   // index of current character
  reg [39:0] dots;          // dots for a single digit
  reg [3:0] nibble;          // hex nibble of current character
  reg [63:0] data;

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

  always @(posedge clock_27mhz)
   if (clock_tick)
     begin
      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;
  end

8'h05:
  begin
    // Latch the control register data / dot data
    disp_ce_b <= 1'b1;
    dot_index <= 39;          // init for single char
    char_index <= 15;         // start with MS char
    data <= data_in;
    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 <= dots[dot_index]; // dot data from msb
    if (dot_index == 0)
      if (char_index == 0)
        state <= 5;               // all done, latch data
      else
        begin
          char_index <= char_index - 1;   // goto next char
          data <= data_in;
```

```
                dot_index <= 39;
              end
            else
              dot_index <= dot_index-1;     // else loop thru all dots
          end

        endcase // casex(state)
      end

    always @ (data or char_index)
      case (char_index)
        4'h0:           nibble <= data[3:0];
        4'h1:           nibble <= data[7:4];
        4'h2:           nibble <= data[11:8];
        4'h3:           nibble <= data[15:12];
        4'h4:           nibble <= data[19:16];
        4'h5:           nibble <= data[23:20];
        4'h6:           nibble <= data[27:24];
        4'h7:           nibble <= data[31:28];
        4'h8:           nibble <= data[35:32];
        4'h9:           nibble <= data[39:36];
        4'hA:           nibble <= data[43:40];
        4'hB:           nibble <= data[47:44];
        4'hC:           nibble <= data[51:48];
        4'hD:           nibble <= data[55:52];
        4'hE:           nibble <= data[59:56];
        4'hF:           nibble <= data[63:60];
      endcase

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

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    18:11:50 11/18/2010
// Design Name:
// Module Name:    xvga
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////
module xvga(vclock,hcount,vcount,hsync,vsync,blank);
  input vclock;
  output [10:0] hcount;
  output [9:0] vcount;
  output      vsync;
  output      hsync;
  output      blank;

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

  // horizontal: 1056 pixels total
  // display 800 pixels per line
  wire     hsyncon,hsyncoff,hreset,hblankon;
  assign    hblankon = (hcount == 799);
  assign    hsyncon = (hcount == 839);
  assign    hsyncoff = (hcount == 967);
  assign    hreset = (hcount == 1055);

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

  // sync and blanking
  wire     next_hblank,next_vblank;
  assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
```

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

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

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

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

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

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

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

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

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

  reg [1:0]   we_delay;

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

  // create two-stage pipeline for write data

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

  // wire to ZBT RAM signals

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

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

endmodule // zbt_6111

//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    15:20:36 11/09/2010
// Design Name:
// Module Name:    vram_display
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

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

  input reset, clk;
  input [10:0] hcount;
  input [9:0]      vcount;
  output [23:0] vr_pixel;
  output [18:0] vram_addr;
  input [35:0]  vram_read_data;

  //forecast hcount & vcount 8 clock cycles ahead to get data from ZBT
  wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount + 8);
  wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount + 1) : vcount;

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


  reg [23:0]   vr_pixel;
  reg [35:0]   vr_data_latched;
  reg [35:0]   last_vr_data;

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

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

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

endmodule // vram_display

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

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

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

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

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

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

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

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

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

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

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

  assign       state = current_state;

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

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

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

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

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

          endcase
        end
      end // always @ (posedge clk)

    // implement our decoding mechanism

    wire y_enable;
    wire cr_enable;
    wire cb_enable;

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

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

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

    reg       f = 0;

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

endmodule
```

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

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

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

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

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

`define VIDEO_QUALITY                   2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE            1'b0
```

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

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

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

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

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

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

`define INTERFACE_SELECT                    2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
```

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

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

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

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

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

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


`define GENERAL_PURPOSE_OUTPUTS         4'b0000
`define GPO_0_1_ENABLE                  1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                  1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI             1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
```

```
`define HLOCK_ENABLE                    1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

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

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

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

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

////////////////////////////////////////////////////////////////////////
////////
// Register 8
////////////////////////////////////////////////////////////////////////
////////

`define INPUT_CONTRAST_ADJUST           8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

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

`define INPUT_SATURATION_ADJUST         8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////
////////
// Register A
```

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

`define INPUT_BRIGHTNESS_ADJUST                 8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

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

`define INPUT_HUE_ADJUST                        8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

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

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

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

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

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

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////
////////
// Register E
```

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

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

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

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

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

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

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

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

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

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

```
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

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

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


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

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

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

```
  $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
  $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
end

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

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

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

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

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

assign reset_slow = reset_count != 0;

//
// I2C driver
//

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

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

//
// State machine
//
```

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

always @(posedge clock_slow)
  if (reset_slow)
   begin
     state <= 0;
     load <= 0;
     tv_in_reset_b <= 0;
     old_source <= 0;
   end
  else
   case (state)
     8'h00:
      begin
        // Assert reset
        load <= 1'b0;
        tv_in_reset_b <= 1'b0;
        if (!ack)
         state <= state+1;
      end
     8'h01:
      state <= state+1;
     8'h02:
      begin
        // Release reset
        tv_in_reset_b <= 1'b1;
        state <= state+1;
           end
     8'h03:
      begin
        // Send ADV7185 address
        data <= 8'h8A;
        load <= 1'b1;
        if (ack)
         state <= state+1;
      end
     8'h04:
      begin
        // Send subaddress of first register
        data <= 8'h00;
        if (ack)
         state <= state+1;
      end
     8'h05:
      begin
        // Write to register 0
        data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
        if (ack)
         state <= state+1;
```

```
  end
8'h06:
  begin
    // Write to register 1
    data <= `ADV7185_REGISTER_1;
    if (ack)
     state <= state+1;
  end
8'h07:
  begin
    // Write to register 2
    data <= `ADV7185_REGISTER_2;
    if (ack)
     state <= state+1;
  end
8'h08:
  begin
    // Write to register 3
    data <= `ADV7185_REGISTER_3;
    if (ack)
     state <= state+1;
  end
8'h09:
  begin
    // Write to register 4
    data <= `ADV7185_REGISTER_4;
    if (ack)
     state <= state+1;
  end
8'h0A:
  begin
    // Write to register 5
    data <= `ADV7185_REGISTER_5;
    if (ack)
     state <= state+1;
  end
8'h0B:
  begin
    // Write to register 6
    data <= 8'h00; // Reserved register, write all zeros
    if (ack)
     state <= state+1;
  end
8'h0C:
  begin
    // Write to register 7
    data <= `ADV7185_REGISTER_7;
    if (ack)
     state <= state+1;
  end
8'h0D:
  begin
```

```
    // Write to register 8
    data <= `ADV7185_REGISTER_8;
    if (ack)
     state <= state+1;
  end
8'h0E:
  begin
    // Write to register 9
    data <= `ADV7185_REGISTER_9;
    if (ack)
     state <= state+1;
  end
8'h0F: begin
   // Write to register A
   data <= `ADV7185_REGISTER_A;
 if (ack)
   state <= state+1;
end
8'h10:
  begin
    // Write to register B
    data <= `ADV7185_REGISTER_B;
    if (ack)
     state <= state+1;
  end
8'h11:
  begin
    // Write to register C
    data <= `ADV7185_REGISTER_C;
    if (ack)
     state <= state+1;
  end
8'h12:
  begin
    // Write to register D
    data <= `ADV7185_REGISTER_D;
    if (ack)
     state <= state+1;
  end
8'h13:
  begin
    // Write to register E
    data <= `ADV7185_REGISTER_E;
    if (ack)
     state <= state+1;
  end
8'h14:
  begin
    // Write to register F
    data <= `ADV7185_REGISTER_F;
    if (ack)
     state <= state+1;
```

```
  end
8'h15:
  begin
    // Wait for I2C transmitter to finish
    load <= 1'b0;
    if (idle)
     state <= state+1;
  end
8'h16:
  begin
    // Write address
    data <= 8'h8A;
    load <= 1'b1;
    if (ack)
     state <= state+1;
  end
8'h17:
  begin
    data <= 8'h33;
    if (ack)
     state <= state+1;
  end
8'h18:
  begin
    data <= `ADV7185_REGISTER_33;
    if (ack)
     state <= state+1;
  end
8'h19:
  begin
    load <= 1'b0;
    if (idle)
     state <= state+1;
  end

8'h1A: begin
   data <= 8'h8A;
   load <= 1'b1;
   if (ack)
    state <= state+1;
end
8'h1B:
  begin
    data <= 8'h33;
    if (ack)
     state <= state+1;
  end
8'h1C:
  begin
    load <= 1'b0;
    if (idle)
     state <= state+1;
```

```
      end
    8'h1D:
      begin
        load <= 1'b1;
        data <= 8'h8B;
        if (ack)
         state <= state+1;
      end
    8'h1E:
      begin
        data <= 8'hFF;
        if (ack)
         state <= state+1;
      end
    8'h1F:
      begin
        load <= 1'b0;
        if (idle)
         state <= state+1;
      end
    8'h20:
      begin
        // Idle
        if (old_source != source) state <= state+1;
        old_source <= source;
      end
    8'h21: begin
      // Send ADV7185 address
      data <= 8'h8A;
      load <= 1'b1;
      if (ack) state <= state+1;
    end
    8'h22: begin
      // Send subaddress of register 0
      data <= 8'h00;
      if (ack) state <= state+1;
    end
    8'h23: begin
      // Write to register 0
      data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
      if (ack) state <= state+1;
    end
    8'h24: begin
      // Wait for I2C transmitter to finish
      load <= 1'b0;
      if (idle) state <= 8'h20;
    end
  endcase

endmodule

// i2c module for use with the ADV7185
```

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

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

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

  reg [7:0] state;

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

  always @(posedge clock4x)
   if (reset)
     begin
       state <= 0;
       ack <= 0;
     end
   else
     case (state)
      8'h00: // idle
        begin
          scl <= 1'b1;
          sdai <= 1'b1;
          ack <= 1'b0;
          idle <= 1'b1;
          if (load)
           begin
             ldata <= data;
             ack <= 1'b1;
             state <= state+1;
           end
        end
      8'h01: // Start
        begin
          ack <= 1'b0;
          idle <= 1'b0;
          sdai <= 1'b0;
          state <= state+1;
        end
      8'h02:
        begin
          scl <= 1'b0;
```

```
    state <= state+1;
  end
8'h03: // Send bit 7
  begin
    ack <= 1'b0;
    sdai <= ldata[7];
    state <= state+1;
  end
8'h04:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h05:
  begin
    state <= state+1;
  end
8'h06:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h07:
  begin
    sdai <= ldata[6];
    state <= state+1;
  end
8'h08:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h09:
  begin
    state <= state+1;
  end
8'h0A:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h0B:
  begin
    sdai <= ldata[5];
    state <= state+1;
  end
8'h0C:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h0D:
```

```
  begin
    state <= state+1;
  end
8'h0E:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h0F:
  begin
    sdai <= ldata[4];
    state <= state+1;
  end
8'h10:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h11:
  begin
    state <= state+1;
  end
8'h12:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h13:
  begin
    sdai <= ldata[3];
    state <= state+1;
  end
8'h14:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h15:
  begin
    state <= state+1;
  end
8'h16:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h17:
  begin
    sdai <= ldata[2];
    state <= state+1;
  end
8'h18:
```

```
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h19:
  begin
    state <= state+1;
  end
8'h1A:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h1B:
  begin
    sdai <= ldata[1];
    state <= state+1;
  end
8'h1C:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h1D:
  begin
    state <= state+1;
  end
8'h1E:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h1F:
  begin
    sdai <= ldata[0];
    state <= state+1;
  end
8'h20:
  begin
    scl <= 1'b1;
    state <= state+1;
  end
8'h21:
  begin
    state <= state+1;
  end
8'h22:
  begin
    scl <= 1'b0;
    state <= state+1;
  end
8'h23: // Acknowledge bit
```

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

endmodule

//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
```

```
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09
//
// Reduced the clock to 40 MHz to avoid timing problem. Resolution
// is now 800 * 600 pixels.
//
// Bug due to memory management will be fixed. It happens because
// the memory addressing protocol is off between ntsc2zbt.v and
// vram_display.v. There are 2 solutions:
// -. Fix the memory addressing in this module (neat addressing protocol)
//    and do memory forecast in vram_display module.
// -. Do nothing in this module and do memory forecast in vram_display
//    module (different forecast count) while cutting off reading from
//    address(0,0,0).
//
// Bug in this module causes 4 pixel on the rightmost side of the camera
// to be stored in the address that belongs to the leftmost side of the
// screen.
//
// In this example, the second method is used. NOTICE will be provided
// on the crucial source of the bug.
//
//////////////////////////////////////////////////////////////////////////
//////
// Prepare data and address values to fill ZBT memory with NTSC data

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

  input   clk;// system clock
  input   vclk;    // video clock from camera
  input [2:0]       fvh;
  input   dv;
  input [29:0]      din;
  output [18:0] ntsc_addr;
  output [35:0] ntsc_data;
  output       ntsc_we;    // write enable for NTSC data
  input   sw;       // switch which determines mode (for debugging)

  parameter       COL_START = 10'd40;
  parameter       ROW_START = 10'd18;

  // here put the luminance data from the ntsc decoder into the ram
  // this is for 800 * 600 XGA display

  reg [9:0]     col = 0;
  reg [9:0]     row = 0;
  reg [29:0]   vdata = 0;
  reg           vwe;
```

```
  reg           old_dv;
  reg           old_frame; // frames are even / odd interlaced
  reg           even_odd;  // decode interlaced frame to this wire

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

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

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

  // synchronize with system clock

  reg [9:0] x[1:0],y[1:0];
  reg [29:0] data[1:0];
  reg       we[1:0];
  reg         eo[1:0];

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

  // edge detection on write enable signal

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

  // shift each set of four bytes into a large register for the ZBT
  wire [7:0] RGB[2:0];
     YCrCb2RGB convert(RGB[2], RGB[1],RGB[0],clk, 0, data[1][29:20], data[1][19:10],
data[1][9:0]);

  reg [36:0] mydata;
```

```
always @(posedge clk)
  if (we_edge)
    mydata <= {mydata[15:0], RGB[2][7:2],RGB[1][7:2], RGB[0][7:2]};

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


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

reg [39:0] x_delay;
reg [39:0] y_delay;
reg [3:0] we_delay;
reg [3:0] eo_delay;

always @ (posedge clk)
begin
  x_delay <= {x_delay[29:0], x[1]};
  y_delay <= {y_delay[29:0], y[1]};
  we_delay <= {we_delay[2:0], we[1]};
  eo_delay <= {eo_delay[2:0], eo[1]};
end

// compute address to store data in

wire [18:0] myaddr = {1'b0, y_delay[38:30], eo_delay[3], x_delay[39:32]};
// Now address (0,0,0) contains pixel data(0,0) etc.

    wire [18:0] myaddr_delay;
    //delayN #(.NDELAY(3), .SIZE(19)) delayaddr(.clk(clk), .in(myaddr), .out(myaddr_delay));


// alternate (256x192) image data and address
wire [31:0] mydata2 = {data[1][29:22],data[1][29:22],data[1][29:22],data[1][29:22]};
wire [18:0] myaddr2 = {1'b0, y_delay[38:30], eo_delay[3], x_delay[37:30]};
```

```
  // update the output address and data only when four bytes ready
     wire wedge_delay;
     //delayN #(.NDELAY(3), .SIZE(1)) delaywe(.clk(clk), .in(we_edge & (x_delay[31]==1'b0)),
.out(wedge_delay));

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

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

endmodule // ntsc_to_zbt

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

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



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

  // module parameters
  parameter       MAX_X = 800;
  parameter       MAX_Y = 600;

  // low level mouse driver

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

//   original 6.111 fall 2005 Verilog - appears to be buggy  so it has been
//   commented out.
//   ps2_mouse m1(clk,reset,ps2_clk,ps2_data,dx,dy,ovf_xy, btn_click,
//         data_ready,streaming);
//
```

```
// using ps2_mouse Verilog from Opencore

// divide the clk by a factor of two so that it works with 65mhz and the original timing
// parameters in the open core source.
// if the Verilog doesn't work the user should update the timing parameters. This  Verilog
assumes
// 50Mhz clock; seems to work with 32.5mhz without problems. GPH  11/23/2008 with
// assist from BG

ps2_mouse_interface

  #(.WATCHDOG_TIMER_VALUE_PP(20800),
  .WATCHDOG_TIMER_BITS_PP(15),
  .DEBOUNCE_TIMER_VALUE_PP(198),
  .DEBOUNCE_TIMER_BITS_PP(8))

  m1(
 .clk(clk),
 .reset(reset),
 .ps2_clk(ps2_clk),
 .ps2_data(ps2_data),
 .x_increment(dx),
 .y_increment(dy),
 .data_ready(data_ready),
 .read(1'b1),  // force a read
 .left_button(btn_click[2]),
 .right_button(btn_click[0])  // rx_read_o
 );

//  error_no_ack  not used


  // Update "absolute" position of mouse

  reg [11:0]  mx, my;
  wire       sx = dx[8];           // signs
  wire       sy = dy[8];
  wire [8:0]  ndx = sx ? {0,~dx[7:0]}+1 : {0,dx[7:0]};      // magnitudes
  wire [8:0]  ndy = sy ? {0,~dy[7:0]}+1 : {0,dy[7:0]};

  always @(posedge clk) begin
    mx <= reset ? 0 :
        data_ready ? (sx ? (mx>ndx ? mx - ndx : 0)
                : (mx < MAX_X - ndx ? mx+ndx : MAX_X)) : mx;
    // note Y is flipped for video cursor use of mouse
    my <= reset ? 0 :
        data_ready ? (sy ? (my < MAX_Y - ndy ? my+ndy : MAX_Y)
                : (my>ndy ? my - ndy : 0))  : my;
//      data_ready ? (sy ? (my>ndy ? my - ndy : 0)
//                : (my < MAX_Y - ndy ? my+ndy : MAX_Y)) : my;
  end
```

```
endmodule

//-----------------------------------------------------------------------------
//
// Author: John Clayton
// Date  : April 30, 2001
// Update: 6/06/01 copied this file from ps2.v (pared down).
// Update: 6/07/01 Finished initial coding efforts.
// Update: 6/09/01 Made minor changes to state machines during debugging.
//              Fixed errors in state transitions.  Added state to m2
//              so that "reset" causes the mouse to be initialized.
//              Removed debug port.
//
//
//
//
//
// Description
//-----------------------------------------------------------------------------
// This is a state-machine driven serial-to-parallel and parallel-to-serial
// interface to the ps2 style mouse.  The state diagram for part of the
// m2 state machine was obtained from the work of Rob Chapman, as published
// at:
// www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1998_w/mouse_notes.html
//
//
// Some aspects of the mouse interface are not implemented (e.g, verifying
// the FA response code from the mouse when enabling streaming mode.)
// However, the mouse interface was designed so that "hot plugging" a mouse
// into the connector should cause the interface to send the F4 code to the
// mouse in order to enable streaming.  By this means, the mouse begins to
// operate, and no reset pulse should be needed.
//
// Similarly, there is a "watchdog" timer implemented, so that during periods
// of inactivity, the bit_count is cleared to zero.  Therefore, the effects of
// a bad count value are corrected, and internal errors of that type are not
// propagated into subsequent packet receive operations.
//
// To enable the streaming mode, F4 is sent to the mouse.
// The mouse responds with FA to acknowledge the command, and then enters
// streaming mode at the default rate of 100 packets per second (transmission
// of packets ceases when the activity at the mouse is not longer sensed.)
//
// There are additional commands to change the sampling rate and resolution
// of the mouse reported data.  Those commands are not implemented here.
// (E8,XX = set resolution 0,1,2,3)
// (E7 = set scaling 2:1)
// (E6 = reset scaling)
// (F3,XX = set sampling rate to XX packets per second.)
//
// At this time I do not know any of the command related to using the
// wheel of a "wheel mouse."
```

```
//
// The packets consists of three bytes transmitted in sequence.  The interval
// between these bytes has been measured on two different mice, and found to
// be different.  On the slower (older) mouse it was approximately 345
// microseconds, while on a newer "wheel" mouse it was approximately 125
// microseconds.  The watchdog timer is designed to cause processing of a
// complete packet when it expires.  Therefore, the watchdog timer must last
// for longer than the "inter-byte delay" between bytes of the packet.
// I have set the default timer value to 400 usec, for my 49.152 MHz clock.
// The timer value and size of the timer counter is settable by parameters,
// so that other clock frequencies and settings may be used.  The setting for
// the watchdog timeout is not critical -- it only needs to be greater than
// the inter-byte delay as data is transmitted from the mouse, and no less
// than 60usec.
//
// Each "byte" of the packet is transmitted from the mouse as follows:
//
// 1 start bit, 8 data bits, 1 odd parity bit, 1 stop bit. == 11 bits total.
// (The data bits are sent LSB first)
//
// The data bits are formatted as follows:
//
// byte 0: YV, XV, YS, XS, 1, 0, R, L
// byte 1: X7..X0
// byte 2: Y7..Y0
//
// Where YV, XV are set to indicate overflow conditions.
//       XS, YS are set to indicate negative quantities (sign bits).
//        R,  L are set to indicate buttons pressed, left and right.
//
//
//
// The interface to the ps2 mouse (like the keyboard) uses clock rates of
// 30-40 kHz, dependent upon the mouse itself.  The mouse generates the
// clock.
// The rate at which the state machine runs should be at least twice the
// rate of the ps2_clk, so that the states can accurately follow the clock
// signal itself.  Four times oversampling is better.  Say 200kHz at least.
// In order to run the state machine extremely fast, synchronizing flip-flops
// have been added to the ps2_clk and ps2_data inputs of the state machine.
// This avoids poor performance related to slow transitions of the inputs.
//
// Because this is a bi-directional interface, while reading from the mouse
// the ps2_clk and ps2_data lines are used as inputs.  While writing to the
// mouse, however (which is done when a "packet" of less than 33 bits is
// received), both the ps2_clk and ps2_data lines are sometime pulled low by
// this interface.  As such, they are bidirectional, and pullups are used to
// return them to the "high" state, whenever the drivers are set to the
// high impedance state.
//
// Pullups MUST BE USED on the ps2_clk and ps2_data lines for this design,
// whether they be internal to an FPGA I/O pad, or externally placed.
```

```
// If internal pullups are used, they may be fairly weak, causing bounces
// due to crosstalk, etc.  There is a "debounce timer" implemented in order
// to eliminate erroneous state transitions which would occur based on bounce.
// Parameters are provided to configure the debounce timer for different
// clock frequencies.  2 or 3 microseconds of debounce should be plenty.
// You may possibly use much less, if your pullups are strong.
//
// A parameters is provided to configure a 60 microsecond period used while
// transmitting to the mouse.  The 60 microsecond period is guaranteed to be
// more than one period of the ps2_clk signal.
//
//
//--------------------------------------------------------------------------------


`resetall
`timescale 1ns/100ps

`define TOTAL_BITS   33  // Number of bits in one full packet


module ps2_mouse_interface (
  clk,
  reset,
  ps2_clk,
  ps2_data,
  left_button,
  right_button,
  x_increment,
  y_increment,
  data_ready,      // rx_read_o
  read,           // rx_read_ack_i
  error_no_ack
  );

// Parameters

// The timer value can be up to (2^bits) inclusive.
parameter WATCHDOG_TIMER_VALUE_PP = 19660; // Number of sys_clks for 400usec.
parameter WATCHDOG_TIMER_BITS_PP  = 15;    // Number of bits needed for timer
parameter DEBOUNCE_TIMER_VALUE_PP = 186;   // Number of sys_clks for debounce
parameter DEBOUNCE_TIMER_BITS_PP  = 8;     // Number of bits needed for timer


// State encodings, provided as parameters
// for flexibility to the one instantiating the module.
// In general, the default values need not be changed.

// There are three state machines: m1, m2 and m3.
// States chosen as "default" states upon power-up and configuration:
//    "m1_clk_h"
//    "m2_wait"
```

```
//    "m3_data_ready_ack"

parameter m1_clk_h = 0;
parameter m1_falling_edge = 1;
parameter m1_falling_wait = 3;
parameter m1_clk_l = 2;
parameter m1_rising_edge = 6;
parameter m1_rising_wait = 4;

parameter m2_reset = 14;
parameter m2_wait = 0;
parameter m2_gather = 1;
parameter m2_verify = 3;
parameter m2_use = 2;
parameter m2_hold_clk_l = 6;
parameter m2_data_low_1 = 4;
parameter m2_data_high_1 = 5;
parameter m2_data_low_2 = 7;
parameter m2_data_high_2 = 8;
parameter m2_data_low_3 = 9;
parameter m2_data_high_3 = 11;
parameter m2_error_no_ack = 15;
parameter m2_await_response = 10;

parameter m3_data_ready = 1;
parameter m3_data_ready_ack = 0;

// I/O declarations
input clk;
input reset;
inout ps2_clk;
inout ps2_data;
output left_button;
output right_button;
output [8:0] x_increment;
output [8:0] y_increment;
output data_ready;
input read;
output error_no_ack;

reg left_button;
reg right_button;
reg [8:0] x_increment;
reg [8:0] y_increment;
reg data_ready;
reg error_no_ack;

// Internal signal declarations
wire watchdog_timer_done;
wire debounce_timer_done;
wire packet_good;
```

```
reg [`TOTAL_BITS-1:0] q;  // Shift register
reg [2:0] m1_state;
reg [2:0] m1_next_state;
reg [3:0] m2_state;
reg [3:0] m2_next_state;
reg m3_state;
reg m3_next_state;
reg [5:0] bit_count;      // Bit counter
reg [WATCHDOG_TIMER_BITS_PP-1:0] watchdog_timer_count;
reg [DEBOUNCE_TIMER_BITS_PP-1:0] debounce_timer_count;
reg ps2_clk_hi_z;     // Without keyboard, high Z equals 1 due to pullups.
reg ps2_data_hi_z;    // Without keyboard, high Z equals 1 due to pullups.
reg clean_clk;        // Debounced output from m1, follows ps2_clk.
reg rising_edge;      // Output from m1 state machine.
reg falling_edge;     // Output from m1 state machine.
reg output_strobe;    // Latches data data into the output registers

//-----------------------------------------------------------------------
// Module code

assign ps2_clk = ps2_clk_hi_z?1'bZ:1'b0;
assign ps2_data = ps2_data_hi_z?1'bZ:1'b0;

// State register
always @(posedge clk)
begin : m1_state_register
  if (reset) m1_state <= m1_clk_h;
  else m1_state <= m1_next_state;
end

// State transition logic
always @(m1_state
        or ps2_clk
        or debounce_timer_done
        or watchdog_timer_done
        )
begin : m1_state_logic

  // Output signals default to this value, unless changed in a state condition.
  clean_clk <= 0;
  rising_edge <= 0;
  falling_edge <= 0;

  case (m1_state)
    m1_clk_h :
      begin
        clean_clk <= 1;
        if (~ps2_clk) m1_next_state <= m1_falling_edge;
        else m1_next_state <= m1_clk_h;
      end

    m1_falling_edge :
```

```
    begin
      falling_edge <= 1;
      m1_next_state <= m1_falling_wait;
    end

  m1_falling_wait :
    begin
      if (debounce_timer_done) m1_next_state <= m1_clk_l;
      else m1_next_state <= m1_falling_wait;
    end

  m1_clk_l :
    begin
      if (ps2_clk) m1_next_state <= m1_rising_edge;
      else m1_next_state <= m1_clk_l;
    end

  m1_rising_edge :
    begin
      rising_edge <= 1;
      m1_next_state <= m1_rising_wait;
    end

  m1_rising_wait :
    begin
      clean_clk <= 1;
      if (debounce_timer_done) m1_next_state <= m1_clk_h;
      else m1_next_state <= m1_rising_wait;
    end
  default : m1_next_state <= m1_clk_h;
 endcase
end


// State register
always @(posedge clk)
begin : m2_state_register
 if (reset) m2_state <= m2_reset;
 else m2_state <= m2_next_state;
end

// State transition logic
always @(m2_state
       or q
       or falling_edge
       or rising_edge
       or watchdog_timer_done
       or bit_count
       or packet_good
       or ps2_data
       or clean_clk
       )
```

```
begin : m2_state_logic

  // Output signals default to this value, unless changed in a state condition.
  ps2_clk_hi_z <= 1;
  ps2_data_hi_z <= 1;
  error_no_ack <= 0;
  output_strobe <= 0;

  case (m2_state)

    m2_reset :    // After reset, sends command to mouse.
      begin
        m2_next_state <= m2_hold_clk_l;
      end

    m2_wait :
      begin
        if (falling_edge) m2_next_state <= m2_gather;
        else m2_next_state <= m2_wait;
      end

    m2_gather :
      begin
        if (watchdog_timer_done && (bit_count == `TOTAL_BITS))
          m2_next_state <= m2_verify;
        else if (watchdog_timer_done && (bit_count < `TOTAL_BITS))
          m2_next_state <= m2_hold_clk_l;
        else m2_next_state <= m2_gather;
      end

    m2_verify :
      begin
        if (packet_good) m2_next_state <= m2_use;
        else m2_next_state <= m2_wait;
      end

    m2_use :
      begin
        output_strobe <= 1;
        m2_next_state <= m2_wait;
      end

    // The following sequence of 9 states is designed to transmit the
    // "enable streaming mode" command to the mouse, and then await the
    // response from the mouse.  Upon completion of this operation, the
    // receive shift register contains 22 bits of data which are "invalid"
    // therefore, the m2_verify state will fail to validate the data, and
    // control will be passed into the m2_wait state once again (but the
    // mouse will then be enabled, and valid data packets will ensue whenever
    // there is activity on the mouse.)
    m2_hold_clk_l :
      begin
```

```
   ps2_clk_hi_z <= 0;   // This starts the watchdog timer!
   if (watchdog_timer_done && ~clean_clk) m2_next_state <= m2_data_low_1;
   else m2_next_state <= m2_hold_clk_l;
  end

m2_data_low_1 :
  begin
   ps2_data_hi_z <= 0;  // Forms start bit, d[0] and d[1]
   if (rising_edge && (bit_count == 3))
     m2_next_state <= m2_data_high_1;
   else m2_next_state <= m2_data_low_1;
  end

m2_data_high_1 :
  begin
   ps2_data_hi_z <= 1;  // Forms d[2]
   if (rising_edge && (bit_count == 4))
     m2_next_state <= m2_data_low_2;
   else m2_next_state <= m2_data_high_1;
  end

m2_data_low_2 :
  begin
   ps2_data_hi_z <= 0;  // Forms d[3]
   if (rising_edge && (bit_count == 5))
     m2_next_state <= m2_data_high_2;
   else m2_next_state <= m2_data_low_2;
  end

m2_data_high_2 :
  begin
   ps2_data_hi_z <= 1;  // Forms d[4],d[5],d[6],d[7]
   if (rising_edge && (bit_count == 9))
     m2_next_state <= m2_data_low_3;
   else m2_next_state <= m2_data_high_2;
  end

m2_data_low_3 :
  begin
   ps2_data_hi_z <= 0;  // Forms parity bit
   if (rising_edge) m2_next_state <= m2_data_high_3;
   else m2_next_state <= m2_data_low_3;
  end

m2_data_high_3 :
  begin
   ps2_data_hi_z <= 1;  // Allow mouse to pull low (ack pulse)
   if (falling_edge && ps2_data) m2_next_state <= m2_error_no_ack;
   else if (falling_edge && ~ps2_data)
     m2_next_state <= m2_await_response;
   else m2_next_state <= m2_data_high_3;
  end
```

```
   m2_error_no_ack :
    begin
     error_no_ack <= 1;
     m2_next_state <= m2_error_no_ack;
    end

   // In order to "cleanly" exit the setting of the mouse into "streaming"
   // data mode, the state machine should wait for a long enough time to
   // ensure the FA response is done being sent by the mouse.  Unfortunately,
   // this is tough to figure out, since the watchdog timeout might be longer
   // or shorter depending upon the user.  If the watchdog timeout is set to
   // a small enough value (less than about 560 usec?) then the bit_count
   // will get reset to zero by the watchdog before the FA response is
   // received.  In that case, bit_count will be 11.
   // If the bit_count is not reset by the watchdog, then the
   // total bit_count will be 22.
   // In either case, when this state is reached, the watchdog timer is still
   // running and it is best to let it expire before returning to normal
   // operation.  One easy way to do this is to check for the bit_count to
   // reach 22 (which it will always do when receiving a normal packet) and
   // then jump to "verify" which will always fail for that time.
   m2_await_response :
    begin
     if (bit_count == 22) m2_next_state <= m2_verify;
     else m2_next_state <= m2_await_response;
    end

   default : m2_next_state <= m2_wait;
  endcase
end



// State register
always @(posedge clk)
begin : m3_state_register
 if (reset) m3_state <= m3_data_ready_ack;
 else m3_state <= m3_next_state;
end

// State transition logic
always @(m3_state or output_strobe or read)
begin : m3_state_logic
 case (m3_state)
  m3_data_ready_ack:
      begin
       data_ready <= 1'b0;
       if (output_strobe) m3_next_state <= m3_data_ready;
       else m3_next_state <= m3_data_ready_ack;
      end
  m3_data_ready:
```

```
        begin
          data_ready <= 1'b1;
          if (read) m3_next_state <= m3_data_ready_ack;
          else m3_next_state <= m3_data_ready;
        end
    default : m3_next_state <= m3_data_ready_ack;
  endcase
end

// This is the bit counter
always @(posedge clk)
begin
  if (reset) bit_count <= 0;  // normal reset
  else if (falling_edge) bit_count <= bit_count + 1;
  else if (watchdog_timer_done) bit_count <= 0;  // rx watchdog timer reset
end

// This is the shift register
always @(posedge clk)
begin
  if (reset) q <= 0;
  else if (falling_edge) q <= {ps2_data,q[`TOTAL_BITS-1:1]};
end

// This is the watchdog timer counter
// The watchdog timer is always "enabled" to operate.
always @(posedge clk)
begin
  if (reset || rising_edge || falling_edge) watchdog_timer_count <= 0;
  else if (~watchdog_timer_done)
    watchdog_timer_count <= watchdog_timer_count + 1;
end
assign watchdog_timer_done = (watchdog_timer_count==WATCHDOG_TIMER_VALUE_PP-1);

// This is the debounce timer counter
always @(posedge clk)
begin
  if (reset || falling_edge || rising_edge) debounce_timer_count <= 0;
//  else if (~debounce_timer_done)
  else debounce_timer_count <= debounce_timer_count + 1;
end
assign debounce_timer_done = (debounce_timer_count==DEBOUNCE_TIMER_VALUE_PP-1);

// This is the logic to verify that a received data packet is "valid"
// or good.
assign packet_good = (
                      (q[0]  == 0)
                   && (q[10] == 1)
                   && (q[11] == 0)
                   && (q[21] == 1)
                   && (q[22] == 0)
                   && (q[32] == 1)
```

```
                && (q[9]  == ~^q[8:1])    // odd parity bit
                && (q[20] == ~^q[19:12])  // odd parity bit
                && (q[31] == ~^q[30:23])  // odd parity bit
                );

// Output the special scan code flags, the scan code and the ascii
always @(posedge clk)
begin
  if (reset)
  begin
    left_button <= 0;
    right_button <= 0;
    x_increment <= 0;
    y_increment <= 0;
  end
  else if (output_strobe)
  begin
    left_button <= q[1];
    right_button <= q[2];
    x_increment <= {q[5],q[19:12]};
    y_increment <= {q[6],q[30:23]};
  end
end


endmodule

//////////////////////////////////////////////////////////////////////////////////
///////////
// Company: MIT
// Engineer: Jorge Simosa
//
// Create Date:    18:57:28 11/14/2010
// Design Name:
// Module Name:    grid_map
// Project Name:      Vehicle Control Using Video Surveillance
// Target Devices: Display Screen
// Description: Creates a grid map size that begins at corner (MAP_X, MAP_Y)
// and has a specified map length and width. The grid size is parameter that controls
//  the number of grids in the map. Default grid line color is white.
//
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module grid_map(clk, hcount, vcount, pixel);
      parameter COLOR = 24'hFFFFFF;     //default color: white
      parameter GRID_SIZE = 64;        //grid size
      parameter MAP_LENGTH = 640;      //80-720
      parameter MAP_HEIGHT = 448;      //44-556
      parameter MAP_X = 80;            //left-upper x corner of map
      parameter MAP_Y = 76;            //left-upper y corner of map
```

```
    input clk;                          //system clock
    input [10:0] hcount;            //hcount of pixel data
    input [9:0] vcount;             //vcount of pixel data
    output [23:0] pixel;            //assignment of pixel at (hcount,vcount)

    reg [23:0] cur_pix;             //register holding current pixel value

    always @(posedge clk) begin
        //assigns vertical grid lines based on hcount
        if ((hcount >= MAP_X && hcount < MAP_X + MAP_LENGTH + 1 && vcount >=
MAP_Y && vcount < MAP_Y + MAP_HEIGHT + 1) &&
                ((hcount == MAP_X) || ((hcount - MAP_X) % GRID_SIZE == 0) || (hcount ==
MAP_X + MAP_LENGTH)))
            cur_pix <= COLOR;
        //assigns horizontal grid lines based on vcount
        else if ((hcount >= MAP_X && hcount < MAP_X + MAP_LENGTH + 1&& vcount >=
MAP_Y && vcount < MAP_Y + MAP_HEIGHT + 1) &&
                ((vcount == MAP_Y) || ((vcount - MAP_Y) % GRID_SIZE == 0) || (vcount ==
MAP_Y + MAP_HEIGHT)))
            cur_pix <= COLOR;
        //all other pixels are assigned to black
        else cur_pix <= 0;
    end

    assign pixel = cur_pix;

endmodule

//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: MIT
// Engineer: Jorge Simosa
//
// Create Date:   15:27:48 11/14/2010
// Design Name:
// Module Name:   mouse_cursor_shape
// Project Name:     Vehicle Control Using Video Surveillance
// Target Devices: Display Screen
// Description:  This modules creates the shape of the mouse cursor, which
//  initally is designed to be a red-colored cross.
//
//
//////////////////////////////////////////////////////////////////////////////////
////////////

module mouse_cursor_shape(vclk,x,y,hcount,vcount,pixel);

  parameter WIDTH = 20;                 //width for display area of cursor
  parameter HEIGHT = 20;                //height for display area of cursor
  parameter COLOR = 24'hFF0000;         //red color
    parameter THICKNESS = 5;            //thickness of cross
```

```
  input vclk;                                   //system clock
     input [10:0] x,hcount;                  //x for upper left corner
  input [9:0]  y,vcount;                  //y for upper left corner
  output [23:0] pixel;                        //assignment of output pixel

     reg [10:0] x_center;                     //register that holds center x of cursor
     reg [9:0]  y_center;                     //register that holds center y of cursor
  reg [23:0]  pixel;                        //register holding pixel value

  always @(posedge vclk) begin
          x_center <= x - WIDTH/2;         //calculation to find center x of cursor
          y_center <= y - HEIGHT/2;        //calculation to find center y of cursor
          //assign the vertical line pixels
    if ((hcount >= x_center+(WIDTH/2)-((THICKNESS-1)/2) && hcount <
x_center+(WIDTH/2)+((THICKNESS-1)/2)) &&
                (vcount >= y_center && vcount < y_center + HEIGHT))
                     pixel = COLOR;
          //assign the horizontal line pixels
          else if ((vcount >= y_center+(HEIGHT/2)-((THICKNESS-1)/2) && vcount <
y_center+(HEIGHT/2)+((THICKNESS-1)/2)) &&
                (hcount >= x_center && hcount < x_center + WIDTH))
                     pixel = COLOR;
          //all other pixels are black
    else pixel = 0;
  end
endmodule

//////////////////////////////////////////////////////////////////////////////////
///////////
// Company: MIT
// Engineer: Jorge Simosa
//
// Create Date:    17:17:24 11/17/2010
// Design Name:
// Module Name:    vehicle_blob
// Project Name:      Vehicle Control Using Video Surveillance
// Target Devices: Display Screen
// Description:  This module creates the vehicle blob to be displayed on the
// virtual map. The vehicle has a specified size and is currently designed
// to be a mock-up of the U.S. flag.
//
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module vehicle_blob(vclk,x,y,hcount,vcount,pixel);

  parameter WIDTH = 20;                    // width from center
  parameter HEIGHT = 8;                    // height from center
  parameter COLOR_Y = 24'hFFFF00;     // yellow color
     parameter COLOR_R = 24'hFF0000; // red color
     parameter COLOR_W = 24'hFFFFFF;      // white color
     parameter COLOR_B = 24'h0000FF; // blue color
```

```
  input vclk;                                    // system clock
     input [10:0] x,hcount;                    // x for center
  input [9:0] y,vcount;                      // y for center
  output reg [23:0] pixel;             // output pixels

  always @(posedge vclk) begin
          //top blue square
    if (hcount >= x - WIDTH && hcount <= x && vcount >= y - HEIGHT && vcount <= y)
                  pixel <= COLOR_B;
          //red stripes of flag
          else if ((hcount > x && hcount <= x + WIDTH && vcount >= y - HEIGHT && vcount <=
y - (HEIGHT / 2)) ||
                (hcount >= x - WIDTH && hcount <= x + WIDTH && vcount > y && vcount <= y +
(HEIGHT / 2)) ||
                (hcount >= x - WIDTH && hcount <= x + WIDTH && vcount > y + HEIGHT &&
vcount <= y + HEIGHT + (HEIGHT / 2)))
                  pixel <= COLOR_R;
          //blue stripes of flag
          else if ((hcount > x && hcount <= x + WIDTH && vcount > y - (HEIGHT / 2) && vcount
<= y) ||
                (hcount >= x - WIDTH && hcount <= x + WIDTH && vcount > y + (HEIGHT / 2)
&& vcount <= y + HEIGHT))
                  pixel <= COLOR_W;
          //all other pixels are assigned black
          else pixel <= 0;
  end

endmodule

//////////////////////////////////////////////////////////////////////////////////
///////////
// Company: MIT
// Engineer: Jorge Simosa
//
// Create Date:    18:11:27 11/16/2010
// Design Name:
// Module Name:    tracing
// Project Name:      Vehicle Control Using Video Surveillance
// Target Devices: Control Mechanism
// Description: This module tracks the mouse movements and records any valid
// mouse clicks that can be translated into desired positions into a buffer
// that currently allows for up to 8 unprocessed desired positions.
//
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module adv_tracing(clk, mx, my, btn_status, r_enable, dx, dy, dp1, dp2, dp3, dp4,
                           dp5, dp6, dp7, dp8);
     parameter MAP_X = 80;                                    //left-upper x corner of map
     parameter MAP_Y = 76;                                    //left-upper y corner of map
     parameter MAP_LENGTH = 640;                                  //80-720
```

```
    parameter MAP_HEIGHT = 448;                                 //44-556
    parameter BUFFER_SIZE = 8;                                  //size of d.p. buffer

    input clk;                                                  // clock
    input [9:0] mx, my;                                         // mouse x, y
    input [2:0] btn_status;                                     // button clicking
    input r_enable;                                             // reached current desired position
    output [9:0] dx, dy;                                        // desired x, y
    output [19:0] dp1, dp2, dp3, dp4, dp5, dp6, dp7, dp8;// current eight desired positions

    reg btn_delay;                                              // used to detect mouse release
    reg [19:0] desired_buffer [BUFFER_SIZE-1:0];        // buffer of desired coordinates
    reg [2:0] cur_buffer_read = 0;                      // index for reading from buffer
    reg [2:0] cur_buffer_write = 0;                     // index for writing to buffer
    reg [3:0] count = 0;                                    // number of filled (valid) spots in
buffer

    always @(posedge clk) begin
        //detect a mouse click after the release
        if(btn_delay & ~btn_status[2]) begin
            //check to see if click is within the boundaries of the map
            if((mx >= MAP_X && mx <= MAP_X + MAP_LENGTH) && (my >= MAP_Y && my
<= MAP_Y + MAP_HEIGHT)) begin
                //make sure the buffer is not full, in order to insert new d.p.
                if (count != 8) begin
                    count <= count + 1;
                    desired_buffer[cur_buffer_write] <= {mx,my};
                    cur_buffer_write <= (cur_buffer_write + 1) % BUFFER_SIZE;
                end
            end
        end

        //used to detect mouse release
        btn_delay <= btn_status[2];

        //after receiving a done signal for the current desired position, check and read
        //next desired position
        if(r_enable && count > 0) begin
                count <= count - 1;
                desired_buffer[cur_buffer_read] <= 0;
                cur_buffer_read <= (cur_buffer_read + 1) % BUFFER_SIZE;
        end
    end

    //seperate instances for desired positions
    assign dp1 = desired_buffer[0];
    assign dp2 = desired_buffer[1];
    assign dp3 = desired_buffer[2];
    assign dp4 = desired_buffer[3];
    assign dp5 = desired_buffer[4];
    assign dp6 = desired_buffer[5];
    assign dp7 = desired_buffer[6];
```

```
    assign dp8 = desired_buffer[7];

    //output the current desired position that the vehicle should move to
    assign dx = desired_buffer[cur_buffer_read][19:10];
    assign dy = desired_buffer[cur_buffer_read][9:0];

endmodule

//////////////////////////////////////////////////////////////////////////////////
// Company: MIT
// Engineer: Jorge Simosa
//
// Create Date:   16:24:25 11/21/2010
// Design Name:
// Module Name:   display_buttons
// Project Name:     Vehicle Control Using Video Surveillance
// Target Devices: Screen Display
// Description:  This module contains the display-view buttons that are
// currently located at the top of the screen. Current functionality includes
// the video, both, and virtual map views. Also in charge of detecting mouse
// clicks within the areas of the buttons in order to switch the views.
//
//////////////////////////////////////////////////////////////////////////////////
module display_buttons(clk,hcount,vcount,mx,my,btn_click,display_sel,btn_pixels);

    parameter BTN1X = 250;              //upper left corner x of video button
    parameter BTN1Y = 15;           //upper left corner y of video button
    parameter BTN2X = 350;              //upper left corner x of both button
    parameter BTN2Y = 15;           //upper left corner y of both button
    parameter BTN3X = 450;              //upper left corner x of virtual button
    parameter BTN3Y = 15;           //upper left corner y of virtual button
    parameter WIDTH = 100;              //width of display buttons
    parameter HEIGHT = 50;              //height of display buttons
    parameter COLOR = 24'h00FFFF;   //light blue color

    input clk;                          //system clock
    input [10:0] hcount;            //hcount of pixels
    input [9:0] vcount;             //vcount of pixels
    input [10:0] mx;                //current x position of mouse
    input [9:0] my;                 //current y position of mouse
    input [2:0] btn_click;          //mouse button click
    output [1:0] display_sel;     //index of display view
    output [23:0] btn_pixels;     //output pixels of display buttons

    reg btn_delay;                      //used to determine mouse release
    reg[23:0] cpixel;               //register to store output pixel values
    reg[1:0] cdisplay = 2'b01;      //register to store index of display, default is video
    always @(posedge clk) begin
        //video button outline
```

```
        if(((hcount == BTN1X || hcount == BTN1X + WIDTH) && (vcount >= BTN1Y &&
vcount <= BTN1Y + HEIGHT))
            || ((vcount == BTN1Y || vcount == BTN1Y + HEIGHT) && (hcount >= BTN1X &&
hcount <= BTN1X + WIDTH)))
            cpixel <= COLOR;
        //both button outline
        else if(((hcount == BTN2X || hcount == BTN2X + WIDTH) && (vcount >= BTN2Y &&
vcount <= BTN2Y + HEIGHT))
            || ((vcount == BTN2Y || vcount == BTN2Y + HEIGHT) && (hcount >= BTN2X &&
hcount <= BTN2X + WIDTH)))
            cpixel <= COLOR;
        //virtual button outline
        else if(((hcount == BTN3X || hcount == BTN3X + WIDTH) && (vcount >= BTN3Y &&
vcount <= BTN3Y + HEIGHT))
                    || ((vcount == BTN3Y || vcount == BTN3Y + HEIGHT) && (hcount >=
BTN3X && hcount <= BTN3X + WIDTH)))
            cpixel <= COLOR;
        //assign all other pixels to black
        else
            cpixel <= 0;

        //used to determine mouse release
        btn_delay <= btn_click[2];

        //detect mouse click after mouse release
        if(btn_delay & ~btn_click[2]) begin
            // button click within video display button
            if((mx > BTN1X && mx < BTN1X + WIDTH) && (my > BTN1Y && my < BTN1Y +
HEIGHT)) begin
                cdisplay <= 2'b01;
            // button click within both display button
            end else if((mx > BTN2X && mx < BTN2X + WIDTH) && (my > BTN2Y && my <
BTN2Y + HEIGHT)) begin
                cdisplay <= 2'b10;
            // button click within virtual display button
            end else if((mx > BTN3X && mx < BTN3X + WIDTH) && (my > BTN3Y && my <
BTN3Y + HEIGHT)) begin
                cdisplay <= 2'b11;
            end
        //maintain current display if no click detected
        end else begin
            cdisplay <= cdisplay;
        end
    end

    assign display_sel = cdisplay;
    assign btn_pixels = cpixel;

endmodule

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

```
// Company: MIT
// Engineer: Jorge Simosa
//
// Create Date:    17:26:33 11/17/2010
// Design Name:
// Module Name:    desired_spot
// Project Name:      Vehicle Control Using Video Surveillance
// Target Devices: Display Screen
// Description:  This modules displays the desired positions that are valid
// in the buffer. The desired positions have the same shape as the mouse cursor
// except in the green color.
//
//////////////////////////////////////////////////////////////////////////////////
//////////
//////////////////////////////////////////////////////////////////////////////////
//
// desired spot: green imprint of mouse cursor
//
//////////////////////////////////////////////////////////////////////////////////
module desired_spot(vclk,x,y,hcount,vcount,pixel);

  parameter WIDTH = 20;                    //width of display area for d.p.
  parameter HEIGHT = 20;                   //height of display area for d.p.
  parameter COLOR = 24'h00FF00;            //green color
     parameter THICKNESS = 5;              //thickness of d.p.
     parameter MAP_X = 80;                 //left-upper x of map
     parameter MAP_Y = 76;                 //left-upper y of map
     parameter MAP_LENGTH = 640;               //length of map
     parameter MAP_HEIGHT = 448;               //height of map

  input vclk;                                  //system of clock
     input [9:0] x,hcount;               //x for upper left corner
  input [9:0] y,vcount;                  //y for upper right corner
  output [23:0] pixel;                       //output pixel values

     reg [9:0] x_center;                     //register for center x
     reg [9:0] y_center;                     //register for center y
  reg [23:0] cpixel;                     //register for output pixel value

  always @(posedge vclk) begin
          //check to make sure the desired positions is within the boundaries of the map
          if(x >= MAP_X && x <= MAP_X + MAP_LENGTH && y >= MAP_Y && y <= MAP_Y +
MAP_HEIGHT) begin
               //calculations to find center of desired position
               x_center <= x - WIDTH/2;
               y_center <= y - HEIGHT/2;
               //assign vertical pixel lines
               if ((hcount >= x_center+(WIDTH/2)-((THICKNESS-1)/2) && hcount <
x_center+(WIDTH/2)+((THICKNESS-1)/2)) &&
                    (vcount >= y_center && vcount < y_center + HEIGHT))
                         cpixel <= COLOR;
               //assign horizontal pixel lines
```

```
            else if ((vcount >= y_center+(HEIGHT/2)-((THICKNESS-1)/2) && vcount <
y_center+(HEIGHT/2)+((THICKNESS-1)/2)) &&
                  (hcount >= x_center && hcount < x_center + WIDTH))
                      cpixel <= COLOR;
            //assign all other pixels to black
            else cpixel <= 0;
        end
        //assign all other pixels to black
    else cpixel <= 0;
  end

     assign pixel = cpixel;
endmodule


//////////////////////////////////////////////////////////////////////////////////
///////////
// Company: MIT
// Engineer: Jorge Simosa
//
// Create Date:    14:41:55 12/02/2010
// Design Name:
// Module Name:    pixel_to_coordinates
// Project Name:      Vehicle Control Using Video Surveillance
// Target Devices: Display Screen
// Description:  This module converts the pixel value into a grid coordinate for
// the display at the bottom of the screen. e.g. the current and desired position
// of the object.
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module pixel_to_coordinates(pixel_x, pixel_y, coor_x, coor_y);
     parameter MAP_LENGTH = 640;                //80-720
     parameter MAP_HEIGHT = 448;                //44-556
     parameter MAP_X = 80;                      //upper-left x corner of map
     parameter MAP_Y = 76;                      //upper-left y corner of map
     parameter GRID_SIZE = 64;                  //grid size of map

     input [11:0] pixel_x, pixel_y;       //input pixel values
     output [3:0] coor_x, coor_y;             //output coordinate values

     reg [3:0] x;                                //register for x coordinate
     reg [3:0] y;                                //register for y coordinate

     always @ * begin
          // check to make sure that the pixel values are within the dimensions of map
          if(pixel_x <= MAP_X + MAP_LENGTH && pixel_x >= MAP_X && pixel_y <= MAP_Y +
MAP_HEIGHT && pixel_y >= MAP_Y) begin
               //calculations to find grid coordinates, based on grid size
               x = (pixel_x - MAP_X) / GRID_SIZE;
               y = (pixel_y - MAP_Y) / GRID_SIZE;
          //otherwise output (0,0)
```

```
        end else begin
            x = 0;
            y = 0;
        end
    end

    assign coor_x = x;
    assign coor_y = y;

endmodule

// File:   big_vga_hexdisp4.v
// Date:   08-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// VGA display of string of hex digits
//
// This version displays 1x1 pixels for each character pixel.
//
// 11-Nov-05 ike: now uses a font rom with only 0-9 and A-F - saves space!
// 15-Nov-05 ike: moved font rom to become a shared global module
//
// This module has three parameters:
//
// NDIGITS   = number of 4-bit hex digits to have in the display
// N_BITS    = number of bits in NDIGITS
// RGB       = color of digit output pixels

module big_vga_hexdisp4 (vclock,pix_clk,hcount,vcount,pixel,data,cx,cy,
                font_addr,font_byte);

  parameter NDIGITS = 8;          // number of 4-bit hex digits to display
  parameter N_BITS = 3;               // number of bits in NDIGITS
  parameter RGB = 24'hFFFFFF; // pixel color

  input vclock;                       // system synchronous clock
  input pix_clk;                      // video pixel clock
  input [10:0] hcount;                // horizontal index of current pixel (0..799)
  input [9:0]  vcount;            // vertical index of current pixel (0..599)
  output [23:0] pixel;                // char display's pixel
  input [NDIGITS*4-1:0] data;    // character string to display
  input [10:0] cx;                    // left-upper corner x of display area
  input [9:0]  cy;                // right-upper corner y of display area
  output [8:0] font_addr;             // address to font rom (OR together)
  input [11:0] font_byte;         // output from font rom

  // 1 line x 6 character display (12 x 24 pixel-sized characters)

  reg [N_BITS-1:0] column;  // which char in string to display
  wire [10:0]       hoff = hcount-cx;
  wire [9:0]     voff = vcount-cy;
  reg [3:0]       h;        // 0 .. 12
```

```
  reg [4:0]       v;             // 0 .. 24

  always @(posedge vclock)
   begin
         h <= ~pix_clk ? h : ( (hoff==0 | h==0) ? 11 : h-1 );
         v <= ~pix_clk ? v : voff[4:0];
         column <= ~pix_clk ? column
               : (hoff==0) ? (NDIGITS-1)
               : (h==0 ? column - 1 : column);
   end

  // look up hex value to display (from data)

  reg [3:0]  hexdig;
  integer  n;
  always @*
    for (n=0 ; n<4 ; n = n+1 )         // 4 bits per digit
      hexdig[n] <= data[column*4+n];

  // flag to determine if we're in the display area for this hexdisp
  wire dispflag = ((hcount > cx) & (vcount >= cy) & (hcount <= cx+NDIGITS*12)
             & (vcount < cy + 24));

  // look up raster row from hex digit font rom: 24 words per character
  wire [8:0] font_addr = dispflag ? (hexdig*24 + v[4:0]) : 9'b0;
  wire [11:0] font_byte;               // each word has 12 bits
  // font1224_hex_rom f(font_addr,vclock,font_byte);  // do this globally

  // generate character pixel if we're in the right h,v area
  wire [23:0] cpixel = font_byte[h[3:0]] ? RGB : 24'b0;

  // latch in pixel, and don't change except on pix_clk
  reg [23:0] pixel;
  always @(posedge vclock)
    pixel <= pix_clk ? (dispflag ? cpixel : 0) : pixel;

endmodule

//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    18:45:01 11/10/2010
// Design Name:
// Module Name:    rgb2hsv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////
module rgb2hsv(clock, reset, r, g, b, h, s, v);
	input wire clock;
	input wire reset;
	input wire [7:0] r;
	input wire [7:0] g;
	input wire [7:0] b;
	output reg [7:0] h;
	output reg [7:0] s;
	output reg [7:0] v;
	reg [7:0] my_r_delay1, my_g_delay1, my_b_delay1;
	reg [7:0] my_r_delay2, my_g_delay2, my_b_delay2;
	reg [7:0] my_r, my_g, my_b;
	reg [7:0] min, max, delta;
	reg [15:0] s_top;
	reg [15:0] s_bottom;
	reg [15:0] h_top;
	reg [15:0] h_bottom;
	wire [15:0] s_quotient;
	wire [15:0] s_remainder;
	wire s_rfd;
	wire [15:0] h_quotient;
	wire [15:0] h_remainder;
	wire h_rfd;
	reg [7:0] v_delay [19:0];
	reg [18:0] h_negative;
	reg [15:0] h_add [18:0];
	reg [4:0] i;
	// Clocks 4-18: perform all the divisions
	//the s_divider (16/16) has delay 18
	//the hue_div (16/16) has delay 18

	divider hue_div1(
	.clk(clock),
	.dividend(s_top),
	.divisor(s_bottom),
	.quotient(s_quotient),
	.remainder(s_remainder),
	.rfd(s_rfd)
	);
	divider hue_div2(
	.clk(clock),
	.dividend(h_top),
	.divisor(h_bottom),
	.quotient(h_quotient),
	.remainder(h_remainder),
```

```
        .rfd(h_rfd)
        );
        always @ (posedge clock) begin

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

            // Clock 2: compute min, max
            {my_r_delay1, my_g_delay1, my_b_delay1} <= {my_r, my_g, my_b};

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

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

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

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


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

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


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

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

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

//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    15:41:44 11/14/2010
// Design Name:
// Module Name:    delayN
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////
```

```
module delayN #(parameter NDELAY = 3, parameter SIZE = 10)(clk,in,out);
  input clk;
  input [SIZE-1:0] in;
  output [SIZE-1:0] out;

     reg [SIZE-1:0] shift_regs [NDELAY:0];
     reg [4:0] i;

     always @ (posedge clk) begin
          shift_regs[0] <= in;
          for (i = 1; i<NDELAY+1; i=i+1) begin
               shift_regs[i] <= shift_regs[i-1];
          end
     end

     assign out = shift_regs[NDELAY];
endmodule // delayN

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:20:49 11/14/2010
// Design Name:
// Module Name:    vehicle_filter
// Project Name:
// Target Devices:
// Tool versions:
// Description: Finds center of mass of the front and back markers on the vehicle. front is read,
back is yellow
//                              HSV filtering
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module vehicle_filter(clk, x, y, hsv, xy_front,xy_back);
   input clk;                          //system clock
   input [10:0] x;                //the x and y (hcount and vcount of the current pixel
   input [9:0] y;

   input [23:0] hsv;              //the hsv value of the current pixel
   output [20:0] xy_front;      //the front markers' x and y coordinates, lower 10 bits are for y, and
upper 11 are for x
      output [20:0] xy_back;        //the back markers' x and y coordinates
```

```
    reg [24:0] x_increment_front = 0;//the x incrementor for front marker
    reg [24:0] y_increment_front = 0;      //the y incrementor for front marker

    reg [24:0] count_front = 0;              //the count of pixels that pass the filtering for front
marker

    reg [24:0] x_dividend_front = 0;    //the x dividend that goes into a divider for front marker
    wire [24:0] x_q_front;                    //the x quotient out of the divider for front marker
    wire [24:0] x_r_front;                    //the x remainder out of the divider for front
marker
    wire x_rfd_front;                              //the ready signal from the divider
    reg [24:0] y_dividend_front = 0;       //the y dividend that goes into a divider for front
marker
    wire [24:0] y_q_front;                    //the y quotient out of the divider for front marker
    wire [24:0] y_r_front;                    //the y remainder out of the divider for front
marker
    wire y_rfd_front;                              //the ready signal from the divider
    reg [24:0] divisor_front = 0;            //the divisor for the front marker

    //instantiate a divider for x coordinate of the front marker
    cm_divider x_divide_front(
        .clk(clk),
        .dividend(x_dividend_front),
        .divisor(divisor_front),
        .quotient(x_q_front),
        .remainder(x_r_front),
        .rfd(x_rfd_front)
        );
    //instantiate a divider for y coordinate of the front marker
    cm_divider y_divide_front(
        .clk(clk),
        .dividend(y_dividend_front),
        .divisor(divisor_front),
        .quotient(y_q_front),
        .remainder(y_r_front),
        .rfd(y_rfd_front)
        );

    reg [24:0] x_increment_back = 0;  //the x incrementor for back marker
    reg [24:0] y_increment_back = 0;  //the y incrementor for back marker

    reg [24:0] count_back = 0;              //the count of pixels that pass the filtering for back
marker

    reg [24:0] x_dividend_back = 0;   //the x dividend that goes into a divider for back marker
    wire [24:0] x_q_back;                    //the x quotient out of the divider for back marker
    wire [24:0] x_r_back;                    //the x remainder out of the divider for back marker
    wire x_rfd_back;                              //the ready signal from the divider
    reg [24:0] y_dividend_back = 0;   //the y dividend that goes into a divider for back marker
    wire [24:0] y_q_back;                    //the y quotient out of the divider for back marker
    wire [24:0] y_r_back;                    //the y remainder out of the divider for back marker
    wire y_rfd_back;                              //the ready signal from the divider
```

```
    reg [24:0] divisor_back = 0;          //the divisor for the back marker

    //instantiate a divider for x coordinate of the back marker
    cm_divider x_divide_back(
        .clk(clk),
        .dividend(x_dividend_back),
        .divisor(divisor_back),
        .quotient(x_q_back),
        .remainder(x_r_back),
        .rfd(x_rfd_back)
        );
    //instantiate a divider for y coordinate of the back marker
    cm_divider y_divide_back(
        .clk(clk),
        .dividend(y_dividend_back),
        .divisor(divisor_back),
        .quotient(y_q_back),
        .remainder(y_r_back),
        .rfd(y_rfd_back)
        );


    always @(posedge clk) begin
        //when reached the beginning of the frame, clears the front and back incrementors and
counts
        if (x == 0 && y == 0)begin
            x_increment_front <= 0;
            y_increment_front <= 0;
            count_front <= 0;

            x_increment_back <= 0;
            y_increment_back <= 0;
            count_back <= 0;
        end
        //if the pixel is within the confined interval of the video on the screen, and it passes thru
the filtering
        //in this case this is the front marker, red. then increments the x and y onto the
incrementor and increments
        //the count
        else if (x > 80 && x < 719 && y > 76 && y < 523
            && (hsv[23:16] < 10 || hsv[23:16] > 240) && hsv[15:8] >120 && hsv[7:0] > 100) begin
                x_increment_front <= x_increment_front + x;
                y_increment_front <= y_increment_front + y;
                count_front <= count_front + 1;
        end
        //if the pixel is within the confined interval of the video on the screen, and it passes thru
the filtering
        //in this case this is the back marker, yellow. then increments the x and y onto the
incrementor and increments
        //the count
        else if (x > 80 && x < 719 && y > 76 && y < 523
         && (hsv[23:16] > 15 && hsv[23:16] < 40) && (hsv[15:8] >150 && hsv[7:0] >150)) begin
```

```
                x_increment_back <= x_increment_back + x;
                y_increment_back <= y_increment_back + y;
                count_back <= count_back + 1;
        end
        //when all pixels are processed in the confined screen shot, we pass the values to
correponding dividends and
        //divisors in each divisor. only sends values if the count exceeds 30, this deals with noise
        //this also guarantees the results will be calculated when vsync asserts
        else if (x == 0 && y == 523 ) begin
            divisor_front <= (count_front == 0)? 1: count_front;
            x_dividend_front <= (count_front > 30)? x_increment_front: 0;
            y_dividend_front <= (count_front > 30)? y_increment_front: 0;

            divisor_back <= (count_back == 0)? 1: count_back;
            x_dividend_back <= (count_back > 30)? x_increment_back: 0;
            y_dividend_back <= (count_back > 30)? y_increment_back: 0;
        end
        //implicit states, keep everything the same in other cases
        else begin
            divisor_front <= divisor_front;
            x_dividend_front <= x_dividend_front;
            y_dividend_front <= y_dividend_front;
            x_increment_front <= x_increment_front;
            y_increment_front <= y_increment_front;
            count_front <= count_front;

            divisor_back <= divisor_back;
            x_dividend_back <= x_dividend_back;
            y_dividend_back <= y_dividend_back;
            x_increment_back <= x_increment_back;
            y_increment_back <= y_increment_back;
            count_back <= count_back;
        end
    end
    //when ready signals asserts, assign corresponding values to outputs
    assign xy_front[20:10] = x_rfd_front? x_q_front[10:0] : xy_front[20:10];
    assign xy_front[9:0] = y_rfd_front? y_q_front[9:0] : xy_front[9:0];

    assign xy_back[20:10] = x_rfd_back? x_q_back[10:0] : xy_back[20:10];
    assign xy_back[9:0] = y_rfd_back? y_q_back[9:0] : xy_back[9:0];
endmodule

//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    17:44:07 11/16/2010
// Design Name:
// Module Name:    avg_buffer
// Project Name:
// Target Devices:
```

```
// Tool versions:
// Description: a size 16 circular buffer, used to average the center of masses of the last 16
frames. deals with noise
//                          This approach introduces delay in position of the vehicle, however it is
tolerable in the system
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module avg_buffer(clk, vsync, xy_front, xy_back, xy_front_avg, xy_back_avg);
   input clk;                                          //system clock
   input vsync;                                        //vsync signal
   input [20:0] xy_front;                      `       //the xy of the front marker as input
   input [20:0] xy_back;                               //the xy of the back marker as input
   output [20:0] xy_front_avg;                         //the average xy of front marker as output
   output [20:0] xy_back_avg;                          //the average xy of back marker as output

      //wires to be used for simplicity of the code
      wire [10:0] x_f = xy_front[20:10];
      wire [9:0] y_f = xy_front[9:0];

      wire [10:0] x_b = xy_back[20:10];
      wire [9:0] y_b = xy_back[9:0];

      //offset pointer in the circular buffer that points at the location in which the next value is
inserted
      reg [3:0] offset_back =0;
      reg [3:0] offset_front = 0;

      //size 16 buffers for the x,y of the front and back markers
      reg [10:0] x_buffer_front [15:0];
      reg [9:0] y_buffer_front [15:0];

      reg [10:0] x_buffer_back [15:0];
      reg [9:0] y_buffer_back [15:0];

      //incrementors used to find the average of the x and y coordinates
      wire [13:0] x_increment_front;
      wire [12:0] y_increment_front;

      wire [13:0] x_increment_back;
      wire [12:0] y_increment_back;

      //finds a new frame pulse
      reg vsync_delay;
      wire frame = vsync_delay & ~vsync;
      always @(posedge clk) vsync_delay <= vsync;
```

```
always @(posedge clk) begin
        //at each frame, sample the position of the vehicle and insert it into the buffer
        if(frame)begin
            //front marker
            if(x_f > 80 && x_f < 719 && y_f > 76 && y_f < 523) begin
                offset_front <= offset_front +1;
                x_buffer_front[offset_front] <= xy_front[20:10];
                y_buffer_front[offset_front] <= xy_front[9:0];
            end
            else offset_front <= offset_front;
            //back marker
            if(x_b > 80 && x_b < 719 && y_b > 76 && y_b < 523) begin
                offset_back <= offset_back +1;
                x_buffer_back[offset_back] <= xy_back[20:10];
                y_buffer_back[offset_back] <= xy_back[9:0];
            end
            else offset_back <= offset_back;
        end
end

//adds up all values in the circular buffers
assign  x_increment_front = x_buffer_front[0]
                                        + x_buffer_front[1]
                                        + x_buffer_front[2]
                                        + x_buffer_front[3]
                                        + x_buffer_front[4]
                                        + x_buffer_front[5]
                                        + x_buffer_front[6]
                                        + x_buffer_front[7]
                                        + x_buffer_front[8]
                                        + x_buffer_front[9]
                                        + x_buffer_front[10]
                                        + x_buffer_front[11]
                                        + x_buffer_front[12]
                                        + x_buffer_front[13]
                                        + x_buffer_front[14]
                                        + x_buffer_front[15];

assign   y_increment_front = y_buffer_front[0]
                                        + y_buffer_front[1]
                                        + y_buffer_front[2]
                                        + y_buffer_front[3]
                                        + y_buffer_front[4]
                                        + y_buffer_front[5]
                                        + y_buffer_front[6]
                                        + y_buffer_front[7]
                                        + y_buffer_front[8]
                                        + y_buffer_front[9]
                                        + y_buffer_front[10]
                                        + y_buffer_front[11]
                                        + y_buffer_front[12]
```

```
                                        + y_buffer_front[13]
                                        + y_buffer_front[14]
                                        + y_buffer_front[15];

    assign   x_increment_back = x_buffer_back[0]
                                        + x_buffer_back[1]
                                        + x_buffer_back[2]
                                        + x_buffer_back[3]
                                        + x_buffer_back[4]
                                        + x_buffer_back[5]
                                        + x_buffer_back[6]
                                        + x_buffer_back[7]
                                        + x_buffer_back[8]
                                        + x_buffer_back[9]
                                        + x_buffer_back[10]
                                        + x_buffer_back[11]
                                        + x_buffer_back[12]
                                        + x_buffer_back[13]
                                        + x_buffer_back[14]
                                        + x_buffer_back[15];

    assign   y_increment_back = y_buffer_back[0]
                                        + y_buffer_back[1]
                                        + y_buffer_back[2]
                                        + y_buffer_back[3]
                                        + y_buffer_back[4]
                                        + y_buffer_back[5]
                                        + y_buffer_back[6]
                                        + y_buffer_back[7]
                                        + y_buffer_back[8]
                                        + y_buffer_back[9]
                                        + y_buffer_back[10]
                                        + y_buffer_back[11]
                                        + y_buffer_back[12]
                                        + y_buffer_back[13]
                                        + y_buffer_back[14]
                                        + y_buffer_back[15];


  //outputs the average values, divide by 16 is right shift by 4
    assign xy_front_avg[20:10] = {x_increment_front >>4};
    assign xy_front_avg[9:0] = {y_increment_front >>4};

    assign xy_back_avg[20:10] = {x_increment_back >>4};
    assign xy_back_avg[9:0] = {y_increment_back >>4};

endmodule

module obstacle_detection(clk, hcount, vcount, r_hcount, r_vcount, frontX, frontY, backX, backY,
  hsv, hasObsCtl, hasObsDis);
    parameter DELTA = 10;              //error margin for rotations
    parameter GRID_SIZE = 64;          //grid size of map
```

```
    parameter MAP_X = 80;               //starting x for map
    parameter MAP_Y = 76;               //starting y for map
    parameter MAP_LENGTH = 639;                 //map length
    parameter MAP_HEIGHT = 447;                 //map height
    parameter COUNT_THRESHOLD =30;              //count threshold for a grid to be decided
as obstacle
    parameter OBSTACLE_THRESHOLD = 60;      //pixel threshold for that pixel to be treated
as an obstacle pixel

    input clk;                                  //system clock
    input [10:0] hcount;                    //the hcount of the current pixel coming in to
be filtered
    input [9:0] vcount;                     //the vcount of the current pixel coming in to
be filtered
    input [10:0] r_hcount;                  //the hcount of the pixel to be displayed on
screen
    input [9:0] r_vcount;                   //the vcount of the pixel to be displayed on
screen
    input [10:0] frontX, backX;             //x coordinates of the front and back markers
    input [9:0] frontY, backY;              //y coordinates of the front and back markers
    input [23:0] hsv;                       //hsv value of the current pixel to be filtered
    output [2:0] hasObsCtl;                 //outputs a three bit value of the obstacle in the
front and on the two sides of the vehicle
    output hasObsDis;                           //tells the display whether the current pixel
is an obstacle

    //a temporary map that maps a grid location to the count of pixels that pass thru the
filtering
    //this map will be cleared at each frame
    reg [11:0] map_temp [69:0];
    //the interface map with other modules, telling other modules whether there is an osbtacle
present
    //according to the hcount and vcount passed in. New values will be copied to this map from
the temporary
    //map at each frame
    reg [11:0] map [69:0];
    //the x and y coordinates of the grids to the front, left and right of the vehicle
    reg [10:0] locX_front;
    reg [9:0] locY_front;
    reg [10:0] locX_left;
    reg [9:0] locY_left;
    reg [10:0] locX_right;
    reg [9:0] locY_right;

    //index variable for a for loop to clear the temporary map
    reg [6:0] i;

    //the grid numbers of the grids to the front, left and right of the vehicle
    wire [7:0] chkLocation_front;
    wire [7:0] chkLocation_left;
    wire [7:0] chkLocation_right;
    //the grid number of the pixel for diplay
```

```
    wire [7:0] dis_location =((r_hcount - MAP_X) / GRID_SIZE) + (((r_vcount - MAP_Y) /
GRID_SIZE) * 10);

    //the grid number fo the pixel that is coming to for filtering
    wire [7:0] locationNum = ((hcount - MAP_X) / GRID_SIZE) + (((vcount - MAP_Y) /
GRID_SIZE) * 10);

    //the x and y coordinates of the center of the vehicle
    wire x_center = (frontX+backX)/2;
    wire y_center = (frontY + backY)/2;

    //write to the map temp. copy to map at each frame and clear map temp
    always @(posedge clk) begin
        //clear the temporary map at each new frame, copy to map at each frame
      if(hcount == 0 && vcount == 0) begin
         for(i = 0; i < 70; i = i+1)begin
           map[i] <= map_temp[i];
           map_temp[i] <= 0;
         end
       end
          //if the incoming pixiel is within the map, and passes the threhold test, then increment
the entry in the map by one
       //we model the vehicle as a 80 by 80 square and elminate the case when the thank confuses
itself with the obstacle
          //as best as possible
          else if(hcount >= MAP_X && hcount <= MAP_X + MAP_LENGTH && vcount >=
MAP_Y && vcount <= MAP_Y + MAP_HEIGHT &&
         hsv[15:8] < OBSTACLE_THRESHOLD && hsv[7:0] < OBSTACLE_THRESHOLD &&
               ~(hcount >= x_center -(GRID_SIZE+GRID_SIZE/4) && hcount <= x_center +
(GRID_SIZE+GRID_SIZE/4)
               && vcount >= y_center - (GRID_SIZE+GRID_SIZE/4) && vcount <= y_center +
(GRID_SIZE+GRID_SIZE/4)) ) begin
          map_temp[locationNum] <= map_temp[locationNum] + 1;
        end
      end

    //find the grid location ahead of the vehicle according to its orientation and position
    always @(posedge clk)begin
      if(frontY >= backY - DELTA && frontY <= backY + DELTA) begin //x-axis
                    locX_front <= (frontX > backX)? frontX + GRID_SIZE : frontX - GRID_SIZE;
                    locY_front <= frontY;
                       locX_left <= backX;
                       locY_left <= (frontX > backX)? backY - GRID_SIZE: backY + GRID_SIZE;
                       locX_right <= backX;
                       locY_right <= (frontX > backX)? backY + GRID_SIZE: backY - GRID_SIZE;
          end
          else if(frontX >= backX - DELTA && frontX <= backX + DELTA) begin //y-axis
                 locX_front <= frontX;
                    locY_front <= (frontY > backY)?frontY + GRID_SIZE:frontY - GRID_SIZE;
                       locX_left <= (frontY>backY)? backX + GRID_SIZE: backX - GRID_SIZE;
                       locY_left <= backY;
                       locX_right <= (frontY > backY)? backX - GRID_SIZE: backX + GRID_SIZE;
```

```
                locY_right <= backY;
            end
            else begin
              {locX_front,locY_front,locX_left,locY_left,locX_right,locY_right} <= 0;
            end
    end


    //finds the grid numbers that has to be checked
    assign chkLocation_front = ((locX_front - MAP_X) / GRID_SIZE) + (((locY_front - MAP_Y) /
GRID_SIZE) * 10);
    assign chkLocation_left = ((locX_left - MAP_X) / GRID_SIZE) + (((locY_left - MAP_Y) /
GRID_SIZE) * 10);
    assign chkLocation_right = ((locX_right- MAP_X) / GRID_SIZE) + (((locY_right - MAP_Y) /
GRID_SIZE) * 10);

    //outputs the has obstacles signal for control
  assign hasObsCtl[0] = (locX_front == 0 && locY_front == 0)? 0 :map[chkLocation_front] >
COUNT_THRESHOLD;
    assign hasObsCtl[1] = (locX_left == 0 && locY_left == 0)? 0 :map[chkLocation_left] >
COUNT_THRESHOLD;
    assign hasObsCtl[2] = (locX_right == 0 && locY_right == 0)? 0 :map[chkLocation_right] >
COUNT_THRESHOLD;

    //outputs the has obstacle signal for display
    assign hasObsDis = (r_hcount > 80 && r_hcount < 719
              && r_vcount > 76 && r_vcount < 523)? map[dis_location] >
COUNT_THRESHOLD :0;

Endmodule

module motion_control(clk,
            vsync,
            frontX,
            frontY,
            backX,
            backY,
            desiredX,
            desiredY,
                        obstacles,
            motion,
                        r_enable,
                        state,
                        dir,
                        bad);
  input clk;                    //system clock
  input vsync;                  //vsync signal
  input [10:0] frontX;          //x coordinate of the front marker
  input [9:0] frontY;           //y coordinate of the front marker
  input [10:0] backX;           //x coordinate of the back marker
  input [9:0] backY;            //y coordinate of the back marker
  input [10:0] desiredX;        //x coordinate of the desired location
```

```
   input [9:0] desiredY;          //y coordinate of the desired location
      input [2:0] obstacles;          //3 bit obstacle input from the obstacle detection module
   output [1:0] motion;          //motion output to the remote control, MSB = right belt, LSB =
left belt
      output r_enable;                  //ready pulse to tell the destination buffer that current
location reached

   wire [10:0] x_center = frontX/2 + backX/2;      //x center of the vehicle, average of the x
coordinate of front and back markers
   wire [9:0] y_center = frontY/2 + backY/2;  //y center of the vehicle, average of the y coordinate
of front and back markers


   wire obstacles_front = obstacles[0];          //if there is an obstacle in front of the vehicle
      wire obstacles_left = obstacles[1];          //if there is an obstacle to the left of the vehicle
      wire obstacles_right = obstacles[2];          //if there is an obstacle to the right of the
vehicle

      //find the posedge of a new frame
   reg vsync_delay;
   wire frame = vsync & ~vsync_delay;
   always @(posedge clk) vsync_delay<= vsync;

   reg [4:0] turn_count;          //a turn counter that gets incremented when turning
   reg [4:0] turn_count_temp;          //dynamically being set according to how much the vehicle
wants to turn
   reg [9:0] forward_count;          //a forward counter that gets incremented when going
forward
   reg [9:0] forward_count_temp;      //dynamically being set according to how much the vehicle
wants to go forward
   reg [4:0] wait_count;          //a wait counter that ets incremented when waiting

   parameter WAIT_COUNT = 15;          //Wait for 16 frames
   parameter MAX_TURN_COUNT = 20;      //maximum number of frames the vehicle can turn
when in opposite direction
   parameter xdev = 20;          //deviation from desired x coordinate in pixels
   parameter ydev = 20;          //deviation from desired y coordinate in pixles
   parameter xangle = 3;          //deviation from right angles in pixels
   parameter yangle = 3;          //deviation from right angles in pixels

   parameter IDLE = 0;                //default state, vehicle waits for a desired position from
user
   parameter WAIT = 1;                //wait state, waiting for the video processing unit to
catch up
   parameter TURN_LEFT = 2;          //turn left (face left)
   parameter TURN_RIGHT =3;          //turn right (face right)
   parameter TURN_UP = 4;          //turn up (face up)
   parameter TURN_DOWN = 5;               //turn down (face down)
   parameter FORWARD = 6;               //move forward
   parameter CHECK_DONE = 7;               //checks for appropriate action
      parameter NAVIGATE = 8;          //when an obstacle is along the axis of the desired
position, we navigate to look for open slot
```

```
    parameter FORWARD_NAVIGATE = 9; //going forward from the navigate state

    //state variables
  output reg [3:0] state = 0;
    reg [3:0] next_state;

    wire face_left = frontY >= backY-yangle && frontY <= backY+yangle && frontX < backX;
//condition when the car is facing left
    wire face_right = frontY >= backY-yangle && frontY <= backY+yangle && frontX > backX;
//condition when the car is facing right
    wire face_down = frontX >= backX-xangle && frontX <= backX+xangle && frontY > backY;
//condition when the car is facing down
    wire face_up = frontX >= backX-xangle && frontX <= backX+xangle && frontY < backY;
//condition when the car is facing up

    wire reach_x =  x_center >= desiredX - xdev && x_center <= desiredX + xdev;
//condition when the car reached x coordinate
    wire reach_y = y_center >= desiredY - ydev && y_center <= desiredY + ydev;
    //condition when the car reached y coordinate

    output reg dir=0;    //direction variable to tell which direction to go, 0 = x 1 = y
    //bad condition variable signifying the bad condition,which is the case when the car sees an
obstacle along the axis of the destination
    output reg bad=0;

    //state transition combinatorial logic
  always @ * begin
    case(state)
            /*****************************IDLE*********************************/
            //when there is no desired destination or when the car reaches the desired, then stay
in IDEL
            //otherwise, go to CHECK_DONE to check for appropriate action
      IDLE:     if( (reach_x &&  reach_y)  || (desiredX ==0 || desiredY == 0))
                next_state = IDLE;
            else next_state = CHECK_DONE;
    /*****************************WAIT*********************************/
            //When the wait counter hasn't reach the WAIT_COUNT, the car keeps waiting
            //otherwise, if the car is in bad scenario, it goes to NAVIGATE, if not, go to
CHECK_DONE
      WAIT:     begin
                              next_state = (wait_count != WAIT_COUNT)? WAIT :
                                                  (bad)? NAVIGATE: CHECK_DONE;
       end
            /*****************************TURN_LEFT*****************************/
            //When the turn counter reaches the temperory turn count, then go to WAIT state
            //otherwise, keep turning left
      TURN_LEFT:  begin
                              next_state =(turn_count == turn_count_temp)? WAIT: TURN_LEFT;
                end
            /*****************************TURN_RIGHT*****************************/
            //When the turn counter reach the temperory turn count, then go to WAIT state
            //otherwise, keep turning right
```

```
        TURN_RIGHT: begin
                                next_state = (turn_count == turn_count_temp)? WAIT:
TURN_RIGHT;
         end
                /*****************************TURN_RIGHT*******************************/
                //When the turn counter reach the temperory turn count, then go to WAIT state
                //otherwise, keep turning up
        TURN_UP:    begin
                                next_state =(turn_count == turn_count_temp)? WAIT: TURN_UP;
         end
                /*****************************TURN_RIGHT*******************************/
                //When the turn counter reach the temperory turn count, then go to WAIT state
                //otherwise, keep turning down
        TURN_DOWN:  begin
                                next_state =(turn_count == turn_count_temp)? WAIT:
TURN_DOWN;
        end
                /*****************************FORWARD*********************************/
                //if there is an obstacle in front of the car, the car checks if it is the bad case in which
it is along the axis of
                //destination and is trying to go there, if so, the car go to the NAVIGATE state, go
to CHECK DONE otherwise
                //if there is no obstacle in front, the car checks if forward counter reached the
temperory count, if so, go to WAIT
                //if the direction is off of 90 degrees, then the car goes to CHECK_DONE to adjust,
otherwise keep going FORWARD
        FORWARD:    begin
                                if(obstacles_front) next_state = ((x_center >= desiredX - 40 &&
x_center <= desiredX + 40&& (face_up || face_down))
                                                                        || (y_center >=
desiredY - 40 && y_center <= desiredY + 40 && (face_left || face_right)))?NAVIGATE:
CHECK_DONE;
                                else next_state = (forward_count == forward_count_temp)? WAIT:
                                                        (~(face_left|| face_right || face_up ||
face_down))? CHECK_DONE:
                                                        FORWARD;
                            end
        /*************************CHECK_DONE*********************************/
                //if the car is at the destination, then go to IDLE
                //if the car wants to go in the y direction, if already facing that direction, then go
FORWARD, otherwise turn to
                //corresponding directions
                //if the car wants to go in the x direction, same thing happens; if already facing the
right direction, go FORWARD,
                //otherwise turn accordingly
        CHECK_DONE: begin
                                    if(reach_x && reach_y) next_state = IDLE;
                                    else if (dir == 1) begin
                                            if(y_center < desiredY) begin
                                                if(face_down) next_state = FORWARD;

                                                else                next_state = TURN_DOWN;
```

```
                                    end
                                    else begin
                                        if(face_up)next_state = FORWARD;
                                        else      next_state = TURN_UP;
                                    end
                                end
                                else if (dir == 0)begin
                                    if(x_center < desiredX) begin
                                        if(face_right)     next_state = FORWARD;
                                        else                    next_state = TURN_RIGHT;
                                    end
                                    else begin
                                        if(face_left)  next_state = FORWARD;
                                        else             next_state = TURN_LEFT;
                                    end
                                end
                            end
                /**********************************NAVIGATE*******************************/
                //the navigate state turns the car to opposite directin and if facing that direction,
then start moving forward
                //right now it turns simply to the direction with more room to navigate, ie if the
car is at the top half of the field
                //navigate to the lower half
                NAVIGATE: if (dir == 0)begin
                                        if(face_left || face_right) next_state =
FORWARD_NAVIGATE;
                                        else      next_state = (frontX >= backX-10 &&
frontX <= backX+10)?
                                                                ((x_center >= 400)?
TURN_LEFT:TURN_RIGHT):
                                                                ((frontX < backX)?
TURN_LEFT: TURN_RIGHT);
                                end
                                else begin
                                        if(face_up || face_down) next_state =
FORWARD_NAVIGATE;
                                        else next_state = (frontY >= backY-10 && frontY <=
backY+10)?
                                                                ((y_center >=
300)?TURN_UP: TURN_DOWN):
                                                                ((frontY < backY)?
TURN_UP:TURN_DOWN);
                                end

    /**********************************FORWARD_NAVIGATE*****************************/
                //when moving forward while navigating, car constantly checks for obstacles
onn the sides, the scenarios are as following:
                //1. facing left, and want to go up, if no obstacles on the right, then go to
CHECK_DONE
                //2. facing left, and want to go down, if no obstacles on the left, then go to
CHECK_DONE
```

```
                //3. facing right, and want to go up, if no obstacles on the left, then go to
CHECK_DONE
                //4. facing right, and want to go down, if no obstacles on the right, then go to
CHECK_DONE
                //5. facing up, and want to go left, if no obstacles on the left, then go to
CHECK_DONE
                //6. facing up, and want to go right, if no obstacles on the right, then go to
CHECK_DONE
                //7. facing down, and want to go left, if no obstacles on the right, then go to
CHECK_DONE
                //8. facing down, and want to go right, if no obstacles on the left, then go to
CHECK_DONE
                //if the car is off 90 degrees directions, go the NAVIGATE to adjust, otherwise,
keep moving forward
                FORWARD_NAVIGATE: next_state = ( (face_left && desiredY < y_center &&
~obstacles_right)||
                                                 (face_left  && desiredY >
y_center && ~obstacles_left)||
                                                 (face_right && desiredY <
y_center && ~obstacles_left)||
                                                 (face_right && desiredY >
y_center && ~obstacles_right)||
                                                 (face_up && desiredX <
x_center && ~obstacles_left)||
                                                 (face_up && desiredX >
x_center && ~obstacles_right)||
                                                 (face_down && desiredX <
x_center && ~obstacles_right)||
                                                 (face_down && desiredX >
x_center && ~obstacles_left))? CHECK_DONE:
                                                 (~(face_left || face_right ||
face_up || face_down))?NAVIGATE: FORWARD_NAVIGATE;
                //default state is IDLE
                default: next_state = IDLE;

    endcase
  end


    //value assignments
  always @(posedge clk) begin
   state <= next_state;   //assign states

        //when going from CHECK_DONE to FORWARD, set the appropriate forward count
according to directions and how far away
        //the car is from the destination
        forward_count_temp <= (state == CHECK_DONE && next_state == FORWARD)?
                              (dir == 0)? (x_center > desiredX)? (x_center - desiredX)>>1:
(desiredX - x_center)>>1:
                                          (y_center > desiredY)? (y_center -
desiredY)>>1: (desiredY - y_center)>>1:
                                          forward_count_temp;
```

```
        //when going from CHECK_DONE to TURN states, set the appropriate turn count
according to how much deviate the car is
        //from the desired 90 degree angles
        turn_count_temp <= ((state == CHECK_DONE || state == NAVIGATE) &&
                                    (next_state == TURN_LEFT || next_state == TURN_RIGHT ||
next_state == TURN_UP || next_state == TURN_DOWN))?
                                    (dir ==0)? ( ( (next_state == TURN_RIGHT && frontX >=
backX) ||
                                                    (next_state == TURN_LEFT && frontX <=
backX)  )?
                                                  (frontY > backY)? (frontY-backY)>>1: (backY-
frontY)>>1:MAX_TURN_COUNT)
                                                : //dir == 1
                                                ( ( (next_state == TURN_UP && frontY <=
backY) ||
                                                    (next_state == TURN_DOWN && frontY
>= backY))?
                                                    (frontX > backX)? (frontX-backX)>>1:
(backX- frontX)>>1:MAX_TURN_COUNT):
                                    turn_count_temp;
        //increment wait count at each frame when in the WAIT state
    wait_count <=    (frame)?
                            (state!= WAIT )? 0:
                            (wait_count == WAIT_COUNT)? 0: wait_count+1
                            :wait_count;
        //increment turn count at each frame when in TURN states
    turn_count <=    (frame)?
                            (state == CHECK_DONE || state == WAIT || state ==
FORWARD)? 0:
                            (turn_count == turn_count_temp)?0: turn_count +1
                            : turn_count;
        //increment forward count at each frame when in FORWARD state
    forward_count <=     (frame)?
                                (state != FORWARD)? 0:
                                (forward_count == forward_count_temp)? 0: forward_count+1
                                : forward_count;
        //default direction is the x direction
        //the car changes direction when moving forward and meets an obstacle
        //or when the car reaches either x or y of the desired position
        //or when the the car transition back to the normal case from the bad case
        dir <= (state == IDLE)? 0:
                (    (state == FORWARD && obstacles_front) ||
                     next_state == CHECK_DONE && reach_x && (face_right || face_left)||
                     next_state == CHECK_DONE && reach_y && (face_up || face_down) ||
                     (state == FORWARD_NAVIGATE && next_state == CHECK_DONE))? ~dir
: dir;
        //the car is in the bad scenario when it is going into NAVIGATE state
        //not bad when it goes into CHECK_DONE
        bad <= (next_state == NAVIGATE)? 1:
                (next_state == CHECK_DONE)? 0: bad;

   end
```

```
    //assign motion according to current states
  assign motion = (state == TURN_LEFT)? ((frontY > backY)? 2'b01: 2'b10):
            (state == TURN_RIGHT)? ((frontY < backY)? 2'b01: 2'b10):
            (state == TURN_UP)? ((frontX > backX)? 2'b10: 2'b01):
            (state == TURN_DOWN)? ((frontX < backX)? 2'b10: 2'b01):
            (state == FORWARD || state == FORWARD_NAVIGATE)? 2'b11: 2'b00;
    //a desination is reached when the car moves into IDLE state from CHECK_DONE; this
ensures a pulse of one clock cycle long
    assign r_enable = (state == CHECK_DONE) && (next_state == IDLE);

endmodule

//
// File:   zbt_6111_sample.v
// Date:   26-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Sample code for the MIT 6.111 labkit demonstrating use of the ZBT
// memories for video display.  Video input from the NTSC digitizer is
// displayed within an XGA 1024x768 window.  One ZBT memory (ram0) is used
// as the video frame buffer, with 8 bits used per pixel (black & white).
//
// Since the ZBT is read once for every four pixels, this frees up time for
// data to be stored to the ZBT during other pixel times.  The NTSC decoder
// runs at 27 MHz, whereas the XGA runs at 65 MHz, so we synchronize
// signals between the two (see ntsc2zbt.v) and let the NTSC data be
// stored to ZBT memory whenever it is available, during cycles when
// pixel reads are not being performed.
//
// We use a very simple ZBT interface, which does not involve any clock
// generation or hiding of the pipelining.  See zbt_6111.v for more info.
//
// switch[7] selects between display of NTSC video and test bars
// switch[6] is used for testing the NTSC decoder
// switch[1] selects between test bar periods; these are stored to ZBT
//         during blanking periods
// switch[0] selects vertical test bars (hardwired; not stored in ZBT)
//
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09



//////////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
```

```
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////
////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////
////////
//
// Complete change history (including bug fixes)
//
// 2009-May-11: Fixed memory management bug by 8 clock cycle forecast.
//              Changed resolution to 800 * 600.
//              Reduced clock speed to 40MHz.
//              Disconnected zbt_6111's ram_clk signal.
//              Added ramclock to control RAM.
//              Added notes about ram1 default values.
//              Commented out clock_feedback_out assignment.
//              Removed delayN modules because ZBT's latency has no more effect.
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
```

```
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
////////////////////////////////////////////////////////////////////////
////////

module zbt_6111_sample(beep, audio_reset_b,
                       ac97_sdata_out, ac97_sdata_in, ac97_synch,
                       ac97_bit_clock,

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

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

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

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

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

                       clock_feedback_out, clock_feedback_in,

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

                       rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

                       mouse_clock, mouse_data, keyboard_clock, keyboard_data,

                       clock_27mhz, clock1, clock2,

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

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

        switch,

        led,

        user1, user2, user3, user4,

        daughtercard,

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

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

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

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

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

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

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

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

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
```

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

  output rs232_txd, rs232_rts;
  input  rs232_rxd, rs232_cts;

  inout  mouse_clock, mouse_data;
    input keyboard_clock, keyboard_data;

  input  clock_27mhz, clock1, clock2;

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

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

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

  inout [43:0] daughtercard;

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

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


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

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

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

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

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

  // SRAMs

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

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

/* enable RAM pins */

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

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

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

  //These values has to be set to 0 like ram0 if ram1 is used.
  assign ram1_cen_b = 1'b1;
```

```
  assign ram1_ce_b = 1'b1;
  assign ram1_oe_b = 1'b1;
  assign ram1_we_b = 1'b1;
  assign ram1_bwe_b = 4'hF;

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

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

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

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

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

  // Buttons, Switches, and Individual LEDs
  //lab3
  // button0, button1, button2, button3, button_enter, button_right,
  // button_left, button_down, button_up, and switches are inputs

  // User I/Os

  assign user2 = 32'hZ;
  assign user3 = 32'hZ;
  assign user4 = 32'hZ;

  // Daughtercard Connectors
  assign daughtercard = 44'hZ;
```

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

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


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

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

  wire clk = clock_40mhz;
     wire locked;
     assign clock_feedback_out = 0;

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


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

  // ENTER button is user reset
  wire reset,user_reset;
```

```
debounce db1(power_on_reset, clk, ~button_enter, user_reset);
assign reset = user_reset | power_on_reset;

// display module for debugging

reg [63:0] dispdata;
display_16hex hexdisp1(reset, clk, dispdata,
                disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out);

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

// wire up to ZBT ram

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

zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
           vram_write_data, vram_read_data,
           ram0_clk,                  //to get good timing, don't connect ram_clk to zbt_6111
           ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

// generate pixel value from reading ZBT memory
wire [23:0]      vr_pixel;
wire [18:0]      vram_addr1;

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

// ADV7185 NTSC decoder interface code
// adv7185 initialization module
adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz),
               .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
               .tv_in_i2c_clock(tv_in_i2c_clock),
               .tv_in_i2c_data(tv_in_i2c_data));

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

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

// code to write NTSC data to video memory
```

```
  wire [18:0] ntsc_addr;
  wire [35:0] ntsc_data;
  wire        ntsc_we;
  ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, ycrcb,
              ntsc_addr, ntsc_data, ntsc_we, switch[6]);

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

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

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

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

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

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

  // select output pixel data
  reg [23:0]  pixel;
  reg     b,hs,vs;
     reg [10:0] cursor_x = 0;
     reg [9:0] cursor_y =0;
     wire [20:0] xy_front;
     wire [20:0] xy_back;
     wire [20:0] xy_front_avg;
     wire [20:0] xy_back_avg;
     wire [23:0] hsv;

     // mouse
  wire [11:0] mx,my;
  wire [2:0]  btn_click;
  ps2_mouse_xy m1(clk, reset, mouse_clock, mouse_data, mx, my, btn_click);
  defparam    m1.MAX_X = 800-10;
  defparam    m1.MAX_Y = 600-10;

     // virtual grid map
     wire [23:0] vgpix;
     grid_map v_grid(clk, hcount, vcount,vgpix);
```

```
    // mouse cursor display
    wire [23:0] mcpix;
  mouse_cursor_shape ball(clk, mx[10:0],my[9:0],hcount,vcount,mcpix);

    // vehicle blob display
    wire [23:0] vhpix;
    vehicle_blob my_tank(clk, (xy_front_avg[20:10]/2 + xy_back_avg[20:10]/2),
                                        (xy_front_avg[9:0]/2 + xy_back_avg[9:0]/2), hcount,
vcount, vhpix);

    // mouse tracing
  wire [11:0] dx;
    wire [11:0] dy;
    wire d_enable;
    wire r_enable;
    wire [19:0] dp1;
    wire [19:0] dp2;
    wire [19:0] dp3;
    wire [19:0] dp4;
    wire [19:0] dp5;
    wire [19:0] dp6;
    wire [19:0] dp7;
    wire [19:0] dp8;
    adv_tracing m_trace(clk, mx, my, btn_click, r_enable, dx, dy, dp1, dp2, dp3, dp4, dp5, dp6,
dp7, dp8);
    defparam m_trace.BUFFER_SIZE = 8;

    // display buttons: changes the view
    wire [1:0] display_sel;
    wire [23:0] btn_pix;
    display_buttons dbuttons(clk,hcount,vcount,mx[9:0],my[9:0],btn_click,display_sel,btn_pix);

    // character display module
  wire [39:0] vidbtns = "VIDEO";
  wire [23:0]  vidbtnpix;
  char_string_display vidlbl(clk,hcount,vcount,vidbtnpix,vidbtns,11'd260,10'd28);
    defparam vidlbl.NCHAR = 5;
    defparam vidlbl.NCHAR_BITS = 4;
  wire [31:0] bothbtns = "BOTH";
  wire [23:0]  bothpix;
  char_string_display bothlbl(clk,hcount,vcount,bothpix,bothbtns,11'd370,10'd28);
    defparam bothlbl.NCHAR = 4;
    defparam bothlbl.NCHAR_BITS = 4;
  wire [23:0] virbtns = "MAP";
  wire [23:0]  virpix;
  char_string_display virlbl(clk,hcount,vcount,virpix,virbtns,11'd475,10'd28);
    defparam virlbl.NCHAR = 3;
    defparam virlbl.NCHAR_BITS = 4;

    // desired position display
    wire [23:0] dp1pix;
    desired_spot my_spot1(clk, dp1[19:10], dp1[9:0], hcount, vcount, dp1pix);
```

```
    wire [23:0] dp2pix;
    desired_spot my_spot2(clk, dp2[19:10], dp2[9:0], hcount, vcount, dp2pix);
    wire [23:0] dp3pix;
    desired_spot my_spot3(clk, dp3[19:10], dp3[9:0], hcount, vcount, dp3pix);
    wire [23:0] dp4pix;
    desired_spot my_spot4(clk, dp4[19:10], dp4[9:0], hcount, vcount, dp4pix);
    wire [23:0] dp5pix;
    desired_spot my_spot5(clk, dp5[19:10], dp5[9:0], hcount, vcount, dp5pix);
    wire [23:0] dp6pix;
    desired_spot my_spot6(clk, dp6[19:10], dp6[9:0], hcount, vcount, dp6pix);
    wire [23:0] dp7pix;
    desired_spot my_spot7(clk, dp7[19:10], dp7[9:0], hcount, vcount, dp7pix);
    wire [23:0] dp8pix;
    desired_spot my_spot8(clk, dp8[19:10], dp8[9:0], hcount, vcount, dp8pix);

  // global shared font rom
  wire [8:0] font_addr;          // we'll OR together all addresses
  wire [11:0] font_byte;
  font1224_hex_rom fr(font_addr,clk,font_byte);

     // pixel to coordinate display for current vehicle position
   wire [3:0] cur_x;
   wire [3:0] cur_y;
   pixel_to_coordinates cur_coor(xy_front_avg[20:10]/2 + xy_back_avg[20:10]/2,
                          xy_front_avg[9:0]/2 + xy_back_avg[9:0]/2,
                                                cur_x,cur_y);

   // vga hex digits display of current vehicle position
  wire [23:0]  vcxpix;
  wire [8:0] font_addr1;
  wire [10:0] hx_curx = 11'd246;
  wire [9:0]  hy_curx = 10'd545;
  big_vga_hexdisp4
current_vehiclex(clk,~clk,hcount,vcount,vcxpix,cur_x,hx_curx,hy_curx,font_addr1,font_byte);
  defparam    current_vehiclex.NDIGITS = 1;
  defparam    current_vehiclex.N_BITS = 4;
   wire [23:0]  vcypix;
      wire [8:0] font_addr2;
  wire [10:0] hx_cury = 11'd270;
  wire [9:0]  hy_cury = 10'd545;
  big_vga_hexdisp4 current_vehicley(clk,~clk,hcount,vcount,vcypix,cur_y,hx_cury,hy_cury,
font_addr2,font_byte);
  defparam    current_vehicley.NDIGITS = 1;
  defparam    current_vehicley.N_BITS = 4;

   wire [63:0] curbtns = "ACTUAL:";
  wire [23:0]  curpix;
  char_string_display curlbl(clk,hcount,vcount,curpix,curbtns,11'd98,10'd545);
   defparam curlbl.NCHAR = 7;
   defparam curlbl.NCHAR_BITS = 4;
   wire [63:0] curparen = "( , )";
  wire [23:0]  curparenpix;
```

```
char_string_display curcoor(clk,hcount,vcount,curparenpix,curparen,11'd224,10'd545);
 defparam curcoor.NCHAR = 5;
 defparam curcoor.NCHAR_BITS = 4;

 // pixel to coordinate display for current vehicle position
 wire [3:0] des_x;
 wire [3:0] des_y;
 pixel_to_coordinates des_coor(dx,dy,des_x,des_y);

 // vga hex digits display of desired position
wire [23:0]  vdxpix;
wire [8:0]  font_addr3;
wire [10:0] hx_desx = 11'd640;
wire [9:0]  hy_desx = 10'd545;
big_vga_hexdisp4
   desired_positionx(clk,~clk,hcount,vcount,vdxpix,des_x,hx_desx,hy_desx,
       font_addr3,font_byte);
defparam    desired_positionx.NDIGITS = 1;
defparam    desired_positionx.N_BITS = 4;
wire [23:0]  vdypix;
wire [8:0]  font_addr4;
wire [10:0] hx_desy = 11'd668;
wire [9:0]  hy_desy = 10'd545;
big_vga_hexdisp4 desired_positiony(clk,~clk,hcount,vcount,vdypix,des_y,hx_desy,hy_desy,
       font_addr4,font_byte);
defparam    desired_positiony.NDIGITS = 1;
defparam    desired_positiony.N_BITS = 4;

 wire [63:0] desbtns = "DESIRED:";
 wire [23:0]  despix;
 char_string_display deslbl(clk,hcount,vcount,despix,desbtns,11'd460,10'd545);
 defparam deslbl.NCHAR = 8;
 defparam deslbl.NCHAR_BITS = 4;
 wire [63:0] desparen = "( , )";
    wire [23:0]  desparenpix;
char_string_display descoor(clk,hcount,vcount,desparenpix,desparen,11'd618,10'd545);
 defparam descoor.NCHAR = 5;
 defparam descoor.NCHAR_BITS = 4;

 assign       font_addr = font_addr1 | font_addr2 | font_addr3 |font_addr4;

   //  display_sel selects which video generator to use:
//  switch[0] 01: 1 pixel outline of active video area (adjust screen controls)
//  10: real-time video data [vpix]
   //   11: real-time video overlayed with virtual map [vpix + vgpix]
   //   01: virtual map [vgpix]




   wire hasObsDis;
```

```
  always @(posedge clk) begin
    hs <= hsync;
    vs <= vsync;
    b <= blank;
    if (switch[0] == 1'b1) begin
            // 1 pixel outline of visible area (white)
            pixel <= (hcount==0 || hcount==699 || vcount==0 || vcount==599) ? 24'hFFFFFF :
0;
        end
        else if(display_sel == 2'b01) begin                 //video
            pixel <= (mcpix!=0)?    mcpix:                  //mouse
                        (btn_pix!=0)?   btn_pix:                    //buttons
                        (curpix!=0)?curpix:                 //actual label
                        (curparenpix!=0)? curparenpix:
                        (vcxpix !=0)?   vcxpix:                 //current cursor position
                        (vcypix!=0)?    vcypix:
                        (despix!=0)?    despix:                 //desired label
                        (desparenpix!=0)? desparenpix:
                        (vdxpix !=0)?   vdxpix:                 //desired vehicle position
                        (vdypix!=0)?    vdypix:
                        (vidbtnpix!=0)?vidbtnpix:               //video button label
                        (bothpix!=0)?   bothpix:                    //both button label
                        (virpix!=0)? virpix:                //virtual button label
                        (dp1pix !=0)?   dp1pix:                 //desired spot 1
                        (dp2pix !=0)?   dp2pix:                 //desired spot 2
                        (dp3pix !=0)?   dp3pix:                 //desired spot 3
                        (dp4pix !=0)?   dp4pix:                 //desired spot 4
                        (dp5pix !=0)?   dp5pix:                 //desired spot 5
                        (dp6pix !=0)?   dp6pix:                 //desired spot 6
                        (dp7pix !=0)?   dp7pix:                 //desired spot 7
                        (dp8pix !=0)?   dp8pix:                 //desired spot 8
                        (hcount<80 || hcount>719 || vcount<76 || vcount>523) ? 0 :
                        (hcount == xy_front_avg[20:10] || vcount == xy_front_avg[9:0])?
24'hFF0000:
                        (hcount == xy_back_avg[20:10] || vcount == xy_back_avg[9:0])?
24'hFFFF00:
                        vr_pixel;                                   //video

        end
        else if(display_sel == 2'b10) begin                 //both
            pixel <= (mcpix!=0)?    mcpix:                  //mouse
                        (btn_pix!=0)?   btn_pix:                    //buttons
                        (curpix!=0)?curpix:                 //actual label
                        (curparenpix!=0)? curparenpix:
                        (vcxpix !=0)?   vcxpix:                 //current cursor position
                        (vcypix!=0)?    vcypix:
                        (despix!=0)?    despix:                 //desired label
                        (desparenpix!=0)? desparenpix:
                        (vdxpix !=0)?   vdxpix:                 //desired vehicle position
                        (vdypix!=0)?    vdypix:
                        (vidbtnpix!=0)?vidbtnpix:               //video button label
                        (bothpix!=0)?   bothpix:                    //both button label
```

```
                        (virpix!=0)? virpix:                //virtual button label
                        (dp1pix !=0)?   dp1pix:             //desired spot 1
                        (dp2pix !=0)?   dp2pix:             //desired spot 2
                        (dp3pix !=0)?   dp3pix:             //desired spot 3
                        (dp4pix !=0)?   dp4pix:             //desired spot 4
                        (dp5pix !=0)?   dp5pix:             //desired spot 5
                        (dp6pix !=0)?   dp6pix:             //desired spot 6
                        (dp7pix !=0)?   dp7pix:             //desired spot 7
                        (dp8pix !=0)?   dp8pix:             //desired spot 8
                        (vgpix !=0)? vgpix:                 //grid map
                        (hcount<80 || hcount>719 || vcount<76 || vcount>523) ? 0 :
                        (hcount == xy_front_avg[20:10] || vcount == xy_front_avg[9:0])?
24'hFF0000:
                        (hcount == xy_back_avg[20:10] || vcount == xy_back_avg[9:0])?
24'hFFFF00:
                        (hasObsDis)?    24'hAAAAAA:         //obstacle
                        vr_pixel;//video
    end
        else begin
            pixel <= (mcpix!=0)?    mcpix:                  //mouse
                        (btn_pix!=0)?   btn_pix:                //buttons
                        (curpix!=0)?curpix:                 //actual label
                        (curparenpix!=0)? curparenpix:
                        (vcxpix !=0)?   vcxpix:             //current cursor position
                        (vcypix!=0)?    vcypix:
                        (despix!=0)?    despix:             //desired label
                        (desparenpix!=0)? desparenpix:
                        (vdxpix !=0)?   vdxpix:             //desired vehicle position
                        (vdypix!=0)?    vdypix:
                        (vidbtnpix!=0)?vidbtnpix:           //video button label
                        (bothpix!=0)?   bothpix:                //both button label
                        (virpix!=0)? virpix:                //virtual button label
                        (vhpix !=0)?    vhpix:              //vehicle blob
                        (dp1pix !=0)?   dp1pix:             //desired spot 1
                        (dp2pix !=0)?   dp2pix:             //desired spot 2
                        (dp3pix !=0)?   dp3pix:             //desired spot 3
                        (dp4pix !=0)?   dp4pix:             //desired spot 4
                        (dp5pix !=0)?   dp5pix:             //desired spot 5
                        (dp6pix !=0)?   dp6pix:             //desired spot 6
                        (dp7pix !=0)?   dp7pix:             //desired spot 7
                        (dp8pix !=0)?   dp8pix:             //desired spot 8
                        (hasObsDis)?    24'hAAAAAA:         //obstacle
                        vgpix;                              //grid map
    end
end



// VGA Output.  In order to meet the setup and hold times of the
// AD7125, we send it ~clk.
assign vga_out_red = pixel[23:16];
assign vga_out_green = pixel[15:8];
```

```
  assign vga_out_blue = pixel[7:0];
  assign vga_out_sync_b = 1'b1;    // not used
  assign vga_out_pixel_clock = ~clk;
  assign vga_out_blank_b = ~b;
  assign vga_out_hsync = hs;
  assign vga_out_vsync = vs;

     //instantiate rgb to hsv converter
     rgb2hsv rgb2hsv (.clock(clk), .reset(reset),
                                  .r(vr_pixel[23:16]),
                                  .g(vr_pixel[15:8]),
                                  .b(vr_pixel[7:0]),
                                  .h(hsv[23:16]),
                                  .s(hsv[15:8]),
                                  .v(hsv[7:0]));

     wire [10:0] hcount_filter;
     wire [9:0] vcount_filter;

     //delay the hcount and vcount by 22 clock cycles to match the rgb2hsv delay
     delayN #(.NDELAY(22),.SIZE(11)) delayx(.clk(clk), .in(hcount), .out(hcount_filter));
     delayN #(.NDELAY(22),.SIZE(10)) delayy(.clk(clk), .in(vcount), .out(vcount_filter));

  //instantiate the vehicle filter module to find the instantaneous position of the car
     vehicle_filter vf(.clk(clk),
                                  .x(hcount_filter),
                                  .y(vcount_filter),
                                  .hsv(hsv),
                                  .xy_front(xy_front),
                                  .xy_back(xy_back));
     //pass the instananeous position of the car to the average buffer to get the averaged value of
the position
     avg_buffer ab(.clk(clk),
                               .vsync(vsync),
                               .xy_front(xy_front),
                               .xy_back(xy_back),
                               .xy_front_avg(xy_front_avg),
                               .xy_back_avg(xy_back_avg));


     wire [2:0] hasObsCtl;
     wire  [3:0] state;
     wire bad;
   wire dir;
     //pass some variables to led output for debugging
     assign led[4:0] = {~bad, ~dir,~hasObsCtl[2], ~hasObsCtl[1],~hasObsCtl[0]};
     assign led[5] = (state == 9)? 1: led[5];
     assign led[7:6] = 2'b11;
     //instantiate obstacle detection module to find obstacles
     obstacle_detection obsdetec(.clk(clk),
                                                  .hcount(hcount_filter),
                                                  .vcount(vcount_filter),
```

```
                                        .r_hcount(hcount),
                                        .r_vcount(vcount),
                                        .frontX(xy_front_avg[20:10]),
                                        .frontY(xy_front_avg[9:0]),
                                        .backX(xy_back_avg[20:10]),
                                        .backY(xy_back_avg[9:0]),
                                        .hsv(hsv),
                                        .hasObsCtl(hasObsCtl),
                                        .hasObsDis(hasObsDis));

    //instantiate the motion control module and outputs corresponding motions
    motion_control mc( .clk(clk),
                                    .vsync(vsync),
                                    .frontX(xy_front_avg[20:10]),
                                    .frontY(xy_front_avg[9:0]),
                                    .backX(xy_back_avg[20:10]),
                                    .backY(xy_back_avg[9:0]),
                                    .desiredX(dx[10:0]),
                                    .desiredY(dy[9:0]),
                                    .obstacles(hasObsCtl),
                                    .motion(user1[31:30]),
                                    .r_enable(r_enable),
                                    .state(state),
                                    .dir(dir),
                                    .bad(bad));

  always @(posedge clk)begin
    // dispdata <= {vram_read_data,9'b0,vram_addr};
    if(vsync & ~vs) begin
            if(~button_up) cursor_y <= (cursor_y == 0)?0:cursor_y-1;
            else if(~button_down) cursor_y <= (cursor_y == 599)? 599: cursor_y+1;

            if(~button_left) cursor_x <= (cursor_x == 0)? 0 : cursor_x-1;
            else if(~button_right)cursor_x<= (cursor_x == 799)? 799: cursor_x+1;
      end
      else begin
              cursor_x <= cursor_x;
              cursor_y<= cursor_y;
      end
      //display hsv value and state for debugging purposes
      dispdata <= (hcount == cursor_x && vcount == cursor_y)?{state, 2'b00, user1[31:30],
32'd0,hsv}: dispdata;
    end
endmodule



//////////////////////////////////////////////////////////////////////
//////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
```

```
// We take care of that by latching the data at an appropriate time.
//
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.
//
// Bug due to memory management will be fixed. The bug happens because
// memory is called based on current hcount & vcount, which will actually
// shows up 2 cycle in the future. Not to mention that these incoming data
// are latched for 2 cycles before they are used. Also remember that the
// ntsc2zbt's addressing protocol has been fixed.

// The original bug:
// -. At (hcount, vcount) = (100, 201) data at memory address(0,100,49)
//    arrives at vram_read_data, latch it to vr_data_latched.
// -. At (hcount, vcount) = (100, 203) data at memory address(0,100,49)
//    is latched to last_vr_data to be used for display.
// -. Remember that memory address(0,100,49) contains camera data
//    pixel(100,192) - pixel(100,195).
// -. At (hcount, vcount) = (100, 204) camera pixel data(100,192) is shown.
// -. At (hcount, vcount) = (100, 205) camera pixel data(100,193) is shown.
// -. At (hcount, vcount) = (100, 206) camera pixel data(100,194) is shown.
// -. At (hcount, vcount) = (100, 207) camera pixel data(100,195) is shown.
//
// Unfortunately this means that at (hcount == 0) to (hcount == 11) data from
// the right side of the camera is shown instead (including possible sync signals).

// To fix this, two corrections has been made:
// -. Fix addressing protocol in ntsc_to_zbt module.
// -. Forecast hcount & vcount 8 clock cycles ahead and use that
//    instead to call data from ZBT.




//////////////////////////////////////////////////////////////////////////////
//////
// parameterized delay line


//////////////////////////////////////////////////////////////////////////////
/////
// ramclock module

//////////////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
```

```
////////////////////////////////////////////////////////////////////////
////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
////////////////////////////////////////////////////////////////////////
////////

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

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

  wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;


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

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

     assign ref_clk = ref_clock;

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

  DCM int_dcm (.CLKFB(fpga_clock),
          .CLKIN(ref_clk),
          .RST(dcm_reset),
          .CLK0(fpga_clk),
          .LOCKED(lock1));
```

```
  // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
  // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
  // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
  // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
  // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
  // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
  // synthesis attribute PHASE_SHIFT of int_dcm is 0

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

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

  DCM ext_dcm (.CLKFB(fb_clk),
               .CLKIN(ref_clk),
               .RST(dcm_reset),
               .CLK0(ram_clk),
               .LOCKED(lock2));
  // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
  // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
  // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
  // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
  // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
  // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
  // synthesis attribute PHASE_SHIFT of ext_dcm is 0

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


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

  assign locked = lock1 && lock2;

endmodule

//
// File:   cstringdisp.v
// Date:   24-Oct-05
// Author: I. Chuang, C. Terman
//
// Display an ASCII encoded character string in a video window at some
// specified x,y pixel location.
//
// INPUTS:
//
//   vclock      - video pixel clock
```

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


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

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

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

  input vclock;                          // system clock
  input [10:0] hcount;                   // horizontal index of current pixel (0..799)
  input [9:0]      vcount;             // vertical index of current pixel (0..599)
  output [23:0] pixel;                   // char display's pixel
  input [NCHAR*8-1:0] cstring;  // character string to display
  input [10:0] cx;
  input [9:0]  cy;

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

  wire [10:0] hoff = hcount-1-cx;
  wire [9:0]  voff = vcount-cy;
  wire [NCHAR_BITS-1:0] column = NCHAR-1-hoff[NCHAR_BITS-1+4:4];  // < NCHAR
```

```
  wire [2:0]  h = hoff[3:1];            // 0 .. 7
  wire [3:0]  v = voff[4:1];                          // 0 .. 11

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

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

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

endmodule
```