Virtual Pool

Abdulhamid Ghandour, Thomas John, Jaime Peretzman, Bharadwaj Vellore
ag2672,tj2183,jp2642,vrb2102 @columbia.edu

May 11, 2008

Contents

1 Introduction

Virtual pool is a pool or billiards-like game played on an image of a pool table. Game play is based on a projected image of a pool-table-like surface, with balls positioned on it. A player can then use a cue or cue-like object to 'strike' a ball. The ball which was struck is then projected in the direction it was struck, and made to settle at a new final position, possibly following collisions with other balls on the table. After the balls on the table have settled to their new positons, the player can strike them once again.

The detection of the 'strike' is done using a camera which captures the projected image with the cue stick over it. The image is then processed to determine the direction and speed of the movement of the cue tip relative to the position of the balls. The data gathered from this processing stage is then used to compute the trajectory and distance of motion of the balls, and reposition the balls appropriately. As the balls move and are repositioned, new images of the table and the balls are redrawn and projected for the player to be able to admire his or her stroke and plan the next one.

1.1 Gameplay

The system should be started up with the camera pointing roughly in the direction of the monitor or screen which is used to display the pool table. When the system begins, it automatically begins to calibrate. This process involves a degree of human intervention. During the calibration processes, the system directs to user to move the camera so that the image of the table is visible to the camera. The directions may be to move the camera right or left, up or down and forwards or backwards. When the system is ready, it requires the user to wait briefly while it completes the calibration, and then the game begins.

When the game is in progress, the user can employ keys on the board to trigger a variety of actions. Pressing a key at any time will initiate recalibration of the system. This is particularly useful if a user accidentally distubs the camera during play. When recalibration is requested, the state of the game is saved, and restored later. The game can continue where it was disrupted.

The game is complete when all balls on the table are pocketed. The user is then required to press another key to begin a new game. In fact, at any time during a game, the user can employ this key to reset, and begin a new game.

It goes without saying that mastering virtual pool requires practice! To help novices, the system provides a switch that the user can throw to turn on a crosshair on screen. This serves as a guide to the player on the position of the cue as he or she moves it. Experts can play without the crosshair displayed.

Messages from the system to the user as displayed on the LCD screen on the board. Players' points are displayed on the seven-segment display system. Players always take alternate turns. A player who pockets the white cue able incurs a penalty. When the white ball is pocketed, it is returned to the table and placed a random new position which is guaranteed not to be occupied by another ball.

1.2 Game Configuration

The following switches are available to the user to select a configuration of the game and to trigger events.

Switch	Function
Key 0	Reset System
Key 1	Calibrate System
Key 2	Start new game
SW 10	Turn ON/OFF crosshair
SW 11	Turn ON/OFF striking colored balls
SW 9-0	Green Threshold

2 Design Overview

The ”Virtual Pool” or ”Interactive Projection Pool” game system is built out of a combination of hardware and software components. The system is centred around a NIOS-2 processor[2], a 32-bit general purpose embedded processor. The NIOS-II is a configurable soft-core processor, and in this case, it is targeted to be downloaded to the Cyclone-II[1] family FPGA from Altera.

The system comprises a camera and a projection system connected to the Altera DE2 board comprising the FPGA, memories and other peripherals for connectivity. The physical configuration of the board is illustrated in Figure 2, along with an equivalent block view.

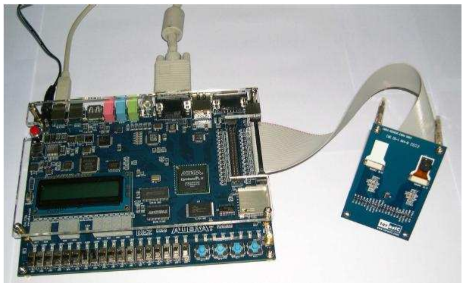


Figure 1: Board Level Connection

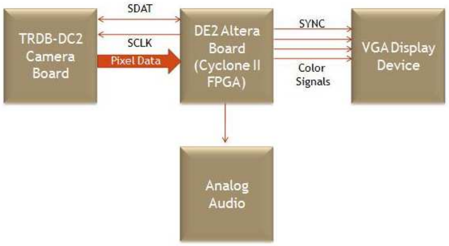


Figure 2: Block View of Physical Interfaces

2.1 High Level View

2.1.1 Basic Ideas

The implementation of the cue-detection is based on the color scheme adopted for the projected image. The pool table is colored green and all the balls placed on the table are colors that have large green components. The module receiving pixel data from the camera (when the camera is pointed at the image of the pool table) expects to see an image which is largely green (within a threshold to allow for environmental noise). As the module scans the image, it is therefore able to identify the presence of objects between the camera and the table by identifying portions of the picture that are distinctly (based on a threshold) different from green. The module then applies a set of image processing algorithms to determine whether the obstacle resembles a cue, and if yes, the position of its tip. This result is then applied to determine whether the cue will impact or has impacted a ball drawn on the table, and what the consequent displacement of the ball is.

2.1.2 Block View

The IPG architecture is based on a NIOS II/f processor and six custom made peripherals. The processor and the six modules are interconnected through an Avalon Data Bus, as shown in figure 3. The six hardware modules are Camera I^2C Interface, the Vision System, SRAM, Sound Driver, VGA Controller and User Interface module.

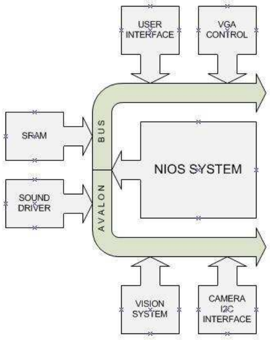


Figure 3: Block Diagram View

The main task of the modules and the processor are as follows.

- Camera I^2C Interface: This module communicates with the camera and enables a driver to customize the configuration of the camera as required. Parameters that are selectable include such values as the frame rate, resolution, active pixel area.
- Vision System: This module consists of three submodules: the Camera Interface Pixel Processor, the Calibration System and the Recognition System. Together, these modules receive pixel data from the camera, select the required portion of the image, process the image and identify the cue-like objects in the image.
- SRAM: The SRAM stores the instructions and data used in the software program that runs on the NIOS processor, and is accessed via a SRAM controller Avalon component.
- Sound Driver: This module implements the interface with on on-board DAC and helps generate the clatter associated with collisions among balls and between balls and the boundaries of the table.
- VGA controller: This module generates the sprites for the balls and the picture of the pool table, and controls the VGA display.
- User Interface: This module comprises all components that are required for the system to communicate with users, including the LCD display for sending messages to the user, the seven segment display for throwing up scores, and the switches and keys to receive configuration preferences and event triggers from the user.
- NIOS II/f Processor: This is the centre of the system. All preipheral control and communication, calibration, ball-dynamics simulation, including tranfer of momentum on collision, acceleration and damping, and game logic happens in software running on the NIOS.

2.2 Hardware-Software partitioning

The modules listed in the earlier section are done in hardware, in software, or in a mixture of both. Presented here is a short summary of the division of labor.

The interface to the camera is implemented as a simple piece of hardware that implements the I^2C physical layer, and a piece of software that uses the register interface exported by the hardware to implement the I^2C protocol.

The vision system is mostly done in hardware. However, the front end of the system as a whole comprises several components that need to work in synchronization and exchange data. This synchronization happens via software. For instance, the calibration module within the front end identifies, the active green area of the picture captured by the camera, and communicates it to the software for the information to be relayed to the image cropper module (detailed later). Similarly, the vision processing algorithm communicates the position of the tip of the cue to software once every frame.

The sound driver works almost entirely in hardware and the role of the software is restricted to requesting that the sound be played.

The VGA controller is highly configurable and offers an extensive set of options that the software can choose from to format the image that is displayed. The options include the size of the pool table to be drawn on screen, the size of the margins around the pool table, the number, position and colours of the balls that are drawn on the table, anso on.

2.3 System Configuration

The NIOS II processor family uses a 32-bit RISC architecture. The instance that it is used in this project is the Nios II/f processor, clocked at 50 MHz and attached to an instruction cache of 4 KB and a data Cache of 2 KB. Also, the processor is built with hardware multiplication and hardware division units along with a dynamic Branch Prediction and narrel Shifter logic. These last features are an important factor in being able to scale up the system to perform vector physics simulations smoothly even for a large number of balls which suffer near-simultaneous collisions.

3 Detailed Design

Some of the significant challenges in the design on the system are the following:

- The output of the camera has considerable noise, and filtering out the noise is important for correctly identifying obstacles in the camera's view.
- The camera and the display using different resolutions, and the span of the camera's view may be different from the size of the projected image. This implies that most algorithms running within the system are always dealing with two sets of co-ordinate systems. This also imposes the need for additional eror detection and correction schemes.
- Users are expected to employ objects that are discernably cue-like when playing. However, the algorithm should also be robust enough to deal with scenarios where random objects appear before the camera. This is particularly necessary in order to be able to deal with users' hands being extended into the 'playing field'.
- The simulation of the movement, collisions and deceleration of the balls involves a significant amount of non-trivial vector mathematics to be implemented.

3.1 Camera Controller

This section details the interfacing of the external camera with the FPGA. The camera used in this system has the Micron MT9M011 CMOS active-pixel digital image sensor[3], which is able to capture frames at SXGA, VGA and CIF resolutions at close-to-video refresh rates.

3.1.1 Camera Physical Interface

The camera, a TRDB-DC2 from Terasic[4], interfaces with the board via a 40-pin flat cable as illustrated in Figure 2. The DE2 board provides two 40 pin expansion headers. Each header connects directly to 36 pins on the Cyclone-II FPGA. In this case, the GPIO_1 slot is used for connecting the camera. Of the two sensors available in the MT9M011, sensor 1 is used. The signals corresponding to this sensor - serial control, clock and data - are carried on pins 1 to 18 of the 40-pin interface. Details of the pin specification can be obtained from [4].

3.1.2 Camera Register Configuration

Table 1 gives a full list of the registers available to be configured on the MT9M011 and the manner in which they are expected to be configured for purposes of this application. This configuration is subject to change on the basis of choices, particularly in the matter of the frame rate and resolution, and for colour-specific gains, which are expected to be based on observations from initial tests. Hence some of these register values are left to be undefined. It may be noted that the configuration of these registers is controlled in software, which enables the application to use these setting flexibly. The hardware for the camera interface only provided the I^{2}C interface to send values to the camera hardware and receive values from it.

3.1.3 Camera Control Module

The camera control module is a combination of a hardware block and a software driver that work together to implement the I^{2}C-like protocol that is used to configure the registers of the camera. The hardware module simply implements a bit level logic that is responsible for putting a '1' or a '0' on a pin, or reading data from it. The entire I^{2}C protocol is implemented in software. This includes controlling the clock that accompanies the data.

The protocol for the camera control interface is simple. Handshaking during data transfer happens via a Start bit, a Stop bit and ACK/NACK bits. The camera control module behaves as the master and is responsible for generating the clocks for all transactions with the camera. As master, it is also responsible for generating the Start and Stop bit. Start and Stop bits on the SDAT line are generated only when the clock is HIGH. Data bits are put on the SDAT line only when clock is LOW. A Start bit involves a HIGH to LOW transition when the clock is HIGH. A stop bit involves a transition from LOW to HIGH when the CLOCK is high.

I^{2}C Interface The I^{2}C interface comprises two lines - a clock, and a serial data line. Each write to a register in the sensor happens in the following steps

- Send a START bit; this is done by first pulling the data line low and then pulling the clock line low.
- Send the WRITE mode slave address (0xBA) with the SDATA being clocked by the SCLK line
- Receive a single bit ACK
- Send the register address (8 bits) on the SDATA line, again accompanied by the SCLK
- Receive a single bit ACK

Table 1: TRDB-DC2 Register Settings

Register	Offset	Default	Configured	Notes
Chip Version	0x00	0x1433	-	Read Only
Row Start	0x01	0x000C	0x00D5	There are 8 dark rows and 4 rows skipped to allow for boundary effects
Column Start	0x02	0x001E	0x0140	There are 26 dark column and 4 columns skipped to allow for boundary effects
Row Width	0x03	0x0400	0x01E0	480 rows of active video
Column Width	0x04	0x0500	0x0280	640 columns of active video pixels
Horizontal Blanking B	0x05	0x018C	0x00CA	202 (minimum permitted when using two ADCs) pixel horizontal blanking
Vertical Blanking B	0x06	0x0032	0x0019	25 row vertical blanking
Horizontal Blanking A	0x07	0x00C6	0x00C6	Unused (Relevant only when context switching is employed)
Vertical Blanking A	0x08	0x0019	0x0019	Unused (Relevant only when context switching is employed)
Shutter Width	0x09	0x0432	0x022A	Reduced to increase frame rate
Row Speed	0x0A	0x0001	0x0001	Unchanged
Extra Delay	0x0B	0x0000	0x0000	Unchanged
Shutter Delay	0x0C	0x0000	0x0000	Unchanged
Reset	0x0D	0x0008	0x0008	Unchanged
FRAME_VALID Control	0x1F	0x0000	0x0000	Unchanged
Read Mode - Context B	0x20	0x0020	0x0020	Unchanged
Read Mode - Context A	0x21	0x040C	0x040C	Unchanged
Show Control	0x22	0x0129	0x0129	Unchanged
Flash Control	0x23	0x0608	0x0608	Unchanged
Green 1 Gain	0x2B	0x0020	0x0020	Unchanged
Blue Gain	0x2C	0x0020	0x0020	Unchanged
Red Gain	0x2D	0x0020	0x0020	Unchanged
Green 2 Gain	0x2E	0x0020	0x0020	Unchanged
Global Gain	0x2F	0x0020	0x0020	Unchanged
Context Control	0xC8	0x000B	0x000B	Unchanged

Table 2: Register Description for I2C Controller

Offset	Bits	Function
0	0	Value to be output on SCLK line
1	0	Data to be output on the SDAT line
	1	Enable write on ’1’, Enable Read on ’0’

- Send the MSB of the value to be written to the register on the SDATA line
- Receive a single bit ACK
- Send the LSB of the value to be written to the register on the SDATA line
- Receive a single bit ACK
- Send a STOP bit; this is done by pulling up the clock line and then pulling up the data line

This is implemented by having the software send a series of commands to hardware by setting a registers corresponding to the data to be sent on the SCLK and SDAT lines. The register corresponding to SCLK is set to ‘0’ to pull the SCLK line low and ‘1’ to pull it high. In contrast, the SDAT line is used to write as well as read data. Whenever a read is being performed (for instance, to receive the acknowledge from the camera), the internal driver of the SDAT line needs to be tri-stated. To enable this, the software requires an extra enable bit in the register used to control the SDAT line. This register comprises an Enable bit that causes the SDAT line to be tri-stated when ‘0’ and enabled when ‘1’. When enabled, the value of the data line is controlled by another bit just as in the case of the SCLK.

3.1.4 Programming the camera interface

The registers that the camera control interface exposes to software running on the NIOS are listed in 2.

3.2 Pixel Processing Front End

The system always functions in one of two modes - calibration and gameplay. Calibration mode always runs first, and may run again upon user request. Calibration is performed by drawing an image of the pool table on the display and then moving the camera until it is positioned such that the entire table lies within the view of the camera. To enable this, black colored margins are drawn around the table so that some basic pixel color recignition can be used to identify the objects that the camera is currently looking at, and therefore, how the camera should be moved so that it can see more of the active green pixel area.

Clearly, during and after calibration, the camera is positioned such that the image captured by the camera contains the entire pool table and then some. However, the margins should be clipped during game play so that they are not visible to the vision algorithm. To enable this, an image cropper component is used that crops the portion of the image that is guaranteed to contain only information about the green area on screen.

To accommodate the calibration and game play requirements, the front end of the system has the following architecture. The interface to the camera is provided by a pixel processing component that receives the pixel data from the camera along with some synchronization signals. The component simply forwards all data. However, it transforms the synchronization signals such that they can be conveniently used by downstream components. Essentially, the frame-valid signal from the camera is transformed into an end-of-frame, and the line valid signal is transformed to an end-of-line. Finally, this front end component generates an important signal called the valid-green. This

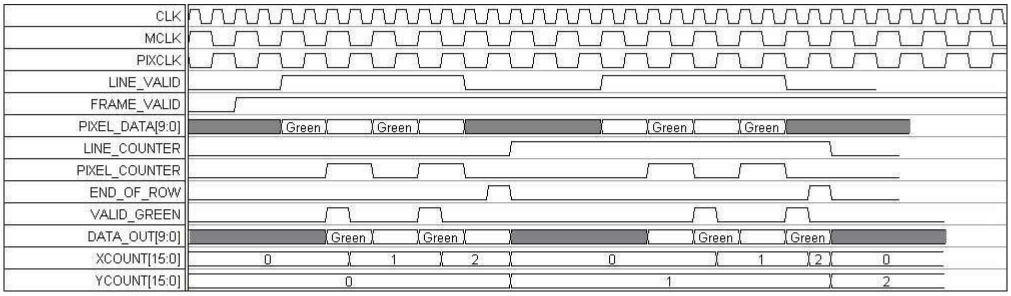


Figure 4: Pixel Processor Component Timing

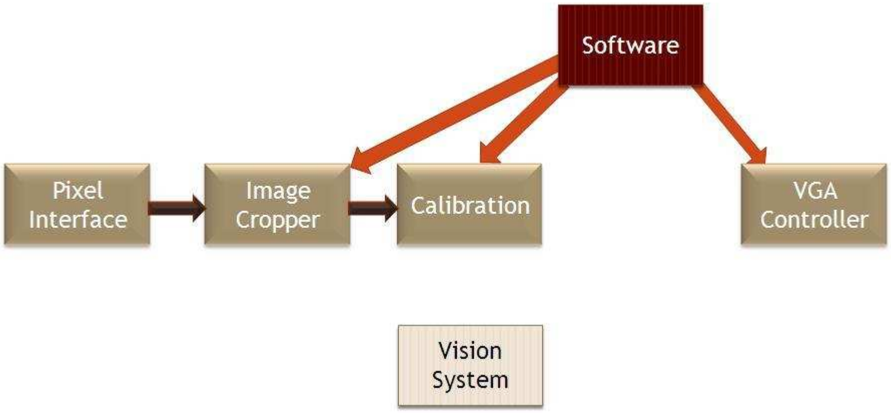


Figure 5: Calibration Mode Data Flow

is a signal that becomes '1' only when the pixel corresponding to a green in the Bayer pattern received from the camera is on the data lines. Every alternate sample is a green in the Bayer pattern. Therefore, the pixel processor generates a valid-green for every second pixel. The timing of these signal is indicated in Figure 4.

Figure 5 and Figure 6 indicate the data flow paths in the calibration and game play modes. In the calibration mode, the image cropper is configured by software to crop no part of the picture. At this time, the image cropper feeds the calibration module. Once calibration is complete, and the start and end co-ordinates of the pool table are determined, the cropper is configured to crop the image to roughly (there is some room left for errors and noise) these co-ordinates. At this time, the cropped image is fed to the vision algorithm. Clearly, the vision algorithm receives only those pixels that green samples in the Bayer pattern, and since there is known to be ne object with low green on the table, the algorithm can identify objects from their colour.

3.3 Calibration

Due to the dependence of our system on the camera, it is really important to properly guide the user in the correct positioning of the camera. The camera calibration algorithm guides the user until the camera is able to recognize the whole active area (pool table). The active area is completely within the camera view range when the algorithm:

- Detects a minimum number of consecutive green pixels in a row, after which the row is marked as a green row
- Recognizes a minimum number of consecutive green rows

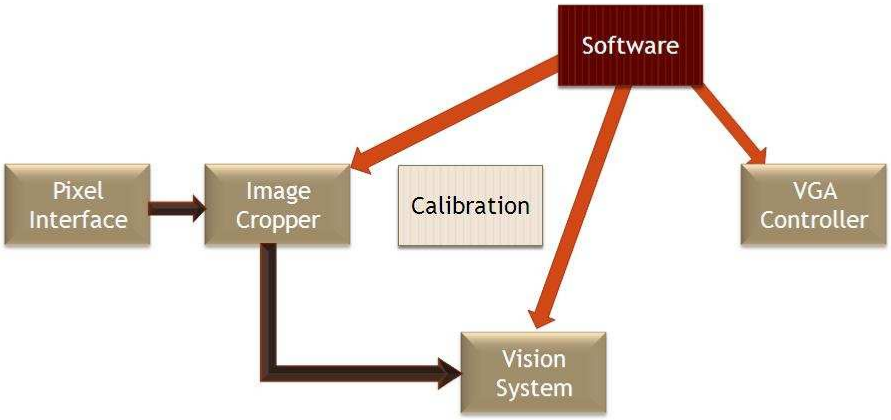


Figure 6: Gameplay Mode Data Flow

- Distinguishes at least a non-green row before and after the block of green rows
- Identifies a minimum number of non-green pixels before and after the green pixels on each row

In order to keep track of these requirements, two signals have been created. The first signal is a three bit signal called active_row, which consists of green_row, left_column and right_column. In this first signal, when the number of consecutive green pixels crosses a minimum threshold, green_row is set to '1'. At the same time, if a number of consecutive non-green pixels is detected, two scenarios might happen. If the threshold for the minimum number of consecutive green pixels has been already crossed, the right_column bit is set to '1' otherwise the left_column bit is set to '1'. The second signal, which is called changes_sig, is a two bit signal. When at least a non green row is detected followed by a consecutive number of green rows, the first bit of the changes_sig signal is set to '1'. The same way, when after a minimum number of green rows a non green row is detected, the second bit is set to '1'.

Using the five bits mentioned above, we can orient the user towards calibrating the camera. First, if the green_row bit is set to '0', it is assumed that the user is not aiming to the display. Consequently, the UI asks the user to move the camera towards the screen. Once the green_row is set to '1', the UI will ask the user to move the camera depending on the other four bits. The different responses are summarized in the truth table on Table 3.3. This status will continue until a successful calibration is achieved. After the calibration is successful, the X and Y coordinates of the upper leftmost corner and lower rightmost corner of the identified green area are returned. Therefore, this algorithm is designed in such a way that a fixed green area can be displayed in the VGA, and the algorithm will find the proper coordinates to crop the received image. Because of this property, the pool table area becomes completely independent of the camera and it can be positioned wherever it is desired. Also, in case the camera is disturbed during game play, the user will have the option to recalibrate the camera without losing the game status, including the position of the balls and player scores.

3.4 Vision System

The Vision System is the hardware block which processes input from the camera to identify the tip of the cue stick or the hand. During development of the system, two separate designs for the vision system were tested. The first design did not support use of the hand to play the game whereas the second design does, limited to certain orientations of the hand. The second design was integrated into the final system and is described here.

Table 3: Truth Table for calibration decisions

Active_row	changes_sig	Instruction
0XX	XX	Point the camera towards the display
1XX	00	Move the camera Backwards
100	XX	Move the camera Backwards
110	XX	Move the camera to the Right
101	XX	Move the camera to the Left
1XX	01	Move the camera down
1XX	10	Move the camera up

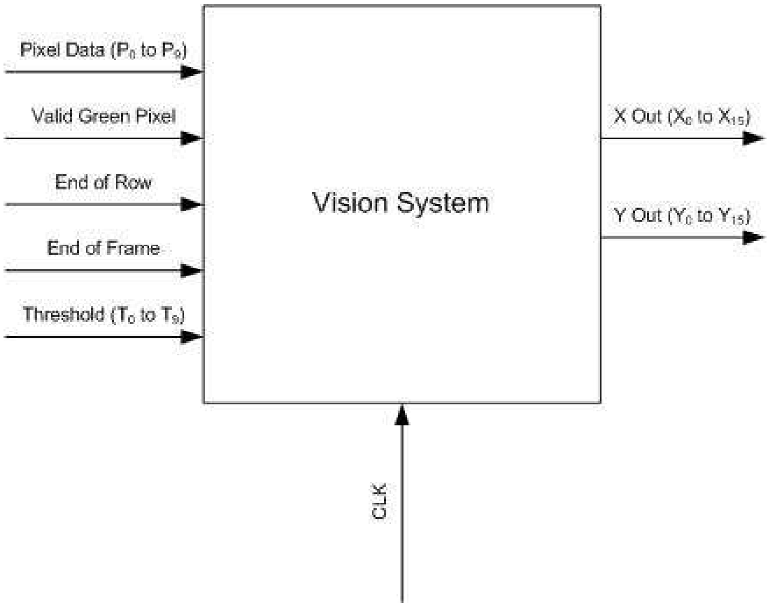

Pixel Data (P_0 to P_9)
Valid Green Pixel
End of Row
End of Frame
Threshold (T_0 to T_9)
Vision System
X Out (X_0 to X_{15})
Y Out (Y_0 to Y_{15})
CLK

Figure 7: Vision System Block Diagram

3.4.1 Interface

The interface signals to the vision system are shown in Figure **??** and are described below.

- Pixel_Data: This input is the 10-bit color data from the camera.
- Valid_Green: The camera uses a Bayer color system, with every alternate pixel on Pixel_Data being a green pixel color value. Given the different clock frequencies of the camera and the vision system, this translates to new green color data once every four vision system clock cycles. Further, the Pixel_Data input is invalid during the blanking intervals of the camera. To indicate when the Pixel_Data input has valid green data, the Valid_Green signal is asserted for one clock cycle when there is new green data on the Pixel_Data line.
- End_of_Row and End_Of_Frame: The End_of_Row signal is asserted for a period of one clock cycle at the end of one row of pixel data. Similarly, End_of_Frame is asserted for a period of one clock at the end of each frame. End_of_Frame also serves as a reset for the Vision System and must be asserted during system startup.
- Threshold: Threshold is a 10-bit color signal which indicates the threshold color value. Any pixel darker than this threshold is interpreted as part of the cue stick by the Vision System. The Threshold is wired to the switches on the board so that it can be adjusted.

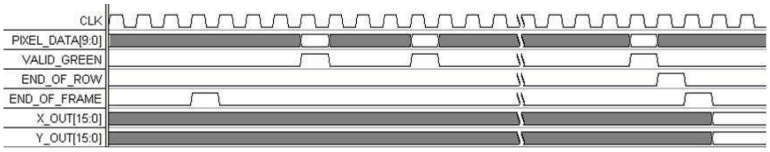


Figure 8: Vision System IO Timing Diagram

- X_Out and Y_Out: These are 16-bit output ports which provide the position of the tip of the cue stick or hand. Each period of logic ‘1’ on Valid_Green is interpreted as a new pixel in the row and therefore, the units for the X co-ordinate is the number of green pixels. Similarly, Y_Out gives the number of rows, each de-limited by a pulse on the End_of_Row input. The output registers X_Out and Y_Out are updated everytime End_of_Frame is asserted with the value computed during the frame.

The timing of these signals is illustrated in Figure 3.4.1.

3.4.2 The Working

Basic Concept The vision system looks at the green channel pixel data from the camera and compares it with the threshold value to obtain a binary image. By looking at this binary image, the vision system finds the extremities of the dark portion of the image, viz. the top most, left most, right most and bottom most dark pixel co-ordinates. Using this information, the vision system branches out into different cases, each taking care of a possible orientation of the cue stick or hand and finally outputs one of the four extremity co-ordinates. In certain cases, the vision system uses data about the width of the image a certain distance below or above the top or bottom extremity respectively to come to a decision about which of the four extremities is the tip.

It was realized early during the design phase that a sophisticated hand recognition algorithm with the ability to locate the index finger tip under all conditions is beyond the scope of this project. Therefore, certain heuristic assumptions were made regarding the possible orientations of the hand. These various orientations were divided into specific cases and conditions on the extremity co-ordinates and the widths mentioned above were developed for choosing between the different cases.

The conditions are based on the idea of an extremity *lying on an edge*. For example, when the left extremity is said to lie on an edge, it means the left most dark point in the image is on the left, top or bottom screen edges. It must be noted that when there are multiple points on the image which qualify for the left most (or right most) extremity, the bottom most amongst them is chosen. Similarly, for the top and bottom extremities, the right most is chosen. Another idea that is used is the concept of *entry edge*. For example when the left extremity is on the left edge, the image is said to enter from the left.

The various possible cases accounted for, the conditions for identifying a particular case and the resulting output co-ordinates are described below.

Bottom Left When the left and bottom extremities lie on an edge, the hand or cue stick is assumed to enter from the bottom left. The tip is either the top extremity or the right extremity and a decision has to be made between them for the cases shown in Figure 9, Figure 10 and Figure 11.

Bottom Right The ideas used for Bottom Left are mirrored and used for the Bottom Right case.

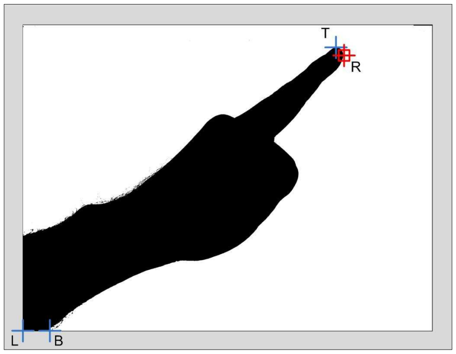


Figure 9: In this case, the top and right extremities are close to each other. Under such a condition, the right extremity is chosen as the output. This is the only case when a cue stick is used instead of a hand.

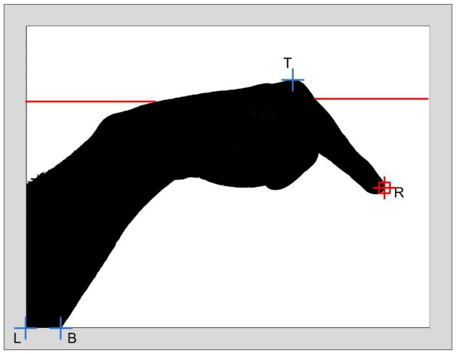


Figure 10: When the width measured a certain distance below the top extremity as shown is greater than a threshold value, it is assumed that the top extremity is not the finger tip. The right extremity is output in this case.

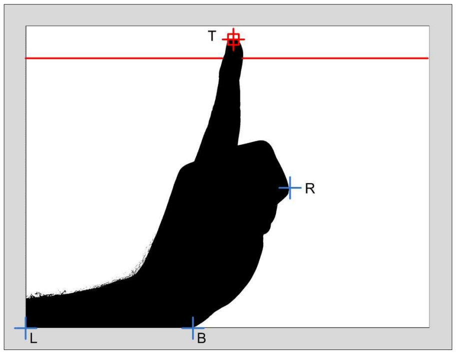


Figure 11: When the width measured a certain distance below the top extremity is lesser than the finger width threshold, the top extremity is assumed to be the finger tip.

Figure 12: In this case, the bottom and right extremities are close to each other. Under such a condition, the right extremity is chosen as the output. This is the only case when a cue stick is used instead of a hand.

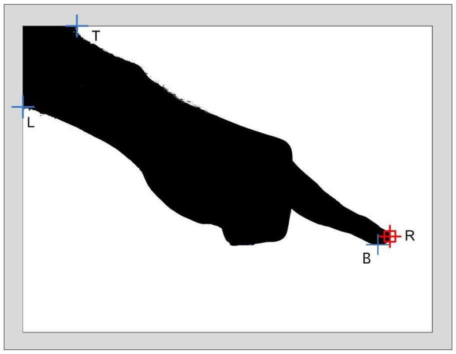
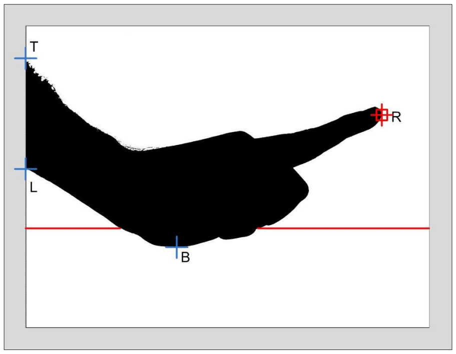


Figure 13: When the width measured a certain distance above the bottom extremity is greater than the finger width threshold, it is assumed that the bottom extremity is not the tip and the right extremity is output. It must be noted that varying results were obtained with this case. When the thumb projects downwards, towards the bottom, the finger width test gives incorrect results as it detects the thumb to be the index finger. Using the right extremity always as the tip for this case solves the problem. This causes problems when the wrist is bent downwards as shown in the Figure 14. However, it is rare that such an orientation is encountered and therefore, can be neglected.



Top Left When the left and top extremities lie on an edge, the hand or cue stick is assumed to enter from the top left. The tip is either the bottom extremity or the right extremity and a decision has to be made between them for the cases shown in Figure 12, Figure 13 and Figure 14.

Top Right The ideas used for Top Left are mirrored and used for the Top Right case.

Left If the left extremity alone lies on an edge, it follows that the entry edge is the left edge. In such a case the right extremity is the tip. The opposite applies when the right extremity alone lies on an edge. Figure 15 illustrates this.

Top If the top extremity alone lies on an edge, it follows that the entry edge is the top edge. In such a case the bottom extremity is the tip. The opposite applies when the bottom extremity alone lies on an edge. Figure 16 demonstrates this case.

3.4.3 Filtering

Making decisions based on a single finger width is inherently prone to errors as depending on the hand orientation, the measured width may occasionally cross the finger width threshold incorrectly. To avoid such noise, a filtering scheme was implemented in software which locks onto a bounding box around the detected tip and discards occasional excursions outside this locked

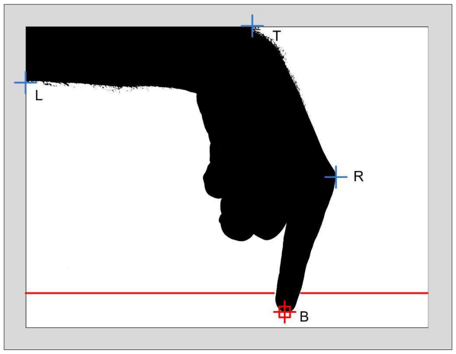


Figure 14: When the width measured a certain distance above the bottom extremity is lesser than the finger width threshold, the bottom extremity is assumed to be the finger tip.

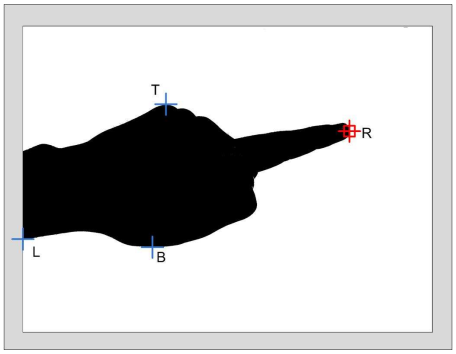


Figure 15: If the left extremity alone lies on an edge, it follows that the entry edge is the left edge. In such a case the right extremity is the tip. The opposite applies when the right extremity alone lies on an edge.

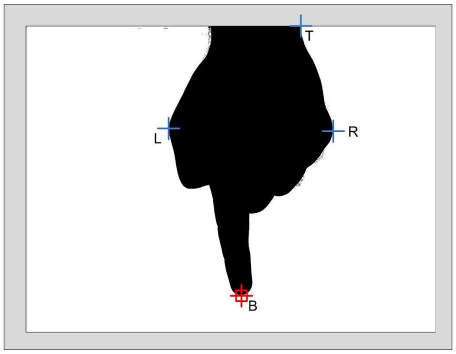


Figure 16: If the top extremity alone lies on an edge, it follows that the entry edge is the top edge. In such a case the bottom extremity is the tip. The opposite applies when the bottom extremity alone lies on an edge.

region. Moreover, a four point moving average filter is also implemented in software to smoothen out the output from the vision system.

3.4.4 Implementation

The entire vision system was implemented in hardware. The information to be extracted from each frame of the camera input includes: the top, bottom, left and right extremities, and the horizontal width a fixed distance below the top extremity and a fixed distance above the bottom extremity. This data extraction is performed on the fly as data comes in from the camera. This eliminates the need for a frame buffer. At the end of the frame, this data is processed based on the conditions specified above.

3.5 Ball Physics Simulation

3.5.1 Basics

The simulation of the movement and collisions of the balls on the pool table is done is software running on the NIOS processor. Each ball is treated as an object which has such properties as position co-ordinates (or a position vector), a velocity vector, a colour, and a visibility state.

Velocity along the x direction is considered positive for a ball whose x co-ordinate is increasing; i.e. the ball is being advanced from left to right on screen. Similarly, velocity aong the y direction is considered positive for a ball whose y co-ordinate is increasing; i.e the ball is moving from top to bottom on screen. For the opposite direction of motion along either axis, the velocity component along that axis is considered negative. The position of the ball is maintained in absolute screen co-ordinates.

Ball visibility helps dealing with balls that have been pocketed and do not have to be considered for computations of motion and collision any further. Balls start out visible and are marked invisible as soon as they are pocketed.

3.5.2 Collision Event Handling

The game logic is handled entirely within a single loop that begins following all initializations and runs until either all balls are pocketed or the user requests either that calibration be performed or a new game be started. The loop maintains a notion of time and all calculates all events over normalized timesteps. At the start of each iteration of the loop, current time is regarded as 1 and the end of the timestep is regarded as 0. Then, given the positions and velocities of all visible balls at the current time, the times after which balls with suffer collisions are calculated. These collisions might be collisions with other balls, collisions with the wall or collisions with the cue.

To simplify the algorithms used in the implementation, the tip of the cue is regarded as a ball of infinitismal size. At each iteration, the position of the cue as last recorded by the vision system hardware is retrieved. The distance that the cue has traversed since the last measurement is determined, and this distance is scaled to calculate a velocity of the cue. Clearly, the cue has the same properties as a ball and via this abstraction, the same mathematical functions can be able to calculate the impact of the cue on a ball as of balls on other such balls.

When the time-to-next-collision has been calculated for all balls, the time to the earliest of these collisions is picked as the size of the next incremental time step. If there are no collisions scheduled to occur in the unit time step, the full time step is used. The game is then advanced by this time step and the process is repeated until a unit time step has elapsed. This constitutes one iteration. Figure 17 illustrates the time steps in a single iteration when two collisions occur within the iteration.

At the beginning of each iteration, a new cue position is sought from hardware. When a new cue position is retrieved, a smoothing filter is applied to the cue position to help mitigate the effects on noise on the accuracy with which the hardware determines the position of the tip of the cue. The smoothing filter uses a weighted average of the current and previous three cue positions to arrive at the filtered value of the new cue position. The weights for the filter are determined empirically.

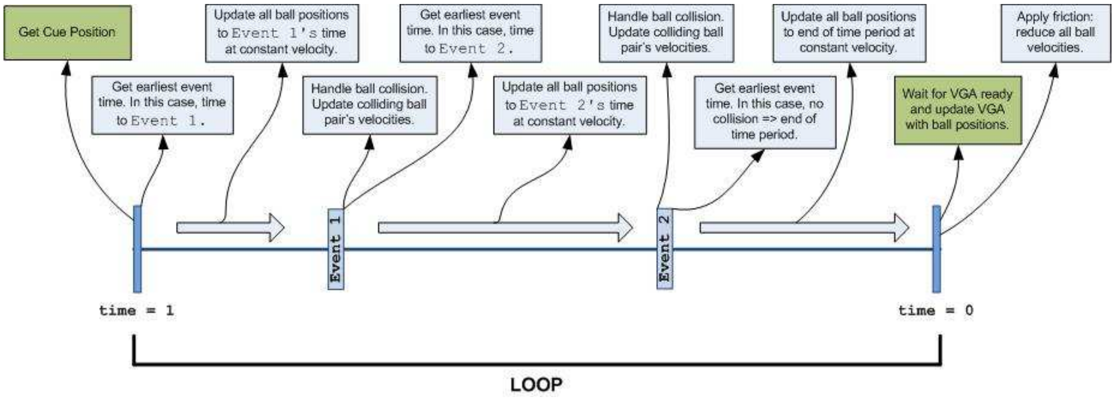

Get Cue Position
Get earliest event time. In this case, time to Event 1.
Update all ball positions to Event 1's time at constant velocity.
Handle ball collision. Update colliding ball pair's velocities.
Get earliest event time. In this case, time to Event 2.
Update all ball positions to Event 2's time at constant velocity.
Handle ball collision. Update colliding ball pair's velocities.
Get earliest event time. In this case, no collision => end of time period.
Update all ball positions to end of time period at constant velocity.
Wait for VGA ready and update VGA with ball positions.
Apply friction: reduce all ball velocities.
Event 1
Event 2
time = 1
time = 0
LOOP

Figure 17: Collision Event Handling Loop

3.5.3 Collision Simulation

The software that handles the collisions among balls implements full vector mathematics to compute the transfer of momentum in terms of new velocity magnitudes and directions for colliding balls. Each balls is associated with a position vector and a velocity vector. The mathematics is as follows. We would like to acknowledge the use of ideas from the gtkpool project for the implementation of the collision handling algorithm. Also shown here is a figure that illustrates this mathematics.

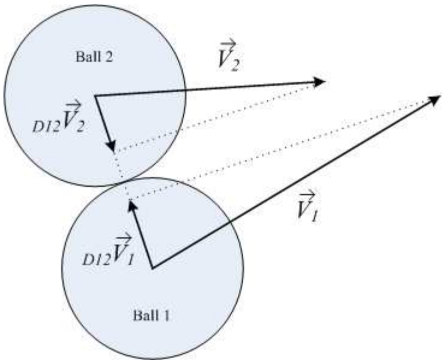


Figure 18: Ball Collisions

Assume $\vec{D_1}$ and $\vec{D_2}$ are the displacement vectors of the two colliding balls. The relative displacement of ball two with respect to ball 1 is

$$\vec{D_{12}} = \vec{D_2} - \vec{D_1}$$

The unit vector in the direction of $\vec{D_{12}}$ is

$$\hat{D_{12}} = \frac{\vec{D_{12}}}{|\vec{D_{12}}|}$$

The component of the first balls velocity along the line joining the centers of the two balls, or along $\vec{D_{12}}$ is

$$|\vec{V_{1D_{12}}}| = \vec{V_1}.\vec{D_{12}}$$

Similarly, for the second ball,

$$|\vec{V_{2D_{12}}}| = \vec{V_2}.\vec{D_{12}}$$

The velocity component for the balls along this direction are

$$\begin{aligned} |\vec{V_{1D_{12}New}}| &= A.\vec{V_{1D_{12}}} - B.\vec{V_{2D_{12}}} \\ |\vec{V_{2D_{12}New}}| &= C.\vec{V_{2D_{12}}} - D.\vec{V_{1D_{12}}} \end{aligned}$$

The new velocities for the two balls are

$$\begin{aligned} \vec{V_{1New}} &= \vec{V_1} - \vec{V_{1D_{12}New}}.\vec{D_{12}} \\ \vec{V_{2New}} &= \vec{V_2} - \vec{V_{2D_{12}New}}.\vec{D_{12}} \end{aligned}$$

3.6 VGA Interface

The VGA Controller is an Avalon component that is responsible for displaying the pool table along with the borders and the seven balls. The balls are pre-drawn, and are displayed like a sprite. Each ball can have a color from a defined color matrix, in addition to an option of being invisible as controlled by the software. This color matrix contains the RGB value for seven different colors that will be used throughout the game. Basically everything is built in a dynamic way so that the software sets all positions and value. One of these things is the black border which is around the table boundaries, and the software can send values through a single register setting the black areas of the top and bottom as well as for the sides. Next the software can control the size of the pool table to be displayed by sending the horizontal and vertical start and end pint of the pool table. Within this area that was sent by the software, the VGA will draw the table will yellow borders and yellow pocket, and setting the background of the table as yellow. The positions of these pockets are also dynamic, and the software can send their coordinates to the VGA controller in order to display them in the correct position.

At this point calibration is ready to start, and the pool table with the black margin is already displayed at this point. The calibration module will locate the area of the pool table drawn by the VGA but in Camera pixel coordinates. Mapping between the area displayed and the area seen by the camera will determine the scaling coefficient to be used.

For the balls in the game, the VGA can support up to seven balls, completely controlled by the software which will send their coordinates on the screen, their color, and whether they will be displayed or not. For that the VGA will use 21 registers to read the data for the balls and sets an internal flag after each read register. Once the VGA reads the 21 registers correctly, it sends to the software a signal saying that it is ready now to take the new coordinate and colors of the balls.

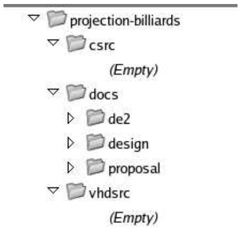
This means the controller will have to wait for all information about all balls to be received, wait till the end of the frame it is already displaying, update the current position values in its registers and then signal the software that it is ready for the next data. At the same time it starts displaying the new frame with the new ball positions. Since square sprites around the balls overlap if the balls are colliding, the module reads only within the circular area of the sprite to make a circular pattern. Basically there is a process running for every ball, and this will indicate the location of the circular area on the screen where its ball will be displayed.

In addition to that, there is a white cross hair that will be displayed to indicate the position of the cue tip. This information is also provided by the software after scaling and translating it to change between camera and VGA coordinates. This cross hair has the highest priority over all other objects and will always be on top. This cross hair can be disabled using the software during game play.

4 Project Management

4.1 Versioning

Configuration management for all project artefacts, code as well as documentation, is done online using Google Code. All users employ an SVN client to access the repository. The project can be accessed online at http://code.google.com/p/projection-billiards.



```
▽ projection-billiards
  ▽ csrc
        (Empty)
  ▽ docs
    ▷ de2
    ▷ design
    ▷ proposal
  ▽ vhdsrc
        (Empty)
```

Figure 19: Directory Tree Structure

The code tree appears as indicated in Figure 19. Test benches for the VHDL sources are included within the vhdsrc directory.

5 Glossary of Terms

ADC	Analog to Digital Converter
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
I^{2}C	Inter-IC Communication
IC	Integrated Circuit
MMIO	Memory Mapped Input Output
VGA	Video Graphics Adapter
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

6 Source Code

References

[1] Altera Corporation. *Cyclone II Device Handbook*. www.altera.com, San Jose, CA, 2007.

[2] Altera Corporation. *NIOS II Processor Reference Handbook*. www.altera.com, San Jose, CA, 2007.

[3] Micron Technology Inc. *1/3-Inch Megapixel CMOS Active-Pixel Digital Image Sensor*. Preliminary, www.micron.com/imaging, 2004.

[4] Terasic. *TRDB-DC2 - 1.3 Mega Pixel Digital Camera Development Kit*. Version 1.1, Preliminary, www.terasic.com, 2006.

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *    Abdulhamid Ghandour
 *    Thomas John
 *    Jaime Peretzman
 *    Bharadwaj Vellore
 *
 * Desc:
 */
#ifndef _GAME_CONFIG_H
#define _GAME_CONFIG_H

#include "fixedpoint.h"

extern long long tableStartX;
extern long long tableEndX;
extern long long tableStartY;
extern long long tableEndY;

extern long long camStartX;
extern long long camStartY;
extern long long camEndX;
extern long long camEndY;

#define TABLE_START_X           INT2FP(tableStartX)
#define TABLE_START_Y           INT2FP(tableStartY)
#define TABLE_END_X             INT2FP(tableEndX)
#define TABLE_END_Y             INT2FP(tableEndY)
#define TABLE_WIDTH             INT2FP(tableEndX - tableStartX)
#define TABLE_HEIGHT            INT2FP(tableEndY - tableStartY)

#define TIME_STEP               20 /* in milliseconds? */
#define RAW_POCKET_RADIUS       14
#define NUM_BALLS               7
#define BALL_RADIUS             INT2FP(14LL)

#define POCKET_RADIUS           INT2FP(10LL)

#define TOP_LEFT_POCKET_X       TABLE_START_X
#define TOP_LEFT_POCKET_Y       TABLE_START_Y
#define TOP_RIGHT_POCKET_X      TABLE_END_X
#define TOP_RIGHT_POCKET_Y      TABLE_START_Y
#define TOP_MID_POCKET_X        FPDIV((TABLE_START_X + TABLE_END_X),INT2FP(2LL));
#define TOP_MID_POCKET_Y        TABLE_START_Y
#define BOTTOM_LEFT_POCKET_X    TABLE_START_X
#define BOTTOM_LEFT_POCKET_Y    TABLE_END_Y
#define BOTTOM_RIGHT_POCKET_X   TABLE_END_X
#define BOTTOM_RIGHT_POCKET_Y   TABLE_END_Y
#define BOTTOM_MID_POCKET_X     TABLE_START_X + FPDIV(TABLE_WIDTH,INT2FP(2LL)) \
                                 - INT2FP(1LL)
#define BOTTOM_MID_POCKET_Y     TABLE_END_Y

#define DAMPING_COEFF           FPDIV(INT2FP(2LL),INT2FP(100LL))

#endif /* _GAME_CONFIG_H */
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *    Abdulhamid Ghandour
 *    Thomas John
 *    Jaime Peretzman
 *    Bharadwaj Vellore
 *
 * Desc:
 */
#ifndef _TYPES_H
#define _TYPES_H

typedef enum{
  FALSE,
  TRUE
}bool_t;

typedef enum{
  NONE,
  LEFT,
  RIGHT,
  TOP,
  BOTTOM
}edge_t;

typedef enum{
  NO_COLLISION,
  BALL_COLLISION,
  POCKET_COLLISION,
  CUE_COLLISION,
  TABLE_COLLISION
}event_t;

struct vector{
  long long x;
  long long y;
};

/*
// Every ball has the following properties
//    Position (x,y) - The centre of the circle
//    Velocity vector (x,y) - This is positive for a ball moving right and/or
//      down, and negative for a ball moving left and/or up.
//    Radius
//    Visibility state
*/
typedef enum {
  BALL_VISIBLE = 0,
  BALL_INVISIBLE = 1
}BallState_e;

struct ball_t{
  struct vector pos;
  struct vector vel;
  long long radius;
  BallState_e ballState;
  unsigned char colour;
  int points;
};

struct player_t{
  int points;
};

#endif /* _TYPES_H */
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *    Abdulhamid Ghandour
 *    Thomas John
 *    Jaime Peretzman
 *    Bharadwaj Vellore
 *
 * Desc:
 */
#ifndef _DEBUG_H
#define _DEBUG_H

#include <stdio.h>
#include <assert.h>

#define DP_INFO              printf("%s,␣%s():␣", __FILE__, __FUNCTION__)
#define DP_PREFIX            printf("(dbg)␣"),DP_INFO
#define print(x)             DP_PREFIX, printf(x)
#define print1(x,x1)         DP_PREFIX, printf((x),(x1))
#define print2(x,x1,x2)      DP_PREFIX, printf((x),(x1),(x2))
#define print3(x,x1,x2,x3)   DP_PREFIX, printf((x),(x1),(x2),(x3))

#define DP_ASSERT(x,y)       (x)?1:(print(y),assert(0));

#ifdef ALT_DEBUG

#define DP(x)                print(x)
#define DP1(x,x1)            print1((x),(x1))
#define DP2(x,x1,x2)         print2((x),(x1),(x2))
#define DP3(x,x1,x2,x3)      print3((x),(x1),(x2),(x3))
#define DP_HI                DP("Enter\n");
#define DP_BYE               DP("Leave\n");

#else /* ALT_DEBUG */

#define DP(x)
#define DP1(x,x1)
#define DP2(x,x1,x2)
#define DP3(x,x1,x2,x3)
#define DP_HI
#define DP_BYE

#endif /* ALT_DEBUG */

#endif /* _DEBUG_H */
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *    Abdulhamid Ghandour
 *    Thomas John
 *    Jaime Peretzman
 *    Bharadwaj Vellore
 *
 * Desc:
 */
#ifndef _BALL_H
#define _BALL_H

#include "types.h"
#include "system.h"
#include "io.h"

 bool_t isBallMoving(const struct ball_t *ball);
long long collisionWithTableTime(const struct ball_t *ball, edge_t *edge);
long long collisionWithBallTime(const struct ball_t *ball1, const struct ball_t *ball2);
void handleBallCollision(struct ball_t *ball1, struct ball_t *ball2);
long long handleCollisionWithCue(struct ball_t *ball, const struct ball_t *cue);
void moveBalls(struct ball_t *balls, long long time);
void drawBalls(struct ball_t *balls);
void applyFriction(struct ball_t *balls);

#define BALL_X                           0
#define BALL_Y                           1
#define BALL_COLOUR                      2

#define BALL_0_BASE                      0
#define BALL_1_BASE                      3
#define BALL_2_BASE                      6
#define BALL_3_BASE                      9
#define BALL_4_BASE                     22
#define BALL_5_BASE                     25
#define BALL_6_BASE                     28

#define COL_WHITE                        0
#define COL_YELLOW                       1
#define COL_CYAN                         2
#define COL_INVISIBLE                    3
#define COL_K1                           4
#define COL_K2                           5
#define COL_K3                           6


#define VGA_FLAG                         12

#define SPRITE_X(ball)                   FP2INT((ball).pos.x - (ball).radius)
#define SPRITE_Y(ball)                   FP2INT((ball).pos.y - (ball).radius)

#define IOWR_POS(base,offset,data)       IOWR_16DIRECT(base, (offset) * 2, data)
#define IOWR_VAL(offset,data)            IOWR_POS(VGA_BASE, offset, data)

#endif /* _BALL_H */
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *    Abdulhamid Ghandour
 *    Thomas John
 *    Jaime Peretzman
 *    Bharadwaj Vellore
 *
 * Desc:
 */
#ifndef I2C_H_
#define I2C_H_

/*
 * Start Bit
 */
#define START                 \
      SDAT_SET;               \
      HALF_CLOCK_DELAY;       \
      SCLK_SET;               \
      HALF_CLOCK_DELAY;       \
      SDAT_CLR;               \
      HALF_CLOCK_DELAY;       \
      SCLK_CLR;               \
      HALF_CLOCK_DELAY;

/*
 * Stop Bit
 */
#define STOP                  \
      SDAT_CLR;               \
      HALF_CLOCK_DELAY;       \
      SCLK_SET;               \
      HALF_CLOCK_DELAY;       \
      SDAT_SET;

/*
 * Sequence for a '1' bit
 */
#define SEND_BIT_1   \
   SDAT_SET;         \
   HALF_CLOCK_DELAY; \
   SCLK_SET;         \
   ONE_CLOCK_DELAY;  \
   SCLK_CLR;         \
   HALF_CLOCK_DELAY;\
   //SDAT_CLR;

/*
 * Sequence for a '0' bit
 */
#define SEND_BIT_0   \
   SDAT_CLR;         \
   HALF_CLOCK_DELAY; \
   SCLK_SET;         \
   ONE_CLOCK_DELAY;  \
   SCLK_CLR;         \
   HALF_CLOCK_DELAY;\
   //SDAT_SET;

#define SEND_0  \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_0;

#define SEND_1  \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_1;

#define SEND_2  \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_0;
```

```
#define SEND_3  \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_1;

#define SEND_4  \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_0;

#define SEND_5  \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_1;

#define SEND_6  \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_0;

#define SEND_7  \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_1;

#define SEND_8  \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_0;

#define SEND_9  \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_0; \
      SEND_BIT_1;

#define SEND_A  \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_0;

#define SEND_B  \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_1; \
      SEND_BIT_1;

#define SEND_C  \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_0;

#define SEND_D  \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_0; \
      SEND_BIT_1;

#define SEND_E  \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_0;

#define SEND_F  \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_1; \
      SEND_BIT_1;

#define READ(ack)         \
      SDAT_TRISTATE;      \
    HALF_CLOCK_DELAY;    \
    SCLK_SET;            \
    HALF_CLOCK_DELAY;    \
    ack = RD_ACK;        \
    HALF_CLOCK_DELAY;    \
    SCLK_CLR;            \
    HALF_CLOCK_DELAY;

#define ACK              \
    SDAT_CLR;            \
    HALF_CLOCK_DELAY;    \
    SCLK_SET;            \
    ONE_CLOCK_DELAY;     \
    SCLK_CLR;            \
    HALF_CLOCK_DELAY;

#define NACK             \
    SDAT_SET;            \
    HALF_CLOCK_DELAY;    \
    SCLK_SET;            \
    ONE_CLOCK_DELAY;     \
    SCLK_CLR;            \
    HALF_CLOCK_DELAY;    \
    SDAT_CLR;

#define COMM_INIT        \
    START;               \
    SEND_B;              \
    SEND_A;

#define READ_ACK(ack)    READ(ack)

int configureCamera();

#endif /*I2C_H_*/
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#ifndef _FIXED_POINT_H
#define _FIXED_POINT_H

#define FRAC_PRECISION          12
#define MAG_PRECISION           (64 - FRAC_PRECISION)

#define FPSUM(x,y)              ((x) + (y))
#define FPSUB(x,y)              ((x) - (y))
#define FPMUL(x,y)              (((x) * (y)) >> FRAC_PRECISION)
#define FPDIV(x,y)              (((x) << FRAC_PRECISION) / (y))
#define FPSQR(x)                FPMUL((x),(x))
#define FP2INT(x)               ((x) >> FRAC_PRECISION)
#define INT2FP(x)               ((x) << FRAC_PRECISION)

long long FPSQRT(long long num);
void printFP(long long fpnum);

#endif /* _FIXED_POINT_H */
```

```
#ifndef _CALIBRATION_H_
#define _CALIBRATION_H_

#include "system.h"
#include "io.h"

#define NO_CUE_DETECTED          (IORD_32DIRECT(VISION_BASE, 0) >> 31)

#define START_CALIBRATION        IOWR_32DIRECT(VISION_BASE, 4 * 4, 1)
#define STOP_CALIBRATION         IOWR_32DIRECT(VISION_BASE, 4 * 4, 0)
#define READ_REPOS_REG           IORD_32DIRECT(VISION_BASE, 3 * 4)
#define READ_CAMERA_START        IORD_32DIRECT(VISION_BASE, 1 * 4)
#define READ_CAMERA_END          IORD_32DIRECT(VISION_BASE, 2 * 4)

#define SET_CAMERA_SANDBOX_START_X(data)  \
                                 IOWR_32DIRECT(VISION_BASE, 5 * 4, (data))
#define SET_CAMERA_SANDBOX_END_X(data)  \
                                 IOWR_32DIRECT(VISION_BASE, 6 * 4, (data))
#define SET_CAMERA_SANDBOX_START_Y(data)  \
                                 IOWR_32DIRECT(VISION_BASE, 7 * 4, (data))
#define SET_CAMERA_SANDBOX_END_Y(data)  \
                                 IOWR_32DIRECT(VISION_BASE, 8 * 4, (data))

#define TABLE_START_HOZ_POS     140
#define TABLE_END_HOZ_POS       500
#define TABLE_START_VER_POS     120
#define TABLE_END_VER_POS       360

#define CAMERA_CROP_MARGIN_HOZ 10
#define CAMERA_CROP_MARGIN_VER 20
#define BLACK_MARGIN_HOZ        110
#define BLACK_MARGIN_VER        90

#define BLACK_MARGIN_CONFIG     ((BLACK_MARGIN_VER << 8) | BLACK_MARGIN_HOZ)
#define BLACK_MARGIN_REG        31
#define BLACK_MARGIN_SET(size) IOWR_16DIRECT(VGA_BASE, 31 * 2, size)

#define GREEN_COLUMN_THRESHOLD 120
#define GREEN_ROW_THRESHOLD     240

#define SET_GREEN_ROW_THR        IOWR_32DIRECT(VISION_BASE, 10 * 4, (GREEN_ROW_THRESHOLD))
#define SET_GREEN_COL_THR        IOWR_32DIRECT(VISION_BASE, 9 * 4, (GREEN_COLUMN_THRESHOLD))

void doCalibration();
int calibrate();

#endif /*CALIBRATION_H_*/
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc: LCD implementation borrowed from code by Prof. Stephen Edwards,
 * Columbia University
 */
#ifndef _UI_H_
#define _UI_H_

#include "system.h"
#include "io.h"
#include "ball.h"

// LCD Module 16*2
#define lcd_write_cmd(base, data)              IOWR(base, 0, data)
#define lcd_read_cmd(base)                     IORD(base, 1)
#define lcd_write_data(base, data)             IOWR(base, 2, data)
#define lcd_read_data(base)                    IORD(base, 3)

#define IOWR_LED_DATA(base, offset, data)      \
                               IOWR_16DIRECT(base, (offset) * 2, data)
#define IORD_LED_DATA(base, offset)            \
                               IORD_16DIRECT(base, (offset) * 2)
#define IOWR_LED_SPEED(base, data)             \
                               IOWR_16DIRECT(base + 32, 0, data)
#define IORD_FLAG (base, offset)               \
                               IORD_16DIRECT(base, (offset) * 2)

#define PLAY_SOUND             IOWR_32DIRECT(SOUNDDRIVER_BASE, 0, 1)
#define HEXWRITE(reg,data)     IOWR_32DIRECT(UICONTROL_BASE, (reg) * 4, data)

#define HEX0(data)             HEXWRITE(0, (data))
#define HEX1(data)             HEXWRITE(1, (data))
#define HEX2(data)             HEXWRITE(2, (data))
#define HEX3(data)             HEXWRITE(3, (data))
#define HEX4(data)             HEXWRITE(4, (data))
#define HEX5(data)             HEXWRITE(5, (data))
#define HEX6(data)             HEXWRITE(6, (data))
#define HEX7(data)             HEXWRITE(7, (data))

#define CALIBRATION_REQUESTED  IORD_32DIRECT(UICONTROL_BASE, 8 * 4)
#define CALIBRATED             IOWR_32DIRECT(UICONTROL_BASE, 8 * 4, 1)
#define NEW_GAME_REQUESTED     IORD_32DIRECT(UICONTROL_BASE, 9 * 4)
#define STARTED_GAME           IOWR_32DIRECT(UICONTROL_BASE, 9 * 4, 1)

extern unsigned long sevensegment[];

void LCD_Init();
void LCD_Show_Text(char* Text);
void LCD_Line2();
void initPointsDisplay();

#endif /* _UI_H_ */
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#include <stdio.h>
#include "debug.h"
#include "fixedpoint.h"
#include "gameconfig.h"
#include "types.h"
#include "ball.h"
#include "i2c.h"
#include "ui.h"
#include "calibration.h"

#define NO_CUE_DETECTED (IORD_32DIRECT(VISION_BASE, 0) >> 31)

static void initPockets();
static void initBalls();
void initPlayers();
static void initCue();
static int play();
extern int calibrate();
struct ball_t balls[NUM_BALLS];
struct ball_t cue;
struct ball_t pockets[6];
struct player_t player1, player2;
struct player_t *pCurrentPlayer;

typedef enum gameState_e {
  GAME_PLAYING,
  GAME_WAITING_TO_PLAY
}gameState_t;
gameState_t gameState = GAME_WAITING_TO_PLAY;

void showPoints(struct player_t *player);

int main(){
  DP_HI;

  DP("Welcome␣to␣Projection␣Pool\n");

  LCD_Init();
  initPointsDisplay();

  LCD_Show_Text("Welcome␣to␣Pool!");

  initPlayers();
  configureCamera();
  doCalibration();
  CALIBRATED;
  initCue();
  initPockets();
  initBalls();
  initPointsDisplay();
  drawBalls(balls);
  STARTED_GAME;

  while(1){
    if(-1 == play()){
      doCalibration();
      CALIBRATED;
      drawBalls(balls);
    }else{
      initPlayers();
      initCue();
      initBalls();
      initPointsDisplay();
      drawBalls(balls);
      STARTED_GAME;
    }
  }
  DP_BYE;
}

long long calibScaleHoz, calibOffsetHoz;
long long calibScaleVer, calibOffsetVer;
void configureScaling(){
  long long vgaHozRes = 640;
  long long vgaVerRes = 480;
  long long blackMarginHoz = BLACK_MARGIN_HOZ;
  long long blackMarginVer = BLACK_MARGIN_VER;

  calibScaleHoz = FPDIV(
    INT2FP(vgaHozRes - 2*blackMarginHoz),
    INT2FP(camEndX - camStartX)
  );
  calibOffsetHoz =
    FPMUL(calibScaleHoz,INT2FP(CAMERA_CROP_MARGIN_HOZ)) +
    INT2FP(blackMarginHoz);

  calibScaleVer = FPDIV(
    INT2FP(vgaVerRes - 2*blackMarginVer),
    INT2FP(camEndY - camStartY)
  );
  calibOffsetVer =
    FPMUL(calibScaleVer,INT2FP(CAMERA_CROP_MARGIN_VER)) +
    INT2FP(blackMarginVer);
}

int play(){
  int i,j;
  long long time, earliestEventTime, eventTime;
  event_t eventType;
  struct ball_t *collidingBall1, *collidingBall2;
  edge_t tableEdge, collisionTableEdge;
  bool_t ballsMoved = FALSE;
  bool_t repositionedWhite;

  long long xin = 0, prevXin = 0;
  long long yin = 0, prevYin = 0;
  long long xinBeforeFilter = 0, yinBeforeFilter = 0;
  long long A = FPDIV(INT2FP(18),INT2FP(10));
  long long W0, W1, W2, W3;

  W0 = FPDIV(INT2FP(5),INT2FP(10));
  W1 = FPDIV(INT2FP(3),INT2FP(10));
  W2 = FPDIV(INT2FP(1),INT2FP(10));
  W3 = FPDIV(INT2FP(1),INT2FP(10));

  struct vector prevCuePos1, prevCuePos2, prevCuePos3;
  prevCuePos1.x = 0;
  prevCuePos1.y = 0;
  prevCuePos2.x = 0;
  prevCuePos2.y = 0;
  prevCuePos3.x = 0;
  prevCuePos3.y = 0;

  long long prevXinBeforeFilter = 0, prevYinBeforeFilter = 0;
  long long absPreFilterXDiff, absPreFilterYDiff;

  int numBallsPocketed = 0;
  gameState = GAME_WAITING_TO_PLAY;

  configureScaling();
  LCD_Init();
  LCD_Show_Text(”Player”);
  LCD_Line2();
  LCD_Show_Text(”1”);

  while(1){
    time = INT2FP(1LL);
    if(!NO_CUE_DETECTED){
      xin = (long long)IORD_32DIRECT(VISION_BASE, 0);
      yin = ((xin >> 16) & 0x7FFF);
      xin = (xin & 0xFFFF);
      xinBeforeFilter = FPMUL(calibScaleHoz,INT2FP(xin)) + calibOffsetHoz;
      yinBeforeFilter = FPMUL(calibScaleVer,INT2FP(yin)) + calibOffsetVer;

      /*
       * Filter to limit step changes in cue position
       */
      absPreFilterXDiff = (xinBeforeFilter > prevXinBeforeFilter)?
                          (xinBeforeFilter - prevXinBeforeFilter):
                          (prevXinBeforeFilter > xinBeforeFilter);
      absPreFilterYDiff = (yinBeforeFilter > prevYinBeforeFilter)?
                          (yinBeforeFilter - prevYinBeforeFilter):
                          (prevYinBeforeFilter > yinBeforeFilter);

      if((prevXinBeforeFilter != 0) && (prevXinBeforeFilter !=0)){
        if((absPreFilterXDiff > INT2FP(75)) || (absPreFilterYDiff > INT2FP(75))){
          xinBeforeFilter = prevXinBeforeFilter;
          yinBeforeFilter = prevYinBeforeFilter;
        }else{
          prevXinBeforeFilter = xinBeforeFilter;
          prevYinBeforeFilter = yinBeforeFilter;
        }
      }else{
        prevXinBeforeFilter = xinBeforeFilter;
        prevYinBeforeFilter = yinBeforeFilter;
      }

      /*
       * Smoothing filter for cue position
       */
      xin = FPMUL(W0, xinBeforeFilter) +
            FPMUL(W1, prevCuePos1.x) +
            FPMUL(W2, prevCuePos2.x) +
            FPMUL(W3, prevCuePos3.x);
      yin = FPMUL(W0, yinBeforeFilter) +
            FPMUL(W1, prevCuePos1.y) +
            FPMUL(W2, prevCuePos2.y) +
            FPMUL(W3, prevCuePos3.y);

      prevCuePos3.x = prevCuePos2.x;
      prevCuePos3.y = prevCuePos2.y;
      prevCuePos2.x = prevCuePos1.x;
      prevCuePos2.y = prevCuePos1.y;
      prevCuePos1.x = xinBeforeFilter;
      prevCuePos1.y = yinBeforeFilter;

      cue.ballState = BALL_VISIBLE;

      if ((prevXin != 0) && (prevYin != 0)){
        cue.pos.x = prevXin;
        cue.pos.y = prevYin;
        if(IORD_32DIRECT(UICONTROL_BASE, 10 * 4) & 0x00000400){
            IOWR_POS(VGA_BASE, 13, FP2INT(prevXin));
            IOWR_POS(VGA_BASE, 14, FP2INT(prevYin));
        }
        else{
            IOWR_POS(VGA_BASE, 13, 999);
            IOWR_POS(VGA_BASE, 14, 999);
        }
        cue.vel.x = FPMUL(A,(xin - prevXin));
        cue.vel.y = FPMUL(A,(yin - prevYin));
        prevXin = xin;
        prevYin = yin;
      }
      else{
        prevXin = xin;
        prevYin = yin;
      }
    }else{
      cue.ballState = BALL_INVISIBLE;
      prevXin = 0;
      prevYin = 0;
      prevXinBeforeFilter = 0;
      prevYinBeforeFilter = 0;
    }

    ballsMoved = FALSE;
    do{
      earliestEventTime = time;
      eventType = NO_COLLISION;
      collidingBall1 = NULL;
      collidingBall2 = NULL;
      tableEdge = NONE;

      for(i=0; i<NUM_BALLS; i++){
        if(BALL_INVISIBLE == balls[i].ballState){
          continue;
        }
        if(isBallMoving(&balls[i])){
          ballsMoved = TRUE;
        }

        /*
        // Check for collisions with the table boundaries
        */
        if(isBallMoving(&balls[i]) == TRUE){
          eventTime = collisionWithTableTime(&balls[i],&tableEdge);
          if((eventTime >= 0) && (eventTime < earliestEventTime)){
            earliestEventTime = eventTime;
            collidingBall1 = &balls[i];
            collidingBall2 = NULL;
            eventType = TABLE_COLLISION;
            collisionTableEdge = tableEdge;
          }
        }

        /*
        // Check for "collision" with cue
        */
        if(BALL_INVISIBLE != cue.ballState){
          if((balls[i].colour == COL_WHITE) ||
              ((IORD_32DIRECT(UICONTROL_BASE, 10 * 4) & 0x1000) == 0)
          ){
            if(((balls[i].vel.x == 0) && (balls[i].vel.y == 0)) ||
               ((IORD_32DIRECT(UICONTROL_BASE, 10 * 4) & 0x800) == 0)
            ){
              eventTime = collisionWithBallTime(&balls[i],&cue);
              if((eventTime >= 0) && (eventTime < earliestEventTime)){
                earliestEventTime = eventTime;
                collidingBall1 = &balls[i];
                collidingBall2 = &cue;
                eventType = CUE_COLLISION;
              }
            }
          }
        }

        /*
        // Check for "collision" with pockets
        */
        if(isBallMoving(&balls[i]) == TRUE){
          for(j=0; j<6; j++){
            eventTime = collisionWithBallTime(&balls[i], &pockets[j]);
            if((eventTime >= 0) && (eventTime < earliestEventTime)){
              earliestEventTime = eventTime;
              collidingBall1 = &balls[i];
              collidingBall2 = &pockets[j];
              eventType = POCKET_COLLISION;
            }
          }
        }

        /*
        // Collision with other balls
        */
        if(isBallMoving(&balls[i]) == TRUE){
          for(j=0; j<NUM_BALLS; j++){
            if(BALL_INVISIBLE == balls[j].ballState){
              continue;
            }

            eventTime = collisionWithBallTime(&balls[i],&balls[j]);
            if((eventTime >= 0) && (eventTime < earliestEventTime)){
              earliestEventTime = eventTime;
              collidingBall1 = &balls[i];
              collidingBall2 = &balls[j];
              eventType = BALL_COLLISION;
            }
          }
        }
      }

      moveBalls(balls, earliestEventTime);

      switch(eventType){
        case NO_COLLISION:
        break;
```

```

case POCKET_COLLISION:
   collidingBall1->ballState = BALL_INVISIBLE;
   numBallsPocketed++;
   PLAY_SOUND;
   pCurrentPlayer->points += collidingBall1->points;
   showPoints(pCurrentPlayer);

   if(collidingBall1->colour == COL_WHITE){
      int k;
      repositionedWhite = TRUE;
      numBallsPocketed--;
      collidingBall1->ballState = BALL_VISIBLE;
      collidingBall1->pos.x = TABLE_END_X - INT2FP(140);
      collidingBall1->pos.y = TABLE_END_Y - INT2FP(120);
      printf("New␣Pos␣=␣%lld,␣%lld\n",
         FP2INT(collidingBall1->pos.x),
         FP2INT(collidingBall1->pos.y)
      );
      collidingBall1->vel.x = 0;
      collidingBall1->vel.y = 0;
      while(1){
         long long absXDiff, absYDiff;
         for(k=0; k<NUM_BALLS; k++){
            if(&balls[k] == collidingBall1){
               continue;
            }
            absXDiff = (collidingBall1->pos.x > balls[k].pos.x)?
                       (collidingBall1->pos.x - balls[k].pos.x):
                       (balls[k].pos.x - collidingBall1->pos.x);
            absYDiff = (collidingBall1->pos.y > balls[k].pos.y)?
                       (collidingBall1->pos.y - balls[k].pos.y):
                       (balls[k].pos.y - collidingBall1->pos.y);
            if((absXDiff < INT2FP(30)) && (absYDiff < INT2FP(30))){
               repositionedWhite = FALSE;
               break;
            }
         }
         if(repositionedWhite == FALSE){
            collidingBall1->pos.x += INT2FP(15);
            collidingBall1->pos.y += INT2FP(10);
            if(collidingBall1->pos.x > (TABLE_END_X - INT2FP(20))){
               collidingBall1->pos.x -= INT2FP(200);
            }
            if(collidingBall1->pos.y > (TABLE_END_Y - INT2FP(20))){
               collidingBall1->pos.y -= INT2FP(150);
            }
            printf("New␣Pos␣=␣%lld,␣%lld\n",
               FP2INT(collidingBall1->pos.x),
               FP2INT(collidingBall1->pos.y)
            );
            repositionedWhite = TRUE;
         }else{
            break;
         }
      }
   }
break;

case CUE_COLLISION:
   handleCollisionWithCue(collidingBall1,&cue);
   PLAY_SOUND;
break;

case TABLE_COLLISION:
   switch(collisionTableEdge){
      case LEFT:
      case RIGHT:
         collidingBall1->vel.x *= -1LL;
      break;

      case TOP:
      case BOTTOM:
         collidingBall1->vel.y *= -1LL;
      break;

      default:
         DP_ASSERT(0,"Collision␣with␣non-existent␣table␣edge!");
      break;
   }
```

```
          PLAY_SOUND;
        break;

        case BALL_COLLISION:
          handleBallCollision(collidingBall1, collidingBall2);
          PLAY_SOUND;
        break;

        default:
          DP_ASSERT(0,"Invalid␣event");
        break;
      };
      drawBalls(balls);
      time -= earliestEventTime;
    }while(time > 0);
    drawBalls(balls);
    applyFriction(balls);

    /*
     * Check if there have been balls moved in the last time step
     * If yes, and if the previous state was WAITING_TO_PLAY, then
     * switch to PLAYING_STATE. Continue with the same player.
     * If not, and if the previous state was PLAYING, then switch
     * to the WAITING_TO_PLAY. Also switch players.
     */
    if((GAME_PLAYING == gameState) && (FALSE == ballsMoved)){
      gameState = GAME_WAITING_TO_PLAY;
      if(pCurrentPlayer == &player1){
        pCurrentPlayer = &player2;
        LCD_Line2();
        LCD_Show_Text("2");
      }else if(pCurrentPlayer == &player2){
        pCurrentPlayer = &player1;
        LCD_Line2();
        LCD_Show_Text("1");
      }else{
        fflush(stdout);
        DP_ASSERT(0,"Invalid␣player\n");
      }
    }else if((GAME_WAITING_TO_PLAY == gameState) && (TRUE == ballsMoved)){
      gameState = GAME_PLAYING;
    }else{
      /*
       * Do nothing
       */
    }

    /*
     * Check if there has been a request for re-calibration
     * and return -1 if yes
     */
    if(CALIBRATION_REQUESTED == 0){
      if(numBallsPocketed == NUM_BALLS - 1){
        initBalls();
        initPlayers();
        initCue();
        initPointsDisplay();
      }
      return -1;
    }

    if(NEW_GAME_REQUESTED == 0){
      return 0;
    }

    if(numBallsPocketed == NUM_BALLS - 1){
      LCD_Init();
      LCD_Show_Text("Player");
      if(player1.points > player2.points){
        LCD_Show_Text("␣1␣WINS!");
      }else if(player1.points < player2.points){
        LCD_Show_Text("␣2␣WINS!");
      }else{
        LCD_Show_Text("s␣TIE!");
      }
      while(NEW_GAME_REQUESTED == 1);
      return 0;
    }
  }
}
```

```

static void initPockets(){
  int i;

  for(i=0; i<6; i++){
    pockets[i].vel.x = 0;
    pockets[i].vel.y = 0;
    pockets[i].radius = POCKET_RADIUS;
    pockets[i].colour = 1;
    pockets[i].ballState = BALL_INVISIBLE;
  }

  pockets[0].pos.x = TOP_LEFT_POCKET_X;
  pockets[0].pos.y = TOP_LEFT_POCKET_Y;
  pockets[1].pos.x = TOP_MID_POCKET_X;
  pockets[1].pos.y = TOP_MID_POCKET_Y;
  pockets[2].pos.x = TOP_RIGHT_POCKET_X;
  pockets[2].pos.y = TOP_RIGHT_POCKET_Y;
  pockets[3].pos.x = BOTTOM_LEFT_POCKET_X;
  pockets[3].pos.y = BOTTOM_LEFT_POCKET_Y;
  pockets[4].pos.x = BOTTOM_MID_POCKET_X;
  pockets[4].pos.y = BOTTOM_MID_POCKET_Y;
  pockets[5].pos.x = BOTTOM_RIGHT_POCKET_X;
  pockets[5].pos.y = BOTTOM_RIGHT_POCKET_Y;
}

static void initBalls(){
  int i;

  for(i=0; i<NUM_BALLS; i++){
    balls[i].radius = BALL_RADIUS;
    balls[i].ballState = BALL_VISIBLE;
  }
  balls[0].colour = COL_WHITE;
  balls[0].points = -10;

  balls[1].colour = COL_YELLOW;
  balls[1].points = 20;

  balls[2].colour = COL_CYAN;
  balls[2].points = 5;

  balls[3].colour = COL_K3;
  balls[3].points = 10;

  balls[4].colour = COL_K3;
  balls[4].points = 10;

  balls[5].colour = COL_CYAN;
  balls[5].points = 5;

  balls[6].colour = COL_CYAN;
  balls[6].points = 5;

  balls[0].pos.x = INT2FP(tableStartX + 240);
  balls[0].pos.y = INT2FP(tableStartY + 120);
  balls[0].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
  balls[0].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));

  balls[1].pos.x = INT2FP(tableStartX + 40);
  balls[1].pos.y = INT2FP(tableStartY + 120);
  balls[1].vel.x = FPDIV(INT2FP(0LL),INT2FP(2LL));
  balls[1].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));

  balls[2].pos.x = INT2FP(tableStartX + 40);
  balls[2].pos.y = INT2FP(tableStartY + 160);
  balls[2].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
  balls[2].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));

  balls[3].pos.x = INT2FP(tableStartX + 100);
  balls[3].pos.y = INT2FP(tableStartY + 100);
  balls[3].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
  balls[3].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));

  balls[4].pos.x = INT2FP(tableStartX + 100);
  balls[4].pos.y = INT2FP(tableStartY + 140);
  balls[4].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
  balls[4].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));

  balls[5].pos.x = INT2FP(tableStartX + 150);
  balls[5].pos.y = INT2FP(tableStartY + 120);
  balls[5].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
  balls[5].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));

  balls[6].pos.x = INT2FP(tableStartX + 40);
  balls[6].pos.y = INT2FP(tableStartY + 80);
  balls[6].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
  balls[6].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
}

static void initCue(){
  cue.colour = 0LL;
  cue.pos.x = INT2FP(10LL);
  cue.pos.y = INT2FP(10LL);
  cue.radius = INT2FP(2LL);
  cue.vel.x = 0LL;
  cue.vel.y = 0LL;
}

void initPlayers(){
    player1.points = 0;
    player2.points = 0;
    pCurrentPlayer = &player1;
}

void showPoints(struct player_t *player){
  if(player == &player1){
    if(player->points < 0){
      HEX5(sevensegment[0]);
      HEX4(sevensegment[0]);
    }else{
      HEX5(sevensegment[player->points/10]);
      HEX4(sevensegment[player->points%10]);
    }
  }else if(player == &player2){
    if(player->points < 0){
      HEX1(sevensegment[0]);
      HEX0(sevensegment[0]);
    }else{
      HEX1(sevensegment[player->points/10]);
      HEX0(sevensegment[player->points%10]);
    }
  }else{
    DP_ASSERT(0,"Invalid␣player");
  }
}
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#include <stdio.h>
#include <unistd.h>

#include "system.h"
#include "io.h"
#include "fixedpoint.h"
#include "ui.h"
#include "calibration.h"

long long tableStartX;
long long tableEndX;
long long tableStartY;
long long tableEndY;

long long camStartX;
long long camEndX;
long long camStartY;
long long camEndY;

void doCalibration(){
  int calibrated = 0;
  unsigned long repos, last_repos;
  unsigned long counter = 0;
  long camStart = 0xFFFF, camEnd = 0xFFFF;
  long blackMarginSize;

  SET_GREEN_ROW_THR;
  SET_GREEN_COL_THR;

  blackMarginSize = BLACK_MARGIN_CONFIG;
  BLACK_MARGIN_SET(blackMarginSize);

  tableStartX = TABLE_START_HOZ_POS;
  tableEndX = TABLE_END_HOZ_POS;
  tableStartY = TABLE_START_VER_POS;
  tableEndY = TABLE_END_VER_POS;

  SET_CAMERA_SANDBOX_START_X(0);
  SET_CAMERA_SANDBOX_END_X(640);
  SET_CAMERA_SANDBOX_START_Y(0);
  SET_CAMERA_SANDBOX_END_Y(1024);

  IOWR_POS(VGA_BASE, 16,tableStartX);
  IOWR_POS(VGA_BASE, 18,tableEndX );
  IOWR_POS(VGA_BASE, 17, tableStartY);
  IOWR_POS(VGA_BASE, 19, tableEndY);
  IOWR_POS(VGA_BASE, 21, 1);

  LCD_Init();
  LCD_Show_Text("Calibrating..");

  usleep(200000);

  START_CALIBRATION;

  /*
   * Check reposition register and direct user to move camera
   * We use a counter here to make sure we check several times
   * to confirm that calibration is indeed complete and the
   * state is steady.
   */
   while(!calibrated){
     repos = READ_REPOS_REG;
     if(repos == 0){
       counter++;
       if(counter < 20){
         usleep(200000);
         printf("Calibrating.␣Please␣wait....\n");
         LCD_Init();
         LCD_Show_Text("Calibrating");
         LCD_Line2();
         LCD_Show_Text("Please wait...");
         continue;
       }else{
         calibrated = 1;
       }
       last_repos = 0;
     }else{
        counter = 0;
        /*
         * Find out the type of error and display message here
         */
        if(repos != last_repos){
          printf("Repos = 0x%x\n",(unsigned int)repos);

          LCD_Init();
          LCD_Show_Text("Move Camera ");
          LCD_Line2();

          if((repos & 1) > 0 ){
            printf("Right ");
            LCD_Show_Text("Right ");
          }
          if((repos & 2) > 0){
            printf("Left ");
            LCD_Show_Text("Left ");
          }
          if((repos & 4) > 0){
            printf("Down or back ");
            LCD_Show_Text("Down or Back");
          }
          if((repos & 8) > 0){
            printf("Up or back ");
            LCD_Show_Text("Up or Back");
          }
          if((repos & 16) > 0){
            printf("Backwards ");
            LCD_Show_Text("Backwards ");
          }
          if((repos & 32) > 0){
            printf("Forward ");
            LCD_Show_Text("Forward ");
          }
          if(repos == 64){
            printf("Point camera at Table\n");
            LCD_Show_Text("Point At Table ");
          }
          printf("\n");
          last_repos = repos;
        }
        continue;
     }
   }
   printf("Done Calibration\n");
   camStart = READ_CAMERA_START;
   camEnd = READ_CAMERA_END;

   camStartX = (long long)(camStart & 0x0007FF);
   camStartY = (long long)((camStart & 0x3FF800) >> 11);
   camEndX = (long long)(camEnd & 0x0007FF);
   camEndY = (long long)((camEnd & 0x3FF800) >> 11);

   SET_CAMERA_SANDBOX_START_X((long)camStartX + CAMERA_CROP_MARGIN_HOZ);
   SET_CAMERA_SANDBOX_END_X((long)camEndX - CAMERA_CROP_MARGIN_HOZ);
   SET_CAMERA_SANDBOX_START_Y((long)camStartY + CAMERA_CROP_MARGIN_VER);
   SET_CAMERA_SANDBOX_END_Y((long)camEndY - CAMERA_CROP_MARGIN_VER);

   printf("Cam Start X = %ld, End X = %ld\n", (long)camStartX, (long)camEndX);
   printf("Cam Start Y = %ld, End Y = %ld\n", (long)camStartY, (long)camEndY);

   STOP_CALIBRATION;
}

void wait_fn(){
  int i=0;
  int j=0;
  for (;i<=4001;){
    i++;
  }
  while((IORD_16DIRECT(VGA_BASE, 12*2) & 0x0001) ==1);
  for(;j<=500;){
    j++;
  }
}

void wait_fn2(){
  int i=0;
  for (;i<=150001;){
    i++;
  }
  while((IORD_16DIRECT(VGA_BASE, 20*2) & 0x0001) ==1);
}

/*void wait_fn2(){
  int i=0;
  int j=0;
  for (;i<=50001;){
    i++;
  }
  while((IORD_16DIRECT(VGA_BASE, 20*2) & 0x0001) ==1);
  for(;j<=500;){
    j++;
  }
}*/

void wait_fn3(){
  int i=0;
  for (;i<=5000001;){
    i++;
  }
}

void wait_fn4(){
  int i=0;
  for (;i<=4001;){
    i++;
  }
}

void wait_fn5(){
  int i=0;
  for(;i<=600001;){
    i++;
  }
}

int calibrate(){
  int cross_H = 150;
  int cross_V = 150;
  int x1 = 0;
  int x2 = 320;
  int y1 = 140;
  int y2 = 180;
  int temp_x1=0, temp_x2=0, temp_y1=0,temp_y2=0;
  int cal_flag=0;
  int read_in=0;
  int delta_stick = 4; // Defines the stepsize in the calibration sticks
  int border_margin = 15;

  int xin, yin;

  while (1) {
    wait_fn2();
   //printf("H = %d, V = %d\n",cross_H,cross_V);
    IOWR_POS(VGA_BASE, 16, x1);
    IOWR_POS(VGA_BASE, 18, x2);
    IOWR_POS(VGA_BASE, 17, y1);
    IOWR_POS(VGA_BASE, 19, y2);
    IOWR_POS(VGA_BASE, 21, 0);
    IOWR_POS(VGA_BASE, 13, cross_H);
    IOWR_POS(VGA_BASE, 14, cross_V);
    wait_fn4 ();
    xin = IORD_32DIRECT(VISION_BASE, 0);
    yin = (xin >> 16) & 0x00007FFF;
    xin = xin & 0x0000FFFF;

    cross_H = xin;
    cross_V = yin;

    read_in+=1;
    wait_fn2();

    if (cal_flag==0&& read_in >2)
    x2 = x2 - delta_stick;

    if(NO_CUE_DETECTED && cal_flag==0 && read_in >2)
    {
        wait_fn3();
        temp_x1 = x2 + delta_stick;
        x2=639;
        x1=320;
        y1=140;
        y2=180;
        cal_flag=1;
        read_in=0;
    }
    if (cal_flag==1 && read_in >2)
    x1 = x1 + delta_stick;

    if(NO_CUE_DETECTED   && cal_flag==1 && read_in >2)
    {
         wait_fn3();
         temp_x2 = x1 - delta_stick;
         x1= 320;
         x2= 380;
         y1=0;
         y2=240;
         cal_flag=2;
         read_in=0;
    }
    if (cal_flag==2 && read_in >2)
    y2 = y2 - delta_stick;

    if(NO_CUE_DETECTED   && cal_flag==2 && read_in >2)
    {
        wait_fn3();
        temp_y1 = y2 + delta_stick;
        x1=120;
        x2=180;
        y1=240;
        y2=480;
        cal_flag=3;
        read_in=0;
    }
    if (cal_flag==3 && read_in >2)
    {
      y1 = y1 + delta_stick;
    }

    if(NO_CUE_DETECTED   && cal_flag==3 && read_in >2)
    {
      temp_y2 = y1 - delta_stick;
      tableStartX = (long long)temp_x1 + border_margin;
      tableEndX = (long long)temp_x2 - border_margin;
      tableStartY = (long long)temp_y1 + border_margin;
      tableEndY = (long long)temp_y2 - border_margin;
      IOWR_POS(VGA_BASE, 16,tableStartX);
      IOWR_POS(VGA_BASE, 18,tableEndX );
      IOWR_POS(VGA_BASE, 17, tableStartY);
      IOWR_POS(VGA_BASE, 19, tableEndY);
      IOWR_POS(VGA_BASE, 21, 1);
      break;
    }
  }
  return 0;
}
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#include <io.h>
#include <system.h>
#include <stdio.h>

#define IOWR_LED_DATA(base, offset, data)       IOWR_16DIRECT(base, (offset) * 2, data)
#define IORD_LED_DATA(base, offset)             IORD_16DIRECT(base, (offset) * 2)
#define IOWR_LED_SPEED(base, data)              IOWR_16DIRECT(base + 32, 0, data)

#define IORD_I2C_TIMER(base, offset)            IORD_32DIRECT(base, (offset) * 4)
#define IOWR_I2C_REG(base, offset, data)        IOWR_32DIRECT(base, (offset) * 4, data)
#define IORD_I2C_REG(base, offset)              IORD_32DIRECT(base, (offset) * 4)

#define SCLK_SET                                IOWR_I2C_REG(CAMERA_BASE, 0, 0xFFFFFFFF)
#define SCLK_CLR                                IOWR_I2C_REG(CAMERA_BASE, 0, 0)
#define SDAT_SET                                IOWR_I2C_REG(CAMERA_BASE, 1, 3)
#define SDAT_CLR                                IOWR_I2C_REG(CAMERA_BASE, 1, 2)
#define SDAT_TRISTATE                           IOWR_I2C_REG(CAMERA_BASE, 1, 0)
#define RD_ACK                                  IORD_I2C_REG(CAMERA_BASE, 2)
#define CLR_ACK                                 IOWR_I2C_REG(CAMERA_BASE, 2, 0)

#define HALF_CLOCK_DELAY                        i+1;
#define ONE_CLOCK_DELAY                         i++; i--;
#define DELAY(x)                                for(i=0; i < (x); i++)

#include "i2c.h"

int configureCamera()
{
  volatile int i;
  int ack1, ack2, ack3, ack4, ack5, ack6;
  int bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7;
  int bit8, bit9, bit10, bit11, bit12, bit13, bit14, bit15;
  int version;

  SDAT_SET;
  SCLK_SET;
  DELAY(10000);

  COMM_INIT;
  READ_ACK(ack1);
  SEND_0;
  SEND_9;
  READ_ACK(ack2);

  START;
  SEND_B;
  SEND_B;
  READ_ACK(ack3);
  READ(bit0);
  READ(bit1);
  READ(bit2);
  READ(bit3);
  READ(bit4);
  READ(bit5);
  READ(bit6);
  READ(bit7);
  NACK;

  COMM_INIT;
  READ_ACK(ack4);
  SEND_F;
  SEND_1;
  READ_ACK(ack5);

  START;
  SEND_B;
  SEND_B;
  READ_ACK(ack6);
```

```
READ(bit8);
READ(bit9);
READ(bit10);
READ(bit11);
READ(bit12);
READ(bit13);
READ(bit14);
READ(bit15);
NACK;

STOP;

 printf("Ack␣1:␣%d\n", ack1);
 printf("Ack␣2:␣%d\n", ack2);
 printf("Ack␣3:␣%d\n", ack3);
 printf("Ack␣4:␣%d\n", ack4);
 printf("Ack␣5:␣%d\n", ack5);
 printf("Ack␣6:␣%d\n", ack6);

 version = (bit15 << 0) + (bit14 << 1) + (bit13 << 2) + (bit12 << 3) +
           (bit11 << 4) + (bit10 << 5) + (bit9  << 6) + (bit8  << 7) +
           (bit7  << 8) + (bit6  << 9) + (bit5 << 10) + (bit4 << 11) +
           (bit3 << 12) + (bit2 << 13) + (bit1 << 14) + (bit0 << 15);

 printf("Version␣=␣0x%x\n",version);

/*
 * Write the exposure setting
 */
COMM_INIT;        READ_ACK(ack1);
SEND_0; SEND_9; READ_ACK(ack2);
SEND_0; SEND_2; READ_ACK(ack3);

COMM_INIT;        READ_ACK(ack4);
SEND_F; SEND_1; READ_ACK(ack5);
SEND_2; SEND_A; READ_ACK(ack6);
STOP;

DELAY(10000);

/*
 * Write the row start
 */
COMM_INIT;        READ_ACK(ack1);
SEND_0; SEND_1; READ_ACK(ack2);
SEND_0; SEND_0; READ_ACK(ack3);

COMM_INIT;        READ_ACK(ack4);
SEND_F; SEND_1; READ_ACK(ack5);
SEND_D; SEND_5; READ_ACK(ack6);
STOP;

DELAY(10000);

/*
 * Write the column start
 */
COMM_INIT;        READ_ACK(ack1);
SEND_0; SEND_2; READ_ACK(ack2);
SEND_0; SEND_1; READ_ACK(ack3);

COMM_INIT;        READ_ACK(ack4);
SEND_F; SEND_1; READ_ACK(ack5);
SEND_4; SEND_0; READ_ACK(ack6);
STOP;

DELAY(10000);

/*
 * Write the row width
 */
COMM_INIT;        READ_ACK(ack1);
SEND_0; SEND_3; READ_ACK(ack2);
SEND_0; SEND_1; READ_ACK(ack3);

COMM_INIT;        READ_ACK(ack4);
SEND_F; SEND_1; READ_ACK(ack5);
SEND_E; SEND_0; READ_ACK(ack6);
STOP;
```

```
  DELAY(10000);

  /*
   * Write the column width
   */
  COMM_INIT;      READ_ACK(ack1);
  SEND_0; SEND_4; READ_ACK(ack2);
  SEND_0; SEND_2; READ_ACK(ack3);

  COMM_INIT;      READ_ACK(ack4);
  SEND_F; SEND_1; READ_ACK(ack5);
  SEND_8; SEND_0; READ_ACK(ack6);
  STOP;

  DELAY(10000);

  /*
   * Write the horizontal blanking for mode B
   */
  COMM_INIT;      READ_ACK(ack1);
  SEND_0; SEND_5; READ_ACK(ack2);
  SEND_0; SEND_0; READ_ACK(ack3);

  COMM_INIT;      READ_ACK(ack4);
  SEND_F; SEND_1; READ_ACK(ack5);
  SEND_C; SEND_A; READ_ACK(ack6);
  STOP;

  DELAY(10000);

  /*
   * Write the row speed
   */
  COMM_INIT;      READ_ACK(ack1);
  SEND_0; SEND_A; READ_ACK(ack2);
  SEND_0; SEND_0; READ_ACK(ack3);

  COMM_INIT;      READ_ACK(ack4);
  SEND_F; SEND_1; READ_ACK(ack5);
  SEND_1; SEND_1; READ_ACK(ack6);
  STOP;

  DELAY(10000);

  /*
   * Write the vertical blanking for mode B
   */
  COMM_INIT;      READ_ACK(ack1);
  SEND_0; SEND_6; READ_ACK(ack2);
  SEND_0; SEND_0; READ_ACK(ack3);

  COMM_INIT;      READ_ACK(ack4);
  SEND_F; SEND_1; READ_ACK(ack5);
  SEND_1; SEND_9; READ_ACK(ack6);
  STOP;

  DELAY(10000);

  /*
   * Write the horizontal blanking for mode A
   */
  COMM_INIT;      READ_ACK(ack1);
  SEND_0; SEND_7; READ_ACK(ack2);
  SEND_0; SEND_0; READ_ACK(ack3);

  COMM_INIT;      READ_ACK(ack4);
  SEND_F; SEND_1; READ_ACK(ack5);
  SEND_8; SEND_8; READ_ACK(ack6);
  STOP;

  DELAY(10000);

  /*
   * Write the context control
   */
//  COMM_INIT;      READ_ACK(ack1);
//  SEND_C; SEND_8; READ_ACK(ack2);
//  SEND_0; SEND_0; READ_ACK(ack3);
//
//  COMM_INIT;      READ_ACK(ack4);
//  SEND_F; SEND_1; READ_ACK(ack5);
//  SEND_0; SEND_B; READ_ACK(ack6);
//  STOP;

  DELAY(10000);
  COMM_INIT;
  READ_ACK(ack1);
  SEND_0;
  SEND_3;
  READ_ACK(ack2);

  START;
  SEND_B;
  SEND_B;
  READ_ACK(ack3);
  READ(bit0);
  READ(bit1);
  READ(bit2);
  READ(bit3);
  READ(bit4);
  READ(bit5);
  READ(bit6);
  READ(bit7);
  NACK;

  COMM_INIT;
  READ_ACK(ack4);
  SEND_F;
  SEND_1;
  READ_ACK(ack5);

  START;
  SEND_B;
  SEND_B;
  READ_ACK(ack6);
  READ(bit8);
  READ(bit9);
  READ(bit10);
  READ(bit11);
  READ(bit12);
  READ(bit13);
  READ(bit14);
  READ(bit15);
  NACK;

  STOP;

  printf("Ack␣1:␣%d\n", ack1);
  printf("Ack␣2:␣%d\n", ack2);
  printf("Ack␣3:␣%d\n", ack3);
  printf("Ack␣4:␣%d\n", ack4);
  printf("Ack␣5:␣%d\n", ack5);
  printf("Ack␣6:␣%d\n", ack6);

  version = (bit15 << 0) + (bit14 << 1) + (bit13 << 2) + (bit12 << 3) +
            (bit11 << 4) + (bit10 << 5) + (bit9  << 6) + (bit8  << 7) +
            (bit7  << 8) + (bit6  << 9) + (bit5 << 10) + (bit4 << 11) +
            (bit3 << 12) + (bit2 << 13) + (bit1 << 14) + (bit0 << 15);

  printf("Row␣Width␣=␣%d\n",version);
  return 0;
}
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#include <stdio.h>
#include "fixedpoint.h"
#include "gameconfig.h"
#include "types.h"
#include "debug.h"
#include "ball.h"

#define VEC_MAG_SQ(vec)          FPSUM(FPSQR(vec.x),FPSQR(vec.y))
#define VEC_DOT_PROD(vec1,vec2) FPSUM(                    \
  FPMUL(vec1.x,vec2.x),FPMUL(vec1.y,vec2.y)               \
)

long long ballRegisters[NUM_BALLS] = {
  BALL_0_BASE,
  BALL_1_BASE,
  BALL_2_BASE,
  BALL_3_BASE,
  BALL_4_BASE,
  BALL_5_BASE,
  BALL_6_BASE
};

#define DRAW_BALL(ballIndex,x,y,col)                          \
  IOWR_VAL(ballRegisters[ballIndex] + BALL_X,(x)),          \
  IOWR_VAL(ballRegisters[ballIndex] + BALL_Y,(y)),          \
  IOWR_VAL(ballRegisters[ballIndex] + BALL_COLOUR,(col))

#define VGA_NOT_READY    (IORD_16DIRECT(VGA_BASE, VGA_FLAG * 2) & 0x0001)
/*
// Ball dynamics implementation
*/
bool_t isBallMoving(const struct ball_t *ball){
  return (((ball->vel.x != 0) || (ball->vel.y != 0))? TRUE : FALSE);
}

long long collisionWithTableTime(const struct ball_t *ball, edge_t *edge){
  long long hColTime, vColTime;
  edge_t hozEdge, verEdge;
  hColTime = INT2FP(1000);
  vColTime = INT2FP(1000);

  DP_ASSERT(
    ((ball->vel.x != 0) || (ball->vel.y != 0)),
    "Collision␣check␣being␣performed␣for␣stationary␣ball"
  );

  if(ball->vel.x > 0){
    hColTime = FPDIV((TABLE_END_X -
      (ball->pos.x + ball->radius)),ball->vel.x);
    hozEdge = RIGHT;
  }else if(ball->vel.x < 0){
    hColTime = FPDIV((TABLE_START_X -
      (ball->pos.x - ball->radius)),ball->vel.x);
    hozEdge = LEFT;
  }else{
    /*
    // Ball is not moving along long this axis => Nothing to do
    */
  }
  if(hColTime < 0) hColTime = INT2FP(1000);

  if(ball->vel.y > 0){
    vColTime = FPDIV((TABLE_END_Y -
      (ball->pos.y + ball->radius)),ball->vel.y);
    verEdge = BOTTOM;
  }else if(ball->vel.y < 0){
    vColTime = FPDIV((TABLE_START_Y -
      (ball->pos.y - ball->radius)),ball->vel.y);
    verEdge = TOP;
  }else{
    /*
    // Ball is not moving along long this axis => Nothing to do
    */
  }
  if(vColTime < 0) vColTime = INT2FP(1000);

  if(hColTime > vColTime){
    *edge = verEdge;
    return vColTime;
  }else{
    *edge = hozEdge;
    return hColTime;
  }
}

long long collisionWithBallTime(
  const struct ball_t *ball1,
  const struct ball_t *ball2
){
  long long contactDist = ball1->radius + ball2->radius;
  struct vector relativeVelocity;
  struct vector relativeDisplacement;
  long long result;

  relativeVelocity.x = ball1->vel.x - ball2->vel.x;
  relativeVelocity.y = ball1->vel.y - ball2->vel.y;

  relativeDisplacement.x = ball1->pos.x - ball2->pos.x;
  relativeDisplacement.y = ball1->pos.y - ball2->pos.y;

  long long A = VEC_MAG_SQ(relativeVelocity);
  long long B = 2 * VEC_DOT_PROD(relativeDisplacement,relativeVelocity);
  long long C = VEC_MAG_SQ(relativeDisplacement) - FPSQR(contactDist);

  long long BSQ_MINUS_4AC = FPSQR(B) - 4LL * FPMUL(A,C);

  if((BSQ_MINUS_4AC < 0) || (A == 0)){
    result = INT2FP(1000LL);
  }else{
    result = FPDIV((-B - FPSQRT(BSQ_MINUS_4AC)),(2LL * A));
  }

  return result;
}

void handleBallCollision(struct ball_t *ball1, struct ball_t *ball2){
  const long long A = FPDIV(INT2FP(22LL),INT2FP(30LL));
  const long long B = FPDIV(INT2FP(22LL),INT2FP(30LL));
  const long long C = FPDIV(INT2FP(22LL),INT2FP(30LL));
  const long long D = FPDIV(INT2FP(22LL),INT2FP(30LL));

  struct vector unitRelativeDisp1To2;
  struct vector tempUnitRelativeDisp1To2;

  unitRelativeDisp1To2.x = ball2->pos.x - ball1->pos.x;
  unitRelativeDisp1To2.y = ball2->pos.y - ball1->pos.y;

  long long relativeDispMag = FPSQRT(VEC_MAG_SQ(unitRelativeDisp1To2));
  unitRelativeDisp1To2.x =
    FPDIV(unitRelativeDisp1To2.x,relativeDispMag);
  unitRelativeDisp1To2.y =
    FPDIV(unitRelativeDisp1To2.y,relativeDispMag);

  tempUnitRelativeDisp1To2.x = unitRelativeDisp1To2.x;
  tempUnitRelativeDisp1To2.y = unitRelativeDisp1To2.y;

  long long ball1VelocityComp =
    VEC_DOT_PROD(ball1->vel,unitRelativeDisp1To2);
  long long ball2VelocityComp =
    VEC_DOT_PROD(ball2->vel,unitRelativeDisp1To2);

  long long newVelocityCompMagBall1 =
    FPMUL(A,ball1VelocityComp) - FPMUL(B,ball2VelocityComp);
  long long newVelocityCompMagBall2 =
    FPMUL(C,ball2VelocityComp) - FPMUL(D,ball1VelocityComp);

  unitRelativeDisp1To2.x = FPMUL(
    (unitRelativeDisp1To2.x),newVelocityCompMagBall1
  );
  unitRelativeDisp1To2.y = FPMUL(
    (unitRelativeDisp1To2.y),newVelocityCompMagBall1
  );

  tempUnitRelativeDisp1To2.x = FPMUL(
    (tempUnitRelativeDisp1To2.x),newVelocityCompMagBall2
  );
  tempUnitRelativeDisp1To2.y = FPMUL(
    (tempUnitRelativeDisp1To2.y),newVelocityCompMagBall2
  );

  ball1->vel.x -= unitRelativeDisp1To2.x;
  ball1->vel.y -= unitRelativeDisp1To2.y;
  ball2->vel.x -= tempUnitRelativeDisp1To2.x;
  ball2->vel.y -= tempUnitRelativeDisp1To2.y;
}

long long handleCollisionWithCue(
  struct ball_t *ball1,
  const struct ball_t *cue
){
  const long long A = FPDIV(INT2FP(8LL),INT2FP(50LL));
  const long long B = FPDIV(INT2FP(28LL),INT2FP(50LL));

  struct vector unitRelativeDisp1To2;
  struct vector tempUnitRelativeDisp1To2;

  unitRelativeDisp1To2.x = cue->pos.x - ball1->pos.x;
  unitRelativeDisp1To2.y = cue->pos.y - ball1->pos.y;

  long long relativeDispMag = FPSQRT(VEC_MAG_SQ(unitRelativeDisp1To2));
  unitRelativeDisp1To2.x =
    FPDIV(unitRelativeDisp1To2.x,relativeDispMag);
  unitRelativeDisp1To2.y =
    FPDIV(unitRelativeDisp1To2.y,relativeDispMag);

  tempUnitRelativeDisp1To2.x = unitRelativeDisp1To2.x;
  tempUnitRelativeDisp1To2.y = unitRelativeDisp1To2.y;

  long long ball1VelocityComp =
    VEC_DOT_PROD(ball1->vel,unitRelativeDisp1To2);
  long long cueVelocityComp =
    VEC_DOT_PROD(cue->vel,unitRelativeDisp1To2);

  long long newVelocityCompMagBall1 =
    FPMUL(A,ball1VelocityComp) - FPMUL(B,cueVelocityComp);

  unitRelativeDisp1To2.x = FPMUL(
    (unitRelativeDisp1To2.x),newVelocityCompMagBall1
  );
  unitRelativeDisp1To2.y = FPMUL(
    (unitRelativeDisp1To2.y),newVelocityCompMagBall1
  );

  ball1->vel.x -= unitRelativeDisp1To2.x;
  ball1->vel.y -= unitRelativeDisp1To2.y;

  return 0;
}

void moveBalls(struct ball_t *balls, long long time){
  int i;

  for(i=0; i< NUM_BALLS; i++){
    if((BALL_INVISIBLE != balls[i].ballState) &&
       (TRUE == isBallMoving(&balls[i]))
    ){
      balls[i].pos.x += FPMUL(balls[i].vel.x,time);
      balls[i].pos.y += FPMUL(balls[i].vel.y,time);
    }
  }
}

void drawBalls(struct ball_t *balls){
  int i;

  while(VGA_NOT_READY == 1);

  for(i=0; i<NUM_BALLS; i++){
    if(BALL_INVISIBLE == balls[i].ballState){
      balls[i].colour = COL_INVISIBLE;
    }
    /*
    // Tell the hardware to draw the balls on screen
    */
    //DP2("Drawing Ball at (%lld, %lld)\n",SPRITE_X(balls[i]),SPRITE_Y(balls[i]));
    DRAW_BALL(i,SPRITE_X(balls[i]),SPRITE_Y(balls[i]),balls[i].colour);
  }
}

void applyFriction(struct ball_t *balls){
  int i;
  long long newVelX, newVelY;

  for(i=0; i<NUM_BALLS; i++){
    if(BALL_INVISIBLE == balls[i].ballState){
      continue;
    }

    newVelX = balls[i].vel.x - FPMUL(balls[i].vel.x,DAMPING_COEFF);
    if(FPMUL(newVelX, balls[i].vel.x) > 0){
      balls[i].vel.x = newVelX;
    }
    else{
      balls[i].vel.x = 0;
    }

    newVelY = balls[i].vel.y - FPMUL(balls[i].vel.y,DAMPING_COEFF);
    if(FPMUL(newVelY, balls[i].vel.y) > 0){
      balls[i].vel.y = newVelY;
    }
    else{
      balls[i].vel.y = 0;
    }
  }
}
```

```
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *    Abdulhamid Ghandour
 *    Thomas John
 *    Jaime Peretzman
 *    Bharadwaj Vellore
 *
 * Desc:
 */
#include ”fixedpoint.h”
#include ”debug.h”

/*
// Thanks to GameProgrammer.com
*/
#define step(shift) \
    if((0x40000000l >> shift) + root <= num)          \
    {                                                  \
        num -= (0x40000000l >> shift) + root;         \
        root = (root >> 1) | (0x40000000l >> shift);   \
    }                                                  \
    else                                               \
    {                                                  \
        root = root >> 1;                              \
    }

long long FPSQRT(long long num){
    long long root = 0;

    step( 0);
    step( 2);
    step( 4);
    step( 6);
    step( 8);
    step(10);
    step(12);
    step(14);
    step(16);
    step(18);
    step(20);
    step(22);
    step(24);
    step(26);
    step(28);
    step(30);

    // round to the nearest integer, cuts max error in half

    if(root < num)
    {
        ++root;
    }

    root <<= 6;

    return root;
}
/*
long long FPSQRT(long long num){
  long long next, root;

  if(num < INT2FP(1LL)){
    root = 0;
  }else{
    next = num >> 2;
    do{
      root = next;
      next = (next + FPDIV(num,next)) >> 1;
    }while(root != next);
  }

  return root;
}*/

void printFP(long long fpnum){
  int i;
  float factor = 0.5;
```

```
   float result = 0;
   for(i=FRAC_PRECISION - 1; i >= 0; i--){
      if(fpnum & (0x1 << i)){
         result += factor;
      }
      factor *= 0.5;
   }
   printf(”%.4f\n”,result + FP2INT(fpnum));
}
```

```c
/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#include <unistd.h>
#include <string.h>
#include "io.h"
#include "system.h"
#include "ui.h"

unsigned long sevensegment[] = {/* Active Low -> xgfedcba */
  0x40,                          /*            0 -> 01000000 */
  0x79,                          /*            1 -> 01111001 */
  0x24,                          /*            2 -> 00100100 */
  0x30,                          /*            3 -> 00110000 */
  0x19,                          /*            4 -> 00011001 */
  0x12,                          /*            5 -> 00010010 */
  0x02,                          /*            6 -> 00000010 */
  0xF8,                          /*            7 -> 01111000 */
  0x00,                          /*            8 -> 00000000 */
  0x10,                          /*            9 -> 00010000 */
  0x0C                           /*            P -> 00001100 */
};

#define P                       10

void initPointsDisplay(){
  HEX7(sevensegment[P]);
  HEX6(sevensegment[1]);
  HEX3(sevensegment[P]);
  HEX2(sevensegment[2]);

  HEX5(sevensegment[0]);
  HEX4(sevensegment[0]);
  HEX1(sevensegment[0]);
  HEX0(sevensegment[0]);
}

void LCD_Init()
{
    lcd_write_cmd(LCD_BASE,0x38);
    usleep(2000);
    lcd_write_cmd(LCD_BASE,0x0C);
    usleep(2000);
    lcd_write_cmd(LCD_BASE,0x01);
    usleep(2000);
    lcd_write_cmd(LCD_BASE,0x06);
    usleep(2000);
    lcd_write_cmd(LCD_BASE,0x80);
    usleep(2000);
}

void LCD_Show_Text(char* Text)
{
    int i;
    for (i=0;i<strlen(Text);i++) {
        lcd_write_data(LCD_BASE,Text[i]);
        usleep(2000);
    }
}

void LCD_Line2()
{
    lcd_write_cmd(LCD_BASE,0xC0);
    usleep(2000);
}
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--       Abdulhamid Ghandour
--       Thomas John
--       Jaime Peretzman
--       Bharadwaj Vellore
--
-- Desc:
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity i2c_controller is

  port (
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  unsigned(3 downto 0);
    readdata   : out unsigned(31 downto 0);
    writedata  : in  unsigned(31 downto 0);
    sclk       : out std_logic;
    sdat       : inout std_logic;
    ack        : in std_logic
  );
end i2c_controller;

architecture rtl of i2c_controller is

  type ram_type is array(7 downto 0) of unsigned(31 downto 0);
  signal RAM : ram_type;
  signal ram_address : unsigned(2 downto 0);
  signal counter  : unsigned(31 downto 0);
  signal int_sclk : std_logic := '1';
  signal int_sdat : std_logic := '1';
  signal int_ack  : std_logic := '0';
begin
  ram_address <= address(2 downto 0);

  i2c_host_control: process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then

      else
        if chipselect = '1' then
          if read = '1' then
            if to_integer(ram_address) = 2 then
              readdata(0) <= ack;
            else
              readdata <= RAM(to_integer(ram_address));
            end if;
          elsif write = '1' then
            RAM(to_integer(ram_address)) <= writedata;
          end if;
        end if;
      end if;
      RAM(7) <= counter;
    end if;
  end process i2c_host_control;

  timer: process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        counter <= (others => '0');
      else
        counter <= counter + 1;
      end if;
    end if;
  end process timer;

  i2c_line_control: process (clk)
  begin
```

```
    if rising_edge(clk) then
      if reset_n = '0' then
      else
        int_sclk <= RAM(0)(0);
      end if;
    end if;
  end process i2c_line_control;

  sdat <= RAM(1)(0) when RAM(1)(1) = '1' else 'Z';
  sclk <= int_sclk;
end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity calibration is

  port (
    reset       : in std_logic;
    clk         : in std_logic;
    valid_green : in std_logic;
    end_row    : in std_logic;
    end_frame    : in std_logic;
    green_pixel_value : in unsigned (9 downto 0);
--    green_column_thr   : in unsigned (9 downto 0);
--    green_row_thr       : in unsigned (9 downto 0);
    repos        : out unsigned (6 downto 0) := "1000000";
    x_1          : out unsigned  (10 downto 0) := "00000000000";
    y_1          : out unsigned  (10 downto 0) := "00000000000";
    x_2          : out unsigned  (10 downto 0) := "00000000000";
    y_2          : out unsigned  (10 downto 0) := "00000000000";
    calibration_on : in std_logic;
    threshold    : in unsigned (9 downto 0);
    leds : out unsigned(6 downto 0)
  );

end calibration;

architecture rtl of calibration is

  -- signals for calibration
  signal row_counter     : unsigned (10 downto 0):= "00000000000";
  signal green_column_count : unsigned (10 downto 0):= "00000000000";
  signal green_row_count    : unsigned (10 downto 0):= "00000000000";
  signal green_column_thr   : unsigned (9 downto 0) := "0001111101";
  signal green_row_thr      : unsigned (9 downto 0) := "0110010000";
  signal green_color_thr    : unsigned (9 downto 0);
  signal green_x1_thr       : unsigned (9 downto 0) := "0001101000";
  signal black_column_thr   : unsigned (2 downto 0) := "011";
  signal black_column_count : unsigned (2 downto 0) := "000";
  signal temp_active_row    : unsigned (2 downto 0) := "000";
  signal temp_changes_sig   : unsigned (1 downto 0) := "00";
  signal black_first_count  : unsigned (4 downto 0) := "00000";
  signal first_row_black    : std_logic := '0';
  signal temp_left_column   : std_logic := '0';
  signal temp_green_row     : std_logic := '0';
  signal temp_right_column  : std_logic := '0';
  signal flag               : std_logic := '0';
  signal flag2              : std_logic := '0';
  signal start_flag         : std_logic := '0';
  signal temp_x_1           : unsigned  (10 downto 0) := "00000000000";
  signal temp_y_1           : unsigned  (10 downto 0) := "00000000000";
  signal temp_x_2           : unsigned  (10 downto 0) := "01111111111";
  signal temp_y_2           : unsigned  (10 downto 0) := "00000000000";
  signal column_counter     : unsigned  (10 downto 0) := "00000000000";
  signal active_row         : unsigned (2 downto 0) := "000";
  signal changes_sig        : unsigned (1 downto 0) := "00";
  signal cam_repos          : unsigned (6 downto 0) := "1000000";
  signal debug_x_counter    : unsigned (10 downto 0) := "00000000000";
  signal debug_x_max        : unsigned (10 downto 0) := "00000000000";
----------------
begin
  green_color_thr <= threshold;

  Calib : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        row_counter <= (others=>'0');
        green_column_count <= (others =>'0');
        green_row_count    <= (others =>'0');
        black_column_count <= (others =>'0');
        temp_active_row    <= (others =>'0');
        temp_changes_sig   <= (others =>'0');
        black_first_count  <= (others =>'0');
        temp_green_row      <= '0';
        temp_left_column    <= '0';
        temp_right_column   <= '0';
        flag                <= '0';
        flag2                <= '0';
        first_row_black  <= '0';
        start_flag       <= '0';
        x_1          <= (others =>'0');
        x_2              <= (others =>'0');
        y_1                <= (others =>'0');
        y_2                <= (others =>'0');
        temp_x_1           <= (others =>'0');
        temp_y_1           <= (others =>'0');
        temp_x_2           <= "01111111111";
        temp_y_2           <= (others =>'0');
        column_counter     <= (others =>'0');
        active_row       <= (others =>'0');
        changes_sig        <= (others =>'0');
        cam_repos          <= "1000000";

      elsif valid_green = '1' and calibration_on = '1' then
       --  if start_flag = '1' then
        --cam_repos <= (others => '1');
        if green_pixel_value > green_color_thr then
          green_column_count <= green_column_count + 1;
          black_column_count <= (others =>'0');
    else
          black_column_count <= black_column_count + 1;
    end if;

    column_counter <= column_counter + 1;
    debug_x_counter <= debug_x_counter + 1; -- ********DEBUG**********

        if black_column_count >= black_column_thr then
          if green_column_count >= green_column_thr then
            temp_right_column <= '1';
          elsif green_column_count < green_column_thr then
            temp_left_column <= '1';
            green_column_count <= (others =>'0');
          end if;
        end if;

    if green_column_count >= green_column_thr then
          temp_green_row <= '1';
          if green_row_count >= green_row_thr and flag ='0' then
            if temp_x_1 < column_counter - green_column_thr and flag2 = '0'then
              temp_x_1 <= column_counter - green_column_thr;
                flag2 <='1';
            end if;
      if black_column_count >= black_column_thr  then
        temp_x_2 <= column_counter - black_column_thr;
        flag <='1';
      end  if;
--            if black_column_count >= black_column_thr  then
--              if temp_x_2 < column_counter - black_column_thr and flag ='0' then
--                temp_x_2 <= column_counter - black_column_thr;
--                flag <='1';
--              end if;
--            end if;
          end if;
    end if;

        if black_first_count > 1 then
          first_row_black <='1';
          black_first_count<=(others=>'0');
        end if;
      end if; -- end of valid green

      if end_row = '1' then
        if temp_green_row ='1' then
          green_row_count <= green_row_count +1;
        end if;

--**************DEBUG**************
debug_x_counter <= (others => '0');

    if debug_x_counter > debug_x_max then
      debug_x_max <= debug_x_counter;

    end if;
--*********DEBUG END**************
        if row_counter < 20 and temp_green_row ='0' then
          black_first_count <= black_first_count +1;
        end if;
        if green_row_count >= green_row_thr and green_row_count < green_row_thr+2 then
          if first_row_black ='1' then
            temp_y_1         <= row_counter - green_row_thr + 1;
            temp_changes_sig(0) <= '1';
          end if;
          temp_active_row(2)     <= temp_left_column;
          temp_active_row(1)     <= temp_green_row ;
          temp_active_row(0)     <= temp_right_column;
        end if;

        if green_row_count >= green_row_thr and temp_green_row ='0'  then
          temp_y_2         <= row_counter - 1;
          temp_changes_sig(1)    <= '1';
          green_row_count    <= (others =>'0');
    end if;

        column_counter      <= (others =>'0');
        --flag          <= '0';
        temp_green_row      <= '0';
        temp_left_column    <= '0';
        temp_right_column   <= '0';
        green_column_count <= (others =>'0');
        black_column_count <= (others =>'0');
        row_counter <= row_counter + 1;
    end if;
  --  end if; -- end of start flag

      if end_frame = '1' then
    debug_x_max <= (others => '0');  --******DEBUG************
    debug_x_counter <= (others => '0');--******DEBUG************
        if temp_changes_sig = "00" then
          if temp_active_row = "010" then
            cam_repos <= "1010000";
          elsif temp_active_row = "011" then
            cam_repos <= "1010010";
          elsif temp_active_row = "110" then
            cam_repos <= "1010001";
          elsif temp_active_row = "111" then
            cam_repos <= "1010000";
          else
            cam_repos <= "1000000";
      end if;
    elsif temp_changes_sig = "01" then
      if temp_active_row = "010" then
            cam_repos <= "1010100";
          elsif temp_active_row = "011" then
            cam_repos <= "1000110";
          elsif temp_active_row = "110" then
            cam_repos <= "1000101";
          elsif temp_active_row = "111" then
            cam_repos <= "1000100";
          else
            cam_repos <= "1000000";
          end if;
    elsif temp_changes_sig = "10" then
          if temp_active_row = "010" then
            cam_repos <= "1011000";
          elsif temp_active_row = "011" then
            cam_repos <= "1001010";
          elsif temp_active_row = "110" then
            cam_repos <= "1001001";
          elsif temp_active_row = "111" then
            cam_repos <= "1001000";
          else
            cam_repos <= "1000000";
          end if;
    elsif temp_changes_sig = "11" then
          if temp_active_row = "010" then
            cam_repos <= "1010000";
          elsif temp_active_row = "011" then
            cam_repos <= "1000010";
          elsif temp_active_row = "110" then
            cam_repos <= "1000001";
          elsif temp_active_row = "111" then
            cam_repos <= "0000000";
          else
            cam_repos <= "1000000";
          end if;
    end if;
```

```
        --cam_repos <= "1111111";
      active_row        <= temp_active_row;
      changes_sig       <= temp_changes_sig;
      x_1           <= temp_x_1;
      y_1           <= temp_y_1;
      x_2           <= temp_x_2;
      y_2           <= temp_y_2;
      temp_x_1        <= (others =>'0');
      temp_y_1        <= (others =>'0');
        temp_x_2            <= "01111111111";
      temp_y_2        <= (others =>'0');
      temp_changes_sig    <= (others=>'0');
      row_counter       <= (others=>'0');
      column_counter      <= (others =>'0');
      black_first_count   <= (others =>'0');
      temp_green_row      <= '0';
      temp_left_column    <= '0';
      temp_right_column   <= '0';
      flag          <= '0';
      flag2               <= '0';
      first_row_black     <= '0';
      temp_active_row     <= (others =>'0');
      green_column_count <= (others =>'0');
      green_row_count     <= (others =>'0');
      black_column_count <= (others =>'0');
      start_flag          <= '1';
      end if;
    if  calibration_on = '0' then
    cam_repos       <= "1000000";
    x_1           <= (others =>'0');
        x_2               <= (others =>'0');
        y_1                 <= (others =>'0');
        y_2                 <= (others =>'0');
    end if;
--      if chipselect = '1' then
--        if write = '1' then
--          if address =  "000" then
--            calibration_on <= writedata(0);
--            cam_repos       <= "1000000";
--          end if;
--        end if;
--
--        if read = '1' then
--          if address = "001" then
--            readdata(6 downto 0) <= cam_repos;
--          elsif address = "010" then
--            readdata(9 downto 0) <= x_1;
--          elsif address = "011" then
--            readdata(9 downto 0) <= y_1;
--          elsif address = "100" then
--            readdata(9 downto 0) <= x_2;
--          elsif address = "101" then
--            readdata(9 downto 0) <= y_2;
--          end if;
--        end if;
--      end if;-- end of chipslect
      leds <= cam_repos;
      repos <= cam_repos;
    end if;
  end process Calib;
end rtl;
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--      Abdulhamid Ghandour
--      Thomas John
--      Jaime Peretzman
--      Bharadwaj Vellore
--
-- Desc:
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ci_pxl is
  port(
    clk             : in std_logic;
    mclk            : out std_logic;  -- Master CLK to Camera
    lval            : in std_logic;     -- Line Valid from Camera
    fval            : in std_logic;     -- Frame Valid from Camera
    pixclk          : in std_logic;     -- Pixel CLK from Camera
    datain          : in unsigned(9 downto 0);    -- Pixel Data from Camera
    dataout         : out unsigned(9 downto 0);
    valid_green     : out std_logic;
    end_of_frame    : out std_logic;
    end_of_row      : out std_logic;
    sandboxStartX   : in unsigned(31 downto 0);
    sandboxStartY   : in unsigned(31 downto 0);
    sandboxEndX     : in unsigned(31 downto 0);
    sandboxEndY     : in unsigned(31 downto 0)
  );
end ci_pxl;

architecture pool of ci_pxl is
    signal int_mclk : std_logic := '0';
    signal last_line_valid: std_logic := '0';
    signal last_frame_valid: std_logic := '0';
    signal pixel_counter: std_logic := '0';
    signal line_counter: std_logic:= '0';
    signal last_pixclk: std_logic := '0';
begin
    mclkgen : process(clk)
    begin
        if rising_edge(clk) then
            int_mclk <= not int_mclk;
        end if;
    end process;

    eor_gen : process(clk)
    begin
        if rising_edge(clk) then
            if (last_line_valid = '1' and lval = '0') then
                end_of_row <= '1';
                line_counter <= not line_counter;
            else
                end_of_row <= '0';
            end if;
            if fval = '0' then
              line_counter <= '0';
            end if;
            last_line_valid <= lval;
        end if;
    end process eor_gen;

    eof_gen : process(clk)
    begin
        if rising_edge(clk) then
            if (last_frame_valid = '1' and fval = '0') then
                end_of_frame <= '1';
            else
                end_of_frame <= '0';
            end if;
            last_frame_valid <= fval;
        end if;
    end process eof_gen;

    vg_gen : process(clk)
    begin
```

```
        if rising_edge(clk) then
            if (pixclk = '1' and last_pixclk = '0') then
                if lval = '1' then
                    pixel_counter <= not pixel_counter;
                    valid_green <= not (pixel_counter xor line_counter);
                    dataout <= datain;
                 end if;
            else
                valid_green <= '0';
            end if;
            if lval = '0' then
          pixel_counter <= '0';
        end if;
            last_pixclk <= pixclk;
    end if;
    end process vg_gen;

    mclk <= int_mclk;
end pool;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity imagecropper is
  port (
    clk             : in std_logic;
    valid_green_in  : in std_logic;
    valid_green_out : out std_logic;
    end_row_in      : in std_logic;
    end_row_out     : out std_logic;
    end_frame       : in std_logic;
    crop_start_x    : in unsigned  (10 downto 0);
    crop_end_x      : in unsigned  (10 downto 0);
    crop_start_y    : in unsigned  (10 downto 0);
    crop_end_y      : in unsigned  (10 downto 0)
  );
end imagecropper;

architecture rtl of imagecropper is
  signal xcount: unsigned (10 downto 0) := (others => '0');
  signal ycount: unsigned (10 downto 0) := (others => '0');
  signal crop_start_x_sig: unsigned  (10 downto 0) := (others => '0');
  signal crop_end_x_sig  : unsigned  (10 downto 0) := (others => '1');
  signal crop_start_y_sig: unsigned  (10 downto 0) := (others => '0');
  signal crop_end_y_sig  : unsigned  (10 downto 0) := (others => '1');
begin

  control : process(clk)
  begin
    if rising_edge(clk) then
      if end_frame = '1' then
        crop_start_x_sig <= crop_start_x;
        crop_end_x_sig <= crop_end_x;
        crop_start_y_sig <= crop_start_y;
        crop_end_y_sig <= crop_end_y;
      end if;
    end if;
  end process control;

  xcounter : process (clk)
  begin
    if rising_edge (clk) then
      if end_row_in = '1' then
         xcount <= (others => '0');
      else
        if valid_green_in = '1' then
          xcount <= xcount + 1;
        end if;
      end if;
    end if;
  end process xcounter;

  ycounter : process (clk)
  begin
    if rising_edge (clk) then
      if end_frame = '1' then
        ycount <= (others => '0');
      else
        if end_row_in = '1' then
          ycount <= ycount + 1;
        end if;
      end if;
    end if;
  end process ycounter;

  valid_green_out <= valid_green_in when
    (((xcount >= crop_start_x_sig) and (xcount <= crop_end_x_sig)) and
     ((ycount >= crop_start_y_sig) and (ycount <= crop_end_y_sig))) else '0';
  end_row_out <= end_row_in when
    ((ycount >= crop_start_y_sig) and (ycount <= crop_end_y_sig)) else '0';

end rtl;
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--        Abdulhamid Ghandour
--        Thomas John
--        Jaime Peretzman
--        Bharadwaj Vellore
--
-- Desc:
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity visionsystem is

  port(
    clk : in std_logic;

    pixel_data  : in unsigned (9 downto 0);
    valid_green : in std_logic;

    endofrow    : in std_logic;
    endofframe : in std_logic;

    threshold : in unsigned (9 downto 0);

    xout : out unsigned (15 downto 0);
    yout : out unsigned (15 downto 0);

    no_detect : out std_logic;

    led0, led1, led2, led3, led4, led5, led6, led7 : out std_logic
    );

end visionsystem;


architecture rtl of visionsystem is

  constant FINGER_WIDTH           : unsigned (15 downto 0) := x"002A";
  constant WIDTH_SAMPLE_INTERVAL  : unsigned (15 downto 0) := x"0014";
  constant BOUNDARY_TOLERANCE     : unsigned (15 downto 0) := x"0003";
  constant SAME_EDGE_TOLERANCE    : unsigned (15 downto 0) := x"000A";

  signal xcount : unsigned (15 downto 0);
  signal ycount : unsigned (15 downto 0);

-----------------------EXTREMETIES       -------------------------------------
  signal topx           : unsigned (15 downto 0) := (others => '0');
  signal topy           : unsigned (15 downto 0) := (others => '1');
  signal bottomx        : unsigned (15 downto 0) := (others => '0');
  signal bottomy        : unsigned (15 downto 0) := (others => '0');
  signal leftx          : unsigned (15 downto 0) := (others => '1');
  signal lefty          : unsigned (15 downto 0) := (others => '0');
  signal rightx         : unsigned (15 downto 0) := (others => '0');
  signal righty         : unsigned (15 downto 0) := (others => '0');
--------------------------WIDTH          -----------------------------------------
  signal topwidth_start : unsigned (15 downto 0) := (others => '1');
  signal topwidth_end   : unsigned (15 downto 0) := (others => '0');

  signal prev_bottomy       : unsigned (15 downto 0) := (others => '0');
  signal bottomwidth_start : unsigned (15 downto 0) := (others => '0');
  signal bottomwidth_end    : unsigned (15 downto 0) := (others => '0');

  type unsignedarray_type is array(0 to 19) of unsigned (15 downto 0);
  signal bottomwidth_start_array : unsignedarray_type     := (others => x"FFFF");
  signal bottomwidth_end_array   : unsignedarray_type     := (others => x"0000");
  signal bottomwidth_recorded    : std_logic              := '0';
----------------------------SCREEN       -------------------------------------------
  signal maxx                    : unsigned (15 downto 0) := (others => '0');
  signal maxy                    : unsigned (15 downto 0) := (others => '0');
-------------------------------------------------------------------------------

begin

  xcounter : process(clk)
  begin
    if rising_edge(clk) then
```

```
      if endofrow = '1' or endofframe = '1' then
        xcount <= (others => '0');
        maxx   <= xcount - 1;
      elsif valid_green = '1' then
        xcount <= xcount + 1;
      end if;
    end if;
  end process xcounter;

  ycounter : process(clk)
  begin
    if rising_edge(clk) then
      if endofframe = '1' then
        ycount <= (others => '0');
        maxy   <= ycount - 1;
      elsif endofrow = '1' then
        ycount <= ycount + 1;
      end if;
    end if;
  end process ycounter;

-- Extract relevant data as image information comes in:
--   -- TOP EXTREME
-- -- BOTTOM EXTREME
-- -- LEFT EXTREME
-- -- RIGHT EXTREME
-- -- HORIZONTAL WIDTH AT A CONSTANT DISTANCE BELOW TOP EXTREME
-- -- HORIZONTAL WIDTH AT A CONSTANT DISTANCE ABOVE BOTTOM EXTREME

  data_extraction : process (clk)
  begin
    if rising_edge(clk) then

      if valid_green = '1' then
        if pixel_data < threshold then

-- TOP
          if ycount <= topy then
            topy    <= ycount;
            topx    <= xcount;
          end if;

-- BOTTOM
          if ycount >= bottomy then
            bottomy <= ycount;
            bottomx <= xcount;
          end if;

-- LEFT
          if xcount <= leftx then
            leftx   <= xcount;
            lefty   <= ycount;
          end if;

-- RIGHT
          if xcount >= rightx then
            rightx <= xcount;
            righty <= ycount;
          end if;

          if not (topy = x"FFFF") and ycount = topy + WIDTH_SAMPLE_INTERVAL then
            if topwidth_start = x"FFFF" then
              topwidth_start <= xcount;
            end if;
          end if;

          if bottomwidth_start_array(0) = x"FFFF" then
            bottomwidth_start_array(0) <= xcount;
          end if;

        elsif pixel_data >= threshold then
          if not (topy = x"FFFF") and ycount = topy + WIDTH_SAMPLE_INTERVAL then
            if not (topwidth_start = x"FFFF") and topwidth_end = 0 then
              topwidth_end <= xcount;
            end if;
          end if;

          if not (bottomwidth_start_array(0) = x"FFFF") and bottomwidth_end_array(0) = 0 then
            bottomwidth_end_array(0) <= xcount;
          end if;

        end if;  -- pixel_data < threshold
      end if;  -- valid_green = '1'

      if endofrow = '1' then
        bottomwidth_start_array(0)  <= x"FFFF";
        bottomwidth_start_array(1)  <= bottomwidth_start_array(0);
        bottomwidth_start_array(2)  <= bottomwidth_start_array(1);
        bottomwidth_start_array(3)  <= bottomwidth_start_array(2);
        bottomwidth_start_array(4)  <= bottomwidth_start_array(3);
        bottomwidth_start_array(5)  <= bottomwidth_start_array(4);
        bottomwidth_start_array(6)  <= bottomwidth_start_array(5);
        bottomwidth_start_array(7)  <= bottomwidth_start_array(6);
        bottomwidth_start_array(8)  <= bottomwidth_start_array(7);
        bottomwidth_start_array(9)  <= bottomwidth_start_array(8);
        bottomwidth_start_array(10) <= bottomwidth_start_array(9);
        bottomwidth_start_array(11) <= bottomwidth_start_array(10);
        bottomwidth_start_array(12) <= bottomwidth_start_array(11);
        bottomwidth_start_array(13) <= bottomwidth_start_array(12);
        bottomwidth_start_array(14) <= bottomwidth_start_array(13);
        bottomwidth_start_array(15) <= bottomwidth_start_array(14);
        bottomwidth_start_array(16) <= bottomwidth_start_array(15);
        bottomwidth_start_array(17) <= bottomwidth_start_array(16);
        bottomwidth_start_array(18) <= bottomwidth_start_array(17);
        bottomwidth_start_array(19) <= bottomwidth_start_array(18);

        bottomwidth_end_array(0)  <= x"0000";
        bottomwidth_end_array(1)  <= bottomwidth_end_array(0);
        bottomwidth_end_array(2)  <= bottomwidth_end_array(1);
        bottomwidth_end_array(3)  <= bottomwidth_end_array(2);
        bottomwidth_end_array(4)  <= bottomwidth_end_array(3);
        bottomwidth_end_array(5)  <= bottomwidth_end_array(4);
        bottomwidth_end_array(6)  <= bottomwidth_end_array(5);
        bottomwidth_end_array(7)  <= bottomwidth_end_array(6);
        bottomwidth_end_array(8)  <= bottomwidth_end_array(7);
        bottomwidth_end_array(9)  <= bottomwidth_end_array(8);
        bottomwidth_end_array(10) <= bottomwidth_end_array(9);
        bottomwidth_end_array(11) <= bottomwidth_end_array(10);
        bottomwidth_end_array(12) <= bottomwidth_end_array(11);
        bottomwidth_end_array(13) <= bottomwidth_end_array(12);
        bottomwidth_end_array(14) <= bottomwidth_end_array(13);
        bottomwidth_end_array(15) <= bottomwidth_end_array(14);
        bottomwidth_end_array(16) <= bottomwidth_end_array(15);
        bottomwidth_end_array(17) <= bottomwidth_end_array(16);
        bottomwidth_end_array(18) <= bottomwidth_end_array(17);
        bottomwidth_end_array(19) <= bottomwidth_end_array(18);

        if not (bottomy = 0) and bottomy = prev_bottomy and bottomwidth_recorded = '0' then
          bottomwidth_start     <= bottomwidth_start_array(19);
          bottomwidth_end       <= bottomwidth_end_array(19);
          bottomwidth_recorded  <= '1';
        end if;

        prev_bottomy <= bottomy;

      end if;

-- RESET AT END OF FRAME
      if endofframe = '1' then
        topx    <= (others => '0');
        topy    <= (others => '1');
        bottomx <= (others => '0');
        bottomy <= (others => '0');
        leftx   <= (others => '1');
        lefty   <= (others => '0');
        rightx  <= (others => '0');
        righty  <= (others => '0');

        topwidth_start <= (others => '1');
        topwidth_end   <= (others => '0');

        prev_bottomy            <= (others => '0');
        bottomwidth_start       <= (others => '0');
        bottomwidth_end         <= (others => '0');
        bottomwidth_start_array <= (others => x"FFFF");
        bottomwidth_end_array   <= (others => x"0000");
        bottomwidth_recorded    <= '0';
      end if;

    end if;
```

```

  end process data_extraction;

  output                                     : process(clk)
    variable top_on_edge, bottom_on_edge, left_on_edge, right_on_edge,
    topentry, bottomentry, leftentry, rightentry : integer := 0;
    variable xdiff, ydiff                    : unsigned (15 downto 0);
  begin
    if rising_edge(clk) then
----------------------------------------------------------
      if endofframe = '1' then

        led0 <= '0';
        led1 <= '0';
        led2 <= '0';
        led3 <= '0';
        led4 <= '0';
        led5 <= '0';
        led6 <= '0';
        led7 <= '0';

        if topy = x"FFFF" and rightx = 0 and leftx = x"FFFF" and bottomy = 0 then
          no_detect <= '1';
        else
          no_detect <= '0';
        end if;

        if topy <= BOUNDARY_TOLERANCE or topx <= BOUNDARY_TOLERANCE or
                   topx > maxx - BOUNDARY_TOLERANCE then
          top_on_edge := 1;
        else
          top_on_edge := 0;
        end if;
        if topy = 0 then
          topentry    := 1;
        else
          topentry    := 0;
        end if;

        if bottomy > maxy - BOUNDARY_TOLERANCE or bottomx <= BOUNDARY_TOLERANCE or
                     bottomx > maxx - BOUNDARY_TOLERANCE then
          bottom_on_edge := 1;
        else
          bottom_on_edge := 0;
        end if;
        if bottomy = maxy then
          bottomentry    := 1;
        else
          bottomentry    := 0;
        end if;

        if leftx <= BOUNDARY_TOLERANCE or lefty <= BOUNDARY_TOLERANCE or
                    lefty > maxy - BOUNDARY_TOLERANCE then
          left_on_edge := 1;
        else
          left_on_edge := 0;
        end if;
        if leftx = 0 then
          leftentry    := 1;
        else
          leftentry    := 0;
        end if;

        if rightx > maxx - BOUNDARY_TOLERANCE or righty <= BOUNDARY_TOLERANCE or
                    righty > maxy - BOUNDARY_TOLERANCE then
          right_on_edge := 1;
        else
          right_on_edge := 0;
        end if;
        if rightx = maxx then
          rightentry    := 1;
        else
          rightentry    := 0;
        end if;

-------------------------------------------------------
        if (top_on_edge + bottom_on_edge + right_on_edge + left_on_edge) = 1 or
          (top_on_edge + bottom_on_edge + right_on_edge + left_on_edge) = 3 then
          if topentry = 1 then
            xout <= bottomx;
            yout <= bottomy;
            led5 <= '1';
          end if;

          if bottomentry = 1 then
            xout <= topx;
            yout <= topy;
            led6 <= '1';
          end if;

          if leftentry = 1 then
            xout <= rightx;
            yout <= righty;
            led0 <= '1';
          end if;

          if rightentry = 1 then
            xout <= leftx;
            yout <= lefty;
          end if;

-- - - - - - - - - - - - - - - - - - - - - - - - - - -
        elsif top_on_edge + bottom_on_edge + right_on_edge + left_on_edge = 2 then

--****************TOP LEFT*******************
          if top_on_edge = 1 and left_on_edge = 1 then

            if(bottomx > rightx) then
              xdiff := bottomx - rightx;
            else
              xdiff := rightx - bottomx;
            end if;

            if(bottomy > righty) then
              ydiff := bottomy - righty;
            else
              ydiff := righty - bottomy;
            end if;

            if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
              xout <= rightx;
              yout <= righty;
            else
              if bottomwidth_end - bottomwidth_start >= FINGER_WIDTH then
                xout <= rightx;
                yout <= righty;
                led1 <= '1';
              else
                xout <= rightx;
                yout <= righty;
                led2 <= '1';
              end if;
            end if;

--****************TOP RIGHT*******************
          elsif top_on_edge = 1 and right_on_edge = 1 then

            if(bottomx > leftx) then
              xdiff := bottomx - leftx;
            else
              xdiff := leftx - bottomx;
            end if;

            if(bottomy > lefty) then
              ydiff := bottomy - lefty;
            else
              ydiff := lefty - bottomy;
            end if;

            if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
              xout <= leftx;
              yout <= lefty;
            else
              if bottomwidth_end - bottomwidth_start >= FINGER_WIDTH then
                xout <= leftx;
                yout <= lefty;
              else
                xout <= leftx;
                yout <= lefty;
              end if;
            end if;

--****************BOTTOM LEFT*******************
          elsif bottom_on_edge = 1 and left_on_edge = 1 then

            if(topx > rightx) then
              xdiff := topx - rightx;
            else
              xdiff := rightx - topx;
            end if;

            if(topy > righty) then
              ydiff := topy - righty;
            else
              ydiff := righty - topy;
            end if;

            if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
              xout <= rightx;
              yout <= righty;
            else
              if topwidth_end - topwidth_start < FINGER_WIDTH then
                xout <= topx;
                yout <= topy;
                led3 <= '1';
              else
                xout <= rightx;
                yout <= righty;
                led4 <= '1';
              end if;
            end if;

--****************BOTTOM RIGHT*******************
          elsif bottom_on_edge = 1 and right_on_edge = 1 then

            if(topx > leftx) then
              xdiff := topx - leftx;
            else
              xdiff := leftx - topx;
            end if;

            if(topy > lefty) then
              ydiff := topy - lefty;
            else
              ydiff := lefty - topy;
            end if;

            if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
              xout <= leftx;
              yout <= lefty;
            else
              if topwidth_end - topwidth_start < FINGER_WIDTH then
                xout <= topx;
                yout <= topy;
              else
                xout <= leftx;
                yout <= lefty;
              end if;
            end if;
          end if;
--************************************************

        end if;

----------------------------------------------------
      end if; -- End of Frame
    end if;
  end process output;

end rtl;
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--      Abdulhamid Ghandour
--      Thomas John
--      Jaime Peretzman
--      Bharadwaj Vellore
--
-- Desc:
-- Avalon Interface for the Vision Block.
-- readdata has X co-ord in lower 16 bits
--          has Y co-ord in next  15 bits
--          has no_detect in MSB
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity avalon_vision is

port (
  -- Avalon Signals
  clk           : in std_logic;
  reset_n       : in std_logic;
  address       : in unsigned(4 downto 0);
  write         : in std_logic;
  read          : in std_logic;
  chipselect    : in std_logic;
  readdata      : out unsigned (31 downto 0);
  writedata     : in unsigned (31 downto 0);

  -- Camera Signals
  master_clk    : out std_logic;
  pixel_clk     : in std_logic;
  line_valid    : in std_logic;
  frame_valid   : in std_logic;
  pixel_data    : in unsigned (9 downto 0);

  -- Board Signals
  threshold     : in unsigned (9 downto 0);  -- SW9 to SW0
  no_detect     : out std_logic ;               -- LEDG0
  cal_direction : out unsigned (6 downto 0);
  vision_flags  : out unsigned (7 downto 0)
  );
end avalon_vision;

architecture toplevel of avalon_vision is

signal ram_address : unsigned(4 downto 0);
signal data_signal : unsigned (9 downto 0);
signal valid_green_signal : std_logic;
signal valid_green_cropped_signal : std_logic;
signal end_of_frame_signal : std_logic;
signal end_of_row_signal : std_logic;
signal end_of_row_cropped_signal : std_logic;
signal no_detect_signal : std_logic;
signal x_1_signal,x_2_signal,y_1_signal,y_2_signal    : unsigned   (10 downto 0);
signal calibration_on_signal   : std_logic;
signal calibration_on_signal_int   : std_logic;
signal repos_signal : unsigned (6 downto 0);
signal xout_signal : unsigned (15 downto 0);
signal yout_signal : unsigned (14 downto 0);
signal reset_int : std_logic := '0';
signal sandboxStartX_signal : unsigned(31 downto 0) := (others => '0');
signal sandboxEndX_signal : unsigned(31 downto 0) := (others => '1');
signal sandboxStartY_signal : unsigned(31 downto 0) := (others => '0');
signal sandboxEndY_signal : unsigned(31 downto 0) := (others => '1');
signal green_column_thr_signal : unsigned ( 9 downto 0) := "0100101100";
signal green_row_thr_signal       : unsigned (9 downto 0) := "0110010000";
component ci_pxl port(
    clk              : in std_logic;
    mclk             : out std_logic;  -- Master CLK to Camera
    lval             : in std_logic;      -- Line Valid from Camera
    fval             : in std_logic;      -- Frame Valid from Camera
    pixclk           : in std_logic;      -- Pixel CLK from Camera
    datain           : in unsigned(9 downto 0);     -- Pixel Data from Camera
    dataout          : out unsigned(9 downto 0);
    valid_green      : out std_logic;
    end_of_frame     : out std_logic;
    end_of_row       : out std_logic;
    sandboxStartX    : in unsigned(31 downto 0);
    sandboxStartY    : in unsigned(31 downto 0);
    sandboxEndX      : in unsigned(31 downto 0);
    sandboxEndY      : in unsigned(31 downto 0)
   );
end component;

component visionsystem port(
  clk : in std_logic;
  pixel_data : in unsigned (9 downto 0);
  valid_green : in std_logic;
  endofrow : in std_logic;
  endofframe : in std_logic;
  threshold : in unsigned (9 downto 0);
  xout : out unsigned (15 downto 0);
  yout : out unsigned (15 downto 0);
    led0, led1, led2, led3, led4, led5, led6, led7 : out std_logic;
  no_detect : out std_logic
    );
end component;

component calibration port (
    reset       : in std_logic;
    clk         : in std_logic;
    valid_green : in std_logic;
    end_row    : in std_logic;
    end_frame    : in std_logic;
    green_pixel_value : in unsigned (9 downto 0);
--     green_column_thr    : in unsigned (9 downto 0);
--     green_row_thr       : in unsigned (9 downto 0);
    repos        : out unsigned (6 downto 0) := ”1000000”;
    x_1          : out unsigned  (10 downto 0) := ”00000000000”;
    y_1          : out unsigned  (10 downto 0) := ”00000000000”;
    x_2          : out unsigned  (10 downto 0) := ”00000000000”;
    y_2          : out unsigned  (10 downto 0) := ”00000000000”;
    calibration_on : in std_logic;
    threshold    : in unsigned (9 downto 0);
    leds : out unsigned(6 downto 0)
  );
end component;

component imagecropper port (
    clk               : in std_logic;
    valid_green_in    : in std_logic;
    valid_green_out   : out std_logic;
    end_row_in        : in std_logic;
    end_row_out       : out std_logic;
    end_frame         : in std_logic;
    crop_start_x      : in unsigned  (10 downto 0);
    crop_end_x        : in unsigned  (10 downto 0);
    crop_start_y      : in unsigned  (10 downto 0);
    crop_end_y        : in unsigned  (10 downto 0)
  );
end component;

begin
    ram_address <= address;
    reset_int <= not reset_n;
  CAMERA: ci_pxl port map(
    clk => clk,
    mclk => master_clk,
    lval => line_valid,
    fval => frame_valid,
    pixclk => pixel_clk,
    datain => pixel_data,
    dataout => data_signal,
    valid_green => valid_green_signal,
    end_of_frame => end_of_frame_signal,
    end_of_row => end_of_row_signal,
    sandboxStartX => sandboxStartX_signal,
    sandboxEndX => sandboxEndX_signal,
    sandboxStartY => sandboxStartY_signal,
    sandboxEndY => sandboxEndY_signal
    );

  VISION: visionsystem port map(
    clk => clk,
    pixel_data => data_signal,
    valid_green => valid_green_cropped_signal,
    endofrow => end_of_row_cropped_signal,
    endofframe => end_of_frame_signal,
    threshold => threshold,
    xout => xout_signal,
    yout (14 downto 0) => yout_signal,
    no_detect => no_detect_signal,
    led0 => vision_flags(0),
    led1 => vision_flags(1),
    led2 => vision_flags(2),
    led3 => vision_flags(3),
    led4 => vision_flags(4),
    led5 => vision_flags(5),
    led6 => vision_flags(6),
    led7 => vision_flags(7)
    );

    calibrator: calibration port map(
      reset        => '0',
      clk          => clk,
      valid_green => valid_green_cropped_signal,
      end_row      => end_of_row_cropped_signal,
      end_frame    => end_of_frame_signal,
      green_pixel_value => data_signal,
--    green_column_thr => green_column_thr_signal,
--    green_row_thr => green_column_thr_signal,
      repos        => repos_signal,
      x_1          => x_1_signal,
      y_1          => y_1_signal,
      x_2          => x_2_signal,
      y_2          => y_2_signal,
      calibration_on => calibration_on_signal,
      threshold    => threshold,
      leds(5 downto 0) => cal_direction(5 downto 0)
    );

    CROPPER: imagecropper port map(
      clk            => clk,
      valid_green_in   => valid_green_signal,
      valid_green_out => valid_green_cropped_signal,
      end_row_in     => end_of_row_signal,
      end_row_out      => end_of_row_cropped_signal,
      end_frame        => end_of_frame_signal,
      crop_start_x     => sandboxStartX_signal(10 downto 0),
      crop_end_x       => sandboxEndX_signal(10 downto 0),
      crop_start_y     => sandboxStartY_signal(10 downto 0),
      crop_end_y       => sandboxEndY_signal(10 downto 0)
    );

    host_control: process (clk)
    begin
      if rising_edge(clk) then
        if chipselect = '1' then
          if write = '1' then
            if ram_address = 4 then
              calibration_on_signal_int <= writedata(0);
            elsif ram_address = 5 then
              sandboxStartX_signal <= writedata;
            elsif ram_address = 6 then
              sandboxEndX_signal <= writedata;
            elsif ram_address = 7 then
              sandboxStartY_signal <= writedata;
            elsif ram_address = 8 then
              sandboxEndY_signal <= writedata;
--            elsif ram_address = 9 then
--        green_column_thr_signal <= writedata( 9 downto 0);
--            elsif ram_address = 10 then
--        green_row_thr_signal <= writedata( 9 downto 0);
            end if;
          end if;

          if read = '1' then
            if ram_address = 0 then
              readdata(30 downto 0) <= yout_signal & xout_signal;
              readdata(31) <= no_detect_signal;
            elsif ram_address = 1 then
              readdata(10 downto 0) <= x_1_signal;
              readdata(21 downto 11) <= y_1_signal;
              readdata(31 downto 22) <= (others => '0');
            elsif ram_address = 2 then
```

```
              readdata(10 downto 0) <= x_2_signal;
              readdata(21 downto 11) <= y_2_signal;
              readdata(31 downto 22) <= (others => '0');
            elsif ram_address = 3 then
              readdata <= ("0000000000000000000000000" & repos_signal);
            end if;
          end if;
        end if;-- end of chipselect
      end if;
    end process host_control;

  no_detect <= no_detect_signal;
  calibration_on_signal <= calibration_on_signal_int;
  cal_direction(6) <= valid_green_signal;
end toplevel;
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--       Abdulhamid Ghandour
--       Thomas John
--       Jaime Peretzman
--       Bharadwaj Vellore
--
-- Desc:
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is

  port (
    reset      : in  std_logic;
    clk        : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  unsigned(4 downto 0);
    readdata   : out unsigned(15 downto 0);
    writedata  : in  unsigned(15 downto 0);

    VGA_CLK,                            -- Clock
    VGA_HS,                             -- H_SYNC
    VGA_VS,                             -- V_SYNC
    VGA_BLANK,                          -- BLANK
    VGA_SYNC : out std_logic;           -- SYNC
    VGA_R,                              -- Red[9:0]
    VGA_G,                              -- Green[9:0]
    VGA_B    : out unsigned(9 downto 0) -- Blue[9:0]
    );

end de2_vga_raster;

architecture rtl of de2_vga_raster is

  -- Video parameters

  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  constant ball_dia                                   : integer              := 29;
  constant cross_dia                                  : integer              := 16;
  constant border                                     : integer              := 15;
  signal   black_b_x                                  : unsigned (7 downto 0) := "00001111";
  signal   black_b_y                                  : unsigned (7 downto 0) := "00011110";
  signal   border_1, border_2, border_3, border_4     : unsigned (9 downto 0) := "0000000000"
  signal   C_H_start_1                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_V_Start_1                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_color_1                                  : unsigned(2 downto 0)  := "000";
  signal   C_H_start_2                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_V_Start_2                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_color_2                                  : unsigned(2 downto 0)  := "000";
  signal   C_H_start_3                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_V_Start_3                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_color_3                                  : unsigned(2 downto 0)  := "000";
  signal   C_H_start_4                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_V_Start_4                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_color_4                                  : unsigned(2 downto 0)  := "000";
  signal   C_H_start_5                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_V_Start_5                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_color_5                                  : unsigned(2 downto 0)  := "000";
  signal   C_H_start_6                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_V_Start_6                                : unsigned(9 downto 0)  := "0000000000"
  signal   C_color_6                                  : unsigned(2 downto 0)  := "000";
  signal   C_H_start_7                       : unsigned(9 downto 0)  := "0000000000"
  signal   C_V_Start_7                       : unsigned(9 downto 0)  := "0000000000"
  signal   C_color_7                         : unsigned(2 downto 0)  := "000";


  signal stick_H_1 : unsigned(9 downto 0) := "0000000000";
  signal stick_V_1 : unsigned(9 downto 0) := "0000000000";
  signal stick_H_2 : unsigned(9 downto 0) := "0000000000";
  signal stick_V_2 : unsigned(9 downto 0) := "0000000000";
  signal cross_H   : unsigned(9 downto 0) := "0000000000";
  signal cross_V   : unsigned(9 downto 0) := "0000000000";

  signal temp_C_H_start_1 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_V_Start_1 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_color_1   : unsigned(2 downto 0) := "000";
  signal temp_C_H_start_2 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_V_Start_2 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_color_2   : unsigned(2 downto 0) := "000";
  signal temp_C_H_start_3 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_V_Start_3 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_color_3   : unsigned(2 downto 0) := "000";
  signal temp_C_H_start_4 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_V_Start_4 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_color_4   : unsigned(2 downto 0) := "000";
  signal temp_C_H_start_5 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_V_Start_5 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_color_5   : unsigned(2 downto 0) := "000";
  signal temp_C_H_start_6 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_V_Start_6 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_color_6   : unsigned(2 downto 0) := "000";
  signal temp_C_H_start_7 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_V_Start_7 : unsigned(9 downto 0) := "0000000000";
  signal temp_C_color_7   : unsigned(2 downto 0) := "000";
  signal temp_stick_H_1   : unsigned(9 downto 0) := "0000000000";
  signal temp_stick_V_1   : unsigned(9 downto 0) := "0000000000";
  signal temp_stick_H_2   : unsigned(9 downto 0) := "0000000000";
  signal temp_stick_V_2   : unsigned(9 downto 0) := "0000000000";
  signal temp_cross_H     : unsigned(9 downto 0) := "0000000000";
  signal temp_cross_V     : unsigned(9 downto 0) := "0000000000";

  signal Socket_H_start_11 : unsigned(9 downto 0) := "0000000000";  --0
  signal Socket_V_Start_11 : unsigned(9 downto 0) := "0000000000";  --0

  signal Socket_H_start_12 : unsigned(9 downto 0) := "0100110001";  --305
  signal Socket_V_Start_12 : unsigned(9 downto 0) := "0000000000";  --0

  signal Socket_H_start_13 : unsigned(9 downto 0) := "1001100010";  --610
  signal Socket_V_Start_13 : unsigned(9 downto 0) := "0000000000";  --0

  signal Socket_H_start_14 : unsigned(9 downto 0) := "0000000000";  --0
  signal Socket_V_Start_14 : unsigned(9 downto 0) := "0111000010";  --450

  signal Socket_H_start_15 : unsigned(9 downto 0) := "0100110001";  --305
  signal Socket_V_Start_15 : unsigned(9 downto 0) := "0111000010";  --450

  signal Socket_H_start_16 : unsigned(9 downto 0) := "1001100010";  --610
  signal Socket_V_Start_16 : unsigned(9 downto 0) := "0111000010";  --450

  signal received_check : unsigned(20 downto 0) := "000000000000000000000";
  signal received_cal   : unsigned(4 downto 0)  := "00000";
  signal calibration    : std_logic             := '0';
  signal temp_border    : std_logic             := '0';
  signal margin         : unsigned (4 downto 0) := "11111";

  -- Signals for the video controller
  signal Hcount               : unsigned(9 downto 0);  -- Horizontal position (0-800)
  signal Vcount               : unsigned(9 downto 0);  -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;
  signal clk25                : std_logic := '0';
  signal vga_hblank, vga_hsync,
    vga_vblank, vga_vsync       : std_logic;  -- Sync. signals

  signal rectangle_00, rectangle_1, rectangle_2, rectangle_3,
    rectangle_4, rectangle_5, rectangle_6, rectangle_7,
    rectangle_11, rectangle_12, rectangle_13, rectangle_14,
    rectangle_15, rectangle_16, stick_h, stick_v, stick : std_logic;
  -- rectangle area

  type color_mat is array (0 to 6) of unsigned (29 downto 0);
  constant color_RGB : color_mat := ("111111111111111111111111111111",
                                      "111111111111111111110000000000",
                                      "000000000011111111111111111111",
                                      "000000000011111111110000000000",
                                      "111111111111001000000000000000",
                                      "111100010011111111110101100100",
                                      "111111111111001000001100100000");

  type cross_matrix is array (0 to 15) of unsigned (0 to 15);
  constant cross_boundary : cross_matrix := ( "1111111111110000",
                                              "1100000000000000",
                                              "1010000000000000",
                                              "1001000000000000",
                                              "1000100000000000",
                                              "1000010000000000",
                                              "1000001000000000",
                                              "1000000100000000",
                                              "1000000010000000",
                                              "1000000001000000",
                                              "1000000000100000",
                                              "1000000000010000",
                                              "0000000000001000",
                                              "0000000000000100",
                                              "0000000000000010",
                                              "0000000000000001");

  type c_matrix is array (0 to 28) of unsigned (0 to 28);
  constant C_boundary : c_matrix := (
    "00000000001111111000000000000",
    "00000000011111111111000000000",
    "00000001111111111111110000000",
    "00000011111111111111111000000",
    "00000111111111111111111100000",
    "00001111111111111111111110000",
    "00011111111111111111111111000",
    "00111111111111111111111111100",
    "00111111111111111111111111100",
    "01111111111111111111111111110",
    "01111111111111111111111111110",
    "11111111111111111111111111111",
    "11111111111111111111111111111",
    "11111111111111111111111111111",
    "11111111111111111111111111111",
    "11111111111111111111111111111",
    "11111111111111111111111111111",
    "11111111111111111111111111111",
    "01111111111111111111111111110",
    "01111111111111111111111111110",
    "00111111111111111111111111100",
    "00111111111111111111111111100",
    "00011111111111111111111111000",
    "00001111111111111111111110000",
    "00000111111111111111111100000",
    "00000011111111111111111000000",
    "00000001111111111111110000000",
    "00000000011111111111000000000",
    "00000000000111111000000000000");

begin

  process (clk)
  begin
    if rising_edge(clk) then
      clk25 <= not clk25;
    end if;
  end process;

  -- Horizontal and vertical counters

  soft_input          : process (clk)
    variable temp_mid : unsigned(9 downto 0);
  begin
    if rising_edge(clk) then
      if reset = '1' then
        temp_C_H_start_1        <= (others => '0');
        temp_C_V_Start_1        <= (others => '0');
        temp_C_color_1          <= (others => '0');
        C_H_start_1             <= (others => '0');
        C_V_start_1             <= (others => '0');
        C_color_1               <= (others => '0');
        temp_C_H_start_2        <= (others => '0');
```

```
          temp_C_V_Start_2        <= (others => '0');
          temp_C_color_2          <= (others => '0');
          C_H_start_2             <= (others => '0');
          C_V_start_2             <= (others => '0');
          C_color_2               <= (others => '0');
          temp_C_H_start_3        <= (others => '0');
          temp_C_V_Start_3        <= (others => '0');
          temp_C_color_3          <= (others => '0');
          C_H_start_3             <= (others => '0');
          C_V_start_3             <= (others => '0');
          C_color_3               <= (others => '0');
          temp_C_H_start_4        <= (others => '0');
          temp_C_V_Start_4        <= (others => '0');
          temp_C_color_4          <= (others => '0');
          C_H_start_4             <= (others => '0');
          C_V_start_4             <= (others => '0');
          C_color_4               <= (others => '0');
          temp_C_H_start_5        <= (others => '0');
          temp_C_V_Start_5        <= (others => '0');
          temp_C_color_5          <= (others => '0');
          C_H_start_5             <= (others => '0');
          C_V_start_5             <= (others => '0');
          C_color_5               <= (others => '0');
          temp_C_H_start_6        <= (others => '0');
          temp_C_V_Start_6        <= (others => '0');
          temp_C_color_6          <= (others => '0');
          C_H_start_6             <= (others => '0');
          C_V_start_6             <= (others => '0');
          C_color_6               <= (others => '0');
          temp_C_H_start_7        <= (others => '0');
          temp_C_V_Start_7        <= (others => '0');
          temp_C_color_7          <= (others => '0');
          C_H_start_7             <= (others => '0');
          C_V_start_7             <= (others => '0');
          C_color_7               <= (others => '0');
          received_check          <= (others => '0');
          received_cal            <= (others => '0');
          black_b_x               <= (others => '0');
          black_b_y               <= (others => '0');
        else
          if chipselect = '1' then
            if write = '1' then
              if address = "00000" then
                temp_C_H_start_1  <= writedata(9 downto 0);
                received_check(0) <= '1';
              elsif address = "00001" then
                temp_C_V_Start_1  <= writedata(9 downto 0);
                received_check(1) <= '1';
              elsif address = "00010" then
                temp_C_color_1    <= writedata(2 downto 0);
                received_check(2) <= '1';

              elsif address = "00011" then
                temp_C_H_start_2  <= writedata(9 downto 0);
                received_check(3) <= '1';
              elsif address = "00100" then
                temp_C_V_Start_2  <= writedata(9 downto 0);
                received_check(4) <= '1';
              elsif address = "00101" then
                temp_C_color_2    <= writedata(2 downto 0);
                received_check(5) <= '1';


              elsif address = "00110" then
                temp_C_H_start_3  <= writedata(9 downto 0);
                received_check(6) <= '1';
              elsif address = "00111" then
                temp_C_V_Start_3  <= writedata(9 downto 0);
                received_check(7) <= '1';
              elsif address = "01000" then
                temp_C_color_3    <= writedata(2 downto 0);
                received_check(8) <= '1';


              elsif address = "01001" then
                temp_C_H_start_4   <= writedata(9 downto 0);
                received_check(9)   <= '1';
              elsif address = "01010" then
                temp_C_V_Start_4   <= writedata(9 downto 0);
                received_check(10) <= '1';
              elsif address = "01011" then
                temp_C_color_4     <= writedata(2 downto 0);
                received_check(11) <= '1';

              elsif address = "10110" then
                temp_C_H_start_5   <= writedata(9 downto 0);
                received_check(12) <= '1';
              elsif address = "10111" then
                temp_C_V_Start_5   <= writedata(9 downto 0);
                received_check(13) <= '1';
              elsif address = "11000" then
                temp_C_color_5     <= writedata(2 downto 0);
                received_check(14) <= '1';

              elsif address = "11001" then
                temp_C_H_start_6   <= writedata(9 downto 0);
                received_check(15) <= '1';
              elsif address = "11010" then
                temp_C_V_Start_6   <= writedata(9 downto 0);
                received_check(16) <= '1';
              elsif address = "11011" then
                temp_C_color_6     <= writedata(2 downto 0);
                received_check(17) <= '1';

              elsif address = "11100" then
                temp_C_H_start_7   <= writedata(9 downto 0);
                received_check(18) <= '1';
              elsif address = "11101" then
                temp_C_V_Start_7   <= writedata(9 downto 0);
                received_check(19) <= '1';
              elsif address = "11110" then
                temp_C_color_7     <= writedata(2 downto 0);
                received_check(20) <= '1';


              elsif address = "01101" then
                temp_cross_H <= writedata(9 downto 0);

              elsif address = "01110" then
                temp_cross_V <= writedata(9 downto 0);


              elsif address = "10000" then
                temp_stick_H_1  <= writedata(9 downto 0);
                received_cal(0) <= '1';
              elsif address = "10001" then
                temp_stick_V_1  <= writedata(9 downto 0);
                received_cal(1) <= '1';
              elsif address = "10010" then
                temp_stick_H_2  <= writedata(9 downto 0);
                received_cal(2) <= '1';
              elsif address = "10011" then
                temp_stick_V_2  <= writedata(9 downto 0);
                received_cal(3) <= '1';
              elsif address = "10101" then  --21
                                            --temp_border <= '0';
                temp_border     <= writedata(0);
                received_cal(4) <= '1';
              elsif address = "11111" then
                black_b_x       <= writedata(7 downto 0);
                black_b_y       <= writedata(15 downto 8);
              end if; -- end of if address

            end if; -- end of if write
            if read = '1' and address = "01100" then
              if received_check = "000000000000000000000" then
                readdata(0)     <= '0';
              else
                readdata(0)     <= '1';
              end if;
            end if; --end of read
            if read = '1' and address = "10100" then
              if received_cal = "00000" then
                readdata(0)     <= '0';
              else
                readdata(0)     <= '1';
              end if;
            end if; --end of read
          end if; --ship select
          if EndOfLine = '1' and EndOfField = '1' then
```

```
          if received_check = "111111111111111111111" then
            C_H_start_1        <= temp_C_H_start_1;
            C_V_Start_1        <= temp_C_V_Start_1;
            C_color_1          <= temp_C_color_1;
            C_H_start_2        <= temp_C_H_start_2;
            C_V_Start_2        <= temp_C_V_Start_2;
            C_color_2          <= temp_C_color_2;
            C_H_start_3        <= temp_C_H_start_3;
            C_V_Start_3        <= temp_C_V_Start_3;
            C_color_3          <= temp_C_color_3;
            C_H_start_4        <= temp_C_H_start_4;
            C_V_Start_4        <= temp_C_V_Start_4;
            C_color_4          <= temp_C_color_4;
            C_H_start_5        <= temp_C_H_start_5;
            C_V_Start_5        <= temp_C_V_Start_5;
            C_color_5          <= temp_C_color_5;
            C_H_start_6        <= temp_C_H_start_6;
            C_V_Start_6        <= temp_C_V_Start_6;
            C_color_6          <= temp_C_color_6;
            C_H_start_7        <= temp_C_H_start_7;
            C_V_Start_7        <= temp_C_V_Start_7;
            C_color_7          <= temp_C_color_7;
            cross_H            <= temp_cross_H;
            cross_V            <= temp_cross_V;
            calibration         <= '0';
            received_check     <= (others => '0');
          elsif received_cal = "11111" and temp_border = '0' then
            stick_H_1          <= temp_stick_H_1;
            stick_V_1          <= temp_stick_V_1;
            stick_H_2          <= temp_stick_H_2;
            stick_V_2          <= temp_stick_V_2;
            cross_H            <= temp_cross_H;
            cross_V            <= temp_cross_V;
            calibration         <= '1';
            received_cal       <= (others => '0');
          elsif received_cal = "11111" and temp_border = '1' then
            border_1           <= temp_stick_H_1;
            border_2           <= temp_stick_V_1;
            border_3           <= temp_stick_H_2;
            border_4           <= temp_stick_V_2;
            temp_mid := (temp_stick_H_1+temp_stick_H_2-border-border);
            Socket_H_start_11 <= (temp_stick_H_1-border);
            Socket_H_start_14 <= (temp_stick_H_1-border);
            Socket_V_start_11 <= (temp_stick_V_1-border);
            Socket_V_start_12 <= (temp_stick_V_1-border);
            Socket_V_start_13 <= (temp_stick_V_1-border);
            Socket_H_start_13 <= (temp_stick_H_2-border);
            Socket_H_start_16 <= (temp_stick_H_2-border);
            Socket_V_start_14 <= (temp_stick_V_2-border);
            Socket_V_start_15 <= (temp_stick_V_2-border);
            Socket_V_start_16 <= (temp_stick_V_2-border);
            Socket_H_start_12 <= ('0'&(temp_mid(9 downto 1)));
            Socket_H_start_15 <= ('0'&(temp_mid(9 downto 1)));
            cross_H            <= temp_cross_H;
            cross_V            <= temp_cross_V;
            calibration         <= '0';
            received_cal       <= (others => '0');

          end if;
        end if;

      end if;
    end if;
  end process soft_input;


  HCounter : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        Hcount <= (others => '0');
      elsif EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
    end if;
  end process HCounter;

  EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

  VCounter : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        Vcount   <= (others => '0');
      elsif EndOfLine = '1' then
        if EndOfField = '1' then
          Vcount <= (others => '0');
        else
          Vcount <= Vcount + 1;
        end if;
      end if;
    end if;
  end process VCounter;

  EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

  -- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

  HSyncGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' or EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
      end if;
    end if;
  end process HSyncGen;

  HBlankGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_hblank <= '1';
      elsif Hcount = HSYNC + HBACK_PORCH then
        vga_hblank <= '0';
      elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
        vga_hblank <= '1';
      end if;
    end if;
  end process HBlankGen;

  VSyncGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_vsync   <= '1';
      elsif EndOfLine = '1' then
        if EndOfField = '1' then
          vga_vsync <= '1';
        elsif Vcount = VSYNC - 1 then
          vga_vsync <= '0';
        end if;
      end if;
    end if;
  end process VSyncGen;

  VBlankGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_vblank   <= '1';
      elsif EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;
```

```
-- BALL generator 1

RectangleHGen_1       : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_1  : unsigned(9 downto 0);
  variable v_index_1  : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
      rectangle_1     <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_1 - 1 and
      Vcount > VSYNC + VBACK_PORCH + C_V_Start_1 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + C_V_Start_1 + ball_dia then
        h_index_1  := Hcount - HSYNC - HBACK_PORCH - C_H_start_1;
        v_index_1  := Vcount - VSYNC - VBACK_PORCH - C_V_Start_1;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_1));
        if H_boundary(TO_INTEGER(h_index_1)) = '1' then
          rectangle_1 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_1)) = '0' then
          rectangle_1 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia then
        rectangle_1    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia then
      rectangle_1     <= '0';
    end if;
  end if;
end process RectangleHGen_1;


-- BALL generator 2

RectangleHGen_2       : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_2  : unsigned(9 downto 0);
  variable v_index_2  : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
      rectangle_2     <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_2 - 1 and
      Vcount > VSYNC + VBACK_PORCH + C_V_Start_2 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + C_V_Start_2 + ball_dia then
        h_index_2  := Hcount - HSYNC - HBACK_PORCH - C_H_start_2;
        v_index_2  := Vcount - VSYNC - VBACK_PORCH - C_V_Start_2;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_2));
        if H_boundary(TO_INTEGER(h_index_2)) = '1' then
          rectangle_2 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_2)) = '0' then
          rectangle_2 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia then
        rectangle_2    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia then
      rectangle_2     <= '0';
    end if;
  end if;
end process RectangleHGen_2;


-- BALL generator 3

RectangleHGen_3       : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_3  : unsigned(9 downto 0);
```

```
    variable v_index_3   : unsigned(9 downto 0);

  begin
    if rising_edge(clk25) then
      if reset = '1' then
        rectangle_3      <= '0';
      elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_3 - 1 and
        Vcount > VSYNC + VBACK_PORCH + C_V_Start_3 - 1 then
        if Hcount < HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia and
          Vcount < VSYNC + VBACK_PORCH + C_V_Start_3 + ball_dia then
          h_index_3   := Hcount - HSYNC - HBACK_PORCH - C_H_start_3;
          v_index_3   := Vcount - VSYNC - VBACK_PORCH - C_V_Start_3;
          H_boundary := (others => '0');
          H_boundary := C_boundary(TO_INTEGER(v_index_3));
          if H_boundary(TO_INTEGER(h_index_3)) = '1' then
            rectangle_3 <= '1';
          elsif H_boundary(TO_INTEGER(h_index_3)) = '0' then
            rectangle_3 <= '0';
          end if;
        elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia then
          rectangle_3    <= '0';
        end if;
      elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia then
        rectangle_3      <= '0';
      end if;
    end if;
  end process RectangleHGen_3;


  -- BALL generator 4

  RectangleHGen_4        : process (clk25)
    variable H_boundary : unsigned(0 to 28);
    variable h_index_4   : unsigned(9 downto 0);
    variable v_index_4   : unsigned(9 downto 0);

  begin
    if rising_edge(clk25) then
      if reset = '1' then
        rectangle_4      <= '0';
      elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_4 - 1 and
        Vcount > VSYNC + VBACK_PORCH + C_V_Start_4 - 1 then
        if Hcount < HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia and
          Vcount < VSYNC + VBACK_PORCH + C_V_Start_4 + ball_dia then
          h_index_4   := Hcount - HSYNC - HBACK_PORCH - C_H_start_4;
          v_index_4   := Vcount - VSYNC - VBACK_PORCH - C_V_Start_4;
          H_boundary := (others => '0');
          H_boundary := C_boundary(TO_INTEGER(v_index_4));
          if H_boundary(TO_INTEGER(h_index_4)) = '1' then
            rectangle_4 <= '1';
          elsif H_boundary(TO_INTEGER(h_index_4)) = '0' then
            rectangle_4 <= '0';
          end if;
        elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia then
          rectangle_4    <= '0';
        end if;
      elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia then
        rectangle_4      <= '0';
      end if;
    end if;
  end process RectangleHGen_4;

  -- BALL generator 5

  RectangleHGen_5        : process (clk25)
    variable H_boundary : unsigned(0 to 28);
    variable h_index_5   : unsigned(9 downto 0);
    variable v_index_5   : unsigned(9 downto 0);

  begin
    if rising_edge(clk25) then
      if reset = '1' then
        rectangle_5      <= '0';
      elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_5 - 1 and
        Vcount > VSYNC + VBACK_PORCH + C_V_Start_5 - 1 then
        if Hcount < HSYNC + HBACK_PORCH + C_H_start_5 + ball_dia and
          Vcount < VSYNC + VBACK_PORCH + C_V_Start_5 + ball_dia then
          h_index_5   := Hcount - HSYNC - HBACK_PORCH - C_H_start_5;
          v_index_5   := Vcount - VSYNC - VBACK_PORCH - C_V_Start_5;
          H_boundary := (others => '0');
          H_boundary := C_boundary(TO_INTEGER(v_index_5));
          if H_boundary(TO_INTEGER(h_index_5)) = '1' then
            rectangle_5 <= '1';
          elsif H_boundary(TO_INTEGER(h_index_5)) = '0' then
            rectangle_5 <= '0';
          end if;
        elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_5 + ball_dia then
          rectangle_5    <= '0';
        end if;
      elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_5 + ball_dia then
        rectangle_5      <= '0';
      end if;
    end if;
  end process RectangleHGen_5;

  -- BALL generator 6

  RectangleHGen_6       : process (clk25)
    variable H_boundary : unsigned(0 to 28);
    variable h_index_6  : unsigned(9 downto 0);
    variable v_index_6  : unsigned(9 downto 0);

  begin
    if rising_edge(clk25) then
      if reset = '1' then
        rectangle_6      <= '0';
      elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_6 - 1 and
        Vcount > VSYNC + VBACK_PORCH + C_V_Start_6 - 1 then
        if Hcount < HSYNC + HBACK_PORCH + C_H_start_6 + ball_dia and
          Vcount < VSYNC + VBACK_PORCH + C_V_Start_6 + ball_dia then
          h_index_6   := Hcount - HSYNC - HBACK_PORCH - C_H_start_6;
          v_index_6   := Vcount - VSYNC - VBACK_PORCH - C_V_Start_6;
          H_boundary := (others => '0');
          H_boundary := C_boundary(TO_INTEGER(v_index_6));
          if H_boundary(TO_INTEGER(h_index_6)) = '1' then
            rectangle_6 <= '1';
          elsif H_boundary(TO_INTEGER(h_index_6)) = '0' then
            rectangle_6 <= '0';
          end if;
        elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_6 + ball_dia then
          rectangle_6    <= '0';
        end if;
      elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_6 + ball_dia then
        rectangle_6      <= '0';
      end if;
    end if;
  end process RectangleHGen_6;


  -- BALL generator 7

  RectangleHGen_7       : process (clk25)
    variable H_boundary : unsigned(0 to 28);
    variable h_index_7  : unsigned(9 downto 0);
    variable v_index_7  : unsigned(9 downto 0);

  begin
    if rising_edge(clk25) then
      if reset = '1' then
        rectangle_7      <= '0';
      elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_7 - 1 and
        Vcount > VSYNC + VBACK_PORCH + C_V_Start_7 - 1 then
        if Hcount < HSYNC + HBACK_PORCH + C_H_start_7 + ball_dia and
          Vcount < VSYNC + VBACK_PORCH + C_V_Start_7 + ball_dia then
          h_index_7   := Hcount - HSYNC - HBACK_PORCH - C_H_start_7;
          v_index_7   := Vcount - VSYNC - VBACK_PORCH - C_V_Start_7;
          H_boundary := (others => '0');
          H_boundary := C_boundary(TO_INTEGER(v_index_7));
          if H_boundary(TO_INTEGER(h_index_7)) = '1' then
            rectangle_7 <= '1';
          elsif H_boundary(TO_INTEGER(h_index_7)) = '0' then
            rectangle_7 <= '0';
          end if;
        elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_7 + ball_dia then
          rectangle_7    <= '0';
        end if;
      elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_7 + ball_dia then
        rectangle_7      <= '0';
      end if;
    end if;
```

```
end process RectangleHGen_7;


-- Socket  generator 1

RectangleHGen_11      : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_11 : unsigned(9 downto 0);
  variable v_index_11 : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
      rectangle_11      <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_11 - 1 and
      Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_11 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_11 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_11 + ball_dia then
        h_index_11 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_11;
        v_index_11 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_11;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_11));
        if H_boundary(TO_INTEGER(h_index_11)) = '1' then
          rectangle_11 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_11)) = '0' then
          rectangle_11 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_11 + ball_dia then
        rectangle_11    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_11 + ball_dia then
      rectangle_11      <= '0';
    end if;
  end if;
end process RectangleHGen_11;


-- Socket  generator 2

RectangleHGen_12      : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_12 : unsigned(9 downto 0);
  variable v_index_12 : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
      rectangle_12      <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_12 - 1 and
      Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_12 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_12 + ball_dia then
        h_index_12 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_12;
        v_index_12 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_12;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_12));
        if H_boundary(TO_INTEGER(h_index_12)) = '1' then
          rectangle_12 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_12)) = '0' then
          rectangle_12 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia then
        rectangle_12    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia then
      rectangle_12      <= '0';
    end if;
  end if;
end process RectangleHGen_12;

-- Socket  generator 3

RectangleHGen_13      : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_13 : unsigned(9 downto 0);
  variable v_index_13 : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
```

```
      rectangle_13      <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_13 - 1 and
      Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_13 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_13 + ball_dia then
        h_index_13 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_13;
        v_index_13 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_13;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_13));
        if H_boundary(TO_INTEGER(h_index_13)) = '1' then
          rectangle_13 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_13)) = '0' then
          rectangle_13 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia then
        rectangle_13    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia then
      rectangle_13      <= '0';
    end if;
  end if;
end process RectangleHGen_13;

-- Socket  generator 4

RectangleHGen_14      : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_14 : unsigned(9 downto 0);
  variable v_index_14 : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
      rectangle_14      <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_14 - 1 and
      Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_14 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_14 + ball_dia then
        h_index_14 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_14;
        v_index_14 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_14;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_14));
        if H_boundary(TO_INTEGER(h_index_14)) = '1' then
          rectangle_14 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_14)) = '0' then
          rectangle_14 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia then
        rectangle_14    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia then
      rectangle_14      <= '0';
    end if;
  end if;
end process RectangleHGen_14;

-- Socket  generator 5

RectangleHGen_15      : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_15 : unsigned(9 downto 0);
  variable v_index_15 : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
      rectangle_15      <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_15 - 1 and
      Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_15 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_15 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_15 + ball_dia then
        h_index_15 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_15;
        v_index_15 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_15;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_15));
        if H_boundary(TO_INTEGER(h_index_15)) = '1' then
          rectangle_15 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_15)) = '0' then
          rectangle_15 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_15 + ball_dia then
        rectangle_15    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_15 + ball_dia then
      rectangle_15      <= '0';
    end if;
  end if;
end process RectangleHGen_15;

-- Socket  generator 6

RectangleHGen_16       : process (clk25)
  variable H_boundary : unsigned(0 to 28);
  variable h_index_16 : unsigned(9 downto 0);
  variable v_index_16 : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = '1' then
      rectangle_16      <= '0';
    elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_16 - 1 and
      Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_16 - 1 then
      if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_16 + ball_dia and
        Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_16 + ball_dia then
        h_index_16 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_16;
        v_index_16 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_16;
        H_boundary := (others => '0');
        H_boundary := C_boundary(TO_INTEGER(v_index_16));
        if H_boundary(TO_INTEGER(h_index_16)) = '1' then
          rectangle_16 <= '1';
        elsif H_boundary(TO_INTEGER(h_index_16)) = '0' then
          rectangle_16 <= '0';
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_16 + ball_dia then
        rectangle_16    <= '0';
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_16 + ball_dia then
      rectangle_16      <= '0';
    end if;
  end if;
end process RectangleHGen_16;



------stick for calibration

RectangleHGen : process (clk25)
begin
  if rising_edge(clk) then
    if reset = '1' or Hcount = HSYNC + HBACK_PORCH + stick_H_1 then
      stick_h <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH + stick_H_2 then
      stick_h <= '0';
    end if;
  end if;
end process RectangleHGen;

RectangleVGen : process (clk25)
begin
  if rising_edge(clk) then
    if reset = '1' then
      stick_v    <= '0';
    elsif EndOfLine = '1' then
      if Vcount = VSYNC + VBACK_PORCH - 1 + stick_V_1 then
        stick_v <= '1';
      elsif Vcount = VSYNC + VBACK_PORCH - 1 + stick_V_2 then
        stick_v <= '0';
      end if;
    end if;
  end if;
end process RectangleVGen;

stick <= stick_h and stick_v;


------------crosshair                    --------
RectangleHGen_00       : process (clk25)
  variable H_boundary : unsigned(0 to 15);
  variable h_index_00 : unsigned(9 downto 0);
  variable v_index_00 : unsigned(9 downto 0);

begin
  if rising_edge(clk25) then
    if reset = ’1’ then
      rectangle_00      <= ’0’;
    elsif Hcount > HSYNC + HBACK_PORCH + cross_H  - 1 and
      Vcount > VSYNC + VBACK_PORCH + cross_V  - 1 then
      if Hcount < HSYNC + HBACK_PORCH + cross_H + cross_dia and
        Vcount < VSYNC + VBACK_PORCH + cross_V + cross_dia then
        h_index_00 := Hcount - HSYNC - HBACK_PORCH - cross_H;
        v_index_00 := Vcount - VSYNC - VBACK_PORCH - cross_V;
        H_boundary := (others => ’0’);
        H_boundary := cross_boundary(TO_INTEGER(v_index_00));
        if H_boundary(TO_INTEGER(h_index_00)) = ’1’ then
          rectangle_00 <= ’1’;
        elsif H_boundary(TO_INTEGER(h_index_00)) = ’0’ then
          rectangle_00 <= ’0’;
        end if;
      elsif Hcount >= HSYNC + HBACK_PORCH + cross_H + cross_dia then
        rectangle_00    <= ’0’;
      end if;
    elsif Hcount = HSYNC + HBACK_PORCH + cross_H + cross_dia then
      rectangle_00      <= ’0’;
    end if;
  end if;
end process RectangleHGen_00;

----------------output
VideoOut : process (clk25, reset)

begin
  if reset = ’1’ then
    VGA_R      <= ”0000000000”;
    VGA_G      <= ”0000000000”;
    VGA_B      <= ”0000000000”;
  elsif clk25’event and clk25 = ’1’ then
    if calibration = ’1’ then
      if rectangle_00 = ’1’ then
        VGA_R <= ”1111111111”;
        VGA_G <= ”1111111111”;
        VGA_B <= ”1111111111”;
      elsif stick = ’1’ then
        VGA_R <= ”0000000000”;
        VGA_G <= ”0000000000”;
        VGA_B <= ”0000000000”;
      elsif vga_hblank = ’0’ and vga_vblank = ’0’ then
        VGA_R <= ”0000000000”;
        VGA_G <= ”1111111111”;
        VGA_B <= ”0000000000”;
      else
        VGA_R <= ”0000000000”;
        VGA_G <= ”0000000000”;
        VGA_B <= ”0000000000”;
      end if;
    else


      if rectangle_00 = ’1’ then
        VGA_R           <= ”1111111111”;
        VGA_G           <= ”1111111111”;
        VGA_B           <= ”1111111111”;
      elsif ( Hcount >= HSYNC + HBACK_PORCH and Hcount < HSYNC + HBACK_PORCH + 641 and
              ((Vcount >= VSYNC + VBACK_PORCH and
                Vcount < VSYNC + VBACK_PORCH+ to_integer(black_b_y) + 1)or
               (Vcount > VSYNC + VBACK_PORCH+ 480 - to_integer(black_b_y) and
                Vcount < VSYNC + VBACK_PORCH+480)))or
        ( Vcount >= VSYNC + VBACK_PORCH and Vcount < VSYNC + VBACK_PORCH + 480 and
          ((Hcount >= HSYNC + HBACK_PORCH and Hcount < HSYNC + HBACK_PORCH +
            to_integer(black_b_x) + 1 )or
           (Hcount >= HSYNC + HBACK_PORCH+ 640- to_integer(black_b_x)
            and Hcount < HSYNC + HBACK_PORCH+641)))then
        VGA_R           <= ”0000000000”;
        VGA_G           <= ”0000000000”;
        VGA_B           <= ”0000000000”;
      elsif ( Hcount >= HSYNC + HBACK_PORCH and
              Hcount < HSYNC + HBACK_PORCH + 641 and
              ((Vcount >= VSYNC + VBACK_PORCH and
                Vcount < VSYNC + VBACK_PORCH +
                to_integer(border_2)+ 1 -to_integer(margin))or
               (Vcount > VSYNC + VBACK_PORCH +
                to_integer(border_4)-1+to_integer(margin)
                and Vcount < VSYNC + VBACK_PORCH+480)))or
        ( Vcount >= VSYNC + VBACK_PORCH and
          Vcount < VSYNC + VBACK_PORCH + 480 and
          ((Hcount >= HSYNC + HBACK_PORCH and
            Hcount < HSYNC + HBACK_PORCH +
            to_integer(border_1) + 1 -to_integer(margin))or
           (Hcount >= HSYNC + HBACK_PORCH+
            to_integer(border_3)-1+to_integer(margin)
            and Hcount < HSYNC + HBACK_PORCH+641)))then
        VGA_R           <= "1111111111";
        VGA_G           <= "1111111111";
        VGA_B           <= "0000000000";
      elsif ( Hcount >= HSYNC + HBACK_PORCH and
              Hcount < HSYNC + HBACK_PORCH + 641 and
              ((Vcount >= VSYNC + VBACK_PORCH+
                to_integer(border_2)+ 1 -to_integer(margin)
                and Vcount < VSYNC + VBACK_PORCH+ to_integer(border_2)+ 1)or
               (Vcount > VSYNC + VBACK_PORCH +
                to_integer(border_4)-1 and
                Vcount <= VSYNC + VBACK_PORCH+
                to_integer(border_4)-1+to_integer(margin))))or
        ( Vcount >= VSYNC + VBACK_PORCH and
          Vcount < VSYNC + VBACK_PORCH + 480 and
          ((Hcount >= HSYNC + HBACK_PORCH +
            to_integer(border_1) + 1 -to_integer(margin) and
            Hcount < HSYNC + HBACK_PORCH+
            to_integer(border_1) + 1)or
           (Hcount > HSYNC + HBACK_PORCH+
            to_integer(border_3)-1 and
            Hcount < HSYNC + HBACK_PORCH+
            to_integer(border_3)-1+to_integer(margin))))then
        VGA_R           <= "1111111111";
        VGA_G           <= "1111111111";
        VGA_B           <= "0000000000";

      elsif rectangle_1 = '1' and C_color_1 /="011"then
        VGA_R <= color_RGB(TO_INTEGER(C_color_1))(29 downto 20);
        VGA_G <= color_RGB(TO_INTEGER(C_color_1))(19 downto 10);
        VGA_B <= color_RGB(TO_INTEGER(C_color_1))(9 downto 0);
      elsif rectangle_2 = '1' and C_color_2 /="011"then
        VGA_R <= color_RGB(TO_INTEGER(C_color_2))(29 downto 20);
        VGA_G <= color_RGB(TO_INTEGER(C_color_2))(19 downto 10);
        VGA_B <= color_RGB(TO_INTEGER(C_color_2))(9 downto 0);
      elsif rectangle_3 = '1' and C_color_3 /="11"then
        VGA_R <= color_RGB(TO_INTEGER(C_color_3))(29 downto 20);
        VGA_G <= color_RGB(TO_INTEGER(C_color_3))(19 downto 10);
        VGA_B <= color_RGB(TO_INTEGER(C_color_3))(9 downto 0);
      elsif rectangle_4 = '1' and C_color_4 /="011"then
        VGA_R <= color_RGB(TO_INTEGER(C_color_4))(29 downto 20);
        VGA_G <= color_RGB(TO_INTEGER(C_color_4))(19 downto 10);
        VGA_B <= color_RGB(TO_INTEGER(C_color_4))(9 downto 0);
      elsif rectangle_5 = '1' and C_color_5 /="011"then
        VGA_R <= color_RGB(TO_INTEGER(C_color_5))(29 downto 20);
        VGA_G <= color_RGB(TO_INTEGER(C_color_5))(19 downto 10);
        VGA_B <= color_RGB(TO_INTEGER(C_color_5))(9 downto 0);
      elsif rectangle_6 = '1' and C_color_6 /="011"then
        VGA_R <= color_RGB(TO_INTEGER(C_color_6))(29 downto 20);
        VGA_G <= color_RGB(TO_INTEGER(C_color_6))(19 downto 10);
        VGA_B <= color_RGB(TO_INTEGER(C_color_6))(9 downto 0);
      elsif rectangle_7 = '1' and C_color_7 /="011"then
        VGA_R <= color_RGB(TO_INTEGER(C_color_7))(29 downto 20);
        VGA_G <= color_RGB(TO_INTEGER(C_color_7))(19 downto 10);
        VGA_B <= color_RGB(TO_INTEGER(C_color_7))(9 downto 0);
      elsif rectangle_11 = '1' or
        rectangle_12 = '1' or
        rectangle_13 = '1' or
        rectangle_14 = '1' or
        rectangle_15 = '1' or
        rectangle_16 = '1' then
        VGA_R <= "1111111111";
        VGA_G <= "1111111111";
        VGA_B <= "0000000000";
      elsif vga_hblank = '0' and vga_vblank = '0' then
        VGA_R <= "0000000000";
        VGA_G <= "1111111111";
        VGA_B <= "0000000000";
      else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end if;
end process VideoOut;

VGA_CLK    <= clk25;
VGA_HS     <= not vga_hsync;
VGA_VS     <= not vga_vsync;
VGA_SYNC   <= '0';
VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--       Abdulhamid Ghandour
--       Thomas John
--       Jaime Peretzman
--       Bharadwaj Vellore
--
-- Desc:
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity soundcontroller is

  port (
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  unsigned(3 downto 0);
    readdata   : out unsigned(31 downto 0);
    writedata  : in  unsigned(31 downto 0);
    aud_xck    : out std_logic;
    aud_adclrck: out std_logic;
    aud_adcdat : in std_logic;
    aud_daclrck: out std_logic;
    aud_dacdat : out std_logic;
    aud_bclk   : inout std_logic
  );
end soundcontroller;

architecture rtl of soundcontroller is

  type ram_type is array(7 downto 0) of unsigned(31 downto 0);
  signal RAM : ram_type;
  signal ram_address : unsigned(2 downto 0);
  signal counter   : unsigned(31 downto 0);
  signal audio_clock : unsigned(1 downto 0) := "00";
  signal audio_request : std_logic;
  signal audio_ctrl: std_logic := '0';

  component de2_wm8731_audio port (
    clk : in std_logic;        --  Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;        --    Audio CODEC controller test mode
    audio_request : out std_logic;  --    Audio controller request new data
    data : in unsigned(15 downto 0);

    -- Audio interface signals
    AUD_ADCLRCK   : out   std_logic;   --    Audio CODEC ADC LR Clock
    AUD_ADCDAT    : in    std_logic;   --    Audio CODEC ADC Data
    AUD_DACLRCK   : out   std_logic;   --    Audio CODEC DAC LR Clock
    AUD_DACDAT    : out   std_logic;   --    Audio CODEC DAC Data
    AUD_BCLK      : inout std_logic  --    Audio CODEC Bit-Stream Clock
  );
  end  component;

begin
  ram_address <= address(2 downto 0);

  audio_clk_gen: process (clk)
  begin
    if rising_edge(clk) then
      audio_clock <= audio_clock + "1";
    end if;
  end process audio_clk_gen;

  audio_host_control: process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then

      else
        if chipselect = '1' then
          if read = '1' then
```

```
            readdata <= RAM(to_integer(ram_address));
          elsif write = '1' then
            RAM(to_integer(ram_address)) <= writedata;
          end if;
        end if;
-------------------------------------
    if audio_clock = "00" then
      if RAM(0)(0) = '1' then
        audio_ctrl <= '1';
        RAM(0)(0) <= '0';
      end if;

      if audio_ctrl = '1' then
        audio_ctrl <= '0';
      end if;
        end if;
--------------------------------------
      end if;
    end if;
  end process audio_host_control;

--  audio_state_ctrl: process (clk)
--  begin
--    if rising_edge (clk) then
--
----        counter <= (others => '0');
----      else
----        if counter = 100 then
----          counter <= (others => '0');
----          reset_ctrl <= '0';
----        else
----          counter <= counter + 1;
----        end if;
--      end if;
--    end if;
--  end process audio_state_ctrl;

  aud_xck <= audio_clock(1);

  beeper: de2_wm8731_audio port map (
    clk => audio_clock(1),
    reset_n => '1',
    test_mode => audio_ctrl,               -- Output a sine wave
    audio_request => audio_request,
    data => "0000000000000000",
    AUD_ADCLRCK  => aud_adclrck,
    AUD_ADCDAT   => aud_adcdat,
    AUD_DACLRCK  => aud_daclrck,
    AUD_DACDAT   => aud_dacdat,
    AUD_BCLK     => aud_bclk
  );
end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_wm8731_audio is
port (
    clk : in std_logic;       --  Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;       --    Audio CODEC controller test mode
    audio_request : out std_logic;  --    Audio controller request new data
    data : in unsigned(15 downto 0);

    -- Audio interface signals
    AUD_ADCLRCK   : out   std_logic;   --     Audio CODEC ADC LR Clock
    AUD_ADCDAT    : in    std_logic;   --     Audio CODEC ADC Data
    AUD_DACLRCK   : out   std_logic;   --     Audio CODEC DAC LR Clock
    AUD_DACDAT    : out   std_logic;   --     Audio CODEC DAC Data
    AUD_BCLK      : inout std_logic  --     Audio CODEC Bit-Stream Clock
  );
end  de2_wm8731_audio;

architecture rtl of de2_wm8731_audio is

    signal lrck : std_logic;
    signal bclk : std_logic;
    signal xck  : std_logic;

    signal lrck_divider : unsigned(7 downto 0);
    signal bclk_divider : unsigned(3 downto 0);

    signal set_bclk : std_logic;
    signal set_lrck : std_logic;
    signal clr_bclk : std_logic;
    signal lrck_lat : std_logic;

    signal shift_out : unsigned(15 downto 0);

    signal sin_out     : unsigned(7 downto 0);
    signal sin_counter : unsigned(11 downto 0);
    signal audio_on    : std_logic;
begin

    -- LRCK divider
    -- Audio chip main clock is 18.432MHz / Sample rate 48KHz
    -- Divider is 18.432 MHz / 48KHz = 192 (X"C0")
    -- Left justify mode set by I2C controller

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck_divider <= (others => '0');
      elsif lrck_divider = X"BF"  then         -- "C0" minus 1
        lrck_divider <= X"00";
      else
        lrck_divider <= lrck_divider + 1;
      end if;
    end if;
  end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if audio_on = '0' then
        audio_on <= test_mode;
      end if;
      if sin_counter = x"36E" then
        audio_on <= '0';
      end if;
    end if;
  end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        bclk_divider <= (others => '0');
      elsif bclk_divider = X"B" or set_lrck = '1'  then
        bclk_divider <= X"0";
      else
        bclk_divider <= bclk_divider + 1;
      end if;
    end if;
  end process;

  set_lrck <= '1' when lrck_divider = X"BF" else '0';

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck <= '0';
      elsif set_lrck = '1' then
        lrck <= not lrck;
      end if;
    end if;
  end process;

  -- BCLK divider
  set_bclk <= '1' when bclk_divider(3 downto 0) = "0101" else '0';
  clr_bclk <= '1' when bclk_divider(3 downto 0) = "1011" else '0';

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        bclk <= '0';
      elsif set_lrck = '1' or clr_bclk = '1' then
        bclk <= '0';
      elsif set_bclk = '1' then
        bclk <= '1';
      end if;
    end if;
  end process;

  -- Audio data shift output
  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        shift_out <= (others => '0');
      elsif set_lrck = '1' then
        if audio_on = '1' then
          shift_out <= ("00" & sin_out & "000000");
        else
          shift_out <= data;
        end if;
      elsif clr_bclk = '1' then
        shift_out <= shift_out (14 downto 0) & '0';
      end if;
    end if;
  end process;

    -- Audio outputs

    AUD_ADCLRCK  <= lrck;
    AUD_DACLRCK  <= lrck;
    AUD_DACDAT   <= shift_out(15);
    AUD_BCLK     <= bclk;

    -- Self test with Sin wave

    process(clk)
    begin
      if rising_edge(clk) then
        if reset_n = '0' then
            sin_counter <= (others => '0');
        elsif lrck_lat = '1' and lrck = '0'  then
          if sin_counter = x"36E" then
            sin_counter <= x"000";
          elsif audio_on = '1' then
            sin_counter <= sin_counter + 1;
          end if;
        end if;
      end if;
    end process;

    process(clk)
    begin
      if rising_edge(clk) then
        lrck_lat <= lrck;
      end if;
    end process;

    process (clk)
    begin
      if rising_edge(clk) then
        if lrck_lat = '1' and lrck = '0' then
          audio_request <= '1';
        else
          audio_request <= '0';
        end if;
      end if;
    end process;

  with sin_counter select sin_out <=
     x"49" when x"001",
   x"52" when x"002",
   x"46" when x"003",
   x"46" when x"004",
   x"03" when x"005",
   x"66" when x"006",
   x"00" when x"007",
   x"00" when x"008",
   x"04" when x"009",
   x"57" when x"00A",
   x"45" when x"00B",
   x"56" when x"00C",
   x"6d" when x"00D",
   x"06" when x"00E",
   x"20" when x"00F",
   x"74" when x"010",
   x"00" when x"011",
   x"10" when x"012",
   x"00" when x"013",
   x"00" when x"014",
   x"00" when x"015",
   x"01" when x"016",
   x"00" when x"017",
   x"01" when x"018",
   x"bb" when x"019",
   x"80" when x"01A",
   x"00" when x"01B",
   x"00" when x"01C",
   x"bb" when x"01D",
   x"80" when x"01E",
   x"00" when x"01F",
   x"00" when x"020",
   x"00" when x"021",
   x"01" when x"022",
   x"00" when x"023",
   x"08" when x"024",
   x"61" when x"025",
   x"64" when x"026",
   x"61" when x"027",
   x"74" when x"028",
   x"02" when x"029",
   x"ea" when x"02A",
   x"00" when x"02B",
   x"00" when x"02C",
   x"8b" when x"02D",
   x"90" when x"02E",
   x"7d" when x"02F",
   x"86" when x"030",
   x"71" when x"031",
   x"76" when x"032",
   x"6c" when x"033",
   x"6b" when x"034",
   x"75" when x"035",
   x"6e" when x"036",
   x"8f" when x"037",
   x"82" when x"038",
   x"a9" when x"039",
   x"9e" when x"03A",
   x"a2" when x"03B",
   x"aa" when x"03C",
   x"73" when x"03D",
   x"8f" when x"03E",
   x"3a" when x"03F",
      x”57”  when x”040”,
      x”1f”  when x”041”,
      x”26”  when x”042”,
      x”32”  when x”043”,
      x”21”  when x”044”,
      x”71”  when x”045”,
      x”4e”  when x”046”,
      x”d3”  when x”047”,
      x”a0”  when x”048”,
      x”fc”  when x”049”,
      x”f7”  when x”04A”,
      x”fd”  when x”04B”,
      x”f6”  when x”04C”,
      x”b8”  when x”04D”,
      x”f3”  when x”04E”,
      x”26”  when x”04F”,
      x”68”  when x”050”,
      x”04”  when x”051”,
      x”06”  when x”052”,
      x”03”  when x”053”,
      x”08”  when x”054”,
      x”29”  when x”055”,
      x”08”  when x”056”,
      x”91”  when x”057”,
      x”5f”  when x”058”,
      x”dc”  when x”059”,
      x”bb”  when x”05A”,
      x”e7”  when x”05B”,
      x”ea”  when x”05C”,
      x”bd”  when x”05D”,
      x”d7”  when x”05E”,
      x”94”  when x”05F”,
      x”a6”  when x”060”,
      x”86”  when x”061”,
      x”89”  when x”062”,
      x”81”  when x”063”,
      x”84”  when x”064”,
      x”76”  when x”065”,
      x”7e”  when x”066”,
      x”6c”  when x”067”,
      x”6f”  when x”068”,
      x”6f”  when x”069”,
      x”6b”  when x”06A”,
      x”78”  when x”06B”,
      x”74”  when x”06C”,
      x”80”  when x”06D”,
      x”7e”  when x”06E”,
      x”84”  when x”06F”,
      x”82”  when x”070”,
      x”84”  when x”071”,
      x”84”  when x”072”,
      x”83”  when x”073”,
      x”85”  when x”074”,
      x”82”  when x”075”,
      x”83”  when x”076”,
      x”7e”  when x”077”,
      x”80”  when x”078”,
      x”74”  when x”079”,
      x”7a”  when x”07A”,
      x”66”  when x”07B”,
      x”6d”  when x”07C”,
      x”5c”  when x”07D”,
      x”61”  when x”07E”,
      x”5a”  when x”07F”,
      x”5a”  when x”080”,
      x”60”  when x”081”,
      x”5c”  when x”082”,
      x”6f”  when x”083”,
      x”67”  when x”084”,
      x”86”  when x”085”,
      x”79”  when x”086”,
      x”9d”  when x”087”,
      x”92”  when x”088”,
      x”ab”  when x”089”,
      x”a6”  when x”08A”,
      x”ad”  when x”08B”,
      x”ae”  when x”08C”,
      x”a5”  when x”08D”,
      x”aa”  when x”08E”,
      x”94”  when x”08F”,
          x"9d" when x"090",
          x"7d" when x"091",
          x"89" when x"092",
          x"6b" when x"093",
          x"73" when x"094",
          x"65" when x"095",
          x"65" when x"096",
          x"6c" when x"097",
          x"67" when x"098",
          x"79" when x"099",
          x"72" when x"09A",
          x"8c" when x"09B",
          x"82" when x"09C",
          x"a0" when x"09D",
          x"96" when x"09E",
          x"ab" when x"09F",
          x"a8" when x"0A0",
          x"a5" when x"0A1",
          x"aa" when x"0A2",
          x"8f" when x"0A3",
          x"9b" when x"0A4",
          x"74" when x"0A5",
          x"82" when x"0A6",
          x"5a" when x"0A7",
          x"67" when x"0A8",
          x"3f" when x"0A9",
          x"4d" when x"0AA",
          x"2a" when x"0AB",
          x"33" when x"0AC",
          x"23" when x"0AD",
          x"24" when x"0AE",
          x"35" when x"0AF",
          x"29" when x"0B0",
          x"5d" when x"0B1",
          x"47" when x"0B2",
          x"90" when x"0B3",
          x"76" when x"0B4",
          x"c5" when x"0B5",
          x"ab" when x"0B6",
          x"ea" when x"0B7",
          x"db" when x"0B8",
          x"f6" when x"0B9",
          x"f4" when x"0BA",
          x"e4" when x"0BB",
          x"f0" when x"0BC",
          x"be" when x"0BD",
          x"d3" when x"0BE",
          x"8f" when x"0BF",
          x"a7" when x"0C0",
          x"67" when x"0C1",
          x"79" when x"0C2",
          x"50" when x"0C3",
          x"58" when x"0C4",
          x"4b" when x"0C5",
          x"4c" when x"0C6",
          x"51" when x"0C7",
          x"4d" when x"0C8",
          x"59" when x"0C9",
          x"55" when x"0CA",
          x"63" when x"0CB",
          x"5d" when x"0CC",
          x"6f" when x"0CD",
          x"69" when x"0CE",
          x"79" when x"0CF",
          x"75" when x"0D0",
          x"7f" when x"0D1",
          x"7c" when x"0D2",
          x"82" when x"0D3",
          x"81" when x"0D4",
          x"81" when x"0D5",
          x"82" when x"0D6",
          x"7d" when x"0D7",
          x"7f" when x"0D8",
          x"7e" when x"0D9",
          x"7c" when x"0DA",
          x"84" when x"0DB",
          x"80" when x"0DC",
          x"8a" when x"0DD",
          x"88" when x"0DE",
          x"8c" when x"0DF",
      x"8b" when x"0E0",
      x"8d" when x"0E1",
      x"8d" when x"0E2",
      x"90" when x"0E3",
      x"8e" when x"0E4",
      x"92" when x"0E5",
      x"91" when x"0E6",
      x"95" when x"0E7",
      x"94" when x"0E8",
      x"93" when x"0E9",
      x"94" when x"0EA",
      x"87" when x"0EB",
      x"8f" when x"0EC",
      x"75" when x"0ED",
      x"7e" when x"0EE",
      x"67" when x"0EF",
      x"6d" when x"0F0",
      x"63" when x"0F1",
      x"64" when x"0F2",
      x"67" when x"0F3",
      x"64" when x"0F4",
      x"71" when x"0F5",
      x"6c" when x"0F6",
      x"7c" when x"0F7",
      x"77" when x"0F8",
      x"84" when x"0F9",
      x"80" when x"0FA",
      x"88" when x"0FB",
      x"86" when x"0FC",
      x"8c" when x"0FD",
      x"89" when x"0FE",
      x"8f" when x"0FF",
      x"8e" when x"100",
      x"8d" when x"101",
      x"8f" when x"102",
      x"80" when x"103",
      x"88" when x"104",
      x"6f" when x"105",
      x"76" when x"106",
      x"65" when x"107",
      x"69" when x"108",
      x"67" when x"109",
      x"64" when x"10A",
      x"78" when x"10B",
      x"6e" when x"10C",
      x"90" when x"10D",
      x"84" when x"10E",
      x"a4" when x"10F",
      x"9b" when x"110",
      x"aa" when x"111",
      x"a9" when x"112",
      x"a0" when x"113",
      x"a7" when x"114",
      x"8b" when x"115",
      x"96" when x"116",
      x"77" when x"117",
      x"81" when x"118",
      x"6b" when x"119",
      x"70" when x"11A",
      x"6a" when x"11B",
      x"69" when x"11C",
      x"6f" when x"11D",
      x"6c" when x"11E",
      x"74" when x"11F",
      x"72" when x"120",
      x"74" when x"121",
      x"75" when x"122",
      x"71" when x"123",
      x"73" when x"124",
      x"71" when x"125",
      x"70" when x"126",
      x"76" when x"127",
      x"73" when x"128",
      x"80" when x"129",
      x"7b" when x"12A",
      x"88" when x"12B",
      x"83" when x"12C",
      x"90" when x"12D",
      x"8c" when x"12E",
      x"94" when x"12F",
      x"93" when x"130",
      x"92" when x"131",
      x"94" when x"132",
      x"8b" when x"133",
      x"90" when x"134",
      x"7e" when x"135",
      x"85" when x"136",
      x"6c" when x"137",
      x"75" when x"138",
      x"5e" when x"139",
      x"64" when x"13A",
      x"5e" when x"13B",
      x"5c" when x"13C",
      x"6d" when x"13D",
      x"64" when x"13E",
      x"80" when x"13F",
      x"77" when x"140",
      x"92" when x"141",
      x"8a" when x"142",
      x"9d" when x"143",
      x"98" when x"144",
      x"9e" when x"145",
      x"9f" when x"146",
      x"96" when x"147",
      x"9a" when x"148",
      x"8a" when x"149",
      x"90" when x"14A",
      x"7d" when x"14B",
      x"84" when x"14C",
      x"71" when x"14D",
      x"77" when x"14E",
      x"69" when x"14F",
      x"6c" when x"150",
      x"6a" when x"151",
      x"69" when x"152",
      x"75" when x"153",
      x"6f" when x"154",
      x"83" when x"155",
      x"7c" when x"156",
      x"8f" when x"157",
      x"89" when x"158",
      x"96" when x"159",
      x"94" when x"15A",
      x"8f" when x"15B",
      x"94" when x"15C",
      x"7b" when x"15D",
      x"86" when x"15E",
      x"62" when x"15F",
      x"6d" when x"160",
      x"55" when x"161",
      x"59" when x"162",
      x"5c" when x"163",
      x"56" when x"164",
      x"72" when x"165",
      x"65" when x"166",
      x"88" when x"167",
      x"7e" when x"168",
      x"91" when x"169",
      x"8f" when x"16A",
      x"8d" when x"16B",
      x"90" when x"16C",
      x"8a" when x"16D",
      x"8a" when x"16E",
      x"92" when x"16F",
      x"8c" when x"170",
      x"a3" when x"171",
      x"9b" when x"172",
      x"b0" when x"173",
      x"ab" when x"174",
      x"a9" when x"175",
      x"af" when x"176",
      x"8c" when x"177",
      x"9d" when x"178",
      x"64" when x"179",
      x"79" when x"17A",
      x"45" when x"17B",
      x"52" when x"17C",
      x"43" when x"17D",
      x"40" when x"17E",
      x"60" when x"17F",
      x”4f” when x”180”,
      x”8d” when x”181”,
      x”76” when x”182”,
      x”b1” when x”183”,
      x”a1” when x”184”,
      x”bd” when x”185”,
      x”bb” when x”186”,
      x”ad” when x”187”,
      x”b8” when x”188”,
      x”89” when x”189”,
      x”9c” when x”18A”,
      x”67” when x”18B”,
      x”76” when x”18C”,
      x”59” when x”18D”,
      x”5d” when x”18E”,
      x”63” when x”18F”,
      x”5c” when x”190”,
      x”72” when x”191”,
      x”6b” when x”192”,
      x”76” when x”193”,
      x”76” when x”194”,
      x”70” when x”195”,
      x”74” when x”196”,
      x”67” when x”197”,
      x”6b” when x”198”,
      x”65” when x”199”,
      x”65” when x”19A”,
      x”6e” when x”19B”,
      x”68” when x”19C”,
      x”82” when x”19D”,
      x”77” when x”19E”,
      x”97” when x”19F”,
      x”8c” when x”1A0”,
      x”a3” when x”1A1”,
      x”9e” when x”1A2”,
      x”a2” when x”1A3”,
      x”a5” when x”1A4”,
      x”97” when x”1A5”,
      x”9d” when x”1A6”,
      x”88” when x”1A7”,
      x”8f” when x”1A8”,
      x”7e” when x”1A9”,
      x”82” when x”1AA”,
      x”7d” when x”1AB”,
      x”7d” when x”1AC”,
      x”81” when x”1AD”,
      x”7e” when x”1AE”,
      x”84” when x”1AF”,
      x”83” when x”1B0”,
      x”82” when x”1B1”,
      x”84” when x”1B2”,
      x”7e” when x”1B3”,
      x”81” when x”1B4”,
      x”7b” when x”1B5”,
      x”7d” when x”1B6”,
      x”79” when x”1B7”,
      x”79” when x”1B8”,
      x”7a” when x”1B9”,
      x”79” when x”1BA”,
      x”83” when x”1BB”,
      x”7e” when x”1BC”,
      x”95” when x”1BD”,
      x”8b” when x”1BE”,
      x”a2” when x”1BF”,
      x”9c” when x”1C0”,
      x”9d” when x”1C1”,
      x”a2” when x”1C2”,
      x”82” when x”1C3”,
      x”92” when x”1C4”,
      x”5b” when x”1C5”,
      x”70” when x”1C6”,
      x”3b” when x”1C7”,
      x”48” when x”1C8”,
      x”37” when x”1C9”,
      x”34” when x”1CA”,
      x”54” when x”1CB”,
      x”43” when x”1CC”,
      x”77” when x”1CD”,
      x”68” when x”1CE”,
      x”7a” when x”1CF”,
      x"7c" when x"1D0",
      x"61" when x"1D1",
      x"6e" when x"1D2",
      x"56" when x"1D3",
      x"59" when x"1D4",
      x"7f" when x"1D5",
      x"65" when x"1D6",
      x"d8" when x"1D7",
      x"a5" when x"1D8",
      x"fb" when x"1D9",
      x"f7" when x"1DA",
      x"f7" when x"1DB",
      x"f9" when x"1DC",
      x"f3" when x"1DD",
      x"fc" when x"1DE",
      x"75" when x"1DF",
      x"bc" when x"1E0",
      x"0a" when x"1E1",
      x"38" when x"1E2",
      x"0a" when x"1E3",
      x"02" when x"1E4",
      x"08" when x"1E5",
      x"03" when x"1E6",
      x"52" when x"1E7",
      x"28" when x"1E8",
      x"98" when x"1E9",
      x"79" when x"1EA",
      x"b2" when x"1EB",
      x"ac" when x"1EC",
      x"9d" when x"1ED",
      x"ab" when x"1EE",
      x"75" when x"1EF",
      x"8a" when x"1F0",
      x"51" when x"1F1",
      x"62" when x"1F2",
      x"43" when x"1F3",
      x"46" when x"1F4",
      x"54" when x"1F5",
      x"47" when x"1F6",
      x"80" when x"1F7",
      x"68" when x"1F8",
      x"af" when x"1F9",
      x"9a" when x"1FA",
      x"c8" when x"1FB",
      x"c0" when x"1FC",
      x"c3" when x"1FD",
      x"c9" when x"1FE",
      x"a7" when x"1FF",
      x"b6" when x"200",
      x"88" when x"201",
      x"97" when x"202",
      x"71" when x"203",
      x"7b" when x"204",
      x"66" when x"205",
      x"6a" when x"206",
      x"67" when x"207",
      x"65" when x"208",
      x"6b" when x"209",
      x"69" when x"20A",
      x"70" when x"20B",
      x"6e" when x"20C",
      x"74" when x"20D",
      x"72" when x"20E",
      x"76" when x"20F",
      x"75" when x"210",
      x"77" when x"211",
      x"76" when x"212",
      x"7d" when x"213",
      x"79" when x"214",
      x"87" when x"215",
      x"82" when x"216",
      x"8e" when x"217",
      x"8b" when x"218",
      x"8c" when x"219",
      x"8e" when x"21A",
      x"83" when x"21B",
      x"88" when x"21C",
      x"79" when x"21D",
      x"7e" when x"21E",
      x"72" when x"21F",
      x"74" when x"220",
      x"77" when x"221",
      x"73" when x"222",
      x"87" when x"223",
      x"7e" when x"224",
      x"95" when x"225",
      x"8f" when x"226",
      x"96" when x"227",
      x"98" when x"228",
      x"8a" when x"229",
      x"91" when x"22A",
      x"7f" when x"22B",
      x"84" when x"22C",
      x"79" when x"22D",
      x"7c" when x"22E",
      x"76" when x"22F",
      x"78" when x"230",
      x"74" when x"231",
      x"75" when x"232",
      x"73" when x"233",
      x"73" when x"234",
      x"75" when x"235",
      x"74" when x"236",
      x"7b" when x"237",
      x"78" when x"238",
      x"84" when x"239",
      x"7f" when x"23A",
      x"90" when x"23B",
      x"8a" when x"23C",
      x"9a" when x"23D",
      x"96" when x"23E",
      x"9e" when x"23F",
      x"9d" when x"240",
      x"99" when x"241",
      x"9c" when x"242",
      x"89" when x"243",
      x"92" when x"244",
      x"73" when x"245",
      x"7f" when x"246",
      x"60" when x"247",
      x"68" when x"248",
      x"58" when x"249",
      x"5a" when x"24A",
      x"5d" when x"24B",
      x"59" when x"24C",
      x"6f" when x"24D",
      x"64" when x"24E",
      x"8c" when x"24F",
      x"7c" when x"250",
      x"a7" when x"251",
      x"9b" when x"252",
      x"af" when x"253",
      x"ae" when x"254",
      x"a1" when x"255",
      x"ab" when x"256",
      x"80" when x"257",
      x"92" when x"258",
      x"59" when x"259",
      x"6c" when x"25A",
      x"3b" when x"25B",
      x"48" when x"25C",
      x"3c" when x"25D",
      x"37" when x"25E",
      x"5d" when x"25F",
      x"49" when x"260",
      x"8d" when x"261",
      x"75" when x"262",
      x"b9" when x"263",
      x"a5" when x"264",
      x"d1" when x"265",
      x"c8" when x"266",
      x"cd" when x"267",
      x"d3" when x"268",
      x"b0" when x"269",
      x"c1" when x"26A",
      x"86" when x"26B",
      x"9b" when x"26C",
      x"60" when x"26D",
      x"71" when x"26E",
      x"4c" when x"26F",
      x"54" when x"270",
      x"4d" when x"271",
      x"4a" when x"272",
      x"62" when x"273",
      x"56" when x"274",
      x"7d" when x"275",
      x"70" when x"276",
      x"90" when x"277",
      x"89" when x"278",
      x"94" when x"279",
      x"94" when x"27A",
      x"8b" when x"27B",
      x"90" when x"27C",
      x"7d" when x"27D",
      x"84" when x"27E",
      x"73" when x"27F",
      x"77" when x"280",
      x"71" when x"281",
      x"71" when x"282",
      x"79" when x"283",
      x"74" when x"284",
      x"84" when x"285",
      x"7e" when x"286",
      x"8d" when x"287",
      x"8a" when x"288",
      x"8e" when x"289",
      x"8e" when x"28A",
      x"89" when x"28B",
      x"8c" when x"28C",
      x"82" when x"28D",
      x"86" when x"28E",
      x"7e" when x"28F",
      x"80" when x"290",
      x"80" when x"291",
      x"7e" when x"292",
      x"85" when x"293",
      x"82" when x"294",
      x"88" when x"295",
      x"87" when x"296",
      x"87" when x"297",
      x"88" when x"298",
      x"81" when x"299",
      x"84" when x"29A",
      x"77" when x"29B",
      x"7c" when x"29C",
      x"71" when x"29D",
      x"74" when x"29E",
      x"71" when x"29F",
      x"70" when x"2A0",
      x"76" when x"2A1",
      x"73" when x"2A2",
      x"7d" when x"2A3",
      x"79" when x"2A4",
      x"86" when x"2A5",
      x"82" when x"2A6",
      x"8b" when x"2A7",
      x"89" when x"2A8",
      x"89" when x"2A9",
      x"8b" when x"2AA",
      x"81" when x"2AB",
      x"86" when x"2AC",
      x"7a" when x"2AD",
      x"7d" when x"2AE",
      x"75" when x"2AF",
      x"77" when x"2B0",
      x"74" when x"2B1",
      x"75" when x"2B2",
      x"76" when x"2B3",
      x"75" when x"2B4",
      x"79" when x"2B5",
      x"78" when x"2B6",
      x"7b" when x"2B7",
      x"7a" when x"2B8",
      x"7a" when x"2B9",
      x"7b" when x"2BA",
      x"7a" when x"2BB",
      x"7a" when x"2BC",
      x"7b" when x"2BD",
      x"7a" when x"2BE",
      x"7f" when x"2BF",
      x"7c" when x"2C0",
      x"86" when x"2C1",
      x"82" when x"2C2",
      x"8d" when x"2C3",
      x"8a" when x"2C4",
      x"91" when x"2C5",
      x"8f" when x"2C6",
      x"91" when x"2C7",
      x"92" when x"2C8",
      x"8e" when x"2C9",
      x"90" when x"2CA",
      x"88" when x"2CB",
      x"8c" when x"2CC",
      x"81" when x"2CD",
      x"85" when x"2CE",
      x"7a" when x"2CF",
      x"7d" when x"2D0",
      x"75" when x"2D1",
      x"77" when x"2D2",
      x"71" when x"2D3",
      x"72" when x"2D4",
      x"6d" when x"2D5",
      x"6f" when x"2D6",
      x"6e" when x"2D7",
      x"6d" when x"2D8",
      x"74" when x"2D9",
      x"70" when x"2DA",
      x"7a" when x"2DB",
      x"77" when x"2DC",
      x"7f" when x"2DD",
      x"7d" when x"2DE",
      x"82" when x"2DF",
      x"81" when x"2E0",
      x"81" when x"2E1",
      x"82" when x"2E2",
      x"7d" when x"2E3",
      x"7f" when x"2E4",
      x"7a" when x"2E5",
      x"7b" when x"2E6",
      x"7e" when x"2E7",
      x"7b" when x"2E8",
      x"86" when x"2E9",
      x"81" when x"2EA",
      x"8f" when x"2EB",
      x"8b" when x"2EC",
      x"92" when x"2ED",
      x"91" when x"2EE",
      x"8d" when x"2EF",
      x"90" when x"2F0",
      x"86" when x"2F1",
      x"89" when x"2F2",
      x"82" when x"2F3",
      x"83" when x"2F4",
      x"86" when x"2F5",
      x"83" when x"2F6",
      x"8c" when x"2F7",
      x"89" when x"2F8",
      x"8c" when x"2F9",
      x"8d" when x"2FA",
      x"82" when x"2FB",
      x"88" when x"2FC",
      x"73" when x"2FD",
      x"7b" when x"2FE",
      x"69" when x"2FF",
      x"6d" when x"300",
      x"6b" when x"301",
      x"68" when x"302",
      x"78" when x"303",
      x"71" when x"304",
      x"86" when x"305",
      x"7f" when x"306",
      x"8b" when x"307",
      x"8a" when x"308",
      x"85" when x"309",
      x"8a" when x"30A",
      x"7b" when x"30B",
      x"80" when x"30C",
      x"73" when x"30D",
      x"76" when x"30E",
      x"73" when x"30F",
      x"72" when x"310",
      x"7a" when x"311",
      x"76" when x"312",
      x"83" when x"313",
      x"7f" when x"314",
      x"8a" when x"315",
      x"87" when x"316",
      x"49" when x"317",
      x"4c" when x"318",
      x"54" when x"319",
      x"53" when x"31A",
      x"00" when x"31B",
      x"50" when x"31C",
      x"00" when x"31D",
      x"00" when x"31E",
      x"4e" when x"31F",
      x"49" when x"320",
      x"4f" when x"321",
      x"46" when x"322",
      x"43" when x"323",
      x"49" when x"324",
      x"44" when x"325",
      x"52" when x"326",
      x"00" when x"327",
      x"0c" when x"328",
      x"00" when x"329",
      x"00" when x"32A",
      x"30" when x"32B",
      x"32" when x"32C",
      x"38" when x"32D",
      x"30" when x"32E",
      x"30" when x"32F",
      x"2d" when x"330",
      x"2d" when x"331",
      x"35" when x"332",
      x"39" when x"333",
      x"30" when x"334",
      x"00" when x"335",
      x"00" when x"336",
      x"45" when x"337",
      x"49" when x"338",
      x"47" when x"339",
      x"4e" when x"33A",
      x"00" when x"33B",
      x"11" when x"33C",
      x"00" when x"33D",
      x"00" when x"33E",
      x"61" when x"33F",
      x"4a" when x"340",
      x"6d" when x"341",
      x"69" when x"342",
      x"20" when x"343",
      x"65" when x"344",
      x"65" when x"345",
      x"50" when x"346",
      x"65" when x"347",
      x"72" when x"348",
      x"7a" when x"349",
      x"74" when x"34A",
      x"61" when x"34B",
      x"6d" when x"34C",
      x"00" when x"34D",
      x"6e" when x"34E",
      x"01" when x"34F",
      x"00" when x"350",
      x"53" when x"351",
      x"49" when x"352",
      x"54" when x"353",
      x"46" when x"354",
      x"00" when x"355",
      x"16" when x"356",
      x"00" when x"357",
      x"00" when x"358",
      x"6f" when x"359",
      x"53" when x"35A",
      x"79" when x"35B",
      x"6e" when x"35C",
      x"53" when x"35D",
      x"20" when x"35E",
      x"75" when x"35F",
      x”6f” when x”360”,
      x”64” when x”361”,
      x”6e” when x”362”,
      x”46” when x”363”,
      x”20” when x”364”,
      x”72” when x”365”,
      x”6f” when x”366”,
      x”65” when x”367”,
      x”67” when x”368”,
      x”38” when x”369”,
      x”20” when x”36A”,
      x”30” when x”36B”,
      x”2e” when x”36C”,
      x”00” when x”36D”,
      x”00” when x”36E”,
       X”00” when others;

  end architecture;
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--        Abdulhamid Ghandour
--        Thomas John
--        Jaime Peretzman
--        Bharadwaj Vellore
--
-- Desc:
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uicontroller is

  port (
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  unsigned(4 downto 0);
    readdata   : out unsigned(31 downto 0);
    writedata  : in  unsigned(31 downto 0);
    hex0       : out std_logic_vector(7 downto 0);
    hex1       : out std_logic_vector(7 downto 0);
    hex2       : out std_logic_vector(7 downto 0);
    hex3       : out std_logic_vector(7 downto 0);
    hex4       : out std_logic_vector(7 downto 0);
    hex5       : out std_logic_vector(7 downto 0);
    hex6       : out std_logic_vector(7 downto 0);
    hex7       : out std_logic_vector(7 downto 0);
    key        : in std_logic_vector(3 downto 0);
    switch     : in unsigned (17 downto 0)
  );
end uicontroller;

architecture rtl of uicontroller is

  type ram_type is array(31 downto 0) of unsigned(31 downto 0);
  signal RAM : ram_type;
  signal ram_address : unsigned(4 downto 0);
  signal int_key1 : std_logic;
  signal int_key2 : std_logic;
begin
  ram_address <= address;

  reg_loader: process(clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        RAM(0) <= (others => '1');
        RAM(1) <= (others => '1');
        RAM(2) <= (others => '1');
        RAM(3) <= (others => '1');
        RAM(4) <= (others => '1');
        RAM(5) <= (others => '1');
        RAM(6) <= (others => '1');
        RAM(7) <= (others => '1');
      else
        if chipselect = '1' then
          if write = '1' then
            RAM(to_integer(ram_address)) <= writedata;
          elsif read = '1' then
            readdata <=RAM(to_integer(ram_address));
          end if;
        else
      if RAM(8)(0) = '1' then
            RAM(8)(0) <= int_key1;
          end if;
          if RAM(9)(0) = '1' then
            RAM(9)(0) <= int_key2;
          end if;
      RAM(10)(17 downto 0) <= switch(17 downto 0);
        end if;
      end if;
    end if;
```

```vhdl
  end process reg_loader;

  seven_segment_driver: process(clk)
    begin
    if rising_edge(clk) then
      if reset_n = '0' then
        -- do nothing
      else
        hex0(0) <= RAM(0)(0);
        hex0(1) <= RAM(0)(1);
        hex0(2) <= RAM(0)(2);
        hex0(3) <= RAM(0)(3);
        hex0(4) <= RAM(0)(4);
        hex0(5) <= RAM(0)(5);
        hex0(6) <= RAM(0)(6);
        hex0(7) <= RAM(0)(7);

        hex1(0) <= RAM(1)(0);
        hex1(1) <= RAM(1)(1);
        hex1(2) <= RAM(1)(2);
        hex1(3) <= RAM(1)(3);
        hex1(4) <= RAM(1)(4);
        hex1(5) <= RAM(1)(5);
        hex1(6) <= RAM(1)(6);
        hex1(7) <= RAM(1)(7);

        hex2(0) <= RAM(2)(0);
        hex2(1) <= RAM(2)(1);
        hex2(2) <= RAM(2)(2);
        hex2(3) <= RAM(2)(3);
        hex2(4) <= RAM(2)(4);
        hex2(5) <= RAM(2)(5);
        hex2(6) <= RAM(2)(6);
        hex2(7) <= RAM(2)(7);

        hex3(0) <= RAM(3)(0);
        hex3(1) <= RAM(3)(1);
        hex3(2) <= RAM(3)(2);
        hex3(3) <= RAM(3)(3);
        hex3(4) <= RAM(3)(4);
        hex3(5) <= RAM(3)(5);
        hex3(6) <= RAM(3)(6);
        hex3(7) <= RAM(3)(7);

        hex4(0) <= RAM(4)(0);
        hex4(1) <= RAM(4)(1);
        hex4(2) <= RAM(4)(2);
        hex4(3) <= RAM(4)(3);
        hex4(4) <= RAM(4)(4);
        hex4(5) <= RAM(4)(5);
        hex4(6) <= RAM(4)(6);
        hex4(7) <= RAM(4)(7);

        hex5(0) <= RAM(5)(0);
        hex5(1) <= RAM(5)(1);
        hex5(2) <= RAM(5)(2);
        hex5(3) <= RAM(5)(3);
        hex5(4) <= RAM(5)(4);
        hex5(5) <= RAM(5)(5);
        hex5(6) <= RAM(5)(6);
        hex5(7) <= RAM(5)(7);

        hex6(0) <= RAM(6)(0);
        hex6(1) <= RAM(6)(1);
        hex6(2) <= RAM(6)(2);
        hex6(3) <= RAM(6)(3);
        hex6(4) <= RAM(6)(4);
        hex6(5) <= RAM(6)(5);
        hex6(6) <= RAM(6)(6);
        hex6(7) <= RAM(6)(7);

        hex7(0) <= RAM(7)(0);
        hex7(1) <= RAM(7)(1);
        hex7(2) <= RAM(7)(2);
        hex7(3) <= RAM(7)(3);
        hex7(4) <= RAM(7)(4);
        hex7(5) <= RAM(7)(5);
        hex7(6) <= RAM(7)(6);
        hex7(7) <= RAM(7)(7);
        int_key1 <= key(1); -- key(0) is not captured.
        int_key2 <= key(2);
      end if;
    end if;
  end process seven_segment_driver;
end rtl;
```

```
--
-- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
--        Abdulhamid Ghandour
--        Thomas John
--        Jaime Peretzman
--        Bharadwaj Vellore
--
-- Desc:
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity niostop is

  port (
    -- Clocks

    CLOCK_27,                                        -- 27 MHz
    CLOCK_50,                                        -- 50 MHz
    EXT_CLOCK : in std_logic;                        -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);           -- Push buttons
    SW : in std_logic_vector(17 downto 0);           -- DPDT switches

    -- LED displays

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
       : out std_logic_vector(6 downto 0);           -- (active low)
    LEDG : out std_logic_vector(8 downto 0);         -- Green LEDs (active high)
    LEDR : out unsigned(17 downto 0);       -- Red LEDs (active high)

    -- RS-232 interface

    UART_TXD : out std_logic;                        -- UART transmitter
    UART_RXD : in std_logic;                         -- UART receiver

    -- IRDA interface

    --IRDA_TXD : out std_logic;                        -- IRDA Transmitter
    IRDA_RXD : in std_logic;                         -- IRDA Receiver

    -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                       -- Low-byte Data Mask
    DRAM_UDQM,                                       -- High-byte Data Mask
    DRAM_WE_N,                                       -- Write Enable
    DRAM_CAS_N,                                      -- Column Address Strobe
    DRAM_RAS_N,                                      -- Row Address Strobe
    DRAM_CS_N,                                       -- Chip Select
    DRAM_BA_0,                                       -- Bank Address 0
    DRAM_BA_1,                                       -- Bank Address 0
    DRAM_CLK,                                        -- Clock
    DRAM_CKE : out std_logic;                        -- Clock Enable

    -- FLASH

    FL_DQ : inout std_logic_vector(7 downto 0);        -- Data bus
    FL_ADDR : out std_logic_vector(21 downto 0);  -- Address bus
    FL_WE_N,                                           -- Write Enable
    FL_RST_N,                                          -- Reset
    FL_OE_N,                                           -- Output Enable
    FL_CE_N : out std_logic;                           -- Chip Enable

    -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0);           -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0);           -- Address bus 18 Bits
    SRAM_UB_N,                                       -- High-byte Data Mask
    SRAM_LB_N,                                       -- Low-byte Data Mask
    SRAM_WE_N,                                           -- Write Enable
    SRAM_CE_N,                                           -- Chip Enable
    SRAM_OE_N : out std_logic;                           -- Output Enable

    -- USB controller

    OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
    OTG_ADDR : out std_logic_vector(1 downto 0);     -- Address
    OTG_CS_N,                                         -- Chip Select
    OTG_RD_N,                                         -- Write
    OTG_WR_N,                                         -- Read
    OTG_RST_N,                                        -- Reset
    OTG_FSPEED,                     -- USB Full Speed, 0 = Enable, Z = Disable
    OTG_LSPEED : out std_logic;     -- USB Low Speed, 0 = Enable, Z = Disable
    OTG_INT0,                                         -- Interrupt 0
    OTG_INT1,                                         -- Interrupt 1
    OTG_DREQ0,                                        -- DMA Request 0
    OTG_DREQ1 : in std_logic;                         -- DMA Request 1
    OTG_DACK0_N,                                      -- DMA Acknowledge 0
    OTG_DACK1_N : out std_logic;                      -- DMA Acknowledge 1

    -- 16 X 2 LCD Module

    LCD_ON,                       -- Power ON/OFF
    LCD_BLON,                     -- Back Light ON/OFF
    LCD_RW,                       -- Read/Write Select, 0 = Write, 1 = Read
    LCD_EN,                       -- Enable
    LCD_RS : out std_logic;       -- Command/Data Select, 0 = Command, 1 = Data
    LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

    -- SD card interface

    SD_DAT : in std_logic;        -- SD Card Data       SD pin 7 "DAT 0/DataOut"
    SD_DAT3 : out std_logic;      -- SD Card Data 3     SD pin 1 "DAT 3/nCS"
    SD_CMD : out std_logic;       -- SD Card Command    SD pin 2 "CMD/DataIn"
    SD_CLK : out std_logic;       -- SD Card Clock      SD pin 5 "CLK"

    -- USB JTAG link

    TDI,                          -- CPLD -> FPGA (data in)
    TCK,                          -- CPLD -> FPGA (clk)
    TCS : in std_logic;           -- CPLD -> FPGA (CS)
    TDO : out std_logic;          -- FPGA -> CPLD (data out)

    -- I2C bus

    I2C_SDAT : inout std_logic; -- I2C Data
    I2C_SCLK : out std_logic;   -- I2C Clock

    -- PS/2 port

    PS2_DAT,                      -- Data
    PS2_CLK : in std_logic;       -- Clock

    -- VGA output

    VGA_CLK,                               -- Clock
    VGA_HS,                                -- H_SYNC
    VGA_VS,                                -- V_SYNC
    VGA_BLANK,                             -- BLANK
    VGA_SYNC : out std_logic;              -- SYNC
    VGA_R,                                 -- Red[9:0]
    VGA_G,                                 -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0);    -- Blue[9:0]

    --  Ethernet Interface

    ENET_DATA : inout unsigned(15 downto 0);     -- DATA bus 16 Bits
    ENET_CMD,             -- Command/Data Select, 0 = Command, 1 = Data
    ENET_CS_N,                                   -- Chip Select
    ENET_WR_N,                                   -- Write
    ENET_RD_N,                                   -- Read
    ENET_RST_N,                                  -- Reset
    ENET_CLK : out std_logic;                    -- Clock 25 MHz
    ENET_INT : in std_logic;                     -- Interrupt

    -- Audio CODEC

    AUD_ADCLRCK : inout std_logic;                       -- ADC LR Clock
    AUD_ADCDAT : in std_logic;                           -- ADC Data
    AUD_DACLRCK : inout std_logic;                    -- DAC LR Clock
    AUD_DACDAT : out std_logic;                       -- DAC Data
    AUD_BCLK : inout std_logic;                       -- Bit-Stream Clock
    AUD_XCK : out std_logic;                          -- Chip Clock

    -- Video Decoder

    TD_DATA : in std_logic_vector(7 downto 0);  -- Data bus 8 bits
    TD_HS,                                      -- H_SYNC
    TD_VS : in std_logic;                       -- V_SYNC
    TD_RESET : out std_logic;                   -- Reset

    -- General-purpose I/O

    GPIO_0,                                      -- GPIO Connection 0
    GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
    );

end niostop;

architecture datapath of niostop is
  signal clk25 : std_logic := '0';
  signal reset_n : std_logic := '1';
  signal int_sclk : std_logic;
  signal int_sdat : std_logic;
  signal stop_counter : std_logic := '0';
  signal frameCount : unsigned(31 downto 0) := x"00000000";
  signal tickCount : unsigned(31 downto 0) := x"00000000";
  signal vision_flags_signal : std_logic_vector(7 downto 0);

  component de2_i2c_av_config is
  port (
    iCLK : in std_logic;
    iRST_N : in std_logic;
    I2C_SCLK : out std_logic;
    I2C_SDAT : inout std_logic
  );
  end component;

begin
  reset_n <= KEY(0);

  process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
      clk25 <= not clk25;
    end if;
  end process;

  niossystem: entity work.pool port map (
    clk => CLOCK_50,
    reset_n => KEY(0),

    -- the_sram
    SRAM_ADDR_from_the_sram => SRAM_ADDR,
    SRAM_CE_N_from_the_sram => SRAM_CE_N,
    SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
    SRAM_LB_N_from_the_sram => SRAM_LB_N,
    SRAM_OE_N_from_the_sram => SRAM_OE_N,
    SRAM_UB_N_from_the_sram => SRAM_UB_N,
    SRAM_WE_N_from_the_sram => SRAM_WE_N,

    -- the_vga
    VGA_BLANK_from_the_vga => VGA_BLANK,
    VGA_B_from_the_vga => VGA_B,
    VGA_CLK_from_the_vga => VGA_CLK,
    VGA_G_from_the_vga => VGA_G,
    VGA_HS_from_the_vga => VGA_HS,
    VGA_R_from_the_vga => VGA_R,
    VGA_SYNC_from_the_vga => VGA_SYNC,
    VGA_VS_from_the_vga => VGA_VS,

    -- the_vision
    frame_valid_to_the_vision => GPIO_1(13),
    line_valid_to_the_vision => GPIO_1(12),
    master_clk_from_the_vision => GPIO_1(11),
    no_detect_from_the_vision => LEDG(0),
    cal_direction_from_the_vision(0) => LEDR(11),
    cal_direction_from_the_vision(1) => LEDR(12),
    cal_direction_from_the_vision(2) => LEDR(13),
    cal_direction_from_the_vision(3) => LEDR(14),
    cal_direction_from_the_vision(4) => LEDR(15),
    cal_direction_from_the_vision(5) => LEDR(16),
    cal_direction_from_the_vision(6) => LEDR(17),
    pixel_clk_to_the_vision => GPIO_1(10),
    pixel_data_to_the_vision(0) => GPIO_1(0),
    pixel_data_to_the_vision(1) => GPIO_1(1),
    pixel_data_to_the_vision(2) => GPIO_1(5),
    pixel_data_to_the_vision(3) => GPIO_1(3),
    pixel_data_to_the_vision(4) => GPIO_1(2),
    pixel_data_to_the_vision(5) => GPIO_1(4),
    pixel_data_to_the_vision(6) => GPIO_1(6),
    pixel_data_to_the_vision(7) => GPIO_1(7),
    pixel_data_to_the_vision(8) => GPIO_1(8),
    pixel_data_to_the_vision(9) => GPIO_1(9),
    threshold_to_the_vision => SW(9 downto 0),
    vision_flags_from_the_vision => vision_flags_signal,

    -- the_audio
    aud_adcdat_to_the_sounddriver => AUD_ADCDAT,
    aud_adclrck_from_the_sounddriver => AUD_ADCLRCK,
    aud_bclk_to_and_from_the_sounddriver => AUD_BCLK,
    aud_dacdat_from_the_sounddriver => AUD_DACDAT,
    aud_daclrck_from_the_sounddriver => AUD_DACLRCK,
    aud_xck_from_the_sounddriver => AUD_XCK,

    -- the_lcd
    LCD_E_from_the_lcd => LCD_EN,
    LCD_RS_from_the_lcd => LCD_RS,
    LCD_RW_from_the_lcd => LCD_RW,
    LCD_data_to_and_from_the_lcd => LCD_DATA,

    -- the_uicontrol
    hex0_from_the_uicontrol(6 downto 0) => HEX0,
    hex1_from_the_uicontrol(6 downto 0) => HEX1,
    hex2_from_the_uicontrol(6 downto 0) => HEX2,
    hex3_from_the_uicontrol(6 downto 0) => HEX3,
    hex4_from_the_uicontrol(6 downto 0) => HEX4,
    hex5_from_the_uicontrol(6 downto 0) => HEX5,
    hex6_from_the_uicontrol(6 downto 0) => HEX6,
    hex7_from_the_uicontrol(6 downto 0) => HEX7,
    key_to_the_uicontrol => KEY,
    switch_to_the_uicontrol => SW,

    -- the_camera
    sclk_from_the_camera         => int_sclk,
    sdat_to_and_from_the_camera => int_sdat,
    ack_to_the_camera => GPIO_1(15)
  );

  frame_counter: process (GPIO_1(13))
  begin
    if rising_edge(GPIO_1(13)) then
      if reset_n = '0' then
        frameCount <= x"00000000";
      else
        if stop_counter = '1' then

        else
          frameCount <= frameCount + 1;
        end if;
      end if;
    end if;
  end process;

  tick_counter: process (clk25)
  begin
    if rising_edge(clk25) then
      if reset_n = '0' then
        stop_counter <= '0';
        tickCount <= x"00000000";
      else
        if stop_counter = '1' then

        else
          tickCount <= tickCount + 1;
          if(tickCount > x"05f5e100") then
            stop_counter <= '1';
          end if;
        end if;
      end if;
    end if;
  end process;

--  with SW(17) select
--    LEDR(9) <= frameCount(12) when '1',
--                '0' when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(8) <= frameCount(11) when '1',
--                '0' when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(7) <= frameCount(10) when '1',
--                vision_flags_signal(7) when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(6) <= frameCount(9) when '1',
--                vision_flags_signal(6) when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(5) <= frameCount(8) when '1',
--                vision_flags_signal(5) when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(4) <= frameCount(7) when '1',
--                vision_flags_signal(4) when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(3) <= frameCount(6) when '1',
--                vision_flags_signal(3) when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(2) <= frameCount(5) when '1',
--                vision_flags_signal(2) when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(1) <= frameCount(4) when '1',
--                vision_flags_signal(1) when '0',
--                'X' when others;
--  with SW(17) select
--    LEDR(0) <= frameCount(3) when '1',
--                vision_flags_signal(0) when '0',
--                'X' when others;

--  LEDR(0) <= vision_flags_signal(0);
--  LEDR(1) <= vision_flags_signal(1);
--  LEDR(2) <= vision_flags_signal(2);
--  LEDR(3) <= vision_flags_signal(3);
--  LEDR(4) <= vision_flags_signal(4);
--  LEDR(5) <= vision_flags_signal(5);
--  LEDR(6) <= vision_flags_signal(6);
--  LEDR(7) <= vision_flags_signal(7);


  i2c : de2_i2c_av_config port map (
    iCLK     => CLOCK_50,
    iRST_n   => '1',
    I2C_SCLK => I2C_SCLK,
    I2C_SDAT => I2C_SDAT
  );

  LEDG(2) <= int_sclk;
  LEDG(1) <= int_sdat;
  LEDG(7) <= GPIO_1(13);
  LEDG(6) <= GPIO_1(12);
  GPIO_1(14) <= int_sclk;
  GPIO_1(15) <= int_sdat;

  GPIO_0(14) <= GPIO_1(14);
  GPIO_0(15) <= GPIO_1(15);
  GPIO_0(11) <= int_sdat;

  LCD_ON   <= '1';
  LCD_BLON <= '1';
  FL_RST_N <= '1';

  FL_ADDR(21 downto 20) <= "00";

  SD_DAT3 <= '1';
  SD_CMD <= '1';
  SD_CLK <= '1';

  UART_TXD <= '0';
  DRAM_ADDR <= (others => '0');
  DRAM_LDQM <= '0';
  DRAM_UDQM <= '0';
  DRAM_WE_N <= '1';
  DRAM_CAS_N <= '1';
  DRAM_RAS_N <= '1';
  DRAM_CS_N <= '1';
  DRAM_BA_0 <= '0';
  DRAM_BA_1 <= '0';
  DRAM_CLK <= '0';
  DRAM_CKE <= '0';
  FL_WE_N <= '1';

  FL_OE_N <= '1';
  FL_CE_N <= '1';
  OTG_ADDR <= (others => '0');
  OTG_CS_N <= '1';
  OTG_RD_N <= '1';
  OTG_RD_N <= '1';
  OTG_WR_N <= '1';
  OTG_RST_N <= '1';
  OTG_FSPEED <= '1';
  OTG_LSPEED <= '1';
  OTG_DACK0_N <= '1';
  OTG_DACK1_N <= '1';

  ENET_CMD <= '0';
  ENET_CS_N <= '1';
  ENET_WR_N <= '1';
  ENET_RD_N <= '1';
  ENET_RST_N <= '1';
  ENET_CLK <= '0';

  TDO <= '0';
  TD_RESET <= '0';

  -- Set all bidirectional ports to tri-state
  DRAM_DQ      <= (others => 'Z');
  FL_DQ        <= (others => 'Z');
  OTG_DATA     <= (others => 'Z');
  ENET_DATA    <= (others => 'Z');
end datapath;
```