

Virtual Pool

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

May 11, 2008

Contents

1	Introduction	2
1.1	Gameplay	2
1.2	Game Configuration	2
2	Design Overview	3
2.1	High Level View	3
2.1.1	Basic Ideas	3
2.1.2	Block View	4
2.2	Hardware-Software partitioning	5
2.3	System Configuration	5
3	Detailed Design	5
3.1	Camera Controller	6
3.1.1	Camera Physical Interface	6
3.1.2	Camera Register Configuration	6
3.1.3	Camera Control Module	6
3.1.4	Programming the camera interface	8
3.2	Pixel Processing Front End	8
3.3	Calibration	9
3.4	Vision System	10
3.4.1	Interface	11
3.4.2	The Working	12
3.4.3	Filtering	14
3.4.4	Implementation	16
3.5	Ball Physics Simulation	16
3.5.1	Basics	16
3.5.2	Collision Event Handling	16
3.5.3	Collision Simulation	17
3.6	VGA Interface	18
4	Project Management	19
4.1	Versioning	19
5	Glossary of Terms	19
6	Source Code	20

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 positions, 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 the 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 disturbs 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 ball incurs a penalty. When the white ball is pocketed, it is returned to the table and placed at 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.

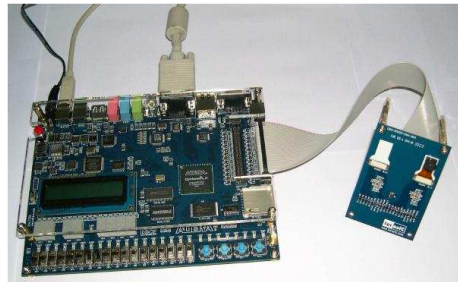


Figure 1: Board Level Connection

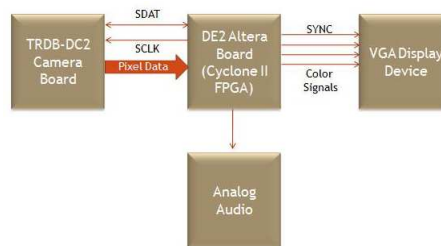


Figure 2: Block View of Physical Interfaces

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 I²C Interface, the Vision System, SRAM, Sound Driver, VGA Controller and User Interface module.

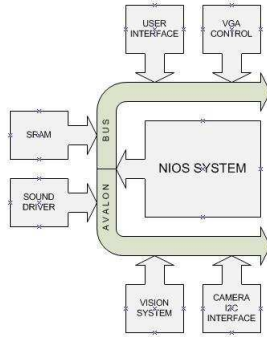


Figure 3: Block Diagram View

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

- Camera I²C 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-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 peripheral control and communication, calibration, ball-dynamics simulation, including transfer 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 I²C physical layer, and a piece of software that uses the register interface exported by the hardware to implement the I²C 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, and so 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 barrel 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 error 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 I²C 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 I²C-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 I²C 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.

I²C Interface The I²C 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 recognition 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

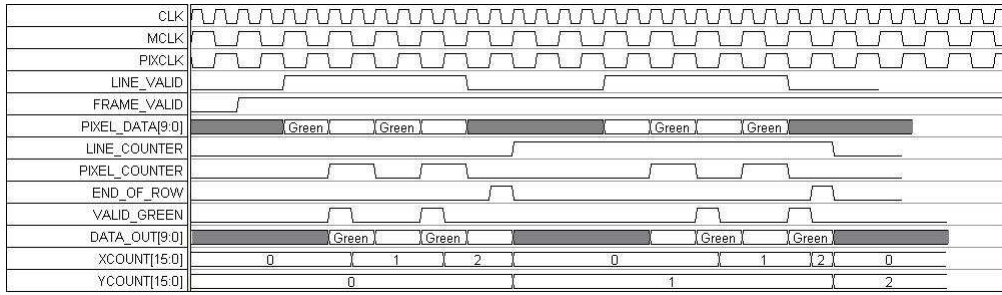


Figure 4: Pixel Processor Component Timing

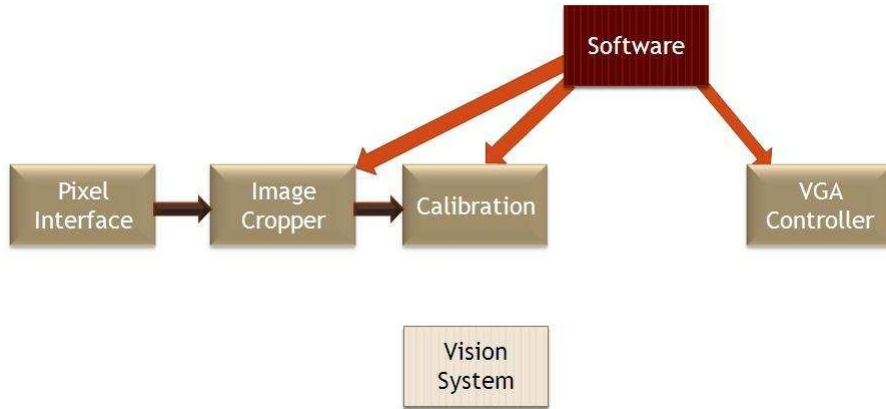


Figure 5: Calibration Mode Data Flow

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

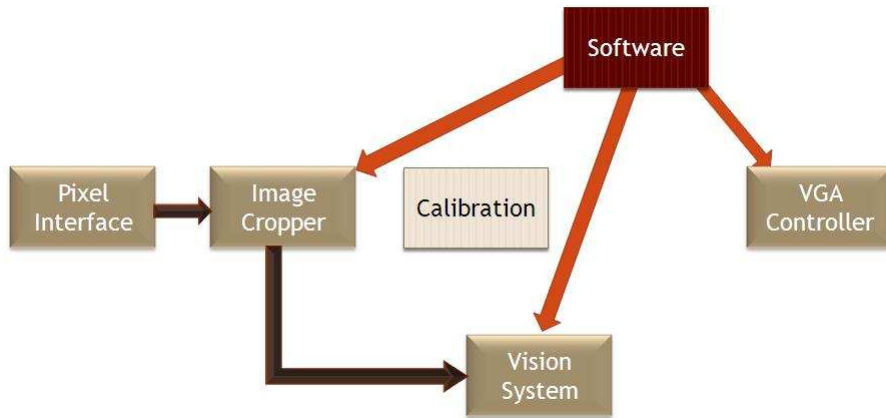


Figure 6: Gameplay Mode Data Flow

- 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

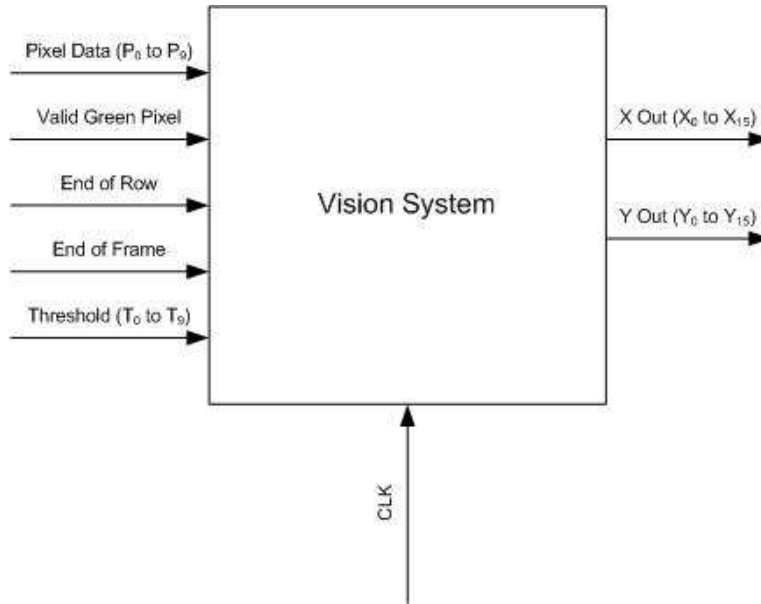


Figure 7: Vision System Block Diagram

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.

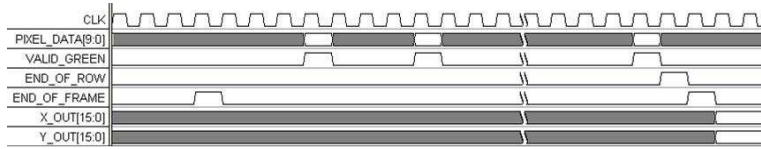


Figure 8: Vision System IO Timing Diagram

- **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.

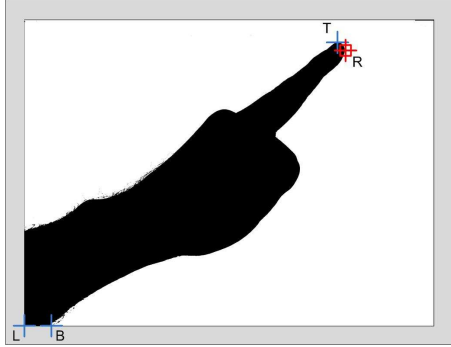


Figure 9: In this case, the top and right extremities are close to each other. Under such a condition, the right extremity is chosen as the output. This is the only case when a cue stick is used instead of a hand.

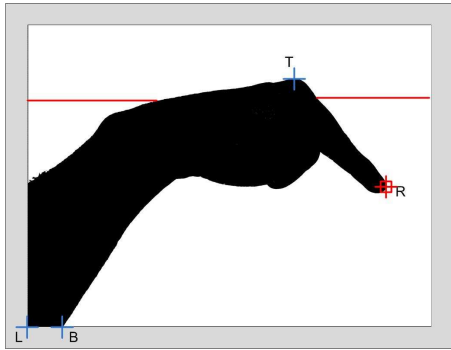


Figure 10: When the width measured a certain distance below the top extremity as shown is greater than a threshold value, it is assumed that the top extremity is not the finger tip. The right extremity is output in this case.

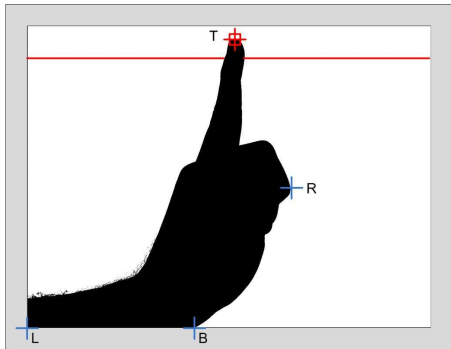


Figure 11: When the width measured a certain distance below the top extremity is lesser than the finger width threshold, the top extremity is assumed to be the finger tip.

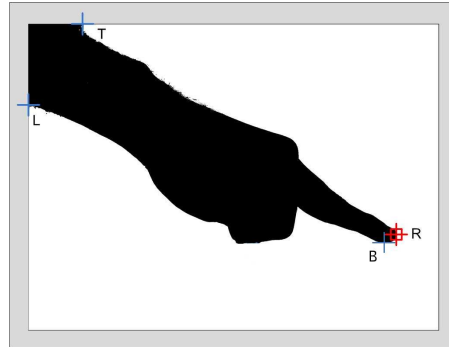
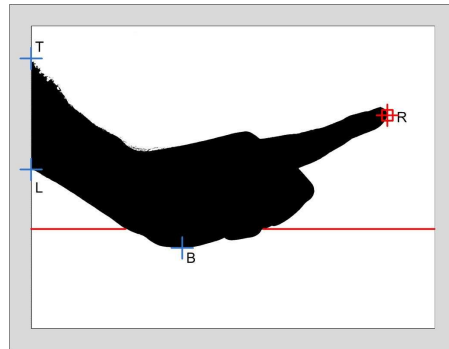


Figure 12: In this case, the bottom and right extremities are close to each other. Under such a condition, the right extremity is chosen as the output. This is the only case when a cue stick is used instead of a hand.

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

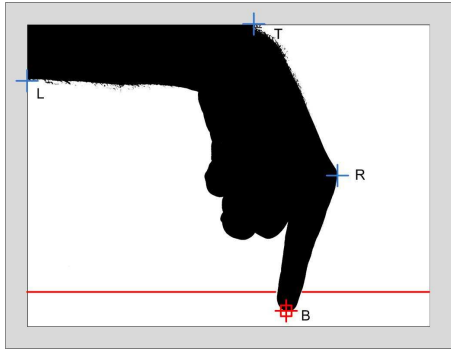


Figure 14: When the width measured a certain distance above the bottom extremity is lesser than the finger width threshold, the bottom extremity is assumed to be the finger tip.

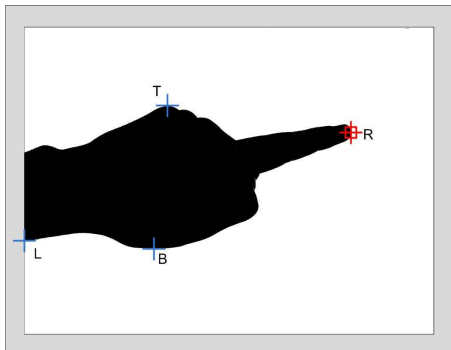


Figure 15: If the left extremity alone lies on an edge, it follows that the entry edge is the left edge. In such a case the right extremity is the tip. The opposite applies when the right extremity alone lies on an edge.

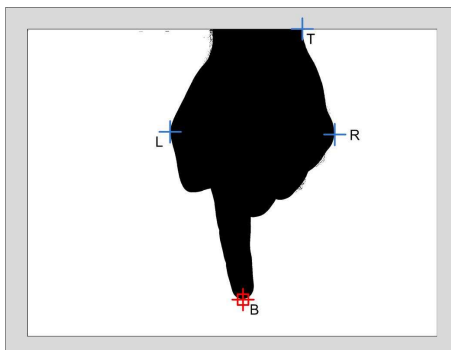


Figure 16: If the top extremity alone lies on an edge, it follows that the entry edge is the top edge. In such a case the bottom extremity is the tip. The opposite applies when the bottom extremity alone lies on an edge.

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 in 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 along 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 will 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 infinitesimal 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.

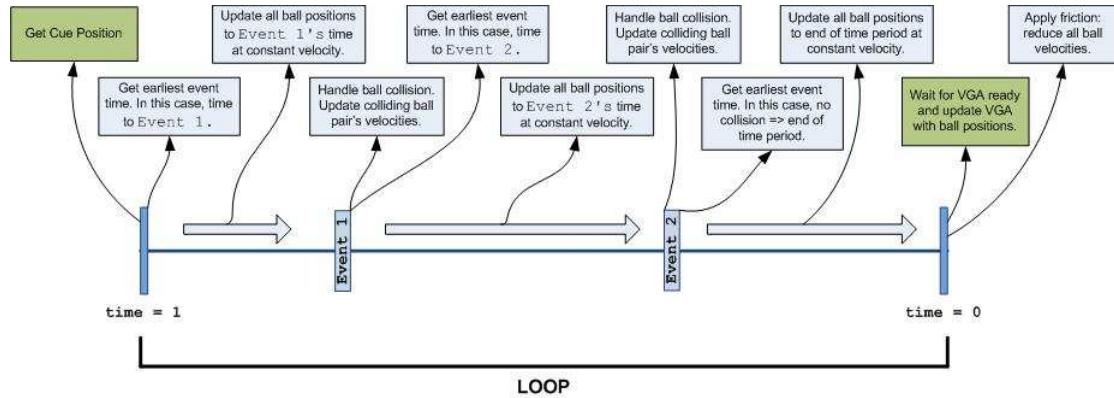


Figure 17: Collision Event Handling Loop

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.

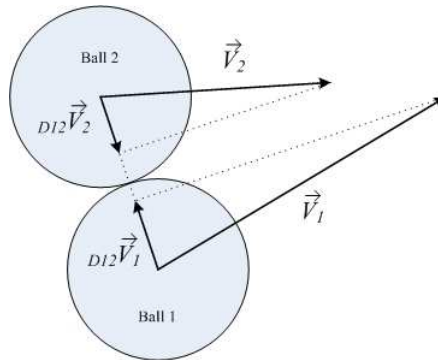


Figure 18: Ball Collisions

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

$$|V_{1D_{12}}| = \vec{V}_1 \cdot \vec{D}_{12}$$

Similarly, for the second ball,

$$|V_{2D_{12}}| = \vec{V}_2 \cdot \vec{D}_{12}$$

The velocity component for the balls along this direction are

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

The new velocities for the two balls are

$$\begin{aligned} \vec{V}_{1New} &= \vec{V}_1 - V_{1D_{12}New} \cdot \vec{D}_{12} \\ \vec{V}_{2New} &= \vec{V}_2 - V_{2D_{12}New} \cdot \vec{D}_{12} \end{aligned}$$

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>.

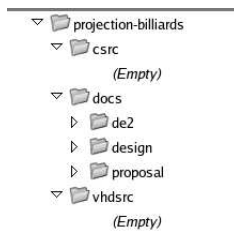


Figure 19: Directory Tree Structure

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.

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 * Desc:
11 */
12 #ifndef _GAME_CONFIG_H
13 #define _GAME_CONFIG_H
14 #include "fixedpoint.h"
15
16 extern long long tableStartX;
17 extern long long tableEndX;
18 extern long long tableStartY;
19 extern long long tableEndY;
20
21 extern long long camStartX;
22 extern long long camStartY;
23 extern long long camEndX;
24 extern long long camEndY;
25
26 #define TABLE_START_X      INT2FP(tableStartX)
27 #define TABLE_START_Y      INT2FP(tableStartY)
28 #define TABLE_END_X        INT2FP(tableEndX)
29 #define TABLE_END_Y        INT2FP(tableEndY)
30 #define TABLE_WIDTH        INT2FP(tableEndX - tableStartX)
31 #define TABLE_HEIGHT       INT2FP(tableEndY - tableStartY)
32
33 #define TIME_STEP           20 /* in milliseconds? */
34 #define RAW_POCKET_RADIUS   14
35 #define NUM_BALLS           7
36 #define BALL_RADIUS         INT2FP(14LL)
37
38 #define POCKET_RADIUS       INT2FP(10LL)
39
40 #define TOP_LEFT_POCKET_X   TABLE_START_X
41 #define TOP_LEFT_POCKET_Y   TABLE_START_Y
42 #define TOP_RIGHT_POCKET_X  TABLE_END_X
43 #define TOP_RIGHT_POCKET_Y  TABLE_START_Y
44 #define TOP_MID_POCKET_X    FPDIV((TABLE_START_X + TABLE_END_X), INT2FP(2LL));
45 #define TOP_MID_POCKET_Y    TABLE_START_Y
46 #define BOTTOM_LEFT_POCKET_X TABLE_START_X
47 #define BOTTOM_LEFT_POCKET_Y TABLE_END_Y
48 #define BOTTOM_RIGHT_POCKET_X TABLE_END_X
49 #define BOTTOM_RIGHT_POCKET_Y TABLE_END_Y
50 #define BOTTOM_MID_POCKET_X  TABLE_START_X + FPDIV(TABLE_WIDTH, INT2FP(2LL)) \
51                               - INT2FP(1LL)
52 #define BOTTOM_MID_POCKET_Y  TABLE_END_Y
53
54 #define DAMPING_COEFF        FPDIV(INT2FP(2LL), INT2FP(100LL))
55
56 #endif /* _GAME_CONFIG_H */

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 *
11 * Desc:
12 */
13 #ifndef _TYPES_H
14 #define _TYPES_H
15
16 typedef enum{
17     FALSE,
18     TRUE
19 }bool_t;
20
21 typedef enum{
22     NONE,
23     LEFT,
24     RIGHT,
25     TOP,
26     BOTIOM
27 }edge_t;
28
29 typedef enum{
30     NO_COLLISION,
31     BALL_COLLISION,
32     POCKET_COLLISION,
33     CUE_COLLISION,
34     TABLE_COLLISION
35 }event_t;
36
37 struct vector{
38     long long x;
39     long long y;
40 };
41
42 /*
43 // Every ball has the following properties
44 // Position (x,y) - The centre of the circle
45 // Velocity vector (x,y) - This is positive for a ball moving right and/or
46 // down, and negative for a ball moving left and/or up.
47 // Radius
48 // Visibility state
49 */
50 typedef enum {
51     BALL_VISIBLE = 0,
52     BALL_INVISIBLE = 1
53 }BallState_e;
54
55 struct ball_t{
56     struct vector pos;
57     struct vector vel;
58     long long radius;
59     BallState_e ballState;
60     unsigned char colour;
61     int points;
62 };
63
64 struct player_t{
65     int points;
66 };
67
68 #endif /* _TYPES_H */

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 *
11 * Desc:
12 */
13 #ifndef _DEBUG_H
14 #define _DEBUG_H
15
16 #include <stdio.h>
17 #include <assert.h>
18
19 #define DP_INFO          printf("%s, %s(): _", __FILE__, __FUNCTION__)
20 #define DP_PREFIX       printf(" (dbg)_", DP_INFO)
21 #define print(x)        DP_PREFIX, printf(x)
22 #define print1(x, x1)   DP_PREFIX, printf((x), (x1))
23 #define print2(x, x1, x2) DP_PREFIX, printf((x), (x1), (x2))
24 #define print3(x, x1, x2, x3) DP_PREFIX, printf((x), (x1), (x2), (x3))
25
26 #define DP_ASSERT(x, y) (x)?1:(print(y), assert(0));
27
28 #ifdef ALT_DEBUG
29 #define DP(x)            print(x)
30 #define DP1(x, x1)      print1((x), (x1))
31 #define DP2(x, x1, x2) print2((x), (x1), (x2))
32 #define DP3(x, x1, x2, x3) print3((x), (x1), (x2), (x3))
33 #define DP_HI           DP(" Enter\n");
34 #define DP_BYE          DP(" Leave\n");
35
36 #else /* ALT_DEBUG */
37
38 #define DP(x)
39 #define DP1(x, x1)
40 #define DP2(x, x1, x2)
41 #define DP3(x, x1, x2, x3)
42 #define DP_HI
43 #define DP_BYE
44
45 #endif /* ALT_DEBUG */
46 #endif /* _DEBUG_H */

```

```

2  /*
   * Software for Interactive Project Pool Game
   * Columbia University. New York, 2008
4  * Authors:
   *   Abdulhamid Ghandour
6  *   Thomas John
   *   Jaime Peretzman
8  *   Bharadwaj Vellore
   *
10 * Desc:
   */
12 #ifndef _BALL_H
13 #define _BALL_H
14
15 #include "types.h"
16 #include "system.h"
17 #include "io.h"
18
19 bool_t isBallMoving(const struct ball_t *ball);
20 long long collisionWithTableTime(const struct ball_t *ball, edge_t *edge);
21 long long collisionWithBallTime(const struct ball_t *ball1, const struct ball_t *ball2);
22 void handleBallCollision(struct ball_t *ball1, struct ball_t *ball2);
23 long long handleCollisionWithCue(struct ball_t *ball, const struct ball_t *cue);
24 void moveBalls(struct ball_t *balls, long long time);
25 void drawBalls(struct ball_t *balls);
26 void applyFriction(struct ball_t *balls);
27
28 #define BALL_X                0
29 #define BALL_Y                1
30 #define BALL_COLOUR          2
31
32 #define BALL_0_BASE           0
33 #define BALL_1_BASE           3
34 #define BALL_2_BASE           6
35 #define BALL_3_BASE           9
36 #define BALL_4_BASE           22
37 #define BALL_5_BASE           25
38 #define BALL_6_BASE           28
39
40 #define COL_WHITE             0
41 #define COL_YELLOW            1
42 #define COL_CYAN              2
43 #define COL_INVISIBLE         3
44 #define COL_K1                4
45 #define COL_K2                5
46 #define COL_K3                6
47
48
49 #define VGA_FLAG              12
50
51 #define SPRITE_X(ball)        FP2INT((ball).pos.x - (ball).radius)
52 #define SPRITE_Y(ball)        FP2INT((ball).pos.y - (ball).radius)
53
54 #define IOWR_POS(base, offset, data)  IOWR_16DIRECT(base, (offset) * 2, data)
55 #define IOWR_VAL(offset, data)       IOWR_POS(VGA_BASE, offset, data)
56 #endif /* _BALL_H */

```



```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 *
11 * Desc:
12 */
13 #ifndef I2C_H_
14 #define I2C_H_
15
16 /*
17 * Start Bit
18 */
19 #define START \
20     SDAT_SET; \
21     HALF_CLOCK_DELAY; \
22     SCLK_SET; \
23     HALF_CLOCK_DELAY; \
24     SDAT_CLR; \
25     HALF_CLOCK_DELAY; \
26     SCLK_CLR; \
27     HALF_CLOCK_DELAY; \
28
29 /*
30 * Stop Bit
31 */
32 #define STOP \
33     SDAT_CLR; \
34     HALF_CLOCK_DELAY; \
35     SCLK_SET; \
36     HALF_CLOCK_DELAY; \
37     SDAT_SET; \
38
39 /*
40 * Sequence for a '1' bit
41 */
42 #define SEND_BIT_1 \
43     SDAT_SET; \
44     HALF_CLOCK_DELAY; \
45     SCLK_SET; \
46     ONE_CLOCK_DELAY; \
47     SCLK_CLR; \
48     HALF_CLOCK_DELAY; \
49     //SDAT_CLR; \
50
51 /*
52 * Sequence for a '0' bit
53 */
54 #define SEND_BIT_0 \
55     SDAT_CLR; \
56     HALF_CLOCK_DELAY; \
57     SCLK_SET; \
58     ONE_CLOCK_DELAY; \
59     SCLK_CLR; \
60     HALF_CLOCK_DELAY; \
61     //SDAT_SET; \
62
63 #define SEND_0 \
64     SEND_BIT_0; \
65     SEND_BIT_0; \
66     SEND_BIT_0; \
67     SEND_BIT_0; \
68
69 #define SEND_1 \
70     SEND_BIT_0; \
71     SEND_BIT_0; \
72     SEND_BIT_0; \
73     SEND_BIT_1; \
74
75 #define SEND_2 \
76     SEND_BIT_0; \
77     SEND_BIT_0; \
78     SEND_BIT_1; \
79     SEND_BIT_0; \

```

```

80 #define SEND_3 \
    SEND_BIT_0; \
82    SEND_BIT_0; \
    SEND_BIT_1; \
84    SEND_BIT_1;

86 #define SEND_4 \
    SEND_BIT_0; \
88    SEND_BIT_1; \
    SEND_BIT_0; \
90    SEND_BIT_0;

92 #define SEND_5 \
    SEND_BIT_0; \
94    SEND_BIT_1; \
    SEND_BIT_0; \
96    SEND_BIT_1;

98 #define SEND_6 \
    SEND_BIT_0; \
100    SEND_BIT_1; \
    SEND_BIT_1; \
102    SEND_BIT_0;

104 #define SEND_7 \
    SEND_BIT_0; \
106    SEND_BIT_1; \
    SEND_BIT_1; \
108    SEND_BIT_1;

110 #define SEND_8 \
    SEND_BIT_1; \
112    SEND_BIT_0; \
    SEND_BIT_0; \
114    SEND_BIT_0;

116 #define SEND_9 \
    SEND_BIT_1; \
118    SEND_BIT_0; \
    SEND_BIT_0; \
120    SEND_BIT_1;

122 #define SEND_A \
    SEND_BIT_1; \
124    SEND_BIT_0; \
    SEND_BIT_1; \
126    SEND_BIT_0;

128 #define SEND_B \
    SEND_BIT_1; \
130    SEND_BIT_0; \
    SEND_BIT_1; \
132    SEND_BIT_1;

134 #define SEND_C \
    SEND_BIT_1; \
136    SEND_BIT_1; \
    SEND_BIT_0; \
138    SEND_BIT_0;

140 #define SEND_D \
    SEND_BIT_1; \
142    SEND_BIT_1; \
    SEND_BIT_0; \
144    SEND_BIT_1;

146 #define SEND_E \
    SEND_BIT_1; \
148    SEND_BIT_1; \
    SEND_BIT_1; \
150    SEND_BIT_0;

152 #define SEND_F \
    SEND_BIT_1; \
154    SEND_BIT_1; \
    SEND_BIT_1; \
156    SEND_BIT_1;

158 #define READ(ack) \
    SDAT_TRISTATE; \
    \

```

```

160     HALF_CLOCK_DELAY;  \
      SCLK_SET;         \
162     HALF_CLOCK_DELAY;  \
      ack = RD_ACK;     \
164     HALF_CLOCK_DELAY;  \
      SCLK_CLR;         \
166     HALF_CLOCK_DELAY;

168 #define ACK           \
      SDAT_CLR;         \
170     HALF_CLOCK_DELAY;  \
      SCLK_SET;         \
172     ONE_CLOCK_DELAY;  \
      SCLK_CLR;         \
174     HALF_CLOCK_DELAY;

176 #define NACK         \
      SDAT_SET;         \
178     HALF_CLOCK_DELAY;  \
      SCLK_SET;         \
180     ONE_CLOCK_DELAY;  \
      SCLK_CLR;         \
182     HALF_CLOCK_DELAY;  \
      SDAT_CLR;

184 #define COMM_INIT    \
186     START;           \
      SEND_B;           \
188     SEND_A;

190 #define READ_ACK(ack)  READ(ack)

192 int configureCamera();

194 #endif /* I2C_H */

```

```

2  /*
   * Software for Interactive Project Pool Game
   * Columbia University. New York, 2008
4  * Authors:
   *   Abdulhamid Ghandour
6  *   Thomas John
   *   Jaime Peretzman
8  *   Bharadwaj Vellore
   *
10 * Desc:
   */
12 #ifndef _FIXED_POINT_H
13 #define _FIXED_POINT_H
14
15 #define FRAC_PRECISION          12
16 #define MAG_PRECISION          (64 - FRAC_PRECISION)
17
18 #define FPSUM(x,y)              ((x) + (y))
19 #define FPSUB(x,y)              ((x) - (y))
20 #define FPMUL(x,y)              (((x) * (y)) >> FRAC_PRECISION)
21 #define FPDIV(x,y)              (((x) << FRAC_PRECISION) / (y))
22 #define FPSQR(x)                FPMUL((x),(x))
23 #define FP2INT(x)               ((x) >> FRAC_PRECISION)
24 #define INT2FP(x)               ((x) << FRAC_PRECISION)
25
26 long long FPSQRT(long long num);
27 void printFP(long long fpnum);
28
29 #endif /* _FIXED_POINT_H */

```

```

2  #ifndef _CALIBRATION_H_
3  #define _CALIBRATION_H_
4  #include "system.h"
5  #include "io.h"
6
7  #define NO_CUE_DETECTED      (IORD_32DIRECT(VISION_BASE, 0) >> 31)
8
9  #define START_CALIBRATION   IOWR_32DIRECT(VISION_BASE, 4 * 4, 1)
10 #define STOP_CALIBRATION    IOWR_32DIRECT(VISION_BASE, 4 * 4, 0)
11 #define READ_REPOS_REG       IORD_32DIRECT(VISION_BASE, 3 * 4)
12 #define READ_CAMERA_START    IORD_32DIRECT(VISION_BASE, 1 * 4)
13 #define READ_CAMERA_END      IORD_32DIRECT(VISION_BASE, 2 * 4)
14
15 #define SET_CAMERA_SANDBOX_START_X(data) \
16     IOWR_32DIRECT(VISION_BASE, 5 * 4, (data))
17 #define SET_CAMERA_SANDBOX_END_X(data) \
18     IOWR_32DIRECT(VISION_BASE, 6 * 4, (data))
19 #define SET_CAMERA_SANDBOX_START_Y(data) \
20     IOWR_32DIRECT(VISION_BASE, 7 * 4, (data))
21 #define SET_CAMERA_SANDBOX_END_Y(data) \
22     IOWR_32DIRECT(VISION_BASE, 8 * 4, (data))
23
24 #define TABLE_START_HOZ_POS 140
25 #define TABLE_END_HOZ_POS 500
26 #define TABLE_START_VER_POS 120
27 #define TABLE_END_VER_POS 360
28
29 #define CAMERA_CROP_MARGIN_HOZ 10
30 #define CAMERA_CROP_MARGIN_VER 20
31 #define BLACK_MARGIN_HOZ 110
32 #define BLACK_MARGIN_VER 90
33
34 #define BLACK_MARGIN_CONFIG ((BLACK_MARGIN_VER << 8) | BLACK_MARGIN_HOZ)
35 #define BLACK_MARGIN_REG 31
36 #define BLACK_MARGIN_SET(size) IOWR_16DIRECT(VGA_BASE, 31 * 2, size)
37
38 #define GREEN_COLUMN_THRESHOLD 120
39 #define GREEN_ROW_THRESHOLD 240
40
41 #define SET_GREEN_ROW_THR IOWR_32DIRECT(VISION_BASE, 10 * 4, (GREEN_ROW_THRESHOLD))
42 #define SET_GREEN_COL_THR IOWR_32DIRECT(VISION_BASE, 9 * 4, (GREEN_COLUMN_THRESHOLD))
43
44 void doCalibration();
45 int calibrate();
46 #endif /*CALIBRATION_H_*/

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 *
11 * Desc: LCD implementation borrowed from code by Prof. Stephen Edwards,
12 *   Columbia University
13 */
14 #ifndef _UI_H_
15 #define _UI_H_
16 #include "system.h"
17 #include "io.h"
18 #include "ball.h"
19
20 // LCD Module 16*2
21 #define lcd_write_cmd(base, data)          IOWR(base, 0, data)
22 #define lcd_read_cmd(base)                IORD(base, 1)
23 #define lcd_write_data(base, data)        IOWR(base, 2, data)
24 #define lcd_read_data(base)                IORD(base, 3)
25
26 #define IOWR_LED_DATA(base, offset, data) \
27     IOWR_16DIRECT(base, (offset) * 2, data)
28 #define IORD_LED_DATA(base, offset) \
29     IORD_16DIRECT(base, (offset) * 2)
30 #define IOWR_LED_SPEED(base, data) \
31     IOWR_16DIRECT(base + 32, 0, data)
32 #define IORD_FLAG(base, offset) \
33     IORD_16DIRECT(base, (offset) * 2)
34
35 #define PLAY_SOUND                IOWR_32DIRECT(SOUND_DRIVER_BASE, 0, 1)
36 #define HEXWRITE(reg, data)        IOWR_32DIRECT(UI_CONTROL_BASE, (reg) * 4, data)
37
38 #define HEX0(data)                 HEXWRITE(0, (data))
39 #define HEX1(data)                 HEXWRITE(1, (data))
40 #define HEX2(data)                 HEXWRITE(2, (data))
41 #define HEX3(data)                 HEXWRITE(3, (data))
42 #define HEX4(data)                 HEXWRITE(4, (data))
43 #define HEX5(data)                 HEXWRITE(5, (data))
44 #define HEX6(data)                 HEXWRITE(6, (data))
45 #define HEX7(data)                 HEXWRITE(7, (data))
46
47 #define CALIBRATION_REQUESTED      IORD_32DIRECT(UI_CONTROL_BASE, 8 * 4)
48 #define CALIBRATED                  IOWR_32DIRECT(UI_CONTROL_BASE, 8 * 4, 1)
49 #define NEW_GAME_REQUESTED         IORD_32DIRECT(UI_CONTROL_BASE, 9 * 4)
50 #define STARTED_GAME                IOWR_32DIRECT(UI_CONTROL_BASE, 9 * 4, 1)
51
52 extern unsigned long sevensegment [];
53
54 void LCD_Init();
55 void LCD_Show_Text(char* Text);
56 void LCD_Line2();
57 void initPointsDisplay();
58
59 #endif /* _UI_H_ */

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 * Desc:
11 */
12 #include <stdio.h>
13 #include "debug.h"
14 #include "fixedpoint.h"
15 #include "gameconfig.h"
16 #include "types.h"
17 #include "ball.h"
18 #include "i2c.h"
19 #include "ui.h"
20 #include "calibration.h"
21
22 #define NO_CUE_DETECTED (IORD_32DIRECT(VISION_BASE, 0) >> 31)
23
24 static void initPockets();
25 static void initBalls();
26 void initPlayers();
27 static void initCue();
28 static int play();
29 extern int calibrate();
30 struct ball_t balls[NUM_BALLS];
31 struct ball_t cue;
32 struct ball_t pockets[6];
33 struct player_t player1, player2;
34 struct player_t *pCurrentPlayer;
35
36 typedef enum gameState_e {
37     GAME_PLAYING,
38     GAME_WAITING_TO_PLAY
39 } gameState_t;
40 gameState_t gameState = GAME_WAITING_TO_PLAY;
41
42 void showPoints(struct player_t *player);
43
44 int main(){
45     DP_HI;
46
47     DP("Welcome to Projection Pool\n");
48
49     LCD_Init();
50     initPointsDisplay();
51
52     LCD_Show_Text("Welcome to Pool!");
53
54     initPlayers();
55     configureCamera();
56     doCalibration();
57     CALIBRATED;
58     initCue();
59     initPockets();
60     initBalls();
61     initPointsDisplay();
62     drawBalls(balls);
63     STARTED_GAME;
64
65     while(1){
66         if(-1 == play()){
67             doCalibration();
68             CALIBRATED;
69             drawBalls(balls);
70         }else{
71             initPlayers();
72             initCue();
73             initBalls();
74             initPointsDisplay();
75             drawBalls(balls);
76             STARTED_GAME;
77         }
78     }

```

```

80 | DP_BYE;
    | }
82 |
    | long long calibScaleHoz , calibOffsetHoz;
84 | long long calibScaleVer , calibOffsetVer;
    | void configureScaling(){
86 |     long long vgaHozRes = 640;
    |     long long vgaVerRes = 480;
88 |     long long blackMarginHoz = BLACK_MARGIN_HOZ;
    |     long long blackMarginVer = BLACK_MARGIN_VER;
90 |
    |     calibScaleHoz = FPDIV(
92 |         INT2FP(vgaHozRes - 2*blackMarginHoz) ,
    |         INT2FP(camEndX - camStartX)
94 |     );
    |     calibOffsetHoz =
96 |         FPMUL(calibScaleHoz ,INT2FP(CAMERA_CROP_MARGIN_HOZ)) +
    |         INT2FP(blackMarginHoz);
98 |
    |     calibScaleVer = FPDIV(
100 |         INT2FP(vgaVerRes - 2*blackMarginVer) ,
    |         INT2FP(camEndY - camStartY)
102 |     );
    |     calibOffsetVer =
104 |         FPMUL(calibScaleVer ,INT2FP(CAMERA_CROP_MARGIN_VER)) +
    |         INT2FP(blackMarginVer);
106 | }

108 | int play(){
    |     int i , j;
110 |     long long time , earliestEventTime , eventTime;
    |     event_t eventType;
112 |     struct ball_t *collidingBall1 , *collidingBall2;
    |     edge_t tableEdge , collisionTableEdge;
114 |     bool_t ballsMoved = FALSE;
    |     bool_t repositionedWhite;
116 |
    |     long long xin = 0 , prevXin = 0;
118 |     long long yin = 0 , prevYin = 0;
    |     long long xinBeforeFilter = 0 , yinBeforeFilter = 0;
120 |     long long A = FPDIV(INT2FP(18) ,INT2FP(10));
    |     long long W0 , W1 , W2 , W3;
122 |
    |     W0 = FPDIV(INT2FP(5) ,INT2FP(10));
124 |     W1 = FPDIV(INT2FP(3) ,INT2FP(10));
    |     W2 = FPDIV(INT2FP(1) ,INT2FP(10));
126 |     W3 = FPDIV(INT2FP(1) ,INT2FP(10));

128 |     struct vector prevCuePos1 , prevCuePos2 , prevCuePos3;
    |     prevCuePos1.x = 0;
130 |     prevCuePos1.y = 0;
    |     prevCuePos2.x = 0;
132 |     prevCuePos2.y = 0;
    |     prevCuePos3.x = 0;
134 |     prevCuePos3.y = 0;

136 |     long long prevXinBeforeFilter = 0 , prevYinBeforeFilter = 0;
    |     long long absPreFilterXDiff , absPreFilterYDiff;
138 |
    |     int numBallsPocketed = 0;
140 |     gameState = GAME_WAITING_TO_PLAY;

142 |     configureScaling();
    |     LCD_Init();
144 |     LCD_Show_Text(" Player");
    |     LCD_Line2();
146 |     LCD_Show_Text(" 1");

148 |     while(1){
    |         time = INT2FP(1LL);
150 |         if(!NO_CUE_DETECTED){
    |             xin = (long long)IORD_32DIRECT(VISION_BASE , 0);
152 |             yin = ((xin >> 16) & 0x7FFF);
    |             xin = (xin & 0xFFFF);
154 |             xinBeforeFilter = FPMUL(calibScaleHoz ,INT2FP(xin)) + calibOffsetHoz;
    |             yinBeforeFilter = FPMUL(calibScaleVer ,INT2FP(yin)) + calibOffsetVer;
156 |
    |             /*
158 |              * Filter to limit step changes in cue position
    |              */

```



```

160     absPreFilterXDiff = (xinBeforeFilter > prevXinBeforeFilter)?
162         (xinBeforeFilter - prevXinBeforeFilter):
            (prevXinBeforeFilter > xinBeforeFilter);
164     absPreFilterYDiff = (yinBeforeFilter > prevYinBeforeFilter)?
            (yinBeforeFilter - prevYinBeforeFilter):
            (prevYinBeforeFilter > yinBeforeFilter);
166
168     if((prevXinBeforeFilter != 0) && (prevYinBeforeFilter !=0)){
170         if((absPreFilterXDiff > INT2FP(75)) || (absPreFilterYDiff > INT2FP(75))){
172             xinBeforeFilter = prevXinBeforeFilter;
174             yinBeforeFilter = prevYinBeforeFilter;
176         }else{
178             prevXinBeforeFilter = xinBeforeFilter;
180             prevYinBeforeFilter = yinBeforeFilter;
182         }
184     }else{
186         prevXinBeforeFilter = xinBeforeFilter;
188         prevYinBeforeFilter = yinBeforeFilter;
190     }
192
194     /*
196     * Smoothing filter for cue position
198     */
200     xin = FPMUL(W0, xinBeforeFilter) +
202         FPMUL(W1, prevCuePos1.x) +
204         FPMUL(W2, prevCuePos2.x) +
206         FPMUL(W3, prevCuePos3.x);
208     yin = FPMUL(W0, yinBeforeFilter) +
210         FPMUL(W1, prevCuePos1.y) +
212         FPMUL(W2, prevCuePos2.y) +
214         FPMUL(W3, prevCuePos3.y);
216
218     prevCuePos3.x = prevCuePos2.x;
220     prevCuePos3.y = prevCuePos2.y;
222     prevCuePos2.x = prevCuePos1.x;
224     prevCuePos2.y = prevCuePos1.y;
226     prevCuePos1.x = xinBeforeFilter;
228     prevCuePos1.y = yinBeforeFilter;
230
232     cue.ballState = BALL_VISIBLE;
234
236     if ((prevXin != 0) && (prevYin != 0)){
238         cue.pos.x = prevXin;
240         cue.pos.y = prevYin;
242         if(IORD_32DIRECT(UICONTROL_BASE, 10 * 4) & 0x00000400){
244             IOWR_POS(VGA_BASE, 13, FP2INT(prevXin));
246             IOWR_POS(VGA_BASE, 14, FP2INT(prevYin));
248         }
250         else{
252             IOWR_POS(VGA_BASE, 13, 999);
254             IOWR_POS(VGA_BASE, 14, 999);
256         }
258         cue.vel.x = FPMUL(A,(xin - prevXin));
260         cue.vel.y = FPMUL(A,(yin - prevYin));
262         prevXin = xin;
264         prevYin = yin;
266     }
268     else{
270         prevXin = xin;
272         prevYin = yin;
274     }
276 }else{
278     cue.ballState = BALL_INVISIBLE;
280     prevXin = 0;
282     prevYin = 0;
284     prevXinBeforeFilter = 0;
286     prevYinBeforeFilter = 0;
288 }
290
292 ballsMoved = FALSE;
294 do{
296     earliestEventTime = time;
298     eventType = NO_COLLISION;
300     collidingBall1 = NULL;
302     collidingBall2 = NULL;
304     tableEdge = NONE;
306
308     for(i=0; i<NUM_BALLS; i++){
310         if(BALL_INVISIBLE == balls[i].ballState){
312             continue;

```

```

240     }
241     if(isBallMoving(&balls[i])){
242         ballsMoved = TRUE;
243     }
244
245     /*
246     // Check for collisions with the table boundaries
247     */
248     if(isBallMoving(&balls[i]) == TRUE){
249         eventTime = collisionWithTableTime(&balls[i],&tableEdge);
250         if((eventTime >= 0) && (eventTime < earliestEventTime)){
251             earliestEventTime = eventTime;
252             collidingBall1 = &balls[i];
253             collidingBall2 = NULL;
254             eventType = TABLE_COLLISION;
255             collisionTableEdge = tableEdge;
256         }
257     }
258
259     /*
260     // Check for "collision" with cue
261     */
262     if(BALL_INVISIBLE != cue.ballState){
263         if((balls[i].colour == COL_WHITE) ||
264            ((IORD_32DIRECT(UICONTROL_BASE, 10 * 4) & 0x1000) == 0)
265        ){
266             if(((balls[i].vel.x == 0) && (balls[i].vel.y == 0)) ||
267                ((IORD_32DIRECT(UICONTROL_BASE, 10 * 4) & 0x800) == 0)
268            ){
269                 eventTime = collisionWithBallTime(&balls[i],&cue);
270                 if((eventTime >= 0) && (eventTime < earliestEventTime)){
271                     earliestEventTime = eventTime;
272                     collidingBall1 = &balls[i];
273                     collidingBall2 = &cue;
274                     eventType = CUE_COLLISION;
275                 }
276             }
277         }
278     }
279
280     /*
281     // Check for "collision" with pockets
282     */
283     if(isBallMoving(&balls[i]) == TRUE){
284         for(j=0; j<6; j++){
285             eventTime = collisionWithBallTime(&balls[i], &pockets[j]);
286             if((eventTime >= 0) && (eventTime < earliestEventTime)){
287                 earliestEventTime = eventTime;
288                 collidingBall1 = &balls[i];
289                 collidingBall2 = &pockets[j];
290                 eventType = POCKET_COLLISION;
291             }
292         }
293     }
294
295     /*
296     // Collision with other balls
297     */
298     if(isBallMoving(&balls[i]) == TRUE){
299         for(j=0; j<NUMBALLS; j++){
300             if(BALL_INVISIBLE == balls[j].ballState){
301                 continue;
302             }
303
304             eventTime = collisionWithBallTime(&balls[i],&balls[j]);
305             if((eventTime >= 0) && (eventTime < earliestEventTime)){
306                 earliestEventTime = eventTime;
307                 collidingBall1 = &balls[i];
308                 collidingBall2 = &balls[j];
309                 eventType = BALL_COLLISION;
310             }
311         }
312     }
313 }
314
315 moveBalls(balls , earliestEventTime);
316
317 switch(eventType){
318     case NO_COLLISION:
319         break;

```

```

320
321 case POCKET_COLLISION:
322     collidingBall1->ballState = BALL_INVISIBLE;
323     numBallsPocketed++;
324     PLAY_SOUND;
325     pCurrentPlayer->points += collidingBall1->points;
326     showPoints(pCurrentPlayer);

327
328     if(collidingBall1->colour == COL_WHITE){
329         int k;
330         repositionedWhite = TRUE;
331         numBallsPocketed--;
332         collidingBall1->ballState = BALL_VISIBLE;
333         collidingBall1->pos.x = TABLE_END_X - INT2FP(140);
334         collidingBall1->pos.y = TABLE_END_Y - INT2FP(120);
335         printf("New_Pos_=%11d ,%11d\n" ,
336             FP2INT(collidingBall1->pos.x) ,
337             FP2INT(collidingBall1->pos.y)
338         );
339         collidingBall1->vel.x = 0;
340         collidingBall1->vel.y = 0;
341         while(1){
342             long long absXDiff , absYDiff;
343             for(k=0; k<NUM_BALLS; k++){
344                 if(&balls[k] == collidingBall1){
345                     continue;
346                 }
347                 absXDiff = (collidingBall1->pos.x > balls[k].pos.x)?
348                     (collidingBall1->pos.x - balls[k].pos.x):
349                     (balls[k].pos.x - collidingBall1->pos.x);
350                 absYDiff = (collidingBall1->pos.y > balls[k].pos.y)?
351                     (collidingBall1->pos.y - balls[k].pos.y):
352                     (balls[k].pos.y - collidingBall1->pos.y);
353                 if((absXDiff < INT2FP(30)) && (absYDiff < INT2FP(30))){
354                     repositionedWhite = FALSE;
355                     break;
356                 }
357             }
358             if(repositionedWhite == FALSE){
359                 collidingBall1->pos.x += INT2FP(15);
360                 collidingBall1->pos.y += INT2FP(10);
361                 if(collidingBall1->pos.x > (TABLE_END_X - INT2FP(20))){
362                     collidingBall1->pos.x -= INT2FP(20);
363                 }
364                 if(collidingBall1->pos.y > (TABLE_END_Y - INT2FP(20))){
365                     collidingBall1->pos.y -= INT2FP(15);
366                 }
367                 printf("New_Pos_=%11d ,%11d\n" ,
368                     FP2INT(collidingBall1->pos.x) ,
369                     FP2INT(collidingBall1->pos.y)
370                 );
371                 repositionedWhite = TRUE;
372             } else{
373                 break;
374             }
375         }
376     }
377     break;

378 case CUE_COLLISION:
379     handleCollisionWithCue(collidingBall1 ,&cue);
380     PLAY_SOUND;
381     break;

382
383 case TABLE_COLLISION:
384     switch(collisionTableEdge){
385     case LEFT:
386     case RIGHT:
387         collidingBall1->vel.x *= -1LL;
388         break;

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

393     default:
394         DP_ASSERT(0, "Collision with non-existent table edge!");
395         break;
396     }
397 }

```

```

400     PLAY_SOUND;
401     break;
402
403     case BALL_COLLISION:
404         handleBallCollision(collidingBall1, collidingBall2);
405         PLAY_SOUND;
406     break;
407
408     default:
409         DP_ASSERT(0, "Invalid event");
410     break;
411 };
412 drawBalls(balls);
413 time -= earliestEventTime;
414 }while(time > 0);
415 drawBalls(balls);
416 applyFriction(balls);
417
418 /*
419  * Check if there have been balls moved in the last time step
420  * If yes, and if the previous state was WAITING_TO_PLAY, then
421  * switch to PLAYING_STATE. Continue with the same player.
422  * If not, and if the previous state was PLAYING, then switch
423  * to the WAITING_TO_PLAY. Also switch players.
424  */
425 if((GAME_PLAYING == gameState) && (FALSE == ballsMoved)){
426     gameState = GAME_WAITING_TO_PLAY;
427     if(pCurrentPlayer == &player1){
428         pCurrentPlayer = &player2;
429         LCD_Line2();
430         LCD_Show_Text("2");
431     }else if(pCurrentPlayer == &player2){
432         pCurrentPlayer = &player1;
433         LCD_Line2();
434         LCD_Show_Text("1");
435     }else{
436         fflush(stdout);
437         DP_ASSERT(0, "Invalid player\n");
438     }
439 }else if((GAME_WAITING_TO_PLAY == gameState) && (TRUE == ballsMoved)){
440     gameState = GAME_PLAYING;
441 }else{
442     /*
443     * Do nothing
444     */
445 }
446
447 /*
448  * Check if there has been a request for re-calibration
449  * and return -1 if yes
450  */
451 if(CALIBRATION_REQUESTED == 0){
452     if(numBallsPocketed == NUM_BALLS - 1){
453         initBalls();
454         initPlayers();
455         initCue();
456         initPointsDisplay();
457     }
458     return -1;
459 }
460
461 if(NEW_GAME_REQUESTED == 0){
462     return 0;
463 }
464
465 if(numBallsPocketed == NUM_BALLS - 1){
466     LCD_Init();
467     LCD_Show_Text("Player");
468     if(player1.points > player2.points){
469         LCD_Show_Text("1 WINS!");
470     }else if(player1.points < player2.points){
471         LCD_Show_Text("2 WINS!");
472     }else{
473         LCD_Show_Text("S_TIE!");
474     }
475     while(NEW_GAME_REQUESTED == 1);
476     return 0;
477 }
478 }

```

```

480 | static void initPockets(){
482 |     int i;
484 |     for(i=0; i<6; i++){
486 |         pockets[i].vel.x = 0;
488 |         pockets[i].vel.y = 0;
488 |         pockets[i].radius = POCKET_RADIUS;
490 |         pockets[i].colour = 1;
490 |         pockets[i].ballState = BALL_INVISIBLE;
492 |     }
492 |     pockets[0].pos.x = TOP_LEFT_POCKET_X;
494 |     pockets[0].pos.y = TOP_LEFT_POCKET_Y;
494 |     pockets[1].pos.x = TOP_MID_POCKET_X;
496 |     pockets[1].pos.y = TOP_MID_POCKET_Y;
496 |     pockets[2].pos.x = TOP_RIGHT_POCKET_X;
498 |     pockets[2].pos.y = TOP_RIGHT_POCKET_Y;
498 |     pockets[3].pos.x = BOTTOM_LEFT_POCKET_X;
500 |     pockets[3].pos.y = BOTTOM_LEFT_POCKET_Y;
500 |     pockets[4].pos.x = BOTTOM_MID_POCKET_X;
502 |     pockets[4].pos.y = BOTTOM_MID_POCKET_Y;
502 |     pockets[5].pos.x = BOTTOM_RIGHT_POCKET_X;
504 |     pockets[5].pos.y = BOTTOM_RIGHT_POCKET_Y;
504 | }
506 | static void initBalls(){
508 |     int i;
508 |     for(i=0; i<NUM_BALLS; i++){
510 |         balls[i].radius = BALL_RADIUS;
512 |         balls[i].ballState = BALL_VISIBLE;
512 |     }
514 |     balls[0].colour = COL_WHITE;
514 |     balls[0].points = -10;
516 |     balls[1].colour = COL_YELLOW;
518 |     balls[1].points = 20;
520 |     balls[2].colour = COL_CYAN;
520 |     balls[2].points = 5;
522 |     balls[3].colour = COL_K3;
524 |     balls[3].points = 10;
526 |     balls[4].colour = COL_K3;
526 |     balls[4].points = 10;
528 |     balls[5].colour = COL_CYAN;
530 |     balls[5].points = 5;
532 |     balls[6].colour = COL_CYAN;
532 |     balls[6].points = 5;
534 |     balls[0].pos.x = INT2FP(tableStartX + 240);
536 |     balls[0].pos.y = INT2FP(tableStartY + 120);
536 |     balls[0].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
538 |     balls[0].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
540 |     balls[1].pos.x = INT2FP(tableStartX + 40);
540 |     balls[1].pos.y = INT2FP(tableStartY + 120);
542 |     balls[1].vel.x = FPDIV(INT2FP(0LL),INT2FP(2LL));
542 |     balls[1].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
544 |     balls[2].pos.x = INT2FP(tableStartX + 40);
546 |     balls[2].pos.y = INT2FP(tableStartY + 160);
546 |     balls[2].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
548 |     balls[2].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
550 |     balls[3].pos.x = INT2FP(tableStartX + 100);
550 |     balls[3].pos.y = INT2FP(tableStartY + 100);
552 |     balls[3].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
552 |     balls[3].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
554 |     balls[4].pos.x = INT2FP(tableStartX + 100);
556 |     balls[4].pos.y = INT2FP(tableStartY + 140);
556 |     balls[4].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
558 |     balls[4].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
558 |     balls[5].pos.x = INT2FP(tableStartX + 150);

```

```

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

564 | balls[6].pos.x = INT2FP(tableStartX + 40);
    | balls[6].pos.y = INT2FP(tableStartY + 80);
566 | balls[6].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
    | balls[6].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
568 | }

570 | static void initCue(){
    | cue.colour = 0LL;
572 | cue.pos.x = INT2FP(10LL);
    | cue.pos.y = INT2FP(10LL);
574 | cue.radius = INT2FP(2LL);
    | cue.vel.x = 0LL;
576 | cue.vel.y = 0LL;
    | }
578 |

580 | void initPlayers(){
    | player1.points = 0;
    | player2.points = 0;
582 | pCurrentPlayer = &player1;
    | }
584 |

586 | void showPoints(struct player_t *player){
    | if(player == &player1){
    |     if(player->points < 0){
588 |         HEX5(sevenssegment[0]);
    |         HEX4(sevenssegment[0]);
590 |     }else{
    |         HEX5(sevenssegment[player->points/10]);
592 |         HEX4(sevenssegment[player->points%10]);
    |     }
594 | }else if(player == &player2){
    |     if(player->points < 0){
596 |         HEX1(sevenssegment[0]);
    |         HEX0(sevenssegment[0]);
598 |     }else{
    |         HEX1(sevenssegment[player->points/10]);
600 |         HEX0(sevenssegment[player->points%10]);
    |     }
602 | }else{
    |     DP_ASSERT(0,"Invalid player");
604 | }
    | }

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 * Desc:
11 */
12 #include <stdio.h>
13 #include <unistd.h>
14
15 #include "system.h"
16 #include "io.h"
17 #include "fixedpoint.h"
18 #include "ui.h"
19 #include "calibration.h"
20
21 long long tableStartX;
22 long long tableEndX;
23 long long tableStartY;
24 long long tableEndY;
25
26 long long camStartX;
27 long long camEndX;
28 long long camStartY;
29 long long camEndY;
30
31 void doCalibration(){
32     int calibrated = 0;
33     unsigned long repos, last_repos;
34     unsigned long counter = 0;
35     long camStart = 0xFFFF, camEnd = 0xFFFF;
36     long blackMarginSize;
37
38     SET_GREEN_ROW_THR;
39     SET_GREEN_COL_THR;
40
41     blackMarginSize = BLACK_MARGIN_CONFIG;
42     BLACK_MARGIN_SET(blackMarginSize);
43
44     tableStartX = TABLE_START_HOZ_POS;
45     tableEndX = TABLE_END_HOZ_POS;
46     tableStartY = TABLE_START_VER_POS;
47     tableEndY = TABLE_END_VER_POS;
48
49     SET_CAMERA_SANDBOX_START_X(0);
50     SET_CAMERA_SANDBOX_END_X(640);
51     SET_CAMERA_SANDBOX_START_Y(0);
52     SET_CAMERA_SANDBOX_END_Y(1024);
53
54     IOWR_POS(VGA_BASE, 16, tableStartX);
55     IOWR_POS(VGA_BASE, 18, tableEndX);
56     IOWR_POS(VGA_BASE, 17, tableStartY);
57     IOWR_POS(VGA_BASE, 19, tableEndY);
58     IOWR_POS(VGA_BASE, 21, 1);
59
60     LCD_Init();
61     LCD_Show_Text(" Calibrating .. ");
62
63     usleep(200000);
64
65     START_CALIBRATION;
66
67     /*
68     * Check reposition register and direct user to move camera
69     * We use a counter here to make sure we check several times
70     * to confirm that calibration is indeed complete and the
71     * state is steady.
72     */
73     while(!calibrated){
74         repos = READ_REPOS_REG;
75         if(repos == 0){
76             counter++;
77             if(counter < 20){
78                 usleep(200000);
79                 printf(" Calibrating .. Please wait ....\n");

```

```

80     LCD_Init();
81     LCD_Show_Text(" Calibrating");
82     LCD_Line2();
83     LCD_Show_Text(" Please_wait...");
84     continue;
85 }else{
86     calibrated = 1;
87 }
88 last_repos = 0;
89 }else{
90     counter = 0;
91     /*
92     * Find out the type of error and display message here
93     */
94     if(repos != last_repos){
95         printf(" Repos=%0x%x\n", (unsigned int)repos);
96
97         LCD_Init();
98         LCD_Show_Text(" Move_Camera");
99         LCD_Line2();
100
101         if((repos & 1) > 0){
102             printf(" Right");
103             LCD_Show_Text(" Right");
104         }
105         if((repos & 2) > 0){
106             printf(" Left");
107             LCD_Show_Text(" Left");
108         }
109         if((repos & 4) > 0){
110             printf(" Down_or_back");
111             LCD_Show_Text(" Down_or_Back");
112         }
113         if((repos & 8) > 0){
114             printf(" Up_or_back");
115             LCD_Show_Text(" Up_or_Back");
116         }
117         if((repos & 16) > 0){
118             printf(" Backwards");
119             LCD_Show_Text(" Backwards");
120         }
121         if((repos & 32) > 0){
122             printf(" Forward");
123             LCD_Show_Text(" Forward");
124         }
125         if(repos == 64){
126             printf(" Point_camera_at_Table\n");
127             LCD_Show_Text(" Point_At_Table");
128         }
129         printf("\n");
130         last_repos = repos;
131     }
132     continue;
133 }
134 }
135 printf(" Done_Calibration\n");
136 camStart = READ_CAMERA_START;
137 camEnd = READ_CAMERA_END;
138
139 camStartX = (long long)(camStart & 0x0007FF);
140 camStartY = (long long)((camStart & 0x3FF800) >> 11);
141 camEndX = (long long)(camEnd & 0x0007FF);
142 camEndY = (long long)((camEnd & 0x3FF800) >> 11);
143
144 SET_CAMERA_SANDBOX_START_X((long)camStartX + CAMERA_CROP_MARGIN_HOZ);
145 SET_CAMERA_SANDBOX_END_X((long)camEndX - CAMERA_CROP_MARGIN_HOZ);
146 SET_CAMERA_SANDBOX_START_Y((long)camStartY + CAMERA_CROP_MARGIN_VER);
147 SET_CAMERA_SANDBOX_END_Y((long)camEndY - CAMERA_CROP_MARGIN_VER);
148
149 printf(" Cam_Start_X=%ld, End_X=%ld\n", (long)camStartX, (long)camEndX);
150 printf(" Cam_Start_Y=%ld, End_Y=%ld\n", (long)camStartY, (long)camEndY);
151
152 STOP_CALIBRATION;
153 }
154
155 void wait_fn(){
156     int i=0;
157     int j=0;
158     for (; i<=4001;){
159         i++;

```



```

160     }
161     while ((IORD_16DIRECT(VGA_BASE, 12*2) & 0x0001) ==1);
162     for (; j <=500;){
163         j++;
164     }
165 }
166
167 void wait_fn2(){
168     int i=0;
169     for (; i <=150001;){
170         i++;
171     }
172     while ((IORD_16DIRECT(VGA_BASE, 20*2) & 0x0001) ==1);
173 }
174
175 /* void wait_fn2(){
176     int i=0;
177     int j=0;
178     for (; i <=50001;){
179         i++;
180     }
181     while ((IORD_16DIRECT(VGA_BASE, 20*2) & 0x0001) ==1);
182     for (; j <=500;){
183         j++;
184     }
185 }*/
186
187 void wait_fn3(){
188     int i=0;
189     for (; i <=5000001;){
190         i++;
191     }
192 }
193
194 void wait_fn4(){
195     int i=0;
196     for (; i <=4001;){
197         i++;
198     }
199 }
200
201 void wait_fn5(){
202     int i=0;
203     for (; i <=600001;){
204         i++;
205     }
206 }
207
208 int calibrate(){
209     int cross_H = 150;
210     int cross_V = 150;
211     int x1 = 0;
212     int x2 = 320;
213     int y1 = 140;
214     int y2 = 180;
215     int temp_x1=0, temp_x2=0, temp_y1=0, temp_y2=0;
216     int cal_flag=0;
217     int read_in=0;
218     int delta_stick = 4; // Defines the stepsize in the calibration sticks
219     int border_margin = 15;
220
221     int xin, yin;
222
223     while (1) {
224         wait_fn2();
225         //printf("H = %d, V = %d\n", cross_H, cross_V);
226         IOWR_POS(VGA_BASE, 16, x1);
227         IOWR_POS(VGA_BASE, 18, x2);
228         IOWR_POS(VGA_BASE, 17, y1);
229         IOWR_POS(VGA_BASE, 19, y2);
230         IOWR_POS(VGA_BASE, 21, 0);
231         IOWR_POS(VGA_BASE, 13, cross_H);
232         IOWR_POS(VGA_BASE, 14, cross_V);
233         wait_fn4 ();
234         xin = IORD_32DIRECT(VISION_BASE, 0);
235         yin = (xin >> 16) & 0x00007FFF;
236         xin = xin & 0x0000FFFF;
237
238         cross_H = xin;
239         cross_V = yin;

```

```

240     read_in+=1;
242     wait_fn2 ();

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

248     if (NO_CUE_DETECTED && cal_flag==0 && read_in>2)
250     {
252         wait_fn3 ();
254         temp_x1 = x2 + delta_stick;
256         x2=639;
258         x1=320;
260         y1=140;
262         y2=180;
264         cal_flag=1;
266         read_in=0;
268     }
270     if ( cal_flag==1 && read_in >2)
272         x1 = x1 + delta_stick;

274     if (NO_CUE_DETECTED  && cal_flag==1 && read_in >2)
276     {
278         wait_fn3 ();
280         temp_x2 = x1 - delta_stick;
282         x1 = 320;
284         x2 = 380;
286         y1=0;
288         y2=240;
290         cal_flag=2;
292         read_in=0;
294     }
296     if ( cal_flag==2 && read_in >2)
298         y2 = y2 - delta_stick;

300     if (NO_CUE_DETECTED  && cal_flag==2 && read_in >2)
302     {
304         wait_fn3 ();
306         temp_y1 = y2 + delta_stick;
308         x1=120;
310         x2=180;
312         y1=240;
314         y2=480;
316         cal_flag=3;
318         read_in=0;
320     }
322     if ( cal_flag==3 && read_in >2)
324     {
326         y1 = y1 + delta_stick;
328     }

330     if (NO_CUE_DETECTED  && cal_flag==3 && read_in >2)
332     {
334         temp_y2 = y1 - delta_stick;
336         tableStartX = (long long)temp_x1 + border_margin;
338         tableEndX = (long long)temp_x2 - border_margin;
340         tableStartY = (long long)temp_y1 + border_margin;
342         tableEndY = (long long)temp_y2 - border_margin;
344         IOWR_POS(VGA_BASE, 16 ,tableStartX);
346         IOWR_POS(VGA_BASE, 18 ,tableEndX );
348         IOWR_POS(VGA_BASE, 17 , tableStartY);
350         IOWR_POS(VGA_BASE, 19 , tableEndY);
352         IOWR_POS(VGA_BASE, 21 , 1);
354         break;
356     }
358 }
360 }
362 return 0;
364 }

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 * Desc:
11 */
12 #include <io.h>
13 #include <system.h>
14 #include <stdio.h>
15
16 #define IOWR_LED_DATA(base, offset, data)    IOWR_16DIRECT(base, (offset) * 2, data)
17 #define IORD_LED_DATA(base, offset)         IORD_16DIRECT(base, (offset) * 2)
18 #define IOWR_LED_SPEED(base, data)         IOWR_16DIRECT(base + 32, 0, data)
19
20 #define IORD_I2C_TIMER(base, offset)        IORD_32DIRECT(base, (offset) * 4)
21 #define IOWR_I2C_REG(base, offset, data)   IOWR_32DIRECT(base, (offset) * 4, data)
22 #define IORD_I2C_REG(base, offset)         IORD_32DIRECT(base, (offset) * 4)
23
24 #define SCLK_SET                            IOWR_I2C_REG(CAMERA_BASE, 0, 0xFFFFFFFF)
25 #define SCLK_CLR                            IOWR_I2C_REG(CAMERA_BASE, 0, 0)
26 #define SDAT_SET                            IOWR_I2C_REG(CAMERA_BASE, 1, 3)
27 #define SDAT_CLR                            IOWR_I2C_REG(CAMERA_BASE, 1, 2)
28 #define SDAT_TRISTATE                       IOWR_I2C_REG(CAMERA_BASE, 1, 0)
29 #define RD_ACK                              IORD_I2C_REG(CAMERA_BASE, 2)
30 #define CLR_ACK                             IOWR_I2C_REG(CAMERA_BASE, 2, 0)
31
32 #define HALF_CLOCK_DELAY                    i+1;
33 #define ONE_CLOCK_DELAY                    i++; i--;
34 #define DELAY(x)                            for(i=0; i < (x); i++)
35
36 #include "i2c.h"
37
38 int configureCamera()
39 {
40     volatile int i;
41     int ack1, ack2, ack3, ack4, ack5, ack6;
42     int bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7;
43     int bit8, bit9, bit10, bit11, bit12, bit13, bit14, bit15;
44     int version;
45
46     SDAT_SET;
47     SCLK_SET;
48     DELAY(10000);
49
50     COMM_INIT;
51     READ_ACK(ack1);
52     SEND_0;
53     SEND_9;
54     READ_ACK(ack2);
55
56     START;
57     SEND_B;
58     SEND_B;
59     READ_ACK(ack3);
60     READ(bit0);
61     READ(bit1);
62     READ(bit2);
63     READ(bit3);
64     READ(bit4);
65     READ(bit5);
66     READ(bit6);
67     READ(bit7);
68     NACK;
69
70     COMM_INIT;
71     READ_ACK(ack4);
72     SEND_F;
73     SEND_1;
74     READ_ACK(ack5);
75
76     START;
77     SEND_B;
78     SEND_B;
79     READ_ACK(ack6);

```

```

80 READ(bit8);
   READ(bit9);
82 READ(bit10);
   READ(bit11);
84 READ(bit12);
   READ(bit13);
86 READ(bit14);
   READ(bit15);
88 NACK;

90 STOP;

92 printf("Ack_1:_%d\n", ack1);
   printf("Ack_2:_%d\n", ack2);
94 printf("Ack_3:_%d\n", ack3);
   printf("Ack_4:_%d\n", ack4);
96 printf("Ack_5:_%d\n", ack5);
   printf("Ack_6:_%d\n", ack6);
98
100 version = (bit15 << 0) + (bit14 << 1) + (bit13 << 2) + (bit12 << 3) +
            (bit11 << 4) + (bit10 << 5) + (bit9 << 6) + (bit8 << 7) +
102            (bit7 << 8) + (bit6 << 9) + (bit5 << 10) + (bit4 << 11) +
            (bit3 << 12) + (bit2 << 13) + (bit1 << 14) + (bit0 << 15);
104 printf("Version =_0x%x\n", version);

106 /*
   * Write the exposure setting
108 */
110 COMM_INIT; READ_ACK(ack1);
   SEND_0; SEND_9; READ_ACK(ack2);
   SEND_0; SEND_2; READ_ACK(ack3);
112
   COMM_INIT; READ_ACK(ack4);
114 SEND_F; SEND_1; READ_ACK(ack5);
   SEND_2; SEND_A; READ_ACK(ack6);
116 STOP;

118 DELAY(10000);

120 /*
   * Write the row start
122 */
124 COMM_INIT; READ_ACK(ack1);
   SEND_0; SEND_1; READ_ACK(ack2);
   SEND_0; SEND_0; READ_ACK(ack3);
126
   COMM_INIT; READ_ACK(ack4);
128 SEND_F; SEND_1; READ_ACK(ack5);
   SEND_D; SEND_5; READ_ACK(ack6);
130 STOP;

132 DELAY(10000);

134 /*
   * Write the column start
136 */
138 COMM_INIT; READ_ACK(ack1);
   SEND_0; SEND_2; READ_ACK(ack2);
   SEND_0; SEND_1; READ_ACK(ack3);
140
   COMM_INIT; READ_ACK(ack4);
142 SEND_F; SEND_1; READ_ACK(ack5);
   SEND_4; SEND_0; READ_ACK(ack6);
144 STOP;

146 DELAY(10000);

148 /*
   * Write the row width
150 */
152 COMM_INIT; READ_ACK(ack1);
   SEND_0; SEND_3; READ_ACK(ack2);
   SEND_0; SEND_1; READ_ACK(ack3);
154
   COMM_INIT; READ_ACK(ack4);
156 SEND_F; SEND_1; READ_ACK(ack5);
   SEND_E; SEND_0; READ_ACK(ack6);
158 STOP;

```

```

160 DELAY(10000);
162 /*
163  * Write the column width
164  */
165 COMM_INIT;      READ_ACK(ack1);
166 SEND_0; SEND_4; READ_ACK(ack2);
167 SEND_0; SEND_2; READ_ACK(ack3);
168
169 COMM_INIT;      READ_ACK(ack4);
170 SEND_F; SEND_1; READ_ACK(ack5);
171 SEND_8; SEND_0; READ_ACK(ack6);
172 STOP;
173
174 DELAY(10000);
175
176 /*
177  * Write the horizontal blanking for mode B
178  */
179 COMM_INIT;      READ_ACK(ack1);
180 SEND_0; SEND_5; READ_ACK(ack2);
181 SEND_0; SEND_0; READ_ACK(ack3);
182
183 COMM_INIT;      READ_ACK(ack4);
184 SEND_F; SEND_1; READ_ACK(ack5);
185 SEND_C; SEND_A; READ_ACK(ack6);
186 STOP;
187
188 DELAY(10000);
189
190 /*
191  * Write the row speed
192  */
193 COMM_INIT;      READ_ACK(ack1);
194 SEND_0; SEND_A; READ_ACK(ack2);
195 SEND_0; SEND_0; READ_ACK(ack3);
196
197 COMM_INIT;      READ_ACK(ack4);
198 SEND_F; SEND_1; READ_ACK(ack5);
199 SEND_1; SEND_1; READ_ACK(ack6);
200 STOP;
201
202 DELAY(10000);
203
204 /*
205  * Write the vertical blanking for mode B
206  */
207 COMM_INIT;      READ_ACK(ack1);
208 SEND_0; SEND_6; READ_ACK(ack2);
209 SEND_0; SEND_0; READ_ACK(ack3);
210
211 COMM_INIT;      READ_ACK(ack4);
212 SEND_F; SEND_1; READ_ACK(ack5);
213 SEND_1; SEND_9; READ_ACK(ack6);
214 STOP;
215
216 DELAY(10000);
217
218 /*
219  * Write the horizontal blanking for mode A
220  */
221 COMM_INIT;      READ_ACK(ack1);
222 SEND_0; SEND_7; READ_ACK(ack2);
223 SEND_0; SEND_0; READ_ACK(ack3);
224
225 COMM_INIT;      READ_ACK(ack4);
226 SEND_F; SEND_1; READ_ACK(ack5);
227 SEND_8; SEND_8; READ_ACK(ack6);
228 STOP;
229
230 DELAY(10000);
231
232 /*
233  * Write the context control
234  */
235 // COMM_INIT;      READ_ACK(ack1);
236 // SEND_C; SEND_8; READ_ACK(ack2);
237 // SEND_0; SEND_0; READ_ACK(ack3);
238 //
239 // COMM_INIT;      READ_ACK(ack4);

```

```

240 // SEND_F; SEND_1; READ_ACK(ack5);
241 // SEND_0; SEND_B; READ_ACK(ack6);
242 // STOP;

244 DELAY(10000);
    COMM_INIT;
246 READ_ACK(ack1);
    SEND_0;
248 SEND_3;
    READ_ACK(ack2);
250
251 START;
252 SEND_B;
    SEND_B;
254 READ_ACK(ack3);
    READ(bit0);
256 READ(bit1);
    READ(bit2);
258 READ(bit3);
    READ(bit4);
260 READ(bit5);
    READ(bit6);
262 READ(bit7);
    NACK;
264
265 COMM_INIT;
266 READ_ACK(ack4);
    SEND_F;
268 SEND_1;
    READ_ACK(ack5);
270
271 START;
272 SEND_B;
    SEND_B;
274 READ_ACK(ack6);
    READ(bit8);
276 READ(bit9);
    READ(bit10);
278 READ(bit11);
    READ(bit12);
280 READ(bit13);
    READ(bit14);
282 READ(bit15);
    NACK;
284
285 STOP;
286
287 printf("Ack_1:_%d\n", ack1);
288 printf("Ack_2:_%d\n", ack2);
289 printf("Ack_3:_%d\n", ack3);
290 printf("Ack_4:_%d\n", ack4);
291 printf("Ack_5:_%d\n", ack5);
292 printf("Ack_6:_%d\n", ack6);

293
294 version = (bit15 << 0) + (bit14 << 1) + (bit13 << 2) + (bit12 << 3) +
            (bit11 << 4) + (bit10 << 5) + (bit9 << 6) + (bit8 << 7) +
296            (bit7 << 8) + (bit6 << 9) + (bit5 << 10) + (bit4 << 11) +
            (bit3 << 12) + (bit2 << 13) + (bit1 << 14) + (bit0 << 15);
298
299 printf("Row_Width=_%d\n", version);
300 return 0;
}

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 * Desc:
11 */
12 #include <stdio.h>
13 #include "fixedpoint.h"
14 #include "gameconfig.h"
15 #include "types.h"
16 #include "debug.h"
17 #include "ball.h"
18
19 #define VEC_MAG_SQ(vec)          FPSUM(FPSQR(vec.x),FPSQR(vec.y))
20 #define VECDOT_PROD(vec1,vec2) FPSUM(          \
21     FPMUL(vec1.x,vec2.x),FPMUL(vec1.y,vec2.y)  \
22 )
23
24 long long ballRegisters[NUM_BALLS] = {
25     BALL_0_BASE,
26     BALL_1_BASE,
27     BALL_2_BASE,
28     BALL_3_BASE,
29     BALL_4_BASE,
30     BALL_5_BASE,
31     BALL_6_BASE
32 };
33
34 #define DRAW_BALL(ballIndex,x,y,col)          \
35     IOWR_VAL(ballRegisters[ballIndex] + BALL_X,(x)), \
36     IOWR_VAL(ballRegisters[ballIndex] + BALL_Y,(y)), \
37     IOWR_VAL(ballRegisters[ballIndex] + BALL_COLOUR,(col))
38
39 #define VGA_NOT_READY (IORD_16DIRECT(VGA_BASE, VGA_FLAG * 2) & 0x0001)
40
41 /*
42 // Ball dynamics implementation
43 */
44 bool_t isBallMoving(const struct ball_t *ball){
45     return (((ball->vel.x != 0) || (ball->vel.y != 0))? TRUE : FALSE);
46 }
47
48 long long collisionWithTableTime(const struct ball_t *ball, edge_t *edge){
49     long long hColTime, vColTime;
50     edge_t hozEdge, verEdge;
51     hColTime = INT2FP(1000);
52     vColTime = INT2FP(1000);
53
54     DP_ASSERT(
55         ((ball->vel.x != 0) || (ball->vel.y != 0)),
56         "Collision check being performed for stationary ball"
57     );
58
59     if(ball->vel.x > 0){
60         hColTime = FPDIV((TABLE_END_X -
61             (ball->pos.x + ball->radius)), ball->vel.x);
62         hozEdge = RIGHT;
63     } else if(ball->vel.x < 0){
64         hColTime = FPDIV((TABLE_START_X -
65             (ball->pos.x - ball->radius)), ball->vel.x);
66         hozEdge = LEFT;
67     } else{
68         /*
69         // Ball is not moving along long this axis => Nothing to do
70         */
71     }
72     if(hColTime < 0) hColTime = INT2FP(1000);
73
74     if(ball->vel.y > 0){
75         vColTime = FPDIV((TABLE_END_Y -
76             (ball->pos.y + ball->radius)), ball->vel.y);
77         verEdge = BOTTOM;
78     } else if(ball->vel.y < 0){
79         vColTime = FPDIV((TABLE_START_Y -
80             (ball->pos.y - ball->radius)), ball->vel.y);

```

```

80     verEdge = TOP;
81 }else{
82     /*
83     // Ball is not moving along long this axis => Nothing to do
84     */
85 }
86 if(vColTime < 0) vColTime = INT2FP(1000);
87
88 if(hColTime > vColTime){
89     *edge = verEdge;
90     return vColTime;
91 }else{
92     *edge = hozEdge;
93     return hColTime;
94 }
95 }
96
97 long long collisionWithBallTime (
98     const struct ball_t *ball1 ,
99     const struct ball_t *ball2
100 ){
101     long long contactDist = ball1->radius + ball2->radius;
102     struct vector relativeVelocity;
103     struct vector relativeDisplacement;
104     long long result;
105
106     relativeVelocity.x = ball1->vel.x - ball2->vel.x;
107     relativeVelocity.y = ball1->vel.y - ball2->vel.y;
108
109     relativeDisplacement.x = ball1->pos.x - ball2->pos.x;
110     relativeDisplacement.y = ball1->pos.y - ball2->pos.y;
111
112     long long A = VEC_MAG_SQ(relativeVelocity);
113     long long B = 2 * VEC_DOT_PROD(relativeDisplacement, relativeVelocity);
114     long long C = VEC_MAG_SQ(relativeDisplacement) - FPSQR(contactDist);
115
116     long long BSQ_MINUS_4AC = FPSQR(B) - 4LL * FPMUL(A,C);
117
118     if((BSQ_MINUS_4AC < 0) || (A == 0)){
119         result = INT2FP(1000LL);
120     }else{
121         result = FPDIV((-B - FPSQRT(BSQ_MINUS_4AC)),(2LL * A));
122     }
123
124     return result;
125 }
126
127 void handleBallCollision(struct ball_t *ball1, struct ball_t *ball2){
128     const long long A = FPDIV(INT2FP(22LL),INT2FP(30LL));
129     const long long B = FPDIV(INT2FP(22LL),INT2FP(30LL));
130     const long long C = FPDIV(INT2FP(22LL),INT2FP(30LL));
131     const long long D = FPDIV(INT2FP(22LL),INT2FP(30LL));
132
133     struct vector unitRelativeDisp1To2;
134     struct vector tempUnitRelativeDisp1To2;
135
136     unitRelativeDisp1To2.x = ball2->pos.x - ball1->pos.x;
137     unitRelativeDisp1To2.y = ball2->pos.y - ball1->pos.y;
138
139     long long relativeDispMag = FPSQRT(VEC_MAG_SQ(unitRelativeDisp1To2));
140     unitRelativeDisp1To2.x =
141         FPDIV(unitRelativeDisp1To2.x, relativeDispMag);
142     unitRelativeDisp1To2.y =
143         FPDIV(unitRelativeDisp1To2.y, relativeDispMag);
144
145     tempUnitRelativeDisp1To2.x = unitRelativeDisp1To2.x;
146     tempUnitRelativeDisp1To2.y = unitRelativeDisp1To2.y;
147
148     long long ball1VelocityComp =
149         VEC_DOT_PROD(ball1->vel, unitRelativeDisp1To2);
150     long long ball2VelocityComp =
151         VEC_DOT_PROD(ball2->vel, unitRelativeDisp1To2);
152
153     long long newVelocityCompMagBall1 =
154         FPMUL(A, ball1VelocityComp) - FPMUL(B, ball2VelocityComp);
155     long long newVelocityCompMagBall2 =
156         FPMUL(C, ball2VelocityComp) - FPMUL(D, ball1VelocityComp);
157
158     unitRelativeDisp1To2.x = FPMUL(
159         (unitRelativeDisp1To2.x), newVelocityCompMagBall1

```



```

160 );
161 unitRelativeDisp1To2.y = FPMUL(
162     (unitRelativeDisp1To2.y), newVelocityCompMagBall1
163 );
164 tempUnitRelativeDisp1To2.x = FPMUL(
165     (tempUnitRelativeDisp1To2.x), newVelocityCompMagBall2
166 );
167 tempUnitRelativeDisp1To2.y = FPMUL(
168     (tempUnitRelativeDisp1To2.y), newVelocityCompMagBall2
169 );
170 );
171 ball1->vel.x -= unitRelativeDisp1To2.x;
172 ball1->vel.y -= unitRelativeDisp1To2.y;
173 ball2->vel.x -= tempUnitRelativeDisp1To2.x;
174 ball2->vel.y -= tempUnitRelativeDisp1To2.y;
175 }
176 }
177
178 long long handleCollisionWithCue (
179     struct ball_t *ball1 ,
180     const struct ball_t *cue
181 ){
182     const long long A = FPDIV(INT2FP(8LL),INT2FP(50LL));
183     const long long B = FPDIV(INT2FP(28LL),INT2FP(50LL));
184
185     struct vector unitRelativeDisp1To2;
186     struct vector tempUnitRelativeDisp1To2;
187
188     unitRelativeDisp1To2.x = cue->pos.x - ball1->pos.x;
189     unitRelativeDisp1To2.y = cue->pos.y - ball1->pos.y;
190
191     long long relativeDispMag = FPSQRT(VEC_MAG_SQ(unitRelativeDisp1To2));
192     unitRelativeDisp1To2.x =
193         FPDIV(unitRelativeDisp1To2.x, relativeDispMag);
194     unitRelativeDisp1To2.y =
195         FPDIV(unitRelativeDisp1To2.y, relativeDispMag);
196
197     tempUnitRelativeDisp1To2.x = unitRelativeDisp1To2.x;
198     tempUnitRelativeDisp1To2.y = unitRelativeDisp1To2.y;
199
200     long long ball1VelocityComp =
201         VEC_DOT_PROD(ball1->vel, unitRelativeDisp1To2);
202     long long cueVelocityComp =
203         VEC_DOT_PROD(cue->vel, unitRelativeDisp1To2);
204
205     long long newVelocityCompMagBall1 =
206         FPMUL(A, ball1VelocityComp) - FPMUL(B, cueVelocityComp);
207
208     unitRelativeDisp1To2.x = FPMUL(
209         (unitRelativeDisp1To2.x), newVelocityCompMagBall1
210     );
211     unitRelativeDisp1To2.y = FPMUL(
212         (unitRelativeDisp1To2.y), newVelocityCompMagBall1
213     );
214
215     ball1->vel.x -= unitRelativeDisp1To2.x;
216     ball1->vel.y -= unitRelativeDisp1To2.y;
217
218     return 0;
219 }
220
221 void moveBalls(struct ball_t *balls, long long time){
222     int i;
223
224     for(i=0; i<NUMBALLS; i++){
225         if((BALL_INVISIBLE != balls[i].ballState) &&
226             (TRUE == isBallMoving(&balls[i])))
227         ){
228             balls[i].pos.x += FPMUL(balls[i].vel.x, time);
229             balls[i].pos.y += FPMUL(balls[i].vel.y, time);
230         }
231     }
232 }
233
234 void drawBalls(struct ball_t *balls){
235     int i;
236
237     while(VGA_NOT_READY == 1);
238
239     for(i=0; i<NUMBALLS; i++){

```

```

240     if(BALL_INVISIBLE == balls[i].ballState){
241         balls[i].colour = COL_INVISIBLE;
242     }
243     /*
244     // Tell the hardware to draw the balls on screen
245     */
246     //DP2("Drawing Ball at (%lld, %lld)\n",SPRITE_X(balls[i]),SPRITE_Y(balls[i]));
247     DRAW_BALL(i,SPRITE_X(balls[i]),SPRITE_Y(balls[i]),balls[i].colour);
248 }
249 }
250
251 void applyFriction(struct ball_t *balls){
252     int i;
253     long long newVelX, newVelY;
254
255     for(i=0; i<NUMBALLS; i++){
256         if(BALL_INVISIBLE == balls[i].ballState){
257             continue;
258         }
259
260         newVelX = balls[i].vel.x - FPMUL(balls[i].vel.x,DAMPING_COEFF);
261         if(FPMUL(newVelX, balls[i].vel.x) > 0){
262             balls[i].vel.x = newVelX;
263         }
264         else{
265             balls[i].vel.x = 0;
266         }
267
268         newVelY = balls[i].vel.y - FPMUL(balls[i].vel.y,DAMPING_COEFF);
269         if(FPMUL(newVelY, balls[i].vel.y) > 0){
270             balls[i].vel.y = newVelY;
271         }
272         else{
273             balls[i].vel.y = 0;
274         }
275     }
276 }

```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 *
11 * Desc:
12 */
13 #include "fixedpoint.h"
14 #include "debug.h"
15
16 /*
17 // Thanks to GameProgrammer.com
18 */
19 #define step(shift) \
20     if((0x40000000l >> shift) + root <= num) \
21     { \
22         num -= (0x40000000l >> shift) + root; \
23         root = (root >> 1) | (0x40000000l >> shift); \
24     } \
25     else \
26     { \
27         root = root >> 1; \
28     }
29
30 long long FPSQRT(long long num){
31     long long root = 0;
32
33     step( 0);
34     step( 2);
35     step( 4);
36     step( 6);
37     step( 8);
38     step(10);
39     step(12);
40     step(14);
41     step(16);
42     step(18);
43     step(20);
44     step(22);
45     step(24);
46     step(26);
47     step(28);
48     step(30);
49
50     // round to the nearest integer, cuts max error in half
51
52     if (root < num)
53     {
54         ++root;
55     }
56
57     root <<= 6;
58
59     return root;
60 }
61 /*
62 long long FPSQRT(long long num){
63     long long next, root;
64
65     if (num < INT2FP(1LL)){
66         root = 0;
67     } else {
68         next = num >> 2;
69         do {
70             root = next;
71             next = (next + FPDIV(num, next)) >> 1;
72         } while (root != next);
73     }
74
75     return root;
76 }*/
77
78 void printFP(long long fpnum){
79     int i;
80     float factor = 0.5;

```

```
80 | float result = 0;
81 | for(i=FRAC_PRECISION - 1; i >= 0; i--){
82 |     if(fpnum & (0x1 << i)){
83 |         result += factor;
84 |     }
85 |     factor *= 0.5;
86 | }
87 | printf("%.4f\n", result + FP2INT(fpnum));
88 | }
```

```

2  /*
3  * Software for Interactive Project Pool Game
4  * Columbia University. New York, 2008
5  * Authors:
6  *   Abdulhamid Ghandour
7  *   Thomas John
8  *   Jaime Peretzman
9  *   Bharadwaj Vellore
10 *
11 * Desc:
12 */
13 #include <unistd.h>
14 #include <string.h>
15 #include "io.h"
16 #include "system.h"
17 #include "ui.h"
18 unsigned long sevensegment [] = { /* Active Low -> xgfedcba */
19     0x40, /* 0 -> 01000000 */
20     0x79, /* 1 -> 01111001 */
21     0x24, /* 2 -> 00100100 */
22     0x30, /* 3 -> 00110000 */
23     0x19, /* 4 -> 00011001 */
24     0x12, /* 5 -> 00010010 */
25     0x02, /* 6 -> 00000010 */
26     0xF8, /* 7 -> 01111000 */
27     0x00, /* 8 -> 00000000 */
28     0x10, /* 9 -> 00010000 */
29     0x0C, /* P -> 00001100 */
30 };
31
32 #define P 10
33
34 void initPointsDisplay () {
35     HEX7(sevensegment [P]);
36     HEX6(sevensegment [1]);
37     HEX3(sevensegment [P]);
38     HEX2(sevensegment [2]);
39
40     HEX5(sevensegment [0]);
41     HEX4(sevensegment [0]);
42     HEX1(sevensegment [0]);
43     HEX0(sevensegment [0]);
44 }
45
46 void LCD_Init ()
47 {
48     lcd_write_cmd (LCD_BASE, 0x38);
49     usleep (2000);
50     lcd_write_cmd (LCD_BASE, 0x0C);
51     usleep (2000);
52     lcd_write_cmd (LCD_BASE, 0x01);
53     usleep (2000);
54     lcd_write_cmd (LCD_BASE, 0x06);
55     usleep (2000);
56     lcd_write_cmd (LCD_BASE, 0x80);
57     usleep (2000);
58 }
59
60 void LCD_Show_Text (char * Text)
61 {
62     int i;
63     for (i=0; i<strlen (Text); i++) {
64         lcd_write_data (LCD_BASE, Text [i]);
65         usleep (2000);
66     }
67 }
68
69 void LCD_Line2 ()
70 {
71     lcd_write_cmd (LCD_BASE, 0xC0);
72     usleep (2000);
73 }

```

```

--
2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
4  --     Abdulhamid Ghandour
--     Thomas John
6  --     Jaime Peretzman
--     Bharadwaj Vellore
8  --
-- Desc:
10 --
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
16 entity i2c_controller is
18     port (
19         clk          : in  std_logic;
20         reset_n     : in  std_logic;
21         read        : in  std_logic;
22         write       : in  std_logic;
23         chipselect  : in  std_logic;
24         address     : in  unsigned(31 downto 0);
25         readdata    : out unsigned(31 downto 0);
26         writedata   : in  unsigned(31 downto 0);
27         sclk        : out std_logic;
28         sdat        : inout std_logic;
29         ack         : in  std_logic
30     );
31 end i2c_controller;
32
33 architecture rtl of i2c_controller is
34     type ram_type is array(7 downto 0) of unsigned(31 downto 0);
35     signal RAM : ram_type;
36     signal ram_address : unsigned(2 downto 0);
37     signal counter : unsigned(31 downto 0);
38     signal int_sclk : std_logic := '1';
39     signal int_sdat : std_logic := '1';
40     signal int_ack : std_logic := '0';
41 begin
42     ram_address <= address(2 downto 0);
43
44     i2c_host_control: process (clk)
45     begin
46         if rising_edge(clk) then
47             if reset_n = '0' then
48
49             else
50                 if chipselect = '1' then
51                     if read = '1' then
52                         if to_integer(ram_address) = 2 then
53                             readdata(0) <= ack;
54                         else
55                             readdata <= RAM(to_integer(ram_address));
56                         end if;
57                     elsif write = '1' then
58                         RAM(to_integer(ram_address)) <= writedata;
59                     end if;
60                 end if;
61             end if;
62             RAM(7) <= counter;
63         end if;
64     end process i2c_host_control;
65
66     timer: process (clk)
67     begin
68         if rising_edge(clk) then
69             if reset_n = '0' then
70                 counter <= (others => '0');
71             else
72                 counter <= counter + 1;
73             end if;
74         end if;
75     end process timer;
76
77     i2c_line_control: process (clk)
78     begin

```

```
80     if rising_edge(clk) then
81         if reset_n = '0' then
82             else
83                 int_sclk <= RAM(0)(0);
84             end if;
85         end if;
86     end process i2c_line_control;
87
88     sdat <= RAM(1)(0) when RAM(1)(1) = '1' else 'Z';
89     sclk <= int_sclk;
90 end rtl;
```

```

library ieee;
2 use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
4
entity calibration is
6
    port (
8         reset      : in std_logic;
          clk        : in std_logic;
10        valid_green : in std_logic;
          end_row    : in std_logic;
12        endframe   : in std_logic;
          green_pixel_value : in unsigned (9 downto 0);
14        green_column_thr : in unsigned (9 downto 0);
          green_row_thr  : in unsigned (9 downto 0);
16        repos      : out unsigned (6 downto 0) := "1000000";
          x_1         : out unsigned (10 downto 0) := "000000000000";
18        y_1         : out unsigned (10 downto 0) := "000000000000";
          x_2         : out unsigned (10 downto 0) := "000000000000";
20        y_2         : out unsigned (10 downto 0) := "000000000000";
          calibration_on : in std_logic;
22        threshold  : in unsigned (9 downto 0);
          leds       : out unsigned (6 downto 0)
24    );
26 end calibration;
28 architecture rtl of calibration is
30     -- signals for calibration
          signal row_counter : unsigned (10 downto 0) := "000000000000";
32     signal green_column_count : unsigned (10 downto 0) := "000000000000";
          signal green_row_count : unsigned (10 downto 0) := "000000000000";
34     signal green_column_thr : unsigned (9 downto 0) := "0001111101";
          signal green_row_thr  : unsigned (9 downto 0) := "0110010000";
36     signal green_color_thr : unsigned (9 downto 0);
          signal green_xl_thr   : unsigned (9 downto 0) := "0001101000";
38     signal black_column_thr : unsigned (2 downto 0) := "011";
          signal black_column_count : unsigned (2 downto 0) := "000";
40     signal temp_active_row : unsigned (2 downto 0) := "000";
          signal temp_changes_sig : unsigned (1 downto 0) := "00";
42     signal black_first_count : unsigned (4 downto 0) := "00000";
          signal first_row_black : std_logic := '0';
44     signal temp_left_column : std_logic := '0';
          signal temp_green_row : std_logic := '0';
46     signal temp_right_column : std_logic := '0';
          signal flag : std_logic := '0';
48     signal flag2 : std_logic := '0';
          signal start_flag : std_logic := '0';
50     signal temp_x_1 : unsigned (10 downto 0) := "000000000000";
          signal temp_y_1 : unsigned (10 downto 0) := "000000000000";
52     signal temp_x_2 : unsigned (10 downto 0) := "011111111111";
          signal temp_y_2 : unsigned (10 downto 0) := "000000000000";
54     signal column_counter : unsigned (10 downto 0) := "000000000000";
          signal active_row : unsigned (2 downto 0) := "000";
56     signal changes_sig : unsigned (1 downto 0) := "00";
          signal cam_repos : unsigned (6 downto 0) := "1000000";
58     signal debug_x_counter : unsigned (10 downto 0) := "000000000000";
          signal debug_x_max : unsigned (10 downto 0) := "000000000000";
60
begin
62     green_color_thr <= threshold;
64
    Calib : process (clk)
        begin
66         if rising_edge(clk) then
            if reset = '1' then
68                 row_counter <= (others => '0');
                    green_column_count <= (others => '0');
70                 green_row_count <= (others => '0');
                    black_column_count <= (others => '0');
72                 temp_active_row <= (others => '0');
                    temp_changes_sig <= (others => '0');
74                 black_first_count <= (others => '0');
                    temp_green_row <= '0';
76                 temp_left_column <= '0';
                    temp_right_column <= '0';
78                 flag <= '0';
                    flag2 <= '0';
            end if;
        end if;
    end process Calib;

```



```

80     first_row_black    <= '0';
      start_flag        <= '0';
82     x_1                <= (others => '0');
      x_2                <= (others => '0');
84     y_1                <= (others => '0');
      y_2                <= (others => '0');
86     temp_x_1          <= (others => '0');
      temp_y_1          <= (others => '0');
88     temp_x_2          <= "011111111111";
      temp_y_2          <= (others => '0');
90     column_counter    <= (others => '0');
      active_row        <= (others => '0');
92     changes_sig       <= (others => '0');
      cam_repos         <= "1000000";
94
      elsif valid_green = '1' and calibration_on = '1' then
96         -- if start_flag = '1' then
          --cam_repos <= (others => '1');
98         if green_pixel_value > green_color_thr then
100            green_column_count <= green_column_count + 1;
            black_column_count <= (others => '0');
      else
102         black_column_count <= black_column_count + 1;
      end if;
104
      column_counter <= column_counter + 1;
106     debug_x_counter <= debug_x_counter + 1; -- *****DEBUG*****

108     if black_column_count >= black_column_thr then
110         if green_column_count >= green_column_thr then
            temp_right_column <= '1';
112         elsif green_column_count < green_column_thr then
            temp_left_column <= '1';
            green_column_count <= (others => '0');
114         end if;
      end if;
116
118     if green_column_count >= green_column_thr then
        temp_green_row <= '1';
        if green_row_count >= green_row_thr and flag = '0' then
120            if temp_x_1 < column_counter - green_column_thr and flag2 = '0' then
                temp_x_1 <= column_counter - green_column_thr;
122                flag2 <= '1';
            end if;
124            if black_column_count >= black_column_thr then
                temp_x_2 <= column_counter - black_column_thr;
126                flag <= '1';
            end if;
128            -- if black_column_count >= black_column_thr then
                -- if temp_x_2 < column_counter - black_column_thr and flag = '0' then
130                -- temp_x_2 <= column_counter - black_column_thr;
                -- flag <= '1';
132            -- end if;
            -- end if;
134        end if;
      end if;
136
138     if black_first_count > 1 then
        first_row_black <= '1';
        black_first_count <= (others => '0');
140     end if;
      end if; -- end of valid green
142
144     if end_row = '1' then
        if temp_green_row = '1' then
            green_row_count <= green_row_count + 1;
146        end if;
148 --*****DEBUG*****
      debug_x_counter <= (others => '0');
150
152     if debug_x_counter > debug_x_max then
        debug_x_max <= debug_x_counter;
154     end if;
      --*****DEBUG END*****
156     if row_counter < 20 and temp_green_row = '0' then
        black_first_count <= black_first_count + 1;
158     end if;

```

```

160     if green_row_count >= green_row_thr and green_row_count < green_row_thr+2 then
161         if first_row_black = '1' then
162             temp_y_1 <= row_counter - green_row_thr + 1;
163             temp_changes_sig(0) <= '1';
164         end if;
165         temp_active_row(2) <= temp_left_column;
166         temp_active_row(1) <= temp_green_row ;
167         temp_active_row(0) <= temp_right_column;
168     end if;

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

176     column_counter <= (others => '0');
177     --flag <= '0';
178     temp_green_row <= '0';
179     temp_left_column <= '0';
180     temp_right_column <= '0';
181     green_column_count <= (others => '0');
182     black_column_count <= (others => '0');
183     row_counter <= row_counter + 1;
184     end if;
-- end if; -- end of start flag

186
187     if end_frame = '1' then
188         debug_x_max <= (others => '0'); --*****DEBUG*****
189         debug_x_counter <= (others => '0');--*****DEBUG*****
190         if temp_changes_sig = "00" then
191             if temp_active_row = "010" then
192                 cam_repos <= "1010000";
193             elsif temp_active_row = "011" then
194                 cam_repos <= "1010010";
195             elsif temp_active_row = "110" then
196                 cam_repos <= "1010001";
197             elsif temp_active_row = "111" then
198                 cam_repos <= "1010000";
199             else
200                 cam_repos <= "1000000";
201             end if;
202         elsif temp_changes_sig = "01" then
203             if temp_active_row = "010" then
204                 cam_repos <= "1010100";
205             elsif temp_active_row = "011" then
206                 cam_repos <= "1000110";
207             elsif temp_active_row = "110" then
208                 cam_repos <= "1000101";
209             elsif temp_active_row = "111" then
210                 cam_repos <= "1000100";
211             else
212                 cam_repos <= "1000000";
213             end if;
214         elsif temp_changes_sig = "10" then
215             if temp_active_row = "010" then
216                 cam_repos <= "1011000";
217             elsif temp_active_row = "011" then
218                 cam_repos <= "1001010";
219             elsif temp_active_row = "110" then
220                 cam_repos <= "1001001";
221             elsif temp_active_row = "111" then
222                 cam_repos <= "1001000";
223             else
224                 cam_repos <= "1000000";
225             end if;
226         elsif temp_changes_sig = "11" then
227             if temp_active_row = "010" then
228                 cam_repos <= "1010000";
229             elsif temp_active_row = "011" then
230                 cam_repos <= "1000010";
231             elsif temp_active_row = "110" then
232                 cam_repos <= "1000001";
233             elsif temp_active_row = "111" then
234                 cam_repos <= "0000000";
235             else
236                 cam_repos <= "1000000";
237             end if;
238     end if;

```

```

240     --cam_repos <= "1111111";
active_row      <= temp_active_row;
242 changes_sig  <= temp_changes_sig;
x_1             <= temp_x_1;
244 y_1          <= temp_y_1;
x_2             <= temp_x_2;
246 y_2          <= temp_y_2;
temp_x_1        <= (others => '0');
temp_y_1        <= (others => '0');
temp_x_2        <= "01111111111";
250 temp_y_2     <= (others => '0');
temp_changes_sig <= (others => '0');
252 row_counter <= (others => '0');
column_counter <= (others => '0');
254 black_first_count <= (others => '0');
temp_green_row <= '0';
256 temp_left_column <= '0';
temp_right_column <= '0';
258 flag        <= '0';
flag2         <= '0';
260 first_row_black <= '0';
temp_active_row <= (others => '0');
262 green_column_count <= (others => '0');
green_row_count <= (others => '0');
264 black_column_count <= (others => '0');
start_flag     <= '1';
266 end if;
if calibration_on = '0' then
268 cam_repos <= "1000000";
x_1 <= (others => '0');
270 x_2 <= (others => '0');
y_1 <= (others => '0');
272 y_2 <= (others => '0');
end if;
274 -- if chipselect = '1' then
--     if write = '1' then
276 --         if address = "000" then
--             calibration_on <= writedata(0);
278 --             cam_repos <= "1000000";
--         end if;
280 --     end if;
--
282 --     if read = '1' then
--         if address = "001" then
284 --             readdata(6 downto 0) <= cam_repos;
--             elsif address = "010" then
286 --                 readdata(9 downto 0) <= x_1;
--             elsif address = "011" then
288 --                 readdata(9 downto 0) <= y_1;
--             elsif address = "100" then
290 --                 readdata(9 downto 0) <= x_2;
--             elsif address = "101" then
292 --                 readdata(9 downto 0) <= y_2;
--             end if;
294 --         end if;
--     end if;-- end of chipselect
296 leds <= cam_repos;
repos <= cam_repos;
298 end if;
end process Calib;
300 end rtl;

```

```

--
2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
4  --     Abdulhamid Ghandour
--     Thomas John
6  --     Jaime Peretzman
--     Bharadwaj Vellore
8  --
-- Desc:
10 --
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
16 entity ci_pxl is
17     port(
18         clk          : in std_logic;
19         mclk         : out std_logic; -- Master CLK to Camera
20         lval        : in std_logic;  -- Line Valid from Camera
21         fval        : in std_logic;  -- Frame Valid from Camera
22         pixclk      : in std_logic;  -- Pixel CLK from Camera
23         datain      : in unsigned(9 downto 0); -- Pixel Data from Camera
24         dataout     : out unsigned(9 downto 0);
25         valid_green : out std_logic;
26         end_of_frame : out std_logic;
27         end_of_row   : out std_logic;
28         sandboxStartX : in unsigned(31 downto 0);
29         sandboxStartY : in unsigned(31 downto 0);
30         sandboxEndX   : in unsigned(31 downto 0);
31         sandboxEndY   : in unsigned(31 downto 0);
32     );
33 end ci_pxl;
34
35 architecture pool of ci_pxl is
36     signal int_mclk : std_logic := '0';
37     signal last_line_valid : std_logic := '0';
38     signal last_frame_valid : std_logic := '0';
39     signal pixel_counter : std_logic := '0';
40     signal line_counter : std_logic := '0';
41     signal last_pixclk : std_logic := '0';
42 begin
43     mclkgen : process(clk)
44     begin
45         if rising_edge(clk) then
46             int_mclk <= not int_mclk;
47         end if;
48     end process;
49
50     eor_gen : process(clk)
51     begin
52         if rising_edge(clk) then
53             if (last_line_valid = '1' and lval = '0') then
54                 end_of_row <= '1';
55                 line_counter <= not line_counter;
56             else
57                 end_of_row <= '0';
58             end if;
59             if fval = '0' then
60                 line_counter <= '0';
61             end if;
62             last_line_valid <= lval;
63         end if;
64     end process eor_gen;
65
66     eof_gen : process(clk)
67     begin
68         if rising_edge(clk) then
69             if (last_frame_valid = '1' and fval = '0') then
70                 end_of_frame <= '1';
71             else
72                 end_of_frame <= '0';
73             end if;
74             last_frame_valid <= fval;
75         end if;
76     end process eof_gen;
77
78     vg_gen : process(clk)
79     begin

```

```

80     if rising_edge(clk) then
81         if (pixclk = '1' and last_pixclk = '0') then
82             if lval = '1' then
83                 pixel_counter <= not pixel_counter;
84                 valid_green <= not (pixel_counter xor line_counter);
85                 dataout <= datain;
86             end if;
87         else
88             valid_green <= '0';
89         end if;
90         if lval = '0' then
91             pixel_counter <= '0';
92         end if;
93         last_pixclk <= pixclk;
94     end if;
95 end process vg_gen;
96
97 mclk <= int_mclk;
98 end pool;

```

```

library ieee;
2 use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
4
entity imagecropper is
6   port (
      clk           : in std_logic;
8     valid_green_in : in std_logic;
      valid_green_out : out std_logic;
10    end_row_in    : in std_logic;
      end_row_out    : out std_logic;
12    end_frame     : in std_logic;
      crop_start_x   : in unsigned (10 downto 0);
14    crop_end_x    : in unsigned (10 downto 0);
      crop_start_y   : in unsigned (10 downto 0);
16    crop_end_y    : in unsigned (10 downto 0)
    );
18 end imagecropper;

20 architecture rtl of imagecropper is
      signal xcount: unsigned (10 downto 0) := (others => '0');
22     signal ycount: unsigned (10 downto 0) := (others => '0');
      signal crop_start_x_sig: unsigned (10 downto 0) := (others => '0');
24     signal crop_end_x_sig  : unsigned (10 downto 0) := (others => '1');
      signal crop_start_y_sig: unsigned (10 downto 0) := (others => '0');
26     signal crop_end_y_sig  : unsigned (10 downto 0) := (others => '1');
begin
28
      control : process(clk)
30     begin
          if rising_edge(clk) then
32             if end_frame = '1' then
                  crop_start_x_sig <= crop_start_x;
34                 crop_end_x_sig <= crop_end_x;
                  crop_start_y_sig <= crop_start_y;
36                 crop_end_y_sig <= crop_end_y;
            end if;
          end if;
38     end process control;

40
      xcounter : process (clk)
42     begin
          if rising_edge (clk) then
44             if end_row_in = '1' then
                  xcount <= (others => '0');
46             else
                  if valid_green_in = '1' then
48                     xcount <= xcount + 1;
                    end if;
          end if;
50     end if;
          end process xcounter;
52

      ycounter : process (clk)
54     begin
          if rising_edge (clk) then
56             if end_frame = '1' then
                  ycount <= (others => '0');
58             else
                  if end_row_in = '1' then
60                     ycount <= ycount + 1;
                    end if;
          end if;
62     end if;
          end process ycounter;
64

      valid_green_out <= valid_green_in when
68     (((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';
70     end_row_out <= end_row_in when
      ((ycount >= crop_start_y_sig) and (ycount <= crop_end_y_sig)) else '0';
72
end rtl;

```

```

--
2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
4  --     Abdulhamid Ghandour
--     Thomas John
6  --     Jaime Peretzman
--     Bharadwaj Vellore
8  --
-- Desc:
10 --
11 library ieee;
12 use ieee.std_logic_1164.all;
13 use ieee.numeric_std.all;
14
15 entity visionsystem is
16     port(
17         clk : in std_logic;
18
19         pixel_data : in unsigned (9 downto 0);
20         valid_green : in std_logic;
21
22         endofrow : in std_logic;
23         endofframe : in std_logic;
24
25         threshold : in unsigned (9 downto 0);
26
27         xout : out unsigned (15 downto 0);
28         yout : out unsigned (15 downto 0);
29
30         no_detect : out std_logic;
31
32         led0, led1, led2, led3, led4, led5, led6, led7 : out std_logic
33     );
34
35 end visionsystem;
36
37
38 architecture rtl of visionsystem is
39
40     constant FINGER_WIDTH : unsigned (15 downto 0) := x"002A";
41     constant WIDTH_SAMPLE_INTERVAL : unsigned (15 downto 0) := x"0014";
42     constant BOUNDARY_TOLERANCE : unsigned (15 downto 0) := x"0003";
43     constant SAME_EDGE_TOLERANCE : unsigned (15 downto 0) := x"000A";
44
45     signal xcount : unsigned (15 downto 0);
46     signal ycount : unsigned (15 downto 0);
47
48     -----EXTREMITIES-----
49     signal topx : unsigned (15 downto 0) := (others => '0');
50     signal topy : unsigned (15 downto 0) := (others => '1');
51     signal bottomx : unsigned (15 downto 0) := (others => '0');
52     signal bottomy : unsigned (15 downto 0) := (others => '0');
53     signal leftx : unsigned (15 downto 0) := (others => '1');
54     signal lefty : unsigned (15 downto 0) := (others => '0');
55     signal rightx : unsigned (15 downto 0) := (others => '0');
56     signal righty : unsigned (15 downto 0) := (others => '0');
57
58     -----WIDTH-----
59     signal topwidth_start : unsigned (15 downto 0) := (others => '1');
60     signal topwidth_end : unsigned (15 downto 0) := (others => '0');
61
62     signal prev_bottomy : unsigned (15 downto 0) := (others => '0');
63     signal bottomwidth_start : unsigned (15 downto 0) := (others => '0');
64     signal bottomwidth_end : unsigned (15 downto 0) := (others => '0');
65
66     type unsignedarray_type is array(0 to 19) of unsigned (15 downto 0);
67     signal bottomwidth_start_array : unsignedarray_type := (others => x"FFFF");
68     signal bottomwidth_end_array : unsignedarray_type := (others => x"0000");
69     signal bottomwidth_recorded : std_logic := '0';
70
71     -----SCREEN-----
72     signal maxx : unsigned (15 downto 0) := (others => '0');
73     signal maxy : unsigned (15 downto 0) := (others => '0');
74
75 begin
76
77     xcounter : process(clk)
78     begin
79         if rising_edge(clk) then

```

```

80     if endofrow = '1' or endofframe = '1' then
81         xcount <= (others => '0');
82         maxx <= xcount - 1;
83         elsif valid_green = '1' then
84             xcount <= xcount + 1;
85         end if;
86     end if;
87 end process xcounter;
88
89 ycounter : process(clk)
90 begin
91     if rising_edge(clk) then
92         if endofframe = '1' then
93             ycount <= (others => '0');
94             maxy <= ycount - 1;
95         elsif endofrow = '1' then
96             ycount <= ycount + 1;
97         end if;
98     end if;
99 end process ycounter;
100
101 -- Extract relevant data as image information comes in:
102 -- -- TOP EXTREME
103 -- -- BOTTOM EXTREME
104 -- -- LEFT EXTREME
105 -- -- RIGHT EXTREME
106 -- -- HORIZONTAL WIDTH AT A CONSTANT DISTANCE BELOW TOP EXTREME
107 -- -- HORIZONTAL WIDTH AT A CONSTANT DISTANCE ABOVE BOTTOM EXTREME
108
109 data_extraction : process (clk)
110 begin
111     if rising_edge(clk) then
112         if valid_green = '1' then
113             if pixel_data < threshold then
114
115 -- TOP
116                 if ycount <= topy then
117                     topy <= ycount;
118                     topx <= xcount;
119                 end if;
120
121 -- BOTTOM
122                 if ycount >= bottomy then
123                     bottomy <= ycount;
124                     bottomx <= xcount;
125                 end if;
126
127 -- LEFT
128                 if xcount <= leftx then
129                     leftx <= xcount;
130                     lefty <= ycount;
131                 end if;
132
133 -- RIGHT
134                 if xcount >= rightx then
135                     rightx <= xcount;
136                     righty <= ycount;
137                 end if;
138
139                 if not (topy = x"FFFF") and ycount = topy + WIDTHSAMPLEINTERVAL then
140                     if topwidth_start = x"FFFF" then
141                         topwidth_start <= xcount;
142                     end if;
143                 end if;
144
145                 if bottomwidth_start_array(0) = x"FFFF" then
146                     bottomwidth_start_array(0) <= xcount;
147                 end if;
148
149                 elsif pixel_data >= threshold then
150                     if not (topy = x"FFFF") and ycount = topy + WIDTHSAMPLEINTERVAL then
151                         if not (topwidth_start = x"FFFF") and topwidth_end = 0 then
152                             topwidth_end <= xcount;
153                         end if;
154                     end if;
155
156                     if not (bottomwidth_start_array(0) = x"FFFF") and bottomwidth_end_array(0) = 0 then
157                         bottomwidth_end_array(0) <= xcount;
158                     end if;

```



```

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

164 if endofrow = '1' then
166     bottomwidth_start_array (0) <= x"FFFF";
168     bottomwidth_start_array (1) <= bottomwidth_start_array (0);
170     bottomwidth_start_array (2) <= bottomwidth_start_array (1);
172     bottomwidth_start_array (3) <= bottomwidth_start_array (2);
174     bottomwidth_start_array (4) <= bottomwidth_start_array (3);
176     bottomwidth_start_array (5) <= bottomwidth_start_array (4);
178     bottomwidth_start_array (6) <= bottomwidth_start_array (5);
180     bottomwidth_start_array (7) <= bottomwidth_start_array (6);
182     bottomwidth_start_array (8) <= bottomwidth_start_array (7);
184     bottomwidth_start_array (9) <= bottomwidth_start_array (8);
186     bottomwidth_start_array (10) <= bottomwidth_start_array (9);
188     bottomwidth_start_array (11) <= bottomwidth_start_array (10);
190     bottomwidth_start_array (12) <= bottomwidth_start_array (11);
192     bottomwidth_start_array (13) <= bottomwidth_start_array (12);
194     bottomwidth_start_array (14) <= bottomwidth_start_array (13);
196     bottomwidth_start_array (15) <= bottomwidth_start_array (14);
198     bottomwidth_start_array (16) <= bottomwidth_start_array (15);
200     bottomwidth_start_array (17) <= bottomwidth_start_array (16);
202     bottomwidth_start_array (18) <= bottomwidth_start_array (17);
204     bottomwidth_start_array (19) <= bottomwidth_start_array (18);

206     bottomwidth_end_array (0) <= x"0000";
208     bottomwidth_end_array (1) <= bottomwidth_end_array (0);
210     bottomwidth_end_array (2) <= bottomwidth_end_array (1);
212     bottomwidth_end_array (3) <= bottomwidth_end_array (2);
214     bottomwidth_end_array (4) <= bottomwidth_end_array (3);
216     bottomwidth_end_array (5) <= bottomwidth_end_array (4);
218     bottomwidth_end_array (6) <= bottomwidth_end_array (5);
220     bottomwidth_end_array (7) <= bottomwidth_end_array (6);
222     bottomwidth_end_array (8) <= bottomwidth_end_array (7);
224     bottomwidth_end_array (9) <= bottomwidth_end_array (8);
226     bottomwidth_end_array (10) <= bottomwidth_end_array (9);
228     bottomwidth_end_array (11) <= bottomwidth_end_array (10);
230     bottomwidth_end_array (12) <= bottomwidth_end_array (11);
232     bottomwidth_end_array (13) <= bottomwidth_end_array (12);
234     bottomwidth_end_array (14) <= bottomwidth_end_array (13);
236     bottomwidth_end_array (15) <= bottomwidth_end_array (14);
238     bottomwidth_end_array (16) <= bottomwidth_end_array (15);
240     bottomwidth_end_array (17) <= bottomwidth_end_array (16);
242     bottomwidth_end_array (18) <= bottomwidth_end_array (17);
244     bottomwidth_end_array (19) <= bottomwidth_end_array (18);

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

256     prev_bottomy <= bottomy;

258 end if;

260 -- RESET AT END OF FRAME
262 if endofframe = '1' then
264     topx <= (others => '0');
266     topy <= (others => '1');
268     bottomx <= (others => '0');
270     bottomy <= (others => '0');
272     leftx <= (others => '1');
274     lefty <= (others => '0');
276     rightx <= (others => '0');
278     righty <= (others => '0');

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

284     prev_bottomy <= (others => '0');
286     bottomwidth_start <= (others => '0');
288     bottomwidth_end <= (others => '0');
290     bottomwidth_start_array <= (others => x"FFFF");
292     bottomwidth_end_array <= (others => x"0000");
294     bottomwidth_recorded <= '0';
296 end if;

298 end if;

```

```

240 end process data_extraction;
242
243 output : process(clk)
244     variable top_on_edge, bottom_on_edge, left_on_edge, right_on_edge,
245     toptentry, bottomentry, leftentry, rightentry : integer := 0;
246     variable xdiff, ydiff : unsigned (15 downto 0);
247 begin
248     if rising_edge(clk) then
249
250         if endofframe = '1' then
251
252             led0 <= '0';
253             led1 <= '0';
254             led2 <= '0';
255             led3 <= '0';
256             led4 <= '0';
257             led5 <= '0';
258             led6 <= '0';
259             led7 <= '0';
260
261             if topy = x"FFFF" and rightx = 0 and leftx = x"FFFF" and bottomy = 0 then
262                 no_detect <= '1';
263             else
264                 no_detect <= '0';
265             end if;
266
267             if topy <= BOUNDARY_TOLERANCE or topx <= BOUNDARY_TOLERANCE or
268                 topx > maxx - BOUNDARY_TOLERANCE then
269                 top_on_edge := 1;
270             else
271                 top_on_edge := 0;
272             end if;
273             if topy = 0 then
274                 toptentry := 1;
275             else
276                 toptentry := 0;
277             end if;
278
279             if bottomy > maxy - BOUNDARY_TOLERANCE or bottomx <= BOUNDARY_TOLERANCE or
280                 bottomx > maxx - BOUNDARY_TOLERANCE then
281                 bottom_on_edge := 1;
282             else
283                 bottom_on_edge := 0;
284             end if;
285             if bottomy = maxy then
286                 bottomentry := 1;
287             else
288                 bottomentry := 0;
289             end if;
290
291             if leftx <= BOUNDARY_TOLERANCE or lefty <= BOUNDARY_TOLERANCE or
292                 lefty > maxy - BOUNDARY_TOLERANCE then
293                 left_on_edge := 1;
294             else
295                 left_on_edge := 0;
296             end if;
297             if leftx = 0 then
298                 leftentry := 1;
299             else
300                 leftentry := 0;
301             end if;
302
303             if rightx > maxx - BOUNDARY_TOLERANCE or righty <= BOUNDARY_TOLERANCE or
304                 righty > maxy - BOUNDARY_TOLERANCE then
305                 right_on_edge := 1;
306             else
307                 right_on_edge := 0;
308             end if;
309             if rightx = maxx then
310                 rightentry := 1;
311             else
312                 rightentry := 0;
313             end if;
314
315             if (top_on_edge + bottom_on_edge + right_on_edge + left_on_edge) = 1 or
316                 (top_on_edge + bottom_on_edge + right_on_edge + left_on_edge) = 3 then
317                 if toptentry = 1 then
318                     xout <= bottomx;

```

```

320         yout <= bottomy;
321         led5 <= '1';
322     end if;

323
324     if bottometry = 1 then
325         xout <= topx;
326         yout <= topy;
327         led6 <= '1';
328     end if;

329
330     if leftentry = 1 then
331         xout <= rightx;
332         yout <= righty;
333         led0 <= '1';
334     end if;

335
336     if rightentry = 1 then
337         xout <= leftx;
338         yout <= lefty;
339     end if;

340
341     -----
342     elsif top_on_edge + bottom_on_edge + right_on_edge + left_on_edge = 2 then
343
344     -----TOP LEFT-----
345     if top_on_edge = 1 and left_on_edge = 1 then
346
347         if(bottomx > rightx) then
348             xdiff := bottomx - rightx;
349         else
350             xdiff := rightx - bottomx;
351         end if;
352
353         if(bottomy > righty) then
354             ydiff := bottomy - righty;
355         else
356             ydiff := righty - bottomy;
357         end if;
358
359         if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
360             xout <= rightx;
361             yout <= righty;
362         else
363             if bottomwidth_end - bottomwidth_start >= FINGER_WIDTH then
364                 xout <= rightx;
365                 yout <= righty;
366                 led1 <= '1';
367             else
368                 xout <= rightx;
369                 yout <= righty;
370                 led2 <= '1';
371             end if;
372         end if;
373
374     -----TOP RIGHT-----
375     elsif top_on_edge = 1 and right_on_edge = 1 then
376
377         if(bottomx > leftx) then
378             xdiff := bottomx - leftx;
379         else
380             xdiff := leftx - bottomx;
381         end if;
382
383         if(bottomy > lefty) then
384             ydiff := bottomy - lefty;
385         else
386             ydiff := lefty - bottomy;
387         end if;
388
389         if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
390             xout <= leftx;
391             yout <= lefty;
392         else
393             if bottomwidth_end - bottomwidth_start >= FINGER_WIDTH then
394                 xout <= leftx;
395                 yout <= lefty;
396             else
397                 xout <= leftx;
398                 yout <= lefty;
399             end if;

```

```

400         end if;
402 -----*****BOTTOM LEFT*****
         elsif bottom_on_edge = 1 and left_on_edge = 1 then
404             if(topx > rightx) then
406                 xdiff := topx - rightx;
             else
408                 xdiff := rightx - topx;
             end if;
410             if(topy > righty) then
412                 ydiff := topy - righty;
             else
414                 ydiff := righty - topy;
             end if;
416             if ( xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
418                 xout <= rightx;
                 yout <= righty;
420             else
                 if topwidth_end - topwidth_start < FINGER_WIDTH then
422                     xout <= topx;
                     yout <= topy;
424                     led3 <= '1';
                 else
426                     xout <= rightx;
                     yout <= righty;
428                     led4 <= '1';
                 end if;
430             end if;
432 -----*****BOTTOM RIGHT*****
         elsif bottom_on_edge = 1 and right_on_edge = 1 then
434             if(topx > leftx) then
436                 xdiff := topx - leftx;
             else
438                 xdiff := leftx - topx;
             end if;
440             if(topy > lefty) then
442                 ydiff := topy - lefty;
             else
444                 ydiff := lefty - topy;
             end if;
446             if ( xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
448                 xout <= leftx;
                 yout <= lefty;
450             else
                 if topwidth_end - topwidth_start < FINGER_WIDTH then
452                     xout <= topx;
                     yout <= topy;
454                     else
                         xout <= leftx;
                         yout <= lefty;
456                     end if;
458                 end if;
             end if;
460 -----*****
         end if;
462
464 -----
         end if; -- End of Frame
466     end if;
         end process output;
468
end rtl;

```

```

--
2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
4  --     Abdulhamid Ghandour
--     Thomas John
6  --     Jaime Peretzman
--     Bharadwaj Vellore
8  --
-- Desc:
10 -- Avalon Interface for the Vision Block.
-- readdata has X co-ord in lower 16 bits
12 --     has Y co-ord in next 15 bits
--     has no_detect in MSB
14 --
16 library ieee;
use ieee.std_logic_1164.all;
18 use ieee.numeric_std.all;
20 entity avalon_vision is
22 port (
-- Avalon Signals
24   clk       : in std_logic;
   reset_n    : in std_logic;
26   address   : in unsigned(4 downto 0);
   write      : in std_logic;
28   read      : in std_logic;
   chipselect : in std_logic;
30   readdata  : out unsigned(31 downto 0);
   writedata  : in unsigned(31 downto 0);
32
-- Camera Signals
34   master_clk : out std_logic;
   pixel_clk   : in std_logic;
36   line_valid  : in std_logic;
   frame_valid : in std_logic;
38   pixel_data  : in unsigned(9 downto 0);
40
-- Board Signals
   threshold   : in unsigned(9 downto 0); -- SW9 to SW0
42   no_detect   : out std_logic;         -- LEDGO
   cal_direction : out unsigned(6 downto 0);
44   vision_flags : out unsigned(7 downto 0)
);
46 end avalon_vision;
48 architecture toplevel of avalon_vision is
50 signal ram_address : unsigned(4 downto 0);
signal data_signal : unsigned(9 downto 0);
52 signal valid_green_signal : std_logic;
signal valid_green_cropped_signal : std_logic;
54 signal end_of_frame_signal : std_logic;
signal end_of_row_signal : std_logic;
56 signal end_of_row_cropped_signal : std_logic;
signal no_detect_signal : std_logic;
58 signal x_1_signal, x_2_signal, y_1_signal, y_2_signal : unsigned(10 downto 0);
signal calibration_on_signal : std_logic;
60 signal calibration_on_signal_int : std_logic;
signal repos_signal : unsigned(6 downto 0);
62 signal xout_signal : unsigned(15 downto 0);
signal yout_signal : unsigned(14 downto 0);
64 signal reset_int : std_logic := '0';
signal sandboxStartX_signal : unsigned(31 downto 0) := (others => '0');
66 signal sandboxEndX_signal : unsigned(31 downto 0) := (others => '1');
signal sandboxStartY_signal : unsigned(31 downto 0) := (others => '0');
68 signal sandboxEndY_signal : unsigned(31 downto 0) := (others => '1');
signal green_column_thr_signal : unsigned(9 downto 0) := "0100101100";
70 signal green_row_thr_signal : unsigned(9 downto 0) := "0110010000";
component ci_pxl port(
72   clk       : in std_logic;
   mclk      : out std_logic; -- Master CLK to Camera
74   lval      : in std_logic; -- Line Valid from Camera
   fval      : in std_logic; -- Frame Valid from Camera
76   pixclk    : in std_logic; -- Pixel CLK from Camera
   datain    : in unsigned(9 downto 0); -- Pixel Data from Camera
78   dataout   : out unsigned(9 downto 0);
   valid_green : out std_logic;

```

```

80     end_of_frame      : out std_logic;
81     end_of_row        : out std_logic;
82     sandboxStartX    : in unsigned(31 downto 0);
83     sandboxStartY    : in unsigned(31 downto 0);
84     sandboxEndX      : in unsigned(31 downto 0);
85     sandboxEndY      : in unsigned(31 downto 0)
86 );
87 end component;
88
89 component visionsystem port(
90     clk : in std_logic;
91     pixel_data : in unsigned (9 downto 0);
92     valid_green : in std_logic;
93     endofrow : in std_logic;
94     endofframe : in std_logic;
95     threshold : in unsigned (9 downto 0);
96     xout : out unsigned (15 downto 0);
97     yout : out unsigned (15 downto 0);
98     led0, led1, led2, led3, led4, led5, led6, led7 : out std_logic;
99     no_detect : out std_logic
100 );
101 end component;
102
103 component calibration port (
104     reset      : in std_logic;
105     clk        : in std_logic;
106     valid_green : in std_logic;
107     end_row    : in std_logic;
108     end_frame  : in std_logic;
109     green_pixel_value : in unsigned (9 downto 0);
110     -- green_column_thr : in unsigned (9 downto 0);
111     -- green_row_thr   : in unsigned (9 downto 0);
112     repos      : out unsigned (6 downto 0) := "1000000";
113     x_1        : out unsigned (10 downto 0) := "00000000000";
114     y_1        : out unsigned (10 downto 0) := "00000000000";
115     x_2        : out unsigned (10 downto 0) := "00000000000";
116     y_2        : out unsigned (10 downto 0) := "00000000000";
117     calibration_on : in std_logic;
118     threshold    : in unsigned (9 downto 0);
119     leds        : out unsigned(6 downto 0)
120 );
121 end component;
122
123 component imagecropper port (
124     clk : in std_logic;
125     valid_green_in : in std_logic;
126     valid_green_out : out std_logic;
127     end_row_in : in std_logic;
128     end_row_out : out std_logic;
129     end_frame : in std_logic;
130     crop_start_x : in unsigned (10 downto 0);
131     crop_end_x : in unsigned (10 downto 0);
132     crop_start_y : in unsigned (10 downto 0);
133     crop_end_y : in unsigned (10 downto 0)
134 );
135 end component;
136
137 begin
138     ram_address <= address;
139     reset_int <= not reset_n;
140     CAMERA: ci_pxl port map(
141         clk => clk ,
142         mclk => master_clk ,
143         lval => line_valid ,
144         fval => frame_valid ,
145         pixclk => pixel_clk ,
146         datain => pixel_data ,
147         dataout => data_signal ,
148         valid_green => valid_green_signal ,
149         end_of_frame => end_of_frame_signal ,
150         end_of_row => end_of_row_signal ,
151         sandboxStartX => sandboxStartX_signal ,
152         sandboxEndX => sandboxEndX_signal ,
153         sandboxStartY => sandboxStartY_signal ,
154         sandboxEndY => sandboxEndY_signal
155     );
156     VISION: visionsystem port map(
157         clk => clk ,
158         pixel_data => data_signal ,

```

```

160     valid_green => valid_green_cropped_signal ,
161     endofrow => end_of_row_cropped_signal ,
162     endofframe => end_of_frame_signal ,
163     threshold => threshold ,
164     xout => xout_signal ,
165     yout (14 downto 0) => yout_signal ,
166     no_detect => no_detect_signal ,
167     led0 => vision_flags(0) ,
168     led1 => vision_flags(1) ,
169     led2 => vision_flags(2) ,
170     led3 => vision_flags(3) ,
171     led4 => vision_flags(4) ,
172     led5 => vision_flags(5) ,
173     led6 => vision_flags(6) ,
174     led7 => vision_flags(7)
175 );
176
177 calibrator: calibration port map(
178     reset      => '0' ,
179     clk        => clk ,
180     valid_green => valid_green_cropped_signal ,
181     end_row    => end_of_row_cropped_signal ,
182     end_frame  => end_of_frame_signal ,
183     green_pixel_value => data_signal ,
184     green_column_thr => green_column_thr_signal ,
185     green_row_thr => green_row_thr_signal ,
186     repos      => repos_signal ,
187     x_1        => x_1_signal ,
188     y_1        => y_1_signal ,
189     x_2        => x_2_signal ,
190     y_2        => y_2_signal ,
191     calibration_on => calibration_on_signal ,
192     threshold  => threshold ,
193     leds(5 downto 0) => cal_direction(5 downto 0)
194 );
195
196 CROPPER: imagecropper port map(
197     clk        => clk ,
198     valid_green_in => valid_green_signal ,
199     valid_green_out => valid_green_cropped_signal ,
200     end_row_in    => end_of_row_signal ,
201     end_row_out   => end_of_row_cropped_signal ,
202     end_frame     => end_of_frame_signal ,
203     crop_start_x  => sandboxStartX_signal(10 downto 0) ,
204     crop_end_x    => sandboxEndX_signal(10 downto 0) ,
205     crop_start_y  => sandboxStartY_signal(10 downto 0) ,
206     crop_end_y    => sandboxEndY_signal(10 downto 0)
207 );
208
209 host_control: process (clk)
210 begin
211     if rising_edge(clk) then
212         if chipselect = '1' then
213             if write = '1' then
214                 if ram_address = 4 then
215                     calibration_on_signal_int <= writedata(0);
216                 elsif ram_address = 5 then
217                     sandboxStartX_signal <= writedata;
218                 elsif ram_address = 6 then
219                     sandboxEndX_signal <= writedata;
220                 elsif ram_address = 7 then
221                     sandboxStartY_signal <= writedata;
222                 elsif ram_address = 8 then
223                     sandboxEndY_signal <= writedata;
224                 elsif ram_address = 9 then
225                     green_column_thr_signal <= writedata( 9 downto 0);
226                 elsif ram_address = 10 then
227                     green_row_thr_signal <= writedata( 9 downto 0);
228                 end if;
229             end if;
230
231             if read = '1' then
232                 if ram_address = 0 then
233                     readdata(30 downto 0) <= yout_signal & xout_signal;
234                     readdata(31) <= no_detect_signal;
235                 elsif ram_address = 1 then
236                     readdata(10 downto 0) <= x_1_signal;
237                     readdata(21 downto 11) <= y_1_signal;
238                     readdata(31 downto 22) <= (others => '0');
239                 elsif ram_address = 2 then

```

```
240         readdata(10 downto 0) <= x_2_signal;
241         readdata(21 downto 11) <= y_2_signal;
242         readdata(31 downto 22) <= (others => '0');
243         elsif ram_address = 3 then
244             readdata <= ("000000000000000000000000" & repos_signal);
245             end if;
246         end if;
247     end if;-- end of chipselect
248 end process host_control;
249
250 no_detect <= no_detect_signal;
251 calibration_on_signal <= calibration_on_signal_int;
252 cal_direction(6) <= valid_green_signal;
253 end toplevel;
```



```

--
2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
--  Authors:
4  --     Abdulhamid Ghandour
--     Thomas John
6  --     Jaime Peretzman
--     Bharadwaj Vellore
8  --
--  Desc:
10 --
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
16 entity de2_vga_raster is
17 port (
18     reset      : in  std_logic;
19     clk        : in  std_logic;
20     read       : in  std_logic;
21     write      : in  std_logic;
22     chipselect : in  std_logic;
23     address    : in  unsigned(4 downto 0);
24     readdata   : out unsigned(15 downto 0);
25     writedata  : in  unsigned(15 downto 0);
26
27     VGA_CLK,           -- Clock
28     VGA_HS,           -- H_SYNC
29     VGA_VS,           -- V_SYNC
30     VGA_BLANK,        -- BLANK
31     VGA_SYNC : out std_logic; -- SYNC
32     VGA_R,           -- Red[9:0]
33     VGA_G,           -- Green[9:0]
34     VGA_B : out unsigned(9 downto 0) -- Blue[9:0]
35 );
36
37 end de2_vga_raster;
38
39 architecture rtl of de2_vga_raster is
40 -- Video parameters
41
42
43
44 constant HTOTAL      : integer := 800;
45 constant HSYNC      : integer := 96;
46 constant HBACKPORCH : integer := 48;
47 constant HACTIVE     : integer := 640;
48 constant HFRONTPORCH : integer := 16;
49
50 constant VTOTAL      : integer := 525;
51 constant VSYNC      : integer := 2;
52 constant VBACKPORCH : integer := 33;
53 constant VACTIVE     : integer := 480;
54 constant VFRONTPORCH : integer := 10;
55
56 constant ball_dia    : integer := 29;
57 constant cross_dia  : integer := 16;
58 constant border     : integer := 15;
59 signal black_b_x     : unsigned(7 downto 0) := "00001111";
60 signal black_b_y     : unsigned(7 downto 0) := "00011110";
61 signal border_1, border_2, border_3, border_4 : unsigned(9 downto 0) := "0000000000";
62 signal C_H_start_1   : unsigned(9 downto 0) := "0000000000";
63 signal C_V_Start_1   : unsigned(9 downto 0) := "0000000000";
64 signal C_color_1     : unsigned(2 downto 0) := "000";
65 signal C_H_start_2   : unsigned(9 downto 0) := "0000000000";
66 signal C_V_Start_2   : unsigned(9 downto 0) := "0000000000";
67 signal C_color_2     : unsigned(2 downto 0) := "000";
68 signal C_H_start_3   : unsigned(9 downto 0) := "0000000000";
69 signal C_V_Start_3   : unsigned(9 downto 0) := "0000000000";
70 signal C_color_3     : unsigned(2 downto 0) := "000";
71 signal C_H_start_4   : unsigned(9 downto 0) := "0000000000";
72 signal C_V_Start_4   : unsigned(9 downto 0) := "0000000000";
73 signal C_color_4     : unsigned(2 downto 0) := "000";
74 signal C_H_start_5   : unsigned(9 downto 0) := "0000000000";
75 signal C_V_Start_5   : unsigned(9 downto 0) := "0000000000";
76 signal C_color_5     : unsigned(2 downto 0) := "000";
77 signal C_H_start_6   : unsigned(9 downto 0) := "0000000000";
78 signal C_V_Start_6   : unsigned(9 downto 0) := "0000000000";
79 signal C_color_6     : unsigned(2 downto 0) := "000";

```

```

80  signal    C_H_start_7          : unsigned(9 downto 0) := "0000000000"
      signal    C_V_Start_7          : unsigned(9 downto 0) := "0000000000"
82  signal    C_color_7           : unsigned(2 downto 0) := "000";

84
86  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";
88  signal stick_V_2 : unsigned(9 downto 0) := "0000000000";
      signal cross_H   : unsigned(9 downto 0) := "0000000000";
90  signal cross_V    : unsigned(9 downto 0) := "0000000000";

92  signal temp_C_H_start_1 : unsigned(9 downto 0) := "0000000000";
      signal temp_C_V_Start_1 : unsigned(9 downto 0) := "0000000000";
94  signal temp_C_color_1   : unsigned(2 downto 0) := "000";
      signal temp_C_H_start_2 : unsigned(9 downto 0) := "0000000000";
96  signal temp_C_V_Start_2 : unsigned(9 downto 0) := "0000000000";
      signal temp_C_color_2   : unsigned(2 downto 0) := "000";
98  signal temp_C_H_start_3 : unsigned(9 downto 0) := "0000000000";
      signal temp_C_V_Start_3 : unsigned(9 downto 0) := "0000000000";
100 signal temp_C_color_3   : unsigned(2 downto 0) := "000";
      signal temp_C_H_start_4 : unsigned(9 downto 0) := "0000000000";
102 signal temp_C_V_Start_4 : unsigned(9 downto 0) := "0000000000";
      signal temp_C_color_4   : unsigned(2 downto 0) := "000";
104 signal temp_C_H_start_5 : unsigned(9 downto 0) := "0000000000";
      signal temp_C_V_Start_5 : unsigned(9 downto 0) := "0000000000";
106 signal temp_C_color_5   : unsigned(2 downto 0) := "000";
      signal temp_C_H_start_6 : unsigned(9 downto 0) := "0000000000";
108 signal temp_C_V_Start_6 : unsigned(9 downto 0) := "0000000000";
      signal temp_C_color_6   : unsigned(2 downto 0) := "000";
110 signal temp_C_H_start_7 : unsigned(9 downto 0) := "0000000000";
      signal temp_C_V_Start_7 : unsigned(9 downto 0) := "0000000000";
112 signal temp_C_color_7   : unsigned(2 downto 0) := "000";
      signal temp_stick_H_1   : unsigned(9 downto 0) := "0000000000";
114 signal temp_stick_V_1   : unsigned(9 downto 0) := "0000000000";
      signal temp_stick_H_2   : unsigned(9 downto 0) := "0000000000";
116 signal temp_stick_V_2   : unsigned(9 downto 0) := "0000000000";
      signal temp_cross_H     : unsigned(9 downto 0) := "0000000000";
118 signal temp_cross_V     : unsigned(9 downto 0) := "0000000000";

120 signal Socket_H_start_11 : unsigned(9 downto 0) := "0000000000"; --0
      signal Socket_V_Start_11 : unsigned(9 downto 0) := "0000000000"; --0
122
124 signal Socket_H_start_12 : unsigned(9 downto 0) := "0100110001"; --305
      signal Socket_V_Start_12 : unsigned(9 downto 0) := "0000000000"; --0
126 signal Socket_H_start_13 : unsigned(9 downto 0) := "1001100010"; --610
      signal Socket_V_Start_13 : unsigned(9 downto 0) := "0000000000"; --0
128
130 signal Socket_H_start_14 : unsigned(9 downto 0) := "0000000000"; --0
      signal Socket_V_Start_14 : unsigned(9 downto 0) := "0111000010"; --450
132 signal Socket_H_start_15 : unsigned(9 downto 0) := "0100110001"; --305
      signal Socket_V_Start_15 : unsigned(9 downto 0) := "0111000010"; --450
134
136 signal Socket_H_start_16 : unsigned(9 downto 0) := "1001100010"; --610
      signal Socket_V_Start_16 : unsigned(9 downto 0) := "0111000010"; --450

138 signal received_check : unsigned(20 downto 0) := "00000000000000000000";
      signal received_cal : unsigned(4 downto 0) := "00000";
140 signal calibration : std_logic := '0';
      signal temp_border : std_logic := '0';
142 signal margin : unsigned(4 downto 0) := "11111";

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

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

158 type color_mat is array (0 to 6) of unsigned (29 downto 0);
      constant color_RGB : color_mat := ("111111111111111111111111111111",

```

```

160             "1111111111111111111111100000000000",
162             "00000000001111111111111111111111",
164             "000000000011111111111111111000000000",
166             "11111111111100100000000000000000",
168             "1111000100111111111111010100100",
170             "111111111111001000001100100000");
172
174 type cross_matrix is array (0 to 15) of unsigned (0 to 15);
176 constant cross_boundary : cross_matrix := (
178     "111111111110000",
180     "1100000000000000",
182     "1010000000000000",
184     "1001000000000000",
186     "1000100000000000",
188     "1000010000000000",
190     "1000001000000000",
192     "1000000100000000",
194     "1000000010000000",
196     "1000000001000000",
198     "1000000000100000",
200     "1000000000010000",
202     "0000000000001000",
204     "0000000000000100",
206     "0000000000000010",
208     "0000000000000001");
210
212 type c_matrix is array (0 to 28) of unsigned (0 to 28);
214 constant C_boundary : c_matrix := (
216     "0000000001111111000000000000",
218     "0000000001111111111000000000",
220     "0000000111111111111110000000",
222     "00000011111111111111111000000",
224     "000001111111111111111111100000",
226     "0000111111111111111111111110000",
228     "0001111111111111111111111111000",
230     "0011111111111111111111111111100",
232     "0011111111111111111111111111100",
234     "0111111111111111111111111111110",
236     "0111111111111111111111111111110",
238     "1111111111111111111111111111111",
240     "1111111111111111111111111111111",
242     "1111111111111111111111111111111",
244     "1111111111111111111111111111111",
246     "1111111111111111111111111111111",
248     "1111111111111111111111111111111",
250     "1111111111111111111111111111111",
252     "1111111111111111111111111111111",
254     "0111111111111111111111111111110",
256     "0111111111111111111111111111110",
258     "0011111111111111111111111111100",
260     "0011111111111111111111111111100",
262     "0001111111111111111111111111100",
264     "00001111111111111111111111111000",
266     "000001111111111111111111111110000",
268     "000000111111111111111111111110000",
270     "0000000111111111111111111111100000",
272     "000000000111111111111111111000000",
274     "00000000000111111111111111111000000");
276
278 begin
280
282     process (clk)
284     begin
286         if rising_edge(clk) then
288             clk25 <= not clk25;
290         end if;
292     end process;
294
296     -- Horizontal and vertical counters
298
300     soft_input          : process (clk)
302     variable temp_mid : unsigned(9 downto 0);
304     begin
306         if rising_edge(clk) then
308             if reset = '1' then
310                 temp_C_H_start_1 <= (others => '0');
312                 temp_C_V_start_1 <= (others => '0');
314                 temp_C_color_1 <= (others => '0');
316                 C_H_start_1 <= (others => '0');
318                 C_V_start_1 <= (others => '0');
320                 C_color_1 <= (others => '0');
322                 temp_C_H_start_2 <= (others => '0');

```

```

240     temp_C_V_Start_2      <= (others => '0');
      temp_C_color_2       <= (others => '0');
242     C_H_start_2         <= (others => '0');
      C_V_start_2         <= (others => '0');
244     C_color_2          <= (others => '0');
      temp_C_H_start_3    <= (others => '0');
246     temp_C_V_Start_3    <= (others => '0');
      temp_C_color_3     <= (others => '0');
248     C_H_start_3       <= (others => '0');
      C_V_start_3       <= (others => '0');
250     C_color_3        <= (others => '0');
      temp_C_H_start_4    <= (others => '0');
252     temp_C_V_Start_4    <= (others => '0');
      temp_C_color_4     <= (others => '0');
254     C_H_start_4       <= (others => '0');
      C_V_start_4       <= (others => '0');
256     C_color_4        <= (others => '0');
      temp_C_H_start_5    <= (others => '0');
258     temp_C_V_Start_5    <= (others => '0');
      temp_C_color_5     <= (others => '0');
260     C_H_start_5       <= (others => '0');
      C_V_start_5       <= (others => '0');
262     C_color_5        <= (others => '0');
      temp_C_H_start_6    <= (others => '0');
264     temp_C_V_Start_6    <= (others => '0');
      temp_C_color_6     <= (others => '0');
266     C_H_start_6       <= (others => '0');
      C_V_start_6       <= (others => '0');
268     C_color_6        <= (others => '0');
      temp_C_H_start_7    <= (others => '0');
270     temp_C_V_Start_7    <= (others => '0');
      temp_C_color_7     <= (others => '0');
272     C_H_start_7       <= (others => '0');
      C_V_start_7       <= (others => '0');
274     C_color_7        <= (others => '0');
      received_check     <= (others => '0');
276     received_cal      <= (others => '0');
      black_b_x         <= (others => '0');
278     black_b_y         <= (others => '0');
else
280     if chipselect = '1' then
      if write = '1' then
282         if address = "00000" then
          temp_C_H_start_1 <= writedata(9 downto 0);
          received_check(0) <= '1';
284         elsif address = "00001" then
          temp_C_V_Start_1 <= writedata(9 downto 0);
          received_check(1) <= '1';
286         elsif address = "00010" then
          temp_C_color_1 <= writedata(2 downto 0);
          received_check(2) <= '1';
288
290
292         elsif address = "00011" then
          temp_C_H_start_2 <= writedata(9 downto 0);
          received_check(3) <= '1';
294         elsif address = "00100" then
          temp_C_V_Start_2 <= writedata(9 downto 0);
          received_check(4) <= '1';
296         elsif address = "00101" then
          temp_C_color_2 <= writedata(2 downto 0);
          received_check(5) <= '1';
298
300
302
304         elsif address = "00110" then
          temp_C_H_start_3 <= writedata(9 downto 0);
          received_check(6) <= '1';
306         elsif address = "00111" then
          temp_C_V_Start_3 <= writedata(9 downto 0);
          received_check(7) <= '1';
308         elsif address = "01000" then
          temp_C_color_3 <= writedata(2 downto 0);
          received_check(8) <= '1';
310
312
314         elsif address = "01001" then
          temp_C_H_start_4 <= writedata(9 downto 0);
          received_check(9) <= '1';
316         elsif address = "01010" then
          temp_C_V_Start_4 <= writedata(9 downto 0);
          received_check(10) <= '1';
318

```

```

320     elsif address = "01011" then
322         temp_C_color_4    <= writedata(2 downto 0);
            received_check(11) <= '1';

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

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

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

354
356     elsif address = "01101" then
            temp_cross_H <= writedata(9 downto 0);

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

360

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

382
            end if; -- end of if write
384     if read = '1' and address = "01100" then
            if received_check = "00000000000000000000" then
386         readdata(0) <= '0';
            else
388         readdata(0) <= '1';
            end if;
390     end if; --end of read
            if read = '1' and address = "10100" then
392         if received_cal = "00000" then
            readdata(0) <= '0';
394         else
            readdata(0) <= '1';
396         end if;
            end if; --end of read
398     end if; --ship select
            if EndOfLine = '1' and EndOfField = '1' then

```

```

400     if received_check = "11111111111111111111" then
402         C_H_start_1    <= temp_C_H_start_1;
403         C_V_start_1    <= temp_C_V_start_1;
404         C_color_1      <= temp_C_color_1;
405         C_H_start_2    <= temp_C_H_start_2;
406         C_V_start_2    <= temp_C_V_start_2;
407         C_color_2      <= temp_C_color_2;
408         C_H_start_3    <= temp_C_H_start_3;
409         C_V_start_3    <= temp_C_V_start_3;
410         C_color_3      <= temp_C_color_3;
411         C_H_start_4    <= temp_C_H_start_4;
412         C_V_start_4    <= temp_C_V_start_4;
413         C_color_4      <= temp_C_color_4;
414         C_H_start_5    <= temp_C_H_start_5;
415         C_V_start_5    <= temp_C_V_start_5;
416         C_color_5      <= temp_C_color_5;
417         C_H_start_6    <= temp_C_H_start_6;
418         C_V_start_6    <= temp_C_V_start_6;
419         C_color_6      <= temp_C_color_6;
420         C_H_start_7    <= temp_C_H_start_7;
421         C_V_start_7    <= temp_C_V_start_7;
422         C_color_7      <= temp_C_color_7;
423         cross_H        <= temp_cross_H;
424         cross_V        <= temp_cross_V;
425         calibration    <= '0';
426         received_check <= (others => '0');
427     elsif received_cal = "11111" and temp_border = '0' then
428         stick_H_1      <= temp_stick_H_1;
429         stick_V_1      <= temp_stick_V_1;
430         stick_H_2      <= temp_stick_H_2;
431         stick_V_2      <= temp_stick_V_2;
432         cross_H        <= temp_cross_H;
433         cross_V        <= temp_cross_V;
434         calibration    <= '1';
435         received_cal   <= (others => '0');
436     elsif received_cal = "11111" and temp_border = '1' then
437         border_1       <= temp_stick_H_1;
438         border_2       <= temp_stick_V_1;
439         border_3       <= temp_stick_H_2;
440         border_4       <= temp_stick_V_2;
441         temp_mid := (temp_stick_H_1+temp_stick_H_2-border-border);
442         Socket_H_start_11 <= (temp_stick_H_1-border);
443         Socket_H_start_14 <= (temp_stick_H_1-border);
444         Socket_V_start_11 <= (temp_stick_V_1-border);
445         Socket_V_start_12 <= (temp_stick_V_1-border);
446         Socket_V_start_13 <= (temp_stick_V_1-border);
447         Socket_H_start_13 <= (temp_stick_H_2-border);
448         Socket_H_start_16 <= (temp_stick_H_2-border);
449         Socket_V_start_14 <= (temp_stick_V_2-border);
450         Socket_V_start_15 <= (temp_stick_V_2-border);
451         Socket_V_start_16 <= (temp_stick_V_2-border);
452         Socket_H_start_12 <= ('0'&(amp;(temp_mid(9 downto 1))));
453         Socket_H_start_15 <= ('0'&(amp;(temp_mid(9 downto 1))));
454         cross_H         <= temp_cross_H;
455         cross_V         <= temp_cross_V;
456         calibration    <= '0';
457         received_cal   <= (others => '0');
458     end if;
459 end if;
460
461 end if;
462 end process soft_input;
463
464
465 HCounter : process (clk25)
466 begin
467     if rising_edge(clk25) then
468         if reset = '1' then
469             Hcount <= (others => '0');
470         elsif EndOfLine = '1' then
471             Hcount <= (others => '0');
472         else
473             Hcount <= Hcount + 1;
474         end if;
475     end if;
476 end process HCounter;
477
478 EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

```

```

480 VCounter : process (clk25)
481 begin
482   if rising_edge(clk25) then
483     if reset = '1' then
484       Vcount <= (others => '0');
485     elsif EndOfLine = '1' then
486       if EndOfField = '1' then
487         Vcount <= (others => '0');
488       else
489         Vcount <= Vcount + 1;
490       end if;
491     end if;
492   end if;
493 end process VCounter;
494
495 EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';
496
497 -- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK
498
499 HSyncGen : process (clk25)
500 begin
501   if rising_edge(clk25) then
502     if reset = '1' or EndOfLine = '1' then
503       vga_hsync <= '1';
504     elsif Hcount = HSYNC - 1 then
505       vga_hsync <= '0';
506     end if;
507   end if;
508 end process HSyncGen;
509
510 HBlankGen : process (clk25)
511 begin
512   if rising_edge(clk25) then
513     if reset = '1' then
514       vga_hblank <= '1';
515     elsif Hcount = HSYNC + HBACK_PORCH then
516       vga_hblank <= '0';
517     elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
518       vga_hblank <= '1';
519     end if;
520   end if;
521 end process HBlankGen;
522
523 VSyncGen : process (clk25)
524 begin
525   if rising_edge(clk25) then
526     if reset = '1' then
527       vga_vsync <= '1';
528     elsif EndOfLine = '1' then
529       if EndOfField = '1' then
530         vga_vsync <= '1';
531       elsif Vcount = VSYNC - 1 then
532         vga_vsync <= '0';
533       end if;
534     end if;
535   end if;
536 end process VSyncGen;
537
538 VBlankGen : process (clk25)
539 begin
540   if rising_edge(clk25) then
541     if reset = '1' then
542       vga_vblank <= '1';
543     elsif EndOfLine = '1' then
544       if Vcount = VSYNC + VBACK_PORCH - 1 then
545         vga_vblank <= '0';
546       elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
547         vga_vblank <= '1';
548       end if;
549     end if;
550   end if;
551 end process VBlankGen;
552
553
554
555
556
557
558

```

```

560
562
564
566
568 -- BALL generator 1
RectangleHGen_1      : process (clk25)
570   variable H_boundary : unsigned(0 to 28);
   variable h_index_1   : unsigned(9 downto 0);
572   variable v_index_1   : unsigned(9 downto 0);

574 begin
   if rising_edge(clk25) then
576     if reset = '1' then
       rectangle_1 <= '0';
578     elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_1 - 1 and
       Vcount > VSYNC + VBACK_PORCH + C_V_Start_1 - 1 then
580       if Hcount < HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia and
       Vcount < VSYNC + VBACK_PORCH + C_V_Start_1 + ball_dia then
582         h_index_1 := Hcount - HSYNC - HBACK_PORCH - C_H_start_1;
         v_index_1 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_1;
584         H_boundary := (others => '0');
         H_boundary := C_boundary(TO_INTEGER(v_index_1));
586         if H_boundary(TO_INTEGER(h_index_1)) = '1' then
           rectangle_1 <= '1';
588         elsif H_boundary(TO_INTEGER(h_index_1)) = '0' then
           rectangle_1 <= '0';
590         end if;
         elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia then
592           rectangle_1 <= '0';
         end if;
594         elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia then
           rectangle_1 <= '0';
596         end if;
         end if;
598 end process RectangleHGen_1;

600
602 -- BALL generator 2
RectangleHGen_2      : process (clk25)
604   variable H_boundary : unsigned(0 to 28);
   variable h_index_2   : unsigned(9 downto 0);
606   variable v_index_2   : unsigned(9 downto 0);

608 begin
   if rising_edge(clk25) then
610     if reset = '1' then
       rectangle_2 <= '0';
612     elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_2 - 1 and
       Vcount > VSYNC + VBACK_PORCH + C_V_Start_2 - 1 then
614       if Hcount < HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia and
       Vcount < VSYNC + VBACK_PORCH + C_V_Start_2 + ball_dia then
616         h_index_2 := Hcount - HSYNC - HBACK_PORCH - C_H_start_2;
         v_index_2 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_2;
618         H_boundary := (others => '0');
         H_boundary := C_boundary(TO_INTEGER(v_index_2));
620         if H_boundary(TO_INTEGER(h_index_2)) = '1' then
           rectangle_2 <= '1';
622         elsif H_boundary(TO_INTEGER(h_index_2)) = '0' then
           rectangle_2 <= '0';
624         end if;
         elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia then
626           rectangle_2 <= '0';
         end if;
628         elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia then
           rectangle_2 <= '0';
630         end if;
         end if;
632 end process RectangleHGen_2;

634
636 -- BALL generator 3
RectangleHGen_3      : process (clk25)
638   variable H_boundary : unsigned(0 to 28);
   variable h_index_3   : unsigned(9 downto 0);

```



```

640     variable v_index_3 : unsigned(9 downto 0);
642 begin
643     if rising_edge(clk25) then
644         if reset = '1' then
645             rectangle_3 <= '0';
646         elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_3 - 1 and
647             Vcount > VSYNC + VBACK_PORCH + C_V_Start_3 - 1 then
648             if Hcount < HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia and
649                 Vcount < VSYNC + VBACK_PORCH + C_V_Start_3 + ball_dia then
650                 h_index_3 := Hcount - HSYNC - HBACK_PORCH - C_H_start_3;
651                 v_index_3 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_3;
652                 H_boundary := (others => '0');
653                 H_boundary := C_boundary(TO_INTEGER(v_index_3));
654                 if H_boundary(TO_INTEGER(h_index_3)) = '1' then
655                     rectangle_3 <= '1';
656                 elsif H_boundary(TO_INTEGER(h_index_3)) = '0' then
657                     rectangle_3 <= '0';
658                 end if;
659                 elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia then
660                     rectangle_3 <= '0';
661                 end if;
662                 elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia then
663                     rectangle_3 <= '0';
664                 end if;
665             end if;
666         end process RectangleHGen_3;
668
669     -- BALL generator 4
670
671     RectangleHGen_4 : process (clk25)
672         variable H_boundary : unsigned(0 to 28);
673         variable h_index_4 : unsigned(9 downto 0);
674         variable v_index_4 : unsigned(9 downto 0);
676     begin
677         if rising_edge(clk25) then
678             if reset = '1' then
679                 rectangle_4 <= '0';
680             elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_4 - 1 and
681                 Vcount > VSYNC + VBACK_PORCH + C_V_Start_4 - 1 then
682                 if Hcount < HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia and
683                     Vcount < VSYNC + VBACK_PORCH + C_V_Start_4 + ball_dia then
684                     h_index_4 := Hcount - HSYNC - HBACK_PORCH - C_H_start_4;
685                     v_index_4 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_4;
686                     H_boundary := (others => '0');
687                     H_boundary := C_boundary(TO_INTEGER(v_index_4));
688                     if H_boundary(TO_INTEGER(h_index_4)) = '1' then
689                         rectangle_4 <= '1';
690                     elsif H_boundary(TO_INTEGER(h_index_4)) = '0' then
691                         rectangle_4 <= '0';
692                     end if;
693                     elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia then
694                         rectangle_4 <= '0';
695                     end if;
696                     elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia then
697                         rectangle_4 <= '0';
698                     end if;
699                 end if;
700             end process RectangleHGen_4;
702
703     -- BALL generator 5
704
705     RectangleHGen_5 : process (clk25)
706         variable H_boundary : unsigned(0 to 28);
707         variable h_index_5 : unsigned(9 downto 0);
708         variable v_index_5 : unsigned(9 downto 0);
710     begin
711         if rising_edge(clk25) then
712             if reset = '1' then
713                 rectangle_5 <= '0';
714             elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_5 - 1 and
715                 Vcount > VSYNC + VBACK_PORCH + C_V_Start_5 - 1 then
716                 if Hcount < HSYNC + HBACK_PORCH + C_H_start_5 + ball_dia and
717                     Vcount < VSYNC + VBACK_PORCH + C_V_Start_5 + ball_dia then
718                     h_index_5 := Hcount - HSYNC - HBACK_PORCH - C_H_start_5;
719                     v_index_5 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_5;
720                     H_boundary := (others => '0');

```

```

720     H_boundary := C_boundary(TO_INTEGER(v_index_5));
721     if H_boundary(TO_INTEGER(h_index_5)) = '1' then
722         rectangle_5 <= '1';
723     elsif H_boundary(TO_INTEGER(h_index_5)) = '0' then
724         rectangle_5 <= '0';
725     end if;
726     elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_5 + ball_dia then
727         rectangle_5 <= '0';
728     end if;
729     elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_5 + ball_dia then
730         rectangle_5 <= '0';
731     end if;
732 end if;
733 end process RectangleHGen_5;
734
735 -- BALL generator 6
736
737 RectangleHGen_6      : process (clk25)
738     variable H_boundary : unsigned(0 to 28);
739     variable h_index_6  : unsigned(9 downto 0);
740     variable v_index_6  : unsigned(9 downto 0);
741
742 begin
743     if rising_edge(clk25) then
744         if reset = '1' then
745             rectangle_6 <= '0';
746         elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_6 - 1 and
747             Vcount > VSYNC + VBACK_PORCH + C_V_Start_6 - 1 then
748             if Hcount < HSYNC + HBACK_PORCH + C_H_start_6 + ball_dia and
749                 Vcount < VSYNC + VBACK_PORCH + C_V_Start_6 + ball_dia then
750                 h_index_6 := Hcount - HSYNC - HBACK_PORCH - C_H_start_6;
751                 v_index_6 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_6;
752                 H_boundary := (others => '0');
753                 H_boundary := C_boundary(TO_INTEGER(v_index_6));
754                 if H_boundary(TO_INTEGER(h_index_6)) = '1' then
755                     rectangle_6 <= '1';
756                 elsif H_boundary(TO_INTEGER(h_index_6)) = '0' then
757                     rectangle_6 <= '0';
758                 end if;
759                 elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_6 + ball_dia then
760                     rectangle_6 <= '0';
761                 end if;
762                 elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_6 + ball_dia then
763                     rectangle_6 <= '0';
764                 end if;
765             end if;
766         end process RectangleHGen_6;
767
768 -- BALL generator 7
769
770 RectangleHGen_7      : process (clk25)
771     variable H_boundary : unsigned(0 to 28);
772     variable h_index_7  : unsigned(9 downto 0);
773     variable v_index_7  : unsigned(9 downto 0);
774
775 begin
776     if rising_edge(clk25) then
777         if reset = '1' then
778             rectangle_7 <= '0';
779         elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_7 - 1 and
780             Vcount > VSYNC + VBACK_PORCH + C_V_Start_7 - 1 then
781             if Hcount < HSYNC + HBACK_PORCH + C_H_start_7 + ball_dia and
782                 Vcount < VSYNC + VBACK_PORCH + C_V_Start_7 + ball_dia then
783                 h_index_7 := Hcount - HSYNC - HBACK_PORCH - C_H_start_7;
784                 v_index_7 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_7;
785                 H_boundary := (others => '0');
786                 H_boundary := C_boundary(TO_INTEGER(v_index_7));
787                 if H_boundary(TO_INTEGER(h_index_7)) = '1' then
788                     rectangle_7 <= '1';
789                 elsif H_boundary(TO_INTEGER(h_index_7)) = '0' then
790                     rectangle_7 <= '0';
791                 end if;
792                 elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_7 + ball_dia then
793                     rectangle_7 <= '0';
794                 end if;
795                 elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_7 + ball_dia then
796                     rectangle_7 <= '0';
797                 end if;
798             end if;
799         end process;
800     end if;

```

```

800 end process RectangleHGen_7;
802
803 -- Socket generator 1
804
805 RectangleHGen_11 : process (clk25)
806   variable H_boundary : unsigned(0 to 28);
807   variable h_index_11 : unsigned(9 downto 0);
808   variable v_index_11 : unsigned(9 downto 0);
809
810   begin
811     if rising_edge(clk25) then
812       if reset = '1' then
813         rectangle_11 <= '0';
814       elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_11 - 1 and
815             Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_11 - 1 then
816         if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_11 + ball_dia and
817             Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_11 + ball_dia then
818           h_index_11 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_11;
819           v_index_11 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_11;
820           H_boundary := (others => '0');
821           H_boundary := C_boundary(TO_INTEGER(v_index_11));
822           if H_boundary(TO_INTEGER(h_index_11)) = '1' then
823             rectangle_11 <= '1';
824           elsif H_boundary(TO_INTEGER(h_index_11)) = '0' then
825             rectangle_11 <= '0';
826           end if;
827           elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_11 + ball_dia then
828             rectangle_11 <= '0';
829           end if;
830           elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_11 + ball_dia then
831             rectangle_11 <= '0';
832           end if;
833         end if;
834       end process RectangleHGen_11;
835
836
837 -- Socket generator 2
838
839 RectangleHGen_12 : process (clk25)
840   variable H_boundary : unsigned(0 to 28);
841   variable h_index_12 : unsigned(9 downto 0);
842   variable v_index_12 : unsigned(9 downto 0);
843
844   begin
845     if rising_edge(clk25) then
846       if reset = '1' then
847         rectangle_12 <= '0';
848       elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_12 - 1 and
849             Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_12 - 1 then
850         if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia and
851             Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_12 + ball_dia then
852           h_index_12 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_12;
853           v_index_12 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_12;
854           H_boundary := (others => '0');
855           H_boundary := C_boundary(TO_INTEGER(v_index_12));
856           if H_boundary(TO_INTEGER(h_index_12)) = '1' then
857             rectangle_12 <= '1';
858           elsif H_boundary(TO_INTEGER(h_index_12)) = '0' then
859             rectangle_12 <= '0';
860           end if;
861           elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia then
862             rectangle_12 <= '0';
863           end if;
864           elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia then
865             rectangle_12 <= '0';
866           end if;
867         end if;
868       end process RectangleHGen_12;
869
870
871 -- Socket generator 3
872
873 RectangleHGen_13 : process (clk25)
874   variable H_boundary : unsigned(0 to 28);
875   variable h_index_13 : unsigned(9 downto 0);
876   variable v_index_13 : unsigned(9 downto 0);
877
878   begin
879     if rising_edge(clk25) then
880       if reset = '1' then

```

```

880     rectangle_13     <= '0';
882     elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_13 - 1 and
884     Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_13 - 1 then
886     if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia and
888     Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_13 + ball_dia then
890     h_index_13 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_13;
892     v_index_13 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_13;
894     H_boundary := (others => '0');
896     H_boundary := C_boundary(TO_INTEGER(v_index_13));
898     if H_boundary(TO_INTEGER(h_index_13)) = '1' then
900     rectangle_13 <= '1';
902     elsif H_boundary(TO_INTEGER(h_index_13)) = '0' then
904     rectangle_13 <= '0';
906     end if;
908     elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia then
910     rectangle_13 <= '0';
912     end if;
914     elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia then
916     rectangle_13 <= '0';
918     end if;
920     end if;
922 end process RectangleHGen_13;
924
926 -- Socket generator 4
928
930 RectangleHGen_14 : process (clk25)
932 variable H_boundary : unsigned(0 to 28);
934 variable h_index_14 : unsigned(9 downto 0);
936 variable v_index_14 : unsigned(9 downto 0);
938
940 begin
942 if rising_edge(clk25) then
944 if reset = '1' then
946 rectangle_14 <= '0';
948 elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_14 - 1 and
950 Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_14 - 1 then
952 if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia and
954 Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_14 + ball_dia then
956 h_index_14 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_14;
958 v_index_14 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_14;
960 H_boundary := (others => '0');
962 H_boundary := C_boundary(TO_INTEGER(v_index_14));
964 if H_boundary(TO_INTEGER(h_index_14)) = '1' then
966 rectangle_14 <= '1';
968 elsif H_boundary(TO_INTEGER(h_index_14)) = '0' then
970 rectangle_14 <= '0';
972 end if;
974 elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia then
976 rectangle_14 <= '0';
978 end if;
980 elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia then
982 rectangle_14 <= '0';
984 end if;
986 end if;
988 end process RectangleHGen_14;
990
992 -- Socket generator 5
994
996 RectangleHGen_15 : process (clk25)
998 variable H_boundary : unsigned(0 to 28);
1000 variable h_index_15 : unsigned(9 downto 0);
1002 variable v_index_15 : unsigned(9 downto 0);
1004
1006 begin
1008 if rising_edge(clk25) then
1010 if reset = '1' then
1012 rectangle_15 <= '0';
1014 elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_15 - 1 and
1016 Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_15 - 1 then
1018 if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_15 + ball_dia and
1020 Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_15 + ball_dia then
1022 h_index_15 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_15;
1024 v_index_15 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_15;
1026 H_boundary := (others => '0');
1028 H_boundary := C_boundary(TO_INTEGER(v_index_15));
1030 if H_boundary(TO_INTEGER(h_index_15)) = '1' then
1032 rectangle_15 <= '1';
1034 elsif H_boundary(TO_INTEGER(h_index_15)) = '0' then
1036 rectangle_15 <= '0';
1038 end if;
1040 end if;
1042 end if;
1044 end process RectangleHGen_15;

```

```

960     elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_15 + ball_dia then
961         rectangle_15 <= '0';
962     end if;
963     elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_15 + ball_dia then
964         rectangle_15 <= '0';
965     end if;
966 end if;
967 end process RectangleHGen_15;
968
969 --- Socket generator 6
970
971 RectangleHGen_16 : process (clk25)
972     variable H_boundary : unsigned(0 to 28);
973     variable h_index_16 : unsigned(9 downto 0);
974     variable v_index_16 : unsigned(9 downto 0);
975
976 begin
977     if rising_edge(clk25) then
978         if reset = '1' then
979             rectangle_16 <= '0';
980             elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_16 - 1 and
981                 Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_16 - 1 then
982                 if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_16 + ball_dia and
983                     Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_16 + ball_dia then
984                     h_index_16 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_16;
985                     v_index_16 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_16;
986                     H_boundary := (others => '0');
987                     H_boundary := C_boundary(TO_INTEGER(v_index_16));
988                     if H_boundary(TO_INTEGER(h_index_16)) = '1' then
989                         rectangle_16 <= '1';
990                     elsif H_boundary(TO_INTEGER(h_index_16)) = '0' then
991                         rectangle_16 <= '0';
992                     end if;
993                     elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_16 + ball_dia then
994                         rectangle_16 <= '0';
995                     end if;
996                     elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_16 + ball_dia then
997                         rectangle_16 <= '0';
998                     end if;
999                 end if;
1000 end process RectangleHGen_16;
1001
1002
1003 -----stick for calibration
1004
1005 RectangleHGen : process (clk25)
1006 begin
1007     if rising_edge(clk) then
1008         if reset = '1' or Hcount = HSYNC + HBACK_PORCH + stick_H_1 then
1009             stick_h <= '1';
1010         elsif Hcount = HSYNC + HBACK_PORCH + stick_H_2 then
1011             stick_h <= '0';
1012         end if;
1013     end if;
1014 end process RectangleHGen;
1015
1016 RectangleVGen : process (clk25)
1017 begin
1018     if rising_edge(clk) then
1019         if reset = '1' then
1020             stick_v <= '0';
1021         elsif EndOfLine = '1' then
1022             if Vcount = VSYNC + VBACK_PORCH - 1 + stick_V_1 then
1023                 stick_v <= '1';
1024             elsif Vcount = VSYNC + VBACK_PORCH - 1 + stick_V_2 then
1025                 stick_v <= '0';
1026             end if;
1027         end if;
1028     end if;
1029 end process RectangleVGen;
1030
1031 stick <= stick_h and stick_v;
1032
1033 -----crosshair
1034
1035 RectangleHGen_00 : process (clk25)
1036     variable H_boundary : unsigned(0 to 15);
1037     variable h_index_00 : unsigned(9 downto 0);
1038     variable v_index_00 : unsigned(9 downto 0);

```

```

1040 begin
1042   if rising_edge(clk25) then
1044     if reset = '1' then
1046       rectangle_00 <= '0';
1048       elsif Hcount > HSYNC + HBACK_PORCH + cross_H - 1 and
1050       Vcount > VSYNC + VBACK_PORCH + cross_V - 1 then
1052         if Hcount < HSYNC + HBACK_PORCH + cross_H + cross_dia and
1054         Vcount < VSYNC + VBACK_PORCH + cross_V + cross_dia then
1056           h_index_00 := Hcount - HSYNC - HBACK_PORCH - cross_H;
1058           v_index_00 := Vcount - VSYNC - VBACK_PORCH - cross_V;
1060           H_boundary := (others => '0');
1062           H_boundary := cross_boundary(TO_INTEGER(v_index_00));
1064           if H_boundary(TO_INTEGER(h_index_00)) = '1' then
1066             rectangle_00 <= '1';
1068             elsif H_boundary(TO_INTEGER(h_index_00)) = '0' then
1070               rectangle_00 <= '0';
1072             end if;
1074             elsif Hcount >= HSYNC + HBACK_PORCH + cross_H + cross_dia then
1076               rectangle_00 <= '0';
1078             end if;
1080             elsif Hcount = HSYNC + HBACK_PORCH + cross_H + cross_dia then
1082               rectangle_00 <= '0';
1084             end if;
1086           end if;
1088         end process RectangleHGen_00;
1090
1092         ----- output
1094         VideoOut : process (clk25, reset)
1096         begin
1098           if reset = '1' then
1100             VGA_R <= "0000000000";
1102             VGA_G <= "0000000000";
1104             VGA_B <= "0000000000";
1106             elsif clk25'event and clk25 = '1' then
1108               if calibration = '1' then
1110                 if rectangle_00 = '1' then
1112                   VGA_R <= "1111111111";
1114                   VGA_G <= "1111111111";
1116                   VGA_B <= "1111111111";
1118                   elsif stick = '1' then
1120                     VGA_R <= "0000000000";
1122                     VGA_G <= "0000000000";
1124                     VGA_B <= "0000000000";
1126                     elsif vga_hblank = '0' and vga_vblank = '0' then
1128                       VGA_R <= "0000000000";
1130                       VGA_G <= "1111111111";
1132                       VGA_B <= "0000000000";
1134                     else
1136                       VGA_R <= "0000000000";
1138                       VGA_G <= "0000000000";
1140                       VGA_B <= "0000000000";
1142                     end if;
1144                   else
1146                     if rectangle_00 = '1' then
1148                       VGA_R <= "1111111111";
1150                       VGA_G <= "1111111111";
1152                       VGA_B <= "1111111111";
1154                       elsif ( Hcount >= HSYNC + HBACK_PORCH and Hcount < HSYNC + HBACK_PORCH + 641 and
1156                       ((Vcount >= VSYNC + VBACK_PORCH and
1158                       (Vcount < VSYNC + VBACK_PORCH + to_integer(black_b_y) + 1) or
1160                       (Vcount > VSYNC + VBACK_PORCH + 480 - to_integer(black_b_y) and
1162                       Vcount < VSYNC + VBACK_PORCH + 480)) or
1164                       ( Vcount >= VSYNC + VBACK_PORCH and Vcount < VSYNC + VBACK_PORCH + 480 and
1166                       ((Hcount >= HSYNC + HBACK_PORCH and Hcount < HSYNC + HBACK_PORCH +
1168                       to_integer(black_b_x) + 1) or
1170                       (Hcount >= HSYNC + HBACK_PORCH + 640 - to_integer(black_b_x)
1172                       and Hcount < HSYNC + HBACK_PORCH + 641))) then
1174                         VGA_R <= "0000000000";
1176                         VGA_G <= "0000000000";
1178                         VGA_B <= "0000000000";
1180                       elsif ( Hcount >= HSYNC + HBACK_PORCH and
1182                       Hcount < HSYNC + HBACK_PORCH + 641 and
1184                       ((Vcount >= VSYNC + VBACK_PORCH and
1186                       Vcount < VSYNC + VBACK_PORCH +
1188                       to_integer(border_2) + 1 - to_integer(margin)) or
1190                       (Vcount > VSYNC + VBACK_PORCH +

```

```

1120         to_integer(border_4)-1+to_integer(margin)
1121         and Vcount < VSYNC + VBACK_PORCH+480)))or
1122 ( Vcount >= VSYNC + VBACK_PORCH and
1123   Vcount < VSYNC + VBACK_PORCH + 480 and
1124   ((Hcount >= HSYNC + HBACK_PORCH and
1125     Hcount < HSYNC + HBACK_PORCH +
1126     to_integer(border_1) + 1 - to_integer(margin)))or
1127     (Hcount >= HSYNC + HBACK_PORCH+
1128     to_integer(border_3)-1+to_integer(margin)
1129     and Hcount < HSYNC + HBACK_PORCH+641)))then
1130   VGA_R <= "1111111111";
1131   VGA_G <= "1111111111";
1132   VGA_B <= "0000000000";
1133 elseif ( Hcount >= HSYNC + HBACK_PORCH and
1134   Hcount < HSYNC + HBACK_PORCH + 641 and
1135   ((Vcount >= VSYNC + VBACK_PORCH+
1136     to_integer(border_2)+ 1 - to_integer(margin)
1137     and Vcount < VSYNC + VBACK_PORCH+ to_integer(border_2)+ 1)or
1138     (Vcount > VSYNC + VBACK_PORCH +
1139     to_integer(border_4)-1 and
1140     Vcount <= VSYNC + VBACK_PORCH+
1141     to_integer(border_4)-1+to_integer(margin)))) or
1142 ( Vcount >= VSYNC + VBACK_PORCH and
1143   Vcount < VSYNC + VBACK_PORCH + 480 and
1144   ((Hcount >= HSYNC + HBACK_PORCH +
1145     to_integer(border_1) + 1 - to_integer(margin) and
1146     Hcount < HSYNC + HBACK_PORCH+
1147     to_integer(border_1) + 1)or
1148     (Hcount > HSYNC + HBACK_PORCH+
1149     to_integer(border_3)-1 and
1150     Hcount < HSYNC + HBACK_PORCH+
1151     to_integer(border_3)-1+to_integer(margin)))) then
1152   VGA_R <= "1111111111";
1153   VGA_G <= "1111111111";
1154   VGA_B <= "0000000000";
1155
1156   elsif rectangle_1 = '1' and C_color_1 /= "011" then
1157     VGA_R <= color_RGB(TO_INTEGER(C_color_1))(29 downto 20);
1158     VGA_G <= color_RGB(TO_INTEGER(C_color_1))(19 downto 10);
1159     VGA_B <= color_RGB(TO_INTEGER(C_color_1))(9 downto 0);
1160   elsif rectangle_2 = '1' and C_color_2 /= "011" then
1161     VGA_R <= color_RGB(TO_INTEGER(C_color_2))(29 downto 20);
1162     VGA_G <= color_RGB(TO_INTEGER(C_color_2))(19 downto 10);
1163     VGA_B <= color_RGB(TO_INTEGER(C_color_2))(9 downto 0);
1164   elsif rectangle_3 = '1' and C_color_3 /= "11" then
1165     VGA_R <= color_RGB(TO_INTEGER(C_color_3))(29 downto 20);
1166     VGA_G <= color_RGB(TO_INTEGER(C_color_3))(19 downto 10);
1167     VGA_B <= color_RGB(TO_INTEGER(C_color_3))(9 downto 0);
1168   elsif rectangle_4 = '1' and C_color_4 /= "011" then
1169     VGA_R <= color_RGB(TO_INTEGER(C_color_4))(29 downto 20);
1170     VGA_G <= color_RGB(TO_INTEGER(C_color_4))(19 downto 10);
1171     VGA_B <= color_RGB(TO_INTEGER(C_color_4))(9 downto 0);
1172   elsif rectangle_5 = '1' and C_color_5 /= "011" then
1173     VGA_R <= color_RGB(TO_INTEGER(C_color_5))(29 downto 20);
1174     VGA_G <= color_RGB(TO_INTEGER(C_color_5))(19 downto 10);
1175     VGA_B <= color_RGB(TO_INTEGER(C_color_5))(9 downto 0);
1176   elsif rectangle_6 = '1' and C_color_6 /= "011" then
1177     VGA_R <= color_RGB(TO_INTEGER(C_color_6))(29 downto 20);
1178     VGA_G <= color_RGB(TO_INTEGER(C_color_6))(19 downto 10);
1179     VGA_B <= color_RGB(TO_INTEGER(C_color_6))(9 downto 0);
1180   elsif rectangle_7 = '1' and C_color_7 /= "011" then
1181     VGA_R <= color_RGB(TO_INTEGER(C_color_7))(29 downto 20);
1182     VGA_G <= color_RGB(TO_INTEGER(C_color_7))(19 downto 10);
1183     VGA_B <= color_RGB(TO_INTEGER(C_color_7))(9 downto 0);
1184   elsif rectangle_11 = '1' or
1185     rectangle_12 = '1' or
1186     rectangle_13 = '1' or
1187     rectangle_14 = '1' or
1188     rectangle_15 = '1' or
1189     rectangle_16 = '1' then
1190     VGA_R <= "1111111111";
1191     VGA_G <= "1111111111";
1192     VGA_B <= "0000000000";
1193   elsif vga_hblank = '0' and vga_vblank = '0' then
1194     VGA_R <= "0000000000";
1195     VGA_G <= "1111111111";
1196     VGA_B <= "0000000000";
1197   else
1198     VGA_R <= "0000000000";
1199     VGA_G <= "0000000000";

```

```
1200         VGA_B <= "0000000000";
1201         end if;
1202     end if;
1203     end if;
1204 end process VideoOut;
1205
1206 VGA_CLK <= clk25;
1207 VGA_HS <= not vga_hsync;
1208 VGA_VS <= not vga_vsync;
1209 VGA_SYNC <= '0';
1210 VGA_BLANK <= not (vga_hsync or vga_vsync);
1211
1212 end rtl;
```



```

--
2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
-- Authors:
4  --     Abdulhamid Ghandour
--     Thomas John
6  --     Jaime Peretzman
--     Bharadwaj Vellore
8  --
-- Desc:
10 --
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
16 entity soundcontroller is
18     port (
20         clk          : in  std_logic;
21         reset_n      : in  std_logic;
22         read         : in  std_logic;
23         write        : in  std_logic;
24         chipselect   : in  std_logic;
25         address      : in  unsigned(3 downto 0);
26         readdata     : out unsigned(31 downto 0);
27         writedata    : in  unsigned(31 downto 0);
28         aud_xck      : out std_logic;
29         aud_adclrck  : out std_logic;
30         aud_adcdat   : in  std_logic;
31         aud_daclrck  : out std_logic;
32         aud_dacdat   : out std_logic;
33         aud_bclk     : inout std_logic
34     );
35 end soundcontroller;
36 architecture rtl of soundcontroller is
38     type ram_type is array(7 downto 0) of unsigned(31 downto 0);
39     signal ram_address : unsigned(2 downto 0);
40     signal counter    : unsigned(31 downto 0);
41     signal audio_clock : unsigned(1 downto 0) := "00";
42     signal audio_request : std_logic;
43     signal audio_ctrl : std_logic := '0';
44
45     component de2_wm8731_audio port (
46         clk : in std_logic;          -- Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
47         reset_n : in std_logic;
48         test_mode : in std_logic;    -- Audio CODEC controller test mode
49         audio_request : out std_logic; -- Audio controller request new data
50         data : in unsigned(15 downto 0);
51
52         -- Audio interface signals
53         AUD_ADCLRCK : out std_logic; -- Audio CODEC ADC LR Clock
54         AUD_ADCDAT  : in  std_logic; -- Audio CODEC ADC Data
55         AUD_DACLRCK : out std_logic; -- Audio CODEC DAC LR Clock
56         AUD_DACDAT  : out std_logic; -- Audio CODEC DAC Data
57         AUD_BCLK    : inout std_logic -- Audio CODEC Bit-Stream Clock
58     );
59 end component;
60
61 begin
62     ram_address <= address(2 downto 0);
63
64     audio_clk_gen : process (clk)
65     begin
66         if rising_edge(clk) then
67             audio_clock <= audio_clock + "1";
68         end if;
69     end process audio_clk_gen;
70
71     audio_host_control : process (clk)
72     begin
73         if rising_edge(clk) then
74             if reset_n = '0' then
75                 if chipselect = '1' then
76                     if read = '1' then
77

```

```

80         readdata <= RAM(to_integer(ram_address));
81         elsif write = '1' then
82             RAM(to_integer(ram_address)) <= writedata;
83         end if;
84     end if;
85
86     if audio_clock = "00" then
87         if RAM(0)(0) = '1' then
88             audio_ctrl <= '1';
89             RAM(0)(0) <= '0';
90         end if;
91
92         if audio_ctrl = '1' then
93             audio_ctrl <= '0';
94         end if;
95     end if;
96
97     end if;
98 end if;
99 end process audio_host_control;
100
101 -- audio_state_ctrl: process (clk)
102 -- begin
103 --     if rising_edge (clk) then
104 --
105 --         counter <= (others => '0');
106 --         else
107 --             if counter = 100 then
108 --                 counter <= (others => '0');
109 --                 reset_ctrl <= '0';
110 --             else
111 --                 counter <= counter + 1;
112 --             end if;
113 --         end if;
114 --     end if;
115 -- end process audio_state_ctrl;
116
117 aud_xck <= audio_clock(1);
118
119 beeper: de2_wm8731_audio port map (
120     clk => audio_clock(1),
121     reset_n => '1',
122     test_mode => audio_ctrl,           -- Output a sine wave
123     audio_request => audio_request,
124     data => "0000000000000000",
125     AUD_ADCLRCK => aud_adclrck,
126     AUD_ADCDAT => aud_adcdat,
127     AUD_DACLK => aud_daclrck,
128     AUD_DACDAT => aud_dacdat,
129     AUD_BCLK => aud_bclk
130 );
end rtl;

```

```

library ieee;
2 use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
4
entity de2_wm8731_audio is
6 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);
12
    -- Audio interface signals
14 AUD_ADCLRCK : out std_logic; -- Audio CODEC ADC LR Clock
    AUD_ADCDAT : in std_logic;  -- Audio CODEC ADC Data
16 AUD_DACLRCRCK : out std_logic; -- Audio CODEC DAC LR Clock
    AUD_DACDAT : out std_logic; -- Audio CODEC DAC Data
18 AUD_BCLK : inout std_logic -- Audio CODEC Bit-Stream Clock
);
20 end de2_wm8731_audio;
22
architecture rtl of de2_wm8731_audio is
24     signal lrck : std_logic;
    signal bclk : std_logic;
26     signal xck : std_logic;
28
    signal lrck_divider : unsigned(7 downto 0);
    signal bclk_divider : unsigned(3 downto 0);
30
    signal set_bclk : std_logic;
    signal set_lrck : std_logic;
    signal clr_bclk : std_logic;
    signal lrck_lat : std_logic;
34
    signal shift_out : unsigned(15 downto 0);
36
    signal sin_out : unsigned(7 downto 0);
    signal sin_counter : unsigned(11 downto 0);
40     signal audio_on : std_logic;
begin
42
    -- LRCK divider
44     -- Audio chip main clock is 18.432MHz / Sample rate 48KHz
    -- Divider is 18.432 MHz / 48KHz = 192 (X"C0")
46     -- Left justify mode set by I2C controller
48     process (clk)
    begin
50         if rising_edge(clk) then
            if reset_n = '0' then
52                 lrck_divider <= (others => '0');
                    elsif lrck_divider = X"BF" then -- "C0" minus 1
54                         lrck_divider <= X"00";
                    else
56                         lrck_divider <= lrck_divider + 1;
                    end if;
58         end if;
    end process;
60
    process (clk)
    begin
62         if rising_edge(clk) then
            if audio_on = '0' then
64                 audio_on <= test_mode;
            end if;
66             if sin_counter = x"36E" then
                audio_on <= '0';
68             end if;
70         end if;
    end process;
72
    process (clk)
    begin
74         if rising_edge(clk) then
            if reset_n = '0' then
76                 bclk_divider <= (others => '0');
                    elsif bclk_divider = X"B" or set_lrck = '1' then
78                         bclk_divider <= X"0";
            end if;
        end if;
    end process;
end rtl;

```

```

80     else
81         bclk_divider <= bclk_divider + 1;
82     end if;
83 end process;
84
86 set_lrck <= '1' when lrck_divider = X"BF" else '0';
87
88 process (clk)
89 begin
90     if rising_edge(clk) then
91         if reset_n = '0' then
92             lrck <= '0';
93         elsif set_lrck = '1' then
94             lrck <= not lrck;
95         end if;
96     end if;
97 end process;
98
99 -- BCLK divider
100 set_bclk <= '1' when bclk_divider(3 downto 0) = "0101" else '0';
101 clr_bclk <= '1' when bclk_divider(3 downto 0) = "1011" else '0';
102
103 process (clk)
104 begin
105     if rising_edge(clk) then
106         if reset_n = '0' then
107             bclk <= '0';
108         elsif set_lrck = '1' or clr_bclk = '1' then
109             bclk <= '0';
110         elsif set_bclk = '1' then
111             bclk <= '1';
112         end if;
113     end if;
114 end process;
115
116 -- Audio data shift output
117 process (clk)
118 begin
119     if rising_edge(clk) then
120         if reset_n = '0' then
121             shift_out <= (others => '0');
122         elsif set_lrck = '1' then
123             if audio_on = '1' then
124                 shift_out <= ("00" & sin_out & "000000");
125             else
126                 shift_out <= data;
127             end if;
128         elsif clr_bclk = '1' then
129             shift_out <= shift_out (14 downto 0) & '0';
130         end if;
131     end if;
132 end process;
133
134 -- Audio outputs
135 AUD_ADCLRCK <= lrck;
136 AUD_DACLCK <= lrck;
137 AUD_DACDAT <= shift_out(15);
138 AUD_BCLK <= bclk;
139
140 -- Self test with Sin wave
141
142 process (clk)
143 begin
144     if rising_edge(clk) then
145         if reset_n = '0' then
146             sin_counter <= (others => '0');
147         elsif lrck_lat = '1' and lrck = '0' then
148             if sin_counter = x"36E" then
149                 sin_counter <= x"000";
150             elsif audio_on = '1' then
151                 sin_counter <= sin_counter + 1;
152             end if;
153         end if;
154     end if;
155 end process;
156
157 process (clk)
158 begin

```

```

160     if rising_edge(clk) then
161         lrck_lat <= lrck;
162     end if;
163 end process;
164
165 process (clk)
166 begin
167     if rising_edge(clk) then
168         if lrck_lat = '1' and lrck = '0' then
169             audio_request <= '1';
170         else
171             audio_request <= '0';
172         end if;
173     end if;
174 end process;
175
176 with sin_counter select sin_out <=
177     x"49" when x"001",
178     x"52" when x"002",
179     x"46" when x"003",
180     x"46" when x"004",
181     x"03" when x"005",
182     x"66" when x"006",
183     x"00" when x"007",
184     x"00" when x"008",
185     x"04" when x"009",
186     x"57" when x"00A",
187     x"45" when x"00B",
188     x"56" when x"00C",
189     x"6d" when x"00D",
190     x"06" when x"00E",
191     x"20" when x"00F",
192     x"74" when x"010",
193     x"00" when x"011",
194     x"10" when x"012",
195     x"00" when x"013",
196     x"00" when x"014",
197     x"00" when x"015",
198     x"01" when x"016",
199     x"00" when x"017",
200     x"01" when x"018",
201     x"bb" when x"019",
202     x"80" when x"01A",
203     x"00" when x"01B",
204     x"00" when x"01C",
205     x"bb" when x"01D",
206     x"80" when x"01E",
207     x"00" when x"01F",
208     x"00" when x"020",
209     x"00" when x"021",
210     x"01" when x"022",
211     x"00" when x"023",
212     x"08" when x"024",
213     x"61" when x"025",
214     x"64" when x"026",
215     x"61" when x"027",
216     x"74" when x"028",
217     x"02" when x"029",
218     x"ea" when x"02A",
219     x"00" when x"02B",
220     x"00" when x"02C",
221     x"8b" when x"02D",
222     x"90" when x"02E",
223     x"7d" when x"02F",
224     x"86" when x"030",
225     x"71" when x"031",
226     x"76" when x"032",
227     x"6c" when x"033",
228     x"6b" when x"034",
229     x"75" when x"035",
230     x"6e" when x"036",
231     x"8f" when x"037",
232     x"82" when x"038",
233     x"a9" when x"039",
234     x"9e" when x"03A",
235     x"a2" when x"03B",
236     x"aa" when x"03C",
237     x"73" when x"03D",
238     x"8f" when x"03E",
239     x"3a" when x"03F",

```

240 x" 57" when x" 040" ,
 x" 1 f" when x" 041" ,
 242 x" 26" when x" 042" ,
 x" 32" when x" 043" ,
 244 x" 21" when x" 044" ,
 x" 71" when x" 045" ,
 246 x" 4e" when x" 046" ,
 x" d3" when x" 047" ,
 248 x" a0" when x" 048" ,
 x" f c" when x" 049" ,
 250 x" f7" when x" 04A" ,
 x" f d" when x" 04B" ,
 252 x" f6" when x" 04C" ,
 x" b8" when x" 04D" ,
 254 x" f3" when x" 04E" ,
 x" 26" when x" 04F" ,
 256 x" 68" when x" 050" ,
 x" 04" when x" 051" ,
 258 x" 06" when x" 052" ,
 x" 03" when x" 053" ,
 260 x" 08" when x" 054" ,
 x" 29" when x" 055" ,
 262 x" 08" when x" 056" ,
 x" 91" when x" 057" ,
 264 x" 5 f" when x" 058" ,
 x" d c" when x" 059" ,
 266 x" b b" when x" 05A" ,
 x" e7" when x" 05B" ,
 268 x" e a" when x" 05C" ,
 x" b d" when x" 05D" ,
 270 x" d7" when x" 05E" ,
 x" 94" when x" 05F" ,
 272 x" a6" when x" 060" ,
 x" 86" when x" 061" ,
 274 x" 89" when x" 062" ,
 x" 81" when x" 063" ,
 276 x" 84" when x" 064" ,
 x" 76" when x" 065" ,
 278 x" 7e" when x" 066" ,
 x" 6 c" when x" 067" ,
 280 x" 6 f" when x" 068" ,
 x" 6 f" when x" 069" ,
 282 x" 6 b" when x" 06A" ,
 x" 78" when x" 06B" ,
 284 x" 74" when x" 06C" ,
 x" 80" when x" 06D" ,
 286 x" 7e" when x" 06E" ,
 x" 84" when x" 06F" ,
 288 x" 82" when x" 070" ,
 x" 84" when x" 071" ,
 290 x" 84" when x" 072" ,
 x" 83" when x" 073" ,
 292 x" 85" when x" 074" ,
 x" 82" when x" 075" ,
 294 x" 83" when x" 076" ,
 x" 7e" when x" 077" ,
 296 x" 80" when x" 078" ,
 x" 74" when x" 079" ,
 298 x" 7a" when x" 07A" ,
 x" 66" when x" 07B" ,
 300 x" 6 d" when x" 07C" ,
 x" 5 c" when x" 07D" ,
 302 x" 61" when x" 07E" ,
 x" 5 a" when x" 07F" ,
 304 x" 5 a" when x" 080" ,
 x" 60" when x" 081" ,
 306 x" 5 c" when x" 082" ,
 x" 6 f" when x" 083" ,
 308 x" 67" when x" 084" ,
 x" 86" when x" 085" ,
 310 x" 79" when x" 086" ,
 x" 9 d" when x" 087" ,
 312 x" 92" when x" 088" ,
 x" a b" when x" 089" ,
 314 x" a6" when x" 08A" ,
 x" a d" when x" 08B" ,
 316 x" a e" when x" 08C" ,
 x" a5" when x" 08D" ,
 318 x" a a" when x" 08E" ,
 x" 94" when x" 08F" ,

320 x" 9d" when x" 090" ,
 x" 7d" when x" 091" ,
 322 x" 89" when x" 092" ,
 x" 6b" when x" 093" ,
 324 x" 73" when x" 094" ,
 x" 65" when x" 095" ,
 326 x" 65" when x" 096" ,
 x" 6c" when x" 097" ,
 328 x" 67" when x" 098" ,
 x" 79" when x" 099" ,
 330 x" 72" when x" 09A" ,
 x" 8c" when x" 09B" ,
 332 x" 82" when x" 09C" ,
 x" a0" when x" 09D" ,
 334 x" 96" when x" 09E" ,
 x" ab" when x" 09F" ,
 336 x" a8" when x" 0A0" ,
 x" a5" when x" 0A1" ,
 338 x" aa" when x" 0A2" ,
 x" 8f" when x" 0A3" ,
 340 x" 9b" when x" 0A4" ,
 x" 74" when x" 0A5" ,
 342 x" 82" when x" 0A6" ,
 x" 5a" when x" 0A7" ,
 344 x" 67" when x" 0A8" ,
 x" 3f" when x" 0A9" ,
 346 x" 4d" when x" 0AA" ,
 x" 2a" when x" 0AB" ,
 348 x" 33" when x" 0AC" ,
 x" 23" when x" 0AD" ,
 350 x" 24" when x" 0AE" ,
 x" 35" when x" 0AF" ,
 352 x" 29" when x" 0B0" ,
 x" 5d" when x" 0B1" ,
 354 x" 47" when x" 0B2" ,
 x" 90" when x" 0B3" ,
 356 x" 76" when x" 0B4" ,
 x" c5" when x" 0B5" ,
 358 x" ab" when x" 0B6" ,
 x" ea" when x" 0B7" ,
 360 x" db" when x" 0B8" ,
 x" f6" when x" 0B9" ,
 362 x" f4" when x" 0BA" ,
 x" e4" when x" 0BB" ,
 364 x" f0" when x" 0BC" ,
 x" be" when x" 0BD" ,
 366 x" d3" when x" 0BE" ,
 x" 8f" when x" 0BF" ,
 368 x" a7" when x" 0C0" ,
 x" 67" when x" 0C1" ,
 370 x" 79" when x" 0C2" ,
 x" 50" when x" 0C3" ,
 372 x" 58" when x" 0C4" ,
 x" 4b" when x" 0C5" ,
 374 x" 4c" when x" 0C6" ,
 x" 51" when x" 0C7" ,
 376 x" 4d" when x" 0C8" ,
 x" 59" when x" 0C9" ,
 378 x" 55" when x" 0CA" ,
 x" 63" when x" 0CB" ,
 380 x" 5d" when x" 0CC" ,
 x" 6f" when x" 0CD" ,
 382 x" 69" when x" 0CE" ,
 x" 79" when x" 0CF" ,
 384 x" 75" when x" 0D0" ,
 x" 7f" when x" 0D1" ,
 386 x" 7c" when x" 0D2" ,
 x" 82" when x" 0D3" ,
 388 x" 81" when x" 0D4" ,
 x" 81" when x" 0D5" ,
 390 x" 82" when x" 0D6" ,
 x" 7d" when x" 0D7" ,
 392 x" 7f" when x" 0D8" ,
 x" 7e" when x" 0D9" ,
 394 x" 7c" when x" 0DA" ,
 x" 84" when x" 0DB" ,
 396 x" 80" when x" 0DC" ,
 x" 8a" when x" 0DD" ,
 398 x" 88" when x" 0DE" ,
 x" 8c" when x" 0DF" ,

400 x" 8b" when x" 0E0" ,
 x" 8d" when x" 0E1" ,
 402 x" 8d" when x" 0E2" ,
 x" 90" when x" 0E3" ,
 404 x" 8e" when x" 0E4" ,
 x" 92" when x" 0E5" ,
 406 x" 91" when x" 0E6" ,
 x" 95" when x" 0E7" ,
 408 x" 94" when x" 0E8" ,
 x" 93" when x" 0E9" ,
 410 x" 94" when x" 0EA" ,
 x" 87" when x" 0EB" ,
 412 x" 8f" when x" 0EC" ,
 x" 75" when x" 0ED" ,
 414 x" 7e" when x" 0EE" ,
 x" 67" when x" 0EF" ,
 416 x" 6d" when x" 0F0" ,
 x" 63" when x" 0F1" ,
 418 x" 64" when x" 0F2" ,
 x" 67" when x" 0F3" ,
 420 x" 64" when x" 0F4" ,
 x" 71" when x" 0F5" ,
 422 x" 6c" when x" 0F6" ,
 x" 7c" when x" 0F7" ,
 424 x" 77" when x" 0F8" ,
 x" 84" when x" 0F9" ,
 426 x" 80" when x" 0FA" ,
 x" 88" when x" 0FB" ,
 428 x" 86" when x" 0FC" ,
 x" 8c" when x" 0FD" ,
 430 x" 89" when x" 0FE" ,
 x" 8f" when x" 0FF" ,
 432 x" 8e" when x" 100" ,
 x" 8d" when x" 101" ,
 434 x" 8f" when x" 102" ,
 x" 80" when x" 103" ,
 436 x" 88" when x" 104" ,
 x" 6f" when x" 105" ,
 438 x" 76" when x" 106" ,
 x" 65" when x" 107" ,
 440 x" 69" when x" 108" ,
 x" 67" when x" 109" ,
 442 x" 64" when x" 10A" ,
 x" 78" when x" 10B" ,
 444 x" 6e" when x" 10C" ,
 x" 90" when x" 10D" ,
 446 x" 84" when x" 10E" ,
 x" a4" when x" 10F" ,
 448 x" 9b" when x" 110" ,
 x" aa" when x" 111" ,
 450 x" a9" when x" 112" ,
 x" a0" when x" 113" ,
 452 x" a7" when x" 114" ,
 x" 8b" when x" 115" ,
 454 x" 96" when x" 116" ,
 x" 77" when x" 117" ,
 456 x" 81" when x" 118" ,
 x" 6b" when x" 119" ,
 458 x" 70" when x" 11A" ,
 x" 6a" when x" 11B" ,
 460 x" 69" when x" 11C" ,
 x" 6f" when x" 11D" ,
 462 x" 6c" when x" 11E" ,
 x" 74" when x" 11F" ,
 464 x" 72" when x" 120" ,
 x" 74" when x" 121" ,
 466 x" 75" when x" 122" ,
 x" 71" when x" 123" ,
 468 x" 73" when x" 124" ,
 x" 71" when x" 125" ,
 470 x" 70" when x" 126" ,
 x" 76" when x" 127" ,
 472 x" 73" when x" 128" ,
 x" 80" when x" 129" ,
 474 x" 7b" when x" 12A" ,
 x" 88" when x" 12B" ,
 476 x" 83" when x" 12C" ,
 x" 90" when x" 12D" ,
 478 x" 8c" when x" 12E" ,
 x" 94" when x" 12F" ,

480 x" 93" when x" 130" ,
 x" 92" when x" 131" ,
 482 x" 94" when x" 132" ,
 x" 8b" when x" 133" ,
 484 x" 90" when x" 134" ,
 x" 7e" when x" 135" ,
 486 x" 85" when x" 136" ,
 x" 6c" when x" 137" ,
 488 x" 75" when x" 138" ,
 x" 5e" when x" 139" ,
 490 x" 64" when x" 13A" ,
 x" 5e" when x" 13B" ,
 492 x" 5c" when x" 13C" ,
 x" 6d" when x" 13D" ,
 494 x" 64" when x" 13E" ,
 x" 80" when x" 13F" ,
 496 x" 77" when x" 140" ,
 x" 92" when x" 141" ,
 498 x" 8a" when x" 142" ,
 x" 9d" when x" 143" ,
 500 x" 98" when x" 144" ,
 x" 9e" when x" 145" ,
 502 x" 9f" when x" 146" ,
 x" 96" when x" 147" ,
 504 x" 9a" when x" 148" ,
 x" 8a" when x" 149" ,
 506 x" 90" when x" 14A" ,
 x" 7d" when x" 14B" ,
 508 x" 84" when x" 14C" ,
 x" 71" when x" 14D" ,
 510 x" 77" when x" 14E" ,
 x" 69" when x" 14F" ,
 512 x" 6c" when x" 150" ,
 x" 6a" when x" 151" ,
 514 x" 69" when x" 152" ,
 x" 75" when x" 153" ,
 516 x" 6f" when x" 154" ,
 x" 83" when x" 155" ,
 518 x" 7c" when x" 156" ,
 x" 8f" when x" 157" ,
 520 x" 89" when x" 158" ,
 x" 96" when x" 159" ,
 522 x" 94" when x" 15A" ,
 x" 8f" when x" 15B" ,
 524 x" 94" when x" 15C" ,
 x" 7b" when x" 15D" ,
 526 x" 86" when x" 15E" ,
 x" 62" when x" 15F" ,
 528 x" 6d" when x" 160" ,
 x" 55" when x" 161" ,
 530 x" 59" when x" 162" ,
 x" 5c" when x" 163" ,
 532 x" 56" when x" 164" ,
 x" 72" when x" 165" ,
 534 x" 65" when x" 166" ,
 x" 88" when x" 167" ,
 536 x" 7e" when x" 168" ,
 x" 91" when x" 169" ,
 538 x" 8f" when x" 16A" ,
 x" 8d" when x" 16B" ,
 540 x" 90" when x" 16C" ,
 x" 8a" when x" 16D" ,
 542 x" 8a" when x" 16E" ,
 x" 92" when x" 16F" ,
 544 x" 8c" when x" 170" ,
 x" a3" when x" 171" ,
 546 x" 9b" when x" 172" ,
 x" b0" when x" 173" ,
 548 x" ab" when x" 174" ,
 x" a9" when x" 175" ,
 550 x" af" when x" 176" ,
 x" 8c" when x" 177" ,
 552 x" 9d" when x" 178" ,
 x" 64" when x" 179" ,
 554 x" 79" when x" 17A" ,
 x" 45" when x" 17B" ,
 556 x" 52" when x" 17C" ,
 x" 43" when x" 17D" ,
 558 x" 40" when x" 17E" ,
 x" 60" when x" 17F" ,

560 x" 4 f" when x" 180" ,
 x" 8 d" when x" 181" ,
 562 x" 76" when x" 182" ,
 x" b1" when x" 183" ,
 564 x" a1" when x" 184" ,
 x" b d" when x" 185" ,
 566 x" b b" when x" 186" ,
 x" a d" when x" 187" ,
 568 x" b8" when x" 188" ,
 x" 89" when x" 189" ,
 570 x" 9c" when x" 18A" ,
 x" 67" when x" 18B" ,
 572 x" 76" when x" 18C" ,
 x" 59" when x" 18D" ,
 574 x" 5d" when x" 18E" ,
 x" 63" when x" 18F" ,
 576 x" 5c" when x" 190" ,
 x" 72" when x" 191" ,
 578 x" 6b" when x" 192" ,
 x" 76" when x" 193" ,
 580 x" 76" when x" 194" ,
 x" 70" when x" 195" ,
 582 x" 74" when x" 196" ,
 x" 67" when x" 197" ,
 584 x" 6b" when x" 198" ,
 x" 65" when x" 199" ,
 586 x" 65" when x" 19A" ,
 x" 6e" when x" 19B" ,
 588 x" 68" when x" 19C" ,
 x" 82" when x" 19D" ,
 590 x" 77" when x" 19E" ,
 x" 97" when x" 19F" ,
 592 x" 8c" when x" 1A0" ,
 x" a3" when x" 1A1" ,
 594 x" 9e" when x" 1A2" ,
 x" a2" when x" 1A3" ,
 596 x" a5" when x" 1A4" ,
 x" 97" when x" 1A5" ,
 598 x" 9d" when x" 1A6" ,
 x" 88" when x" 1A7" ,
 600 x" 8 f" when x" 1A8" ,
 x" 7e" when x" 1A9" ,
 602 x" 82" when x" 1AA" ,
 x" 7d" when x" 1AB" ,
 604 x" 7d" when x" 1AC" ,
 x" 81" when x" 1AD" ,
 606 x" 7e" when x" 1AE" ,
 x" 84" when x" 1AF" ,
 608 x" 83" when x" 1B0" ,
 x" 82" when x" 1B1" ,
 610 x" 84" when x" 1B2" ,
 x" 7e" when x" 1B3" ,
 612 x" 81" when x" 1B4" ,
 x" 7b" when x" 1B5" ,
 614 x" 7d" when x" 1B6" ,
 x" 79" when x" 1B7" ,
 616 x" 79" when x" 1B8" ,
 x" 7a" when x" 1B9" ,
 618 x" 79" when x" 1BA" ,
 x" 83" when x" 1BB" ,
 620 x" 7e" when x" 1BC" ,
 x" 95" when x" 1BD" ,
 622 x" 8b" when x" 1BE" ,
 x" a2" when x" 1BF" ,
 624 x" 9c" when x" 1C0" ,
 x" 9d" when x" 1C1" ,
 626 x" a2" when x" 1C2" ,
 x" 82" when x" 1C3" ,
 628 x" 92" when x" 1C4" ,
 x" 5b" when x" 1C5" ,
 630 x" 70" when x" 1C6" ,
 x" 3b" when x" 1C7" ,
 632 x" 48" when x" 1C8" ,
 x" 37" when x" 1C9" ,
 634 x" 34" when x" 1CA" ,
 x" 54" when x" 1CB" ,
 636 x" 43" when x" 1CC" ,
 x" 77" when x" 1CD" ,
 638 x" 68" when x" 1CE" ,
 x" 7a" when x" 1CF" ,

640 x" 7c" when x" 1D0" ,
x" 61" when x" 1D1" ,
642 x" 6e" when x" 1D2" ,
x" 56" when x" 1D3" ,
644 x" 59" when x" 1D4" ,
x" 7f" when x" 1D5" ,
646 x" 65" when x" 1D6" ,
x" d8" when x" 1D7" ,
648 x" a5" when x" 1D8" ,
x" fb" when x" 1D9" ,
650 x" f7" when x" 1DA" ,
x" f7" when x" 1DB" ,
652 x" f9" when x" 1DC" ,
x" f3" when x" 1DD" ,
654 x" fc" when x" 1DE" ,
x" 75" when x" 1DF" ,
656 x" bc" when x" 1E0" ,
x" 0a" when x" 1E1" ,
658 x" 38" when x" 1E2" ,
x" 0a" when x" 1E3" ,
660 x" 02" when x" 1E4" ,
x" 08" when x" 1E5" ,
662 x" 03" when x" 1E6" ,
x" 52" when x" 1E7" ,
664 x" 28" when x" 1E8" ,
x" 98" when x" 1E9" ,
666 x" 79" when x" 1EA" ,
x" b2" when x" 1EB" ,
668 x" ac" when x" 1EC" ,
x" 9d" when x" 1ED" ,
670 x" ab" when x" 1EE" ,
x" 75" when x" 1EF" ,
672 x" 8a" when x" 1F0" ,
x" 51" when x" 1F1" ,
674 x" 62" when x" 1F2" ,
x" 43" when x" 1F3" ,
676 x" 46" when x" 1F4" ,
x" 54" when x" 1F5" ,
678 x" 47" when x" 1F6" ,
x" 80" when x" 1F7" ,
680 x" 68" when x" 1F8" ,
x" af" when x" 1F9" ,
682 x" 9a" when x" 1FA" ,
x" c8" when x" 1FB" ,
684 x" c0" when x" 1FC" ,
x" c3" when x" 1FD" ,
686 x" c9" when x" 1FE" ,
x" a7" when x" 1FF" ,
688 x" b6" when x" 200" ,
x" 88" when x" 201" ,
690 x" 97" when x" 202" ,
x" 71" when x" 203" ,
692 x" 7b" when x" 204" ,
x" 66" when x" 205" ,
694 x" 6a" when x" 206" ,
x" 67" when x" 207" ,
696 x" 65" when x" 208" ,
x" 6b" when x" 209" ,
698 x" 69" when x" 20A" ,
x" 70" when x" 20B" ,
700 x" 6e" when x" 20C" ,
x" 74" when x" 20D" ,
702 x" 72" when x" 20E" ,
x" 76" when x" 20F" ,
704 x" 75" when x" 210" ,
x" 77" when x" 211" ,
706 x" 76" when x" 212" ,
x" 7d" when x" 213" ,
708 x" 79" when x" 214" ,
x" 87" when x" 215" ,
710 x" 82" when x" 216" ,
x" 8e" when x" 217" ,
712 x" 8b" when x" 218" ,
x" 8c" when x" 219" ,
714 x" 8e" when x" 21A" ,
x" 83" when x" 21B" ,
716 x" 88" when x" 21C" ,
x" 79" when x" 21D" ,
718 x" 7e" when x" 21E" ,
x" 72" when x" 21F" ,

720 x" 74" when x" 220" ,
x" 77" when x" 221" ,
722 x" 73" when x" 222" ,
x" 87" when x" 223" ,
724 x" 7e" when x" 224" ,
x" 95" when x" 225" ,
726 x" 8 f" when x" 226" ,
x" 96" when x" 227" ,
728 x" 98" when x" 228" ,
x" 8 a" when x" 229" ,
730 x" 91" when x" 22A" ,
x" 7 f" when x" 22B" ,
732 x" 84" when x" 22C" ,
x" 79" when x" 22D" ,
734 x" 7 c" when x" 22E" ,
x" 76" when x" 22F" ,
736 x" 78" when x" 230" ,
x" 74" when x" 231" ,
738 x" 75" when x" 232" ,
x" 73" when x" 233" ,
740 x" 73" when x" 234" ,
x" 75" when x" 235" ,
742 x" 74" when x" 236" ,
x" 7b" when x" 237" ,
744 x" 78" when x" 238" ,
x" 84" when x" 239" ,
746 x" 7 f" when x" 23A" ,
x" 90" when x" 23B" ,
748 x" 8 a" when x" 23C" ,
x" 9 a" when x" 23D" ,
750 x" 96" when x" 23E" ,
x" 9 e" when x" 23F" ,
752 x" 9d" when x" 240" ,
x" 99" when x" 241" ,
754 x" 9 c" when x" 242" ,
x" 89" when x" 243" ,
756 x" 92" when x" 244" ,
x" 73" when x" 245" ,
758 x" 7 f" when x" 246" ,
x" 60" when x" 247" ,
760 x" 68" when x" 248" ,
x" 58" when x" 249" ,
762 x" 5 a" when x" 24A" ,
x" 5 d" when x" 24B" ,
764 x" 59" when x" 24C" ,
x" 6 f" when x" 24D" ,
766 x" 64" when x" 24E" ,
x" 8 c" when x" 24F" ,
768 x" 7 c" when x" 250" ,
x" a7" when x" 251" ,
770 x" 9b" when x" 252" ,
x" a f" when x" 253" ,
772 x" a e" when x" 254" ,
x" a1" when x" 255" ,
774 x" a b" when x" 256" ,
x" 80" when x" 257" ,
776 x" 92" when x" 258" ,
x" 59" when x" 259" ,
778 x" 6 c" when x" 25A" ,
x" 3b" when x" 25B" ,
780 x" 48" when x" 25C" ,
x" 3 c" when x" 25D" ,
782 x" 37" when x" 25E" ,
x" 5 d" when x" 25F" ,
784 x" 49" when x" 260" ,
x" 8 d" when x" 261" ,
786 x" 75" when x" 262" ,
x" b9" when x" 263" ,
788 x" a5" when x" 264" ,
x" d1" when x" 265" ,
790 x" c8" when x" 266" ,
x" c d" when x" 267" ,
792 x" d3" when x" 268" ,
x" b0" when x" 269" ,
794 x" c1" when x" 26A" ,
x" 86" when x" 26B" ,
796 x" 9b" when x" 26C" ,
x" 60" when x" 26D" ,
798 x" 71" when x" 26E" ,
x" 4 c" when x" 26F" ,

800 x" 54" when x" 270" ,
 x" 4d" when x" 271" ,
 802 x" 4a" when x" 272" ,
 x" 62" when x" 273" ,
 804 x" 56" when x" 274" ,
 x" 7d" when x" 275" ,
 806 x" 70" when x" 276" ,
 x" 90" when x" 277" ,
 808 x" 89" when x" 278" ,
 x" 94" when x" 279" ,
 810 x" 94" when x" 27A" ,
 x" 8b" when x" 27B" ,
 812 x" 90" when x" 27C" ,
 x" 7d" when x" 27D" ,
 814 x" 84" when x" 27E" ,
 x" 73" when x" 27F" ,
 816 x" 77" when x" 280" ,
 x" 71" when x" 281" ,
 818 x" 71" when x" 282" ,
 x" 79" when x" 283" ,
 820 x" 74" when x" 284" ,
 x" 84" when x" 285" ,
 822 x" 7e" when x" 286" ,
 x" 8d" when x" 287" ,
 824 x" 8a" when x" 288" ,
 x" 8e" when x" 289" ,
 826 x" 8e" when x" 28A" ,
 x" 89" when x" 28B" ,
 828 x" 8c" when x" 28C" ,
 x" 82" when x" 28D" ,
 830 x" 86" when x" 28E" ,
 x" 7e" when x" 28F" ,
 832 x" 80" when x" 290" ,
 x" 80" when x" 291" ,
 834 x" 7e" when x" 292" ,
 x" 85" when x" 293" ,
 836 x" 82" when x" 294" ,
 x" 88" when x" 295" ,
 838 x" 87" when x" 296" ,
 x" 87" when x" 297" ,
 840 x" 88" when x" 298" ,
 x" 81" when x" 299" ,
 842 x" 84" when x" 29A" ,
 x" 77" when x" 29B" ,
 844 x" 7c" when x" 29C" ,
 x" 71" when x" 29D" ,
 846 x" 74" when x" 29E" ,
 x" 71" when x" 29F" ,
 848 x" 70" when x" 2A0" ,
 x" 76" when x" 2A1" ,
 850 x" 73" when x" 2A2" ,
 x" 7d" when x" 2A3" ,
 852 x" 79" when x" 2A4" ,
 x" 86" when x" 2A5" ,
 854 x" 82" when x" 2A6" ,
 x" 8b" when x" 2A7" ,
 856 x" 89" when x" 2A8" ,
 x" 89" when x" 2A9" ,
 858 x" 8b" when x" 2AA" ,
 x" 81" when x" 2AB" ,
 860 x" 86" when x" 2AC" ,
 x" 7a" when x" 2AD" ,
 862 x" 7d" when x" 2AE" ,
 x" 75" when x" 2AF" ,
 864 x" 77" when x" 2B0" ,
 x" 74" when x" 2B1" ,
 866 x" 75" when x" 2B2" ,
 x" 76" when x" 2B3" ,
 868 x" 75" when x" 2B4" ,
 x" 79" when x" 2B5" ,
 870 x" 78" when x" 2B6" ,
 x" 7b" when x" 2B7" ,
 872 x" 7a" when x" 2B8" ,
 x" 7a" when x" 2B9" ,
 874 x" 7b" when x" 2BA" ,
 x" 7a" when x" 2BB" ,
 876 x" 7a" when x" 2BC" ,
 x" 7b" when x" 2BD" ,
 878 x" 7a" when x" 2BE" ,
 x" 7f" when x" 2BF" ,

880 x" 7 c" when x" 2C0" ,
 x" 86" when x" 2C1" ,
 882 x" 82" when x" 2C2" ,
 x" 8 d" when x" 2C3" ,
 884 x" 8 a" when x" 2C4" ,
 x" 91" when x" 2C5" ,
 886 x" 8 f" when x" 2C6" ,
 x" 91" when x" 2C7" ,
 888 x" 92" when x" 2C8" ,
 x" 8 e" when x" 2C9" ,
 890 x" 90" when x" 2CA" ,
 x" 88" when x" 2CB" ,
 892 x" 8 c" when x" 2CC" ,
 x" 81" when x" 2CD" ,
 894 x" 85" when x" 2CE" ,
 x" 7 a" when x" 2CF" ,
 896 x" 7 d" when x" 2D0" ,
 x" 75" when x" 2D1" ,
 898 x" 77" when x" 2D2" ,
 x" 71" when x" 2D3" ,
 900 x" 72" when x" 2D4" ,
 x" 6 d" when x" 2D5" ,
 902 x" 6 f" when x" 2D6" ,
 x" 6 e" when x" 2D7" ,
 904 x" 6 d" when x" 2D8" ,
 x" 74" when x" 2D9" ,
 906 x" 70" when x" 2DA" ,
 x" 7 a" when x" 2DB" ,
 908 x" 77" when x" 2DC" ,
 x" 7 f" when x" 2DD" ,
 910 x" 7 d" when x" 2DE" ,
 x" 82" when x" 2DF" ,
 912 x" 81" when x" 2E0" ,
 x" 81" when x" 2E1" ,
 914 x" 82" when x" 2E2" ,
 x" 7 d" when x" 2E3" ,
 916 x" 7 f" when x" 2E4" ,
 x" 7 a" when x" 2E5" ,
 918 x" 7 b" when x" 2E6" ,
 x" 7 e" when x" 2E7" ,
 920 x" 7 b" when x" 2E8" ,
 x" 86" when x" 2E9" ,
 922 x" 81" when x" 2EA" ,
 x" 8 f" when x" 2EB" ,
 924 x" 8 b" when x" 2EC" ,
 x" 92" when x" 2ED" ,
 926 x" 91" when x" 2EE" ,
 x" 8 d" when x" 2EF" ,
 928 x" 90" when x" 2F0" ,
 x" 86" when x" 2F1" ,
 930 x" 89" when x" 2F2" ,
 x" 82" when x" 2F3" ,
 932 x" 83" when x" 2F4" ,
 x" 86" when x" 2F5" ,
 934 x" 83" when x" 2F6" ,
 x" 8 c" when x" 2F7" ,
 936 x" 89" when x" 2F8" ,
 x" 8 c" when x" 2F9" ,
 938 x" 8 d" when x" 2FA" ,
 x" 82" when x" 2FB" ,
 940 x" 88" when x" 2FC" ,
 x" 73" when x" 2FD" ,
 942 x" 7 b" when x" 2FE" ,
 x" 69" when x" 2FF" ,
 944 x" 6 d" when x" 300" ,
 x" 6 b" when x" 301" ,
 946 x" 68" when x" 302" ,
 x" 78" when x" 303" ,
 948 x" 71" when x" 304" ,
 x" 86" when x" 305" ,
 950 x" 7 f" when x" 306" ,
 x" 8 b" when x" 307" ,
 952 x" 8 a" when x" 308" ,
 x" 85" when x" 309" ,
 954 x" 8 a" when x" 30A" ,
 x" 7 b" when x" 30B" ,
 956 x" 80" when x" 30C" ,
 x" 73" when x" 30D" ,
 958 x" 76" when x" 30E" ,
 x" 73" when x" 30F" ,

960 x" 72" when x" 310" ,
 x" 7a" when x" 311" ,
 962 x" 76" when x" 312" ,
 x" 83" when x" 313" ,
 964 x" 7f" when x" 314" ,
 x" 8a" when x" 315" ,
 966 x" 87" when x" 316" ,
 x" 49" when x" 317" ,
 968 x" 4c" when x" 318" ,
 x" 54" when x" 319" ,
 970 x" 53" when x" 31A" ,
 x" 00" when x" 31B" ,
 972 x" 50" when x" 31C" ,
 x" 00" when x" 31D" ,
 974 x" 00" when x" 31E" ,
 x" 4e" when x" 31F" ,
 976 x" 49" when x" 320" ,
 x" 4f" when x" 321" ,
 978 x" 46" when x" 322" ,
 x" 43" when x" 323" ,
 980 x" 49" when x" 324" ,
 x" 44" when x" 325" ,
 982 x" 52" when x" 326" ,
 x" 00" when x" 327" ,
 984 x" 0c" when x" 328" ,
 x" 00" when x" 329" ,
 986 x" 00" when x" 32A" ,
 x" 30" when x" 32B" ,
 988 x" 32" when x" 32C" ,
 x" 38" when x" 32D" ,
 990 x" 30" when x" 32E" ,
 x" 30" when x" 32F" ,
 992 x" 2d" when x" 330" ,
 x" 2d" when x" 331" ,
 994 x" 35" when x" 332" ,
 x" 39" when x" 333" ,
 996 x" 30" when x" 334" ,
 x" 00" when x" 335" ,
 998 x" 00" when x" 336" ,
 x" 45" when x" 337" ,
 1000 x" 49" when x" 338" ,
 x" 47" when x" 339" ,
 1002 x" 4e" when x" 33A" ,
 x" 00" when x" 33B" ,
 1004 x" 11" when x" 33C" ,
 x" 00" when x" 33D" ,
 1006 x" 00" when x" 33E" ,
 x" 61" when x" 33F" ,
 1008 x" 4a" when x" 340" ,
 x" 6d" when x" 341" ,
 1010 x" 69" when x" 342" ,
 x" 20" when x" 343" ,
 1012 x" 65" when x" 344" ,
 x" 65" when x" 345" ,
 1014 x" 50" when x" 346" ,
 x" 65" when x" 347" ,
 1016 x" 72" when x" 348" ,
 x" 7a" when x" 349" ,
 1018 x" 74" when x" 34A" ,
 x" 61" when x" 34B" ,
 1020 x" 6d" when x" 34C" ,
 x" 00" when x" 34D" ,
 1022 x" 6e" when x" 34E" ,
 x" 01" when x" 34F" ,
 1024 x" 00" when x" 350" ,
 x" 53" when x" 351" ,
 1026 x" 49" when x" 352" ,
 x" 54" when x" 353" ,
 1028 x" 46" when x" 354" ,
 x" 00" when x" 355" ,
 1030 x" 16" when x" 356" ,
 x" 00" when x" 357" ,
 1032 x" 00" when x" 358" ,
 x" 6f" when x" 359" ,
 1034 x" 53" when x" 35A" ,
 x" 79" when x" 35B" ,
 1036 x" 6e" when x" 35C" ,
 x" 53" when x" 35D" ,
 1038 x" 20" when x" 35E" ,
 x" 75" when x" 35F" ,

```
1040 x"6f" when x"360",
      x"64" when x"361",
1042 x"6e" when x"362",
      x"46" when x"363",
1044 x"20" when x"364",
      x"72" when x"365",
1046 x"6f" when x"366",
      x"65" when x"367",
1048 x"67" when x"368",
      x"38" when x"369",
1050 x"20" when x"36A",
      x"30" when x"36B",
1052 x"2e" when x"36C",
      x"00" when x"36D",
1054 x"00" when x"36E",
      X"00" when others;
1056 end architecture;
```



```

--
2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
--  Authors:
4  --      Abdulhamid Ghandour
--      Thomas John
6  --      Jaime Peretzman
--      Bharadwaj Vellore
8  --
--  Desc:
10 --
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
16 entity uicontroller is
18     port (
19         clk          : in  std_logic;
20         reset_n     : in  std_logic;
21         read        : in  std_logic;
22         write       : in  std_logic;
23         chipselect  : in  std_logic;
24         address     : in  unsigned(4 downto 0);
25         readdata    : out unsigned(31 downto 0);
26         writedata   : in  unsigned(31 downto 0);
27         hex0        : out std_logic_vector(7 downto 0);
28         hex1        : out std_logic_vector(7 downto 0);
29         hex2        : out std_logic_vector(7 downto 0);
30         hex3        : out std_logic_vector(7 downto 0);
31         hex4        : out std_logic_vector(7 downto 0);
32         hex5        : out std_logic_vector(7 downto 0);
33         hex6        : out std_logic_vector(7 downto 0);
34         hex7        : out std_logic_vector(7 downto 0);
35         key         : in  std_logic_vector(3 downto 0);
36         switch     : in  unsigned(17 downto 0)
37     );
38 end uicontroller;
40 architecture rtl of uicontroller is
42     type ram_type is array(31 downto 0) of unsigned(31 downto 0);
43     signal RAM : ram_type;
44     signal ram_address : unsigned(4 downto 0);
45     signal int_key1 : std_logic;
46     signal int_key2 : std_logic;
47 begin
48     ram_address <= address;
49
50     reg_loader : process(clk)
51     begin
52         if rising_edge(clk) then
53             if reset_n = '0' then
54                 RAM(0) <= (others => '1');
55                 RAM(1) <= (others => '1');
56                 RAM(2) <= (others => '1');
57                 RAM(3) <= (others => '1');
58                 RAM(4) <= (others => '1');
59                 RAM(5) <= (others => '1');
60                 RAM(6) <= (others => '1');
61                 RAM(7) <= (others => '1');
62             else
63                 if chipselect = '1' then
64                     if write = '1' then
65                         RAM(to_integer(ram_address)) <= writedata;
66                     elsif read = '1' then
67                         readdata <= RAM(to_integer(ram_address));
68                     end if;
69                 else
70                     if RAM(8)(0) = '1' then
71                         RAM(8)(0) <= int_key1;
72                     end if;
73                     if RAM(9)(0) = '1' then
74                         RAM(9)(0) <= int_key2;
75                     end if;
76                 RAM(10)(17 downto 0) <= switch(17 downto 0);
77                 end if;
78             end if;
79         end if;
80     end process;
81 end architecture;

```

```

80 | end process reg_loader;
82 | seven_segment_driver: process(clk)
      | begin
84 |   if rising_edge(clk) then
86 |     if reset_n = '0' then
88 |       -- do nothing
90 |     else
92 |       hex0(0) <= RAM(0)(0);
94 |       hex0(1) <= RAM(0)(1);
96 |       hex0(2) <= RAM(0)(2);
98 |       hex0(3) <= RAM(0)(3);
100 |      hex0(4) <= RAM(0)(4);
102 |      hex0(5) <= RAM(0)(5);
104 |      hex0(6) <= RAM(0)(6);
106 |      hex0(7) <= RAM(0)(7);
108 |
110 |      hex1(0) <= RAM(1)(0);
112 |      hex1(1) <= RAM(1)(1);
114 |      hex1(2) <= RAM(1)(2);
116 |      hex1(3) <= RAM(1)(3);
118 |      hex1(4) <= RAM(1)(4);
120 |      hex1(5) <= RAM(1)(5);
122 |      hex1(6) <= RAM(1)(6);
124 |      hex1(7) <= RAM(1)(7);
126 |
128 |      hex2(0) <= RAM(2)(0);
130 |      hex2(1) <= RAM(2)(1);
132 |      hex2(2) <= RAM(2)(2);
134 |      hex2(3) <= RAM(2)(3);
136 |      hex2(4) <= RAM(2)(4);
138 |      hex2(5) <= RAM(2)(5);
140 |      hex2(6) <= RAM(2)(6);
142 |      hex2(7) <= RAM(2)(7);
144 |
146 |      hex3(0) <= RAM(3)(0);
148 |      hex3(1) <= RAM(3)(1);
150 |      hex3(2) <= RAM(3)(2);
152 |      hex3(3) <= RAM(3)(3);
154 |      hex3(4) <= RAM(3)(4);
156 |      hex3(5) <= RAM(3)(5);
158 |      hex3(6) <= RAM(3)(6);
158 |      hex3(7) <= RAM(3)(7);
158 |
158 |      hex4(0) <= RAM(4)(0);
158 |      hex4(1) <= RAM(4)(1);
158 |      hex4(2) <= RAM(4)(2);
158 |      hex4(3) <= RAM(4)(3);
158 |      hex4(4) <= RAM(4)(4);
158 |      hex4(5) <= RAM(4)(5);
158 |      hex4(6) <= RAM(4)(6);
158 |      hex4(7) <= RAM(4)(7);
158 |
158 |      hex5(0) <= RAM(5)(0);
158 |      hex5(1) <= RAM(5)(1);
158 |      hex5(2) <= RAM(5)(2);
158 |      hex5(3) <= RAM(5)(3);
158 |      hex5(4) <= RAM(5)(4);
158 |      hex5(5) <= RAM(5)(5);
158 |      hex5(6) <= RAM(5)(6);
158 |      hex5(7) <= RAM(5)(7);
158 |
158 |      hex6(0) <= RAM(6)(0);
158 |      hex6(1) <= RAM(6)(1);
158 |      hex6(2) <= RAM(6)(2);
158 |      hex6(3) <= RAM(6)(3);
158 |      hex6(4) <= RAM(6)(4);
158 |      hex6(5) <= RAM(6)(5);
158 |      hex6(6) <= RAM(6)(6);
158 |      hex6(7) <= RAM(6)(7);
158 |
158 |      hex7(0) <= RAM(7)(0);
158 |      hex7(1) <= RAM(7)(1);
158 |      hex7(2) <= RAM(7)(2);
158 |      hex7(3) <= RAM(7)(3);
158 |      hex7(4) <= RAM(7)(4);
158 |      hex7(5) <= RAM(7)(5);
158 |      hex7(6) <= RAM(7)(6);
158 |      hex7(7) <= RAM(7)(7);

```

```
160         int_key1 <= key(1); — key(0) is not captured.
161         int_key2 <= key(2);
162     end if;
163 end if;
164 end process seven_segment_driver;
end rtl;
```

```

2  -- DE2 (Cyclone-II) Entity for Interactive Project Game
3  -- Authors:
4  --     Abdulhamid Ghandour
5  --     Thomas John
6  --     Jaime Peretzman
7  --     Bharadwaj Vellore
8  --
9  -- Desc:
10 --
11 -- From an original by Terasic Technology, Inc.
12 -- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
13 --
14
15 library ieee;
16 use ieee.std_logic_1164.all;
17 use ieee.numeric_std.all;
18
19 entity niostop is
20
21     port (
22         -- Clocks
23
24         CLOCK_27,                -- 27 MHz
25         CLOCK_50,                -- 50 MHz
26         EXT_CLOCK : in std_logic; -- External Clock
27
28         -- Buttons and switches
29
30         KEY : in std_logic_vector(3 downto 0); -- Push buttons
31         SW  : in std_logic_vector(17 downto 0); -- DPDT switches
32
33         -- LED displays
34
35         HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
36         : out std_logic_vector(6 downto 0);           -- (active low)
37         LEDG : out std_logic_vector(8 downto 0);       -- Green LEDs (active high)
38         LEDR : out unsigned(17 downto 0);             -- Red LEDs (active high)
39
40         -- RS-232 interface
41
42         UART_TXD : out std_logic; -- UART transmitter
43         UART_RXD : in  std_logic; -- UART receiver
44
45         -- IRDA interface
46
47         --IRDA_TXD : out std_logic; -- IRDA Transmitter
48         IRDA_RXD : in  std_logic; -- IRDA Receiver
49
50         -- SDRAM
51
52         DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
53         DRAM_ADDR : out  std_logic_vector(11 downto 0); -- Address Bus
54         DRAM_LDQM, -- Low-byte Data Mask
55         DRAM_LUDQM, -- High-byte Data Mask
56         DRAM_WE_N, -- Write Enable
57         DRAM_CAS_N, -- Column Address Strobe
58         DRAM_RAS_N, -- Row Address Strobe
59         DRAM_CS_N, -- Chip Select
60         DRAM_BA_0, -- Bank Address 0
61         DRAM_BA_1, -- Bank Address 0
62         DRAM_CLK, -- Clock
63         DRAM_CKE : out std_logic; -- Clock Enable
64
65         -- FLASH
66
67         FL_DQ : inout std_logic_vector(7 downto 0); -- Data bus
68         FL_ADDR : out  std_logic_vector(21 downto 0); -- Address bus
69         FL_WE_N, -- Write Enable
70         FL_RST_N, -- Reset
71         FL_OE_N, -- Output Enable
72         FL_CE_N : out std_logic; -- Chip Enable
73
74         -- SRAM
75
76         SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
77         SRAM_ADDR : out  std_logic_vector(17 downto 0); -- Address bus 18 Bits
78         SRAM_UB_N, -- High-byte Data Mask
79         SRAM_LB_N, -- Low-byte Data Mask

```

```

80  SRAM_WEN,           -- Write Enable
81  SRAM_CEN,           -- Chip Enable
82  SRAM_OEN : out std_logic;      -- Output Enable

84  -- USB controller

86  OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
87  OTG_ADDR : out std_logic_vector(1 downto 0);   -- Address
88  OTG_CS_N,           -- Chip Select
89  OTG_RD_N,           -- Write
90  OTG_WR_N,           -- Read
91  OTG_RST_N,          -- Reset
92  OTG_FSPPEED,        -- USB Full Speed, 0 = Enable, Z = Disable
93  OTG_LSPPEED : out std_logic;  -- USB Low Speed, 0 = Enable, Z = Disable
94  OTG_INT0,           -- Interrupt 0
95  OTG_INT1,           -- Interrupt 1
96  OTG_DREQ0,          -- DMA Request 0
97  OTG_DREQ1 : in std_logic;     -- DMA Request 1
98  OTG_DACK0_N,        -- DMA Acknowledge 0
99  OTG_DACK1_N : out std_logic;  -- DMA Acknowledge 1

100 -- 16 X 2 LCD Module

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

108 -- SD card interface

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

113 -- USB JTAG link

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

118 -- I2C bus

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

121 -- PS/2 port

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

124 -- VGA output

125 VGA_CLK,           -- Clock
126 VGA_HS,            -- H_SYNC
127 VGA_VS,            -- V_SYNC
128 VGA_BLANK,         -- BLANK
129 VGASYNC : out std_logic;  -- SYNC
130 VGA_R,             -- Red[9:0]
131 VGA_G,             -- Green[9:0]
132 VGAB : out std_logic_vector(9 downto 0); -- Blue[9:0]

133 -- Ethernet Interface

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

142 -- Audio CODEC

143 AUD_ADCLRCK : inout std_logic;  -- ADC LR Clock
144 AUD_ADCDAT : in std_logic;      -- ADC Data

```

```

160     AUDDACLRCK : inout std_logic;           -- DAC LR Clock
161     AUDDACDAT  : out  std_logic;           -- DAC Data
162     AUD_BCLK   : inout std_logic;         -- Bit-Stream Clock
163     AUD_XCK    : out  std_logic;         -- Chip Clock
164
165     -- Video Decoder
166
167     TD_DATA    : in  std_logic_vector(7 downto 0); -- Data bus 8 bits
168     TD_HS     : in  std_logic;           -- H_SYNC
169     TD_VS     : in  std_logic;           -- V_SYNC
170     TD_RESET   : out  std_logic;         -- Reset
171
172     -- General-purpose I/O
173
174     GPIO_0,    -- GPIO Connection 0
175     GPIO_1    : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
176 );
177
178 end niostop;
179
180 architecture datapath of niostop is
181     signal clk25 : std_logic := '0';
182     signal reset_n : std_logic := '1';
183     signal int_sclk : std_logic;
184     signal int_sdat : std_logic;
185     signal stop_counter : std_logic := '0';
186     signal frameCount : unsigned(31 downto 0) := x"00000000";
187     signal tickCount : unsigned(31 downto 0) := x"00000000";
188     signal vision_flags_signal : std_logic_vector(7 downto 0);
189
190     component de2_i2c_av_config is
191     port (
192         iCLK : in std_logic;
193         iRST_N : in std_logic;
194         I2C_SCLK : out std_logic;
195         I2C_SDAT : inout std_logic
196     );
197     end component;
198
199 begin
200     reset_n <= KEY(0);
201
202     process (CLOCK_50)
203     begin
204         if rising_edge(CLOCK_50) then
205             clk25 <= not clk25;
206         end if;
207     end process;
208
209     niostop : entity work.pool port map (
210         clk => CLOCK_50,
211         reset_n => KEY(0),
212
213         -- the_sram
214         SRAM_ADDR_from_the_sram => SRAM_ADDR,
215         SRAM_CE_N_from_the_sram => SRAM_CE_N,
216         SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
217         SRAM_LB_N_from_the_sram => SRAM_LB_N,
218         SRAM_OE_N_from_the_sram => SRAM_OE_N,
219         SRAM_UB_N_from_the_sram => SRAM_UB_N,
220         SRAM_WE_N_from_the_sram => SRAM_WE_N,
221
222         -- the_vga
223         VGA_BLANK_from_the_vga => VGA_BLANK,
224         VGA_B_from_the_vga => VGA_B,
225         VGA_CLK_from_the_vga => VGA_CLK,
226         VGA_G_from_the_vga => VGA_G,
227         VGA_HS_from_the_vga => VGA_HS,
228         VGA_R_from_the_vga => VGA_R,
229         VGA_SYNC_from_the_vga => VGA_SYNC,
230         VGA_VS_from_the_vga => VGA_VS,
231
232         -- the_vision
233         frame_valid_to_the_vision => GPIO_1(13),
234         line_valid_to_the_vision => GPIO_1(12),
235         master_clk_from_the_vision => GPIO_1(11),
236         no_detect_from_the_vision => LEDG(0),
237         cal_direction_from_the_vision(0) => LEDR(11),
238         cal_direction_from_the_vision(1) => LEDR(12),
239         cal_direction_from_the_vision(2) => LEDR(13),

```

```

240   cal_direction_from_the_vision(3) => LEDR(14),
242   cal_direction_from_the_vision(4) => LEDR(15),
244   cal_direction_from_the_vision(5) => LEDR(16),
246   cal_direction_from_the_vision(6) => LEDR(17),
248   pixel_clk_to_the_vision => GPIO_1(10),
250   pixel_data_to_the_vision(0) => GPIO_1(0),
252   pixel_data_to_the_vision(1) => GPIO_1(1),
254   pixel_data_to_the_vision(2) => GPIO_1(5),
256   pixel_data_to_the_vision(3) => GPIO_1(3),
258   pixel_data_to_the_vision(4) => GPIO_1(2),
260   pixel_data_to_the_vision(5) => GPIO_1(4),
262   pixel_data_to_the_vision(6) => GPIO_1(6),
264   pixel_data_to_the_vision(7) => GPIO_1(7),
266   pixel_data_to_the_vision(8) => GPIO_1(8),
268   pixel_data_to_the_vision(9) => GPIO_1(9),
270   threshold_to_the_vision => SW(9 downto 0),
272   vision_flags_from_the_vision => vision_flags_signal,

274 -- the_audio
276 aud_adcdata_to_the_sounddriver => AUD_ADCDAT,
278 aud_adclck_from_the_sounddriver => AUD_ADCLRCK,
280 aud_bclk_to_and_from_the_sounddriver => AUD_BCLK,
282 aud_dacdata_from_the_sounddriver => AUD_DACDAT,
284 aud_daclck_from_the_sounddriver => AUD_DACLCK,
286 aud_xck_from_the_sounddriver => AUD_XCK,

288 -- the_lcd
290 LCD_E_from_the_lcd => LCD_EN,
292 LCD_RS_from_the_lcd => LCD_RS,
294 LCD_RW_from_the_lcd => LCD_RW,
296 LCD_data_to_and_from_the_lcd => LCD_DATA,

298 -- the_uicontrol
300 hex0_from_the_uicontrol(6 downto 0) => HEX0,
302 hex1_from_the_uicontrol(6 downto 0) => HEX1,
304 hex2_from_the_uicontrol(6 downto 0) => HEX2,
306 hex3_from_the_uicontrol(6 downto 0) => HEX3,
308 hex4_from_the_uicontrol(6 downto 0) => HEX4,
310 hex5_from_the_uicontrol(6 downto 0) => HEX5,
312 hex6_from_the_uicontrol(6 downto 0) => HEX6,
314 hex7_from_the_uicontrol(6 downto 0) => HEX7,
316 key_to_the_uicontrol => KEY,
318 switch_to_the_uicontrol => SW,

320 -- the_camera
322 sclk_from_the_camera      => int_sclk,
324 sdat_to_and_from_the_camera => int_sdat,
326 ack_to_the_camera => GPIO_1(15)
328 );

330 frame_counter: process (GPIO_1(13))
332 begin
334   if rising_edge(GPIO_1(13)) then
336     if reset_n = '0' then
338       frameCount <= x"00000000";
340     else
342       if stop_counter = '1' then
344         frameCount <= frameCount + 1;
346       end if;
348     end if;
350   end if;
352 end process;

354 tick_counter: process (clk25)
356 begin
358   if rising_edge(clk25) then
360     if reset_n = '0' then
362       stop_counter <= '0';
364       tickCount <= x"00000000";
366     else
368       if stop_counter = '1' then
370         tickCount <= tickCount + 1;
372       end if;
374     end if;
376   end if;
378 end process;

```

```

320     end if;
321     end if;
322     end process;

324     with SW(17) select
325         LEDR(9) <= frameCount(12) when '1',
326         '0' when '0',
327         'X' when others;
328     with SW(17) select
329         LEDR(8) <= frameCount(11) when '1',
330         '0' when '0',
331         'X' when others;
332     with SW(17) select
333         LEDR(7) <= frameCount(10) when '1',
334         vision_flags_signal(7) when '0',
335         'X' when others;
336     with SW(17) select
337         LEDR(6) <= frameCount(9) when '1',
338         vision_flags_signal(6) when '0',
339         'X' when others;
340     with SW(17) select
341         LEDR(5) <= frameCount(8) when '1',
342         vision_flags_signal(5) when '0',
343         'X' when others;
344     with SW(17) select
345         LEDR(4) <= frameCount(7) when '1',
346         vision_flags_signal(4) when '0',
347         'X' when others;
348     with SW(17) select
349         LEDR(3) <= frameCount(6) when '1',
350         vision_flags_signal(3) when '0',
351         'X' when others;
352     with SW(17) select
353         LEDR(2) <= frameCount(5) when '1',
354         vision_flags_signal(2) when '0',
355         'X' when others;
356     with SW(17) select
357         LEDR(1) <= frameCount(4) when '1',
358         vision_flags_signal(1) when '0',
359         'X' when others;
360     with SW(17) select
361         LEDR(0) <= frameCount(3) when '1',
362         vision_flags_signal(0) when '0',
363         'X' when others;
364
365     LEDR(0) <= vision_flags_signal(0);
366     LEDR(1) <= vision_flags_signal(1);
367     LEDR(2) <= vision_flags_signal(2);
368     LEDR(3) <= vision_flags_signal(3);
369     LEDR(4) <= vision_flags_signal(4);
370     LEDR(5) <= vision_flags_signal(5);
371     LEDR(6) <= vision_flags_signal(6);
372     LEDR(7) <= vision_flags_signal(7);
373
374     i2c : de2_i2c_av_config port map (
375         iCLK    => CLOCK_50,
376         iRST_n  => '1',
377         I2C_SCLK => I2C_SCLK,
378         I2C_SDAT => I2C_SDAT
379     );
380
381     LEDG(2) <= int_sclk;
382     LEDG(1) <= int_sdat;
383     LEDG(7) <= GPIO_1(13);
384     LEDG(6) <= GPIO_1(12);
385     GPIO_1(14) <= int_sclk;
386     GPIO_1(15) <= int_sdat;
387
388     GPIO_0(14) <= GPIO_1(14);
389     GPIO_0(15) <= GPIO_1(15);
390     GPIO_0(11) <= int_sdat;
391
392     LCD_ON    <= '1';
393     LCD_BLON <= '1';
394     FL_RST_N <= '1';
395
396     FL_ADDR(21 downto 20) <= "00";
397
398     SD_DAT3 <= '1';

```



```

400 SD_CMD <= '1';
      SD_CLK <= '1';
402
      UART_TXD <= '0';
404 DRAM_ADDR <= (others => '0');
      DRAM_LDQM <= '0';
406 DRAM_UDQM <= '0';
      DRAM_WE_N <= '1';
408 DRAM_CAS_N <= '1';
      DRAM_RAS_N <= '1';
410 DRAM_CS_N <= '1';
      DRAM_BA_0 <= '0';
412 DRAM_BA_1 <= '0';
      DRAM_CLK <= '0';
414 DRAM_CKE <= '0';
      FL_WE_N <= '1';
416
      FL_OE_N <= '1';
418 FL_CE_N <= '1';
      OTG_ADDR <= (others => '0');
420 OTG_CS_N <= '1';
      OTG_RD_N <= '1';
422 OTG_RD_N <= '1';
      OTG_WR_N <= '1';
424 OTG_RST_N <= '1';
      OTG_FSPPEED <= '1';
426 OTG_LSPPEED <= '1';
      OTG_DACK0_N <= '1';
428 OTG_DACK1_N <= '1';

430 ENET_CMD <= '0';
      ENET_CS_N <= '1';
432 ENET_WR_N <= '1';
      ENET_RD_N <= '1';
434 ENET_RST_N <= '1';
      ENET_CLK <= '0';
436
      TDO <= '0';
438 TD_RESET <= '0';

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

```