

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 are 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.

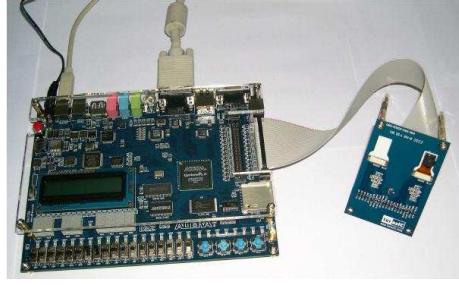


Figure 1: Board Level Connection

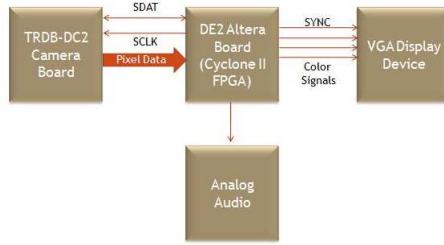


Figure 2: Block View of Physical Interfaces

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 I^2C Interface, the Vision System, SRAM, Sound Driver, VGA Controller and User Interface module.

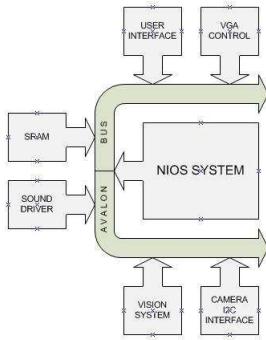


Figure 3: Block Diagram View

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

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

2.2 Hardware-Software partitioning

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

The interface to the camera is implemented as a simple piece of hardware that implements the 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, anso on.

2.3 System Configuration

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

3 Detailed Design

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

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

3.1 Camera Controller

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

3.1.1 Camera Physical Interface

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

3.1.2 Camera Register Configuration

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

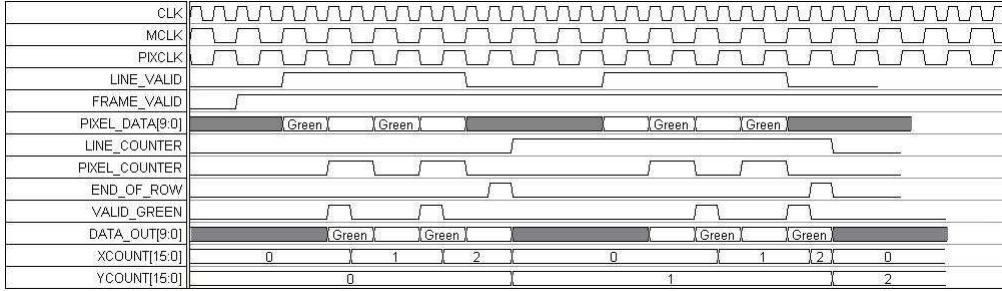


Figure 4: Pixel Processor Component Timing

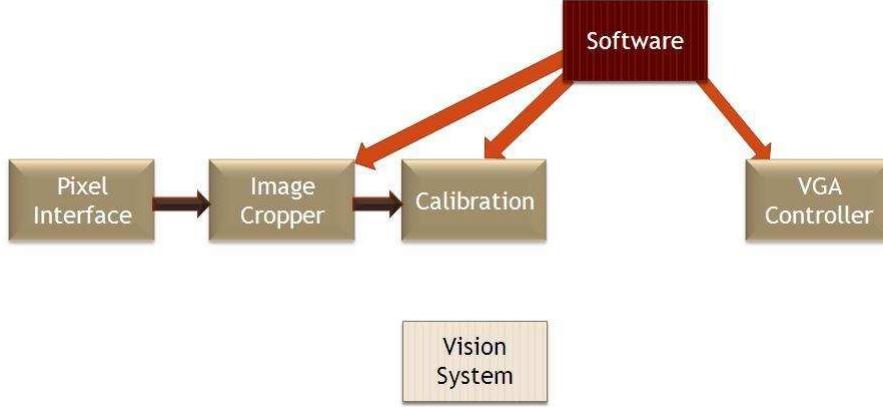


Figure 5: Calibration Mode Data Flow

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 one 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

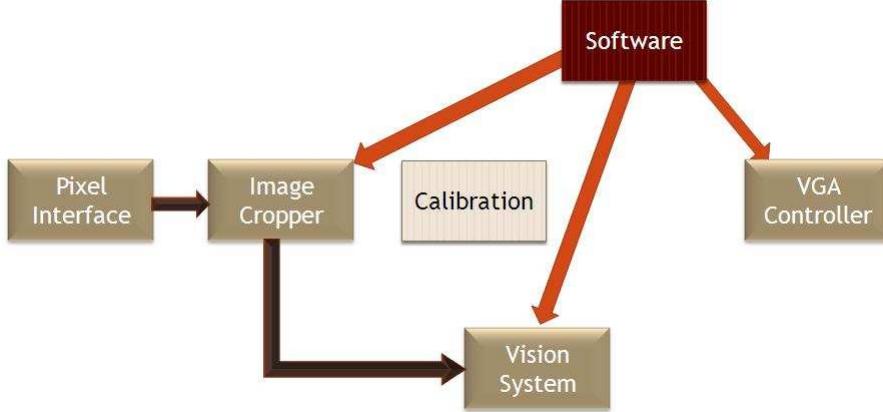


Figure 6: Gameplay Mode Data Flow

- 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

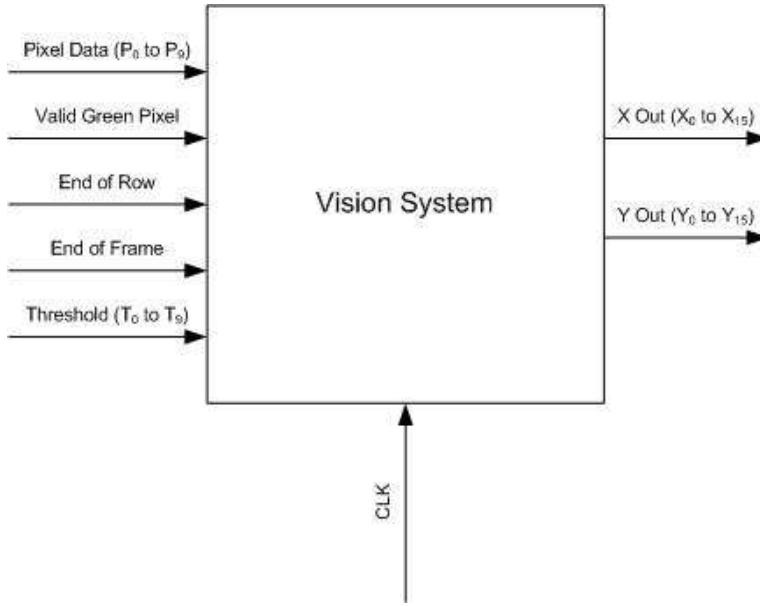


Figure 7: Vision System Block Diagram

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.

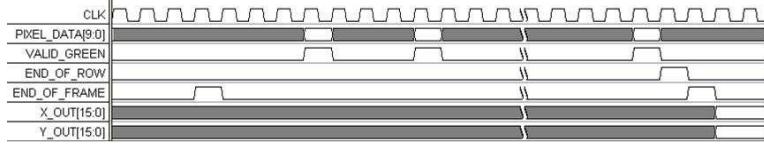


Figure 8: Vision System IO Timing Diagram

- **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.

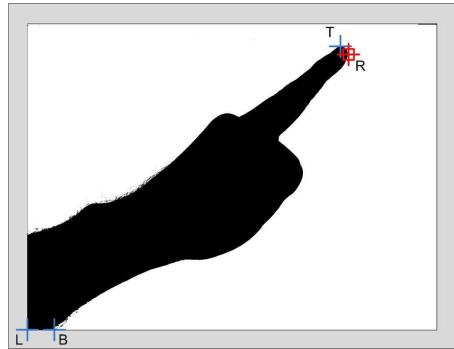


Figure 9: In this case, the top and right extremities are close to each other. Under such a condition, the right extremity is chosen as the output. This is the only case when a cue stick is used instead of a hand.

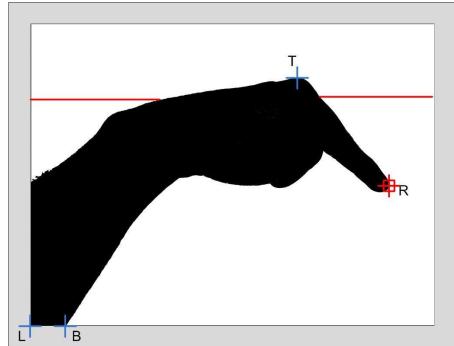


Figure 10: When the width measured a certain distance below the top extremity as shown is greater than a threshold value, it is assumed that the top extremity is not the finger tip. The right extremity is output in this case.

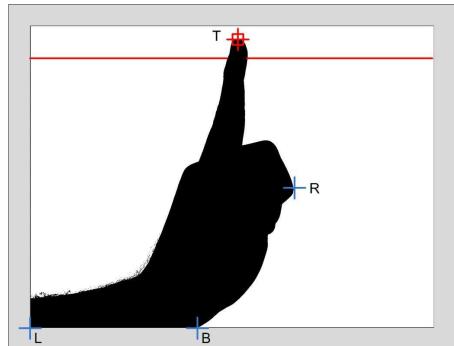


Figure 11: When the width measured a certain distance below the top extremity is lesser than the finger width threshold, the top extremity is assumed to be the finger tip.

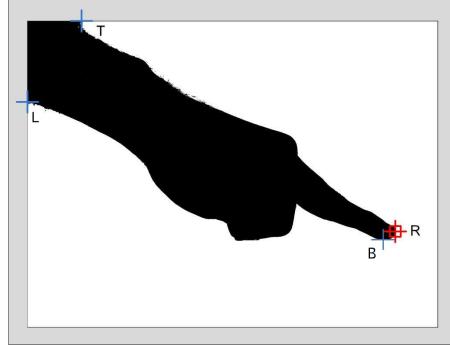
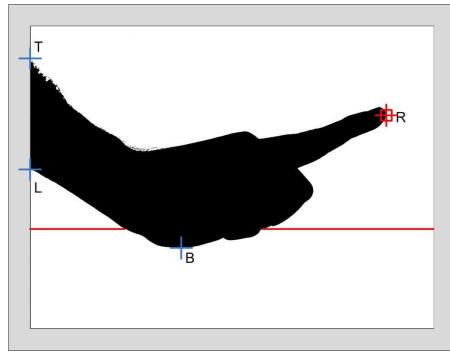


Figure 12: In this case, the bottom and right extremities are close to each other. Under such a condition, the right extremity is chosen as the output. This is the only case when a cue stick is used instead of a hand.

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

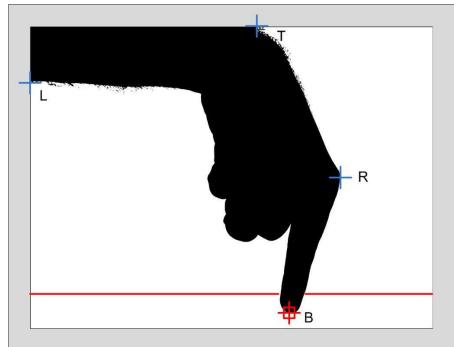


Figure 14: When the width measured a certain distance above the bottom extremity is lesser than the finger width threshold, the bottom extremity is assumed to be the finger tip.

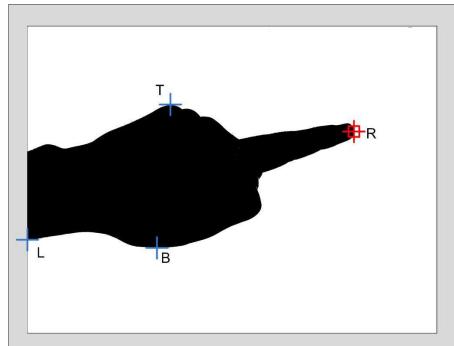


Figure 15: If the left extremity alone lies on an edge, it follows that the entry edge is the left edge. In such a case the right extremity is the tip. The opposite applies when the right extremity alone lies on an edge.

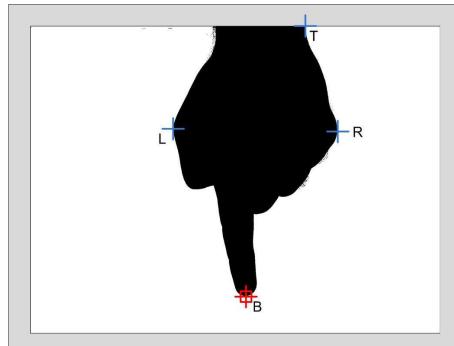


Figure 16: If the top extremity alone lies on an edge, it follows that the entry edge is the top edge. In such a case the bottom extremity is the tip. The opposite applies when the bottom extremity alone lies on an edge.

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.

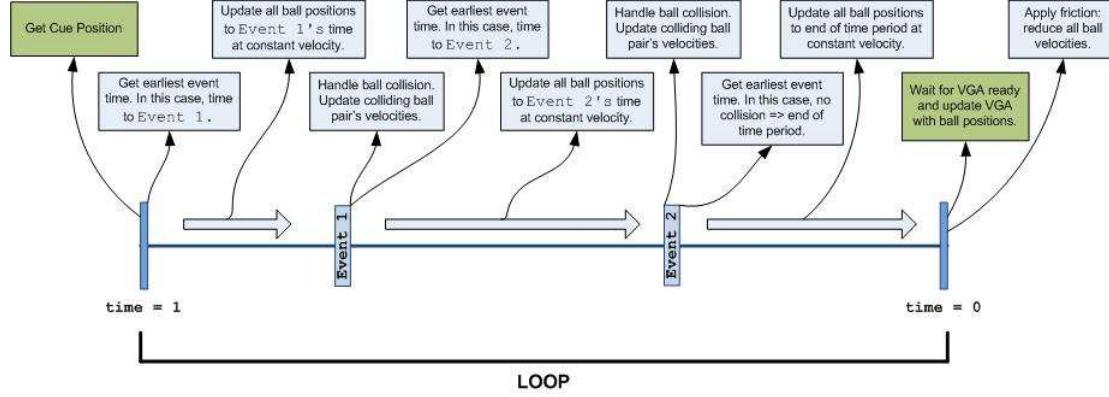


Figure 17: Collision Event Handling Loop

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 ball 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.

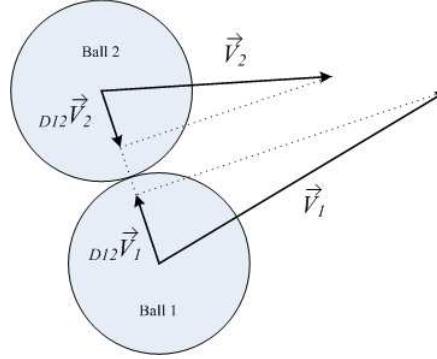


Figure 18: Ball Collisions

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 ball's velocity along the line joining the centers of the two balls, or along \vec{D}_{12} is

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

Similarly, for the second ball,

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

The velocity components for the balls along this direction are

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

The new velocities for the two balls are

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

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 values. 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 point of the pool table. Within this area that was sent by the software, the VGA will draw the table with 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>.

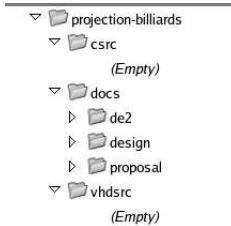


Figure 19: Directory Tree Structure

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.

```

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

#include "fixedpoint.h"

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

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

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

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

#define POCKET_RADIUS            INT2FP(10LL)

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

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

#endif /* _GAME_CONFIG_H */

```

```

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

typedef enum{
    FALSE,
    TRUE
} bool_t;

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

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

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

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

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

struct player_t{
    int points;
};

#endif /* _TYPES_H */

```

```

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

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

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

#endif /* ALT_DEBUG */

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

#endif /* ALT_DEBUG */
#endif /* _DEBUG_H */

```

```

/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#ifndef _BALL_H
#define _BALL_H
#include "types.h"
#include "system.h"
#include "io.h"
bool_t isBallMoving(const struct ball_t *ball);
long long collisionWithTableTime(const struct ball_t *ball, edge_t *edge);
long long collisionWithBallTime(const struct ball_t *ball1, const struct ball_t *ball2);
void handleBallCollision(struct ball_t *ball1, struct ball_t *ball2);
long long handleCollisionWithCue(struct ball_t *ball, const struct ball_t *cue);
void moveBalls(struct ball_t *balls, long long time);
void drawBalls(struct ball_t *balls);
void applyFriction(struct ball_t *balls);

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

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

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

#define VGA_FLAG         12
#define SPRITE_X(ball)   FP2INT((ball).pos.x - (ball).radius)
#define SPRITE_Y(ball)   FP2INT((ball).pos.y - (ball).radius)

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

```

```

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

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

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

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

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

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

158 #define READ(ack) \
158     SDAT_TRISTATE;

```

```

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

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

176 #define NACK             \
177     SDAT_SET;            \
178     HALF_CLOCK_DELAY;    \
179     SCLK_SET;            \
180     ONE_CLOCK_DELAY;    \
181     SCLK_CLR;            \
182     HALF_CLOCK_DELAY;    \
183     SDAT_CLR;

184 #define COMM_INIT         \
185     START;                \
186     SEND_B;               \
187     SEND_A;

190 #define READ_ACK(ack)    READ(ack)

192 int configureCamera();

194 #endif /* I2C_H */

```

```

/*
 * 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
#define _FIXED_POINT_H
14
#define FRAC_PRECISION      12
#define MAG_PRECISION        (64 - FRAC_PRECISION)
16
#define FPSUM(x,y)           ((x) + (y))
#define FPSUB(x,y)           ((x) - (y))
20 #define FPMUL(x,y)          (((x) * (y)) >> FRAC_PRECISION)
#define FPDIV(x,y)           (((x) << FRAC_PRECISION) / (y))
22 #define FPSQR(x)            FPMUL((x),(x))
#define FP2INT(x)             ((x) >> FRAC_PRECISION)
24 #define INT2FP(x)            ((x) << FRAC_PRECISION)

26 long long FPSQRT(long long num);
void printFP(long long fnum);
28
#endif /* _FIXED_POINT_H */

```

```

2 #ifndef _CALIBRATION_H_
3 #define _CALIBRATION_H_
4
5 #include "system.h"
6 #include "io.h"
7
8 #define NO_CUE_DETECTED      (IORD_32DIRECT(VISION_BASE, 0) >> 31)
9
10#define START_CALIBRATION    IOWR_32DIRECT(VISION_BASE, 4 * 4, 1)
11#define STOP_CALIBRATION     IOWR_32DIRECT(VISION_BASE, 4 * 4, 0)
12#define READ_REPOS_REG       IORD_32DIRECT(VISION_BASE, 3 * 4)
13#define READ_CAMERA_START    IORD_32DIRECT(VISION_BASE, 1 * 4)
14#define READ_CAMERA_END      IORD_32DIRECT(VISION_BASE, 2 * 4)
15
16#define SET_CAMERA_SANDBOX_START_X(data) \
17    IOWR_32DIRECT(VISION_BASE, 5 * 4, (data))
18#define SET_CAMERA_SANDBOX_END_X(data) \
19    IOWR_32DIRECT(VISION_BASE, 6 * 4, (data))
20#define SET_CAMERA_SANDBOX_START_Y(data) \
21    IOWR_32DIRECT(VISION_BASE, 7 * 4, (data))
22#define SET_CAMERA_SANDBOX_END_Y(data) \
23    IOWR_32DIRECT(VISION_BASE, 8 * 4, (data))
24
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
47#endif /*CALIBRATION_H_*/

```

```

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

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

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

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

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

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

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

extern unsigned long sevensegment [];

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

#endif /* _UI_H_ */

```

```

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

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

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

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

void showPoints(struct player_t *player);

int main(){
    DP_HI;
    DP("Welcome to Projection Pool\n");
    LCD_Init();
    initPointsDisplay();

    LCD_Show_Text("Welcome to Pool!");

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

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

```

```

80     DP_BYE;
81 }
82 long long calibScaleHoz , calibOffsetHoz;
83 long long calibScaleVer , calibOffsetVer ;
84 void configureScaling(){
85     long long vgaHozRes = 640;
86     long long vgaVerRes = 480;
87     long long blackMarginHoz = BLACK_MARGIN_HOZ;
88     long long blackMarginVer = BLACK_MARGIN_VER;
89
90     calibScaleHoz = FPDIV(
91         INT2FP(vgaHozRes - 2*blackMarginHoz) ,
92         INT2FP(camEndX - camStartX)
93     );
94     calibOffsetHoz =
95         FPMUL(calibScaleHoz ,INT2FP(CAMERA_CROP_MARGIN_HOZ)) +
96         INT2FP(blackMarginHoz);
97
98     calibScaleVer = FPDIV(
99         INT2FP(vgaVerRes - 2*blackMarginVer) ,
100        INT2FP(camEndY - camStartY)
101    );
102    calibOffsetVer =
103        FPMUL(calibScaleVer ,INT2FP(CAMERA_CROP_MARGIN_VER)) +
104        INT2FP(blackMarginVer);
105 }

106 int play(){
107     int i,j;
108     long long time , earliestEventTime , eventTime;
109     event_t eventType;
110     struct ball_t *collidingBall1 , *collidingBall2;
111     edge_t tableEdge , collisionTableEdge;
112     bool_t ballsMoved = FALSE;
113     bool_t repositionedWhite;
114
115     long long xin = 0, prevXin = 0;
116     long long yin = 0, prevYin = 0;
117     long long xinBeforeFilter = 0, yinBeforeFilter = 0;
118     long long A = FPDIV(INT2FP(18),INT2FP(10));
119     long long W0, W1, W2, W3;
120
121     W0 = FPDIV(INT2FP(5),INT2FP(10));
122     W1 = FPDIV(INT2FP(3),INT2FP(10));
123     W2 = FPDIV(INT2FP(1),INT2FP(10));
124     W3 = FPDIV(INT2FP(1),INT2FP(10));
125
126     struct vector prevCuePos1 , prevCuePos2 , prevCuePos3;
127     prevCuePos1.x = 0;
128     prevCuePos1.y = 0;
129     prevCuePos2.x = 0;
130     prevCuePos2.y = 0;
131     prevCuePos3.x = 0;
132     prevCuePos3.y = 0;
133
134     long long prevXinBeforeFilter = 0, prevYinBeforeFilter = 0;
135     long long absPreFilterXDiff , absPreFilterYDiff;
136
137     int numBallsPocketed = 0;
138     gameState = GAME_WAITING_TO_PLAY;
139
140     configureScaling();
141     LCD_Init();
142     LCD_ShowText("Player");
143     LCD_Line2();
144     LCD_ShowText("1");
145
146     while(1){
147         time = INT2FP(1LL);
148         if (!NO_CUE_DETECTED){
149             xin = (long long)IORD_32DIRECT(VISION_BASE, 0);
150             yin = ((xin >> 16) & 0x7FFF);
151             xin = (xin & 0xFFFF);
152             xinBeforeFilter = FPMUL(calibScaleHoz ,INT2FP(xin)) + calibOffsetHoz;
153             yinBeforeFilter = FPMUL(calibScaleVer ,INT2FP(yin)) + calibOffsetVer;
154
155             /*
156             * Filter to limit step changes in cue position
157             */

```

```

160     absPreFilterXDiff = (xinBeforeFilter > prevXinBeforeFilter)?
161         (xinBeforeFilter - prevXinBeforeFilter):
162         (prevXinBeforeFilter > xinBeforeFilter);
164     absPreFilterYDiff = (yinBeforeFilter > prevYinBeforeFilter)?
165         (yinBeforeFilter - prevYinBeforeFilter):
166         (prevYinBeforeFilter > yinBeforeFilter);
167
168     if((prevXinBeforeFilter != 0) && (prevXinBeforeFilter !=0)){
169         if((absPreFilterXDiff > INT2FP(75)) || (absPreFilterYDiff > INT2FP(75))){
170             xinBeforeFilter = prevXinBeforeFilter;
171             yinBeforeFilter = prevYinBeforeFilter;
172         } else{
173             prevXinBeforeFilter = xinBeforeFilter;
174             prevYinBeforeFilter = yinBeforeFilter;
175         }
176     } else{
177         prevXinBeforeFilter = xinBeforeFilter;
178         prevYinBeforeFilter = yinBeforeFilter;
179     }
180
181     /*
182      * Smoothing filter for cue position
183      */
184     xin = FPMUL(W0, xinBeforeFilter) +
185         FPMUL(W1, prevCuePos1.x) +
186         FPMUL(W2, prevCuePos2.x) +
187         FPMUL(W3, prevCuePos3.x);
188     yin = FPMUL(W0, yinBeforeFilter) +
189         FPMUL(W1, prevCuePos1.y) +
190         FPMUL(W2, prevCuePos2.y) +
191         FPMUL(W3, prevCuePos3.y);
192
193     prevCuePos3.x = prevCuePos2.x;
194     prevCuePos3.y = prevCuePos2.y;
195     prevCuePos2.x = prevCuePos1.x;
196     prevCuePos2.y = prevCuePos1.y;
197     prevCuePos1.x = xinBeforeFilter;
198     prevCuePos1.y = yinBeforeFilter;
199
200     cue.ballState = BALL_VISIBLE;
201
202     if ((prevXin != 0) && (prevYin != 0)){
203         cue.pos.x = prevXin;
204         cue.pos.y = prevYin;
205         if(IORD_32DIRECT(UICONTROL_BASE, 10 * 4) & 0x000000400){
206             IOWR_POS(VGA_BASE, 13, FP2INT(prevXin));
207             IOWR_POS(VGA_BASE, 14, FP2INT(prevYin));
208         } else{
209             IOWR_POS(VGA_BASE, 13, 999);
210             IOWR_POS(VGA_BASE, 14, 999);
211         }
212         cue.vel.x = FPMUL(A,(xin - prevXin));
213         cue.vel.y = FPMUL(A,(yin - prevYin));
214         prevXin = xin;
215         prevYin = yin;
216     } else{
217         prevXin = xin;
218         prevYin = yin;
219     }
220 } else{
221     cue.ballState = BALL_INVISIBLE;
222     prevXin = 0;
223     prevYin = 0;
224     prevXinBeforeFilter = 0;
225     prevYinBeforeFilter = 0;
226 }
227
228     ballsMoved = FALSE;
229     do{
230         earliestEventTime = time;
231         eventType = NO_COLLISION;
232         collidingBall1 = NULL;
233         collidingBall2 = NULL;
234         tableEdge = NONE;
235
236         for(i=0; i<NUM_BALLS; i++){
237             if(BALL_INVISIBLE == balls[i].ballState){
238                 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(UICONTROLBASE, 10 * 4) & 0x1000) == 0)
265     ){
266         if(((( balls [ i ].vel.x == 0) && ( balls [ i ].vel.y == 0)) ||
267             ((IORD_32DIRECT(UICONTROLBASE, 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 moveBalls( balls , earliestEventTime );
315
316 switch(eventType){
317     case NO_COLLISION:
318         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=%lld,%lld\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(200);
363             }
364             if(collidingBall1->pos.y > (TABLE_END_Y - INT2FP(20))){
365                 collidingBall1->pos.y -= INT2FP(150);
366             }
367             printf("New_Pos=%lld,%lld\n",
368                   FP2INT(collidingBall1->pos.x),
369                   FP2INT(collidingBall1->pos.y));
370             repositionedWhite = TRUE;
371         } else{
372             break;
373         }
374     }
375 }
376 break;
377
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
390         case TOP:
391         case BOTTOM:
392             collidingBall1->vel.y *= -1LL;
393             break;
394
395         default:
396             DP_ASSERT(0,"Collision with non-existent table edge!");
397             break;
398     }

```

```

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 * Check if there have been balls moved in the last time step
419 * If yes, and if the previous state was WAITING_TO_PLAY, then
420 * switch to PLAYING_STATE. Continue with the same player.
421 * If not, and if the previous state was PLAYING, then switch
422 * to the WAITING_TO_PLAY. Also switch players.
423 */
424 if((GAME_PLAYING == gameState) && (FALSE == ballsMoved)){
425     gameState = GAME_WAITING_TO_PLAY;
426     if(pCurrentPlayer == &player1){
427         pCurrentPlayer = &player2;
428         LCD_Line2();
429         LCD_Show_Text("2");
430     }else if(pCurrentPlayer == &player2){
431         pCurrentPlayer = &player1;
432         LCD_Line2();
433         LCD_Show_Text("1");
434     }else{
435         fflush(stdout);
436         DP_ASSERT(0, "Invalid player\n");
437     }
438 }else if((GAME_WAITING_TO_PLAY == gameState) && (TRUE == ballsMoved)){
439     gameState = GAME_PLAYING;
440 }else{
441     /*
442      * Do nothing
443      */
444 }
445

446 /*
447 * Check if there has been a request for re-calibration
448 * and return -1 if yes
449 */
450 if(CALIBRATION_REQUESTED == 0){
451     if(numBallsPocketed == NUM_BALLS - 1){
452         initBalls();
453         initPlayers();
454         initCue();
455         initPointsDisplay();
456     }
457     return -1;
458 }
459

460 if(NEW_GAME_REQUESTED == 0){
461     return 0;
462 }

463 if(numBallsPocketed == NUM_BALLS - 1){
464     LCD_Init();
465     LCD_Show_Text("Player");
466     if(player1.points > player2.points){
467         LCD_Show_Text("1 WINS!");
468     }else if(player1.points < player2.points){
469         LCD_Show_Text("2 WINS!");
470     }else{
471         LCD_Show_Text("TIE!");
472     }
473     while(NEW_GAME_REQUESTED == 1);
474     return 0;
475 }
476 }
477 }
478 }

```

```

480
481     static void initPockets(){
482         int i;
483
484         for(i=0; i<6; i++){
485             pockets[i].vel.x = 0;
486             pockets[i].vel.y = 0;
487             pockets[i].radius = POCKET_RADIUS;
488             pockets[i].colour = 1;
489             pockets[i].ballState = BALL_INVISIBLE;
490         }
491
492         pockets[0].pos.x = TOP_LEFT_POCKET_X;
493         pockets[0].pos.y = TOP_LEFT_POCKET_Y;
494         pockets[1].pos.x = TOP_MID_POCKET_X;
495         pockets[1].pos.y = TOP_MID_POCKET_Y;
496         pockets[2].pos.x = TOP_RIGHT_POCKET_X;
497         pockets[2].pos.y = TOP_RIGHT_POCKET_Y;
498         pockets[3].pos.x = BOTTOM_LEFT_POCKET_X;
499         pockets[3].pos.y = BOTTOM_LEFT_POCKET_Y;
500         pockets[4].pos.x = BOTTOM_MID_POCKET_X;
501         pockets[4].pos.y = BOTTOM_MID_POCKET_Y;
502         pockets[5].pos.x = BOTTOM_RIGHT_POCKET_X;
503         pockets[5].pos.y = BOTTOM_RIGHT_POCKET_Y;
504     }
505
506     static void initBalls(){
507         int i;
508
509         for(i=0; i<NUM_BALLS; i++){
510             balls[i].radius = BALL_RADIUS;
511             balls[i].ballState = BALL_VISIBLE;
512         }
513         balls[0].colour = COL_WHITE;
514         balls[0].points = -10;
515
516         balls[1].colour = COL_YELLOW;
517         balls[1].points = 20;
518
519         balls[2].colour = COL_CYAN;
520         balls[2].points = 5;
521
522         balls[3].colour = COL_K3;
523         balls[3].points = 10;
524
525         balls[4].colour = COL_K3;
526         balls[4].points = 10;
527
528         balls[5].colour = COL_CYAN;
529         balls[5].points = 5;
530
531         balls[6].colour = COL_CYAN;
532         balls[6].points = 5;
533
534         balls[0].pos.x = INT2FP(tableStartX + 240);
535         balls[0].pos.y = INT2FP(tableStartY + 120);
536         balls[0].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
537         balls[0].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
538
539         balls[1].pos.x = INT2FP(tableStartX + 40);
540         balls[1].pos.y = INT2FP(tableStartY + 120);
541         balls[1].vel.x = FPDIV(INT2FP(0LL),INT2FP(2LL));
542         balls[1].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
543
544         balls[2].pos.x = INT2FP(tableStartX + 40);
545         balls[2].pos.y = INT2FP(tableStartY + 160);
546         balls[2].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
547         balls[2].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
548
549         balls[3].pos.x = INT2FP(tableStartX + 100);
550         balls[3].pos.y = INT2FP(tableStartY + 100);
551         balls[3].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
552         balls[3].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
553
554         balls[4].pos.x = INT2FP(tableStartX + 100);
555         balls[4].pos.y = INT2FP(tableStartY + 140);
556         balls[4].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
557         balls[4].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
558
559         balls[5].pos.x = INT2FP(tableStartX + 150);

```

```

560     balls[5].pos.y = INT2FP(tableStartY + 120);
561     balls[5].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
562     balls[5].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
563
564     balls[6].pos.x = INT2FP(tableStartX + 40);
565     balls[6].pos.y = INT2FP(tableStartY + 80);
566     balls[6].vel.x = FPDIV(INT2FP(0LL),INT2FP(1LL));
567     balls[6].vel.y = FPDIV(INT2FP(0LL),INT2FP(1LL));
568 }
569
570 static void initCue(){
571     cue.colour = 0LL;
572     cue.pos.x = INT2FP(10LL);
573     cue.pos.y = INT2FP(10LL);
574     cue.radius = INT2FP(2LL);
575     cue.vel.x = 0LL;
576     cue.vel.y = 0LL;
577 }
578
579 void initPlayers(){
580     player1.points = 0;
581     player2.points = 0;
582     pCurrentPlayer = &player1;
583 }
584
585 void showPoints(struct player_t *player){
586     if(player == &player1){
587         if(player->points < 0){
588             HEX5(sevensegment[0]);
589             HEX4(sevensegment[0]);
590         }else{
591             HEX5(sevensegment[player->points/10]);
592             HEX4(sevensegment[player->points%10]);
593         }
594     }else if(player == &player2){
595         if(player->points < 0){
596             HEX1(sevensegment[0]);
597             HEX0(sevensegment[0]);
598         }else{
599             HEX1(sevensegment[player->points/10]);
600             HEX0(sevensegment[player->points%10]);
601         }
602     }else{
603         DP_ASSERT(0,"Invalid player");
604     }
605 }

```

```

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

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

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

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

    SET_GREEN_ROW_THR;
    SET_GREEN_COL_THR;

    blackMarginSize = BLACK_MARGIN_CONFIG;
    BLACK_MARGIN_SET(blackMarginSize);

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

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

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

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

    usleep(200000);

    START_CALIBRATION;

    /*
     * Check reposition register and direct user to move camera
     * We use a counter here to make sure we check several times
     * to confirm that calibration is indeed complete and the
     * state is steady.
     */
    while(!calibrated){
        repos = READ_REPOS_REG;
        if(repos == 0){
            counter++;
            if(counter < 20){
                usleep(200000);
                printf("Calibrating .. Please wait .... \n");
            }
        }
    }
}

```

```

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

```

```

160 }
161 while((IORD_16DIRECT(VGA_BASE, 12*2) & 0x0001) ==1);
162   for (;j <=500;){  

163     j++;  

164   }  

165  

166 void wait_fn2(){  

167   int i=0;  

168   for (; i <=150001;){  

169     i++;  

170   }  

171   while((IORD_16DIRECT(VGA_BASE, 20*2) & 0x0001) ==1);  

172 }  

173 /* void wait_fn2(){  

174   int i=0;  

175   int j=0;  

176   for (;i <=50001;){  

177     i++;  

178   }  

179   while((IORD_16DIRECT(VGA_BASE, 20*2) & 0x0001) ==1);  

180   for (;j <=500;){  

181     j++;  

182   }  

183 }*/  

184  

185 void wait_fn3(){  

186   int i=0;  

187   for (; i <=5000001;){  

188     i++;  

189   }  

190 }  

191  

192 void wait_fn4(){  

193   int i=0;  

194   for (; i <=4001;){  

195     i++;  

196   }  

197 }  

198  

199 void wait_fn5(){  

200   int i=0;  

201   for (; i <=600001;){  

202     i++;  

203   }  

204 }  

205  

206 int calibrate(){  

207   int cross_H = 150;  

208   int cross_V = 150;  

209   int x1 = 0;  

210   int x2 = 320;  

211   int y1 = 140;  

212   int y2 = 180;  

213   int temp_x1=0, temp_x2=0, temp_y1=0,temp_y2=0;  

214   int cal_flag=0;  

215   int read_in=0;  

216   int delta_stick = 4; // Defines the stepsize in the calibration sticks  

217   int border_margin = 15;  

218  

219   int xin , yin ;  

220  

221   while (1) {  

222     wait_fn2();  

223     //printf("H=%d, V=%d\n",cross_H,cross_V);  

224     IOWR_POS(VGA_BASE, 16 , x1 );  

225     IOWR_POS(VGA_BASE, 18 , x2 );  

226     IOWR_POS(VGA_BASE, 17 , y1 );  

227     IOWR_POS(VGA_BASE, 19 , y2 );  

228     IOWR_POS(VGA_BASE, 21 , 0 );  

229     IOWR_POS(VGA_BASE, 13 , cross_H );  

230     IOWR_POS(VGA_BASE, 14 , cross_V );  

231     wait_fn4 ();  

232     xin = IORD_32DIRECT(VISION_BASE, 0 );  

233     yin = (xin >> 16) & 0x00007FFF;  

234     xin = xin & 0x0000FFFF;  

235  

236     cross_H = xin;  

237     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)
249     {
250         wait_fn3();
251         temp_x1 = x2 + delta_stick;
252         x2=639;
253         x1=320;
254         y1=140;
255         y2=180;
256         cal_flag=1;
257         read_in=0;
258     }
259     if (cal_flag==1 && read_in>2)
260         x1 = x1 + delta_stick;
261     if(NO_CUE_DETECTED && cal_flag==1 && read_in>2)
262     {
263         wait_fn3();
264         temp_x2 = x1 - delta_stick;
265         x1= 320;
266         x2= 380;
267         y1=0;
268         y2=240;
269         cal_flag=2;
270         read_in=0;
271     }
272     if (cal_flag==2 && read_in>2)
273         y2 = y2 - delta_stick;
274     if(NO_CUE_DETECTED && cal_flag==2 && read_in>2)
275     {
276         wait_fn3();
277         temp_y1 = y2 + delta_stick;
278         x1=120;
279         x2=180;
280         y1=240;
281         y2=480;
282         cal_flag=3;
283         read_in=0;
284     }
285     if (cal_flag==3 && read_in>2)
286     {
287         y1 = y1 + delta_stick;
288     }
289     if(NO_CUE_DETECTED && cal_flag==3 && read_in>2)
290     {
291         temp_y2 = y1 - delta_stick;
292         tableStartX = (long long)temp_x1 + border_margin;
293         tableEndX = (long long)temp_x2 - border_margin;
294         tableStartY = (long long)temp_y1 + border_margin;
295         tableEndY = (long long)temp_y2 - border_margin;
296         IOWR_POS(VGA_BASE, 16 ,tableStartX );
297         IOWR_POS(VGA_BASE, 18 ,tableEndX );
298         IOWR_POS(VGA_BASE, 17 , tableStartY );
299         IOWR_POS(VGA_BASE, 19 , tableEndY );
300         IOWR_POS(VGA_BASE, 21 , 1);
301         break;
302     }
303 }
304 return 0;
}

```

```

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

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

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

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

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

#include "i2c.h"

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

    SDAT_SET;
    SCLK_SET;
    DELAY(10000);

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

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

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

    START;
    SEND_B;
    SEND_B;
    READ_ACK(ack6);
}

```

```

80    READ( bit8 );
81    READ( bit9 );
82    READ( bit10 );
83    READ( bit11 );
84    READ( bit12 );
85    READ( bit13 );
86    READ( bit14 );
87    READ( bit15 );
88    NACK;
89
90    STOP;
91
92    printf("Ack_1: %d\n", ack1);
93    printf("Ack_2: %d\n", ack2);
94    printf("Ack_3: %d\n", ack3);
95    printf("Ack_4: %d\n", ack4);
96    printf("Ack_5: %d\n", ack5);
97    printf("Ack_6: %d\n", ack6);
98
99    version = (bit15 << 0) + (bit14 << 1) + (bit13 << 2) + (bit12 << 3) +
100      (bit11 << 4) + (bit10 << 5) + (bit9 << 6) + (bit8 << 7) +
101      (bit7 << 8) + (bit6 << 9) + (bit5 << 10) + (bit4 << 11) +
102      (bit3 << 12) + (bit2 << 13) + (bit1 << 14) + (bit0 << 15);
103
104   printf("Version == 0x%0x\n", version);
105
106  /*
107   * Write the exposure setting
108   */
109  COMM_INIT;      READ_ACK(ack1);
110  SEND_0; SEND_9; READ_ACK(ack2);
111  SEND_0; SEND_2; READ_ACK(ack3);
112
113  COMM_INIT;      READ_ACK(ack4);
114  SEND_F; SEND_1; READ_ACK(ack5);
115  SEND_2; SEND_A; READ_ACK(ack6);
116  STOP;
117
118  DELAY(10000);
119
120  /*
121   * Write the row start
122   */
123  COMM_INIT;      READ_ACK(ack1);
124  SEND_0; SEND_1; READ_ACK(ack2);
125  SEND_0; SEND_0; READ_ACK(ack3);
126
127  COMM_INIT;      READ_ACK(ack4);
128  SEND_F; SEND_1; READ_ACK(ack5);
129  SEND_D; SEND_5; READ_ACK(ack6);
130  STOP;
131
132  DELAY(10000);
133
134  /*
135   * Write the column start
136   */
137  COMM_INIT;      READ_ACK(ack1);
138  SEND_0; SEND_2; READ_ACK(ack2);
139  SEND_0; SEND_1; READ_ACK(ack3);
140
141  COMM_INIT;      READ_ACK(ack4);
142  SEND_F; SEND_1; READ_ACK(ack5);
143  SEND_4; SEND_0; READ_ACK(ack6);
144  STOP;
145
146  DELAY(10000);
147
148  /*
149   * Write the row width
150   */
151  COMM_INIT;      READ_ACK(ack1);
152  SEND_0; SEND_3; READ_ACK(ack2);
153  SEND_0; SEND_1; READ_ACK(ack3);
154
155  COMM_INIT;      READ_ACK(ack4);
156  SEND_F; SEND_1; READ_ACK(ack5);
157  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;

174    DELAY(10000);

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;

188    DELAY(10000);

189    /*
190     * Write the row speed
191     */
192    COMM_INIT;      READ_ACK(ack1);
193    SEND_0; SEND_A; READ_ACK(ack2);
194    SEND_0; SEND_0; READ_ACK(ack3);
195
196    COMM_INIT;      READ_ACK(ack4);
197    SEND_F; SEND_1; READ_ACK(ack5);
198    SEND_1; SEND_1; READ_ACK(ack6);
199    STOP;

202    DELAY(10000);

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;

216    DELAY(10000);

217    /*
218     * Write the horizontal blanking for mode A
219     */
220    COMM_INIT;      READ_ACK(ack1);
221    SEND_0; SEND_7; READ_ACK(ack2);
222    SEND_0; SEND_0; READ_ACK(ack3);
223
224    COMM_INIT;      READ_ACK(ack4);
225    SEND_F; SEND_1; READ_ACK(ack5);
226    SEND_8; SEND_8; READ_ACK(ack6);
227    STOP;

230    DELAY(10000);

231    /*
232     * Write the context control
233     */
234    // COMM_INIT;      READ_ACK(ack1);
235    // SEND_C; SEND_8; READ_ACK(ack2);
236    // SEND_0; SEND_0; READ_ACK(ack3);
237    // COMM_INIT;      READ_ACK(ack4);

```

```

240 // SEND_F; SEND_I; READ_ACK(ack5);
241 // SEND_O; SEND_B; READ_ACK(ack6);
242 // STOP;

244 DELAY(10000);
245 COMM_INIT;
246 READ_ACK(ack1);
247 SEND_O;
248 SEND_I;
249 READ_ACK(ack2);

250 START;
251 SEND_B;
252 SEND_B;
253 READ_ACK(ack3);
254 READ(bit0);
255 READ(bit1);
256 READ(bit2);
257 READ(bit3);
258 READ(bit4);
259 READ(bit5);
260 READ(bit6);
261 READ(bit7);
262 NACK;

264 COMM_INIT;
265 READ_ACK(ack4);
266 SEND_F;
267 SEND_I;
268 READ_ACK(ack5);

270 START;
271 SEND_B;
272 SEND_B;
273 READ_ACK(ack6);
274 READ(bit8);
275 READ(bit9);
276 READ(bit10);
277 READ(bit11);
278 READ(bit12);
279 READ(bit13);
280 READ(bit14);
281 READ(bit15);
282 NACK;

284 STOP;

286 printf("Ack_1:%d\n", ack1);
287 printf("Ack_2:%d\n", ack2);
288 printf("Ack_3:%d\n", ack3);
289 printf("Ack_4:%d\n", ack4);
290 printf("Ack_5:%d\n", ack5);
291 printf("Ack_6:%d\n", ack6);

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

298 printf("Row_Width=%d\n", version);
299 return 0;
300 }

```

```

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

#define VEC_MAG_SQ(vec)           FPSUM(FPSQR(vec.x),FPSQR(vec.y))
#define VEC_DOT_PROD(vec1,vec2)    FPSUM( \
    FPMUL(vec1.x,vec2.x),FPMUL(vec1.y,vec2.y) \
)
long long ballRegisters[NUMBER_OF_BALLS] = {
    BALL_0_BASE,
    BALL_1_BASE,
    BALL_2_BASE,
    BALL_3_BASE,
    BALL_4_BASE,
    BALL_5_BASE,
    BALL_6_BASE
};

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

#define VGA_NOT_READY (IORD_16DIRECT(VGA_BASE, VGA_FLAG * 2) & 0x0001)

/*
// Ball dynamics implementation
*/
bool_t isBallMoving(const struct ball_t *ball){
    return (((ball->vel.x != 0) || (ball->vel.y != 0)) ? TRUE : FALSE);
}

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

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

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

    if(ball->vel.y > 0){
        vColTime = FPDIV((TABLE_END_Y - (ball->pos.y + ball->radius)), ball->vel.y);
        verEdge = BOTTOM;
    } else if(ball->vel.y < 0){
        vColTime = FPDIV((TABLE_START_Y - (ball->pos.y - ball->radius)), ball->vel.y);
    }
}

```

```

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 long long collisionWithBallTime(
97     const struct ball_t *ball1,
98     const struct ball_t *ball2
99 ){
100    long long contactDist = ball1->radius + ball2->radius;
101    struct vector relativeVelocity;
102    struct vector relativeDisplacement;
103    long long result;
104
105    relativeVelocity.x = ball1->vel.x - ball2->vel.x;
106    relativeVelocity.y = ball1->vel.y - ball2->vel.y;
107
108    relativeDisplacement.x = ball1->pos.x - ball2->pos.x;
109    relativeDisplacement.y = ball1->pos.y - ball2->pos.y;
110
111    long long A = VEC_MAG_SQ(relativeVelocity);
112    long long B = 2 * VEC_DOT_PROD(relativeDisplacement, relativeVelocity);
113    long long C = VEC_MAG_SQ(relativeDisplacement) - FPSQR(contactDist);
114
115    long long BSQ_MINUS_4AC = FPSQR(B) - 4LL * FPMUL(A,C);
116
117    if((BSQ_MINUS_4AC < 0) || (A == 0)){
118        result = INT2FP(1000LL);
119    } else{
120        result = FPDIV((-B - FPSQRT(BSQ_MINUS_4AC)), (2LL * A));
121    }
122
123    return result;
124 }
125
126 void handleBallCollision(struct ball_t *ball1, struct ball_t *ball2){
127     const long long A = FPDIV(INT2FP(22LL),INT2FP(30LL));
128     const long long B = FPDIV(INT2FP(22LL),INT2FP(30LL));
129     const long long C = FPDIV(INT2FP(22LL),INT2FP(30LL));
130     const long long D = FPDIV(INT2FP(22LL),INT2FP(30LL));
131
132     struct vector unitRelativeDisp1To2;
133     struct vector tempUnitRelativeDisp1To2;
134
135     unitRelativeDisp1To2.x = ball2->pos.x - ball1->pos.x;
136     unitRelativeDisp1To2.y = ball2->pos.y - ball1->pos.y;
137
138     long long relativeDispMag = FPSQRT(VEC_MAG_SQ(unitRelativeDisp1To2));
139     unitRelativeDisp1To2.x =
140         FPDIV(unitRelativeDisp1To2.x, relativeDispMag);
141     unitRelativeDisp1To2.y =
142         FPDIV(unitRelativeDisp1To2.y, relativeDispMag);
143
144     tempUnitRelativeDisp1To2.x = unitRelativeDisp1To2.x;
145     tempUnitRelativeDisp1To2.y = unitRelativeDisp1To2.y;
146
147     long long ball1VelocityComp =
148         VEC_DOT_PROD(ball1->vel, unitRelativeDisp1To2);
149     long long ball2VelocityComp =
150         VEC_DOT_PROD(ball2->vel, unitRelativeDisp1To2);
151
152     long long newVelocityCompMagBall1 =
153         FPMUL(A, ball1VelocityComp) - FPMUL(B, ball2VelocityComp);
154     long long newVelocityCompMagBall2 =
155         FPMUL(C, ball2VelocityComp) - FPMUL(D, ball1VelocityComp);
156
157     unitRelativeDisp1To2.x = FPMUL(
158         (unitRelativeDisp1To2.x), newVelocityCompMagBall1

```

```

160 );
162     unitRelativeDisp1To2.y = FPMUL(
163         (unitRelativeDisp1To2.y), newVelocityCompMagBall1
164     );
165
166     tempUnitRelativeDisp1To2.x = FPMUL(
167         (tempUnitRelativeDisp1To2.x), newVelocityCompMagBall2
168     );
169     tempUnitRelativeDisp1To2.y = FPMUL(
170         (tempUnitRelativeDisp1To2.y), newVelocityCompMagBall2
171     );
172
173     ball1->vel.x -= unitRelativeDisp1To2.x;
174     ball1->vel.y -= unitRelativeDisp1To2.y;
175     ball2->vel.x -= tempUnitRelativeDisp1To2.x;
176     ball2->vel.y -= tempUnitRelativeDisp1To2.y;
177 }
178
179 long long handleCollisionWithCue(
180     struct ball_t *ball1,
181     const struct ball_t *cue
182 ){
183     const long long A = FPDIV(INT2FP(8LL),INT2FP(50LL));
184     const long long B = FPDIV(INT2FP(28LL),INT2FP(50LL));
185
186     struct vector unitRelativeDisp1To2;
187     struct vector tempUnitRelativeDisp1To2;
188
189     unitRelativeDisp1To2.x = cue->pos.x - ball1->pos.x;
190     unitRelativeDisp1To2.y = cue->pos.y - ball1->pos.y;
191
192     long long relativeDispMag = FPSQRT(VEC_MAG_SQ(unitRelativeDisp1To2));
193     unitRelativeDisp1To2.x =
194         FPDIV(unitRelativeDisp1To2.x, relativeDispMag);
195     unitRelativeDisp1To2.y =
196         FPDIV(unitRelativeDisp1To2.y, relativeDispMag);
197
198     tempUnitRelativeDisp1To2.x = unitRelativeDisp1To2.x;
199     tempUnitRelativeDisp1To2.y = unitRelativeDisp1To2.y;
200
201     long long ball1VelocityComp =
202         VEC_DOT_PROD(ball1->vel, unitRelativeDisp1To2);
203     long long cueVelocityComp =
204         VEC_DOT_PROD(cue->vel, unitRelativeDisp1To2);
205
206     long long newVelocityCompMagBall1 =
207         FPMUL(A, ball1VelocityComp) - FPMUL(B, cueVelocityComp);
208
209     unitRelativeDisp1To2.x = FPMUL(
210         (unitRelativeDisp1To2.x), newVelocityCompMagBall1
211     );
212     unitRelativeDisp1To2.y = FPMUL(
213         (unitRelativeDisp1To2.y), newVelocityCompMagBall1
214     );
215
216     ball1->vel.x -= unitRelativeDisp1To2.x;
217     ball1->vel.y -= unitRelativeDisp1To2.y;
218
219     return 0;
220 }
221
222 void moveBalls(struct ball_t *balls, long long time){
223     int i;
224
225     for(i=0; i<NUM_BALLS; i++){
226         if((BALL_INVISIBLE != balls[i].ballState) &&
227             (TRUE == isBallMoving(&balls[i])))
228         {
229             balls[i].pos.x += FPMUL(balls[i].vel.x, time);
230             balls[i].pos.y += FPMUL(balls[i].vel.y, time);
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<NUM_BALLS; 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     DRAWBALL(i , SPRITE_X(balls [ i ]),SPRITE_Y(balls [ i ]), balls [ i ].colour);
248 }
249
250 void applyFriction(struct ball_t *balls){
251     int i;
252     long long newVelX, newVelY;
253
254     for( i=0; i<NUM_BALLS; i++){
255         if(BALL_INVISIBLE == balls [ i ].ballState){
256             continue;
257         }
258
259         newVelX = balls [ i ].vel.x - FPMUL( balls [ i ].vel.x,DAMPING_COEFF );
260         if(FPMUL( newVelX , balls [ i ].vel.x ) > 0){
261             balls [ i ].vel.x = newVelX;
262         }
263         else{
264             balls [ i ].vel.x = 0;
265         }
266
267         newVelY = balls [ i ].vel.y - FPMUL( balls [ i ].vel.y,DAMPING_COEFF );
268         if(FPMUL( newVelY , balls [ i ].vel.y ) > 0){
269             balls [ i ].vel.y = newVelY;
270         }
271         else{
272             balls [ i ].vel.y = 0;
273         }
274     }
275 }
276 }
```

```

/*
 * Software for Interactive Project Pool Game
 * Columbia University. New York, 2008
 * Authors:
 *   Abdulhamid Ghandour
 *   Thomas John
 *   Jaime Peretzman
 *   Bharadwaj Vellore
 *
 * Desc:
 */
#include "fixedpoint.h"
#include "debug.h"
/*
// Thanks to GameProgrammer.com
*/
#define step(shift) \
    if((0x4000000001 >> shift) + root <= num) \
    { \
        num -= (0x4000000001 >> shift) + root; \
        root = (root >> 1) | (0x4000000001 >> shift); \
    } \
    else \
    { \
        root = root >> 1; \
    }
long long FPSQRT(long long num){
    long long root = 0;

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

    // round to the nearest integer, cuts max error in half
    if( root < num )
    {
        ++root;
    }

    root <<= 6;

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

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

    return root;
} */
void printFP(long long fpnum){
    int i;
    float factor = 0.5;

```

```
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 }
```

```

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

unsigned long sevensegment[] = { /* Active Low -> xgfedcba */
    0x40, /* 0 -> 01000000 */
    0x79, /* 1 -> 01111001 */
    0x24, /* 2 -> 00100100 */
    0x30, /* 3 -> 00110000 */
    0x19, /* 4 -> 00011001 */
    0x12, /* 5 -> 00010010 */
    0x02, /* 6 -> 00000010 */
    0xF8, /* 7 -> 01111000 */
    0x00, /* 8 -> 00000000 */
    0x10, /* 9 -> 00010000 */
    0xC0 /* P -> 00001100 */
};

#define P 10

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

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

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

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

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

```

```

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

14 entity i2c_controller is
15
16     port (
17         clk          : in std_logic;
18         reset_n     : in std_logic;
19         read         : in std_logic;
20         write        : in std_logic;
21         chipselect   : in std_logic;
22         address      : in unsigned(3 downto 0);
23         readdata     : out unsigned(31 downto 0);
24         writedata    : in unsigned(31 downto 0);
25         sclk         : out std_logic;
26         sdat         : inout std_logic;
27         ack          : in std_logic
28     );
29 end i2c_controller;
30
31 architecture rtl of i2c_controller is
32
33     type ram_type is array(7 downto 0) of unsigned(31 downto 0);
34     signal RAM : ram_type;
35     signal ram_address : unsigned(2 downto 0);
36     signal counter : unsigned(31 downto 0);
37     signal int_sclk : std_logic := '1';
38     signal int_sdat : std_logic := '1';
39     signal int_ack : std_logic := '0';
40
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              int_sclk <= RAM(0)(0);
83          end if;
84      end if;
85  end process i2c_line_control;
86
87  sdat <= RAM(1)(0) when RAM(1)(1) = '1' else 'Z';
88  sclk <= int_sclk;
89
90 end rtl;
```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity calibration is
port (
    reset      : in std_logic;
    clk        : in std_logic;
    valid_green : in std_logic;
    end_row    : in std_logic;
    end_frame  : in std_logic;
    green_pixel_value : in unsigned (9 downto 0);
    -- green_column_thr : in unsigned (9 downto 0);
    -- green_row_thr   : in unsigned (9 downto 0);
    repos      : out unsigned (6 downto 0) := "1000000";
    x_1        : out unsigned (10 downto 0) := "000000000000";
    y_1        : out unsigned (10 downto 0) := "000000000000";
    x_2        : out unsigned (10 downto 0) := "000000000000";
    y_2        : out unsigned (10 downto 0) := "000000000000";
    calibration_on : in std_logic;
    threshold   : in unsigned (9 downto 0);
    leds       : out unsigned(6 downto 0)
);
end calibration;

architecture rtl of calibration is

-- signals for calibration
signal row_counter      : unsigned (10 downto 0):= "000000000000";
signal green_column_count : unsigned (10 downto 0):= "000000000000";
signal green_row_count   : unsigned (10 downto 0):= "000000000000";
signal green_column_thr  : unsigned (9 downto 0) := "0001111101";
signal green_row_thr     : unsigned (9 downto 0) := "0110010000";
signal green_color_thr   : unsigned (9 downto 0);
signal green_x1_thr      : unsigned (9 downto 0) := "0001101000";
signal black_column_thr  : unsigned (2 downto 0) := "011";
signal black_column_count : unsigned (2 downto 0) := "000";
signal temp_active_row   : unsigned (2 downto 0) := "000";
signal temp_changes_sig  : unsigned (1 downto 0) := "00";
signal black_first_count : unsigned (4 downto 0) := "00000";
signal first_row_black   : std_logic := '0';
signal temp_left_column  : std_logic := '0';
signal temp_green_row    : std_logic := '0';
signal temp_right_column : std_logic := '0';
signal flag              : std_logic := '0';
signal flag2             : std_logic := '0';
signal start_flag         : std_logic := '0';
signal temp_x1            : unsigned (10 downto 0) := "000000000000";
signal temp_y1            : unsigned (10 downto 0) := "000000000000";
signal temp_x2            : unsigned (10 downto 0) := "011111111111";
signal temp_y2            : unsigned (10 downto 0) := "000000000000";
signal column_counter    : unsigned (10 downto 0) := "000000000000";
signal active_row         : unsigned (2 downto 0) := "000";
signal changes_sig        : unsigned (1 downto 0) := "00";
signal cam_repos          : unsigned (6 downto 0) := "1000000";
signal debug_x_counter   : unsigned (10 downto 0) := "000000000000";
signal debug_x_max         : unsigned (10 downto 0) := "000000000000";
begin
green_color_thr <= threshold;

Calib : process (clk)
begin
if rising_edge(clk) then
if reset = '1' then
row_counter <= (others=>'0');
green_column_count <= (others =>'0');
green_row_count   <= (others =>'0');
black_column_count <= (others =>'0');
temp_active_row   <= (others =>'0');
temp_changes_sig  <= (others =>'0');
black_first_count <= (others =>'0');
temp_green_row    <= '0';
temp_left_column  <= '0';
temp_right_column <= '0';
flag              <= '0';
flag2             <= '0';

```

```

80      first_row_black     <= '0';
81      start_flag          <= '0';
82      x_1                  <= (others =>'0');
83      x_2                  <= (others =>'0');
84      y_1                  <= (others =>'0');
85      y_2                  <= (others =>'0');
86      temp_x_1              <= (others =>'0');
87      temp_y_1              <= (others =>'0');
88      temp_x_2              <= "0111111111";
89      temp_y_2              <= (others =>'0');
90      column_counter        <= (others =>'0');
91      active_row             <= (others =>'0');
92      changes_sig            <= (others =>'0');
93      cam_repos              <= "1000000";
94
95  elsif valid_green = '1' and calibration_on = '1' then
96      -- if start_flag = '1' then
97      -- cam_repos <= (others =>'1');
98      if green_pixel_value > green_color_thr then
99          green_column_count <= green_column_count + 1;
100         black_column_count <= (others =>'0');
101     else
102         black_column_count <= black_column_count + 1;
103     end if;
104
105     column_counter <= column_counter + 1;
106     debug_x_counter <= debug_x_counter + 1; -- *****DEBUG*****
107
108     if black_column_count >= black_column_thr then
109         if green_column_count >= green_column_thr then
110             temp_right_column <= '1';
111         elsif green_column_count < green_column_thr then
112             temp_left_column <= '1';
113             green_column_count <= (others =>'0');
114         end if;
115     end if;
116
117     if green_column_count >= green_column_thr then
118         temp_green_row <= '1';
119         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
121                 temp_x_1 <= column_counter - green_column_thr;
122                 flag2 <='1';
123             end if;
124         if black_column_count >= black_column_thr then
125             temp_x_2 <= column_counter - black_column_thr;
126             flag <='1';
127         end if;
128         if black_column_count >= black_column_thr then
129             if temp_x_2 < column_counter - black_column_thr and flag = '0' then
130                 temp_x_2 <= column_counter - black_column_thr;
131                 flag <='1';
132             end if;
133         end if;
134     end if;
135
136     if black_first_count > 1 then
137         first_row_black <='1';
138         black_first_count <=(others=>'0');
139     end if;
140 end if; -- end of valid green
141
142 if end_row = '1' then
143     if temp_green_row ='1' then
144         green_row_count <= green_row_count + 1;
145     end if;
146
147 --*****DEBUG*****
148 debug_x_counter <= (others => '0');
149
150 if debug_x_counter > debug_x_max then
151     debug_x_max <= debug_x_counter;
152
153 end if;
154 --*****DEBUG END*****
155 if row_counter < 20 and temp_green_row ='0' then
156     black_first_count <= black_first_count +1;
157 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;
169
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;
175
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;
185  -- 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";
242      active_row      <= temp_active_row;
243      changes_sig     <= temp_changes_sig;
244      x_1              <= temp_x_1;
245      y_1              <= temp_y_1;
246      x_2              <= temp_x_2;
247      y_2              <= temp_y_2;
248      temp_x_1          <= (others =>'0');
249      temp_y_1          <= (others =>'0');
250      temp_x_2          <= "011111111111";
251      temp_y_2          <= (others =>'0');
252      temp_changes_sig  <= (others=>'0');
253      row_counter       <= (others=>'0');
254      column_counter    <= (others=>'0');
255      black_first_count <= (others=>'0');
256      temp_green_row    <= '0';
257      temp_left_column  <= '0';
258      temp_right_column <= '0';
259      flag              <= '0';
260      flag2             <= '0';
261      first_row_black   <= '0';
262      temp_active_row   <= (others=>'0');
263      green_column_count <= (others=>'0');
264      green_row_count   <= (others=>'0');
265      black_column_count <= (others=>'0');
266      start_flag         <= '1';
267      end if;
268      if calibration_on = '0' then
269          cam_repos        <= "1000000";
270          x_1              <= (others=>'0');
271          x_2              <= (others=>'0');
272          y_1              <= (others=>'0');
273          y_2              <= (others=>'0');
274      end if;
275      if chipslect = '1' then
276          if write = '1' then
277              if address = "000" then
278                  calibration_on <= writedata(0);
279                  cam_repos        <= "1000000";
280              end if;
281          end if;
282          if read = '1' then
283              if address = "001" then
284                  readdata(6 downto 0) <= cam_repos;
285              elsif address = "010" then
286                  readdata(9 downto 0) <= x_1;
287              elsif address = "011" then
288                  readdata(9 downto 0) <= y_1;
289              elsif address = "100" then
290                  readdata(9 downto 0) <= x_2;
291              elsif address = "101" then
292                  readdata(9 downto 0) <= y_2;
293              end if;
294          end if;
295      end if;— end of chipslect
296      leds <= cam_repos;
297      repos <= cam_repos;
298  end if;
299  end process Calib;
300 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
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
15
16 entity ci_pxl is
17   port(
18     clk          : in std_logic;      -- Master CLK to Camera
19     mclk         : out std_logic;    -- Line Valid from Camera
20     lval         : in std_logic;    -- Frame Valid from Camera
21     fval         : in std_logic;    -- Pixel CLK from Camera
22     datain       : in unsigned(9 downto 0); -- Pixel Data from Camera
23     dataout      : out unsigned(9 downto 0);
24     valid_green : out std_logic;
25     end_of_frame: out std_logic;
26     end_of_row  : out std_logic;
27     sandboxStartX: in unsigned(31 downto 0);
28     sandboxStartY: in unsigned(31 downto 0);
29     sandboxEndX : in unsigned(31 downto 0);
30     sandboxEndY : in unsigned(31 downto 0)
31   );
32 end ci_pxl;
33
34 architecture pool of ci_pxl is
35   signal int_mclk : std_logic := '0';
36   signal last_line_valid: std_logic := '0';
37   signal last_frame_valid: std_logic := '0';
38   signal pixel_counter: std_logic := '0';
39   signal line_counter: std_logic := '0';
40   signal last_pixclk: std_logic := '0';
41 begin
42   mclkgen : process(clk)
43   begin
44     if rising_edge(clk) then
45       int_mclk <= not int_mclk;
46     end if;
47   end process;
48
49   eor_gen : process(clk)
50   begin
51     if rising_edge(clk) then
52       if (last_line_valid = '1' and lval = '0') then
53         end_of_row <= '1';
54         line_counter <= not line_counter;
55       else
56         end_of_row <= '0';
57       end if;
58       if fval = '0' then
59         line_counter <= '0';
60       end if;
61       last_line_valid <= lval;
62     end if;
63   end process eor_gen;
64
65   eof_gen : process(clk)
66   begin
67     if rising_edge(clk) then
68       if (last_frame_valid = '1' and fval = '0') then
69         end_of_frame <= '1';
70       else
71         end_of_frame <= '0';
72       end if;
73       last_frame_valid <= fval;
74     end if;
75   end process eof_gen;
76
77   vg_gen : process(clk)
78   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;
        crop_end_x_sig <= crop_end_x;
        crop_start_y_sig <= crop_start_y;
        crop_end_y_sig <= crop_end_y;
      end if;
    end if;
  end process control;
40
xcounter : process (clk)
42 begin
    if rising_edge (clk) then
44      if end_row_in = '1' then
        xcount <= (others => '0');
      else
46        if valid_green_in = '1' then
          xcount <= xcount + 1;
        end if;
      end if;
    end if;
  end process xcounter;
54
ycounter : process (clk)
begin
56  if rising_edge (clk) then
      if end_frame = '1' then
58      ycount <= (others => '0');
      else
60        if end_row_in = '1' then
          ycount <= ycount + 1;
        end if;
      end if;
    end if;
  end process ycounter;
66
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
3 -- Authors:
4 --     Abdulhamid Ghandour
5 --     Thomas John
6 --     Jaime Peretzman
7 --     Bharadwaj Vellore
8 --
9 -- Desc:
10
11 library ieee;
12 use ieee.std_logic_1164.all;
13 use ieee.numeric_std.all;
14
15 entity visionsystem is
16
17     port(
18         clk : in std_logic;
19
20         pixel_data : in unsigned (9 downto 0);
21         valid_green : in std_logic;
22
23         endofrow : in std_logic;
24         endofframe : in std_logic;
25
26         threshold : in unsigned (9 downto 0);
27
28         xout : out unsigned (15 downto 0);
29         yout : out unsigned (15 downto 0);
30
31         no_detect : out std_logic;
32
33         led0, led1, led2, led3, led4, led5, led6, led7 : out std_logic
34     );
35
36 end visionsystem;
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     --EXTREMETIES
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 + WIDTH_SAMPLE_INTERVAL 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 + WIDTH_SAMPLE_INTERVAL 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
161     end if; — pixel_data < threshold
162 end if; — valid_green = '1'

164 if endofrow = '1' then
165     bottomwidth_start_array(0) <= x"FFFF";
166     bottomwidth_start_array(1) <= bottomwidth_start_array(0);
167     bottomwidth_start_array(2) <= bottomwidth_start_array(1);
168     bottomwidth_start_array(3) <= bottomwidth_start_array(2);
169     bottomwidth_start_array(4) <= bottomwidth_start_array(3);
170     bottomwidth_start_array(5) <= bottomwidth_start_array(4);
171     bottomwidth_start_array(6) <= bottomwidth_start_array(5);
172     bottomwidth_start_array(7) <= bottomwidth_start_array(6);
173     bottomwidth_start_array(8) <= bottomwidth_start_array(7);
174     bottomwidth_start_array(9) <= bottomwidth_start_array(8);
175     bottomwidth_start_array(10) <= bottomwidth_start_array(9);
176     bottomwidth_start_array(11) <= bottomwidth_start_array(10);
177     bottomwidth_start_array(12) <= bottomwidth_start_array(11);
178     bottomwidth_start_array(13) <= bottomwidth_start_array(12);
179     bottomwidth_start_array(14) <= bottomwidth_start_array(13);
180     bottomwidth_start_array(15) <= bottomwidth_start_array(14);
181     bottomwidth_start_array(16) <= bottomwidth_start_array(15);
182     bottomwidth_start_array(17) <= bottomwidth_start_array(16);
183     bottomwidth_start_array(18) <= bottomwidth_start_array(17);
184     bottomwidth_start_array(19) <= bottomwidth_start_array(18);

186     bottomwidth_end_array(0) <= x"0000";
187     bottomwidth_end_array(1) <= bottomwidth_end_array(0);
188     bottomwidth_end_array(2) <= bottomwidth_end_array(1);
189     bottomwidth_end_array(3) <= bottomwidth_end_array(2);
190     bottomwidth_end_array(4) <= bottomwidth_end_array(3);
191     bottomwidth_end_array(5) <= bottomwidth_end_array(4);
192     bottomwidth_end_array(6) <= bottomwidth_end_array(5);
193     bottomwidth_end_array(7) <= bottomwidth_end_array(6);
194     bottomwidth_end_array(8) <= bottomwidth_end_array(7);
195     bottomwidth_end_array(9) <= bottomwidth_end_array(8);
196     bottomwidth_end_array(10) <= bottomwidth_end_array(9);
197     bottomwidth_end_array(11) <= bottomwidth_end_array(10);
198     bottomwidth_end_array(12) <= bottomwidth_end_array(11);
199     bottomwidth_end_array(13) <= bottomwidth_end_array(12);
200     bottomwidth_end_array(14) <= bottomwidth_end_array(13);
201     bottomwidth_end_array(15) <= bottomwidth_end_array(14);
202     bottomwidth_end_array(16) <= bottomwidth_end_array(15);
203     bottomwidth_end_array(17) <= bottomwidth_end_array(16);
204     bottomwidth_end_array(18) <= bottomwidth_end_array(17);
205     bottomwidth_end_array(19) <= bottomwidth_end_array(18);

206 if not (bottomy = 0) and bottomy = prev_bottomy and bottomwidth_recorded = '0' then
207     bottomwidth_start <= bottomwidth_start_array(19);
208     bottomwidth_end <= bottomwidth_end_array(19);
209     bottomwidth_recorded <= '1';
210 end if;
211
212     prev_bottomy <= bottomy;
213
214 end if;
215
216 — RESET AT END OF FRAME
217 if endofframe = '1' then
218     topx <= (others = > '0');
219     topy <= (others = > '1');
220     bottomx <= (others = > '0');
221     bottomy <= (others = > '0');
222     leftx <= (others = > '1');
223     lefty <= (others = > '0');
224     rightx <= (others = > '0');
225     righty <= (others = > '0');

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

228     prev_bottomy <= (others = > '0');
229     bottomwidth_start <= (others = > '0');
230     bottomwidth_end <= (others = > '0');
231     bottomwidth_start_array <= (others => x"FFFF");
232     bottomwidth_end_array <= (others => x"0000");
233     bottomwidth_recorded <= '0';
234 end if;
235
236 end if;

```

```

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

```

```

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

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

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

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

```

```

400         end if;
402 ---*****BOTTOM LEFT***** 
403     elsif bottom_on_edge = 1 and left_on_edge = 1 then
404
405         if(topx > rightx) then
406             xdiff := topx - rightx;
407         else
408             xdiff := rightx - topx;
409         end if;
410
411         if(topy > righty) then
412             ydiff := topy - righty;
413         else
414             ydiff := righty - topy;
415         end if;
416
417         if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
418             xout <= rightx;
419             yout <= righty;
420         else
421             if topwidth_end - topwidth_start < FINGER_WIDTH then
422                 xout <= topx;
423                 yout <= topy;
424                 led3 <= '1';
425             else
426                 xout <= rightx;
427                 yout <= righty;
428                 led4 <= '1';
429             end if;
430         end if;
431
432 ---*****BOTTOM RIGHT***** 
433     elsif bottom_on_edge = 1 and right_on_edge = 1 then
434
435         if(topx > leftx) then
436             xdiff := topx - leftx;
437         else
438             xdiff := leftx - topx;
439         end if;
440
441         if(topy > lefty) then
442             ydiff := topy - lefty;
443         else
444             ydiff := lefty - topy;
445         end if;
446
447         if (xdiff < SAME_EDGE_TOLERANCE and ydiff < SAME_EDGE_TOLERANCE) then
448             xout <= leftx;
449             yout <= lefty;
450         else
451             if topwidth_end - topwidth_start < FINGER_WIDTH then
452                 xout <= topx;
453                 yout <= topy;
454             else
455                 xout <= leftx;
456                 yout <= lefty;
457             end if;
458         end if;
459     end if;
460
461         end if; --- End of Frame
462     end if;
463     end process output;
464
465 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 -- Avalon Interface for the Vision Block.
11 -- readdata has X co-ord in lower 16 bits
12 --         has Y co-ord in next 15 bits
13 --         has no_detect in MSB
14 --
15
16 library ieee;
17 use ieee.std_logic_1164.all;
18 use ieee.numeric_std.all;
19
20 entity avalon_vision is
21
22 port (
23     -- Avalon Signals
24     clk : in std_logic;
25     reset_n : in std_logic;
26     address : in unsigned(4 downto 0);
27     write : in std_logic;
28     read : in std_logic;
29     chipselect : in std_logic;
30     readdata : out unsigned(31 downto 0);
31     writedata : in unsigned(31 downto 0);
32
33     -- Camera Signals
34     master_clk : out std_logic;
35     pixel_clk : in std_logic;
36     line_valid : in std_logic;
37     frame_valid : in std_logic;
38     pixel_data : in unsigned(9 downto 0);
39
40     -- Board Signals
41     threshold : in unsigned(9 downto 0);    -- SW9 to SW0
42     no_detect : out std_logic;                -- LEDG0
43     cal_direction : out unsigned(6 downto 0);
44     vision_flags : out unsigned(7 downto 0)
45 );
46 end avalon_vision;
47
48 architecture toplevel of avalon_vision is
49
50 signal ram_address : unsigned(4 downto 0);
51 signal data_signal : unsigned(9 downto 0);
52 signal valid_green_signal : std_logic;
53 signal valid_green_cropped_signal : std_logic;
54 signal end_of_frame_signal : std_logic;
55 signal end_of_row_signal : std_logic;
56 signal end_of_row_cropped_signal : std_logic;
57 signal no_detect_signal : std_logic;
58 signal x_1_signal, x_2_signal, y_1_signal, y_2_signal : unsigned(10 downto 0);
59 signal calibration_on_signal : std_logic;
60 signal calibration_on_signal_int : std_logic;
61 signal repos_signal : unsigned(6 downto 0);
62 signal xout_signal : unsigned(15 downto 0);
63 signal yout_signal : unsigned(14 downto 0);
64 signal reset_int : std_logic := '0';
65 signal sandboxStartX_signal : unsigned(31 downto 0) := (others => '0');
66 signal sandboxEndX_signal : unsigned(31 downto 0) := (others => '1');
67 signal sandboxStartY_signal : unsigned(31 downto 0) := (others => '0');
68 signal sandboxEndY_signal : unsigned(31 downto 0) := (others => '1');
69 signal green_column_thr_signal : unsigned(9 downto 0) := "0100101100";
70 signal green_row_thr_signal : unsigned(9 downto 0) := "0110010000";
71 component ci_pxl port(
72     clk : in std_logic;
73     mclk : out std_logic;    -- Master CLK to Camera
74     lval : in std_logic;    -- Line Valid from Camera
75     fval : in std_logic;    -- Frame Valid from Camera
76     pixelclk : in std_logic; -- Pixel CLK from Camera
77     datain : in unsigned(9 downto 0);   -- Pixel Data from Camera
78     dataout : out unsigned(9 downto 0);
79     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) := "000000000000";
114   y_1        : out unsigned (10 downto 0) := "000000000000";
115   x_2        : out unsigned (10 downto 0) := "000000000000";
116   y_2        : out unsigned (10 downto 0) := "000000000000";
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
157   VISION: visionsystem port map(
158     clk => clk,
159     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;
242      readdata(21 downto 11) <= y_2_signal;
244      readdata(31 downto 22) <= (others => '0');
246      elsif ram_address = 3 then
247          readdata <= ("00000000000000000000000000000000" & repos_signal);
248      end if;
249      end if;-- end of chipselect
250  end process host_control;
251
252  no_detect <= no_detect_signal;
253  calibration_on_signal <= calibration_on_signal_int;
254  cal_direction(6) <= valid_green_signal;
255
end toplevel;

```

```

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
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
15
16 entity de2_vga_raster is
17
18   port (
19     reset      : in std_logic;
20     clk        : 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(15 downto 0);
26     writedata  : in unsigned(15 downto 0);
27
28     VGA_CLK,           -- Clock
29     VGA_HS,            -- HSYNC
30     VGA_VS,            -- VSYNC
31     VGA_BLANK,         -- BLANK
32     VGA_SYNC : out std_logic; -- SYNC
33     VGA_R,             -- Red[9:0]
34     VGA_G,             -- Green[9:0]
35     VGA_B : out unsigned(9 downto 0) -- Blue[9:0]
36   );
37
38 end de2_vga_raster;
39
40 architecture rtl of de2_vga_raster is
41
42   -- Video parameters
43
44   constant HTOTAL      : integer := 800;
45   constant HSYNC        : integer := 96;
46   constant HBACK_PORCH : integer := 48;
47   constant HACTIVE      : integer := 640;
48   constant HFRONT_PORCH : integer := 16;
49
50   constant VTOTAL      : integer := 525;
51   constant VSYNC        : integer := 2;
52   constant VBACK_PORCH : integer := 33;
53   constant VACTIVE      : integer := 480;
54   constant VFRONT_PORCH : integer := 10;
55
56   constant ball_dia      : integer      := 29;
57   constant cross_dia     : integer      := 16;
58   constant border        : integer      := 15;
59
60   signal black_b_x      : unsigned(7 downto 0) := "00001111";
61   signal black_b_y      : unsigned(7 downto 0) := "00011110";
62   signal border_1, border_2, border_3, border_4 : unsigned(9 downto 0) := "0000000000";
63   signal C_H_start_1    : unsigned(9 downto 0) := "0000000000";
64   signal C_V_Start_1    : unsigned(9 downto 0) := "0000000000";
65   signal C_color_1       : unsigned(2 downto 0) := "000";
66   signal C_H_start_2    : unsigned(9 downto 0) := "0000000000";
67   signal C_V_Start_2    : unsigned(9 downto 0) := "0000000000";
68   signal C_color_2       : unsigned(2 downto 0) := "000";
69   signal C_H_start_3    : unsigned(9 downto 0) := "0000000000";
70   signal C_V_Start_3    : unsigned(9 downto 0) := "0000000000";
71   signal C_color_3       : unsigned(2 downto 0) := "000";
72   signal C_H_start_4    : unsigned(9 downto 0) := "0000000000";
73   signal C_V_Start_4    : unsigned(9 downto 0) := "0000000000";
74   signal C_color_4       : unsigned(2 downto 0) := "000";
75   signal C_H_start_5    : unsigned(9 downto 0) := "0000000000";
76   signal C_V_Start_5    : unsigned(9 downto 0) := "0000000000";
77   signal C_color_5       : unsigned(2 downto 0) := "000";
78   signal C_H_start_6    : unsigned(9 downto 0) := "0000000000";
79   signal C_V_Start_6    : unsigned(9 downto 0) := "0000000000";
80   signal C_color_6       : unsigned(2 downto 0) := "000";

```



```

240      temp_C_V_Start_2      <= (others => '0');
242      temp_C_color_2        <= (others => '0');
244      C_H_start_2           <= (others => '0');
246      C_V_start_2           <= (others => '0');
248      C_color_2             <= (others => '0');
250      temp_C_H_start_3      <= (others => '0');
252      temp_C_V_Start_3      <= (others => '0');
254      temp_C_color_3         <= (others => '0');
256      C_H_start_3           <= (others => '0');
258      C_V_start_3           <= (others => '0');
260      C_color_3             <= (others => '0');
262      temp_C_H_start_4      <= (others => '0');
264      temp_C_V_Start_4      <= (others => '0');
266      temp_C_color_4         <= (others => '0');
268      C_H_start_4           <= (others => '0');
270      C_V_start_4           <= (others => '0');
272      C_color_4             <= (others => '0');
274      temp_C_H_start_5      <= (others => '0');
276      temp_C_V_Start_5      <= (others => '0');
278      temp_C_color_5         <= (others => '0');
280      C_H_start_5           <= (others => '0');
282      C_V_start_5           <= (others => '0');
284      C_color_5             <= (others => '0');
286      temp_C_H_start_6      <= (others => '0');
288      temp_C_V_Start_6      <= (others => '0');
290      temp_C_color_6         <= (others => '0');
292      C_H_start_6           <= (others => '0');
294      C_V_start_6           <= (others => '0');
296      C_color_6             <= (others => '0');
298      temp_C_H_start_7      <= (others => '0');
300      temp_C_V_Start_7      <= (others => '0');
302      temp_C_color_7         <= (others => '0');
304      received_check         <= (others => '0');
306      received_cal           <= (others => '0');
308      black_b_x              <= (others => '0');
310      black_b_y              <= (others => '0');

else
  if chipselect = '1' then
    if write = '1' then
      if address = "00000" then
        temp_C_H_start_1      <= writedata(9 downto 0);
        received_check(0) <= '1';
      elsif address = "00001" then
        temp_C_V_Start_1      <= writedata(9 downto 0);
        received_check(1) <= '1';
      elsif address = "00010" then
        temp_C_color_1         <= writedata(2 downto 0);
        received_check(2) <= '1';

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

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

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

```

```

320      elsif address = "01011" then
321          temp_C_color_4 <= writedata(2 downto 0);
322          received_check(11) <= '1';
323
324      elsif address = "10110" then
325          temp_C_H_start_5 <= writedata(9 downto 0);
326          received_check(12) <= '1';
327      elsif address = "10111" then
328          temp_C_V_Start_5 <= writedata(9 downto 0);
329          received_check(13) <= '1';
330      elsif address = "11000" then
331          temp_C_color_5 <= writedata(2 downto 0);
332          received_check(14) <= '1';
333
334      elsif address = "11001" then
335          temp_C_H_start_6 <= writedata(9 downto 0);
336          received_check(15) <= '1';
337      elsif address = "11010" then
338          temp_C_V_Start_6 <= writedata(9 downto 0);
339          received_check(16) <= '1';
340      elsif address = "11011" then
341          temp_C_color_6 <= writedata(2 downto 0);
342          received_check(17) <= '1';
343
344      elsif address = "11100" then
345          temp_C_H_start_7 <= writedata(9 downto 0);
346          received_check(18) <= '1';
347      elsif address = "11101" then
348          temp_C_V_Start_7 <= writedata(9 downto 0);
349          received_check(19) <= '1';
350      elsif address = "11110" then
351          temp_C_color_7 <= writedata(2 downto 0);
352          received_check(20) <= '1';
353
354      elsif address = "01101" then
355          temp_cross_H <= writedata(9 downto 0);
356
357      elsif address = "01110" then
358          temp_cross_V <= writedata(9 downto 0);
359
360
361      elsif address = "10000" then
362          temp_stick_H_1 <= writedata(9 downto 0);
363          received_cal(0) <= '1';
364      elsif address = "10001" then
365          temp_stick_V_1 <= writedata(9 downto 0);
366          received_cal(1) <= '1';
367      elsif address = "10010" then
368          temp_stick_H_2 <= writedata(9 downto 0);
369          received_cal(2) <= '1';
370      elsif address = "10011" then
371          temp_stick_V_2 <= writedata(9 downto 0);
372          received_cal(3) <= '1';
373      elsif address = "10101" then --21
374          temp_border <= '0';
375          received_cal(4) <= '1';
376      elsif address = "11111" then
377          black_b_x <= writedata(7 downto 0);
378          black_b_y <= writedata(15 downto 8);
379      end if; -- end of if address
380
381      end if; -- end of if write
382      if read = '1' and address = "01100" then
383          if received_check = "000000000000000000000000" then
384              readdata(0) <= '0';
385          else
386              readdata(0) <= '1';
387          end if;
388      end if; --end of read
389      if read = '1' and address = "10100" then
390          if received_cal = "00000" then
391              readdata(0) <= '0';
392          else
393              readdata(0) <= '1';
394          end if;
395      end if; --end of read
396      end if; --ship select
397      if EndOfLine = '1' and EndOfField = '1' then

```

```

400      if received_check = "111111111111111111111111" then
401          C_H_start_1      <= temp_C_H_start_1;
402          C_V_Start_1      <= temp_C_V_Start_1;
403          C_color_1        <= temp_C_color_1;
404          C_H_start_2      <= temp_C_H_start_2;
405          C_V_Start_2      <= temp_C_V_Start_2;
406          C_color_2        <= temp_C_color_2;
407          C_H_start_3      <= temp_C_H_start_3;
408          C_V_Start_3      <= temp_C_V_Start_3;
409          C_color_3        <= temp_C_color_3;
410          C_H_start_4      <= temp_C_H_start_4;
411          C_V_Start_4      <= temp_C_V_Start_4;
412          C_color_4        <= temp_C_color_4;
413          C_H_start_5      <= temp_C_H_start_5;
414          C_V_Start_5      <= temp_C_V_Start_5;
415          C_color_5        <= temp_C_color_5;
416          C_H_start_6      <= temp_C_H_start_6;
417          C_V_Start_6      <= temp_C_V_Start_6;
418          C_color_6        <= temp_C_color_6;
419          C_H_start_7      <= temp_C_H_start_7;
420          C_V_Start_7      <= temp_C_V_Start_7;
421          C_color_7        <= temp_C_color_7;
422          cross_H           <= temp_cross_H;
423          cross_V           <= temp_cross_V;
424          calibration       <= '0';
425          received_check    <= (others => '0');
426      elsif received_cal = "11111" and temp_border = '0' then
427          stick_H_1          <= temp_stick_H_1;
428          stick_V_1          <= temp_stick_V_1;
429          stick_H_2          <= temp_stick_H_2;
430          stick_V_2          <= temp_stick_V_2;
431          cross_H            <= temp_cross_H;
432          cross_V            <= temp_cross_V;
433          calibration        <= '1';
434          received_cal       <= (others => '0');
435      elsif received_cal = "11111" and temp_border = '1' then
436          border_1            <= temp_stick_H_1;
437          border_2            <= temp_stick_V_1;
438          border_3            <= temp_stick_H_2;
439          border_4            <= temp_stick_V_2;
440          temp_mid := (temp_stick_H_1+temp_stick_H_2-border-border);
441          Socket_H_start_11   <= (temp_stick_H_1-border);
442          Socket_H_start_14   <= (temp_stick_H_1-border);
443          Socket_V_start_11   <= (temp_stick_V_1-border);
444          Socket_V_start_12   <= (temp_stick_V_1-border);
445          Socket_V_start_13   <= (temp_stick_V_1-border);
446          Socket_H_start_13   <= (temp_stick_H_2-border);
447          Socket_H_start_16   <= (temp_stick_H_2-border);
448          Socket_V_start_14   <= (temp_stick_V_2-border);
449          Socket_V_start_15   <= (temp_stick_V_2-border);
450          Socket_V_start_16   <= (temp_stick_V_2-border);
451          Socket_H_start_12   <= ('0'&(temp_mid(9 downto 1)));
452          Socket_H_start_15   <= ('0'&(temp_mid(9 downto 1)));
453          cross_H             <= temp_cross_H;
454          cross_V             <= temp_cross_V;
455          calibration         <= '0';
456          received_cal        <= (others => '0');

457      end if;
458  end if;
459
460  end if;
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
481     VCounter : process (clk25)
482 begin
483     if rising_edge(clk25) then
484         if reset = '1' then
485             Vcount <= (others => '0');
486         elsif EndOfLine = '1' then
487             if EndOfField = '1' then
488                 Vcount <= (others => '0');
489             else
490                 Vcount <= Vcount + 1;
491             end if;
492         end if;
493     end if;
494 end process VCounter;
495
496 EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';
497
498 -- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK
499
500 HSyncGen : process (clk25)
501 begin
502     if rising_edge(clk25) then
503         if reset = '1' or EndOfLine = '1' then
504             vga_hsync <= '1';
505         elsif Hcount = HSYNC - 1 then
506             vga_hsync <= '0';
507         end if;
508     end if;
509 end process HSyncGen;
510
511 HBlankGen : process (clk25)
512 begin
513     if rising_edge(clk25) then
514         if reset = '1' then
515             vga_hblank <= '1';
516         elsif Hcount = HSYNC + HBACKPORCH then
517             vga_hblank <= '0';
518         elsif Hcount = HSYNC + HBACKPORCH + HACTIVE then
519             vga_hblank <= '1';
520         end if;
521     end if;
522 end process HBlankGen;
523
524 VSyncGen : process (clk25)
525 begin
526     if rising_edge(clk25) then
527         if reset = '1' then
528             vga_vsync <= '1';
529         elsif EndOfLine = '1' then
530             if EndOfField = '1' then
531                 vga_vsync <= '1';
532             elsif Vcount = VSYNC - 1 then
533                 vga_vsync <= '0';
534             end if;
535         end if;
536     end if;
537 end process VSyncGen;
538
539 VBlankGen : process (clk25)
540 begin
541     if rising_edge(clk25) then
542         if reset = '1' then
543             vga_vblank <= '1';
544         elsif EndOfLine = '1' then
545             if Vcount = VSYNC + VBACKPORCH - 1 then
546                 vga_vblank <= '0';
547             elsif Vcount = VSYNC + VBACKPORCH + VACTIVE - 1 then
548                 vga_vblank <= '1';
549             end if;
550         end if;
551     end if;
552 end process VBlankGen;
553
554
555
556
557

```

```

560
562
564
566
568 — BALL generator 1
569
570 RectangleHGen_1 : process (clk25)
571   variable H_boundary : unsigned(0 to 28);
572   variable h_index_1 : unsigned(9 downto 0);
573   variable v_index_1 : unsigned(9 downto 0);
574
575 begin
576   if rising_edge(clk25) then
577     if reset = '1' then
578       rectangle_1 <= '0';
579     elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_1 - 1 and
580           Vcount > VSYNC + VBACK_PORCH + C_V_Start_1 - 1 then
581       if Hcount < HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia and
582           Vcount < VSYNC + VBACK_PORCH + C_V_Start_1 + ball_dia then
583         h_index_1 := Hcount - HSYNC - HBACK_PORCH - C_H_start_1;
584         v_index_1 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_1;
585         H_boundary := (others => '0');
586         H_boundary := C_boundary(TO_INTEGER(v_index_1));
587       if H_boundary(TO_INTEGER(h_index_1)) = '1' then
588         rectangle_1 <= '1';
589       elsif H_boundary(TO_INTEGER(h_index_1)) = '0' then
590         rectangle_1 <= '0';
591       end if;
592     elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia then
593       rectangle_1 <= '0';
594     end if;
595   elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_1 + ball_dia then
596     rectangle_1 <= '0';
597   end if;
598 end process RectangleHGen_1;
599
600 — BALL generator 2
601
602 RectangleHGen_2 : process (clk25)
603   variable H_boundary : unsigned(0 to 28);
604   variable h_index_2 : unsigned(9 downto 0);
605   variable v_index_2 : unsigned(9 downto 0);
606
607 begin
608   if rising_edge(clk25) then
609     if reset = '1' then
610       rectangle_2 <= '0';
611     elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_2 - 1 and
612           Vcount > VSYNC + VBACK_PORCH + C_V_Start_2 - 1 then
613       if Hcount < HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia and
614           Vcount < VSYNC + VBACK_PORCH + C_V_Start_2 + ball_dia then
615         h_index_2 := Hcount - HSYNC - HBACK_PORCH - C_H_start_2;
616         v_index_2 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_2;
617         H_boundary := (others => '0');
618         H_boundary := C_boundary(TO_INTEGER(v_index_2));
619       if H_boundary(TO_INTEGER(h_index_2)) = '1' then
620         rectangle_2 <= '1';
621       elsif H_boundary(TO_INTEGER(h_index_2)) = '0' then
622         rectangle_2 <= '0';
623       end if;
624     elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia then
625       rectangle_2 <= '0';
626     end if;
627   elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_2 + ball_dia then
628     rectangle_2 <= '0';
629   end if;
630 end process RectangleHGen_2;
631
632 — BALL generator 3
633
634 RectangleHGen_3 : process (clk25)
635   variable H_boundary : unsigned(0 to 28);
636   variable h_index_3 : unsigned(9 downto 0);

```

```

640  variable v_index_3 : unsigned(9 downto 0);
641 begin
642   if rising_edge(clk25) then
643     if reset = '1' then
644       rectangle_3 <= '0';
645     elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_3 - 1 and
646       Vcount > VSYNC + VBACK_PORCH + C_V_Start_3 - 1 then
647       if Hcount < HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia and
648         Vcount < VSYNC + VBACK_PORCH + C_V_Start_3 + ball_dia then
649         h_index_3 := Hcount - HSYNC - HBACK_PORCH - C_H_start_3;
650         v_index_3 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_3;
651         H_boundary := (others => '0');
652         H_boundary := C_boundary(TO_INTEGER(v_index_3));
653       if H_boundary(TO_INTEGER(h_index_3)) = '1' then
654         rectangle_3 <= '1';
655       elsif H_boundary(TO_INTEGER(h_index_3)) = '0' then
656         rectangle_3 <= '0';
657       end if;
658     elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia then
659       rectangle_3 <= '0';
660     end if;
661   elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_3 + ball_dia then
662     rectangle_3 <= '0';
663   end if;
664 end if;
665 end process RectangleHGen_3;

666 — BALL generator 4
667
668 RectangleHGen_4 : process (clk25)
669   variable H_boundary : unsigned(0 to 28);
670   variable h_index_4 : unsigned(9 downto 0);
671   variable v_index_4 : unsigned(9 downto 0);

672 begin
673   if rising_edge(clk25) then
674     if reset = '1' then
675       rectangle_4 <= '0';
676     elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_4 - 1 and
677       Vcount > VSYNC + VBACK_PORCH + C_V_Start_4 - 1 then
678       if Hcount < HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia and
679         Vcount < VSYNC + VBACK_PORCH + C_V_Start_4 + ball_dia then
680         h_index_4 := Hcount - HSYNC - HBACK_PORCH - C_H_start_4;
681         v_index_4 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_4;
682         H_boundary := (others => '0');
683         H_boundary := C_boundary(TO_INTEGER(v_index_4));
684       if H_boundary(TO_INTEGER(h_index_4)) = '1' then
685         rectangle_4 <= '1';
686       elsif H_boundary(TO_INTEGER(h_index_4)) = '0' then
687         rectangle_4 <= '0';
688       end if;
689     elsif Hcount >= HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia then
690       rectangle_4 <= '0';
691     end if;
692   elsif Hcount = HSYNC + HBACK_PORCH + C_H_start_4 + ball_dia then
693     rectangle_4 <= '0';
694   end if;
695 end if;
696 end process RectangleHGen_4;

697 — BALL generator 5
698
699 RectangleHGen_5 : process (clk25)
700   variable H_boundary : unsigned(0 to 28);
701   variable h_index_5 : unsigned(9 downto 0);
702   variable v_index_5 : unsigned(9 downto 0);

703 begin
704   if rising_edge(clk25) then
705     if reset = '1' then
706       rectangle_5 <= '0';
707     elsif Hcount > HSYNC + HBACK_PORCH + C_H_start_5 - 1 and
708       Vcount > VSYNC + VBACK_PORCH + C_V_Start_5 - 1 then
709       if Hcount < HSYNC + HBACK_PORCH + C_H_start_5 + ball_dia and
710         Vcount < VSYNC + VBACK_PORCH + C_V_Start_5 + ball_dia then
711         h_index_5 := Hcount - HSYNC - HBACK_PORCH - C_H_start_5;
712         v_index_5 := Vcount - VSYNC - VBACK_PORCH - C_V_Start_5;
713         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;

```

```

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 -- Socket generator 2
837
838 RectangleHGen_12 : process (clk25)
839   variable H_boundary : unsigned(0 to 28);
840   variable h_index_12 : unsigned(9 downto 0);
841   variable v_index_12 : unsigned(9 downto 0);
842
843 begin
844   if rising_edge(clk25) then
845     if reset = '1' then
846       rectangle_12 <= '0';
847     elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_12 - 1 and
848       Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_12 - 1 then
849       if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia and
850         Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_12 + ball_dia then
851         h_index_12 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_12;
852         v_index_12 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_12;
853         H_boundary := (others => '0');
854         H_boundary := C_boundary(TO_INTEGER(v_index_12));
855       if H_boundary(TO_INTEGER(h_index_12)) = '1' then
856         rectangle_12 <= '1';
857       elsif H_boundary(TO_INTEGER(h_index_12)) = '0' then
858         rectangle_12 <= '0';
859       end if;
860     elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_12 + ball_dia then
861       rectangle_12 <= '0';
862     end if;
863   end if;
864 end if;
865 end process RectangleHGen_12;
866
867 -- Socket generator 3
868
869 RectangleHGen_13 : process (clk25)
870   variable H_boundary : unsigned(0 to 28);
871   variable h_index_13 : unsigned(9 downto 0);
872   variable v_index_13 : unsigned(9 downto 0);
873
874 begin
875   if rising_edge(clk25) then
876     if reset = '1' then

```

```

880     rectangle_13      <= '0';
881   elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_13 - 1 and
882     Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_13 - 1 then
883     if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia and
884       Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_13 + ball_dia then
885       h_index_13 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_13;
886       v_index_13 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_13;
887       H_boundary := (others => '0');
888       H_boundary := C_boundary(TO_INTEGER(v_index_13));
889       if H_boundary(TO_INTEGER(h_index_13)) = '1' then
890         rectangle_13 <= '1';
891       elsif H_boundary(TO_INTEGER(h_index_13)) = '0' then
892         rectangle_13 <= '0';
893       end if;
894     elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia then
895       rectangle_13 <= '0';
896     end if;
897     elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_13 + ball_dia then
898       rectangle_13 <= '0';
899     end if;
900   end if;
901 end process RectangleHGen_13;
902
903 -- Socket generator 4
904
905 RectangleHGen_14 : process (clk25)
906   variable H_boundary : unsigned(0 to 28);
907   variable h_index_14 : unsigned(9 downto 0);
908   variable v_index_14 : unsigned(9 downto 0);
909
910 begin
911   if rising_edge(clk25) then
912     if reset = '1' then
913       rectangle_14 <= '0';
914     elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_14 - 1 and
915       Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_14 - 1 then
916       if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia and
917         Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_14 + ball_dia then
918         h_index_14 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_14;
919         v_index_14 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_14;
920         H_boundary := (others => '0');
921         H_boundary := C_boundary(TO_INTEGER(v_index_14));
922         if H_boundary(TO_INTEGER(h_index_14)) = '1' then
923           rectangle_14 <= '1';
924         elsif H_boundary(TO_INTEGER(h_index_14)) = '0' then
925           rectangle_14 <= '0';
926         end if;
927       elsif Hcount >= HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia then
928         rectangle_14 <= '0';
929       end if;
930     elsif Hcount = HSYNC + HBACK_PORCH + Socket_H_start_14 + ball_dia then
931       rectangle_14 <= '0';
932     end if;
933   end if;
934 end process RectangleHGen_14;
935
936 -- Socket generator 5
937
938 RectangleHGen_15 : process (clk25)
939   variable H_boundary : unsigned(0 to 28);
940   variable h_index_15 : unsigned(9 downto 0);
941   variable v_index_15 : unsigned(9 downto 0);
942
943 begin
944   if rising_edge(clk25) then
945     if reset = '1' then
946       rectangle_15 <= '0';
947     elsif Hcount > HSYNC + HBACK_PORCH + Socket_H_start_15 - 1 and
948       Vcount > VSYNC + VBACK_PORCH + Socket_V_Start_15 - 1 then
949       if Hcount < HSYNC + HBACK_PORCH + Socket_H_start_15 + ball_dia and
950         Vcount < VSYNC + VBACK_PORCH + Socket_V_Start_15 + ball_dia then
951         h_index_15 := Hcount - HSYNC - HBACK_PORCH - Socket_H_start_15;
952         v_index_15 := Vcount - VSYNC - VBACK_PORCH - Socket_V_Start_15;
953         H_boundary := (others => '0');
954         H_boundary := C_boundary(TO_INTEGER(v_index_15));
955         if H_boundary(TO_INTEGER(h_index_15)) = '1' then
956           rectangle_15 <= '1';
957         elsif H_boundary(TO_INTEGER(h_index_15)) = '0' then
958           rectangle_15 <= '0';
959         end if;

```

```

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
1041      begin
1042        if rising_edge(clk25) then
1043          if reset = '1' then
1044            rectangle_00      <= '0';
1045          elsif Hcount > HSYNC + HBACK_PORCH + cross_H - 1 and
1046            Vcount > VSYNC + VBACK_PORCH + cross_V - 1 then
1047            if Hcount < HSYNC + HBACK_PORCH + cross_H + cross_dia and
1048              Vcount < VSYNC + VBACK_PORCH + cross_V + cross_dia then
1049                h_index_00 := Hcount - HSYNC - HBACK_PORCH - cross_H;
1050                v_index_00 := Vcount - VSYNC - VBACK_PORCH - cross_V;
1051                H_boundary := (others => '0');
1052                H_boundary := cross_boundary(TO_INTEGER(v_index_00));
1053                if H_boundary(TO_INTEGER(h_index_00)) = '1' then
1054                  rectangle_00 <= '1';
1055                elsif H_boundary(TO_INTEGER(h_index_00)) = '0' then
1056                  rectangle_00 <= '0';
1057                end if;
1058              elsif Hcount >= HSYNC + HBACK_PORCH + cross_H + cross_dia then
1059                rectangle_00 <= '0';
1060              end if;
1061            elsif Hcount = HSYNC + HBACK_PORCH + cross_H + cross_dia then
1062              rectangle_00 <= '0';
1063            end if;
1064          end if;
1065        end process RectangleHGen_00;
1066
1067      —————— output
1068      VideoOut : process (clk25, reset)
1069
1070      begin
1071        if reset = '1' then
1072          VGA_R      <= "0000000000";
1073          VGA_G      <= "0000000000";
1074          VGA_B      <= "0000000000";
1075        elsif clk25'event and clk25 = '1' then
1076          if calibration = '1' then
1077            if rectangle_00 = '1' then
1078              VGA_R <= "1111111111";
1079              VGA_G <= "1111111111";
1080              VGA_B <= "1111111111";
1081            elsif stick = '1' then
1082              VGA_R <= "0000000000";
1083              VGA_G <= "0000000000";
1084              VGA_B <= "0000000000";
1085            elsif vga_hblank = '0' and vga_vblank = '0' then
1086              VGA_R <= "0000000000";
1087              VGA_G <= "1111111111";
1088              VGA_B <= "0000000000";
1089            else
1090              VGA_R <= "0000000000";
1091              VGA_G <= "0000000000";
1092              VGA_B <= "0000000000";
1093            end if;
1094          else
1095            if rectangle_00 = '1' then
1096              VGA_R      <= "1111111111";
1097              VGA_G      <= "1111111111";
1098              VGA_B      <= "1111111111";
1099            elsif ( Hcount >= HSYNC + HBACK_PORCH and Hcount < HSYNC + HBACK_PORCH + 641 and
1100              ((Vcount >= VSYNC + VBACK_PORCH and
1101                Vcount < VSYNC + VBACK_PORCH+ to_integer(black_b_y) + 1) or
1102                (Vcount > VSYNC + VBACK_PORCH+ 480 - to_integer(black_b_y) and
1103                  Vcount < VSYNC + VBACK_PORCH+480))) or
1104              ( Vcount >= VSYNC + VBACK_PORCH and Vcount < VSYNC + VBACK_PORCH + 480 and
1105                ((Hcount >= HSYNC + HBACK_PORCH and Hcount < HSYNC + HBACK_PORCH +
1106                  to_integer(black_b_x) + 1 ) or
1107                  (Hcount >= HSYNC + HBACK_PORCH+ 640 - to_integer(black_b_x)
1108                  and Hcount < HSYNC + HBACK_PORCH+641))) then
1109              VGA_R      <= "0000000000";
1110              VGA_G      <= "0000000000";
1111              VGA_B      <= "0000000000";
1112            elsif ( Hcount >= HSYNC + HBACK_PORCH and
1113              Hcount < HSYNC + HBACK_PORCH + 641 and
1114              ((Vcount >= VSYNC + VBACK_PORCH and
1115                Vcount < VSYNC + VBACK_PORCH +
1116                  to_integer(border_2)+ 1 - to_integer(margin)) or
1117                  (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        elsif ( 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_hsSync;
1208  VGA_VS     <= not vga_vsSync;
1209  VGA_SYNC   <= '0';
1210  VGA_BLANK  <= not (vga_hsSync or vga_vsSync);
1211
1212 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
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.numeric_std.all;
15
16 entity soundcontroller is
17
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(3 downto 0);
25     readdata : out unsigned(31 downto 0);
26     writedata : in unsigned(31 downto 0);
27     aud_xck : out std_logic;
28     aud_adclrk : out std_logic;
29     aud_adcdat : in std_logic;
30     aud_daclrk : out std_logic;
31     aud_dacd : out std_logic;
32     aud_bclk : inout std_logic
33 );
34 end soundcontroller;
35
36 architecture rtl of soundcontroller is
37
38 type ram_type is array(7 downto 0) of unsigned(31 downto 0);
39 signal RAM : ram_type;
40 signal ram_address : unsigned(2 downto 0);
41 signal counter : unsigned(31 downto 0);
42 signal audio_clock : unsigned(1 downto 0) := "00";
43 signal audio_request : std_logic;
44 signal audio_ctrl : std_logic := '0';
45
46 component de2_wm8731_audio port (
47     clk : in std_logic;           -- Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
48     reset_n : in std_logic;
49     test_mode : in std_logic;      -- Audio CODEC controller test mode
50     audio_request : out std_logic; -- Audio controller request new data
51     data : in unsigned(15 downto 0);
52
53     -- Audio interface signals
54     AUD_ADCLRCK : out std_logic;   -- Audio CODEC ADC LR Clock
55     AUD_ADCDAT : in std_logic;     -- Audio CODEC ADC Data
56     AUD_DACLRCK : out std_logic;   -- Audio CODEC DAC LR Clock
57     AUD_DACDAT : out std_logic;    -- Audio CODEC DAC Data
58     AUD_BCLK : inout std_logic;    -- Audio CODEC Bit-Stream Clock
59 );
60 end component;
61
62 begin
63     ram_address <= address(2 downto 0);
64
65     audio_clk_gen: process (clk)
66     begin
67         if rising_edge(clk) then
68             audio_clock <= audio_clock + "1";
69         end if;
70     end process audio_clk_gen;
71
72     audio_host_control: process (clk)
73     begin
74         if rising_edge(clk) then
75             if reset_n = '0' then
76
77                 else
78                     if chipselect = '1' then
79                         if read = '1' then
80
81                         end if;
82                     end if;
83                 end if;
84             end if;
85         end if;
86     end process audio_host_control;
87
88 end;

```

```

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  if audio_clock = "00" then
86      if RAM(0)(0) = '1' then
87          audio_ctrl <= '1';
88          RAM(0)(0) <= '0';
89      end if;

90      if audio_ctrl = '1' then
91          audio_ctrl <= '0';
92      end if;
93      end if;
94  end if;
95  end if;
96  end if;
97  end if;
98  end process audio_host_control;
99
100 -- audio_state_ctrl: process (clk)
101 begin
102     if rising_edge (clk) then
103         counter <= (others => '0');
104     else
105         if counter = 100 then
106             counter <= (others => '0');
107             reset_ctrl <= '0';
108         else
109             counter <= counter + 1;
110         end if;
111     end if;
112 end process audio_state_ctrl;
113
114 aud_xck <= audio_clock(1);
115
116 beeper: de2_wm8731_audio port map (
117     clk => audio_clock(1),
118     reset_n => '1',
119     test_mode => audio_ctrl,           — Output a sine wave
120     audio_request => audio_request,
121     data => "0000000000000000",
122     AUD_ADCLRCK => aud_adclrck,
123     AUD_ADCDAT => aud_adcdat,
124     AUD_DACLRCK => aud_daclrck,
125     AUD_DACDAT => aud_dacd,
126     AUD_BCLK => aud_bclk
127 );
128
129 end rtl;

```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity de2_wm8731_audio is
port (
    clk : in std_logic;          — Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;      — Audio CODEC controller test mode
    audio_request : out std_logic; — Audio controller request new data
    data : in unsigned(15 downto 0);
    — Audio interface signals
    AUD_ADCLRCK : out std_logic;   — Audio CODEC ADC LR Clock
    AUD_ADCDAT : in std_logic;     — Audio CODEC ADC Data
    AUD_DACLRCK : out std_logic;   — Audio CODEC DAC LR Clock
    AUD_DACDAT : out std_logic;    — Audio CODEC DAC Data
    AUD_BCLK : inout std_logic;    — Audio CODEC Bit-Stream Clock
);
end de2_wm8731_audio;

architecture rtl of de2_wm8731_audio is

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

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

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

signal shift_out : unsigned(15 downto 0);

signal sin_out : unsigned(7 downto 0);
signal sin_counter : unsigned(11 downto 0);
signal audio_on : std_logic;
begin
begin
    — LRCK divider
    — Audio chip main clock is 18.432MHz / Sample rate 48KHz
    — Divider is 18.432 MHz / 48KHz = 192 (X"00")
    — Left justify mode set by I2C controller

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

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

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

```

```

80      else
81          bclk_divider <= bclk_divider + 1;
82      end if;
83  end process;

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

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  -- 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
104  process (clk)
105  begin
106      if rising_edge(clk) then
107          if reset_n = '0' then
108              bclk <= '0';
109          elsif set_lrck = '1' or clr_bclk = '1' then
110              bclk <= '0';
111          elsif set_bclk = '1' then
112              bclk <= '1';
113          end if;
114      end if;
115  end process;

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;

134  -- Audio outputs
135
136     AUD_ADCLRCK <= lrck;
137     AUD_DACLRCK <= lrck;
138     AUD_DACDAT <= shift_out(15);
139     AUD_BCLK <= bclk;
140
142  -- Self test with Sin wave
143
144  process(clk)
145  begin
146      if rising_edge(clk) then
147          if reset_n = '0' then
148              sin_counter <= (others => '0');
149          elsif lrck_lat = '1' and lrck = '0' then
150              if sin_counter = x"36E" then
151                  sin_counter <= x"000";
152              elsif audio_on = '1' then
153                  sin_counter <= sin_counter + 1;
154              end if;
155          end if;
156      end if;
157  end process;

158  process(clk)
159  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" ,
242	x" 1f" when x" 041" ,
244	x" 26" when x" 042" ,
246	x" 32" when x" 043" ,
248	x" 21" when x" 044" ,
250	x" 71" when x" 045" ,
252	x" 4e" when x" 046" ,
254	x" d3" when x" 047" ,
256	x" a0" when x" 048" ,
258	x" fc" when x" 049" ,
260	x" f7" when x" 04A" ,
262	x" fd" when x" 04B" ,
264	x" f6" when x" 04C" ,
266	x" b8" when x" 04D" ,
268	x" f3" when x" 04E" ,
270	x" 26" when x" 04F" ,
272	x" 68" when x" 050" ,
274	x" 04" when x" 051" ,
276	x" 06" when x" 052" ,
278	x" 03" when x" 053" ,
280	x" 08" when x" 054" ,
282	x" 29" when x" 055" ,
284	x" 08" when x" 056" ,
286	x" 91" when x" 057" ,
288	x" 5f" when x" 058" ,
290	x" dc" when x" 059" ,
292	x" bb" when x" 05A" ,
294	x" e7" when x" 05B" ,
296	x" ea" when x" 05C" ,
298	x" bd" when x" 05D" ,
300	x" d7" when x" 05E" ,
302	x" 94" when x" 05F" ,
304	x" a6" when x" 060" ,
306	x" 86" when x" 061" ,
308	x" 89" when x" 062" ,
310	x" 81" when x" 063" ,
312	x" 84" when x" 064" ,
314	x" 76" when x" 065" ,
316	x" 7e" when x" 066" ,
318	x" 6c" when x" 067" ,
	x" 6f" when x" 068" ,
	x" 6f" when x" 069" ,
	x" 6b" when x" 06A" ,
	x" 78" when x" 06B" ,
	x" 74" when x" 06C" ,
	x" 80" when x" 06D" ,
	x" 7e" when x" 06E" ,
	x" 84" when x" 06F" ,
	x" 82" when x" 070" ,
	x" 84" when x" 071" ,
	x" 84" when x" 072" ,
	x" 83" when x" 073" ,
	x" 85" when x" 074" ,
	x" 82" when x" 075" ,
	x" 83" when x" 076" ,
	x" 7e" when x" 077" ,
	x" 80" when x" 078" ,
	x" 74" when x" 079" ,
	x" 7a" when x" 07A" ,
	x" 66" when x" 07B" ,
	x" 6d" when x" 07C" ,
	x" 5c" when x" 07D" ,
	x" 61" when x" 07E" ,
	x" 5a" when x" 07F" ,
	x" 5a" when x" 080" ,
	x" 60" when x" 081" ,
	x" 5c" when x" 082" ,
	x" 6f" when x" 083" ,
	x" 67" when x" 084" ,
	x" 86" when x" 085" ,
	x" 79" when x" 086" ,
	x" 9d" when x" 087" ,
	x" 92" when x" 088" ,
	x" ab" when x" 089" ,
	x" a6" when x" 08A" ,
	x" ad" when x" 08B" ,
	x" ae" when x" 08C" ,
	x" a5" when x" 08D" ,
	x" aa" when x" 08E" ,
	x" 94" when x" 08F" ,

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

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

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

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

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

720	x" 74" when x" 220" ,
722	x" 77" when x" 221" ,
	x" 73" when x" 222" ,
724	x" 87" when x" 223" ,
	x" 7e" when x" 224" ,
	x" 95" when x" 225" ,
726	x" 8f" when x" 226" ,
	x" 96" when x" 227" ,
728	x" 98" when x" 228" ,
	x" 8a" when x" 229" ,
730	x" 91" when x" 22A" ,
	x" 7f" when x" 22B" ,
732	x" 84" when x" 22C" ,
	x" 79" when x" 22D" ,
734	x" 7c" 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" 7f" when x" 23A" ,
	x" 90" when x" 23B" ,
748	x" 8a" when x" 23C" ,
	x" 9a" when x" 23D" ,
750	x" 96" when x" 23E" ,
	x" 9e" when x" 23F" ,
752	x" 9d" when x" 240" ,
	x" 99" when x" 241" ,
754	x" 9c" when x" 242" ,
	x" 89" when x" 243" ,
756	x" 92" when x" 244" ,
	x" 73" when x" 245" ,
758	x" 7f" when x" 246" ,
	x" 60" when x" 247" ,
760	x" 68" when x" 248" ,
	x" 58" when x" 249" ,
762	x" 5a" when x" 24A" ,
	x" 5d" when x" 24B" ,
764	x" 59" when x" 24C" ,
	x" 6f" when x" 24D" ,
766	x" 64" when x" 24E" ,
	x" 8c" when x" 24F" ,
768	x" 7c" when x" 250" ,
	x" a7" when x" 251" ,
770	x" 9b" when x" 252" ,
	x" af" when x" 253" ,
772	x" ae" when x" 254" ,
	x" a1" when x" 255" ,
774	x" ab" when x" 256" ,
	x" 80" when x" 257" ,
776	x" 92" when x" 258" ,
	x" 59" when x" 259" ,
778	x" 6c" when x" 25A" ,
	x" 3b" when x" 25B" ,
780	x" 48" when x" 25C" ,
	x" 3c" when x" 25D" ,
782	x" 37" when x" 25E" ,
	x" 5d" when x" 25F" ,
784	x" 49" when x" 260" ,
	x" 8d" 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" cd" 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" 4c" when x" 26F" ,

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

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

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

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

```

2 -- DE2 (Cyclone-II) Entity for Interactive Project Game
3 -- Authors:
4 --     Abdulhamid Ghandour
5 --     Thomas John
6 --     Jaime Poretzman
7 --     Bharadwaj Vellore
8 --
9 -- 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;

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;

```

```

80  end process reg_loader;
82  seven_segment_driver: process(clk)
83  begin
84    if rising_edge(clk) then
85      if reset_n = '0' then
86        -- do nothing
87      else
88        hex0(0) <= RAM(0)(0);
89        hex0(1) <= RAM(0)(1);
90        hex0(2) <= RAM(0)(2);
91        hex0(3) <= RAM(0)(3);
92        hex0(4) <= RAM(0)(4);
93        hex0(5) <= RAM(0)(5);
94        hex0(6) <= RAM(0)(6);
95        hex0(7) <= RAM(0)(7);
96
97        hex1(0) <= RAM(1)(0);
98        hex1(1) <= RAM(1)(1);
99        hex1(2) <= RAM(1)(2);
100       hex1(3) <= RAM(1)(3);
101       hex1(4) <= RAM(1)(4);
102       hex1(5) <= RAM(1)(5);
103       hex1(6) <= RAM(1)(6);
104       hex1(7) <= RAM(1)(7);
105
106       hex2(0) <= RAM(2)(0);
107       hex2(1) <= RAM(2)(1);
108       hex2(2) <= RAM(2)(2);
109       hex2(3) <= RAM(2)(3);
110       hex2(4) <= RAM(2)(4);
111       hex2(5) <= RAM(2)(5);
112       hex2(6) <= RAM(2)(6);
113       hex2(7) <= RAM(2)(7);
114
115       hex3(0) <= RAM(3)(0);
116       hex3(1) <= RAM(3)(1);
117       hex3(2) <= RAM(3)(2);
118       hex3(3) <= RAM(3)(3);
119       hex3(4) <= RAM(3)(4);
120       hex3(5) <= RAM(3)(5);
121       hex3(6) <= RAM(3)(6);
122       hex3(7) <= RAM(3)(7);
123
124       hex4(0) <= RAM(4)(0);
125       hex4(1) <= RAM(4)(1);
126       hex4(2) <= RAM(4)(2);
127       hex4(3) <= RAM(4)(3);
128       hex4(4) <= RAM(4)(4);
129       hex4(5) <= RAM(4)(5);
130       hex4(6) <= RAM(4)(6);
131       hex4(7) <= RAM(4)(7);
132
133       hex5(0) <= RAM(5)(0);
134       hex5(1) <= RAM(5)(1);
135       hex5(2) <= RAM(5)(2);
136       hex5(3) <= RAM(5)(3);
137       hex5(4) <= RAM(5)(4);
138       hex5(5) <= RAM(5)(5);
139       hex5(6) <= RAM(5)(6);
140       hex5(7) <= RAM(5)(7);
141
142       hex6(0) <= RAM(6)(0);
143       hex6(1) <= RAM(6)(1);
144       hex6(2) <= RAM(6)(2);
145       hex6(3) <= RAM(6)(3);
146       hex6(4) <= RAM(6)(4);
147       hex6(5) <= RAM(6)(5);
148       hex6(6) <= RAM(6)(6);
149       hex6(7) <= RAM(6)(7);
150
151       hex7(0) <= RAM(7)(0);
152       hex7(1) <= RAM(7)(1);
153       hex7(2) <= RAM(7)(2);
154       hex7(3) <= RAM(7)(3);
155       hex7(4) <= RAM(7)(4);
156       hex7(5) <= RAM(7)(5);
157       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
library ieee;
15 use ieee.std_logic_1164.all;
16 use ieee.numeric_std.all;
17
entity niostop is
18
port (
19   -- Clocks
20   CLOCK_27,                                -- 27 MHz
21   CLOCK_50,                                -- 50 MHz
22   EXT_CLOCK : in std_logic;                  -- External Clock
23
24   -- Buttons and switches
25   KEY : in std_logic_vector(3 downto 0);      -- Push buttons
26   SW : in std_logic_vector(17 downto 0);        -- DPDT switches
27
28   -- LED displays
29   HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
30   : out std_logic_vector(6 downto 0);           -- (active low)
31   LEDG : out std_logic_vector(8 downto 0);       -- Green LEDs (active high)
32   LEDR : out unsigned(17 downto 0);             -- Red LEDs (active high)
33
34   -- RS-232 interface
35   UART_TXD : out std_logic;                   -- UART transmitter
36   UART_RXD : in std_logic;                    -- UART receiver
37
38   -- IRDA interface
39   --IRDA_TXD : out std_logic;                 -- IRDA Transmitter
40   IRDA_RXD : in std_logic;                   -- IRDA Receiver
41
42   -- SDRAM
43
44   DRAMLDQ : inout std_logic_vector(15 downto 0); -- Data Bus
45   DRAMADDR : out std_logic_vector(11 downto 0);   -- Address Bus
46   DRAMLDQM,                                -- Low-byte Data Mask
47   DRAMUDQM,                                -- High-byte Data Mask
48   DRAM_WEN,                                 -- Write Enable
49   DRAM_CAS_N,                               -- Column Address Strobe
50   DRAM_RAS_N,                               -- Row Address Strobe
51   DRAM_CS_N,                                -- Chip Select
52   DRAM_BA0,                                 -- Bank Address 0
53   DRAM_BA1,                                 -- Bank Address 0
54   DRAM_CLK,                                 -- Clock
55   DRAM_LCKE : out std_logic;                -- Clock Enable
56
57   -- FLASH
58
59   FL_DQ : inout std_logic_vector(7 downto 0); -- Data bus
60   FL_ADDR : out std_logic_vector(21 downto 0); -- Address bus
61   FL_WE_N,                                  -- Write Enable
62   FL_RST_N,                                 -- Reset
63   FL_OE_N,                                  -- Output Enable
64   FL_CE_N : out std_logic;                  -- Chip Enable
65
66   -- SRAM
67
68   SRAMLDQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
69   SRAMADDR : out std_logic_vector(17 downto 0);   -- Address bus 18 Bits
70   SRAMUB_N,                                 -- High-byte Data Mask
71   SRAMLB_N,                                 -- Low-byte Data Mask
72
73
74

```

```

80  SRAM_WE_N,                                — Write Enable
81  SRAM_CE_N,                                — Chip Enable
82  SRAM_OE_N : 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_FSPEED,                                 — USB Full Speed, 0 = Enable, Z = Disable
93  OTG_LSPEED : 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

109 -- SD card interface

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

116 -- USB JTAG link

118  TDI,                                       — CPLD -> FPGA (data in)
119  TCK,                                       — CPLD -> FPGA (clk)
120  TCS : in std_logic;                     — CPLD -> FPGA (CS)
121  TDO : out std_logic;                    — FPGA -> CPLD (data out)

123 -- I2C bus

126  I2C_SDAT : inout std_logic; — I2C Data
127  I2C_SCLK : out std_logic;   — I2C Clock

129 -- PS/2 port

130  PS2_DAT,                                    — Data
131  PS2_CLK : in std_logic;                  — Clock

133 -- VGA output

136  VGA_CLK,                                    — Clock
137  VGA_HS,                                     — HSYNC
138  VGA_VS,                                     — VSYNC
139  VGA_BLANK,                                 — BLANK
140  VGA_SYNC : out std_logic;                — SYNC
141  VGA_R,                                      — Red[9:0]
142  VGA_G,                                      — Green[9:0]
143  VGA_B : out std_logic_vector(9 downto 0); — Blue[9:0]

145 -- Ethernet Interface

146  ENET_DATA : inout unsigned(15 downto 0); — DATA bus 16 Bits
147  ENET_CMD,                                   — Command/Data Select, 0 = Command, 1 = Data
148  ENET_CS_N,                                 — Chip Select
149  ENET_WR_N,                                 — Write
150  ENET_RD_N,                                 — Read
151  ENET_RST_N,                                — Reset
152  ENET_CLK : out std_logic;                — Clock 25 MHz
153  ENET_INT : in std_logic;                  — Interrupt

155 -- Audio CODEC

158  AUD_ADCLRCK : inout std_logic;           — ADC LR Clock
159  AUD_ADCDAT : in std_logic;                — ADC Data

```

```

160 AUD_DACLRCK : inout std_logic;           — DAC LR Clock
161 AUD_DACDAT : 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,                                — HSYNC
169   TD_VS : in std_logic;                  — VSYNC
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   niossystem: entity work.pool port map (
210     clk => CLOCK_50,
211     reset_n => KEY(0),
212
213     — the_sram
214     SRAM_ADDR_from_the_sram => SRAMADDR,
215     SRAM_CE_N_from_the_sram => SRAMCE_N,
216     SRAM_DQ_to_and_from_the_sram => SRAMDQ,
217     SRAM_LB_N_from_the_sram => SRAMLB_N,
218     SRAM_OE_N_from_the_sram => SRAMOE_N,
219     SRAM_UB_N_from_the_sram => SRAMUB_N,
220     SRAM_WE_N_from_the_sram => SRAMWE_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),
241    cal_direction_from_the_vision(4) => LEDR(15),
242    cal_direction_from_the_vision(5) => LEDR(16),
243    cal_direction_from_the_vision(6) => LEDR(17),
244    pixel_clk_to_the_vision => GPIO_1(10),
245    pixel_data_to_the_vision(0) => GPIO_1(0),
246    pixel_data_to_the_vision(1) => GPIO_1(1),
247    pixel_data_to_the_vision(2) => GPIO_1(5),
248    pixel_data_to_the_vision(3) => GPIO_1(3),
249    pixel_data_to_the_vision(4) => GPIO_1(2),
250    pixel_data_to_the_vision(5) => GPIO_1(4),
251    pixel_data_to_the_vision(6) => GPIO_1(6),
252    pixel_data_to_the_vision(7) => GPIO_1(7),
253    pixel_data_to_the_vision(8) => GPIO_1(8),
254    pixel_data_to_the_vision(9) => GPIO_1(9),
255    threshold_to_the_vision => SW(9 downto 0),
256    vision_flags_from_the_vision => vision_flags_signal,
257
258    -- the_audio
259    aud_adcdat_to_the_sounddriver => AUD_ADCDAT,
260    aud_adclrck_from_the_sounddriver => AUD_ADCLRCK,
261    aud_bclk_to_and_from_the_sounddriver => AUD_BCLK,
262    aud_dacdat_from_the_sounddriver => AUD_DACDAT,
263    aud_daclrck_from_the_sounddriver => AUD_DACLRCK,
264    aud_xck_from_the_sounddriver => AUD_XCK,
265
266    -- the_lcd
267    LCD_E_from_the_lcd => LCD_EN,
268    LCD_RS_from_the_lcd => LCD_RS,
269    LCD_RW_from_the_lcd => LCD_RW,
270    LCD_data_to_and_from_the_lcd => LCD_DATA,
271
272    -- the_uicontrol
273    hex0_from_the_uicontrol(6 downto 0) => HEX0,
274    hex1_from_the_uicontrol(6 downto 0) => HEX1,
275    hex2_from_the_uicontrol(6 downto 0) => HEX2,
276    hex3_from_the_uicontrol(6 downto 0) => HEX3,
277    hex4_from_the_uicontrol(6 downto 0) => HEX4,
278    hex5_from_the_uicontrol(6 downto 0) => HEX5,
279    hex6_from_the_uicontrol(6 downto 0) => HEX6,
280    hex7_from_the_uicontrol(6 downto 0) => HEX7,
281    key_to_the_uicontrol => KEY,
282    switch_to_the_uicontrol => SW,
283
284    -- the_camera
285    sclk_from_the_camera => int_sclk,
286    sdat_to_and_from_the_camera => int_sdat,
287    ack_to_the_camera => GPIO_1(15)
288 );
289
290 frame_counter: process (GPIO_1(13))
291 begin
292     if rising_edge(GPIO_1(13)) then
293         if reset_n = '0' then
294             frameCount <= x"00000000";
295         else
296             if stop_counter = '1' then
297                 frameCount <= frameCount + 1;
298             end if;
299         end if;
300     end if;
301 end process;
302
303 tick_counter: process (clk25)
304 begin
305     if rising_edge(clk25) then
306         if reset_n = '0' then
307             stop_counter <= '0';
308             tickCount <= x"00000000";
309         else
310             if stop_counter = '1' then
311                 tickCount <= tickCount + 1;
312                 if(tickCount > x"05f5e100") then
313                     stop_counter <= '1';
314                 end if;
315             end if;
316         end if;
317 
```

```

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);

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';
401 SD_CLK <= '1';
402
403 UART_TXD <= '0';
404 DRAM_ADDR <= (others => '0');
405 DRAM_LDQM <= '0';
406 DRAM_UDQM <= '0';
407 DRAM_WEN <= '1';
408 DRAM_CAS_N <= '1';
409 DRAM_RAS_N <= '1';
410 DRAM_CS_N <= '1';
411 DRAM_BA_0 <= '0';
412 DRAM_BA_1 <= '0';
413 DRAM_CLK <= '0';
414 DRAM_CKE <= '0';
415 FL_WE_N <= '1';
416
417 FL_OE_N <= '1';
418 FL_CE_N <= '1';
419 OTG_ADDR <= (others => '0');
420 OTG_CS_N <= '1';
421 OTG_RD_N <= '1';
422 OTG_RD_N <= '1';
423 OTG_WR_N <= '1';
424 OTG_RST_N <= '1';
425 OTG_FSPEED <= '1';
426 OTG_LSPEED <= '1';
427 OTG_DACK0_N <= '1';
428 OTG_DACK1_N <= '1';
429
430 ENET_CMD <= '0';
431 ENET_CS_N <= '1';
432 ENET_WR_N <= '1';
433 ENET_RD_N <= '1';
434 ENET_RST_N <= '1';
435 ENET_CLK <= '0';
436
437 TDO <= '0';
438 TD_RESET <= '0';
439
440 — Set all bidirectional ports to tri-state
441 DRAM_DQ <= (others => 'Z');
442 FL_DQ <= (others => 'Z');
443 OTG_DATA <= (others => 'Z');
444 ENET_DATA <= (others => 'Z');
445
446 end datapath;

```