Interactive Projection Game
CSEE 4840 Project
Design Document

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

March 14, 2008

Contents

1 **Introduction** ***3***
2 **Top Level Design** ***4***
2.1 *System Configuration* *4*
2.2 *Functional Description* *6*
3 **Camera Interface** ***7***
3.1 *Camera Physical Interface* *7*
3.2 *Camera Register Configuration* *7*
3.3 *Hardware-Software partitioning* *7*
3.4 *Pixel Timing* *7*
3.4.1 Preliminary Timing Estimates *9*
3.4.2 Implications and Choices *10*
3.4.3 I^2C Interface *10*
4 **Vision-Input Processing Module** ***11***
4.1 *Interfaces* *11*
4.2 *Algorithm* *12*
4.2.1 Tracking side boundaries *12*
4.2.2 Tracking top and bottom boundaries *13*
4.3 *Implementation* *14*
5 **Software Design** ***16***
5.1 *Calibration* *16*
5.2 *Algorithm - Single Ball* *16*
6 **VGA Controller Module** ***19***
7 **Open Points** ***20***
8 **Project Management** ***21***
8.1 *Versioning* *21*
8.2 *Implementation Milestones* *21*
9 **Glossary of Terms** ***22***

List of Figures

1 Board Level Connection *4*
2 Physical Component Organization *5*
3 Vision System Block Diagram *11*
4 Vision System IO Timing *12*
5 Vision System Algorithm Illustration - Cue cuts Left and Right Boundaries *12*
6 Vision System Algorithm Illustration - Cue cuts Top and Bottom Boundaries *14*
7 Vision System Detailed Diagram *15*
8 Calibration Algorithm *17*
9 Primary Ball-Positioning Algorithm *18*
10 Ball movements and collisions *19*
11 Directory Tree Structure *21*

List of Tables

1 TRDB-DC2 Register Settings *8*
2 Camera Interface - Control/Status Register *9*
3 Camera Interface - Register List *9*

1 Introduction

The purpose of this document is to present a detailed design of the components in the ”Interactive Project Game” system. ”Interactive Projection Game” is a virtual pool-like game designed using vision and projection techniques. Game play is based on a projected image of a pool-table-like surface, with a ball positioned on it. A player can then use a real cue or cue-like object to ’strike’ the ball. The ball is then projected in the direction it was struck and made to settle at a new final position from where the player can strike it once again. The images are projected using a projector that receives a VGA input, and a camera is used to capture the projected image and the position and motion of the cue-stick.

In this document, component internals are detailed, as are the interfaces between the components in both physical and logical terms. This document is based on the earlier proposal document for this system which offers a very high level perspective of the data-flow path in the system.

Details of the design are listed starting from a top level and descending to each component by turn. In particular, the interface with the external camera device is elucidated, as is the pixel processing module. The final sections deal with the VGA controller module and a high-level view of the software design.

2 Top Level Design

2.1 System Configuration

The ”Interactive Projection Game”, referred to hereafter as IPG, system is built out of a combination of hardware and software components. The system is centred around a NIOS-2 processor[3], 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[2] family FPGA from Altera.

The IPG systems 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 1.

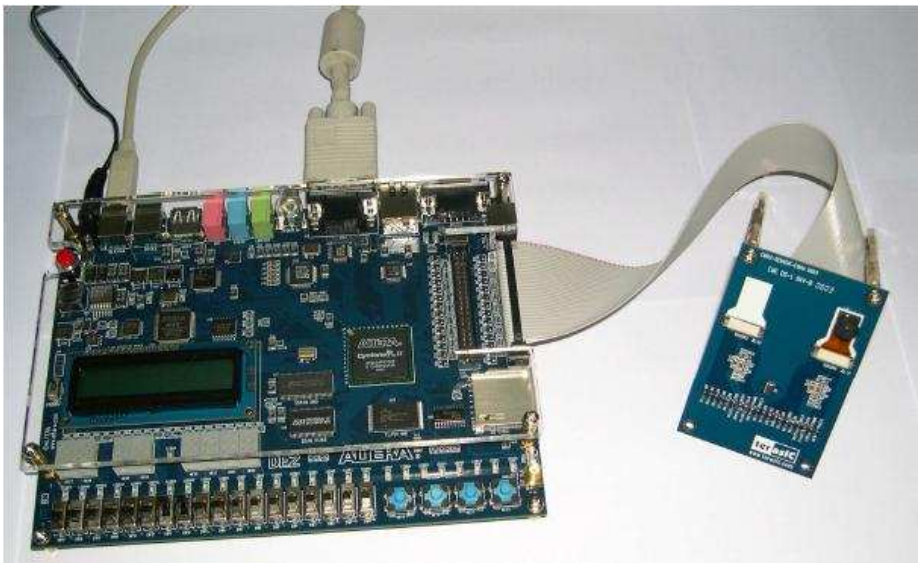

Figure 1: Board Level Connection

The NIOS-II core is connected in the IPG system to the peripherals shown in Figure 2. The connection to the peripherals is via the Avalon[1] system interconnect fabric. The processor interface with an SDRAM, out of which the software for the system is executed. A JTAG module attached to the processor enables debugging. In addition to these standard peripherals, the following custom peripherals are created in hardware on the said FPGA and attached to the Avalon bus in this design. Each of these peripherals is an Avalon slave component. The NIOS-II core is the lone master.

- Camera Interface
- Pixel Processor
- VGA Driver

Each of the listed peripherals offers a programmable register (MMIO) view to software running on the NIOS-II. This enables software to configure these peripherals, track their status, and co-ordinate their activities.

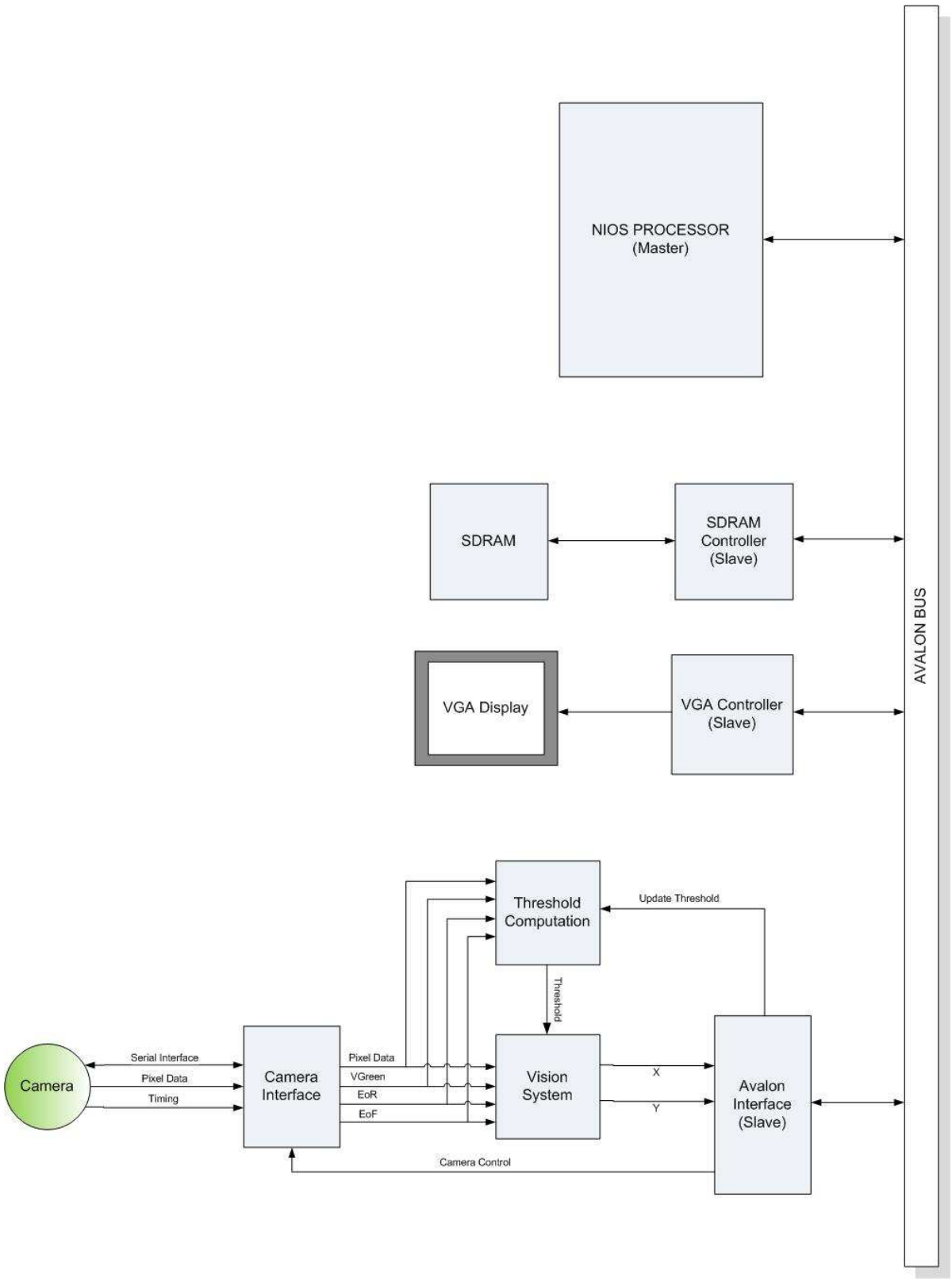

Figure 2: Physical Component Organization

2.2 Functional Description

Figure 2 also illustrates the data-flow and the processing steps undertaken to achieve the required functionality. The processing steps are briefly visited here and explored in detail in future sections.

- The camera interface configures the camera, and then receives images from it at a frame rate and resolution agreed with the camera. The captured image is then forwarded to the pixel processing module one pixel at a time, as and when pixel arrive. These are accompanied by control signals which help the pixel processing module synchronize with frame timing.
- The pixel processing module analyses incoming pixels to determine the position of the tip of the cue-stick. Pixels are processed as they arrive and discarded. An algorithm within this module does suitable book-keeping to enable the position of the tip to be determined. When a frame has been processed and a new position is available, the position is forwarded to a ball-dynamics simulator.
- The ball-dynamics simulator is a piece of software that uses cue-stick position information to calculate the angle and velocity of impact on a ball placed at a known location. It then computes the trajectory of the ball and generates a series of images using a VGA controller module.
- The VGA controller module is a piece of hardware which draws out images of a ball in motion (at video refresh rates) and drives a VGA output that is supplied to a projector.
- A calibration system is implemented in software to synchronize the projected image and the image as perceived by the camera.

3 Camera Interface

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[4], which is able to capture frames at SXGA, VGA and CIF resolutions at close-to-video refresh rates.

3.1 Camera Physical Interface

The camera, a TRDB-DC2 from Terasic[5], interfaces with the board via a 40-pin flat cable as illustrated in Figure 1. 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 [5].

3.2 Camera Register Configuration

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

3.3 Hardware-Software partitioning

To enable flexibility in configuring various parameters in the camera, the configurations are chosen in and set in software running on then NIOS processor. The I^2C controller in hardware is an Avalon slave and an I^2C master! It received configuration settings from software and purely implements the physical communication with the camera.

The configuration happens via a tiny handshake protocol implemented between hardware and software. This handshake is through a Status/Control register in the I^2C controller. The details of the register are as in the Table 2.

Together with the Status/Control register, there are two other registers that are available to be read from/written to by software. Table 3 details their names and purpose.

3.4 Pixel Timing

This section presents some numbers on the timing given the configuration of registers that has been presented earlier. This calculation is based on the selected number of horizontal and vertical active video

Table 1: TRDB-DC2 Register Settings

Register	Offset	Default	Configured	Notes
Chip Version	0x00	0x1433	-	Read Only
Row Start	0x01	0x000C	0x000C	There are 8 dark rows and 4 rows skipped to allow for boundary effects
Column Start	0x02	0x001E	0x001E	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	0x0432	Unchanged
Row Speed	0x0A	0x0001	0x0001	Unchanged
Extra Delay	0x0B	0x0000	0x0000	To be defined
Shutter Delay	0x0C	0x0000	0x0000	To be defined
Reset	0x0D	0x0008	0x0008	Unchanged
FRAME_VALID Control	0x1F	0x0000	0x0000	To be defined
Read Mode - Context B	0x20	0x0020	0x0020	To be defined
Read Mode - Context A	0x21	0x040C	0x040C	Unused
Show Control	0x22	0x0129	0x0129	Unchanged
Flash Control	0x23	0x0608	0x0608	Unchanged
Green 1 Gain	0x2B	0x0020	0x0020	To be defined
Blue Gain	0x2C	0x0020	0x0020	Unchanged
Red Gain	0x2D	0x0020	0x0020	Unchanged
Green 2 Gain	0x2E	0x0020	0x0020	To be defined
Global Gain	0x2F	0x0020	0x0020	To be defined
Context Control	0xC8	0x000B	0x000B	Unchanged

pixels and the number of horizontal and vertical blanking pixels. These are programmable in the sensor. the sensor always produced images in progressive scan. Also, at the start of each line, it generates a LINE_VALID signal, and at the start of each new frame, a FRAME_VALID. Information for each pixel is 10 bits wide and is sent with a pixel clock whose frequency is a function of the window size and the frame rate.

The design will use a VGA resolution of $640*480$ active video pixels at video-like frame rates. This latter choice is based on the need for near error free detection of the movement of the cue stick. The combination of the window size and the frame rate dictates the pixel clock frequency or pixel readout

Table 2: Camera Interface - Control/Status Register

Register Value	Description
0	Indicates that the I^2C bus is idle and that therefore, the I^2C controller is able to take commands from software to begin a new send or receive over the bus. This is the initial state of the register. This is also the state to which the register is restored by hardware each time it completes a send or receive.
1	Indicates that a receive is to be executed or is in execution. This is the value to which the software must set this register to initiate a receive from the camera.
2	Indicate that a send is to be executed or is in execution. This is the value to which the software must set this register to initiate a send to the camera.
>= 3	Invalid

Table 3: Camera Interface - Register List

Register Name	Default Value	Description
REG	0	Holds the address of the register to which a value must be written or from which a value must be read. For reliable operation, this register should be written to by software only when the Status/Control register is ”0”. The value in this register should be valid before a send is initiated by switching the Status/Control register from ”0” to ”2” or a receive is initiated by switching the value from ”0” to ”1”.
DATA	0	Holds the data which must be written or has been read from a register in the camera. For reliable operation, this register should be read from or written to by software only when the Status/Control register is ”0”. The value in this register should be valid before a send is initiated by switching the Status/Control register from ”0” to ”2”. Also the value in this register is valid following a receive only once the hardware has swtched the Status/Control register from ”1” to ”0”. The width of this register is 16 bits.

rate. This read-out rate is constrained by the pixel processing algorithm that the camera interface feeds. The current design aims to eliminate the need for storage of pixels by processing each pixel as it arrives from the camera. The processing of each pixel is complete in a single pixel-clock, following which the pixel is discarded. Evidently, the pixel clock therefore needs to be influenced also by the timerequired to complete processing one pixel.

3.4.1 Preliminary Timing Estimates

Preliminary back-of-the-envelope calculations throw of the following numbers, which indicate that the design is very much feasible at the desired frame rate and window size.

In this design, both ADCs available in the camera are used at all times. Each ADC quantizes at half

the pixel clock frequency; full resolution images are therefore capturesd at all times. A consequence of this is that the camera is configures to perpetually operate in a single context (context B). No rows or columns are ever skipped, and the Bayer pattern is fully preserved.

Master Clock Frequency $= 25MHz$
Number of horizontal active pixels $(A) = 640$
Number of rows of active pixels $(N) = 480$
Number of pixels of horizontal blanking $(Q) = 202$
Number of rows of vertical blanking $(V) = 25$
Frame time $= (A + Q)(N + V) * PxlClkTime$

For a refresh rate of $25fps$, frame time $= 40ms$
Therefore,
PxlClkFreq = 10.63 MHz $\approx 12.5MHz$

3.4.2 Implications and Choices

The above result implies that the pixel clock frequency can be configured to be equal to half the master clock frequency. Further, the modules in the FPGA (including the pixel processsing module are clocked at $50MHz$. This translates to a 4-clock cycle interval for the entire pixel processing chain to operate on each pixel. Clearly, should the algorithm take fewer cycles to complete, the resolution and/or the frame rate may be scaled update to reduce error. The improved resolution may be particularly useful should the area of projection of the picture be large.

3.4.3 I^2C Interface

The configuration of the registers happens through an I^2 interface which 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
- 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

4 Vision-Input Processing Module

The Vision System is a hardware block which processes the input from the camera to identify the tip of the cue stick.

4.1 Interfaces

The interface signals to this block are shown in Figure 3.

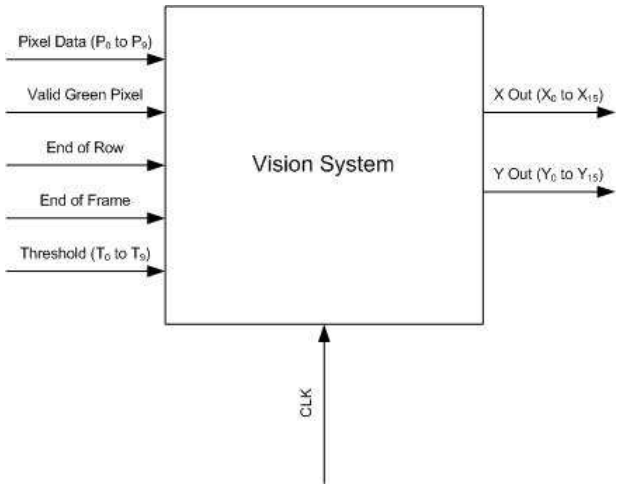

Figure 3: Vision System Block Diagram

The Pixel_Data input is the 10-bit color data from the camera. The camera uses a Bayer color system, with every alternate pixel being a green pixel. Since the camera and the vision system operate at different frequencies, a Valid_Green signal is asserted for a period of one clock cycle to indicate when the Pixel_Data input has new green data. Figure 4 illustrates the timing of the signals mentioned here.

The End_of_Row signal is asserted at the end of one row of pixel data. Similarly, End_of_Frame is asserted for a period of one clock period 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 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 typically calculated during system reset and is kept constant during operation.

X_Out and Y_Out are 16-bit values which provide the position of the tip of the cue stick. 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 latches X_Out and Y_Out are updated everytime End_of_Frame is asserted with the value computed during the previous frame.

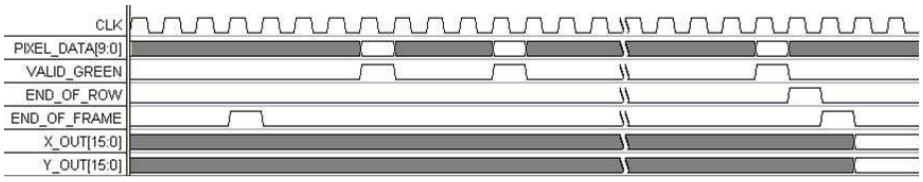

Figure 4: Vision System IO Timing

4.2 Algorithm

The cue-stick blocks out the light from the display and is registered as pixels with color value less than threshold by the camera. The cue-stick can enter the frame from any one of the four sides. The computation is done by breaking down these possible cases into two:

1. the cue stick enters the frame from the top or bottom
2. the cue-stick enters the frame from either side

4.2.1 Tracking side boundaries

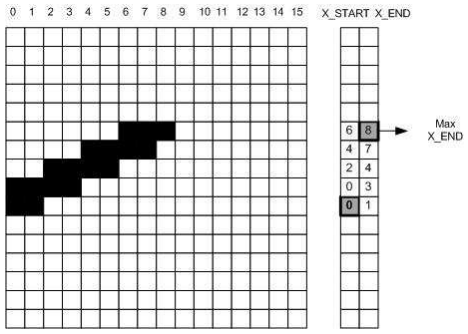

Figure 5: Vision System Algorithm Illustration - Cue cuts Left and Right Boundaries

The information in the frame can be fully represented by the start and end co-ordinates of the cue stick on each row of the frame. If the cue-stick enters the frame from the left, then the maximum value of the end co-ordinates and the corresponding y-co-ordinate gives the location of the tip. On the other hand, if the cue-stick enters the frame from the right, then the minimum value of the start co-ordinates and the corresponding y co-ordinate gives the location of the tip.

The pixel color data is input to the system from left to right, row by row. Scanning each value, we register the cue stick if a pre-defined number CUEWIDTH of contiguous dark pixels are identified. The start and end co-ordinates of the dark pixels are identified in this manner.

As soon as this identification is done, it is compared with the current Min and Max values. If the start co-ordinate for the current row is less than all previous start co-ordinates, the Min value is updated. The same applied for the Max value.

The pseudo-code below demonstrates this in greater detail. Figure 5 illustrates exactly the steps followed in the algorithm, together with the manner in which the minimum and maximum start and end positions are maintained and updated together with their y co-ordinates.

Algorithm 1 Finds the tip of a cue stick from an image map when cue enters from sides

```
1:  MinStartVal ← X_MAX + 1
2:  MaxEndVal ← 0
3:  repeat
4:     DarkPixelCounter ← 0
5:     repeat
6:        Get Pixel Colour
7:        if colour < THRESHOLD then
8:           DarkPixelCounter++
9:        end if
10:       if DarkPixelCounter = CUEWIDTH then
11:          Save X_START
12:          Save X_END
13:       end if
14:       if X_START < MinStartVal then
15:          MinStartVal ← X_START
16:          Save y {y is the y-co-ordinate corresponding to MinStartVal}
17:       end if
18:       if X_END > MaxEndVal then
19:          MaxEndVal ← X_END
20:          Save y {y is the y-co-ordinate corresponding to MaxEndVal}
21:       end if
22:    until EndOfLine = 1
23: until EndOfFrame = 1
24:
25: if MinStartVal = 0 then
26:    return (MaxEndVal, y) {Cue enters frame from left}
27: end if
28: if MaxEndVal = X_MAX then
29:    return (MinStartVal, y) {Cue enters frame from right}
30: end if
```

4.2.2 Tracking top and bottom boundaries

If the cue stick enters the frame from the top of bottom, the tip of the cue can be identified by tagging each row depending on whether the procedure given above registers the cue stick or not as shown in Figure 6. The point where this tag changes gives the tip of the cue stick. The X co-ordinate is obtained from X_START computed for that row in the previous section.

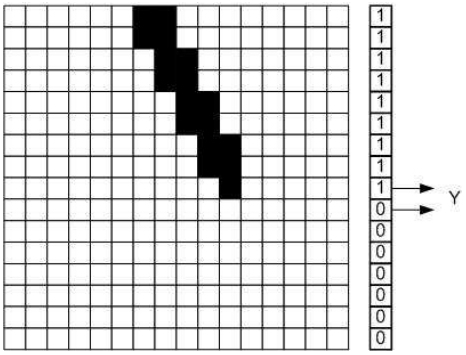

Figure 6: Vision System Algorithm Illustration - Cue cuts Top and Bottom Boundaries

4.3 Implementation

The block diagram of the module is shown in Figure 7. The computation is performed on the fly as each pixel data comes in. Further, the pixels are not stored in a frame buffer, eliminating the need for memory. At the end of each frame, a decision is made as to which of the x, y co-ordinates from case a or b above is to be output. The output latches are updated when Frame Buffer is asserted.

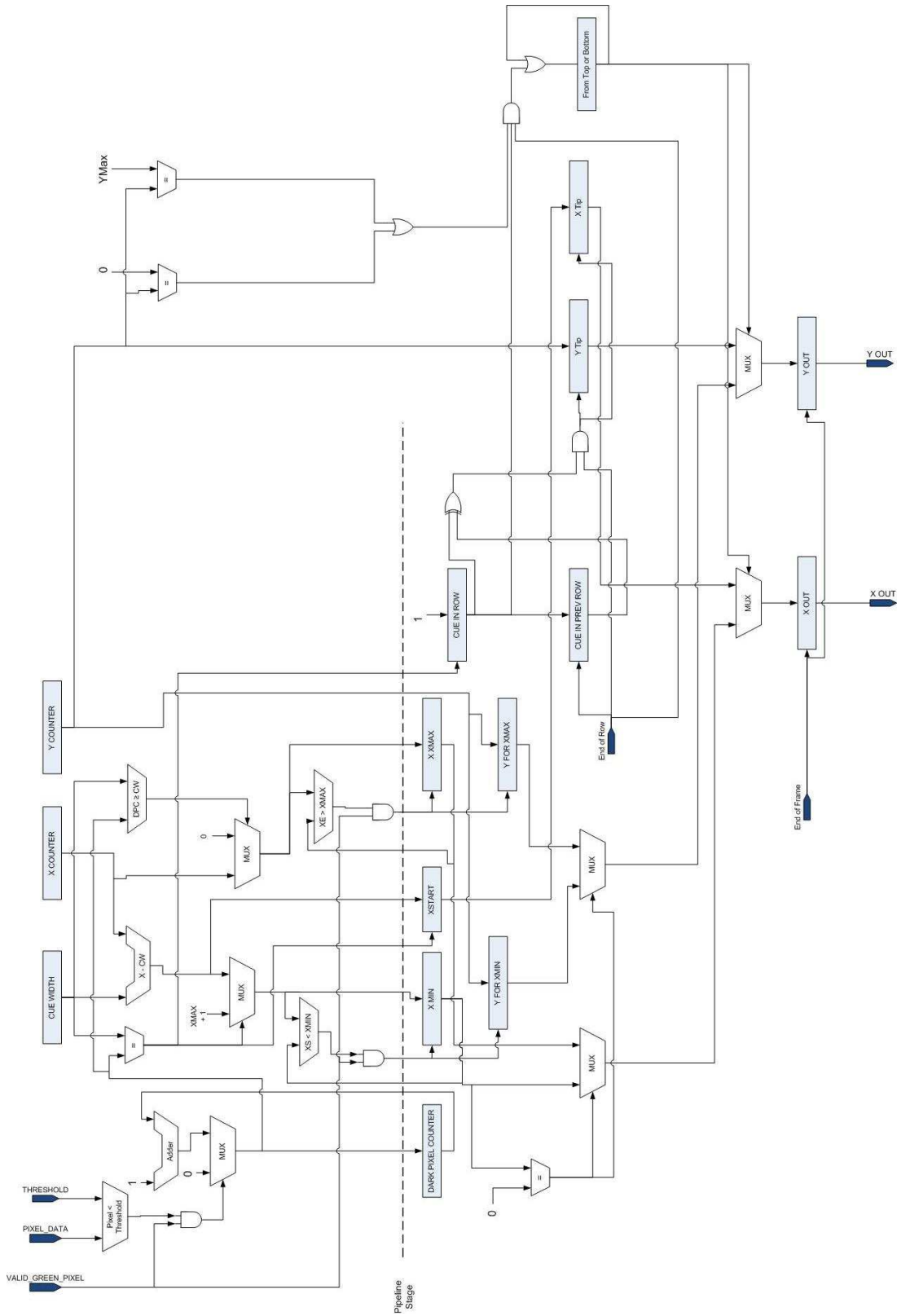

Figure 7: Vision System Detailed Diagram

5 Software Design

This section details the two primary tasks of the software running on the NIOS-II processor. The software is to be written entirely in 'C'. During and after start-up of the application, the software performs these tasks:

- Configure the camera
- Calibrate the system
- Forever, process inputs from the vision system, compute new frame information and provide inputs to the VGA controller

5.1 Calibration

The calibration of the system will be done by using four known frames. Each frame will consist of a cue-stick positioned to be intersecting the frame from a different side (left, right, top, bottom) and will delimit the edge of the table. The software part of the calibration will consist of sending three bits to the VGA controller. The first two bits will choose the frame that has to be projected on the screen, while the third bit will specify the mode of use (calibration, normal).

Depending on the screen that is being displayed, an X sample (Xs) and a Y sample (Ys) will be compared with the incoming bits from the edge detection algorithm. In case that the incoming X1 and Y1 match within a threshold of Xs1 and the Ys1, the second frame will be projected. The calibration algorithm will hold its current state until the X2 and Y2 values arrive. This procedure is repeated until all four frames are matched consecutively. In case the frames do not match, the projector or camera has to be calibrated manually until they match; once they are matched, the software will confirm the normal mode bit to the VGA controller and the calibration procedure will be terminated.

5.2 Algorithm - Single Ball

After the calibration has terminated and the normal mode has been started, the one-ball algorithm will begin reading X and Y values from the edge detection hardware. The X and Y coordinates will be stored in a FIFO queue, where a total number of three coordinates will be stored. Using these coordinates, an estimation of the velocity and location of the stick will be calculated. In case a collision between the stick and the ball is estimated, the direction and velocity of the ball will be calculated.

Once the ball has an initial direction and velocity, a forecast of a ball to wall collision is computed. If there is no imminent collision, the next location and velocity of the ball will be recalculated. In case a collision is forecast, the ball direction will be adjusted to create the effect of hitting the wall and then, the location and the velocity of the ball will be recalculated. In either case, the ball will be damped and the new location will be sent to the VGA controller. Once the ball comes to a complete stop, the algorithm is restarted.

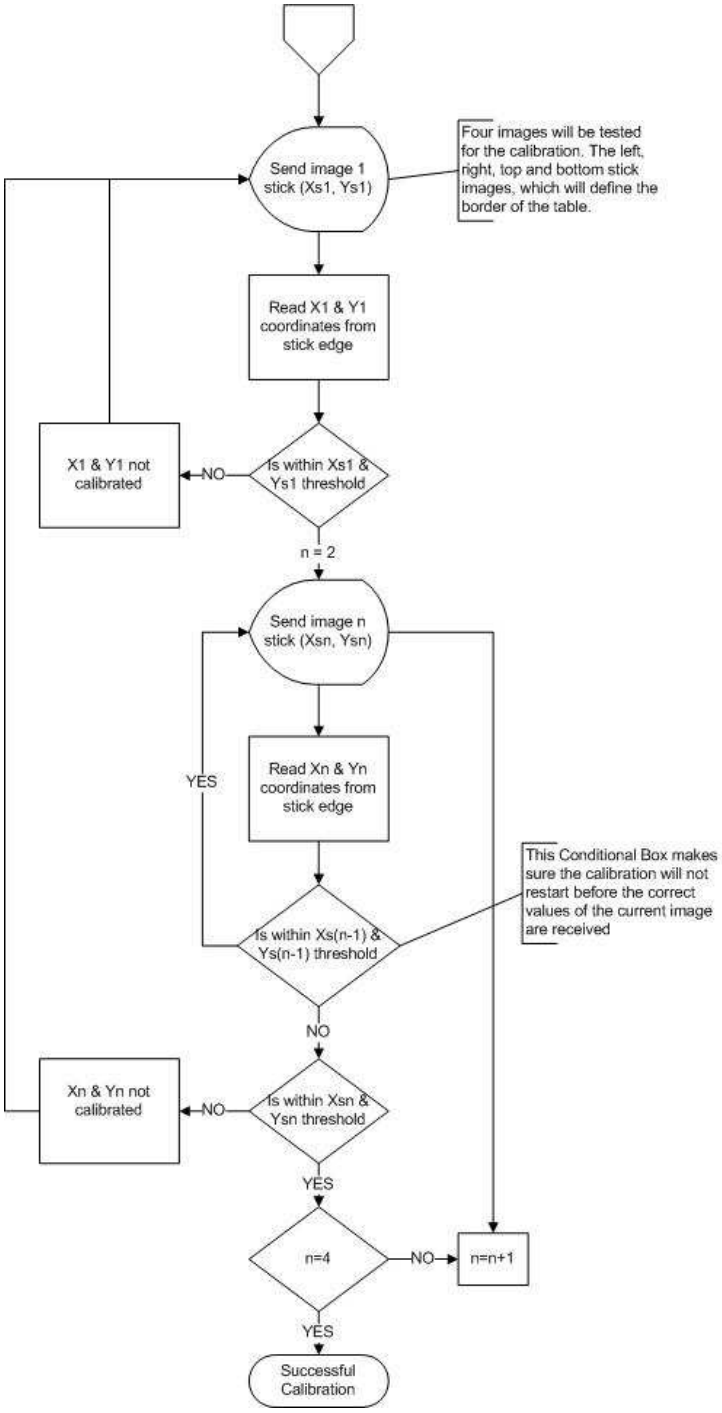

Figure 8: Calibration Algorithm

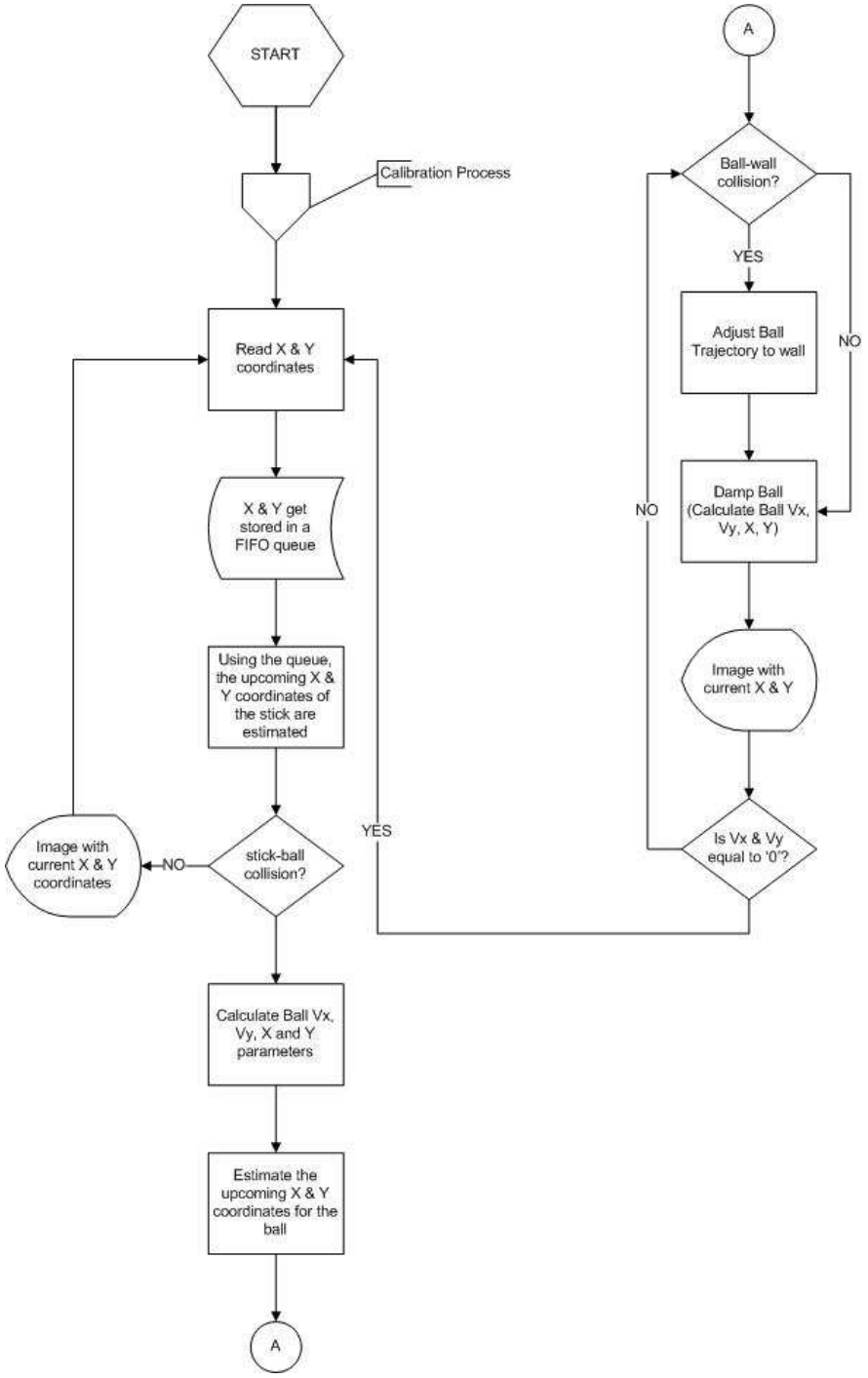

Figure 9: Primary Ball-Positioning Algorithm

6 VGA Controller Module

The VGA Controller is an Avalon component that is responsible for displaying a green background and a colored ball. The ball is pre-drawn, and is displayed like a sprite. There is also a yellow border for the table that will be drawn and will play an important role during calibration.

This controller communicates with the software by reading and writing on a bus. The software provides the coordinates and the color of the ball by writing it on the bus so that the controller can read it. In addition to that, the controller keeps synchrony with the software by writing on the bus when it is ready to receive new data.

Calibration is important in our system, and the controller plays a role in that by displaying a stick somewhere on the screen as requested but the software. During calibration period, the green table with no balls will be displayed, in addition to cue-like sticks that will be displayed horizontally or vertically touching each one of the yellow borders of the table. The software chooses the border where the stick is to be displayed and sends its choice in two bits to the controller in addition to a bit indicating if it is in calibration mode or not. By this we ensure that the hardware and software are referring to the same area. This calibration usually happens when the system starts, but it can also be initiated at any time if the software request that.

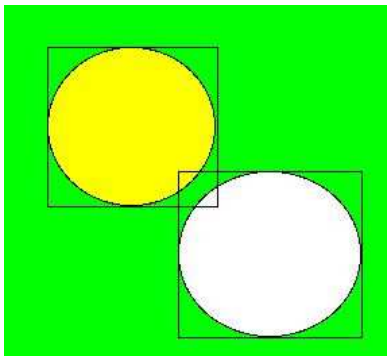

Figure 10: Ball movements and collisions

This design is expandable to a constant number of balls where the software provides the co-ordinates of all balls, their corresponding colors and whether they are to be displayed or not. In this case 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 registry and then signal the software that is it ready for the next data. At the same time it starts displaying the new frame with the new ball positions. Basically is a process running for every ball, and this will indicate the location of the square area on the screen where its sprite will be displayed. Since the ball shape doesn't fill the whole square, there is a chance that two or more squares are intersecting but the balls are still not touching, as shown in Figure 10. For the intersected area, part of it can display a ball from one of the sprites, another part can display the ball of another sprite, in addition to a blank area that gets displayed in the background. To achieve this we have to read from all sprites, compare the values and draw only the dominant one. These conditions will make sure that the display process will include the green table, the yellow border, and complete non-overlapping balls correctly.

7 Open Points

1. One of the signals in the interface between the TRDB-DC2 and the FPGA is a ground. It is unclear whether this should be designed as an inout pin that is tied to 0 within the FPGA.
2. The pixel processing module is designed to interrupt the software running on NIOS-II each time a new position is available for the cue-tip. The mechanism of creating interrupts in the SOPC systems is to be determined.
3. Should a user be able to initiate a calibration at any time during the life of the application? Is an external interface required for this?

8 Project Management

8.1 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.

The code tree appears as indicated in Figure 11. Test benches for the VHDL sources are included within the vhdsrc directory.

8.2 Implementation Milestones

1. Milestone 1
   - Hardware implementation of the camera interface
   - Hardware implementation of the object detection algorithm through pixel scanning
   - Hardware implementation of basic VGA controller module (not including calibration requirements).
2. Milestone 2
   - Thorough calibration of the system to fine-tune the projection and camera modules and achieve accuracy in determining object location.
   - Basic implementation of the software bouncing ball
3. Milestone 3
   - Optimization of hardware
   - Full development of software design, and testing
4. Final Milestone
   - Testing with a projector
   - Report and presentation completion

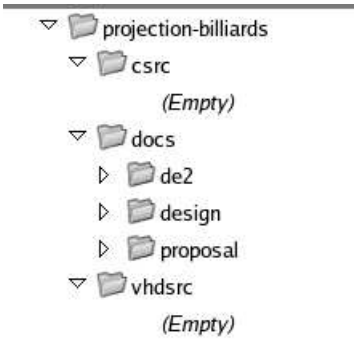

Figure 11: Directory Tree Structure

9 Glossary of Terms

ADC	Analog to Digital Converter
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
I^2C	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

References

[1] Altera Corporation. *Avalon Memory-Mapped Interface Specification.* www.altera.com, San Jose, CA, 2007.

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

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

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

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