Project Report:
Virtual Exercise Game

ECE532 Digital Systems Design

April 8, 2009

Dai (Mike) Wang (993811951)

Chao (Chris) Wang (993750703)

Cheng-Chieh (Jack) Yang (994124318)

Contents

Overview:

The goal of this project is to create an exercise video game that will direct users to move their bodies and flex their muscles utilizing the Xilinx Virtex-II Pro FPGA board. The following describes our overall system:

- A video camera is setup and connected to the board utilizing the video decoder
- The "player" will be required to hold objects of a certain distinctive colour. These objects would be recognized by the video camera using a recognition algorithm
- A video image is displayed on the screen utilizing the VGA controller. A target box will be displayed on the monitor, and the detected player's position will be displayed with another box
- The "player" is then required to move the object into the target box on the screen. Once the player reaches the target position, the target box will move to a new location randomly, and the process repeats.

Our project consists of a digilent's video_capture project which we used as the base pass-through system. We wrote our own game controller, image superimposing modules on top of the sample project to achieve our purpose. PLB, MPMC, Microblaze and UART Xilinx IPs were utilized for our project for various purposes which will be explained in detail in the report. In addition, we instantiated an asynchronous FIFO which was built using Xilinx ISE, and PLB Master/Slave interface. Finally, a software algorithm was written to detect color of interest, which relies on the MicroBlaze.

Block diagram of the project is shown below:

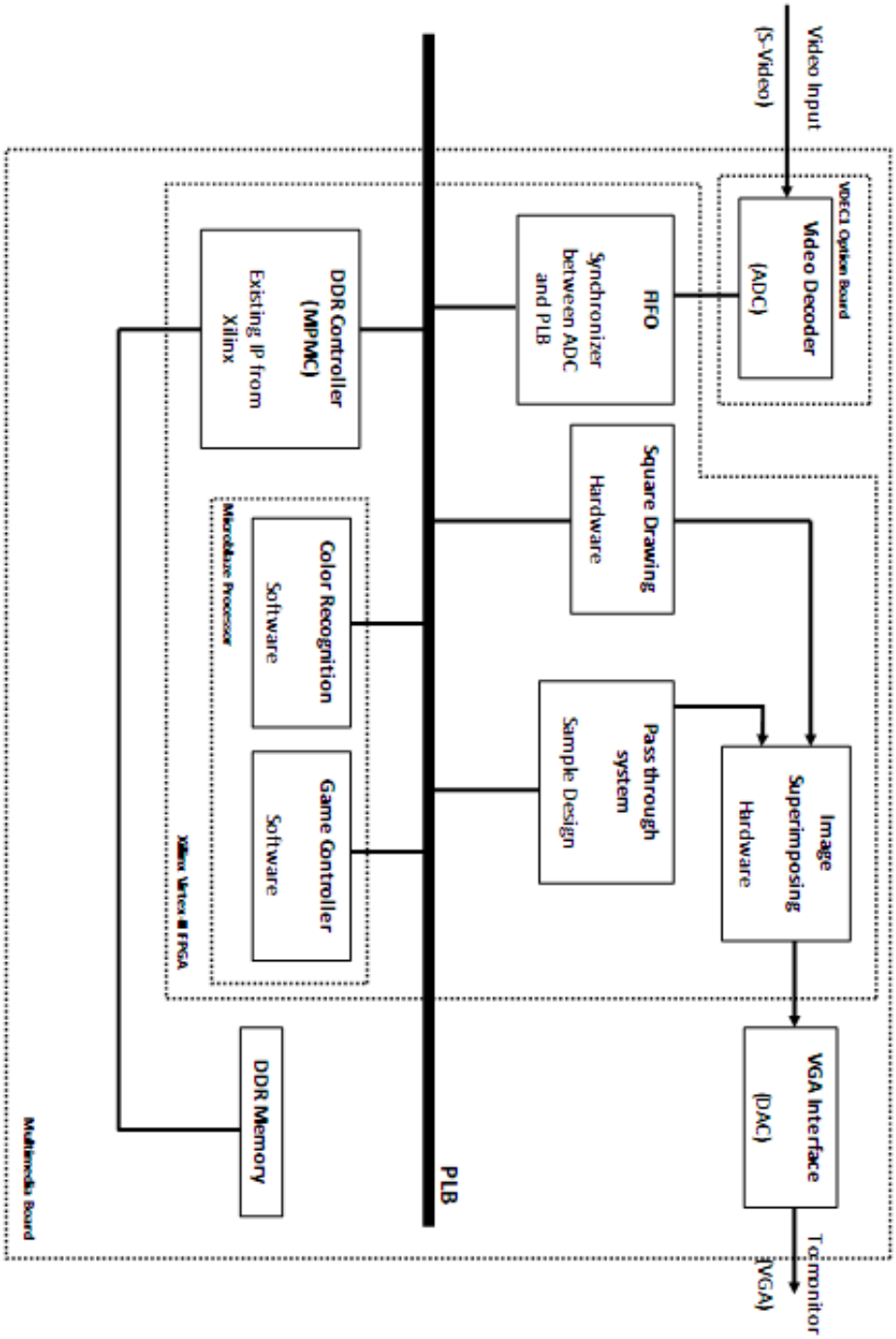

Figure 1 – System Block Diagram

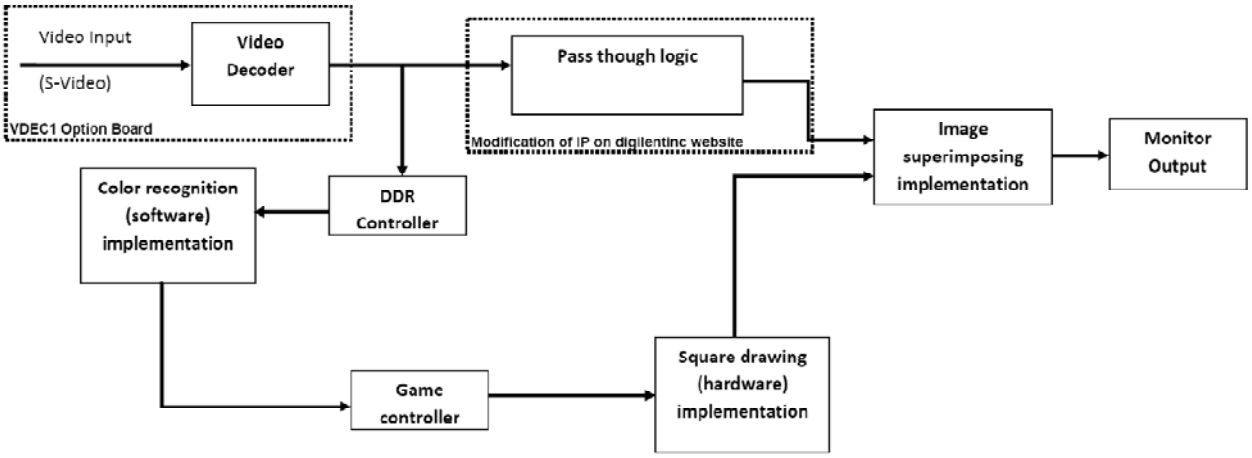

Figure 2 - System Overview

Outcome:

Our project successfully implemented our proposed design and has met initial specifications. The project is able to detect a patch of a specific color and jump to a new location with the location of detected color matched the target location.

There are several of areas where this project can be improved:

1) The color detection algorithm can be optimized as right now the code is 2 nested *for* loops sweeping from the left top corner to the right bottom corner. Instead of doing exhaustive search, perhaps the data could be pre-sorted to eliminate the blocks that for sure do not have the color of interest. For example, the R values for 10 pixels could be summed up and compared against 300, and if the value is smaller, then the chances of red are unlikely. Alternatively, the game detection could be moved to hardware. Instead of storing the RGB values of a frame into the DDR, a true/false table stating whether each pixel matches the color interest or not can be stored. The software can simply just sweep these values which should be dramatically faster.

2) We can add enable the multi-object detection by modifying the C code, which can be used for future creative implementations.
3) Countdown timer and scorekeeping are also some other aspects that can be added to improve the game.
4) As suggested by our TA, David, the color of the detected and the target box can be manipulated to a color other than black so it would be easier to differentiate from the background.
5) A better camera could help with the aesthetic aspects of the outputted video.

Description of the Blocks:

Clock Generator (DCM, clock_generator_0)

Clock_generator (Xilinx IP), Version 2.01a

The clock generator block takes the onboard clock and generate different clock signals for different modules accordingly. It is used to ensure all clock signals are on the clock tree and delays or phase shifts of clock signals are prevented.

A 100 MHz clock is generated and provided to MicroBlaze processor, PLB Bus, MPMC. A 13.5 MHz and a 27 MHz clocks are provided to the video capture block for video processing.

Video Capture / IIC Controller (video_capture_0)

Version 1.01b, (design from Digilentinc) / XPS_IIC(Xilinx IP) Version 2.00a

The video capture core takes video inputs from VDEC board and converts the raw signals into RGB values @ 13.5 MHz. Each line in the video frame is written into 1 of the 2 line buffers (BRAM) and they are read out and displayed on to VGA output at 27 MHz.

The RGB signals are also intercepted and fed into an asynchronous FIFO and in turn the DDR memory for color recognition. A single frame memory mapping design is used as the video capture core runs at a much slower clock rate than the MicroBlaze processor. The target memory address will increment for every new pixel until the frame is done and restart at 0xA0000000.

The core is controlled by UART through the IIC Controller. Where Tx and Rx signals are inputs and outputs to/from the UART. The IIC Sclk and IIC Sdata are used for select appropriate input formats and data transfers. Below is a block diagram of the IIC master control module and how it is linked with other cores through the PLB bus.

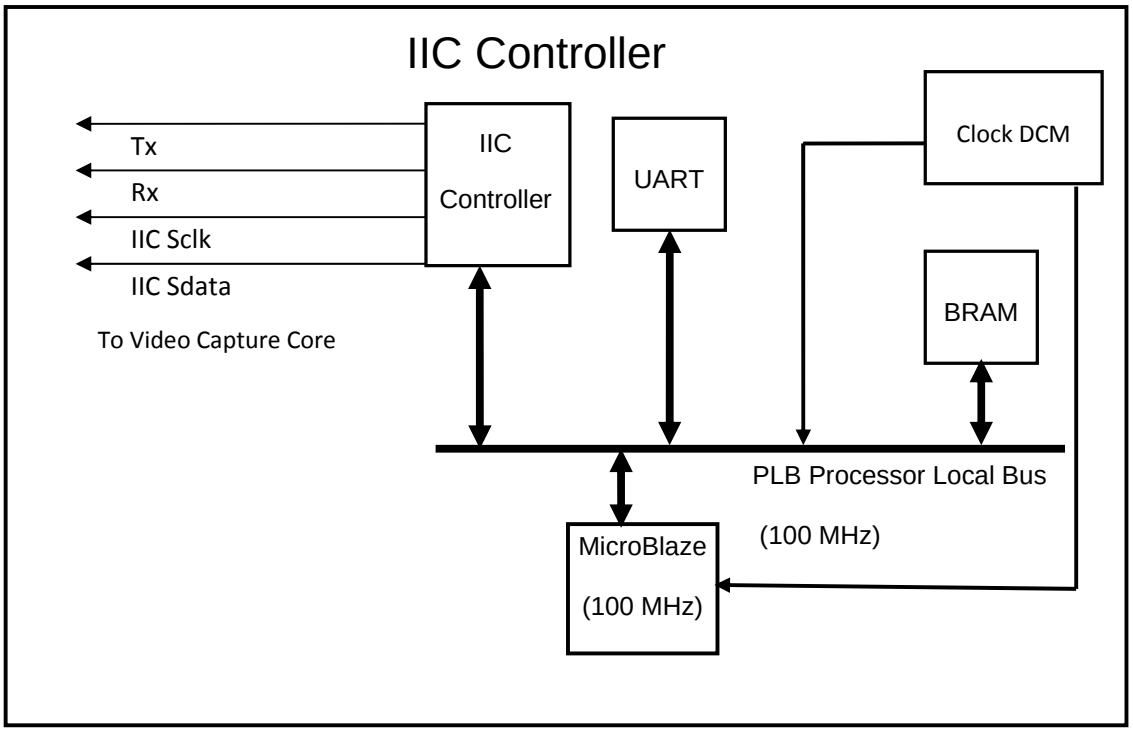

Figure 1 - IIC Controller

BRAM (lmb_bram)

Bram_block (Xilinx IP), Version 1.00a

The video capture core utilizes Block Ram as line buffers where it provids simplicity and speed. The line buffers are written in an alternating fashion with every other line from the video frame and they are also read out and displayed on to the VGA in a similar scheme.

Asynchronous FIFO

FIFO_Generator (Xilinx ISE IP), Version 4.4

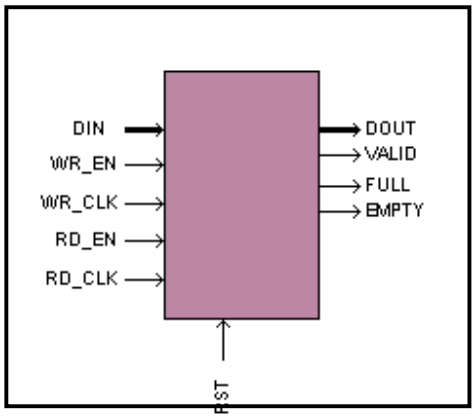

Figure 2 - Asynchronous FIFO

An asynchronous FIFO is used as a synchronizer between the video capture core and the PLB master interface module. The video capture core provides RGB signals of every pixel at 27 MHz and these signals are intersected and written into the FIFO along with the target memory address. On the reading end, the PLB master interface reads from the FIFO at 100 MHz and in turn prevents metastability issues caused by transferring data across multiple clock domains.

PLB Mater Interface (plb_engine_fifo)

Generated from XPS peripheral wizard / modified with FSM and linked with asynchronous FIFO.

The PLB Master Interface is an internal module that acts as a data handler. After it reads RGB data and target memory addresses from the FIFO. The FSM inside will take these data and write them on to the PLB bus whenever possible and in turn written into the DDR memory using the MPMC. A detailed drawing of the finite state machine (FSM) is shown below.

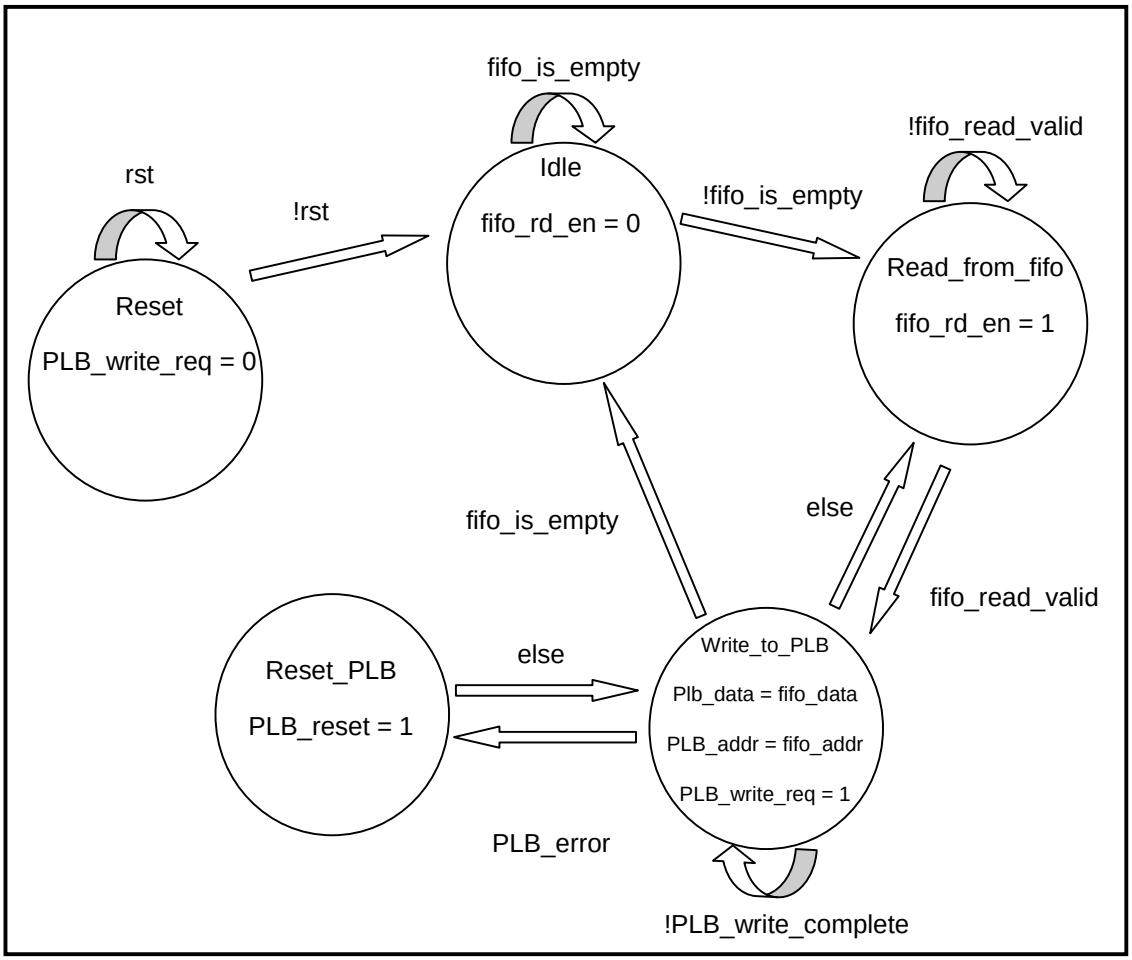

Figure 3 - FSM Diagram

PLB (mb_plb)

PLB_V46 (Xilinx IP), Version: 1.03a

A 128 bit PLB bus is used to provide the bandwidth needed for all data traffic across all modules in the system. Since we do not alter the original video image and it is only used as a “background” to our system, we effectively reduce the bandwidth requirement to only the transfer of a single video frame plus any software activity.

MPMC/DDR RAM (DDR_SDRAM)

MPMC (Xilinx IP), Version: 4.03a

The MPMC is configured to work with 512MB Dual Rank DDR memory with 1 single PLB port. Thanks to our TA David Woods, we were able to solve the only issue we had which was the data reading error.

XMD Debug (debug_module)

Mdm (Xilinx IP), 1.00d

Hardware debug was used for the system through XMD and JTAG. The MDM module uses the MicroBlaze processor and allows us to read and write into different memory addresses of the BRAM, DDR memory and various other memory mapped peripherals.

MicroBlaze (microblaze_0)

Microblaze (Xilinx IP), Version 7.10d

The main processor used in this project. The software application has 3 purposes, configuring the video capture core using the IIC controller drivers, providing communication between UART and internal cores, and finally implementation of the color detection algorithm.

System Reset Generator (proc_sys_reset_0)

Proc_sys_reset (Xilinx IP), Version 2.00a

The onboard reset signal is used in conjunction with other signals generated from different cores to provide individual reset signals to different modules. These reset signals include MicroBlaze_reset, Peripheral_reset, Bus_reset.

UART (RS232_Uart_1)

XPS_Uartlite (Xilinx IP), Version 1.00a

The UART (RS232) is used for configuring the video capture core and for debugging purposes. It provides the basic communication needed in between the FPGA board system and the PC.

Color detection algorithm (mem.c):

Software application (custom written)

The algorithm sweeps the entire frame from the left top corner to the right bottom corner, on a row by row basis. The algorithm consists of a nested *for* loop with x coordinates incrementing inside of y coordinate. The function calls the color_match function at every pixel which returns whether the specific pixel location is the top left corner of a box of interest. There are three parameters that specify the characteristics of the box:

- check_x -> number of bits to check in the horizontal direction
- check_y -> number of bits to check in the vertical direction
- check_freq -> check every 'check_freq' bits. ie - if this value is 3, then horizontally, it checks every 3 bits.

The current algorithm looks for pixels with R value greater than 150 and G,B both less than 80.

Game Controller (Controller.v)

Custom hardware block

The game controller module is implemented in the verilog file *controller.v* and instantiated in the *video_capture.v* file in the video_capture IP core downloaded from www.digilent.com.

The game controller was initially designed to perform two functions:

i) to generate the position of the game target box and output an (x,y) coordinate to the drawing block module

ii) to check if the position of the detected object from software is aligned with the position of the target

In addition to the above list, the controller is also responsible for outputting the (x,y) coordinate of the detected box to the drawing block module. This is to make the design more organized, and structured. Once the colour recognition module outputs the detected (x,y) coordinate, the game controller will compare it with the (x,y) position of the target using combinational logic. This will determine if the player is successful the objective of the game. Once the player has matched the target, the controller will generate a new (x,y) coordinate for the next game target.

The inputs to the game controller module are clock, reset, X, and Y coordinates. The input clock is 13MHz and the reset signal is connected to a user switch on the development board for debugging purposes. The X and Y input signals are sent from the software colour detection system running on the MicroBlaze. Through the peripheral block created to connect to the PLB slave, the values of X and Y are latched to registers, which are then connected to this game controller module. The outputs of the game controller are two sets of (x,y)coordinates , the

detected box and the target box. These outputs are connected to the box drawing circuitry, which generates an image of the boxes on the monitor.

The first part of the controller is to latch the (x,y) values of the detected box to the output using registers on the positive edge of clock. Next, combinational logic is used to determine if the two sets of (x,y) coordinates match within a parameter of 10 pixels. If this requirement is met, the signal “start” goes high (logic “1”), or else remains low (logic “0”).

The X and Y coordinates of the game target box is generated by another module called “random” which gives a variety of game play for the next position of the target. The inputs to the “random” module are clock, reset, and “start” signals, and the outputs are the X and Y values of the target box. This module generates a pattern by continuously adding a constant value to temporary register #1, and a different constant value to temporary register #2 every clock cycle. Once the “start” signal goes high, then the value in temporary register #1 and #2 are latched to the X and Y coordinates of the target box, respectively, on the positive edge of clock. In addition, the values in the temporary registers are constrained to the boundary pixel coordinates of the monitor screen, as defined as XMAX, XMIN, YMAX and YMIN in the verilog code.

For debugging capability and an additional feature to the game, the module takes inputs from the User Switches (up, down, left, right) on the Virtex II FPGA development board. The four directional inputs will modify the X or Y coordinates to move the box by a constant number of pixels on the screen.

Furthermore, a test bench was written in verilog to test the functionality of the module in simulation using ModelSim. The test bench is included in the test bench folder in the design directory.

Drawing Block (video_capture.v)

Modified from video capture core

The drawing block is responsible for outputting an image frame of a black square box line by line, given an input of (x,y) coordinate from the game controller. Each colour component value is stored in 8-bits (256 distinct values), with the value zero being black. The output of the drawing block is superimposed with the video image using bit-wise AND gates. At the (x,y) locations of the square box, the drawing block outputs a value of '0', which is ANDed with the original image to give a final output value of '0' (black). At all other locations, the drawing block outputs a value of '1', to keep the original image colour value after the AND logic implementation. The drawing is implemented in verilog and is integrated within the video_capture.v file, instead of having its own module. To keep track of the x and y coordinate the system is outputting, counters are implemented to compare it with the reference (x,y) in order to figure out the borders of the square box. The size of the box is configured to be 60 x 60 pixels, with each border 10 pixels wide.

Image Superposition Implementation (video_capture.v)

Modified from video capture core

The purpose of this implantation is to superimpose the image from the camera with the image from the square drawing block. This block consists of an AND gate synchronized with the SVGA output clock and the output is produced one line per cycle. The AND gate is suitable due to the colour of the box drawn on the screen is chosen to be black, which in binary consists of only "0"s. Using an AND gate allows the preservation of the original data and alteration only occurs where

black pixels are needed. This image superposition is implemented within the video_capture.v file, and is not separated into a distinct module since it requires signals from several other modules used in the file.

Initially the drawing box appears to be a faint black box instead of having solid boundaries. This was due to the interlacing of odd and even lines and that the RGB values are modified before the line buffer storage. This was solved by ANDing the original RGB values with the desired box RGB values after the line buffer outputs, and therefore makes sure that both odd and even lines have the proper RGB values that incorporates the black box.

PLB Slave Interface (peripheral_bauer)

Generated from XPS peripheral wizard

The purpose of this peripheral is to allow the software running on MicroBlaze software to send the detected (x,y) coordinate to modules written in verilog. The concept of this peripheral involves the software to write into a memory address location, which corresponds to registers in hardware. The interface is implemented by the XPS Peripheral Builder application, which takes care of PLB data transfers, and timing information. This peripheral is generated as a custom IP core, which is added to the XPS project. Furthermore, two HDL files are generated called user_logic.v and peripheral_bauer.vhd. Since each register is memory-mapped to address locations are 32 bits and the (x,y) coordinates are only 11 bits, the X and Y values are defined as the first 11 bits of each register starting with the MSB. This is done in verilog in the user_logic.v file as shown in the figure below.

```
assign x_data = slv_reg0[21:31];

assign y_data = slv_reg1[22:31];
```

Figure 4 - Assignment of X, Y data values to the outputs of memory-mapped registers

Description of Design Tree

Description of notation: Directories are in Normal Text while descriptions are *italicized*

- Project – *root directory*
    - TestApp_Memory/src- *contains all software code of the project*
    - drivers- *contains all the software drivers needed*
    - pcores – *directory containing all the hardware designs and custom IP cores of the project*
        - peripheral_bauer_v1_00_a – *a PLB slave interface that maps registers to address locations*
        - plb_engine_fifo_v1_00_a – *a PLB master interface that takes data from the video capture core through an asynchronous fifo and writes it onto the PLB*
        - -video_capture_v1_01_b - *a modified core from the VDEC1 reference design which stores horizontal lines of video into local BRAMs. RGB values are also sent to the plb_engine_fifo IP core through the asynchronous fifo.*
    - controller_simulation- *contains ModelSim project and test bench for simulating the controller.v design*