ECE532 Digital Systems Design
Project Report

WiiteBoard

Fan Zhu 993837773

Chao Qu 993909026

John Zhou 994066853

April 8, 2009

Table of Contents

Acknowledgement

This creative idea of using Wiimote to convert a screen/surface into a “writable” surface is first introduced by Johnny Chung Lee [1]

1. Overview

1.1 Goal

The goal of our project is using the Xilinx XUPV2P FPGA board to implement an interactive whiteboard. This whiteboard should contain non-trivial features, similar to Microsoft Paint, such as drawing a line and change sizes and colors of the drawing brush.

1.2 Block Diagram

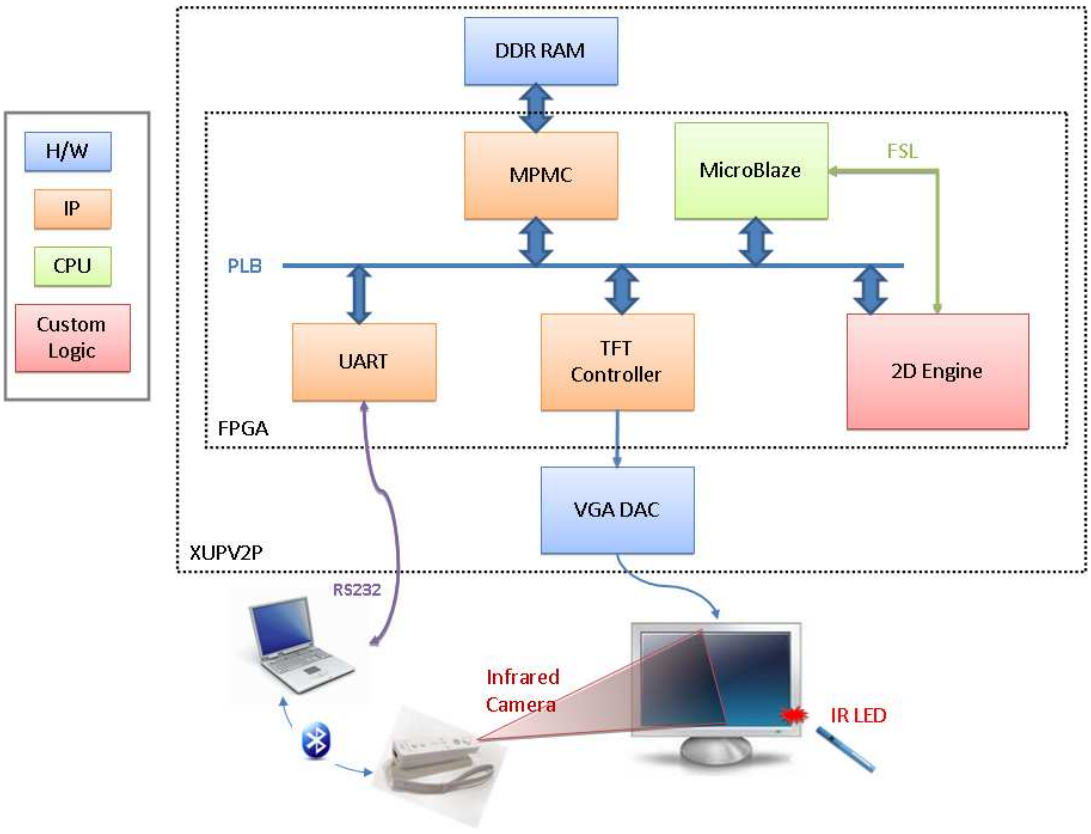

Figure 1

1.3 Overall System Description

Figure 1 will be used to describe the working flow of the system. Please refer to Figure 1 for the connection details. The following steps outline how the system works:

1. Calibrate the Wii remote using traditional 4 point calibration [2].
2. User moves the infrared pen within the calibrated viewing range of the Wii remote.
3. The Wii remote captures the raw coordinates of the infrared pen.
4. Wii remote sends the coordinates to a laptop via Bluetooth.
5. A small C# program (running on a PC) is created (using open source libraries [1, 3, and 4]) to get data from Bluetooth stack, transform it to calibrated coordinates and then send the coordinates to the FPGA board through UART.
6. Depending on the coordinate, MicroBlaze sends different instructions to the 2-D engine through FSL interface.
7. 2-D engine decodes the instruction, for control instructions such as color change or brush size change the 2-D engine updates its internal status registers; for instructions that draw pixels on the screen the 2-D engine writes to the video frame buffer via the PLB interface; for special instructions such as drawing a line the 2-D engine performs linear interpolation between the starting and ending points and writes that series of pixel values into the video frame buffer via the PLB bus.
8. The video frame buffer is implemented using the 256MB DDR external memory, the IP core TFT controller monitors the state of the video frame buffer and writes the content into the VGA DAC which in turn drives the input signals of the LCD monitor.

2. Outcome

2.1 Results

Our final design achieved the proposed functionalities. The 2D engine was implemented in hardware which had a visible speedup when compared to a pure software implementation of the project. The user interface was completed with the exception of DrawRect and DrawCircle.
Users can:

- Free draw on the display device connected to VGA port of the FPGA board.
- Change size of the brush.
- Change color of the brush.
- Erase the stuff drawn on the screen.
- Eraser size can be changed too just like brush.
- Clear the entire screen.
- Draw lines.

2.2 Future Improvements

Sky is the limit! There are literally endless possibilities to improve this project. Listed below are a few of the important ones and are organized into performance improvements and feature improvements.
Performance:

- The Wii remote is only able to detect 100 pixels per second. If the user draws across the screen too fast there will be blank pixels in the trail. Getting a higher performance Wii remote will help.
- The Xilinx board does not have a Bluetooth interface so we have to go through a PC to send the coordinates to the board by UART. It will help if the board had native support for Bluetooth.
- Our 2-D engine is currently running at 50MHz and it meets the timing constraints fine. We could probably crank it up to a higher speed.
- The Xilinx TFT controller IP core only supports 640*480 resolution and 18 bits color. Higher resolution and more color depths will be good.
- Have a better mounting apparatus for the Wii remote, preferably above the screen so there is less risk of the sensor being blocked.

Feature:

- We could improve our 2-D engine to support more shapes.
- We could make a 3-D engine!
- After we finished our project, we discovered the TFT controller API had a

feature for changing the frame buffer pointer. Which means we could implement dragging and dropping of objects.

- Multi touch whiteboard!

3. Block Descriptions

3.1 MicroBlaze

IP Type	IP Version
MicroBlaze	7.10.d

The MicroBlaze processor is used for the following operations:

1. Receives wiimote coordinates from the UART interrupts
2. Issues instructions to the 2D-engine through the FSL master interface
3. Implements the GUI and draws menu items using the TFT
4. Controls the parameters of the user's brush (brush size, colour, mode etc.)

3.2 PLB

IP Type	IP Version
plb_v46	1.03.a

3.3 FSL

IP Type	IP Version
fsl_v20	2.11.a

3.4 TFT Controller

IP Type	IP Version
xps_tft	1.00.a

The TFT controller writes 6-bit rgb values into memory locations corresponding to coordinates on the screen. The formula for the conversion between screen coordinates and memory addresses is:

Addr = BASE_MEM_ADDR + 4096 * ROW + 4 * COL

3.5 MPMC

IP Type	IP Version
mpmc	4.03.a

3.6 UART

IP Type	IP Version
xps_uartlite	1.00.a

3.7 2-D Engine

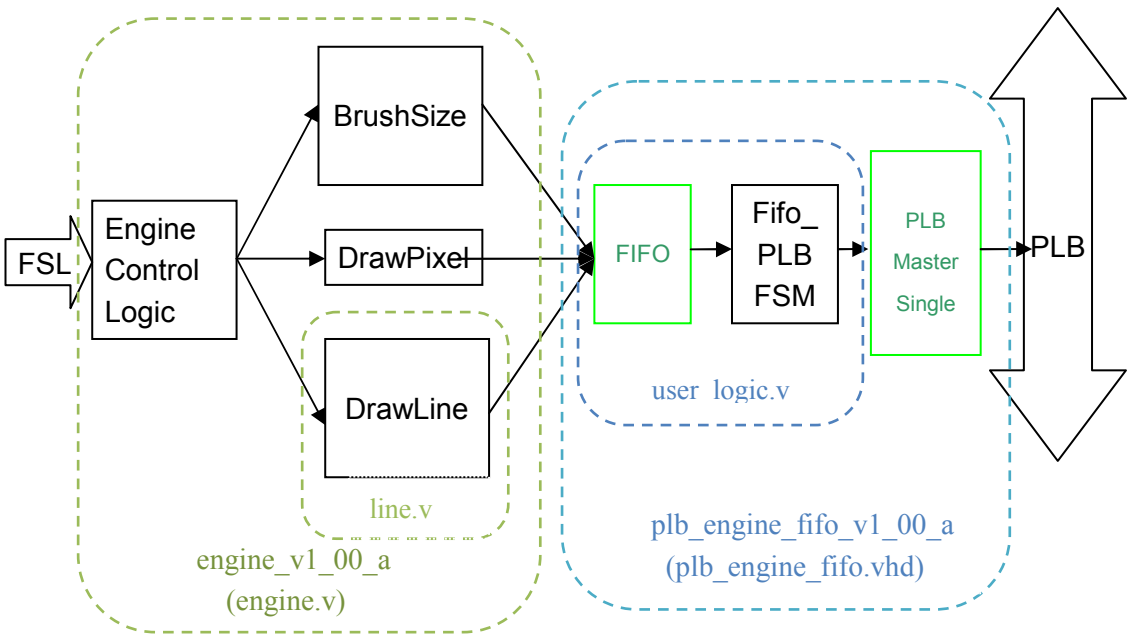

Figure 2

3.7.1 Engine

IP Type	IP Version	Comment
Engine	1.00.a	Custom hardware coded in verilog. Takes in instructions from the FSL and outputs memory addresses and RGB values to be written into memory.
- DrawPixel	N/A	Translates coordinates in screen space to addresses in memory space.

- DrawLine	Flip-Flop version	Bresenham's line algorithm [5] is used to created line module. A version using latches was first made that intended to optimize for performance (5 cycles to output first pixel, following pixels are ready in every two cycle, max clock period ~ 9 ns). As the latch version did not work in hardware, a flip-flop version is created. Though requires more cycles to output data, the shortened clock period enables the module to run at a higher speed (7 cycles to output first pixel, consecutive pixels are ready in every 4 cycles, max clock period ~ 5 ns). Verilog source code and simulation results for both versions are available in design tree.

3.7.1.1 Engine States:

Wait
Default state. Performs reset of several variables. Waits for FSL_Exists signal to rise, indicating data is present. Engine enters this state on reset.

Instruction Fetch
Stores opcode and data from the FSL bus, initializes values needed in later states.

Instruction Decode
Depending on opcode, different variables are set. e.g. brush size.

Data Fetch
Currently only in use if opcode is DrawLine. Reads in the endpoints coordinates of the line.

Execution (Pre)
Currently only in use if opcode is DrawLine. Stall until line module's busy signal drops low.

Execution2
Currently only in use if opcode is DrawLine. Starts the line drawing module and starts reading its output to the main output registers.

Calculation
Calculates the output coordinates for brush sizes larger than a single pixel.

Send
Sends data in the output registers out to PLB FIFO. Waits for acknowledgement signal.

PLB Full

Waits for FIFO's full signal to drop.

Cooldown

Stalls for a predetermined number of cycles to allow PLB FIFO to have more room to process later data.

3.7.1.2 Opcode Specifications

Input from the FSL bus has a width of 32 bits. The instruction fields are arranged as follows:

31:29	**27:16**	**15:4**	**3:0**
Unused	Data 2	Data 1	OpCode

The opcode length is 4 bits to allow for future development. Currently the instructions are:

Reset

31:29	**27:16**	**15:4**	**3:0**
Unused	X	X	0000

Resets the engine to Wait state. Resets all variables.

DrawPixel

31:29	**27:16**	**15:4**	**3:0**
Unused	Y coordinate	X Coordinate	0001

Draws a pixel to the (x, y) coordinate on the screen

DrawLine

31:29	**27:16**	**15:4**	**3:0**
Unused	Y1 coordinate	X1 coordinate	0010

31:29	**27:16**	**15:4**	**3:0**
Unused	Y2 coordinate	X2 coordinate	X

Draws a line from (x1, y1) to (x2, y2). After receiving the first 32 bits, the engine enters the data fetch state upon decoding the DrawLine opcode and stores the endpoint coordinates.

DrawRect (Not implemented)

SetSize

31:29	27:8	7:4	3:0
Unused	Unused	Size	0100

Sets the size of the brush to be N. The output of later DrawPixel commands will be a square block of side length 2N + 1 centered at its (x, y) coordinate.

SelectColour

31:29	28:22	21:16	15:10	9:4	3:0
Unused	Unused	Red	Green	Blue	0010

Changes the brush colour to the specified RGB value.

3.7.2 PLB_ENGINE_FIFO

IP Type	IP Version	Comment
Plb_engine_fifo	1.00.a	Custom made logic, including two Xilinx IPs listed below and one FSM to glue the two modules. The FSM is running at PLB Clk (100MHz).
- plbv46_master_single	1.00.a	Xilinx IP.
- Asynchronize FIFO	FIFO GENERATOR V4.4	CoreGen generated FIFO. DATA_IN works at engine_clk(50MHz), DATA_OUT works at PLB_clk (100MHz). This fifo provides a buffer between our engine and PLB, so that our engine will not stall on PLB transactions.

4. Design Tree

BaseTerm\	C# program
WiimoteToXUP.sln	C# project file
BaseTerm\bin\Release\BaseTerm.exe	executable file
TBs\	
Line\	Test bench for line module
README.txt	Instructions on simulation
line.v	Flip-flop version
line_latchV1.v	Latch version
line_ise\	ISE project
timesime\	post-synthesis testbench
README.txt	Instructions on generating a netlist and run post-synth. simulation.
line_timesim.v	Netlist for post-synth. simulation.
Fifo_PLB_logic\	Testbench for user_logic.v (see 1.7.2)
user_logic.v	The 'glue' FSM for FIFO and PLB Master
README.txt	Instructions on simulation
2D_engine\	Test bench for the entire engine.
ECE532_PROJECT\	XPS project directory
pcores\	Custom logics
engine_v1_00_a\	
hdl\verilog\engine.v	
hdl\verilog\line.v	line module (FF version)
plb_engine_fifo_v1_00_a\	FIFO to PLB logic
hdl\vhdl\plb_engine_fifo.vhd	Top level of this block
hdl\verilog\user_logic.v	The FSM (see 1.7.2)
code\	
xtft_example.c	Example code for using the TFT controller

5. References

[1] "Low-Cost Multi-point Interactive Whiteboards Using the Wiimote"
http://johnnylee.net/projects/wii/

[2] "A camera calibration using 4 point targets"
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=201621&isnumber=5221

[3] "Develop a .NET Base Class Library for Serial Device Communications"
http://msdn.microsoft.com/en-us/magazine/cc301786.aspx

[4] "Managed Library for Nintendo's Wiimote"
http://blogs.msdn.com/coding4fun/archive/2007/03/14/1879033.aspx

[5] "Bresenham's line algorithm"
http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm#Optimization