Group Report

Hardware Assisted Rock Band

Project Team

Name	Student Number
Braiden Brousseau	994805831
Ted Campbell	994539576
Peng Zhao	994478864

Table of Contents

OVERVIEW 3

PROJECT OUTCOME 4

FUTURE IMPROVMENTS 5

SYSTEM DIAGRAMS 6

DETAILED DESCRIPTIONS 9

DESIGN TREE 14

REFERENCES 15

OVERVIEW

Rock Band is a popular music video game developed by Harmonix Music Systems. It is available on the PlayStation 3 and Xbox360 consoles. *Rock Band* allows players to perform in a virtual band by providing up to four players with the ability to play three different peripherals modeled after music instruments (Guitar, Drum, and Microphone). These peripherals are used to simulate the playing of rock music by hitting scrolling notes on-screen. Rock Band is popular to the gaming community because of its simulation allows players to play guitar, or drum to their favourite rock music.

The goal of this project is to create a system that detects the game of Rock Band through video input, detect the position of scrolling notes for guitar, and control the Xbox 360 automatically to play the music with no error. For the scope of this project, we plan to use the Xilinx XUPV2P board for processing the video input, and the DE2 board to communicate with the Xbox 360.

Our hardware implementation of video processing uses the video capture pcore provided by Xilinx. The video capture block takes composite input from Xbox 360 and decodes it for VGA output. Our custom hardware block, counter, takes some signal from the video capture block and process it real-time to gather the information for scrolling notes and provide a 6 bit output for each button the Xbox is supposed to trigger. The output information is relayed to the DE2 board through the serial port via MicroBlaze CPU. The DE2 board acts as a host to the Xbox controller, guitar in our case, and also a client to the Xbox console. Once the DE2 board receives the information from Xilinx board, it will adjust the controls automatically from the guitar, overwriting any control

that's not needed, hence only correct buttons are triggered to reach 100% accuracy for the game Rock Band.

PROJECT OUTCOME

Our project has been successfully implemented. It plays Rock Band automatically, although user have to strum for themselves, the buttons are triggered correctly 95% of the time.

Originally, the plan was to have the input video data stored in memory using Multi-Port Memory Controller (MPMC), however as the project was developed, our group found out it is not necessary to go through memory, it is possible to sample the video data in real-time and extract the required information. It might've been better to skip the MPMC since it introduces delay writing and reading from memory. All other components of the project followed the proposal and were successfully integrated to work as expected.

The input is consisted of Xbox video out in composite format, which is connected to the video decoder on the Xilinx board, and the output is a 6 bit number referencing which button to trigger on the Xbox. The VGA monitor displays the game, and there is a switch controlling whether the decision box on the fret board should be displayed or not. If the switch is high, a box about the size of the note is displayed above each note on the VGA monitor. There is also a second switching controlling the communication between Xilinx board and DE2 board, if the switch is high, communication between the two boards is disabled and use can use guitar to browse the game menu, and if the switch is low, DE2 board will overwrite the controls based on the 6 bit number, thus rendering guitar controller useless.

The accuracy of note detection is not 100%, this is due to multiple notes appearing at the same time, and since the processing is done real-time, it might miss a note by a single frame and missing the note as a result. Also the game will not work for harder songs played on expert mode, since the notes fall so fast, our detection will detect the wrong notes due to timing mismatch.

Overall, the project achieves the original goal, and will assist players playing Rock Band on normal modes, with accuracy up to 95%.

FUTURE IMPROVMENTS

The game can include a timer to automatically adjust for the falling notes. This will eliminate the bug we currently have, if the notes fall too fast and we experience timing mismatch.

Also with the timer, we can add a sixth bit to our output, to indict when the strumming should occur, and completely automate the playing, this will ensure the game to play at 100% even on the hardest difficulties. Currently user have to manually strum when the notes fall, even if our hardware triggers the appropriate the buttons, if user miss the strum or simply cannot strum fast enough, the notes won't be hit and game will not reach 100%.

The counter block can also be updated for precise counting, and function properly during star-power period. Currently the star-power period is faulty due to the bright background overlapping with the notes. However if a filter is implemented to filter out the star-power's luminance value, we can make the hardware assist players during star-power period as well.

SYSTEM DIAGRAMS

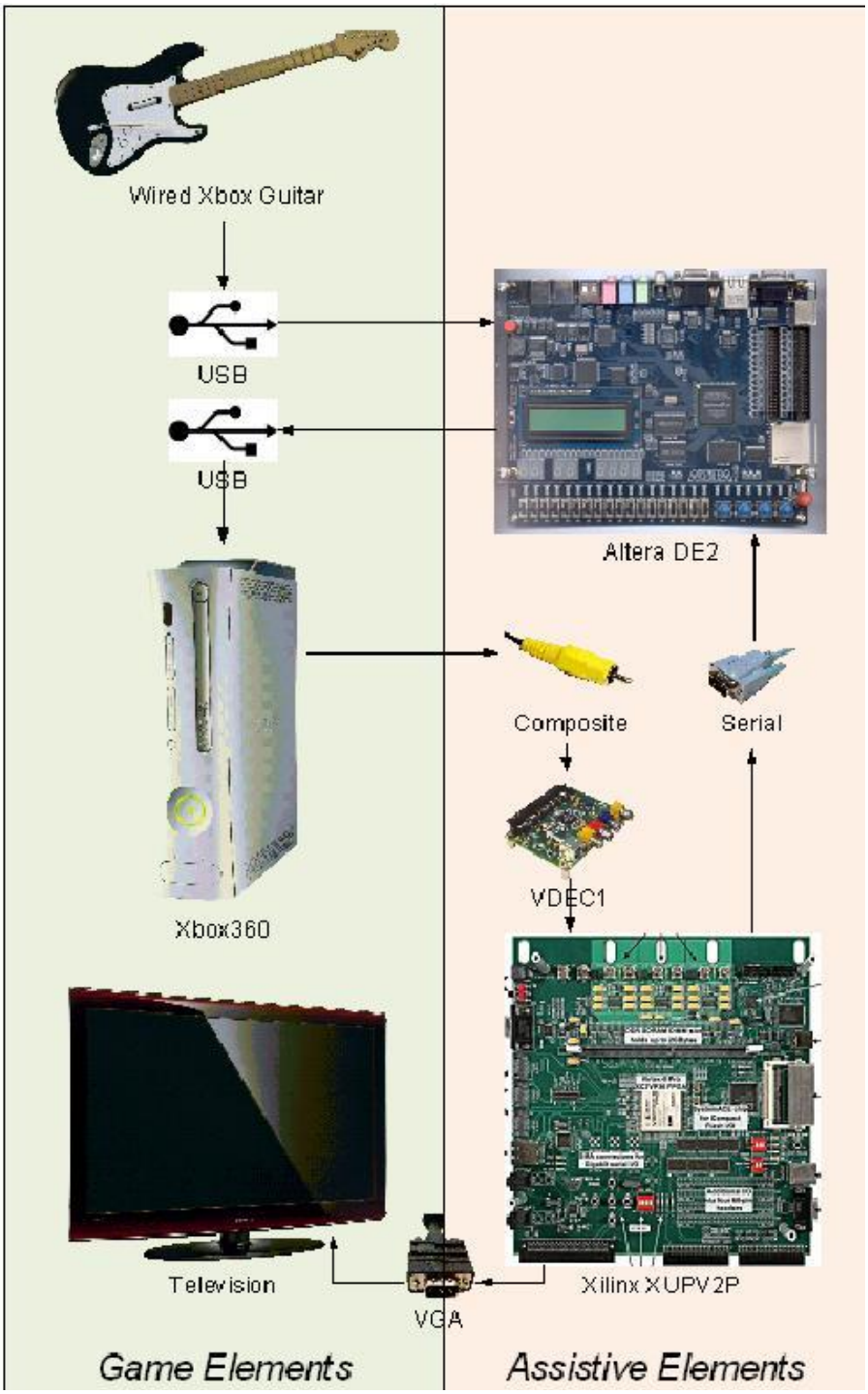


Figure 1. Physical Level

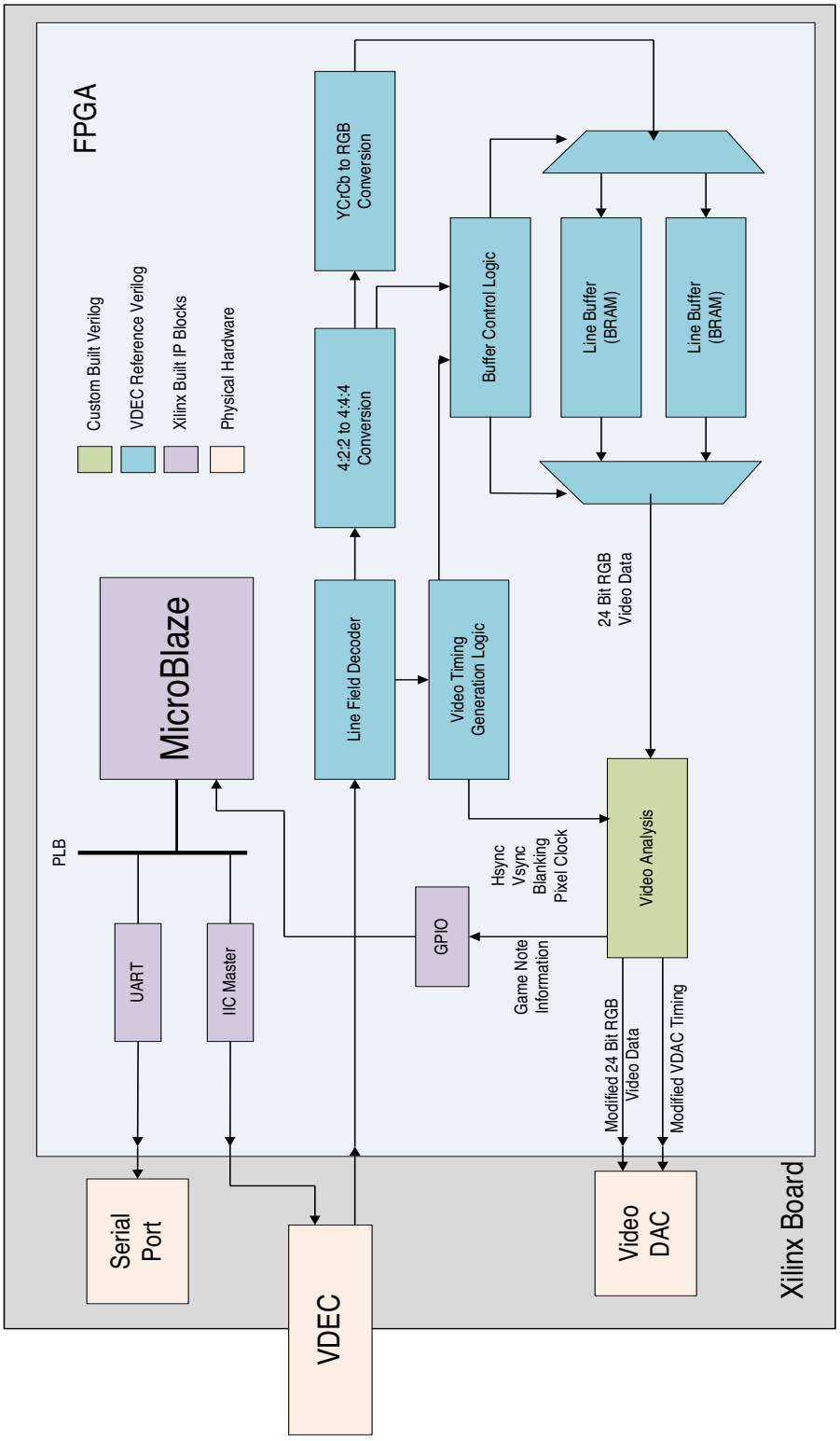


Figure 2. Xilinx Board - XUPV2P Level

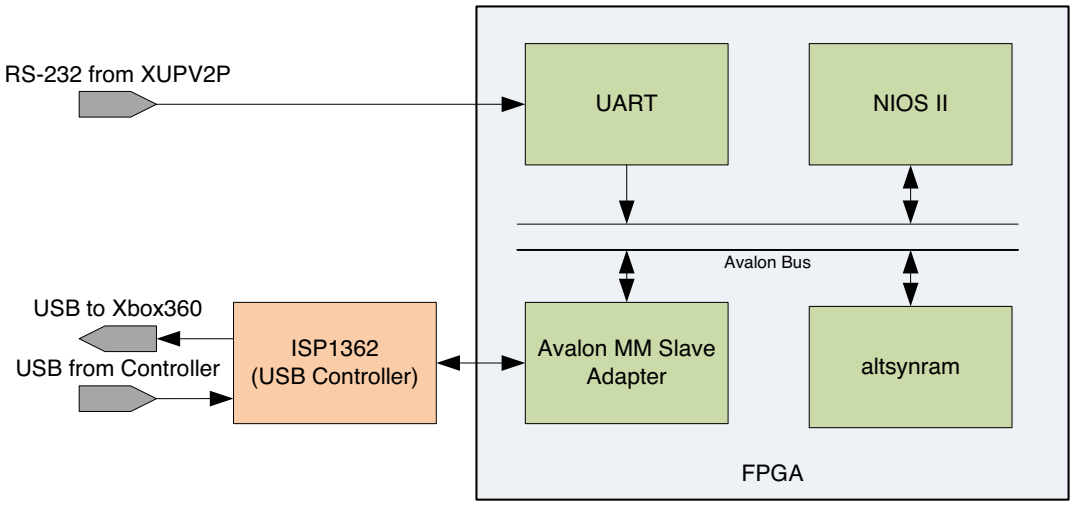


Figure 3. Altera Board - DE2 Level

DETAILED DESCRIPTIONS

Video Capture

Used video_capture_v1_01 from Xilinx website. Xbox composite video output is the input to this board, with output converted to VGA compatible signals and goes through the FPGA out on to VGA display.

Configuring the VDEC

The reference Video decoder project utilized the older OPB bus, which connected to an Interface known as I2C to configure the VDEC registers. The VDEC was then configured in software to listen on the composite video input port which was video data coming from the Xbox. To connect to the PLB bus, we had to replace the I2C pcore with a newer one. This required modifying the configuration code to work with this new interface.

Generating RGB Video

Using pre built verilog blocks found in a VDEC reference project the YCrCb Video was converted to 24bit RGB video data. Only a few modifications were made in this stage. The default blocks removed information from the edge of the video frame and replace it with a black boarder, this was removed. As well the specific YCrCb conversion values were adjusted to provide an output VGA image that more closely matched a strait composite to TV connection.

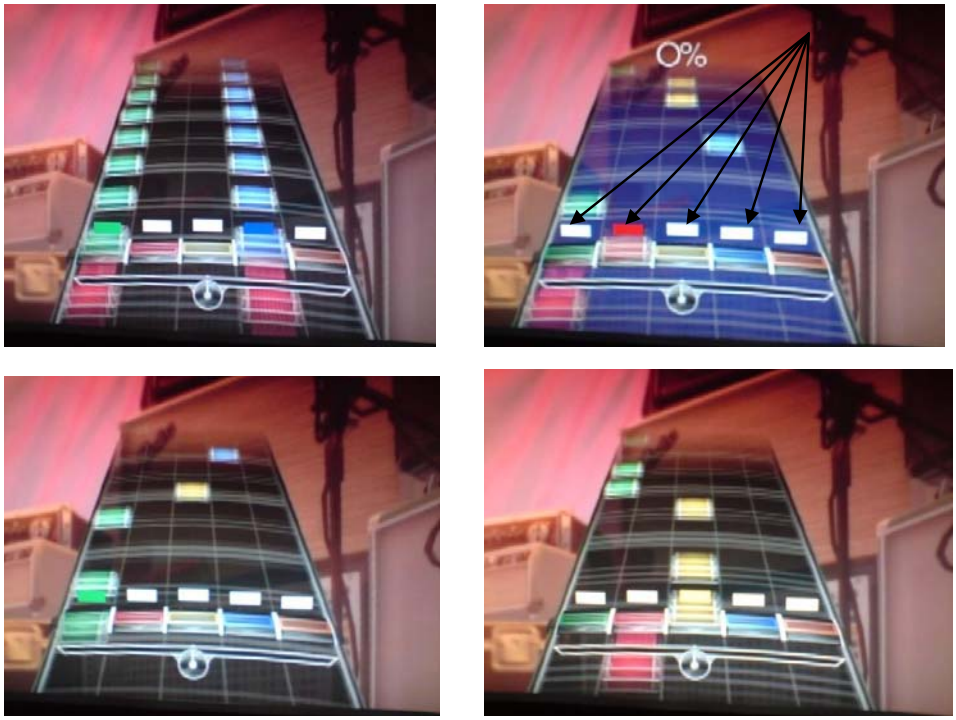
Video Analysis

The video analysis block both provided information about which note should be pressed but also modified RGB data for debugging purposes. The modified RGB data was output from this block with an appropriate delay to the VDAC timing signals where appropriate. One 6 bit signal was also created by this block at the end of each frame, 5 bits denoted which of the 5 notes button needed to be engaged and the final bit was to indicate when to strum the guitar.

Only 5 small rectangles just above the strum bar each the size of one note was considered. Within each of the regions each pixel was tested against an individual formula to see if that pixel was thought to be part of a note block. This formula was hand tuned for each note and looks at the luminance value, R, G and B value of that pixel. If at the end of a frame 75% of the pixels inside a given block are indicated as being part of a note than I say that there is in fact a note. Tuning these parameters I was able to achieve roughly 65% accuracy.

The high error rate was coming as a result of chords which had more than 1 button simultaneously. As that chord was leaving the search box one note would fail to meet the threshold values before the other, this resulting in the hardware holding only 1 note instead 2 (or 3). To fix these sticky notes was added. Before any individual note could be removed from a chord the hardware must agree for 5 frames. This immediately boosted the accuracy to over 95%.

For debugging purposes these search boxes can be displayed on screen. They will change color if they believe a note has just passed by. By watching how they change we can effectively watch how well the algorithm is detecting notes. A picture of that output can be seen below.



Accessing the Video Analysis Data

The controller value which we would like to currently force were sent from the Video Analysis block to the Micro blaze software environment through a simple GPIO connection

Communicating with the DE2

The GPIO is polled in software and, when required, that information is written to the UART and is sent out over serial to the DE2.

Serial connection between Xilinx and DE2

Use crossover cable with two gender changers. Transmits one byte of data at a time from Xilinx board to the DE2 board. The bits contains information on which button should be pressed.

USB Block

The USB block serves the purpose to manipulating USB packets that pass through it, based on the input from the RS232 port. The DE2 is used for this as it has both a Host and Device port and associated controller. For simplicity, we utilize the hardware from the DE2_NIOS_MOUSE_VGA demo that comes on the DE2 system CD. These files are also available on the Altera website for the DE2. We simply download the programmer file to the board and simply worry about its software.

This block proxies USB packets between the upstream Host (the Xbox360) and the downstream device (the controller), while modifying button presses to play the game. The DE2 uses a ISP1362 USB Controller (originally by Phillips, at last check ST-Ericsson owns it). This controller is design for being used as a Host and Device independently and doesn't not lend itself to what we want to do. None the less, we can make use of it if we are willing to ignore certain error conditions and ignore small variations in latency.

A crash course in USB protocol is now needed to understand the challenges and resulting design used. USB is high-speed, packet-based, serial bus with a single master (the 'Host'). It supports hot plugging, auto-configuration using well defined interfaces. Devices on the bus are assigned an address that packets will be labeled with. Since only a single master exists on the bus, this host must request for a device to be able to send data. USB also uses a concept of 'endpoints', such that a field of the packet identifies which endpoint of an address this should be routed to. This allows a device to have multiple independant functions easily.

An endpoint has a fixed direction of transmission and a fixed mode of operation. The two modes of operation we are concerned about for a controller are 'Command' and 'Interrupt'. Interrupt transfers are a little misleading, since the host must initiate the transfer regardless. In this mode the device is polled every few milliseconds and data is returned if ready, else a NACK is sent. The Command mode on the other hand involves three different stages of sending data. First, it sends out data detailing the command. Second, Data is transmitted in the appropriate direction (but only after the host asks for it!). Third, a status stage occurs in which a 0-byte packet is sent in the opposite direction of data.

When a USB device starts up, only a single endpoint is enabled, and it operates in Command mode. This is used for all configuration commands. One of these commands assigns an address to the device, and another command enables other endpoints.

Since there is exactly one Host on a USB network it is clear that what we are doing (with this hardware) will require independent networks. We appear as a device to the Xbox as we pretend we are the real controller. We act as a host to the controller, pretending we are an Xbox. Another concern is that since the Host always initiates transfers, it has strict requirements for us to return a response, either that we have data or that we aren't ready yet. Due to this, the ISP1362 Device Controller requires that data be buffered *before* it is asked for, and the interrupt is received *after* the data has been cleared, informing us to fill the buffer. The next major concern is that both the Host and Device side of the ISP1362 controller read and write to specific addresses; there is no promiscuous mode. This requires us tracking when addresses change and being more of an intelligent

repeated. We also through some error checking out the window because in some cases we must make decisions before we know the answer.

We first solve the problem of Command transfers on the default endpoint. When the SETUP (first stage) of Command is received from Xbox, we send the packet along to the controller. We inspect the request and determine its direction.

- If data is to go upstream, we prematurely request and wait for the data from the controller. This data can then be put in our upstream buffer for when the Xbox wants it. Once the packet has been read by the Xbox, we immediately ask the controller for the next packet and then put it in the upstream buffer. When we are done, the Xbox sends the STATUS (third stage) and we pass this along the controller and the transaction is complete.
- If data is to go downstream, we queue up a 0-byte packet in the upstream buffer. This will be waiting in the buffer until the Xbox is done with us and requires a STATUS stage. At this point we also inspect the packet and determine if it requires special handling. SET_ADDRESS packets require us to change the address the Device Controller listens on, and to change the address we use when talking to the game controller.

By prematurely filling the upstream buffer we are able to successfully support the bidirectional communication the default endpoints need.

The Interrupt mode of transfer is what is actually used for normal button packets from the controller. Data sent from the Xbox is easy to handle as we get an interrupt when received, so we just copy it to the next buffer and send over to the game controller. Receiving data from the game controller is trickier, since data will be periodically requested but not always available. This is handled by ping-ponging between the host and the device. When the Xbox reads our packet and flushes our buffer, we get an interrupt. At this point we perpetually request data from the game controller. When data is finally received, an interrupt is triggered and the data is put in the upstream buffer for when the Xbox next asks. If no data is available, the buffer is empty and the ISP1362 will automatically send NACK responses. This initiate this ping-pong, when we snoop

the SET_CONFIGURATION message on the command pipe indicating to enable endpoints, we start a request for data from the game controller. It will trigger the first interrupt and start the process.

DESIGN TREE

```
-> Rockband
        -> README.txt  Explains the project, steps to set up project
        |
        -> RBMBSys
        |       |  This is the project directory for the Xilinx boards
                   files.
        |       -> system.xmp
        |               |  Xilinx Project File. Open this file to access
                          Project.
        |       -> code
        |               |  Software for the Xilinx board elements.
        |       -> pcores
        |               -> video_capture_modified_v1_04_b
        |                       | Modified version of sample capture code
        |                       -> hdl
        |                               -> verilog
        |                                       |Verilog code containing
                                                |our additions

        -> Rockband_DE2
                |       This is project directory for the Altera tools
                -> rockband
                |       | Software project for Nios IDE
                |       -> USB_README.txt
                |               |Detailed explaination of how the USB
                                |stuff works
                |       -> *.[ch]
                                |Our USB software for the DE2

                -> DE2_NIOS_HOST_MOUSE_VGA.sof
                        | Precompiled hardware to download to the board
                -> nios_0.ptf
                        | The SOPC Builder config file that Nios IDE uses
```

This only shows files we created, changed, or are otherwise of interest. There are other default files the tools use but they aren't not modified.

REFERENCES

DE2 System CD with Demos: ftp://ftp.altera.com/up/pub/de2/DE2_System_v1.2.zip
Linked from: http://www.altera.com/education/univ/materials/boards/unv-de2-board.html

ISP1362 Datasheet: http://www.stericsson.com/technical_documents/ISP1362_6.pdf
Linked from: http://www.stericsson.com/product/222196.jsp

USB 2.0 Spec: http://www.usb.org/developers/docs/usb_20_122208.zip
Linked from: http://www.usb.org/developers/docs/

Xilinx's Video Capture Demo
http://www.xilinx.com/univ/xupv2p_demo_ref_designs.html