ECE532 – Digital System Design

Dance Dance Revolution on FPGA

Name: Jeffrey Puk (992297518)

Dharmendra Gupta (992370206)

Kenny Chan (992462866)

Date: April 2, 2007

Acknowledgements

Special thanks to the following people for making this project a success.

- Mark Jarvin
- Patrick Akl
- Professor Paul Chow

And Xilinx for their generous donation of the Xilinx XUP Virtex II Pro board to the University of Toronto.

Table of Content

1. Overview

1.1 Goals of the Project

The goal of our project is to create a mock-up of the popular arcade game Dance Dance Revolution (DDR). The game will have the following features:

- Custom input interface to capture user input (PlayStation dance pad)
- During game play, a few rows of arrows will instruct the player on when to step on the dance pad. The player will score points when they step on the dance pad with the correct timing, the score will be display on screen
- A DDR IP custom core will be developed to control the game flow (start/stop the game and the music, tell the VGA when to draw which arrow, keep score based on user pad pressed timing)
- Onboard AC97 codec will be used to playback music during game play
- Multiple difficulty settings
- Allow different mp3 to be loaded into memory
- Explore different ways of synchronizing arrow generation with music

1.2 Redefined Goals of the Project

After consultation with our lab TA (Mark), the goals were modified:

- DDR IP core has been replaced by software control
- Beat detection algorithm is used to generate arrows that is synchronized with any music fed into AC97

1.3 System Diagram

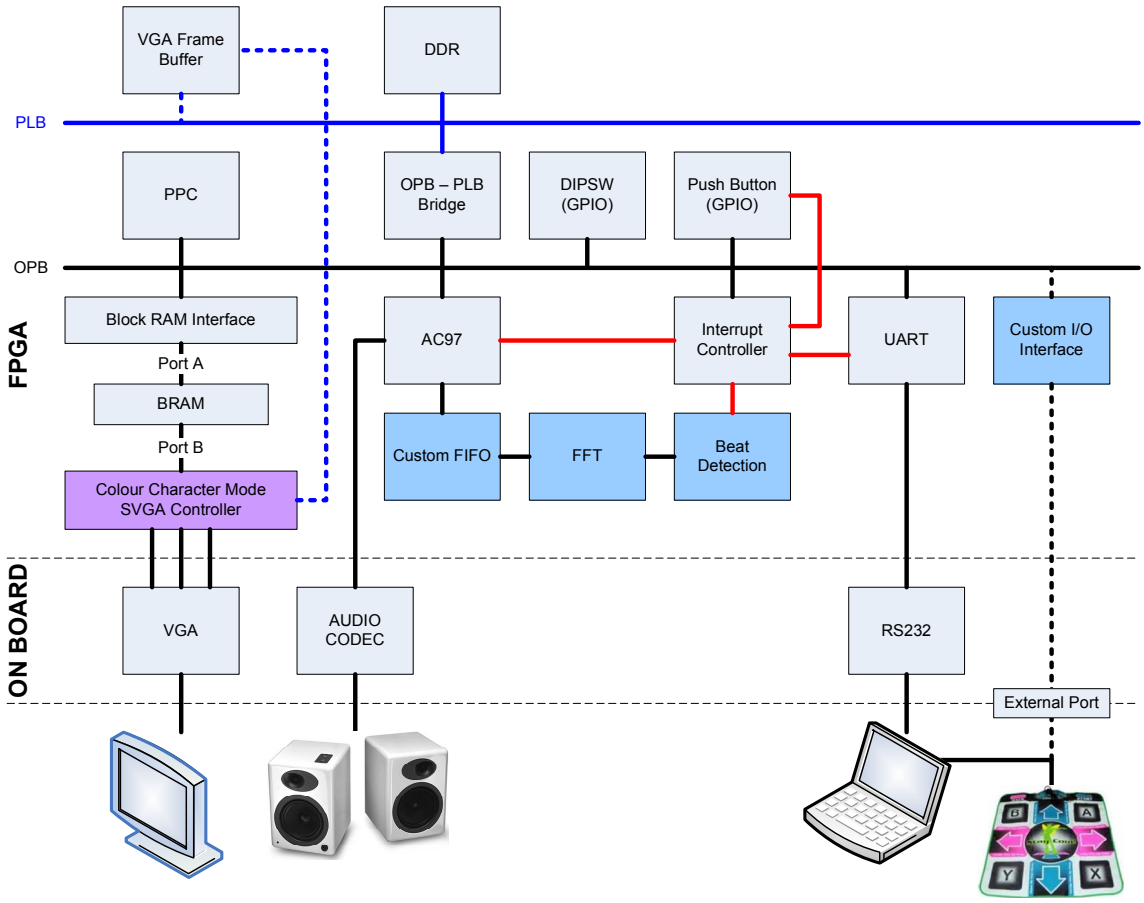

Figure 1: Overall System Diagram

Note: Dotted lines indicate modules that are not fully functional

1.4 High Level Description

Block	Description
SVGA	Obtained from Xilinx's Colour Character Mode Demo [1]. Modified to draw arrows instead of just characters.
DDR-RAM	Generated from XPS project wizard, used to record and playback music during gameplay
AC97 Controller	Obtained from the Xilinx IP repository
Custom AC97 FIFO	Extract audio samples from codec's serial interface
1024 point FFT	Generated through CoreGen. It takes 1024 samples of PCM data and convert them to frequency
Beat Detection Module	Custom module written in verilog to detect beat in streaming music
Interrupt Controller	Generated from XPS project wizard, for software to regain control upon events such as user input.
UART RS232	Generated from XPS project wizard, used to capture user input for gameplay
Dip Switch (OPB-GPIO)	Generated from XPS project wizard, used to control game difficulty settings
Push Buttons (OPB-GPIO)	Generated from XPS project wizard, used to capture user input for gameplay (before UART is implemented)
Custom I/O Interface	(Not integrated) Custom module written in verilog to do handshaking with a PlayStation2 controller
VGA Frame Buffer	(Not integrated) Obtained from Xilinx's SlideShow Demo [2]. For drawing a background image on VGA monitor

2. Outcome

The main goals of the project have been satisfied. The game is controlled through software instead of hardware as it was easier to do. The input consists of a PS2 dance pad, which is connected to the PC, and uses the UART to send its inputs. The char-mapped SVGA block displays static hollow arrows on top of the screen and draws the coloured arrows from the bottom of the screen to the top of the screen. The score of the player is displayed on the right side of the screen. If the user presses the correct pedal on the dance pad, the score is incremented by 100. The dip switches are used to control the speed of arrows on screen.

The AC'97 codec stores the music in DDR memory, which is then retrieved after a small delay for playback. The beat detection block uses the streaming FFT block to detect beats in music and increments score by 1. The accuracy of beat detection is not as accurate as simulated results and we suspect that it is due to beat detection coefficient not being set correctly. The arrow generation is done in software using a random algorithm, instead of using the interrupt signal generated by the beat detection module on presence of a beat.

The PS2 dance-pad was not connected to the board directly due to voltage and capacitance issues. The background image was not displayed as we were unable to integrate the VGA-frame buffer in the time frame.

2.1 Future Improvements

- beat detection coefficient set externally for better compatibility with songs
- use more frequency bands for beat detection to allow detection beat of different instruments
- use VGA frame buffer to implement video overlay
- get the beat interrupt signal to drive the generation of arrows (software code)

3. Detailed Description

SVGA block – The SVGA block was retrieved from the Xilinx Demo. The SVGA block connects to a block ram on one side and drives the output VGA signals such as HSync, VSync and pixel clock on the outside interface. The pixel clock was generated from DCM_0 using a 4/10 ratio for CLOCK_FX to generate a 40Mhz signal. Since all block rams are dual ported we used a BRAM interface to connect to PortA of BRAM to write the data, and used PortB to read the data for the SVGA. The hardware mapping of ASCII characters was stored in the CHAR_GEN_ROM.v. Special Arrows were drawn by hand and then added to the ASCII code map from 0x80 to 0xBC. One drawback of using this module however is that only bits 12-15 were used for color so we could only get 16 different colors, which was sufficient for our use. For future, if more colors are needed than they can be added to CLUT.v and COLOR_CHAR_MODE_SVGA_CTRL.v can be modified to use more bits for color. The module available on Xilinx site is configured to be used as a blackbox and a systhesized .edf files is provided with the module, which is what XPS uses by default. The pcore configuration files were modified to synthesize the verilog files to get the netlist. As a side note, if only new shapes need

to be added to the module then it is not necessary to synthesize the block again, the ROM file parameters can be set directly in the system.ucf file.

DDR-RAM-PLB – The DDR-RAM is connected to PLB Bus. We use a plb-opb bridge to connect it to the AC'97 core which was connected to OPB bus. We used software to write the samples from AC'97 FIFO to the DDR-RAM. The samples were retrieved after a short delay for playback.

OPB-AC'97 Controller - AC'97 is connected to outside DAC for playback, it is accessed through OPB. An interrupt is generated when the FIFO is half full. In the interrupt handler the samples are read from the FIFO and written in the DDR-Memory. Also in the same function, samples are read from Memory and written into the AC'97 for playback, by modifying the read pointer we can add an arbitrary delay to the playback. The VHDL files was modified to duplicate the data signal can be propagated through to the custom AC'97 FIFO for performing FFT.

Custom AC'97 FIFO – Since our beat detection scheme requires a continuous stream of input audio data, we need a custom FIFO interface to get the data from AC97 codec instead of getting data from OPB. A simple state machine is used to communicate with the AC97's serial interface to extract a 16bit PCM data (Left Channel) at 48kHz. The Sync signal is also used to generate a 48kHz clock for use with our FFT module.

1024 point FFT – This block was generated through Coregen. The selected block uses

the pipeline, streaming I/O implementation, with the following modes: unscaled option, convergence rounding, and natural order output ordering with CE pin added. The input is a 16-bit audio sample from Custom AC'97 FIFO, it performs a 1024 point FFT on the input. The output is 1024 complex numbers that represent the frequency spectrum.

Beat Detection Module –The input is the lower sub band of the frequency spectrum (first 32 numbers) and the output is a whether the 1024 samples that went into FFT has a beat. [3]

The beat detection algorithm is as follows: First, 1024 audio samples are obtained from the AC97. Then it is sent to the 1024 FFT which outputs 1024 values of the frequency spectrum. For each sub-band of 32 complex numbers, the summation of power of the sub-band is computed. It is then passed to a shift register to store a history of their values. Approximately 1 second of history is stored in these registers. The latest sub-band power value is then compared with the average of the history. A beat is generated when the value is greater than a certain threshold. If we plot the beats, we would get a few peaks due to noise.

To clean up the signal, a moving average is used. The time since the last beat is recorded and also stored in a history array. The moving average of the history is then compared with the newly generated time value. If the beat occurs a certain time period from the previous beat, then it is considered a valid beat. And if we plot the valid beats on a graph, we would get clean stems of beat signals.

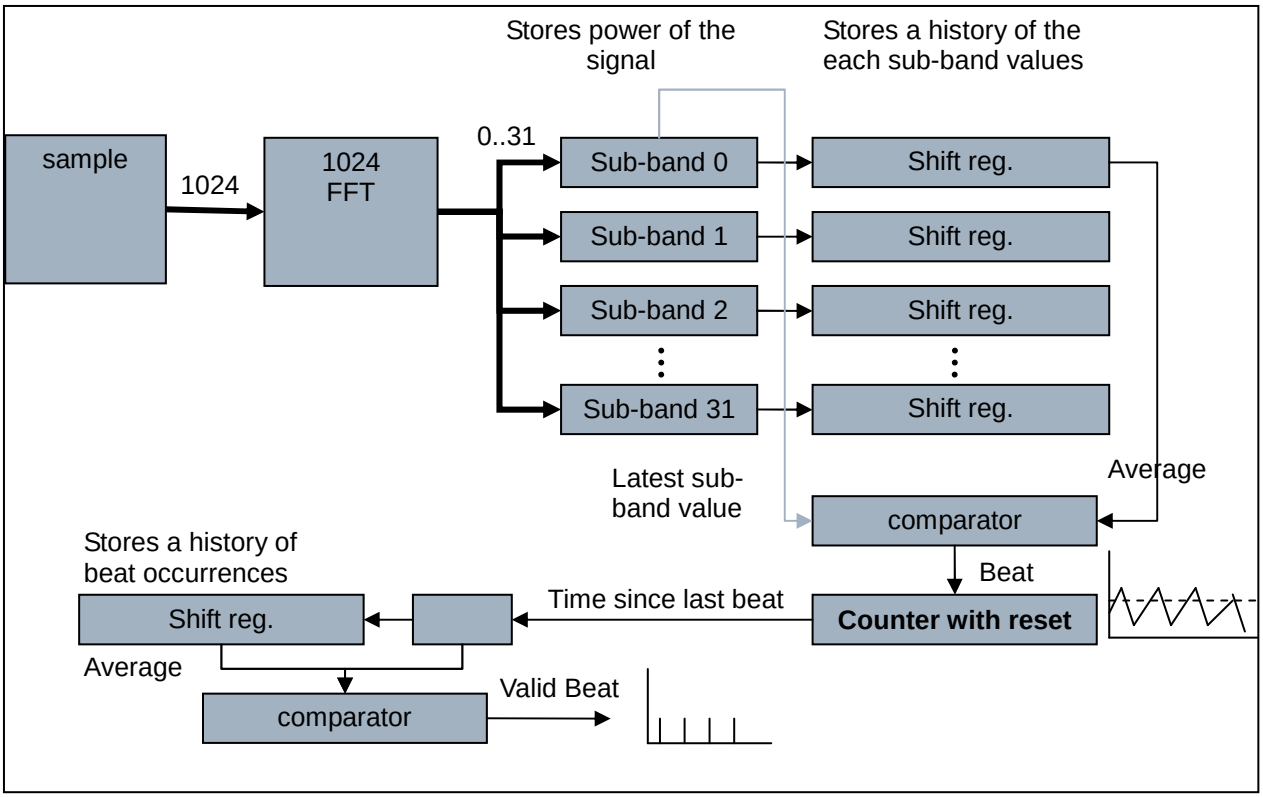

Figure 2: Beat detection algorithm

OPB-Interrupt Controller – The following devices in the system generate interrupts: the OPB-AC'97 Controller, the UART RS232 Interface, the Push button OPB-GPIO Interface and the Beat detection module. Each interrupt generated is serviced by an interrupt service routine that is written in software.

UART RS232 Interface – This interface is used as the input for the DDR-gameplay. On the PC side, the Playstation2 DDR dance pad is connected to USB through a PS2-USB connecter. The dance pad is recognized as a joystick. A small script is written and used to map the joystick inputs to keystrokes which are then transmitted through the RS232 interface. In the RS232 interface, each key press causes an interrupt. In interrupt

handler, the received keystroke is checked if the key press occurred at the correct time by comparing the value of the moving arrow with the value of the static arrow.

Dip Switch OPB-GPIO Interface – The DIP switches interface was connected to the OPB bus. Its inputs were used to set the difficulty of the game. The value of the dip switch is checked every iteration of the screen update.

Push Buttons OPB-GPIO Interface – The push button module is connected to the OPB bus. The inputs for the gameplay were initially these push buttons. The push button causes an interrupt at which time an interrupt handler checks for the position of the moving arrow as described above (UART RS232 Interface).

Custom I/O Interface – Our original design includes this I/O Interface to connect a PS2 (PlayStation 2) controller directly to the OPB using the onboard external port. This module handshakes with the controller, which has a serial interface, and stores the status of the controller to a register (buttons pressed down), which can then be read by software through OPB. However, due to electrical problems, which is potentially caused by the board not being able to provide the current necessary to drive the controller.

Video Overlay – The VGA framebuffer is used for video overlay and is connected to PLB. It reads data in the form of a .bmp file from DDR memory and displays the picture on the VGA. The outputs to the block are similar to the ones in char-mode-VGA. In our design, instead of the output to drive the VGA directly on board, the output was used as

an input to the char-mapped SVGA block. The SVGA block was modified to take these signals as input and multiplex them with character mode to get the video overlay effect. When we tried to integrate the VGA framebuffer and tried to test it by making those pins external, the display did not change when the VGA was driven directly. Due to limited time-frame we shifted our attention to other tasks and the video overlay is not present in the final design.

4. Software Control

The dynamic arrows on the screen are drawn using a loop. For each column of arrows, a link list is declared and the arrows are stored as a struct in it. A function called generate_arrows is used to randomly generate a row containing 1 to 4 arrows. The arrows generated are added to the link list. At the end of each iteration of SVGA loop, the screen is updated with the contents of the link list. A sleep function is used at the end of each iteration to control the speed of arrow. Dip-switches are used to modify the interval of sleep and thus change the difficulty of the game.

The dance pad, connected to PC, is used to cause an interrupt through UART when a button is pressed. In its interrupt handler, the position of tail arrow of the corresponding link list is checked. If the height of the tail arrow of the linked list is near the top of the screen then the score is incremented by 100. The beat detection module is used to detect the beats in the audio samples input and would cause an interrupt when a beat occurs. The score is incremented by 1 for every beat.

5. Description of Design Tree

Directory/File	Description
./__xps	Option files for bitinit, libgen, simgen and platgen
./blkdiagram	Block diagram generated by XPS
./bootloops	PPC boot file
./code/main	The main software control program
./code/svga	The arrow drawing functions
./data/system.ucf	System Constraints files
./doc	Documentation files
./etc	Option files for bitgen and downloading
./lib	Xilinx Processor IP library
./pcores	Custom core directory containing the SVGA character mode IP, the FFT core, the framebuffer core, the beat detection IP and custom AC97 FIFO IP
./ppc405_0	PPC405_0 processor and device drivers
./ppc405_1	PPC405_1 processor and device drivers
./simulation	Simulations containing matlab simulation of the beat detection algorithm and FFT core simulation
./system.xmp	XPS project file
./system.mhs	System Hardware Specification file
./system.mss	System Software Specification file

6. References

5.1 Online references

[1] Colour Character Mode SVGA Demo (XUPV2P Demonstration Design)

http://www.xilinx.com/univ/xupv2p_demo_ref_designs.html

[2] Xilinx's SlideShow Demo

http://www.xilinx.com/univ/xupv2p_demo_ref_designs.html

[3] Beat detection algorithm

http://www.yov408.com/html/articles.php?p=1

5.2 Datasheets

1. FFT Core Datasheet:

%Xilinx%\coregen\ip\xilinx\dsp\com\xilinx\ip\xfft_v4_0\doc\xfft_ds260.pdf

2. DCM Datasheet:

%EDK%\hw\XilinxProcessorIPLib\pcores\dcm_module_v1_00_a\doc\dcm_module.pdf

3. GPIO Datasheet:

%EDK%\hw\XilinxProcessorIPLib\pcores\opb_gpio_v3_01_a\doc\opb_gpio.pdf

4. Interrupt Controller Datasheet:

%EDK%\hw\XilinxProcessorIPLib\pcores\opb_intc_v1_00_c\doc\opb_intc.pdf

5. Block Ram Datasheet:

% EDK%\hw\XilinxProcessorIPLib\pcores\bram_block_v1_00_a\doc\bram_block.pdf

6. DDR PLB Datasheet:

% EDK%\hw\XilinxProcessorIPLib\pcores\plb_ddr_v2_00_a\doc\plb_ddr.pdf

7. OPB UartLite Datasheet:

% EDK%\hw\XilinxProcessorIPLib\pcores\opb_uartlite_v1_00_b\doc\opb_uartlite.pdf

8. SVGA IP datasheet:

pcores\COLOR_CHAR_MODE_SVGA_CTRL_v1_00_b\doc\ASCII_code_map.pdf