ECE532 Group Report

Nerdy Musical Keyboard

Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

April 7th, 2009

Sida Shen
sida.shen@utoronto.ca

ShiKuan Yu
shikuan.yu@utoronto.ca

Eric Pai
eric.pai@utoronto.ca

1 Introduction

In this report, we documented the work completed for our ECE532 project - "Nerdy Musical Keyboard". We designed and implemented a system on FPGA that allows the user to play a game on keyboard that simulates playing the piano. In the game, the user follows the "karaoke-like" prompts on the monitor, and enters the key that is displayed. As user enters the key, a corresponding musical tone is generated in hardware and played on the speakers using the AC97 Codec. The game keeps track of how well the user follows the displayed prompts by keeping score, which is displayed throughout the game.

The remainder of this report is organized as follows. Section 2 documents the goals of our project. A block diagram is included to introduce the overall system. IPs that used in the system are briefly described and summarized in a table. Section 3 presents the outcome of our project. All basic functions set in project goals were successfully completed. In addition to the basic functions, one of the optional features, game mode, was completed. Moreover, further improvements are suggested. Section 4 provides a detail description of the hardware components in this project, focusing on the developed Audio Output Controller. Section 5 presents a detail description of the software components including the Game Module, Keyboard Input Handler, Video Output Controller, and Audio Output Controller. Finally, section 6 provides the details about the software structure of our project.

2 Overview

2.1 Goals of the Project

The goal of the project is to implement a musical keyboard system on FPGA that allows the user to play a game on keyboard that simulates playing the piano. The system involves three input and output devices. They are computer keyboard, speakers, and a monitor. Communications among the devices are controlled by three function modules. They are Keyboard Input Handler, Audio Output

controller, and Video Output Controller. Requirements for these function modules are defined as the following:

The Keyboard Input Handler should be implemented in software. It must be capable of storing the input data from a keyboard whenever a key is pressed by the user. The data is then processed and past to the Audio Output Controller by the handler.

The Audio Output Controller should be designed in hardware. It must be able to receive the data past from the Keyboard Input Handler, and then generate the corresponding musical tone. The musical tone must be played for a set duration, and the duration must not be affected by other inputs of the same time period. Finally, the generated tone must be played using the AC97 Codec.

The Video Output Controller should also be implemented in software. It must be able to prepare the contents that need to be outputted and send them to On-board XSGA Output for displaying. Prepared contents should include location and colour information as well.

In addition to these basic function modules, some optional features are proposed and listed below that can be added on later to make the system fancier if time permits.

- Game Mode that scores player if the key he/she pressed matches with what has been displayed on the screen. Game Mode also keeps the score and displays it at the end of game.
- Music Composer which allows user to compose music by typing a sequence of letters
- Options for user to adjust music's tempo, key, and instrument tones

2.2 System Overview

The overview of the system that was initially proposed is shown in Figure 1. Keyboard inputs are received from RS232 serial port. Whenever there is an input appears in the serial port, an interrupt is requested. Then an interrupt handler function is invoked, in this case is the Keyboard Input Handler module. This module stores the input data, process it and send it to Audio Output Controller. The Audio Output Controller receives user input from the Keyboard Input Handler, generates the

corresponding tone, and provides a synchronous interface to the AC97 Codec by sampling the playback data and control signals. The video Output Controller prepares displaying contents and sends it to Port A of BRAM through a BRAM Interface Controller for temporary storage. Then the SVGA controller reads the stored displaying contents through Port B of the BRAM, and outputs them on to the monitor through the XSGA Output chip on the board. The Game Modules keeps track of how well the user follows the displayed prompts by score the correct inputs, and display the total score at the end of the game. All the modules that implemented in software are complied by MicroBlaze Processor. Complied instructions and data are stored on two BRAMs (for simplicity only one BRAM is shown in the figure). Information on BRAMs is accessed using BRAM Interface block. Communications among the hardware components are done through the PLB bus. Brief descriptions of IPs used in the system are summarized in Table 1.

However, as the Audio Output Controller IP was not implemented successfully in hardware, a modified implementation in software was adopted. The overview of the modified system is shown in Figure 2. Hardware components that got replaced are highlighted in dashed box. Sounds are outputted using a hardware controller, opb_ac97_v2_00_a, provided on the Xilinx website. Since this core was designed to communicate with OPB bus, a bridge IP is used to connect the two buses together, so that the data can be sent to the ac97 core through PLB bus. In addition to these changes, Audio Output Controller is now implemented in software, which takes inputs from the Keyboard input handler, generates a tone in samples accordingly, and output the sound through AC97 Controller.

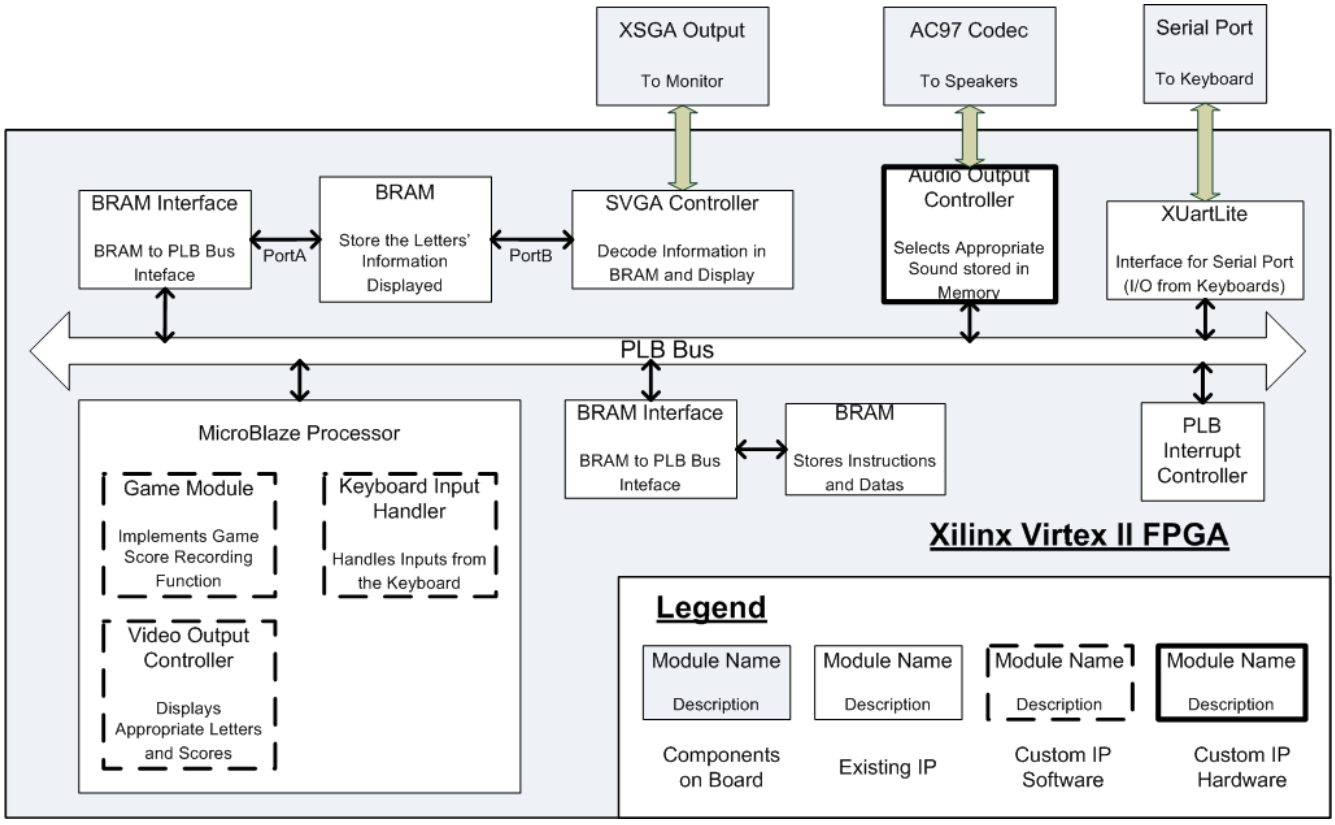


Figure 1 Proposed System Block Diagram

IP Name	Hardware/ Software	Used/ Modified/ Created	Function
MicroBlaze Processor	Hardware	Used	Drives the system, see section 4.1
Processor Local Bus (PLB)	Hardware	Used	System Backbone, see section 4.2
Game Module	Software	Created	score the correct inputs, see section 5.4
Keyboard Input Handler	Software	Created	Handles user input from the keyboard, see section 5.1
Video Output Controller	Software	Modified	Prepare displaying contents, see section 5.3
Audio Output Controller	Hardware	Created	Handles tone generation and playback, see section 4.4
AC97 Codec	Hardware	Used	Audio DAC, see section 4.3
PLB BRAM Interface	Hardware	Used	Passes data from PLB buss to BRAM
SVGA Controller	Hardware	Used	Obtained from Xilinx's Slide Show Demo [3] draws characters on the monitor
XuartLite	Hardware	Used	Receives input from serial port
PLB Interrupt Controller	Hardware	Used	Acts on special events, e.g. user inputs

Table 1 Brief Description of IP's used in Proposed System shown in Figure 1

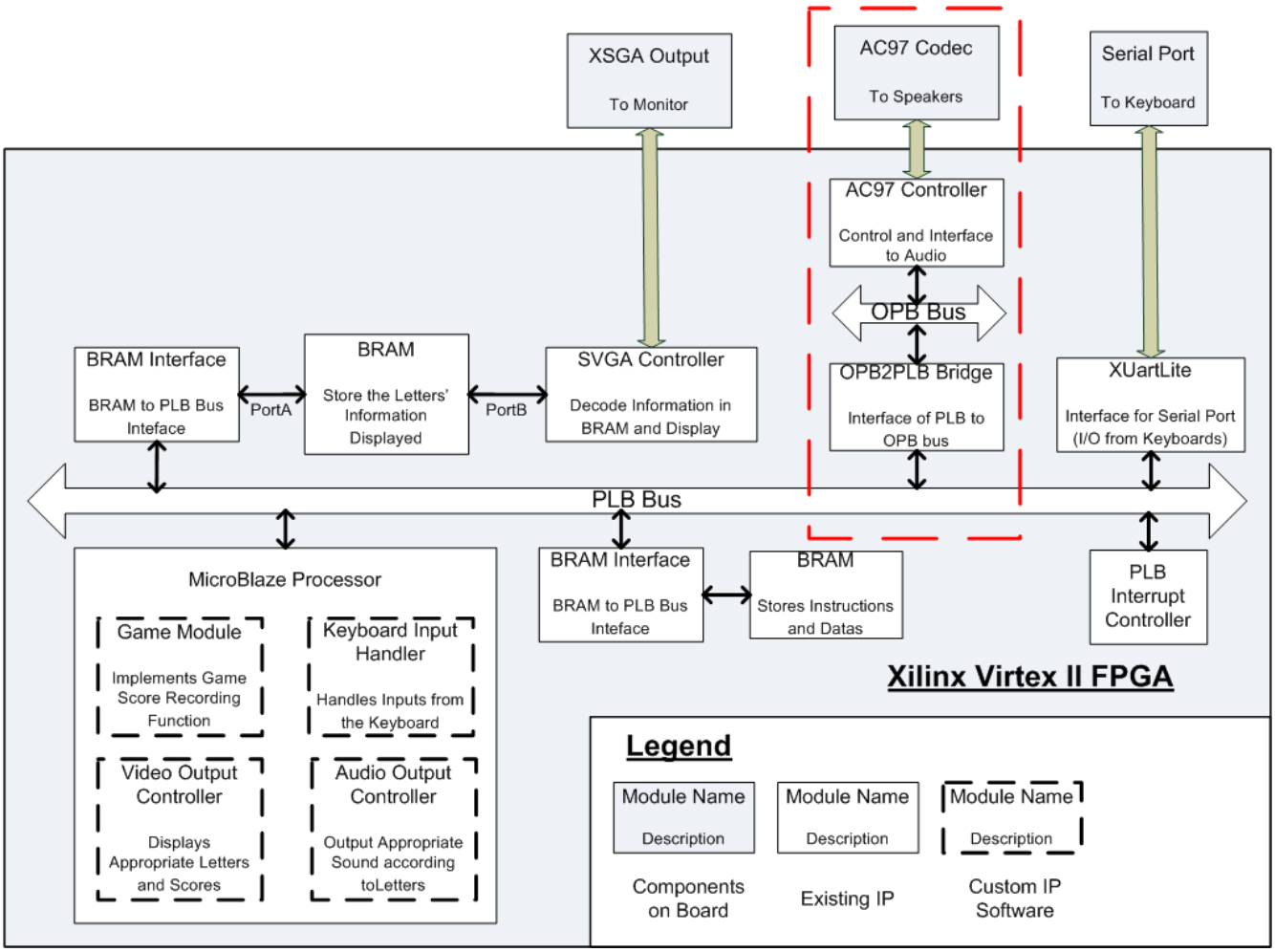


Figure 2 Modified System Block Diagram

IP Name	Hardware/ Software	Used/ Modified/ Created	Function
Audio Output Controller	Software	Modified	Handles tone generation and playback, see section 5.2
OPB2PLB Bridge		Used	OPB and PLB interface

Table 2 Added IP's for Modified System Block Diagram shown in Figure 2

3 Outcome

The basic functions in the project proposal were all successfully completed. They include: displaying user prompt on the monitor, taking user input from the keyboard, generating corresponding tone, and playing the tones using AC97 Codec. However, since the custom hardware IP was not functioning properly, the Audio Output Controller was implemented in software instead. The game mode, which was an optional feature, was also successfully implemented. The other optional features were not implemented due to time constraints and other challenges encountered.

3.1 Future Improvement

Evidently, the project is easily expandable. Firstly, the hardware implementation of the Audio Output Controller can be completed. Also, the project could use better user interface to control the game, more interesting and smoother graphic animations, and more types of musical tones (of different musical instruments).

3.2 Custom IP Simulation Results

Even though the hardware Audio Output Controller did not function properly on the board, the simulation results reflect the achievement of correct functional behavior. As clearly seen in the simulation results below, upon user input (signal *key* changes from 0 to 1), the AC97 Controller output *oAC97_SData_Out* changes corresponding to the *speaker* signal of Tone Generator:

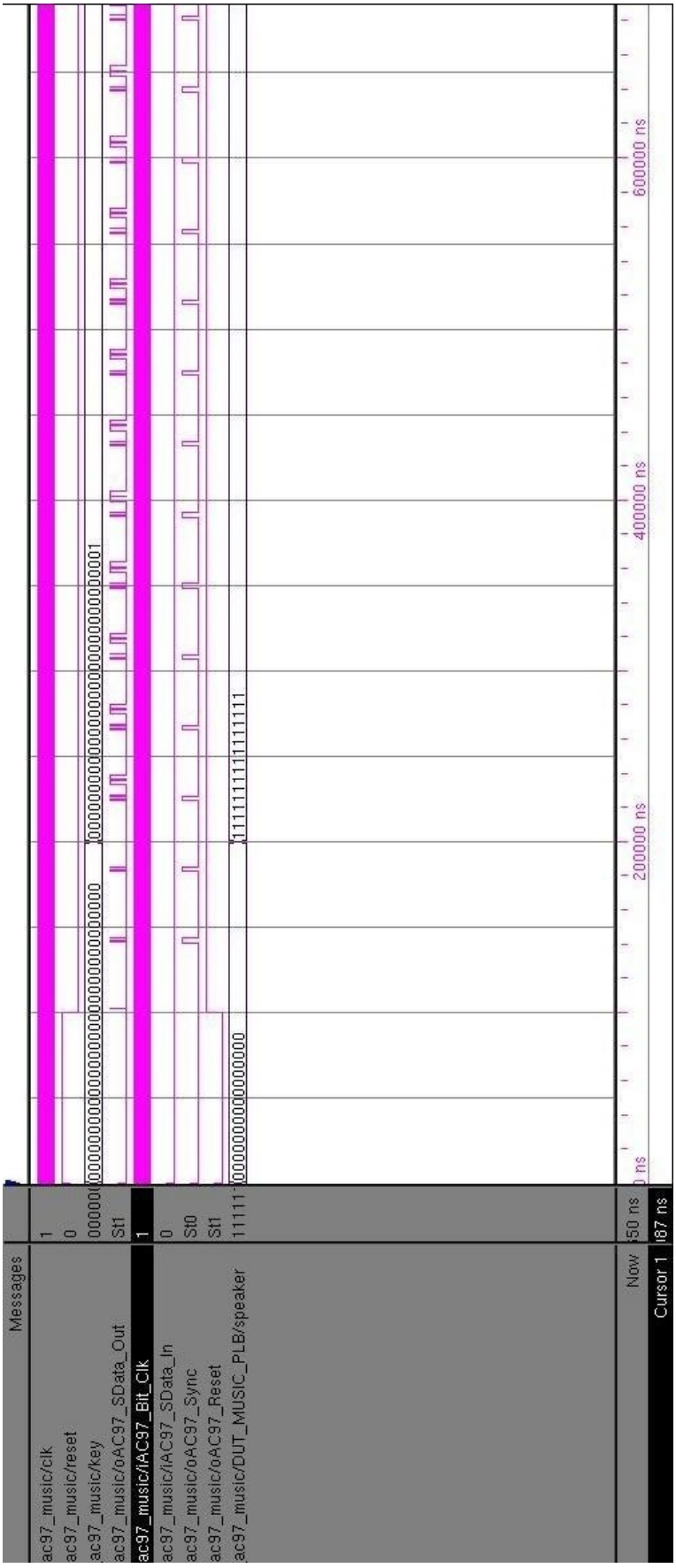


Figure 3 Audio Output Controller Simulation Diagram

4 Hardware Modules

This section provides detail descriptions of the used, modified, and created hardware components. The created IP, Audio Output Controller, is the main focus of this section.

4.1 MicroBlaze Processor

The project uses the MicroBlaze embedded processor, version 7.10d. Please refer to the "Xilinx MicroBlaze Processor Reference Guide", available on the Xilinx website, for more information on the processor.

4.2 Processor Local Bus v4.6

The Processor Local Bus (PLB) v4.6 provides bus infrastructure for connecting PLB masters and slaves into an overall PLB system. IP version v1.03a was used for the project. Please refer to the "Xilinx Processor Local Bus datasheet", available on the Xilinx website, for more information on the bus. In this project, PLB is used to connect Audio Output Controller, XuartLite, PLB Interrupt Controller, BRAM Controller, and MicroBlaze Processor.

4.3 AC97 Codec

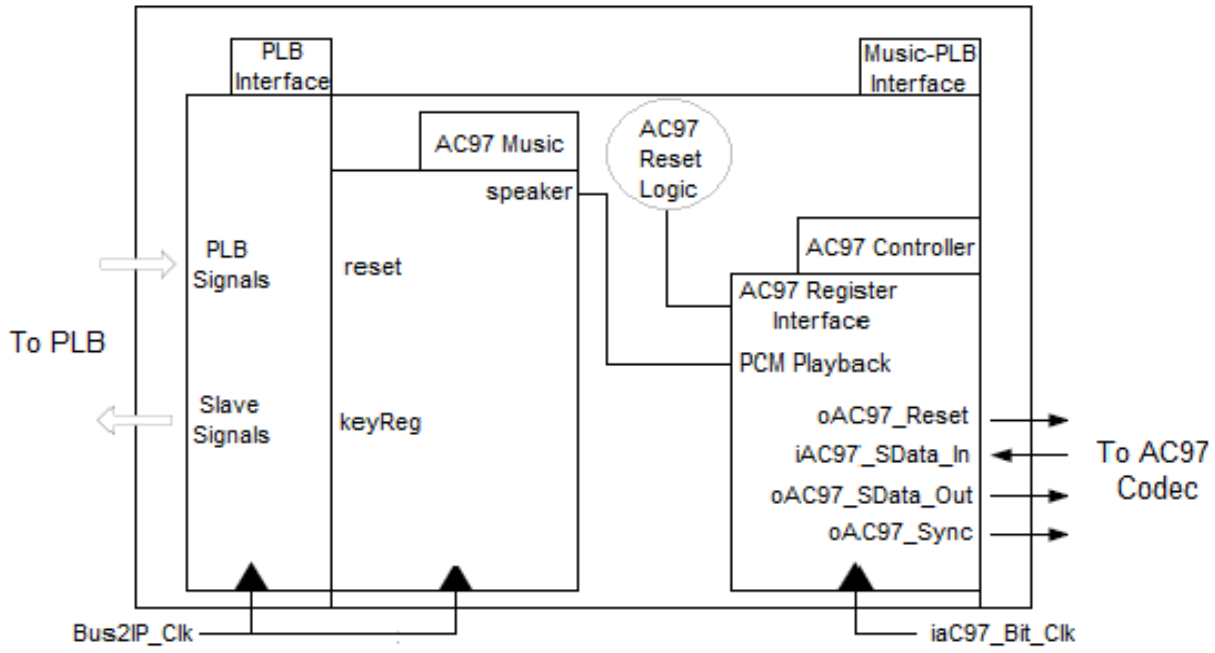
The AC97 Codec is an ASIC that is available on the FPGA board. It outputs the *iAC97_Bit_Clk* and *iAC97_SData_In* signals to the custom IP Audio Output Controller. In return, the custom IP provides the signals *oAC97_Sync*, *oAC97_SData_Out*, *oAC97_Reset* to the AC97 Codec. Once the codec resets into operational mode, the AC97 Controller can write to and read from the codec's registers, hence enabling playback and recording using the codec.

4.4 Audio Output Controller (hardware)

Audio Output Controller is the IP core created for handling audio output for the project. The core is written in both Verilog and VHDL. It is composed of the AC97 Controller, PLB interface, and AC97 Music. The AC97 Music is further broken down into the Tone Generator and Duration Generator.

Functional Description

Figure 4 Audio Output Controller Block Diagram



Audio Output Controller I/O Signals

Signal Name	Interface	I/O	Description
Bus2IP_Clk	PLB	I	Bus to IP clock
Bus2IP_Reset	PLB	I	Bus to IP reset
Bus2IP_Addr	PLB	I	Bus to IP address bus
Bus2IP_Data	PLB	I	Bus to IP data bus
Bus2IP_BE	PLB	I	Bus to IP byte enables
Bus2IP_RdCE	PLB	I	Bus to IP read chip enable
Bus2IP_WrCE	PLB	I	Bus to IP write chip enable
IP2Bus_Data	PLB	O	IP to Bus data bus
IP2Bus_RdAck	PLB	O	IP to Bus read transfer acknowledgement
IP2Bus_WrAck	PLB	O	IP to Bus write transfer acknowledgement
IP2Bus_Error	PLB	O	IP to Bus error response
iAC97_Bit_Clk	AC97	I	Input clock for AC97 Controller. Provides a 12.288 MHz clock for the AC Link
oAC97_Sync	AC97	I	Output to the LM4550 codec. Defines the boundaries of AC Link frames. Sync is a 48 kHz positive pulse with a duty cycle of 6.25% (16/256)
oAC97_SData_Out	AC97	O	Output for AC Link Frames from AC97 Controller to the LM4550 codec
iAC97_SData_In	AC97	O	Not Used
oAC97_Reset	AC97	O	This active low signal causes a hardware reset of the LM4550 codec which returns the control registers and all internal circuits to their default conditions.

Table 3 Audio Output Controller I/O Signals

PLB Register Description

The PLB slave has two registers available, but the project uses only one of them: *slv_reg0* (or *keyReg* in AC97 Music). The 32-bit register stores the user input from the keyboard, which corresponds to specific musical tones. The register's address offset is 0x4.

The other register is available for various testing. It is also a 32-bit register with address offset 0x0.

Additional Constraints

The Audio Output Controller communicates with the AC97 Codec. This is made possible by physical connections between the pins on the codec ASIC and the pins on the FPGA. Please see the UCF file for the entries that were added.

PLB Interface

The PLB interface for Audio Output Controller is implemented in user_logic and ac97_plb_slave, both modules are written in VHDL. The modules are modified versions of the ones auto-generated by Xilinx. They allow for register read and write's, and other communications between the PLB and the Music-PLB Interface.

Music-PLB Interface

The Music-PLB Interface connects the AC97 Music component with the PLB interface that is in written in VHDL. Even though the other modules are written in Verilog, the PLB interface was chosen to be implemented in VHDL because the auto-generated interface backbone is much more thorough in VHDL. Hence, the implementation is simplified. The Music-PLB Interface also includes a state machine that setup the AC97 Codec before attempting playback using the codec. Playback occurs when the tone is available as the signal *speaker* of the AC97 Music module, which is sent to the AC97 Controller.

AC97 Music

The AC97 Music module instantiates the Tone Generator and Duration Generator modules, and passes on the register that contains the user input. It exists to simplify simulations and testing.

AC97 Music – Tone Generator

The Tone Generator reads the value in the register *keyReg*, which corresponds to a specific tone. The frequency of the specified tone is first retrieved from a database, then generated using clock dividers. The resulting waveform generated is a square wave of the specified amplitude, frequency, and 50% duty cycle. Plans to implement waveforms of triangular and sinusoidal nature were abandoned due to difficulties encountered when testing on Xilinx board. The generated waveform is output through signal *speaker* whenever the *play* signal from Duration Generator is high. Tone Generator also ignores new tones that become available while the old tone is being played.

AC97 Music – Duration Generator

The Duration Generator produces a signal *play* to enable activity of the Tone Generator. Upon user input, *play* is high for a predefined duration, which is set to one second by default.

AC97 Controller

The AC97 controller, version 2.00.a is originally developed by Mike Wirthlin. However, as the original version is a standalone OPB peripheral, it was modified to integrate into the Audio Output Controller, which already has its PLB slave interface.

5 Software Modules

5.1 Keyboard Input Handler

There are two parts to the implementation of Keyboard Input Handler. The first part includes initialization and self- test of the interrupt and UartLite IPs. As introduced in the overview in Section 2, an interrupt is requested whenever keyboard inputs are received from RS232 serial port. Therefore, functions from the drivers of thes IPs are used to initialize them properly. Initialization steps including resetting all the interrupts, initializing the drivers, setting and connecting the interrupt handler function, starting the interrupts, and finally, enabling interrupts in Uarlite and the processor.
Second part of the modules stores the input data, processes it and passes the information to a tone generate function. These steps are done in the interrupt handler function.

5.2 Audio Output Controller

Audio Output Controller was completed using functions from the driver of the opb_ac97_v2_00_a pcore, which is provided on Xilinx website. Tones are generated by 256 samples. The 256 samples are created from a sine wave table which is stored in an array. This sine table was found from the Xilinx forum [4]. In addition to this tone generation function, the Audio Output Controller includes an initialization function which sets the registers of the ac97 pcore to an appropriate value for it to function properly.

5.3 Video Output Controller

The Video Output Controller is divided into two main parts: Character/Number Construction and Keynotes Display Algorithm:

Character/Number Construction

Each of the 26 English Characters and 10 numbers are constructed by a 10*10 integer array. Each element of the array represents 8*8 pixels. In total, each character has a resolution of 80*80. The color of characters is determined only by bits 12-15 of the integer element. So we could only get 16 different colors, which was sufficient for our use. For the future, if more colors are needed then we can modify the default SVGA definition file CLUT.v and COLOR_CHAR_MODE_SVGA_CTRL.v.

Keynotes Play Algorithm

Keynotes are represented by alphabetical letters. To display the letters onto the screen, a screen pointer called pChar_Data_Wrd was declared, which initially points to the first pixel of the monitor. Then the function call drawLetter (int * pChar_data_wrd, int X, int Y, int Char, int color) will draw the corresponding "Char" from the character database described in section 5.3.1 to location X and Y with the specified color.

The algorithm for displaying the letters is as follow: First, the letters to be displayed are mapped to numbers and sorted in an integer array. Each line on the monitor contains six letters. Two lines are displayed at the same time. The initial two lines of letters will be displayed in white on the monitor, i.e. the first 2*6 = 12 elements from the array. Once the game started, the letter in white got highlighted by changing to other color. This is done by drawing the characters again with different colour at the same location. When the first line has been played, the system will check if the letters to be displayed in next line are available in the integer array. If they are not, i.e. reach the end of the song, the monitor will refresh a blank row (letters in background colour). Otherwise, the controller keeps displaying the letters until reach the end of the integer array.

5.4 Game Module

The Game Mode module in our design involves two parts: Keyboard Matching and Score System:

Keyboard Matching

To have the software checking correct key inputs, a Keyboard Matching system is constructed. One global variable called Global_key has been declared to convey information from Keynotes Play Algorithm to Keyboard Input Handler. Within one second after the character has changed the colour, the Global_key will be assigned to the number represents the highlighted letter. If the user pressed a key within this one second, the Keyboard Input Handler will be invoked, and the value assigned to Global_key will be compared with the input from keyboard. If they are the same, we have a match.

Score System

The score system updates user's score whenever he/she presses a key. It is implemented as the following: One global variable called Global_Score has been declared. This variable is initialized to be zero for every new game. Every time the user pressed a key, the Keyboard Matching system will check to see if user input matches with the character just got highlighted. If it is the case, the game score will add one point to Global_Score, otherwise it remains the same.

6 Description of the Design Tree

Directory/File	Description
./_xps	Option files for bitinit, libgen, simgen and platgen
./blkdiagram	Block diagram generate by XPS
./code/main.c	The main software control program
./code/header.h	Header files including function declaration, constant assignment, etc
./code/control.c	All the Controllers' functions
./data/system.ucf	System Constraints file; external pins' assignment
./drivers	Includes the drivers for custom built core
./etc	Option files for bitgen and downloading
./microblaze_0	The processor
./pcores	Custom core directory containing SVGA character mode IP, OPB AC97 IP, and Audio Output Controller (hardware)
./simulation	ModelSim simulation of the hardware core
./system.xmp	XPS project file
./system.mhs	System Hardware Specification file
./system.mss	System Software Specification File
./README	Documentation

7 References

7.1 Xilinx Reference Designs

[1] Xilinx XUPV2P Demonstration Design
[2] Xilinx Video Decoder using VDEC-1
[3] Xilinx Slide Show using 256 MB DDR Memory

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

7.2 Online References

[4] Sine Wave Table for Tone Generation

http://forums.xilinx.com/xlnx/attachments/xlnx/Virtex/383/1/AC97_new1.zip

7.3 Past Projects

[5] P. Michaels, and K. Lui, *Real-Time Audio Mixer*, 2007

http://www.eecg.toronto.edu/~pc/courses/432/2007/projects/audiomixer.doc

[6] J.Puk, D. Gupta, and K.Chan, *Dance Dance Revolution on FPGA*, 2007

http://www.eecg.toronto.edu/~pc/courses/432/2007/projects/ddr.pdf

[7] C. Segulja, and B. Dai, *FPGA Implementation of the NES Audio Processing Unit*, 2008

http://www.eecg.toronto.edu/~pc/courses/432/2008/projects/nes-audio.pdf

7.4 Datasheets

[8] On-chip Peripheral Bus V2.0 with Arbiter (v1.10c) datasheet

%EDK%\hw\XilinxProcessorIPLib\pcores\opb_v20_v1_10_c\doc\opb_v20.pdf

[9] XPS Block Ram (BRAM) Interface Controller (v1.00a) datasheet

%EDK%\hw\XilinxProcessorIPLib\pcores\xps_bram_if_cntlr_v1_00_a\doc\xps_bram_if_cntlr.pdf

[10] Block RAM (BRAM) Block (v1.00a) datasheet

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

[11] OPB AC97 (v2.00a) datasheet

pcores\opb_ac97_v2_00_a\doc\opb_ac97.doc

[12] PLBV46 to OPB Bridge (v1.00a) datasheet

%EDK%\hw\XilinxProcessorIPLib\pcores\plbv46_opb_bridge_v1_00_a\doc\plbv46_opb_bridge.pdf

[13] XPS Interrupt Controller (v1.00a) datasheet

%EDK%\hw\XilinxProcessorIPLib\pcores\xps_intc_v1_00_a\doc\xps_intc.pdf

[14] XPS UART Lite (v1.00a) datasheet

%EDK%\hw\XilinxProcessorIPLib\pcores\xps_uartlite_v1_00_a\doc\xps_uartlite.pdf

[15] SVGA IP datasheet

Pcores\COLOR_CHAR_MODE_SVGA_CTRL_v1_00_b\doc\ASCII_code_map.pdf

[16] Digital Clock manger (DCM) Module

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

7.5 Manuals

[17] Xilinx University Program Virtext-II Pro Development System hardware Reference Manual

[18] Embedded System Tools Reference Manual

[19] Device Driver Programmer Guide