Design Report
EGC493 System on Chip
Fall 2013

Altera DE2-115 Audio Synthesizer

Group Members	Department	Major Contribution
Matt Miccio	CE	Design and Development

Course Professor: Dr. Rajeev Narayanan

Abstract

There are an innumerable amount of ways to create music, one of them being a digital synthesizer. This project uses an Altera DE2-115 Development board and it's FPGA to do just that. Sine waves at different frequencies were generated at different frequencies to play the seven notes (C, D, E, F, G, A, B) when a key on a PS/2 keyboard is pressed. Both Verilog HDL and C/C++ programming were used, as well as a CAD program to design the system. There were a variety of problems faced, but were overcome. With most projects, there are a few areas that could have been improved if I had more time.

Table of Contents

Introduction

Today in pop music, most of the instrumentals you hear are either created digitally or altered in some way, even the vocals. Many of the sounds are synthesized, or created, from digital signals. With a field programmable gate array (FPGA) and some crafty programming, a synthesizer can be made. The objective of this project is to do exactly that, create a basic audio synthesizer using an FPGA, specifically an Altera DE2-115 Development board sporting a Cyclone IV EP4CE115F29C7 FPGA. The development board has a very large selection of peripherals and features, but for this project only the Wolfson WM8731 Audio Codec, dual PS/2 port, red LED array, SDRAM, and SRAM modules are used.

The end goal of the project is to work such that when a PS/2 keyboard is connected to the PS/2 port and there are speakers connected to the Line Out port on the board, pressing a key on the keyboard will produce some audio output, a signal that produces a note on a scale. By design, the keyboard will act as a piano, playing one octave of the scale with seven white keys to represent the seven notes, and five black keys to represent the five sharps. Below, Figure 1 displays how the keys will correspond to the notes.

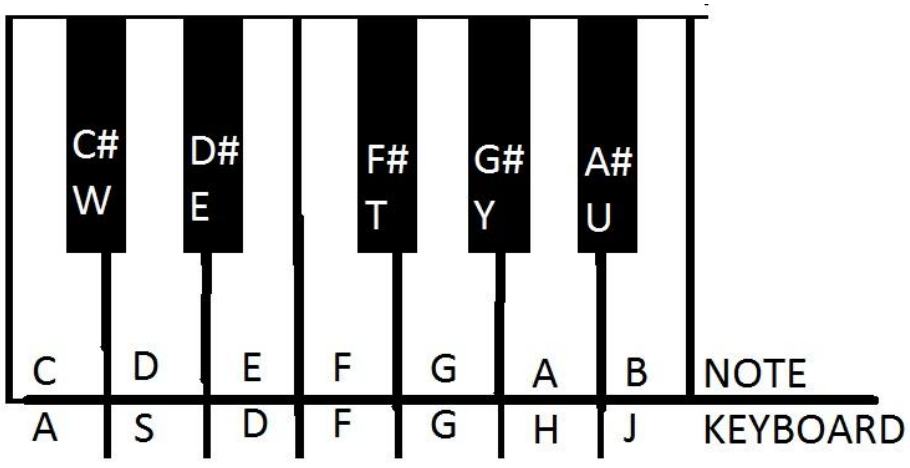


Figure 1 - A keyboard with the note (above) and the key pressed on the keyboard (below)

The tools that will be used to design and develop this project are Terasic DE2-115 System Builder, Quartus II, Qsys, and Eclipse. The System Builder will start the design process by quickly and easily setting up a top-level Verilog program that initializes the input, output, and bidirectional pins that will be used in the final design. Quartus II is a program that can compile and downloads programs to FPGAs to configure which components are being used. Qsys is used with Quartus II to produce the structural Verilog design used by the FPGA. Finally, Eclipse is a very powerful compiler that works with a multitude of programs and languages. In this case, an Eclipse tool designed specifically for Quartus II is used to write C/C++ programs that are downloaded to the DE2-115 board, executing desired programs using the pins and components designated in the other three design tools.

Procedure

There are two major parts to the procedure. The first carefully examines building the system. This includes the use of the System Builder, Qsys, and Quartus II. The combination of these will properly set up the system. The second part explains Eclipse and the C code used to finish the system.

Building the System

The first step to building a system is using the DE2-115 System Builder. This powerful tool, found on the CD that comes with the DE2-115 board, will quickly set up the top-level Verilog design file for using the desired components of the board. For the audio synthesizer, the major parts needed are clock, LEDs, audio, SDRAM , SRAM, PS/2, and the buttons. Figure 2 shows an image of the system builder interface.

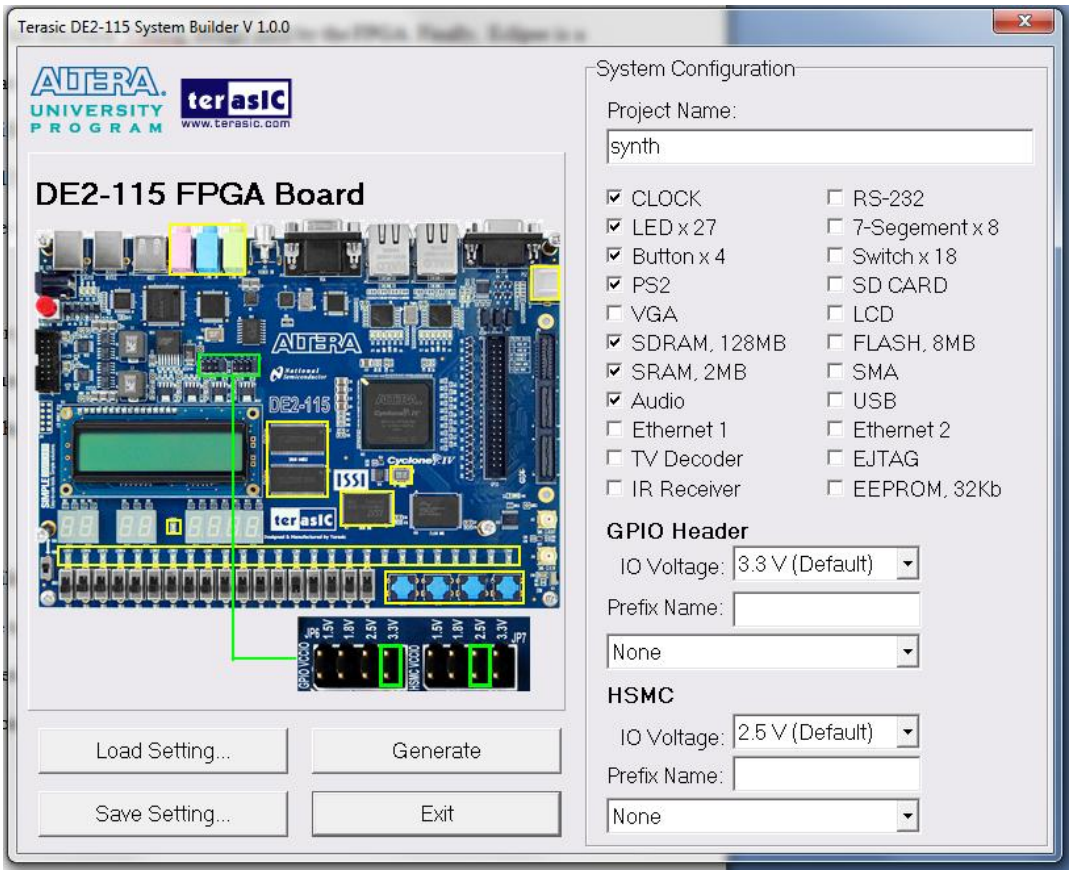


Figure 2 - System builder.

The next step is to hit "Generate." This will create a project file with the proper Verilog file setting up the variables that will be used in the design. Opening that up provides you with the code in Quartus II.

Before moving on to using Qsys and building the system's structural coding, it is necessary to install the Altera University Program Cores. There is a download link found on Altera's website. These University Program (UP) Cores install components to use on Qsys that weren't previously installed with Quartus II, including many components that are needed for this project. There are two ways to build the system after installing the UP: use the pre-made DE2-115 Media Computer that came with the install and removing the unnecessary parts, or adding them separately. The end product will be the same, but using the Media Computer is far easier and will provide all of the settings already configured for the project.

Designing the System with Qsys

For an easier explanation of the C code later on, we will use the Media Computer method. This is because the system base addresses are hardcoded when using the Media Computer, although configuring the C program to use a handmade system is just as simple. With that said, after loading up the Media Computer in Qsys, found in *<Quartus II install path>/University_Program/NiosII_Computer_Systems\DE2-115\DE2-115_Media_Computer\verilog\nios_system.qsys*, you can delete any components that are not needed. For this project, we will be deleting the video components, the HEX displays, character LCD, green LEDs, SD card, flash memory, JP5 expansion, PS2 port dual, USB, serial port, and the IRDA. After all is said and done, you are left with the CPU (Nios II Processor), sysid (System ID Peripheral), merged_resets (Reset Bridge), sys_clk (Clock Bridge), SDRAM (SDRAM Controller), SRAM (SRAM/SSRAM Controller), Red_LEDs (Parallel Port), Slider_Switches (Parallel Port), Pushbuttons (Parallel Port), PS2_Port (PS2 Controller), JTAG_UART (JTAG UART), Interval_Timer (Interval Timer), clk (Clock Source), External_Clocks (Clock Signals for DE-series Board Peripherals), AV_Config (Audio and Video Config), Audio (Audio), CPU_fpoint (Floating Point Hardware), and clk_27 (Clock Source). Inside of the parenthesis is the name of the component used in Qsys in the event the other method is used in creating your own Qsys file. The next series of figures (Figures 3 through 20) shows images of the settings of each Qsys component. The Qsys system shown with all of its connections, including interrupts is shown in Appendix A.

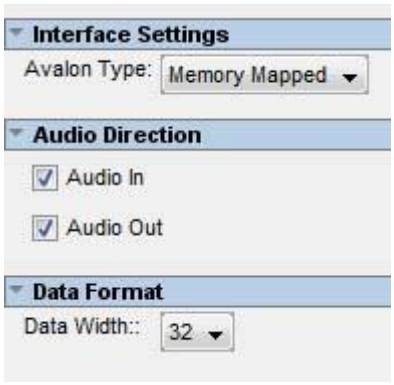


Figure 3 - Audio settings

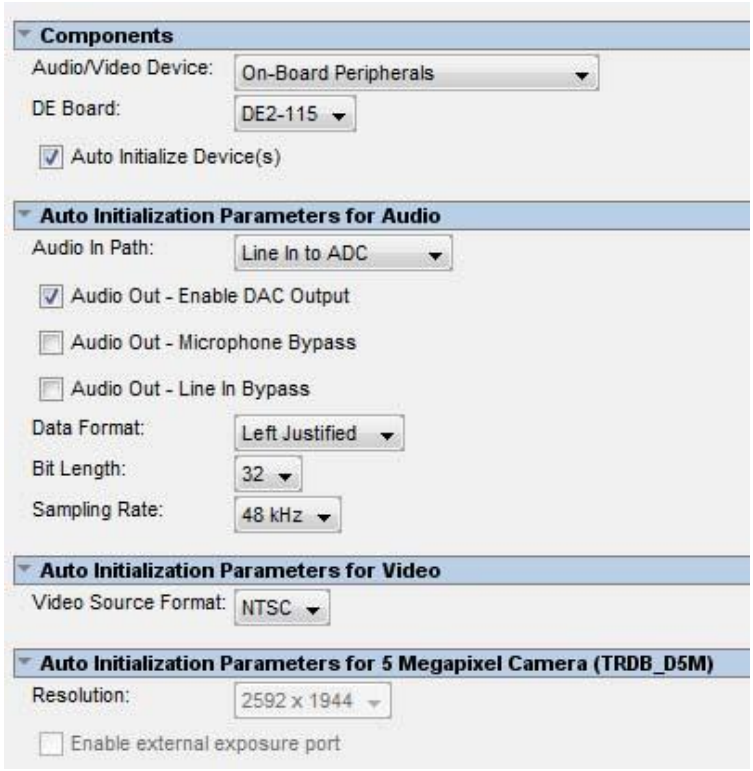


Figure 4 - AV_Config settings

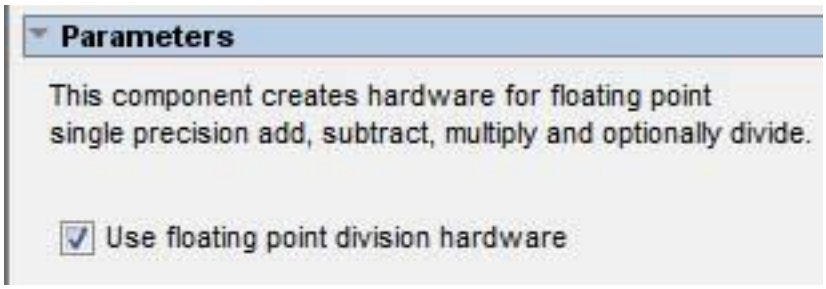


Figure 5 - CPU_Fpoint settings

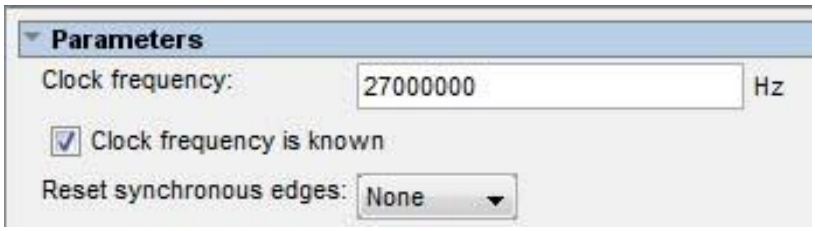


Figure 6 - clk_27 settings

Figure 7 - External Clocks settings

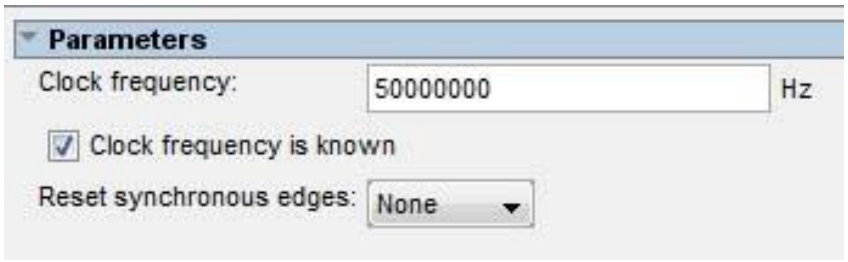


Figure 8 - clk settings

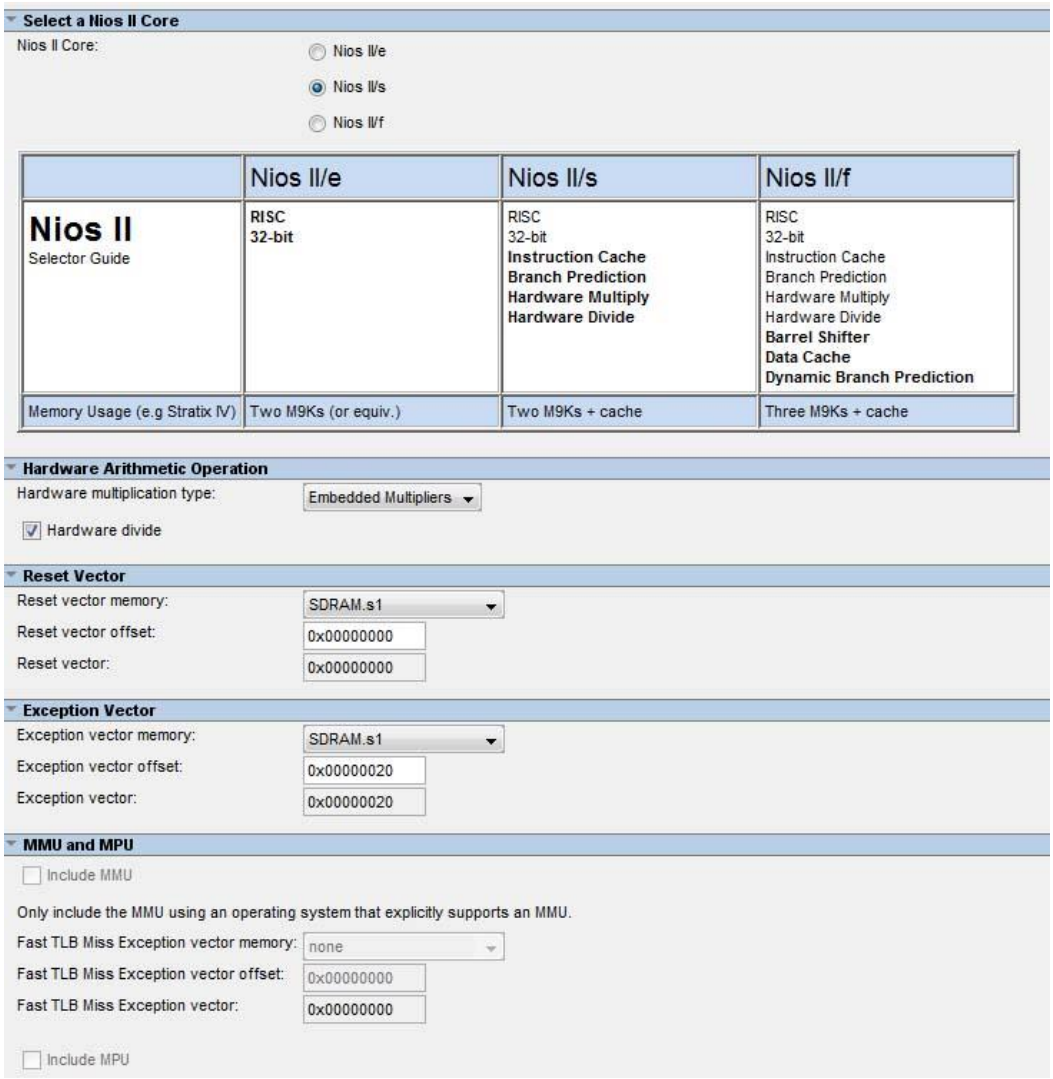


Figure 9 - CPU settings

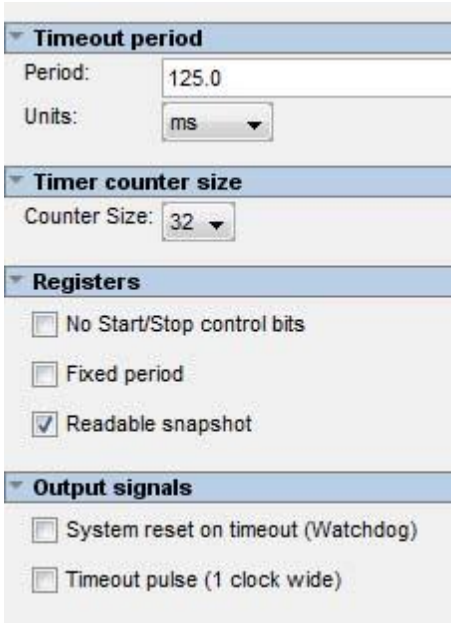


Figure 10 - Interval_Timer settings

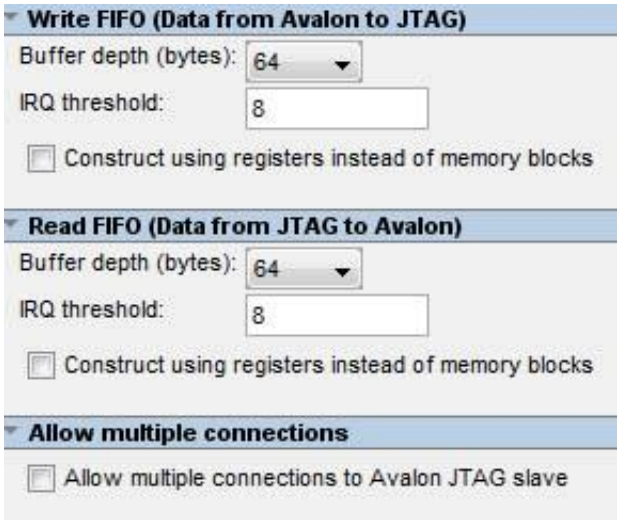


Figure 11 - JTAG_UART settings

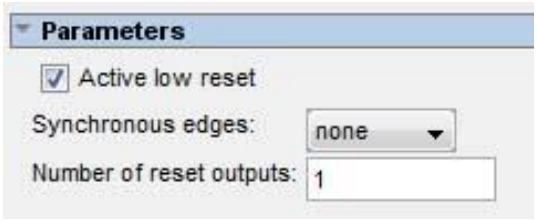


Figure 11 - merged_reset settings

Figure 13 - PS2_Port settings

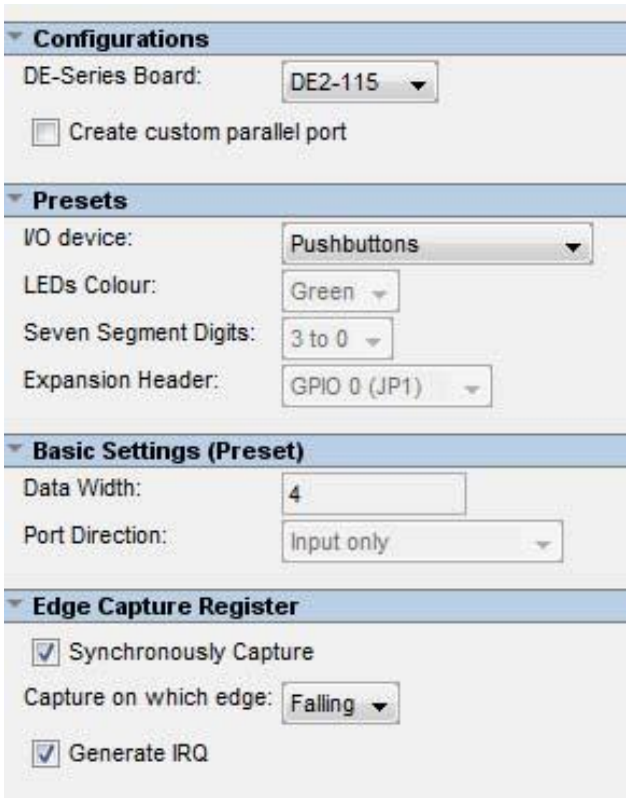


Figure 12 - Pushbuttons settings

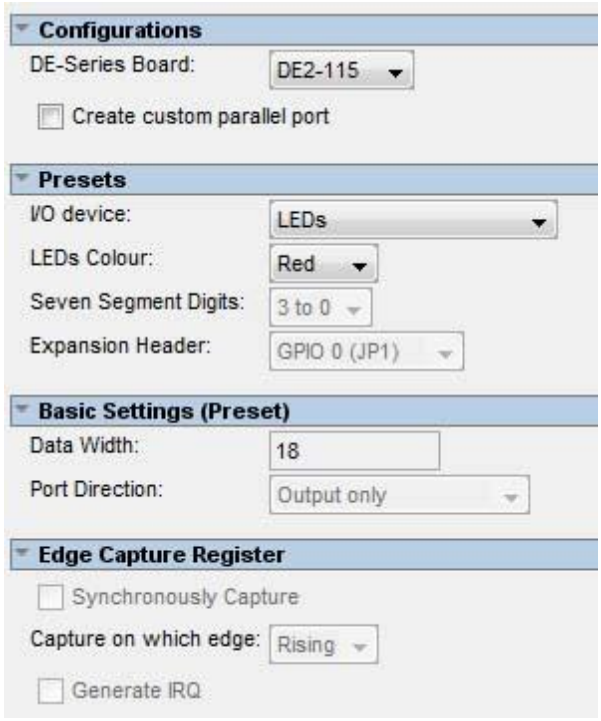


Figure 13 - Red_LEDs settings

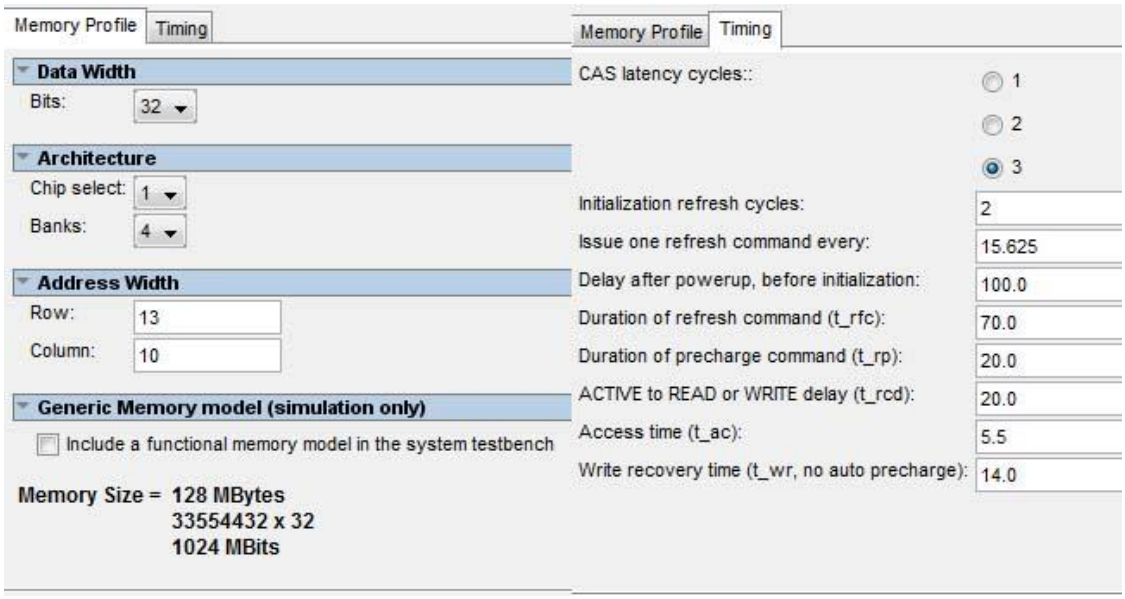


Figure 14 - SDRAM settings

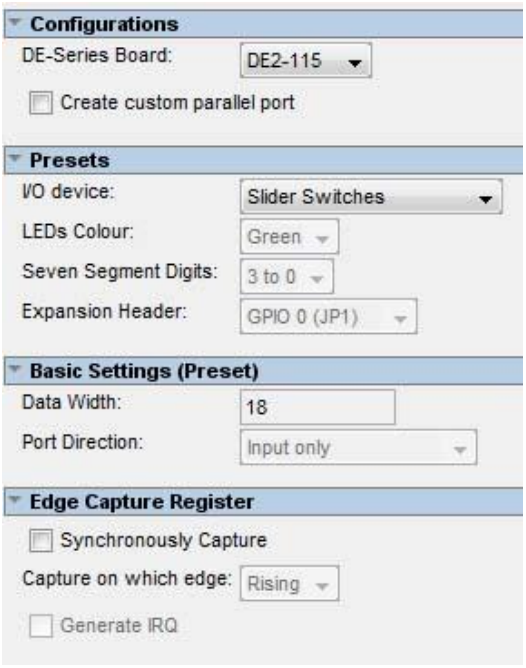


Figure 15 - Slider_Switches settings

Figure 16 - SRAM settings

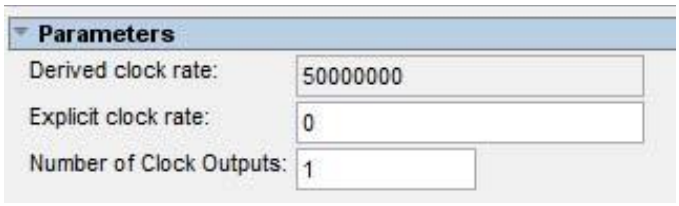


Figure 17 - sys_clk settings

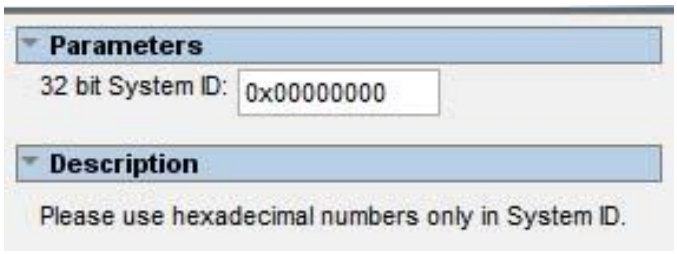


Figure 18 - sys_id settings

The next step in Qsys would be to generate the .sopcinfo file and grab the structural coding, but before proceeding there are a few steps to be taken to avoid errors. In the Media Computer, like stated above, the system base addresses are pre-determined and locked. If this was done by hand, you would have to assign the base addresses by going to System -> Assign Base Addresses in the top left-hand corner. It's also important to make sure that you have connected the SRAM to the CPU Exception and Reset vectors, as shown in Figure 9. Next, head over to the Generate tab, save your Qsys system, and hit generate. This will take about a minute, and when it is finished, you can go to the HDL Example tab and copy the Veriolog HDL structural coding, and paste it into the Verilog file that should be set as your top-level entity in Quartus. It is important to add the new files to the project by going to Project -> Add/Remove Files From Project, and click 'Add All….' Make sure that the .qsys file, .sdc file, and .v file are all added, and be sure to go to Assignments -> Import Assignments… and import the DE2-115 pin assignments, which can be downloaded from the Altera DE2-115 website (it should be a .qsf file). Now, back to the structural coding. The only necessary step here is, after copying the code from Qsys, you will see that there are a lot of lines that say to connect to a pin name. Remove

that, and enter in the name of the pin needed. Appendix B gives the Verilog top-level entity code that lists the proper pins. Now, all that's left is compiling the code and programming it to the board using the Programmer tool. Compiling will take a few minutes, but once it is done, a pop up will saying that it is complete. If it failed, it will list the errors in the Quartus II console. Compiling the system outputs a .sof file, which is used by the Programmer tool to download it to the board. The .sof file generated will be a time limited file, so after programming to the board there will be a pop up that says how much time is left (should be unlimited) and a cancel button. Do not hit the cancel button, instead just leave it open for the remainder of the project. If you need to change the structural coding or Qsys file at any time, you will need to recompile and redownload the program to the board. The next portion of the project requires the Nios II Software Build Tools for Eclipse, found under the Tools toolbar.

Developing the System with Eclipse

The Nios II Software Build Tools for Eclipse are used to download a C/C++ program to the development board, and is probably the more difficult part of the project. It poses more problems because this requires you to look into all of the registers of the components used in Qsys, and manipulating them either directly through memory addresses or through the library of functions provided by Quartus. Begin this step by clicking on New Project from Template, and select NIOS II Application and BSP From Template. It will ask for a .sopcinfo file, which is located in your project folder. Assuming you chose the right one, it will automatically find the CPU name. Enter in a project name and hit next. The file that will be edited is hello_world.c. For the audio synthesizer, during this phase there were a number of errors which were difficult to fix, often due to poor documentation. A troubling and frustrating problem was encountered during the first audio test, to see how the audio component worked. From reading through the Audio

Core documentation provided by Altera, it was understood that there were FIFO address, one for each channel, and it was to be used to read when data was coming in, and written when data was going out. Before new data was read and written, the FIFOs were to be cleared. The documentation provided a very light sample program, that would prove unsuccessful multiple times. On the first trial run, the Media Computer explained above was not used. Instead, there were a few missing components that were not stated anywhere that it was required. For example, not having the Audio and Video Configuration component on Qsys proved to be a big problem. When the code was first run with just the Audio cores, after multiple tests and trying to use switches to change how the program worked, the output was just streaming the input. After a long search through documentation and internet forums, I turned to another Altera example program. It was a lab to create a synthesizer, much like the project, but it started by facing the user with having the board record the input with a press of a push button, and play it back when another button was pressed. It gave a great explanation of how sound works and is synthesized, but nothing about on how to actually go about doing this.

Looking to my professor for assistance, he provided me with the code for that first part. Again after messing around with that code, the Audio core registers, and whatever else I could find, I decided to take a direct look at how the Wolfson WM8731 Audio Codec worked, by reading through the datasheet. I found some incredibly valuable information. By default, the codec was set to bypass the Mic and Line In ports straight through to the Line Out port. With that information, I was set out to find how to configure the codec, and I came across an answer relatively fast, which was to use the Audio and Video Config cores. I added this into my Qsys system, ran the code again, and although making some progress, it still failed to work. Once again, another frustrating day was spent digging through Google searches and more

documentation to figure out why it wasn't working. As a suggestion from the professor, I loaded up the DE2-115 Media Computer system, and ran it. With luck, it was working successfully, bringing me to the next step of understanding how it worked and getting it to play tones.

To make digital music, noises are produced by playing different waveforms at certain frequencies. The most common waveform is a sine wave, but square, sawtooth, or triangle waves are used to create different tones. For this project, a sine wave was used. In order to play a note, a stream of data across the wave at a certain frequency, broken up by time *t*, is loaded into the outgoing audio FIFO for each channel. The sine wave is generated through this formula:

$$v(t) = A \cdot \sin(2 \cdot \pi \cdot \frac{t}{T})$$

In this equation, *A* represents the amplitude, or volume, *t* is time, and *T* is the period. Each of the twelve notes across the eight octaves corresponds to a certain frequency. Only one octave was used for this project due to a limited number of keys, but it would be possible to generate tones for a complete grand piano with all 88 keys. The frequencies used to generate the notes are as follows:

C = 262 Hz
C# = 278 Hz
D = 292 Hz
D# = 310 Hz
E = 328 Hz
F = 348 Hz
F# = 370 Hz
G = 392 Hz
G# = 416 Hz
A = 440 Hz
A# = 466 Hz
B = 494 Hz

This produces notes in the fourth octave, which are a little bit high pitched but work well for this purpose. As it was mentioned above, to produce noise from the Line Out, the outgoing

FIFO needs to be fed a bit stream of values for output. The bit stream sent to the FIFOs in this case was an array of 24000 values, one for each note. The values were generated and each index of the array contains the value of the sine wave at time *t*. All of the tones were provided by the professor, and stored in a C header file named tones.h. In the main C program, the arrays would be called when a key is pressed on the keyboard, but to start, a series of switches were used. The program ran such that when a switch was turned on, the FIFO would be cleared, then loaded with the note array. This was a simple task, and the next step was getting the PS/2 keyboard to interact with the program.

The PS/2 keyboard was first set up on a small test program that read data in from the keyboard and sent it to the on-board LCD. This program used the Altera University Program C functions, although the tone program controlled the registers directly through memory addresses and pointers. This means that instead of being able to get an ASCII value from the keyboard, a decimal value would be output instead. This proved to be a little bit difficult. In the DE2-115 Media Computer manual, there was a bit of code that got the last 3 bytes from the keyboard. I ran that and had it print the bytes received to the console. There appeared to be a patter, where the first two bytes gave the same data, but I figured that using just the third byte would give the most recent key press. When trying to convert this value to a decimal value, none of the keys lined up and I was getting the wrong ASCII value for many of the keys. I decided to go back to using the decimal values, and when a certain key was pressed, it would use the decimal value and an if statement to play the corresponding note. The full C code is found in Appendix C, and looking at it will give a better idea of how the program uses the port addresses for each component, and offsetting the value to manipulate a certain register. The next series of figures (Figures 19 and 20) show the Audio and PS/2 registers, their offsets, and the data they hold.

Offset in bytes	Register Name	R/W	Bit Description									
			31...24	23...16	15...10	9	8	7...4	3	2	1	0
0	control	RW	*(1)*			WI	RI	*(1)*	CW	CR	WE	RE
4	fifospace	R	WS LC	WS RC	RA LC			RA RC				
8	leftdata	RW *(2)*	Left Data									
12	rightdata	RW *(2)*	Right Data									

Figure 19 - Audio register

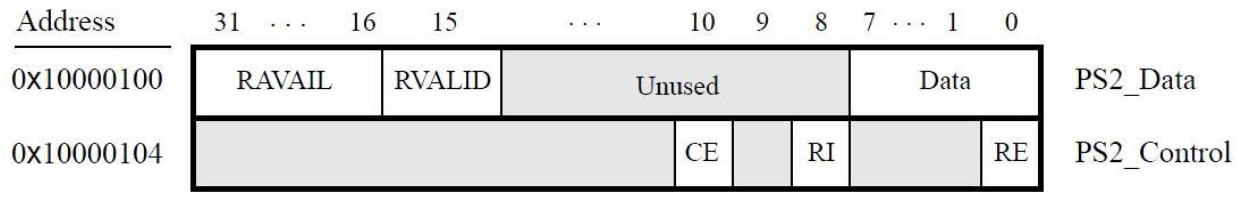


Figure 20 - PS/2 register

Errors Encountered and Their Solutions

Of course, projects very rarely go smoothly, and errors are bound to be ran into. Already mentioned in other sections were a few errors that had to be overcame, but they weren't the only ones. Most of the errors occurred while compiling the C code and downloading the project to the board. To avoid many of the issues, it is advised that when compiling the top-level Verilog program, to double check that all of the proper files are added and the pin assignment has been imported. Those were the biggest causes of errors when dealing with Quartus. Others were small attentions to detail, such as Qsys not being configured properly or a syntax error in the Verilog code. Using the DE2-115 Media Computer cleared up a few errors, and it allowed a better understanding of what components were required for the project and why, such as the SDRAM. The SDRAM is important because it gives the FIFOs somewhere to read and write memory to and from, respectively. Without it, the incoming FIFO space would not change because there would be nowhere to store the data. The bulk of the errors occurred in Eclipse, though.

There were a huge list of problems dealt with when compiling the code. A good first check is to take a few of these steps when loading up the project on a different computer or completely different session. First, make sure that when Eclipse asks for a workspace, use the one you had made the first time you ran Eclipse. This ensures that your project will properly be loaded. Next it is important to clean both of the project files by right clicking on the folder and clicking on Clean Project. Next, also be sure to generate a new BSP file by right clicking on the bsp folder, hovering over NIOS II, and clicking on Generate BSP…. It is now okay to attempt to build and run the project as Nios II Hardware. There's a chance that the Run Configuration prompt will open with an error about mismatching system IDs. If this is the case, make sure the Target Connection tab matches Figure 21. Note that in Figure 21, there are no cables connected, but the Connections section should show the USB-Blaster device if everything is running smoothly.

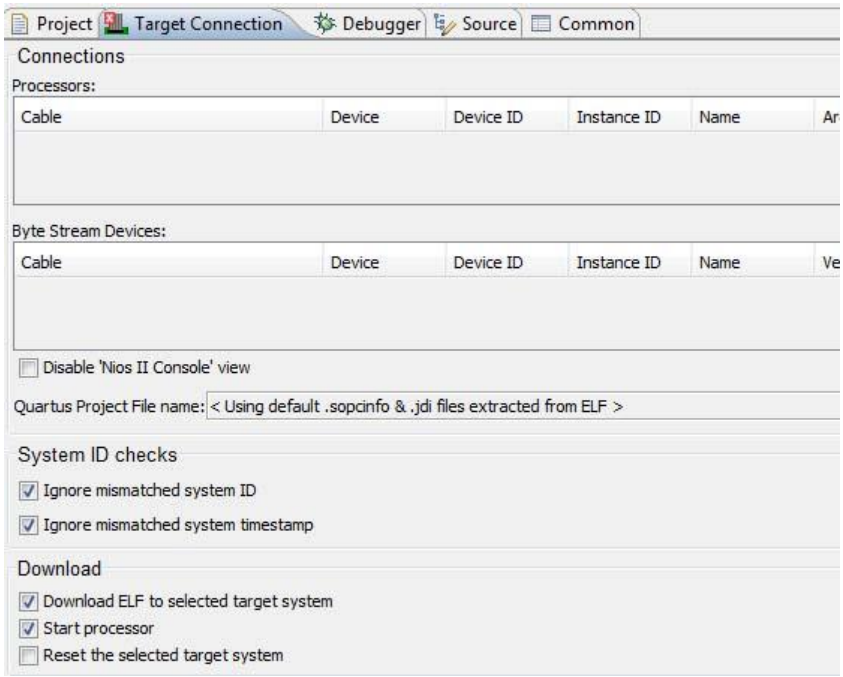


Figure 21 - Target Connection tab

In this project, a few things had to be changed with the BSP settings. At first, a problem was happening that seems like the system did not have enough memory for the program to run. The problem was that the program kept crashing when I used certain commands. In order to solve this, the BSP settings had to be changed to match that of Figure 22. If at first the screen appears grey, you may have to click on "Apply" first to enable the options.

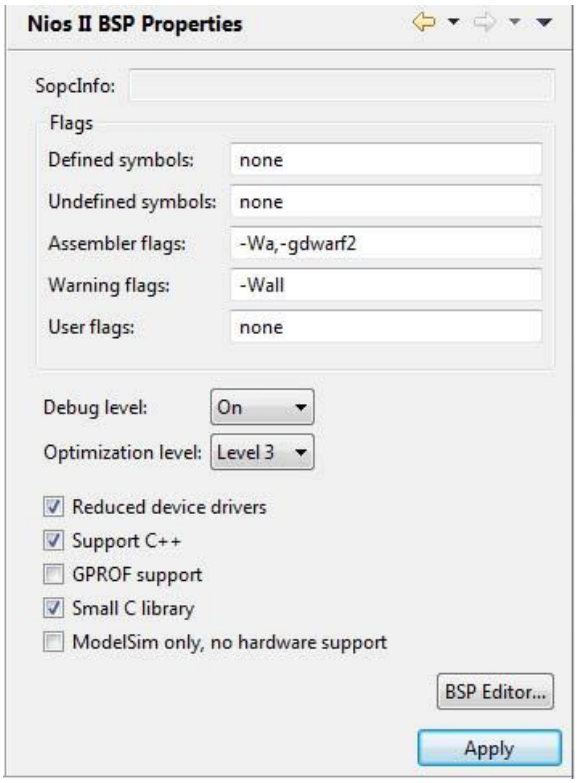


Figure 22 - Enabling 'Reduced device drivers' and 'Small C library' to avoid errors

After this error was solved, another one appeared. This one was caused by the SDRAM not being added, and there not being enough memory for the program to run completely. In the Eclipse console, this error was shown:

warning: Unable to reach (null) (at 0x002759e0) from the global pointer (at 0x00164288) because the offset (1120088) is out of the allowed range, -32678 to 32767.

At this point, I began using the full DE2-115 Media Computer and the error did not show up again. Next up came an error with the linker.x file in the bsp folder. I was getting the same

error as above, but only when I included the tones header file to generate the notes. After searching the Altera website for errors regarding the specific function that was causing the error, I found that the linker.x file had to be edited. In order to fix this linker error, from the Altera website [1]:

> To resolve this issue, open the linker.x linker script in the BSP and edit the following line:
>
> *.rwdata LOADADDR (.rodata) + SIZEOF (.rodata) : AT (LOADADDR (.rodata) + SIZEOF (.rodata)+ SIZEOF (.rwdata))*
>
> Remove the last + SIZEOF (.rwdata) directive, to correctly define the .rwdata section as follows:
>
> *.rwdata LOADADDR (.rodata) + SIZEOF (.rodata) : AT (LOADADDR (.rodata) + SIZEOF (.rodata))*

That was the last error that caused any major problems. The other ones were minor C syntax errors or typos in the program. Fishing through all of these errors was incredibly frustrating. If there had been better documented examples or tutorials in the internet, this may have been a little bit easier.

Results and Conclusions

After so much struggle, finally implementing an audio synthesizer onto the DE2-115 development board was incredibly satisfying. The hours spent chipping away at confusion and errors was well spent. I was very excited to hear a tone that wasn't just either noise or nothing when I first flipped the switch on the board, then later a key on the keyboard. Music and sound has always been an interest of mine, and this project really sparked it up again. It wasn't a surprise to me how the sounds were actually produced from different waveforms, but I didn't expect so much trouble actually implementing it.

There are many ways in which this project could be improved. With more time, there could have been plenty more features. One of them is an echo effect, which was actually asked in Altera's example lab. This would not be so difficult to implement. After a short delay, the tone would be repeated but multiplied by some decimal to decrease the amplitude of the wave. Some more features would be adding more tones by changing the octaves maybe depending on whether or not a switch is activated on the board, or adding in a way to record what is being played to be played back later on. An interesting idea would be using a touchscreen that is purchasable for this development board, and having it play notes when the screen's x axis is touched, and maybe change volume when the screen's y axis is touched. Given enough time and resources, a fully fledged audio synthesizer could definitely be implemented.

Bibliography

[1] Altera. (2012, Aug 12). *Why do I get a linker error with global pointers when trying to compile my Nios II code?* [Online]. Available: http://niosii.com/support/kdb/solutions/rd02132012_291.html

[2] Terasic Technologies, Inc. (2010). *DE2-115 User Manual*. [Online]. Available FTP: ftp.altera.com. Directory: /up/pub/Altera_Material/12.1/Boards/DE2-115/. File: DE2_115_User_Manual.pdf

[3] Altera Corporation. (2010). *Audio Core for DE2 Series Boards*. [Online]. Available FTP: ftp.altera.com. Directory: /up/pub/Altera_Material/12.1/University_Program_IP_Cores/Audio_Video/. File: Audio.pdf

[4] Altera Corporation. (2010). *Audio/Video Configuration Core for DE1/DE2 Series Boards*. [Online]. Available FTP: ftp.altera.com. Directory: /up/pub/Altera_Material/12.1/University_Program_IP_Cores/Audio_Video/. File: Audio_and_Video_Config.pdf

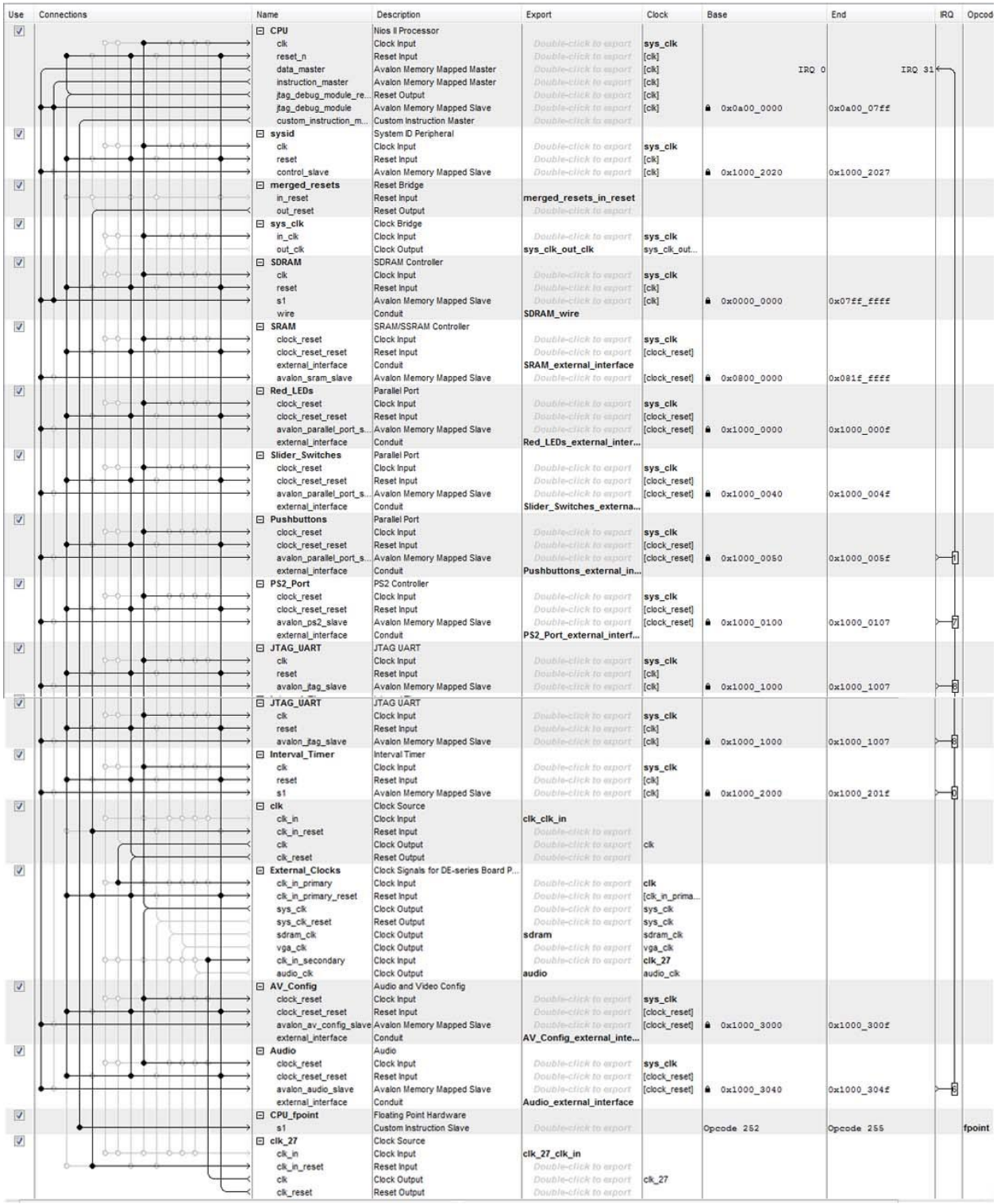
[5] Wolfson Microelectronics. (2010). *WM8731/WM8731L*. [Online]. Available FTP: ftp.altera.com. Directory: /up/pub/datasheets/DE2/Audio CODEC/. File: WM8731_WM8731L.pdf

[6] Altera Corporation. (2010). *Laboratory Excersize 8*. [Online]. Available FTP: ftp.altera.com. Directory: /up/pub/Altera_Material/12.1/Laboratory_Exercises/Computer_Organization/DE2-115/. File: lab8.pdf

[7] Altera Corporation. (2010). *Media Computer System for the Altera DE2-115 Board*. [Online]. Available FTP: ftp.altera.com. Directory: /up/pub/Altera_Material/12.1/Examples/DE2-115/NiosII_Computer_Systems/. File: DE2-115_Media_Computer.pdf

Appendices

Appendix A: Qsys Connections



Appendix B: Top-Level Verilog Code

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module SYNTH_PROJ(

      ///////////// CLOCK /////////
      CLOCK_50,
      CLOCK2_50,
      CLOCK3_50,
      TD_CLK27,

      ///////////// LED /////////
      LEDR,

      ///////////// KEY /////////
      KEY,

      ///////////// SW /////////
      SW,

      ///////////// PS2 for Keyboard and Mouse /////////
      PS2_CLK,
      PS2_CLK2,
      PS2_DAT,
      PS2_DAT2,

      ///////////// Audio /////////
      AUD_ADCDAT,
      AUD_ADCLRCK,
      AUD_BCLK,
      AUD_DACDAT,
      AUD_DACLRCK,
      AUD_XCK,

      ///////////// I2C for Audio  /////////
      I2C_SCLK,
      I2C_SDAT,

      ///////////// SDRAM /////////
      DRAM_ADDR,
      DRAM_BA,
      DRAM_CAS_N,
      DRAM_CKE,
      DRAM_CLK,
      DRAM_CS_N,
      DRAM_DQ,
      DRAM_DQM,
      DRAM_RAS_N,
      DRAM_WE_N,

      ///////////// SRAM /////////
      SRAM_ADDR,
      SRAM_CE_N,
```

```
      SRAM_DQ,
      SRAM_LB_N,
      SRAM_OE_N,
      SRAM_UB_N,
      SRAM_WE_N
);

//=======================================================
//  PARAMETER declarations
//=======================================================


//=======================================================
//  PORT declarations
//=======================================================

////////////// CLOCK //////////
input                               CLOCK_50;
input                               CLOCK2_50;
input                               CLOCK3_50;
input                                          TD_CLK27;

////////////// LED //////////
output                [17:0]        LEDR;

////////////// KEY //////////
input                  [3:0]        KEY;

////////////// SW //////////
input                 [17:0]        SW;

////////////// PS2 for Keyboard and Mouse //////////
inout                               PS2_CLK;
inout                               PS2_CLK2;
inout                               PS2_DAT;
inout                               PS2_DAT2;

////////////// Audio //////////
input                               AUD_ADCDAT;
inout                               AUD_ADCLRCK;
inout                               AUD_BCLK;
output                              AUD_DACDAT;
inout                               AUD_DACLRCK;
output                              AUD_XCK;

////////////// I2C for Audio  //////////
output                              I2C_SCLK;
inout                               I2C_SDAT;

////////////// SDRAM //////////
output                [12:0]        DRAM_ADDR;
output                 [1:0]        DRAM_BA;
output                              DRAM_CAS_N;
output                              DRAM_CKE;
output                              DRAM_CLK;
output                              DRAM_CS_N;
inout                 [31:0]        DRAM_DQ;
```

```
output                  [3:0]        DRAM_DQM;
output                               DRAM_RAS_N;
output                               DRAM_WE_N;

////////////// SRAM //////////
output                 [19:0]        SRAM_ADDR;
output                               SRAM_CE_N;
inout                  [15:0]        SRAM_DQ;
output                               SRAM_LB_N;
output                               SRAM_OE_N;
output                               SRAM_UB_N;
output                               SRAM_WE_N;


//=======================================================
//  REG/WIRE declarations
//=======================================================




//=======================================================
//  Structural coding
//=======================================================
   SYNTH_CORE_mc u0 (
        .SRAM_DQ_to_and_from_the_SRAM           (SRAM_DQ),           //
SRAM_external_interface.DQ
        .SRAM_ADDR_from_the_SRAM                (SRAM_ADDR),
//                                      .ADDR
        .SRAM_LB_N_from_the_SRAM                (SRAM_LB_N),
//                                      .LB_N
        .SRAM_UB_N_from_the_SRAM                (SRAM_UB_N),
//                                      .UB_N
        .SRAM_CE_N_from_the_SRAM                (SRAM_CE_N),
//                                      .CE_N
        .SRAM_OE_N_from_the_SRAM                (SRAM_OE_N),
//                                      .OE_N
        .SRAM_WE_N_from_the_SRAM                (SRAM_WE_N),
//                                      .WE_N

        .sys_clk                                (),
//                      sys_clk_out_clk.clk

        .AUD_ADCDAT_to_the_Audio                (AUD_ADCDAT),
//           Audio_external_interface.ADCDAT
        .AUD_ADCLRCK_to_the_Audio               (AUD_ADCLRCK),
//                                      .ADCLRCK
        .AUD_BCLK_to_the_Audio                  (AUD_BCLK),
//                                      .BCLK
        .AUD_DACDAT_from_the_Audio              (AUD_DACDAT),              //
.DACDAT
        .AUD_DACLRCK_to_the_Audio               (AUD_DACLRCK),
//                                      .DACLRCK

        .LEDR_from_the_Red_LEDs                 (LEDR),                  //
Red_LEDs_external_interface.export
```

```
        .reset_n                                (KEY[0]),
//              merged_resets_in_reset.reset_n

        .zs_addr_from_the_SDRAM                 (DRAM_ADDR),
//                          SDRAM_wire.addr
        .zs_ba_from_the_SDRAM                   (DRAM_BA),
//                                     .ba
        .zs_cas_n_from_the_SDRAM                (DRAM_CAS_N),
//                                     .cas_n
        .zs_cke_from_the_SDRAM                  (DRAM_CKE),
//                                     .cke
        .zs_cs_n_from_the_SDRAM                 (DRAM_CS_N),
//                                     .cs_n
        .zs_dq_to_and_from_the_SDRAM            (DRAM_DQ),            //
.dq
        .zs_dqm_from_the_SDRAM                  (DRAM_DQM),
//                                     .dqm
        .zs_ras_n_from_the_SDRAM                (DRAM_RAS_N),
//                                     .ras_n
        .zs_we_n_from_the_SDRAM                 (DRAM_WE_N),
//                                     .we_n

        .SW_to_the_Slider_Switches              (SW),              //
Slider_Switches_external_interface.export

        .I2C_SDAT_to_and_from_the_AV_Config     (I2C_SDAT),     //
AV_Config_external_interface.SDAT
        .I2C_SCLK_from_the_AV_Config            (I2C_SCLK),            //
.SCLK

        .PS2_CLK_to_and_from_the_PS2_Port       (PS2_CLK),       //
PS2_Port_external_interface.CLK
        .PS2_DAT_to_and_from_the_PS2_Port       (PS2_DAT),       //
.DAT

        .KEY_to_the_Pushbuttons                 ({KEY[3:1], 1'b1}),
//     Pushbuttons_external_interface.export

        .clk                                    (CLOCK_50),
//                         clk_clk_in.clk
        .clk_27                                 (TD_CLK27),
//                       clk_27_clk_in.clk
        .audio_clk                              (AUD_XCK),
//                                audio.clk
        .sdram_clk                              (DRAM_CLK)
//                                sdram.clk
    );


endmodule
```

Appendix C: C Code

```
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>
#include "system.h"
#include "altera_up_avalon_audio.h"
#include "altera_up_avalon_audio_regs.h"
#include "altera_up_avalon_audio_and_video_config.h"
#include "altera_up_avalon_audio_and_video_config_regs.h"
//#include "altera_avalon_pio_regs.h"
#include "altera_up_avalon_ps2.h"
#include "altera_up_ps2_keyboard.h"
//#include "altera_up_ps2_mouse.h"
#include "altera_up_avalon_ps2_regs.h"
#include "sys/alt_stdio.h"
#include "io.h"
#include "tones.h"

/* globals */
#define BUF_SIZE 150000                // about 3 seconds of buffer (@ 48K
samples/sec)

void play_note(int keyb_val);

int main(void)
{
            /* Declare volatile pointers to I/O registers (volatile means that IO
load
         and store instructions will be used to access these pointer locations,
         instead of regular memory loads and stores) */
      volatile int * audio_ptr = (int *) 0x10003040;                    // audio port
address
      volatile int * red_LED_ptr = (int *) 0x10000010;           // red LED address
      volatile int * sw_ptr = (int *) 0x10000040;                       // slider
switches address
      volatile int * PS2_ptr = (int *) 0x10000100;               // PS/2 port addr
```

```
	/* used for audio record/playback */
	int fifospace;
	int PS2_data, RVALID;
	char byte1 = 0, byte2 = 0, byte3 = 0;
	//int leftdata, rightdata;
	//int delay_2_index = -1, delay_1_index = -1;
	int buffer_index = 0;
	int left_buffer[BUF_SIZE];
	int right_buffer[BUF_SIZE];
	int play=0, record=0;
	int i;
	//int tone_freq[24000];

	*(red_LED_ptr) = 0x0;
	/* read and echo audio data */
	while(1)
	{
		PS2_data = *(PS2_ptr);
		RVALID = PS2_data & 0x8000;

		if(RVALID)   //Get last 3 bytes from the keyboard
		{
			byte1 = byte2;
			byte2 = byte3;
			byte3 = PS2_data&0xFF;
			//printf("byte1 = %d\n", byte1);
			//printf("byte2 = %d\n", byte2);
			//printf("byte3 = %d\n", byte3);
		}

		play_note(byte3);
	}
}

void play_note(int keyb_val){
	volatile int * audio_ptr = (int *) 0x10003040;
	volatile int * red_LED_ptr = (int *) 0x10000000;
	int i;

	if(keyb_val == 51) {
		//printf("Sw_ptr = 1\n");
		*(audio_ptr) = 0x8;
		*(audio_ptr) = 0x0;
		for(i = 0;i<=24000;i++){
			*(audio_ptr + 2) = tone_a[i];
			*(audio_ptr + 3) = tone_a[i];
		}
		*(red_LED_ptr) = 0x200;
	}
	else if(keyb_val == 60) {
		//printf("Sw_ptr = 2\n");
		*(audio_ptr) = 0x8;
		*(audio_ptr) = 0x0;
		for(i = 0;i<=24000;i++){
```

```
                *(audio_ptr + 2) = tone_a_sharp[i];
                *(audio_ptr + 3) = tone_a_sharp[i];
        }
        *(red_LED_ptr) = 0x400;
}
else if(keyb_val == 59) {
        //printf("Sw_ptr = 3\n");
        *(audio_ptr) = 0x8;
        *(audio_ptr) = 0x0;
        for(i = 0;i<=24000;i++){
                *(audio_ptr + 2) = tone_b[i];
                *(audio_ptr + 3) = tone_b[i];
        }
        *(red_LED_ptr) = 0x800;
}
else if(keyb_val == 28) {
        //printf("Sw_ptr = 4\n");
        *(audio_ptr) = 0x8;
        *(audio_ptr) = 0x0;
        for(i = 0;i<=24000;i++){
                *(audio_ptr + 2) = tone_c[i];
                *(audio_ptr + 3) = tone_c[i];
        }
        *(red_LED_ptr) = 0x1;
}
else if(keyb_val == 29) {
        //printf("Sw_ptr = 5\n");
        *(audio_ptr) = 0x8;
        *(audio_ptr) = 0x0;
        for(i = 0;i<=24000;i++){
                *(audio_ptr + 2) = tone_c_sharp[i];
                *(audio_ptr + 3) = tone_c_sharp[i];
        }
        *(red_LED_ptr) = 0x2;
}
else if(keyb_val == 27) {
        //printf("Sw_ptr = 6\n");
        *(audio_ptr) = 0x8;
        *(audio_ptr) = 0x0;
        for(i = 0;i<=24000;i++){
                *(audio_ptr + 2) = tone_d[i];
                *(audio_ptr + 3) = tone_d[i];
        }
        *(red_LED_ptr) = 0x4;
}
else if(keyb_val == 36) {
        //printf("Sw_ptr = 7\n");
        *(audio_ptr) = 0x8;
        *(audio_ptr) = 0x0;
        for(i = 0;i<=24000;i++){
                *(audio_ptr + 2) = tone_d_sharp[i];
                *(audio_ptr + 3) = tone_d_sharp[i];
        }
        *(red_LED_ptr) = 0x8;
}
```

```
	else if(keyb_val == 35) {
		//printf("Sw_ptr = 8\n");
		*(audio_ptr) = 0x8;
		*(audio_ptr) = 0x0;
		for(i = 0;i<=24000;i++){
			*(audio_ptr + 2) = tone_e[i];
			*(audio_ptr + 3) = tone_e[i];
		}
		*(red_LED_ptr) = 0x10;
	}
	else if(keyb_val == 43) {
		//printf("Sw_ptr = 9\n");
		*(audio_ptr) = 0x8;
		*(audio_ptr) = 0x0;
		for(i = 0;i<=24000;i++){
			*(audio_ptr + 2) = tone_f[i];
			*(audio_ptr + 3) = tone_f[i];
		}
		*(red_LED_ptr) = 0x20;
	}
	else if(keyb_val == 44) {
		//printf("Sw_ptr = 10\n");
		*(audio_ptr) = 0x8;
		*(audio_ptr) = 0x0;
		for(i = 0;i<=24000;i++){
			*(audio_ptr + 2) = tone_f_sharp[i];
			*(audio_ptr + 3) = tone_f_sharp[i];
		}
		*(red_LED_ptr) = 0x40;
	}
	else if(keyb_val == 52) {
		//printf("Sw_ptr = 11\n");
		*(audio_ptr) = 0x8;
		*(audio_ptr) = 0x0;
		for(i = 0;i<=24000;i++){
			*(audio_ptr + 2) = tone_g[i];
			*(audio_ptr + 3) = tone_g[i];
		}
		*(red_LED_ptr) = 0x80;
	}
	else if(keyb_val == 53) {
		//printf("Sw_ptr = 12\n");
		*(audio_ptr) = 0x8;
		*(audio_ptr) = 0x0;
		for(i = 0;i<=24000;i++){
			*(audio_ptr + 2) = tone_g_sharp[i];
			*(audio_ptr + 3) = tone_g_sharp[i];
		}
		*(red_LED_ptr) = 0x100;
	}
	else
		*(red_LED_ptr) = 0x0;
}
```