Digital Graphic Equalizer Implemented Using an FPGA

by

Anthony Giardina

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2012

Table of Contents

List of Figures, Tables, and Equations

Abstract:

A graphic equalizer is a device that adjusts the tonal quality of an audio signal [1]. When sound is converted from a digital format to analog sound waves, there are amplification and transducing steps in-between the two formats. Common devices to perform these tasks are speakers, amplifiers, DACs, etc. Many of these devices exhibit a non-uniform frequency response over the range of human hearing. Thus, it is possible that certain frequency ranges of the audio signal will be amplified and others will be attenuated. To counteract this, an audio equalizer can be used to boost and attenuate certain frequency ranges within the signal, to counteract the undesirable response of the playback equipment. The equalizer can also be used to attenuate or amplify frequency ranges in the signal that the user finds desirable, such as boosting the bass or attenuating ranges common to certain instruments. In this way, a graphic equalizer provides the user with a highly customizable and intuitive audio experience.

Chapter 1: introduction

This project seeks to implement the functionality of a graphic equalizer on a physical hardware platform, for the processing of real time audio signals, using digital signal processing techniques. For this project, there are four main sections of design: signal processing hardware, signal processing firmware, peripheral hardware, and peripheral interfacing firmware. The end goal for the project is to implement a hardware system that is capable of performing an audio equalizing function on real-time signals. It should be able to do this fast enough that the user cannot hear the delay caused by the signal processing.

The traditional approach to creating consumer-grade audio equalizers is to use a bank of analog FIR filters [2]. Typically, the low and high frequencies are handled by 1st order high and low-pass "shelving filters." The other bands are each modified by individual band-pass filters. These filters are typically of the second order type. This requires precision analog components, and component matching. The approach used in this project is to implement these filters digitally. A digital implementation of these filters will create a system that requires less tuning, and is less likely to have its performance change over time. This methodology also has the possibility to reduce cost and system complexity, as the implementation of digital systems becomes cheaper.

Chapter 2: Requirements and Specifications

Most of the requirements for the project were chosen based on existing specifications for entry-level graphic equalizer systems that are currently being manufactured. Table 1 shows the initial requirements and specifications for this project.

Table 1 Graphic Equalizer Requirements and Specifications

Marketing Requirements	Engineering Specifications	Justification
1	The equalizer should have a minimum of nine frequency bands that can be adjusted.	Typically, graphic equalizers have at least 9 frequency bands that can be adjusted. Many of the higher scale models have more bands. Without an adequate number of bands, the user many not be able to achieve the frequency response that they desire.
1	The attenuation/amplification of each frequency band should have a minimum range of +/-12dB.	+/- 12dB allows for an amplitude amplification of 4x, or an attenuation of 0.25x. This allows reasonable manipulation of the signal, without causing extreme rescaling of the signal.
2	As a user input, the system should have one physical adjustment knob or slider to adjust the amplification/attenuation for each individual frequency band.	Knobs and sliders are quick to operate, and they are intuitive for users.
3	Each band should have a graphic indicator of the peak amplitude in that frequency band, taken as a time average.	Physical displays cannot update fast enough to display real-time information, and the calculation of real-time information would take significant processing power. Thus, a less frequently updated, time averaged signal is appropriate.
4	For audio input and output connections, standard stereo (three conductor) 3.5mm jacks are to be used.	These connections are available on most consumer audio equipment, and will make interfacing easy.
4	The system should use line level signals for both the audio input and audio output.	This will make interfacing with consumer devices easy, as most devices output line level signals, and most amplifiers can accept line level signals as an input.
1	The system should have a time delay of 45 microseconds or less	This is equivalent to 2 sampling periods of the audio signal, and should be sufficient to for real-time audio processing.

Marketing Requirements

1. The system should implement a graphic equalizer functionality for audio signals.
2. The system should have easy to manipulate user controls, to adjust the desired frequency response.
3. The system should have a graphic output that displays the peak amplitude of each frequency band.
4. The system should interface with other standard audio equipment.

Chapter 3: Functional Decomposition & Description

Theory of Operation:

This project implements the functionality of a graphic equalizer on a DSP-optimized FPGA. By nature, FPGAs are able to handle many tasks simultaneously. Thus, in this project the FPGA performs all of the following tasks in parallel:

- Send and receive PCM audio samples to and from the LM4550 audio codec. The codec also must be configured properly at system startup.
- Interface with the ADC to sample the positions of 9 slider switches, which represent the desired frequency response of the system.
- Perform audio filtering, using DSP techniques, to adjust the tonal quality of the audio input signal. This is accomplished by using nine FIR filters in parallel, one for each frequency band, to adjust the response of each frequency band. The outputs of these nine filters are then summed up to generate the output audio signal.
- Display an approximation of the spectrum of the output audio signal, by detecting the peak amplitude in each frequency band, and displaying these values on the LEDs by interfacing with the LED drivers.

A detailed description of each of these tasks is provided later in the report.

Top Level Block Diagram:

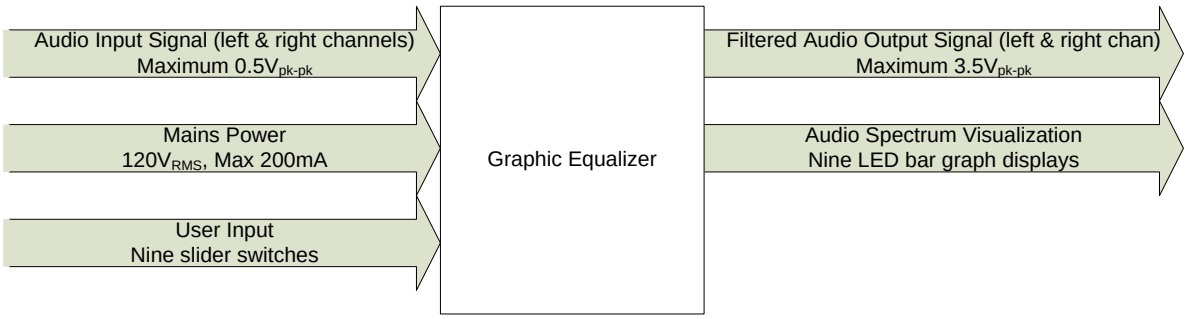

Figure 1: Top Level Block Diagram

At the most abstracted level, the Graphic Equalizer requires an audio signal as an input, mains power, and user input from the slider switches, as shown in Figure 1. The audio input signal is the signal to be filtered by the graphic equalizer. The mains power provides power to the entire system, and the user input from the slider switches determines the frequency response of the graphic equalizer. The outputs are the filtered audio, which is potentially larger in magnitude than the input audio, and the LED bar graphs, which represent the spectrum of the filtered audio signal.

System Level Block Diagram:

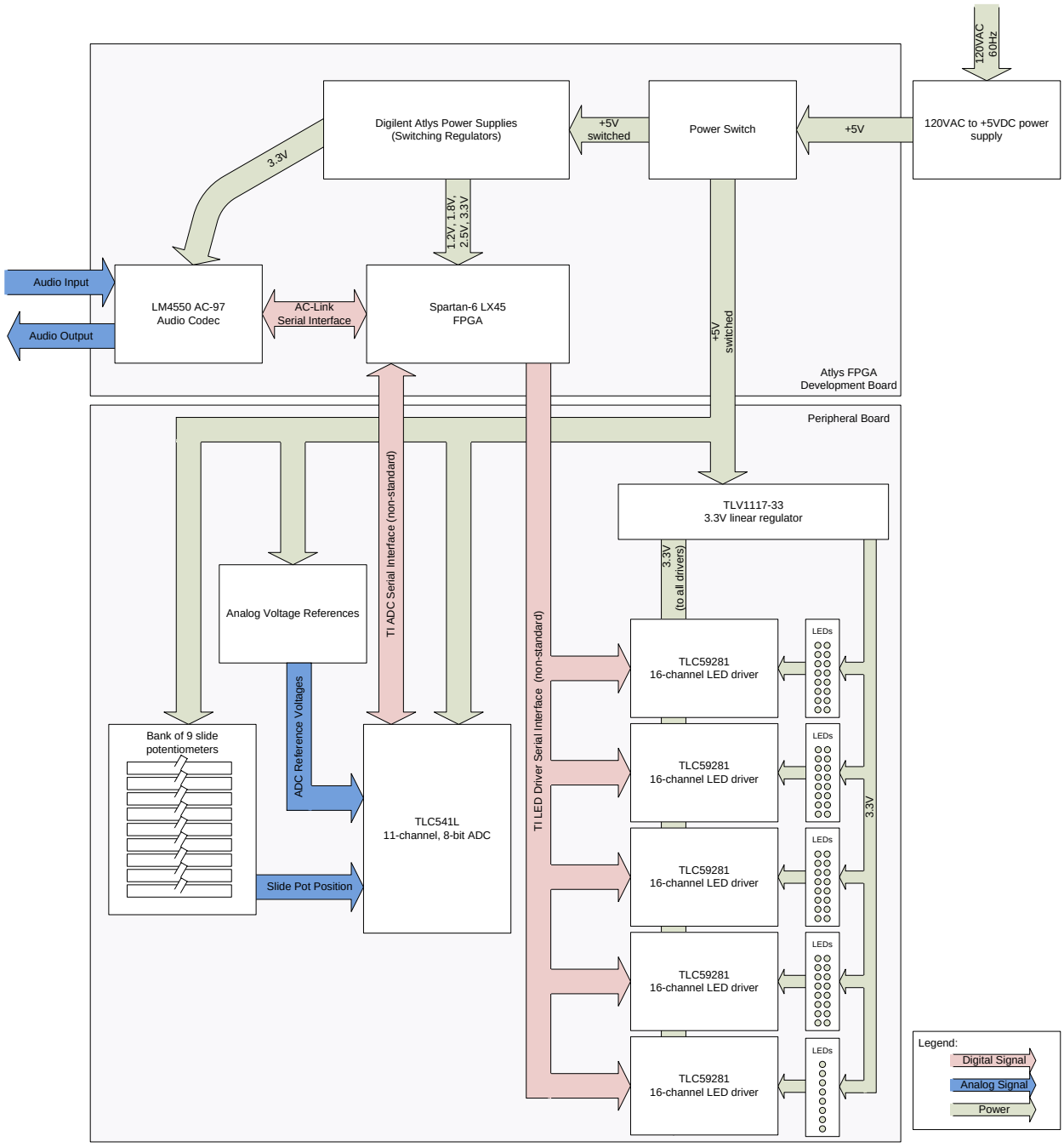

Figure 2: System Level Block Diagram

The system level block diagram in Figure 2 shows how all of the major components in the system are connected to each other. All of these components (and others) are identified in Figure 3, Figure 4, Figure 5, Figure 6 and are briefly described in Table 2. A more in-depth description of these components and associated signals is provided later in the report.

Component Identification:

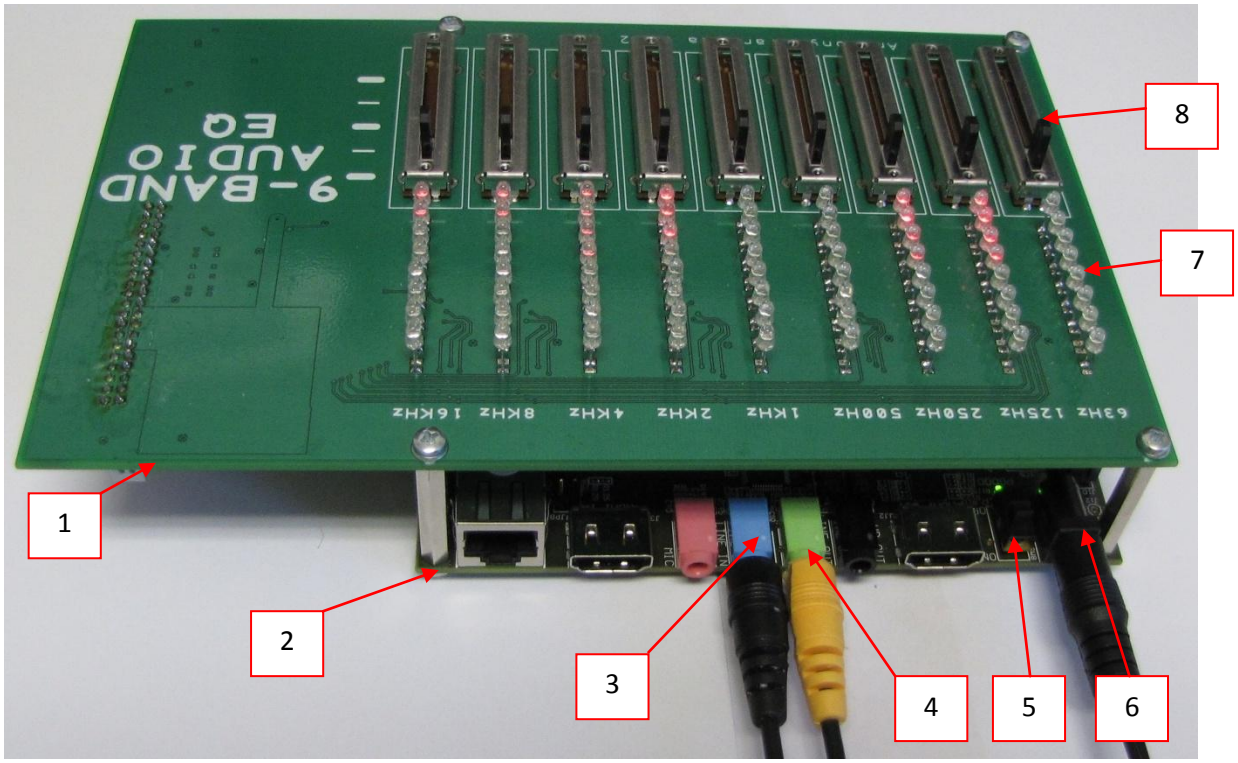

Figure 3: Assembled view of graphic equalizer for component identification

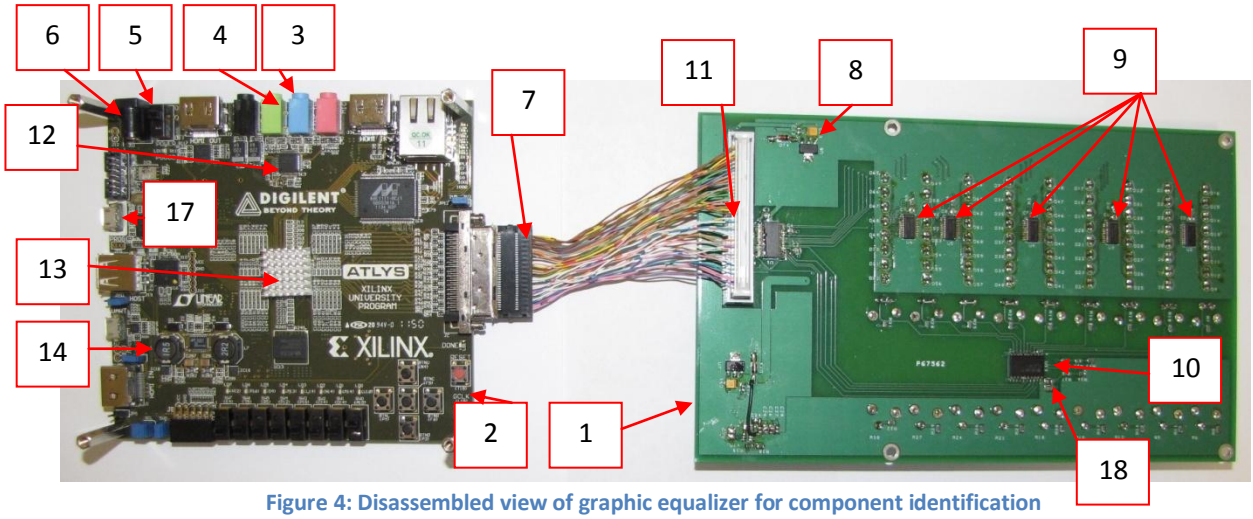

Figure 4: Disassembled view of graphic equalizer for component identification

Figure 5: AC-DC Power Supply

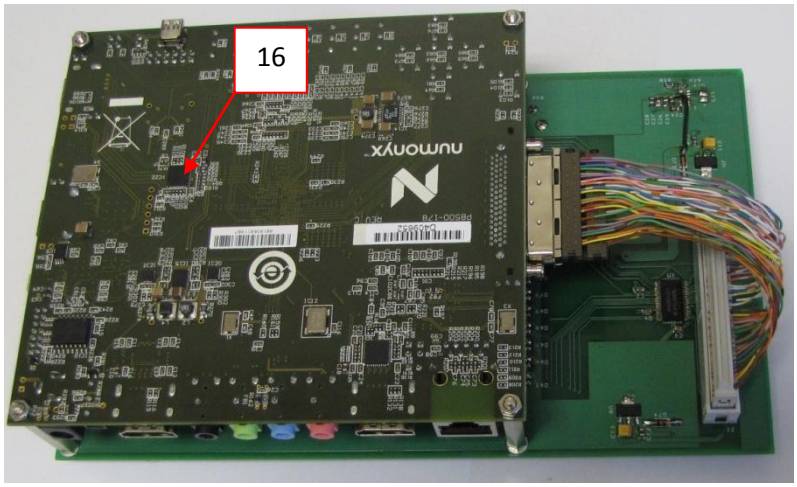

Figure 6: Bottom View of assembled graphic equalizer for component identification

Table 2: Description of components

Name	***Ref***	***Description***
Peripheral Board	1	The peripheral board is a custom designed PCB, which supports the human interface peripherals. It contains the 9 LED displays (with 8 LEDs each,) 9 slide potentiometers, and all of the circuitry necessary to interface these devices with the FPGA. It has four mounting holes which match the mounting holes on the Atlys FPGA Development Board, allowing it to be mounted on top.
Atlys FPGA Development board	2	The Atlys FPGA development board provides a platform for the Spartan-6 FPGA, Flash ROM memory, LM4550 Audio Codec, power supplies, and all other supporting circuitry.
Line-In Port	3	The line-in port is a line level audio input port. Voltages should be kept below $500mV_{Pk-Pk}$ to prevent waveform clipping on the output. The line-in port is connected to the LM4550 audio codec, via a DC-blocking filter.
Line-Out Port	4	The line-out port is a line level audio output port. Voltages can be expected as high as $3V_{pk-pk}$. The output level is limited by the analog supply of the LM4550. The line-out port is connected to the LM4550 by a DC-blocking filter.
Power Switch	5	The power switch controls all power to the board. The switch must be in the "on" position for system operation, and "off " position while the system is not operational. These positions are indicated on the Atlys FPGA development board silkscreen.
Power Socket	6	The power socket provides a location for power input to the entire system. +5VDC is expected at this port, and is provided by the AC-DC power adapter.
Interface Cable	7	The interface cable is a custom cable which provides power and signal interfaces between the Atlys FPGA development board and the peripheral board. One end of the cable is a standard 40-pin (20x2) 2.54mm pitch connector, which connects to the interface board, and the other end is a 68-pin VHDC (SCSI-3) connector.
TLV1117-33	8	The TLV1117-33 is used to provide +3.3VDC to peripheral board, from the input voltage of +5VDC. This supply is used to power the LEDs and associated driving circuitry.

TLC59281	9	The TLC59281 is a 16-channel LED driving IC. It is essentially a serially programmable current sink. 5 of these chips are arranged on one serial bus, so that all 72 of the LEDs on the board can be programmed individually (on or off) by the FPGA.
TLC541L	10	The TLC541L is an 11-channel 8-bit SAR ADC. Channels 0-8 are used for sampling the positions of the 9 slide potentiometers. It is interfaced via a serial bus to the FPGA.
SN74LVCH16244A	11	The SN74LVCH16244A is a 16-bit non-inverting line driver. It is used to interface all digital signals between the FPGA and the peripheral board. It is a 3.3V device, but has 5V tolerant inputs. It is mainly used to interface the TLC541L (which is a 5V part) to the FPGA (which is a 3.3V part) but also provides the added assurance of boosting signal strength across long traces with multiple loads.
LM4550	12	The LM4550 is an AC-97 audio codec. It consists of an 18-bit ADC/DAC pair, which provides the analog input/output for the system. It is interfaced to the FPGA via the Intel standard AC-Link interface.
Spartan-6 LX45	13	The Spartan-6 LX45 CSG324C is the FPGA used to control all of the processing and interfacing on the system.
Atlys Power Supplies	14	The Atlys power supplies reside on the Atlys FPGA development board. They provide regulated power for all of the devices on the Atlys FPGA development board.
AC-DC Power Supply	15	The AC-DC power supply converts $120V_{RMS}$, 60Hz mains power to +5VDC power, which is supplied to the Atlys FPGA development board.
Flash ROM Memory	16	The Flash ROM memory, N25Q12, is used to store the configuration file for the FPGA, which is loaded on the FPGA when the system is powered on. By default, the system will boot off of the flash ROM memory, which is programmable by USB.
USB Programming Port	17	The USB programming port is used to configure the FPGA or flash ROM memory form a PC, using the Digilent Adept application. It is a micro-USB type port.
ADC Voltage References	18	The ADC voltage references are resistor divider circuits, which provide high and low reference voltages for the TLC541L ADC. These references represent the expected highest and lowest voltages from the slide potentiometers.

Chapter 4: Hardware Design

This chapter describes the hardware that was specified, designed, and implemented for this project. This includes the peripheral board and everything on it, the interface cable, the Atlys FPGA development board, and some FPGA specifics. Firmware implementations, timing diagrams, and other firmware related topics are covered in Chapter 5: Firmware Design.

Interface Cable

The interface cable is made from a LVD Fast 40 SCSI cable, manufactured by Madison Cable Corp, type 20276, with 30AWG wires. One end of the cable was cut off, and the jacketing removed, to allow the connection of a standard 2.54mm pitch female header connection manufactured by FCI, part number 71600-040LF, "quickie" series, which mates to the Assmann AWHW40G-0202-T-R pin header on the peripheral board. The connector is shown in Figure 7, and it's pin out is shown in Table 3. The cable was fully tested for continuity and short circuits using an ohmmeter.

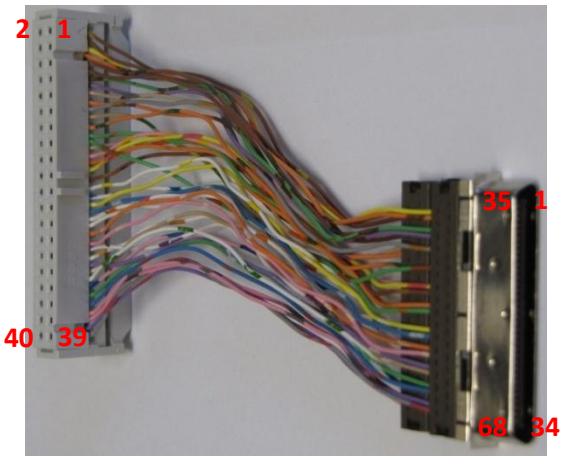

Figure 7: Interface connector, with pin numbers shown

Table 3: Interface Cable Pin out

68-pin conn	Pri color	Sec color	FPGA Pin	40-pin conn	Signal
1	brn	blu			
2	brn	yel	GND	1	GND
3	brn	orn	U15	2	LED_SIN
4	brn	pnk			
5	brn	grn	GND	5	GND
6	brn	ppl	M11	6	LED_BLANK
7	brn	gry			
8	tan	gry	GND	9	GND
9	orn	ppl	T12	10	LED_LAT2
10	orn	blu			
11	orn	grn	GND	13	GND
12	orn	yel			
13	pnk	orn			
14	pnk	yel	GND	17	GND
15	tan	brn			
16	pnk	tan	VCC		+3V3
17	tan	orn	VU	12	+5V

18	tan	grn	VU	14	+5V
19	tan	ppl	VCC		+3V3
20	tan	yel	CLKIN		
21	wht	yel	GND	21	GND
22	wht	blu	R8	22	LED_LAT4
23	tan	blu			
24	wht	orn	GND	25	GND
25	wht	pnk	U8	26	ADC_CS
26	wht	gry			
27	wht	ppl	GND	29	GND
28	wht	tan	N7	30	ADC_IOCLK
29	wht	brn			
30	wht	grn	GND	33	GND
31	pnk	blu	R7	34	ADC_OUTPUT_BUF
32	pnk	grn			
33	pnk	ppl	GND	37	GND
34	pnk	gry	U5	38	NC
35	blu	brn			
36	yel	brn	GND	3	GND
37	orn	brn	V15	4	LED_SCLK
38	pnk	brn			
39	grn	brn	GND	7	GND
40	ppl	brn	N11	8	LED_LAT1
41	gry	brn			
42	gry	tan	GND	11	GND
43	ppl	orn			
44	blu	orn			
45	grn	orn	GND	15	GND
46	yel	orn			
47	orn	pnk			
48	yel	pnk	GND	19	GND
49	brn	tan	T10	20	LED_LAT3
50	tan	pnk	VCC		+3V3
51	orn	tan	VU	16	+5V
52	grn	tan	VU	18	+5V
53	ppl	tan	VCC		+3V3
54	yel	tan	CLKIN		
55	yel	wht	GND	23	GND
56	blu	wht	T8	24	LED_LAT5
57	blu	tan			
58	orn	wht	GND	27	GND
59	pnk	wht	V8	28	ADC_INPUT
60	gry	wht			
61	ppl	wht	GND	31	GND

62	tan	wht	P8	32	ADC_SYSCLK
63	brn	wht			
64	grn	wht	GND	35	GND
65	blu	pnk	T7	36	NC
66	grn	pnk			
67	ppl	pnk	GND	39	GND
68	gry	pnk	V5	40	NC

Peripheral Board

Description

The peripheral board serves two purposes: to display the spectrum of the output audio signal on the LED displays, and to capture the user input from the nine slide potentiometers.

Pins 12, 14, 16 and 18 on the 40-pin connector (J1) provide +5V power to the entire board. This +5V supply is used directly to power the slide potentiometers, ADC, and analog voltage references. The LED display portion of the board requires +3.3V, so a linear regulator is used to provide +3.3V to the LEDs and LED driver ICs.

There are two serial busses on the peripheral board, the LED serial bus and the ADC serial bus. The timing diagrams for these busses are shown in detail in Chapter 5. For signal integrity and interfacing purposes, a line driver IC is used between all devices on the peripheral board and the FPGA. For more detail, see the "components" section.

Schematic

The schematic for the peripheral board was entered using PCB Artist, supplied by Advanced Circuits. It is shown in Figures 8-12. Unfortunately, due to the limitations of the schematic entry software, some of the text on the schematic cannot be enlarged, and the schematic cannot be exported to another format. Because of this, it is best that the schematic is viewed in the PCB Artist program, or in the separate PDF documents.

Layout

The PCB layout for the peripheral board was also done using PCB Artist, supplied by Advanced Circuits. The top and bottom copper layers are shown, with silkscreen information, in Figure 13 and Figure 14. The accompanying silkscreen for the top and bottom layers are shown in Figure 15 and Figure 16. Note that for the solder mask images, any white area receives the solder mask, and the non-white areas are exposed copper for soldering. A color legend for the PCB artwork is given in Table 4.

Table 4: PCB Artwork Color Legend

Layer	***Artwork Color***
Via	
Through Hole Pad	
Top Silkscreen	
Top Copper	
Top Solder Mask	
Top SMT Pads	
Bottom Silkscreen	
Bottom Copper	
Bottom Solder Mask	
Bottom SMT Pad	
Board Outline	

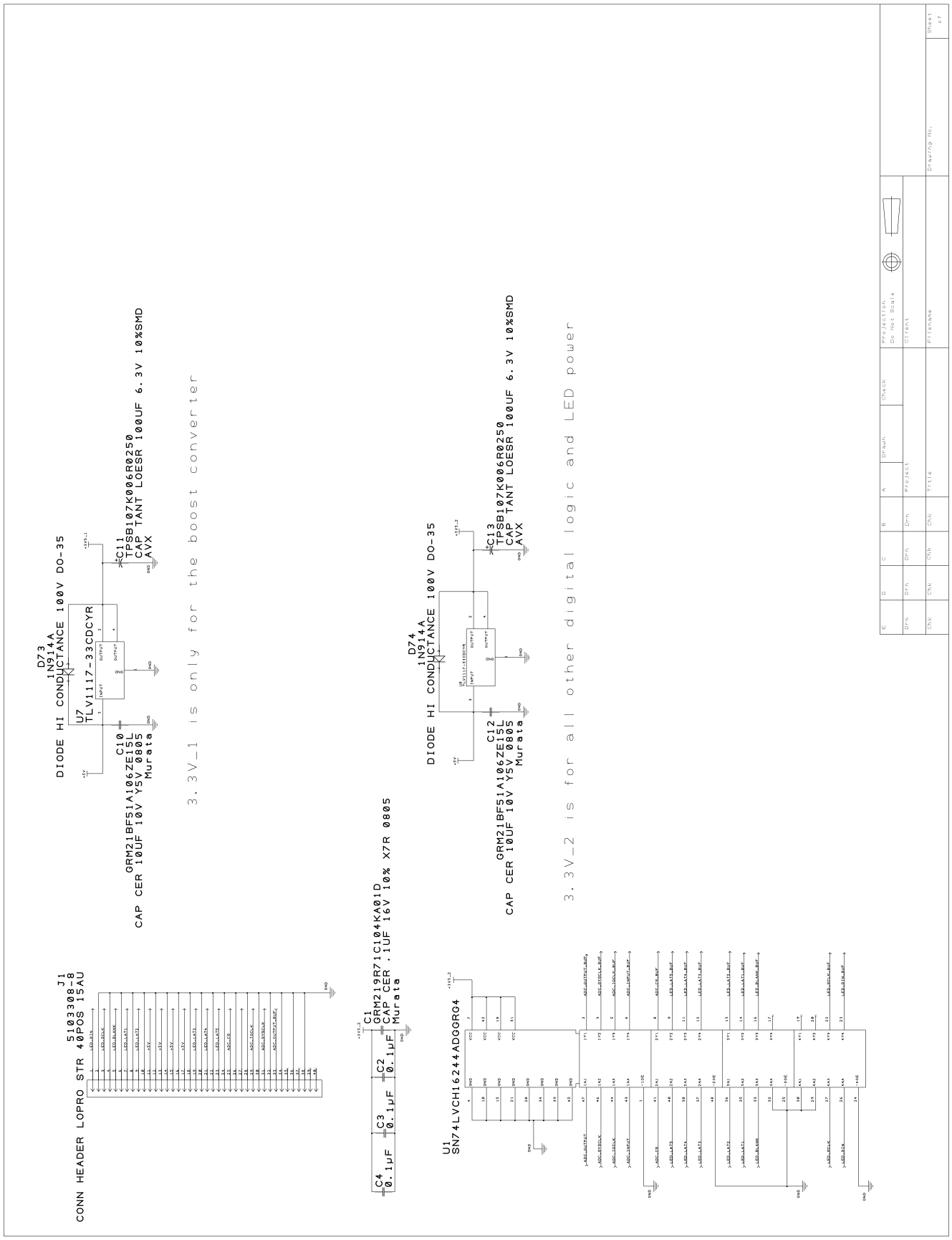

Figure 8: Peripheral Interface Schematic, page 1: Pin header, driver, and linear regulator

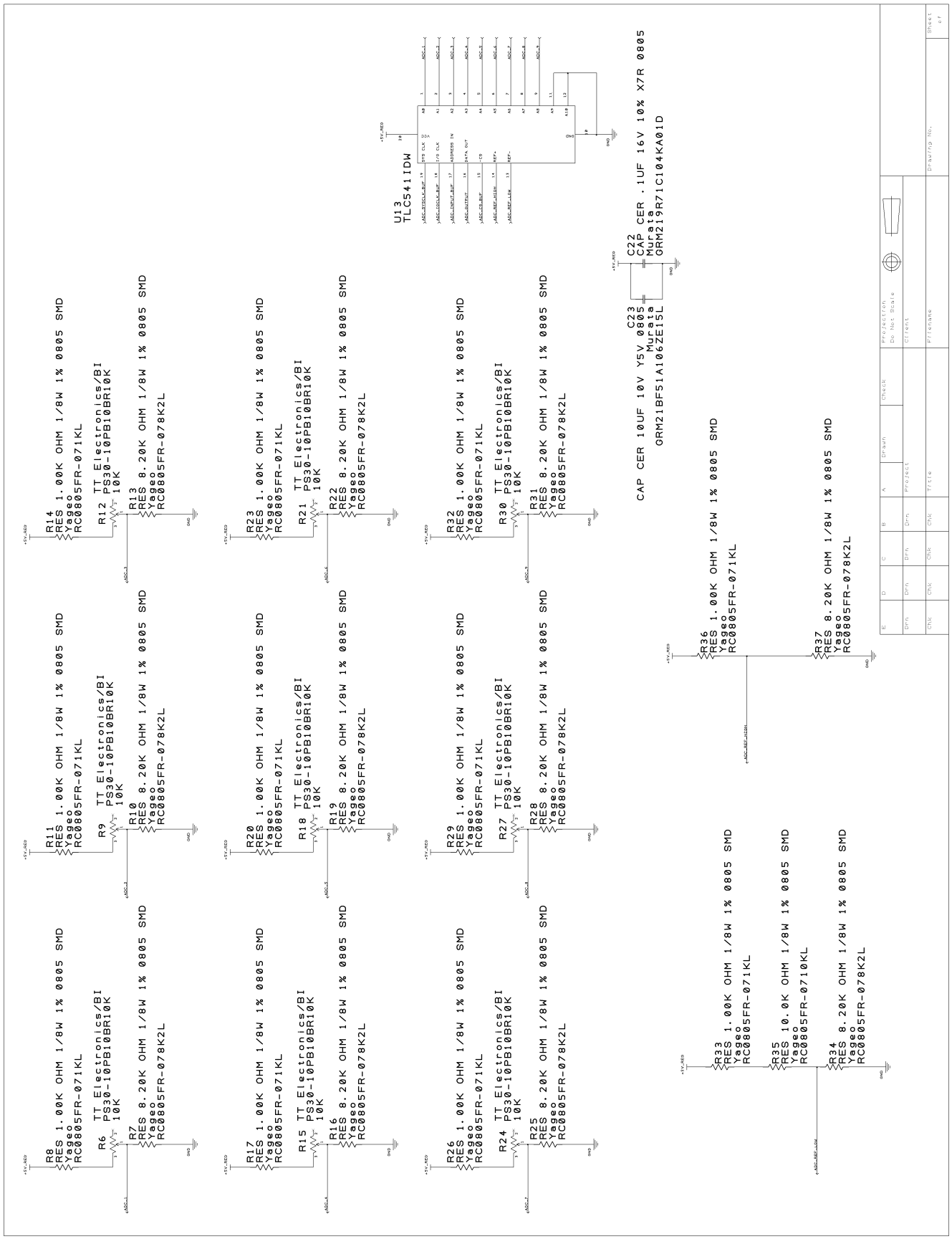

Figure 9: Peripheral Interface Schematic, page 2: slide potentiometers, ADC, and analog references

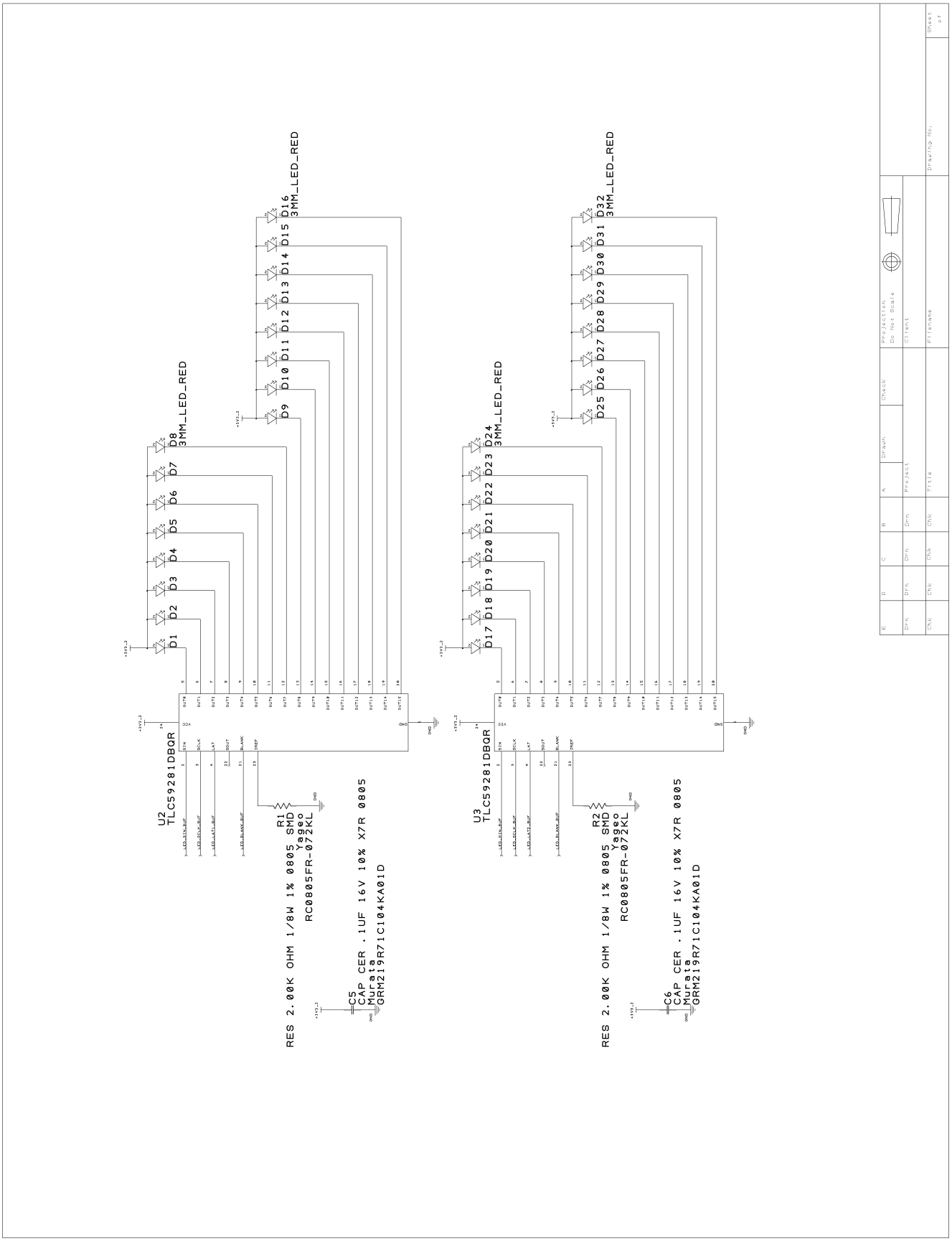

Figure 10: Peripheral Interface Schematic, page 3: LED drivers

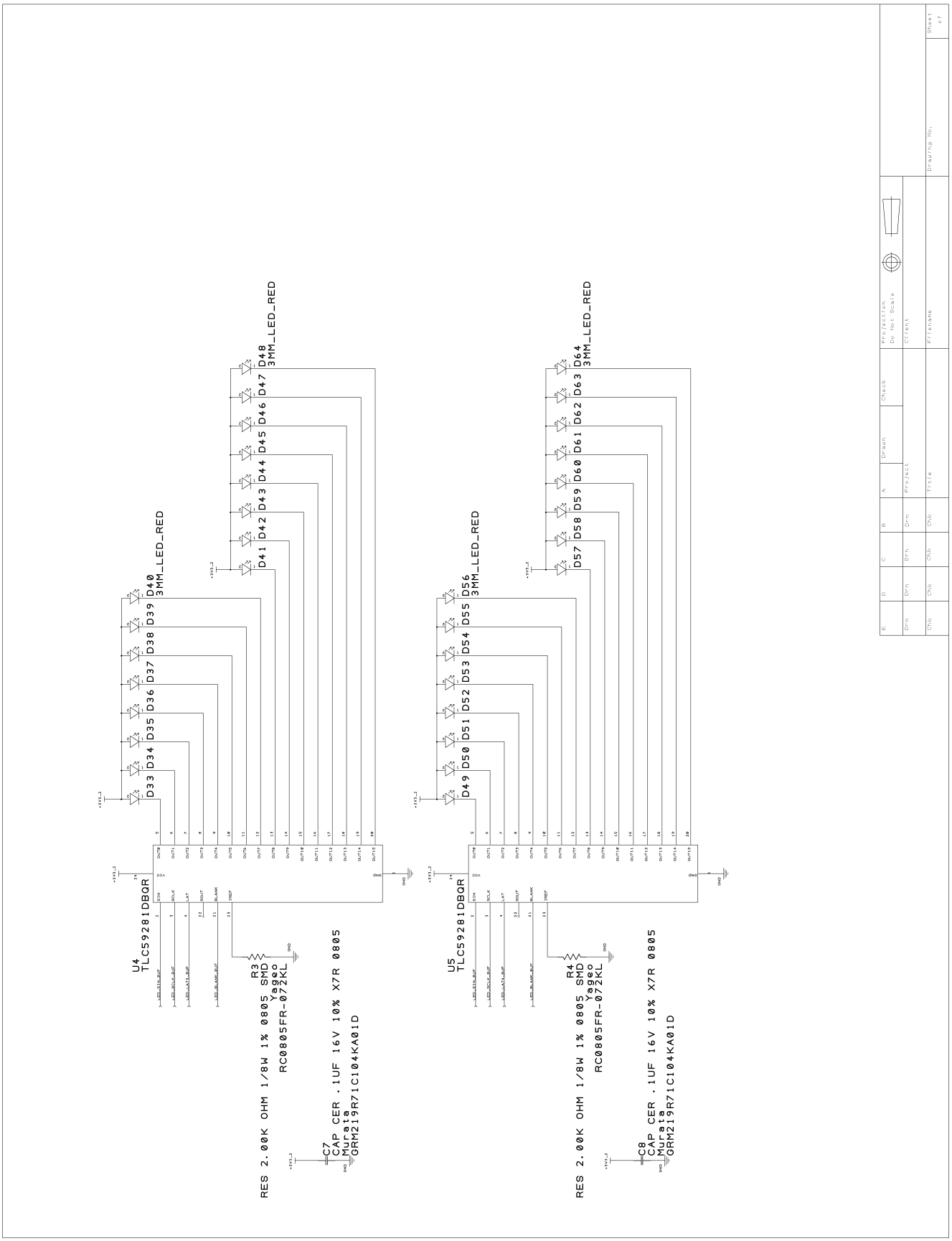

Figure 11: Peripheral Interface Schematic, page 4: LED drivers

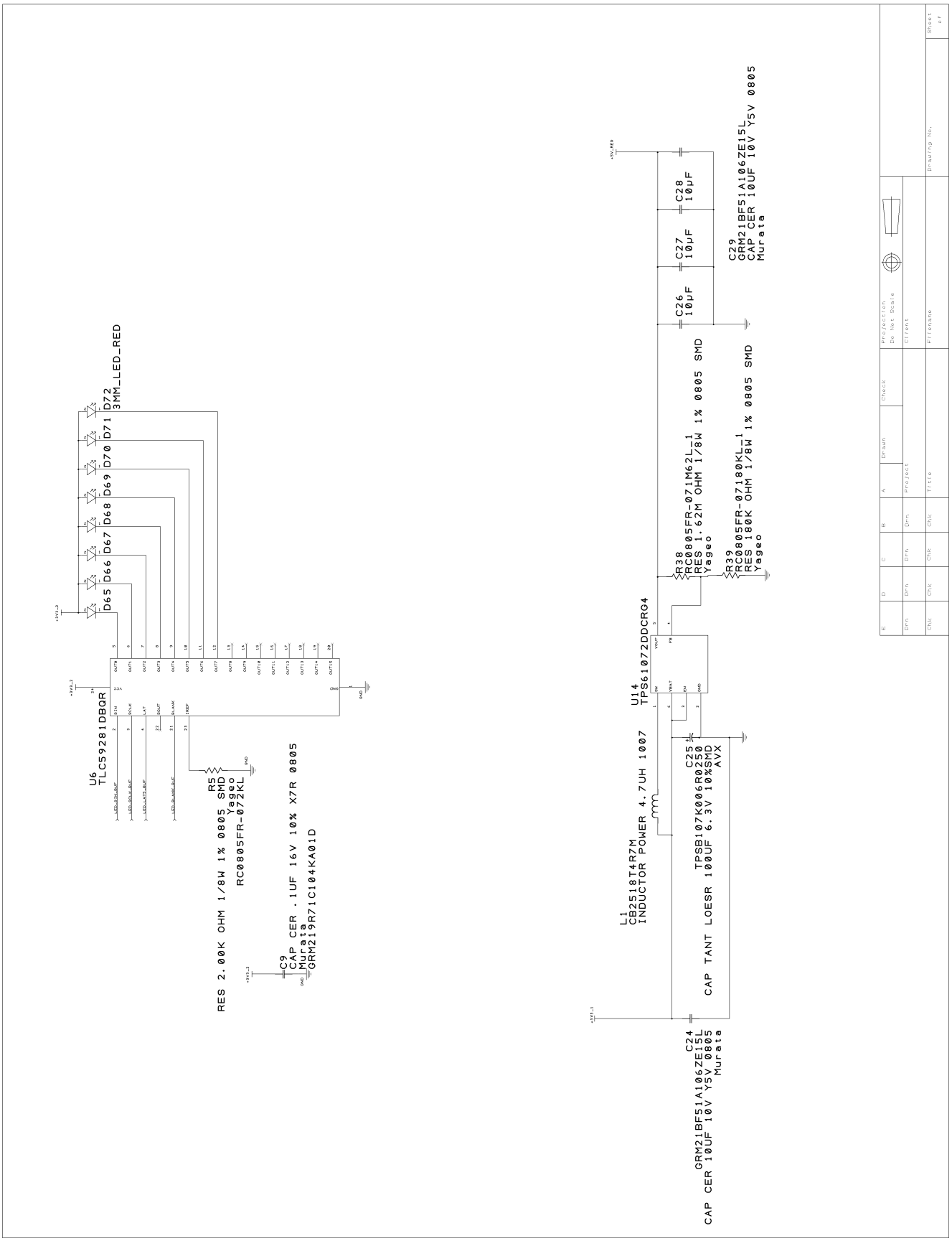

Figure 12: Peripheral Interface Schematic, page 5: LED drivers

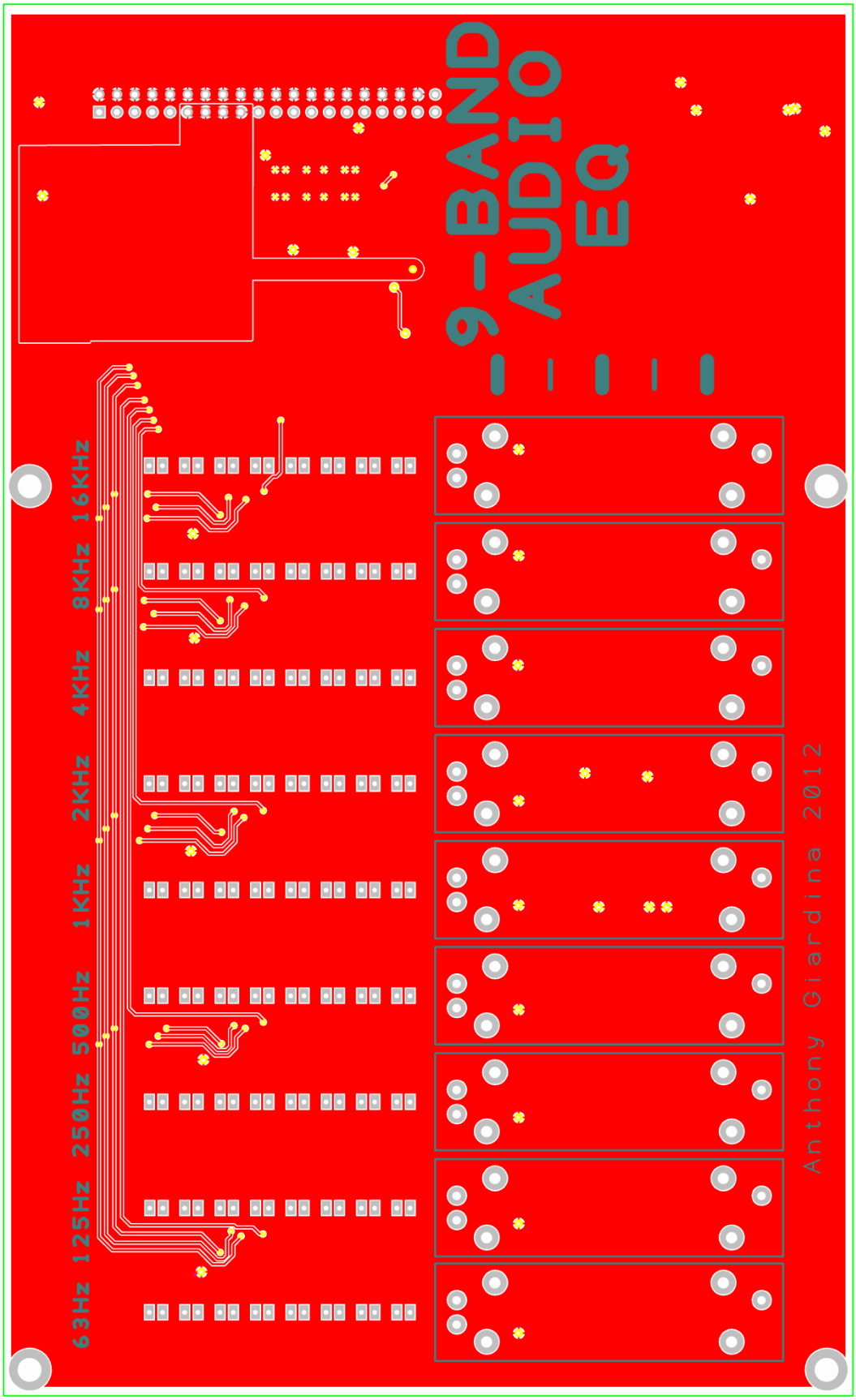

Figure 13: Peripheral Interface Board Artwork, top layer copper and silkscreen

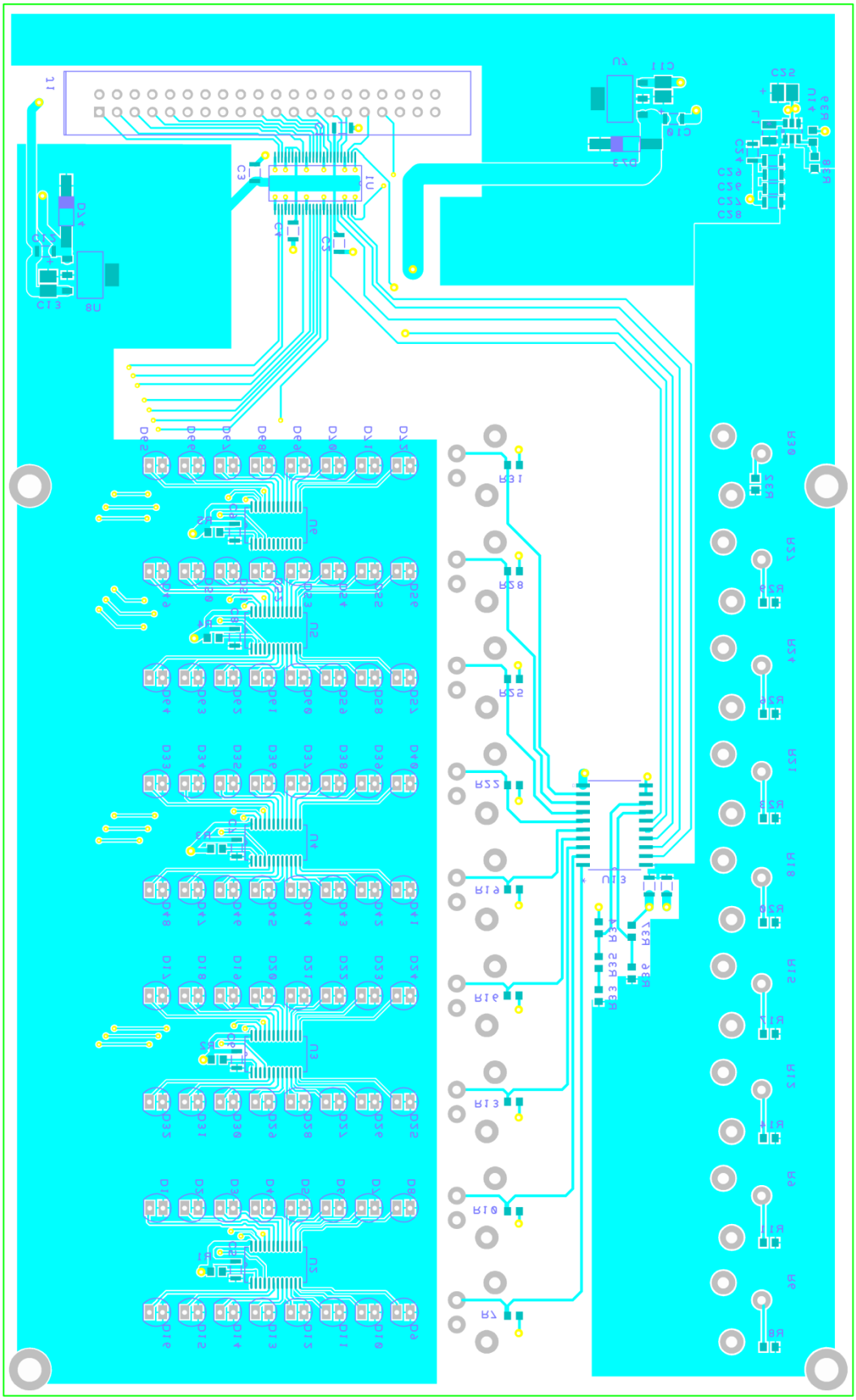

Figure 14: Peripheral Interface Board Artwork, bottom layer copper and silkscreen

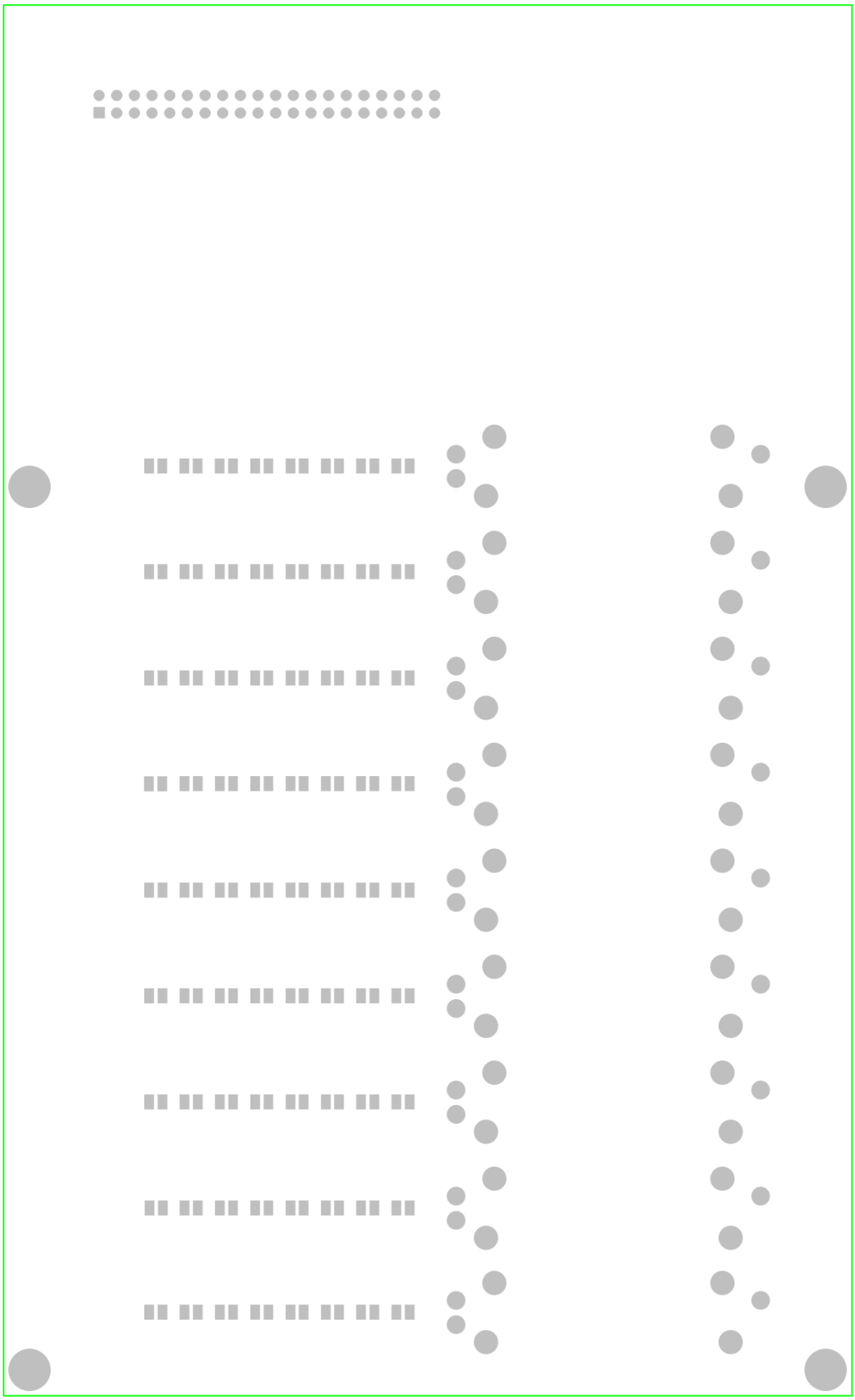

Figure 15: Peripheral Interface Board Artwork, top layer solder mask

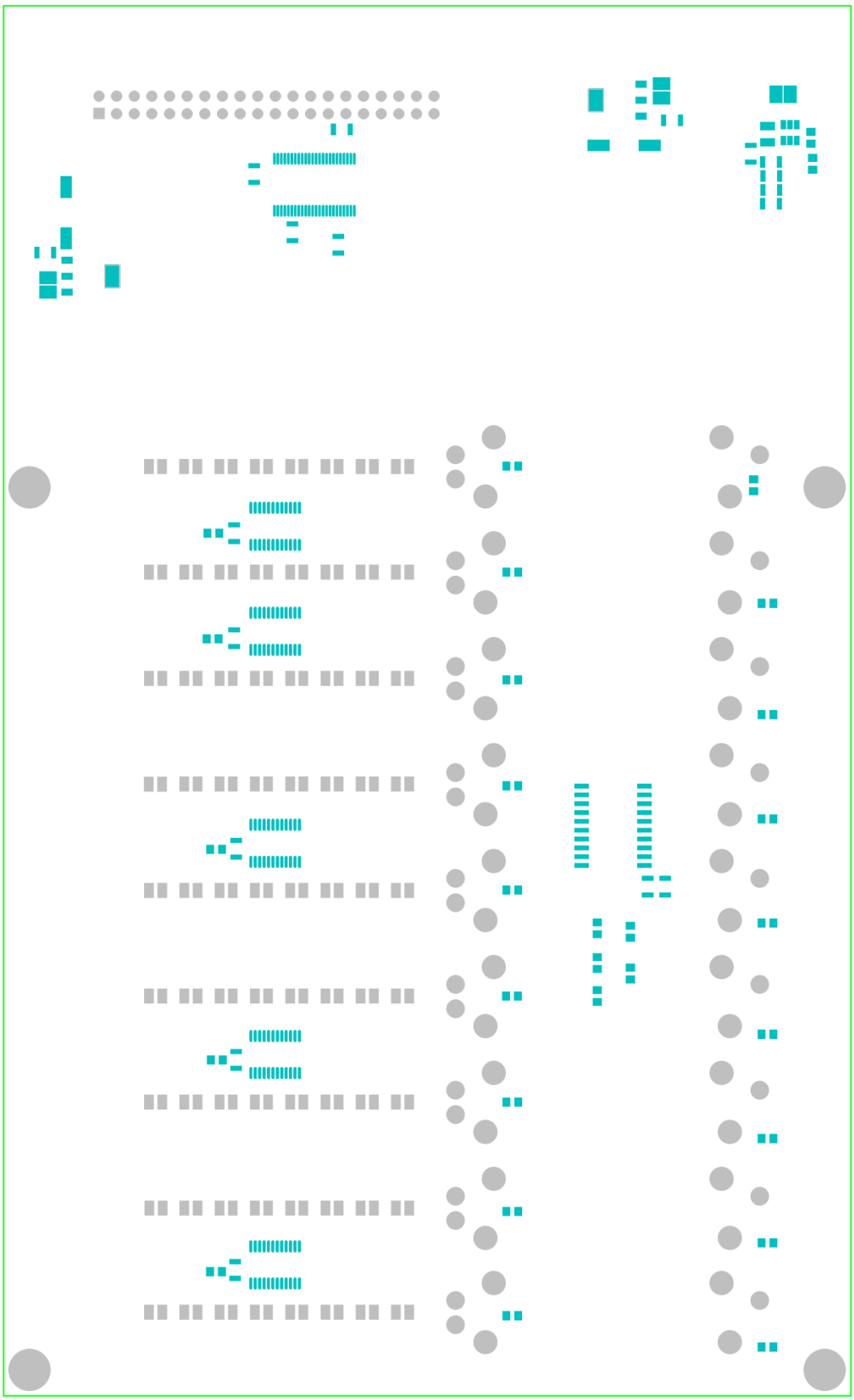

Figure 16: Peripheral Interface Board Artwork, bottom layer solder mask

Components

This section describes the reasons for selecting certain components, and their implementation details. Detailed timing diagrams, serial bus information, and power consumption information is presented later in this report.

TLC59281 16-Channel, Constant-Current LED Driver

LED driver ICs are necessary for this design. Driving all 72 LEDs directly from the FPGA would require more pins than are available on the Atlys FPGA development board [3], and would require the FPGA to sink a large amount of current [4]. The TLC59281 was chosen for the following reasons:

- Ability to drive a large number (16) of LEDs per IC [5]. This reduces the total number of drivers required, and thus reduces board complexity.
- Simple serial bus, supporting multiple devices on a single bus [5]. This reduces the number of pins required on the FPGA, and also reduces board complexity because fewer traces are required for the serial bus.
- LEDs are driven using a constant current source. All of the LEDs on one IC are driven with the same current, which is programmed by a single resistor [5]. This reduces the total part count by not requiring one resistor per LED.

Analysis of the TLC59281 serial interface bus voltage levels is shown in Figure 17. This shows that the device will be able to properly interface with the SN74LVCH16244A and the FPGA, since all of the $V_{OH(min)}$ values are larger than the corresponding $V_{IH(min)}$ values, and all of the $V_{IL(max)}$ values are larger than the $V_{OL(max)}$ vales. The "BUS" signal is a 3-bit bus that includes the signals LED_SCLK, LED_SIN and LED_BLANK. These signals are connected to each TLC59281. Each TLC59281 is connected to its own unique latch signal, LED_LAT1 for driver #1, LED_LAT2 for driver #2, etc. There are 5 drivers and 5 latch signals.

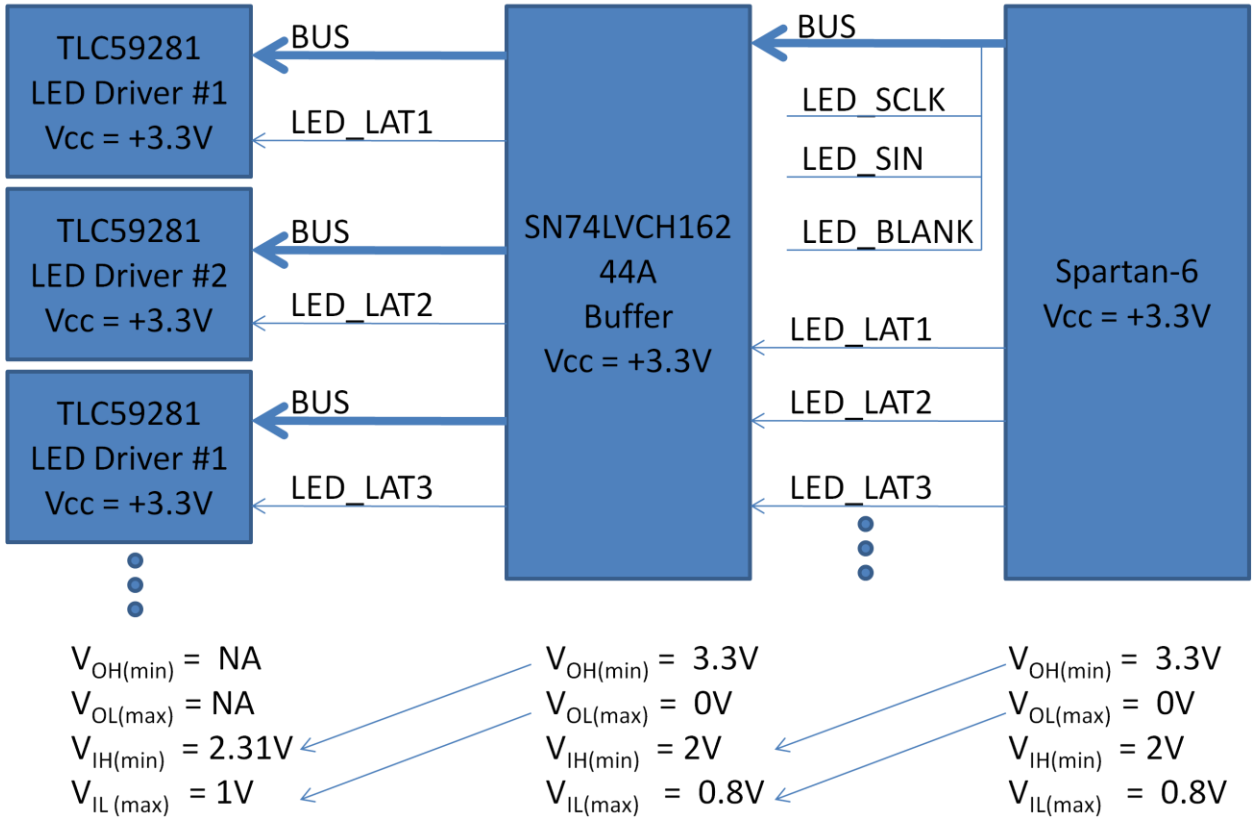

Figure 17: TLC59281 LED Driver serial bus voltage levels

Traces for the TLC59281 serial bus were not routed as transmission lines. The longest trace for this bus is approximately 20cm, and the maximum clock frequency used is 1.024MHz. Equation 1 shows the calculation of the wavelength of a 1.024MHz signal on FR-4 PCB, where λ is the wavelength, *c* is the speed of light, ε_r is the relative permittivity, and *f* is the frequency. Since the wavelength of the signal is greater than 10 times the longest trace length, arbitrary trace routing is acceptable [6].

$$\lambda = \frac{c}{\varepsilon_r * f} = \frac{3x10^8}{4.8 * \ 1.024x10^6} \cong 61m$$

Equation 1: Wavelength calculation for TLC59281 serial bus

To program the drive current for each LED, the resistor R_{IREF} is chosen using Equation 2 [5]. A drive current of 2mA was experimentally determined to be acceptable (see section on passive/discrete components and connectors,) so a 24.9KΩ resistor is used for R_{IREF}.

$$I_{OUT\ (IDEAL)} = 42 * \frac{1.20}{R_{IREF}} = 42 * \frac{1.20}{24900} = 2.02mA$$

Equation 2: Drive current calculation for TLC59281 programming resistor selection

TLC541L 8-Bit Analog-To-Digital Converter with 11 Inputs

The TLC541L is used to sample the positions of the nine slide potentiometers. An external ADC is required, because neither the Atlys FPGA development board, nor the Spartan-6 FPGA contain a single ADC. The TLC541L was chosen for the following reasons:

- 8-bit precision is sufficient to approximate a continuous scale of movement on the slide potentiometers (see Equation 3.)
- Simple serial interface facilitates ease of integration with the FPGA.
- High and low voltage references can be set independently of each other, at any voltage between -0.1V and 2.5V (for the low voltage reference) and 2.5V to V_{CC}+0.1V (for the high voltage reference. [7]) This allows for easy dynamic range calibration.
- A single TLC541L can sample all nine potentiometers. This dramatically decreases the system complexity, since only a single ADC and corresponding serial bus is required.

The slide potentiometers have a sliding range of 30mm [8]. Assuming that every ADC PCM value is used over the range of potentiometer settings, Equation 3 shows that there is one PCM code for each 0.12mm of potentiometer movement. It is unlikely that a user would be able to discriminate the difference in potentiometer position of 0.12mm.

$$ADC\ Precision = \frac{Slide\ potentiometer\ range}{Possible\ ADC\ PCM\ codes} = \frac{30mm}{2^8\ PCM\ codes} \cong \frac{0.12mm}{PCM\ code}$$

Equation 3: ADC precision calculation

Analysis of the TLC541L is especially important, because the TLC541L has a supply voltage of +5V, and is interfacing to components with supply voltages of +3.3V. The analysis of the TLC541L interface bus in Figure 18 shows that even though the supply voltages differ between the TLC541L and the SN74LVCH16244A, they can still interface with each other.

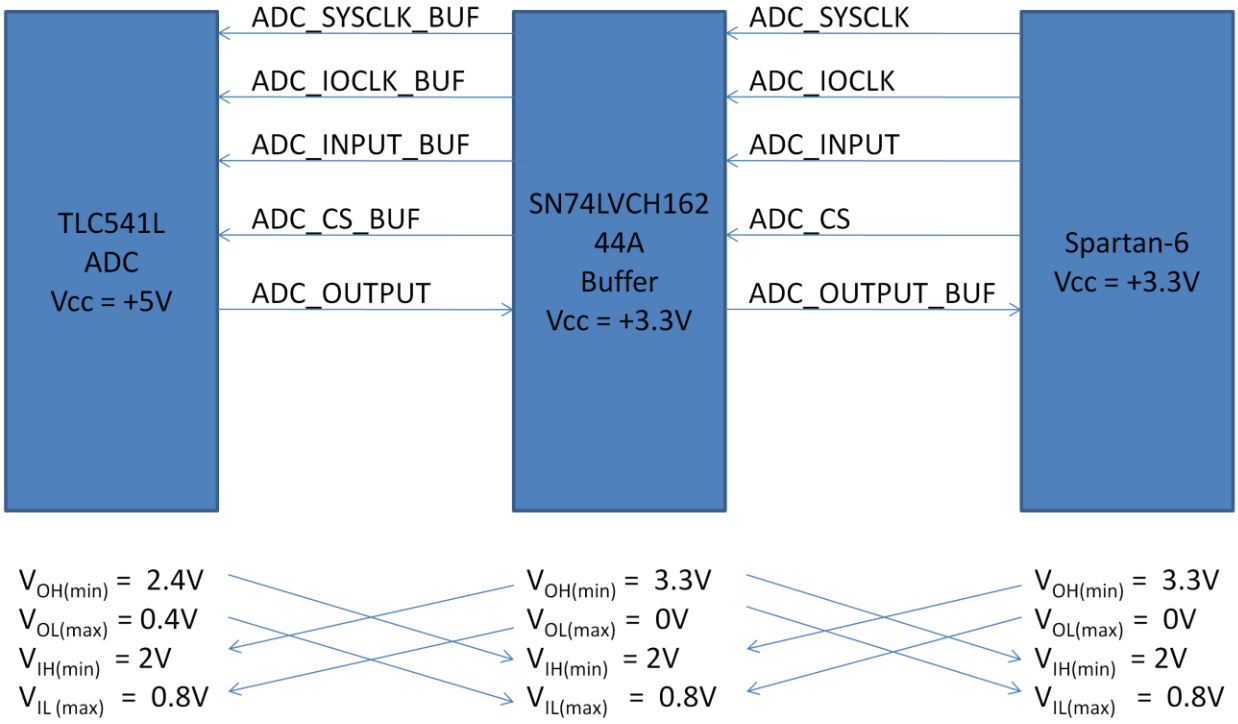

Figure 18: TLC541L ADC serial bus voltage levels

The traces for the TLC541L serial bus are not routed as transmission lines. Since the serial bus uses a maximum clock speed of 2.048 MHz, the wavelength on the bus is about 30.5m, as shown in Equation 4. This is much greater than the longest trace length of approximately 13cm, so arbitrary trace routing is acceptable [6].

$$\lambda = \frac{c}{\varepsilon_r * f} = \frac{3x10^8}{4.8 * 2.048x10^6} \cong 30.5m$$

Equation 4: Wavelength calculation for TLC541L serial bus

SN74LVCH16244A 16-Bit Buffer/Driver

The main purpose of the SN74LVCH16244A is to provide proper interfacing between the TLC541L and the Spartan-6. The Spartan-6 operates on a 3.3V voltage supply [9] [10], and it's inputs are not tolerant of +5V [10]. Thus, the TLC541L could damage the Spartan-6 if they were directly interfaced together. The SN74LVCH16244A is a 3.3V part, but has 5V tolerant inputs, and the proper voltage levels to facilitate interfacing between the TLC541L and the Spartan-6. Since the SN74LVCH16244A IC is already on the board, all of the other digital signals are buffered with it, for added signal integrity.

TLV1117-33 3.3V Linear Regulator

The TLV1117-33 linear voltage regulator is used to generate 3.3V supply voltages for the LEDs and driving circuitry. It is simple, requires a small number of external components, and provides sufficient output current for all the devices it powers (see power estimation section.)

Passive/Discrete Components and Connectors

Resistors

For all of the resistors on the peripheral board, Yageo RC series chip resistors are used. These resistors have a ±1% tolerance, 1/8W power rating, wide range of values, and come in the 0805 package, so that they are easy to solder by hand. They are also inexpensive, at $0.019/each when priced at the quantities used for this project.

Capacitors

For decoupling capacitors, the Murata GRM219R71C104KA01D 0.1uF ceramic capacitor is used. It has a 16V rating, which is more than 3 times the largest supply voltage of +5V.

Two different types of capacitors were used for bulk capacitance. When a ceramic capacitor is required, the Murata GRM21BF51A106ZE15L 10uF capacitor is used. For applications requiring low ESR, the AVX TPSB107K006R0250 100uF tantalum capacitor is used, because of its low ESR value of 250mΩ.

Diodes

For the TLV1117-33 linear regulator, a 1N914A diode is used as suggested by the datasheet [11]. Although this diode uses the DO-35 leaded package, it is soldered on to surface mount pads to reduce board complexity. This diode was used instead of a surface mount part because it was freely available, and would not add cost to the project.

For the LED displays, Kingbright WP710A10SRC/E red LEDs were chosen, because of their low cost and small size. The LEDs were experimentally determined to have acceptable brightness at 2mA of supply current.

Connectors

The only connector on the board is the Assmann AWHW40G-0202-T-R pin header. This connector was chosen because it is an industry standard, and mating connectors are readily available.

Implementation, Testing, and Rework

The board was built and tested in two steps. First, the connector J1 and the power supplies were populated onto the board. Passive loads were soldered onto the output terminals of the power supplies, to simulate the loads that they would be driving. The board was then powered on, and the supply voltages were checked. The +5V supply coming onto the board (J1, pins 12,14,16,18) was good, and the +3.3V supply from the TLV1117-33 was good. However, the +5V supply coming from U14 was very low, and U14 was too hot to touch. After multiple debugging attempts of U14, it was removed from the board, and the +5V_REG net was jumped directly to the +5V net, shown in Figure 19. There were no further problems using the +5V net as a supply voltage for the ADC circuit. Another problem found during board testing was in incorrect connection to the slide potentiometers. The nine ADC inputs were supposed to be connected to the potentiometer wiper, pin 2. Instead, they were connected to a NC body pin, resulting in a constant voltage at the ADC inputs, regardless of wiper position. This situation was remedied by placing a jumper between pin 2 of the slide potentiometers, and the incorrectly connected body pin, as shown in Figure 20. After this rework, the slide potentiometers worked as expected.

For a functional test of the board, the entire board was populated. Using a test program written and programmed onto the FPGA, the slide potentiometer values were read from the ADC, and those values were displayed on the corresponding LED banks. This proved that the peripheral board functioned as planned.

The fully populated board is shown in Figure 21 and Figure 22.

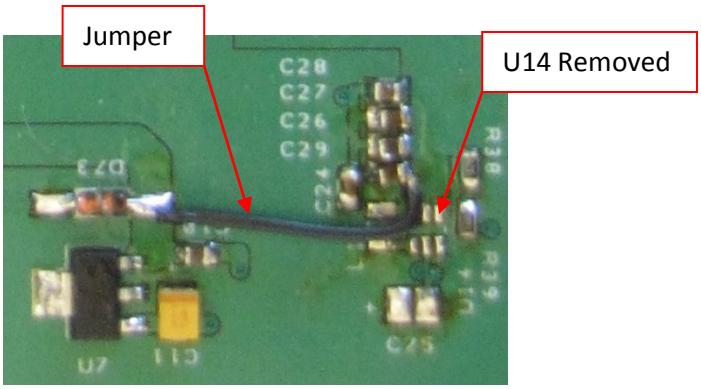

Figure 19: Rework #1: +5V_REG and +5V nets connected with wire

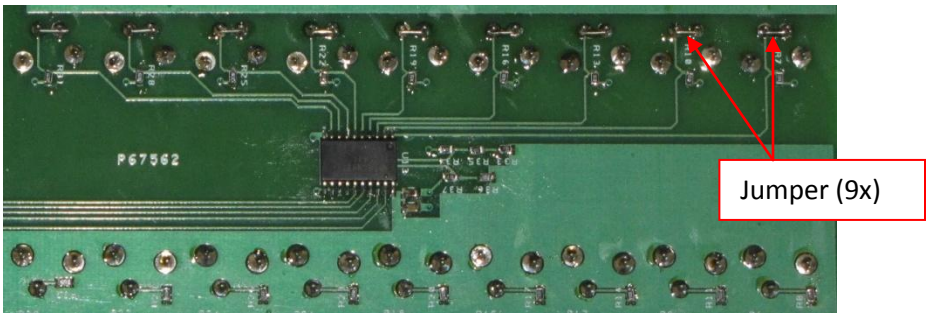

Figure 20: Rework #2: Jumper soldered onto slide potentiometers to connect wiper to ADC input

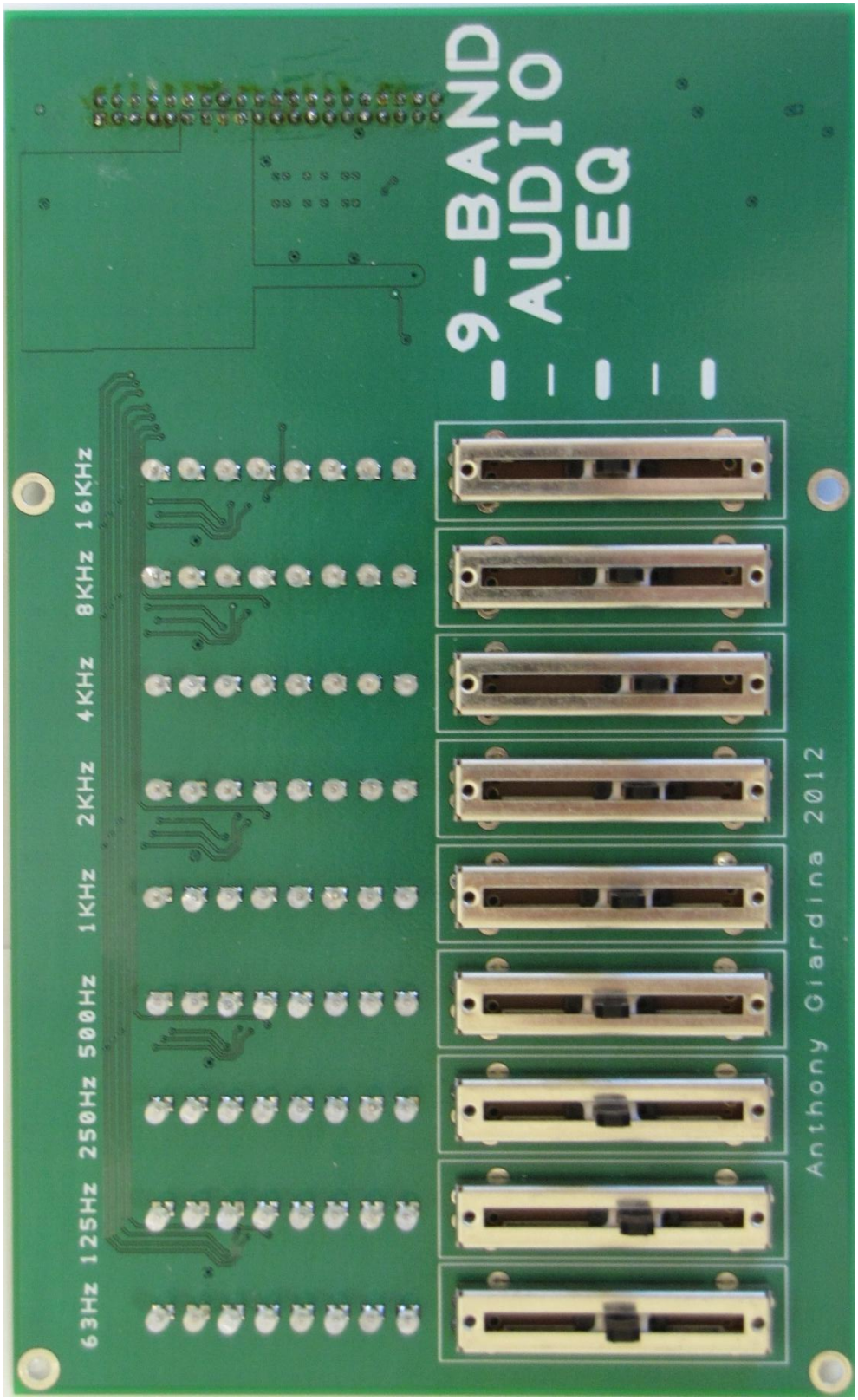

Figure 21: Peripheral board, fully populated, top view

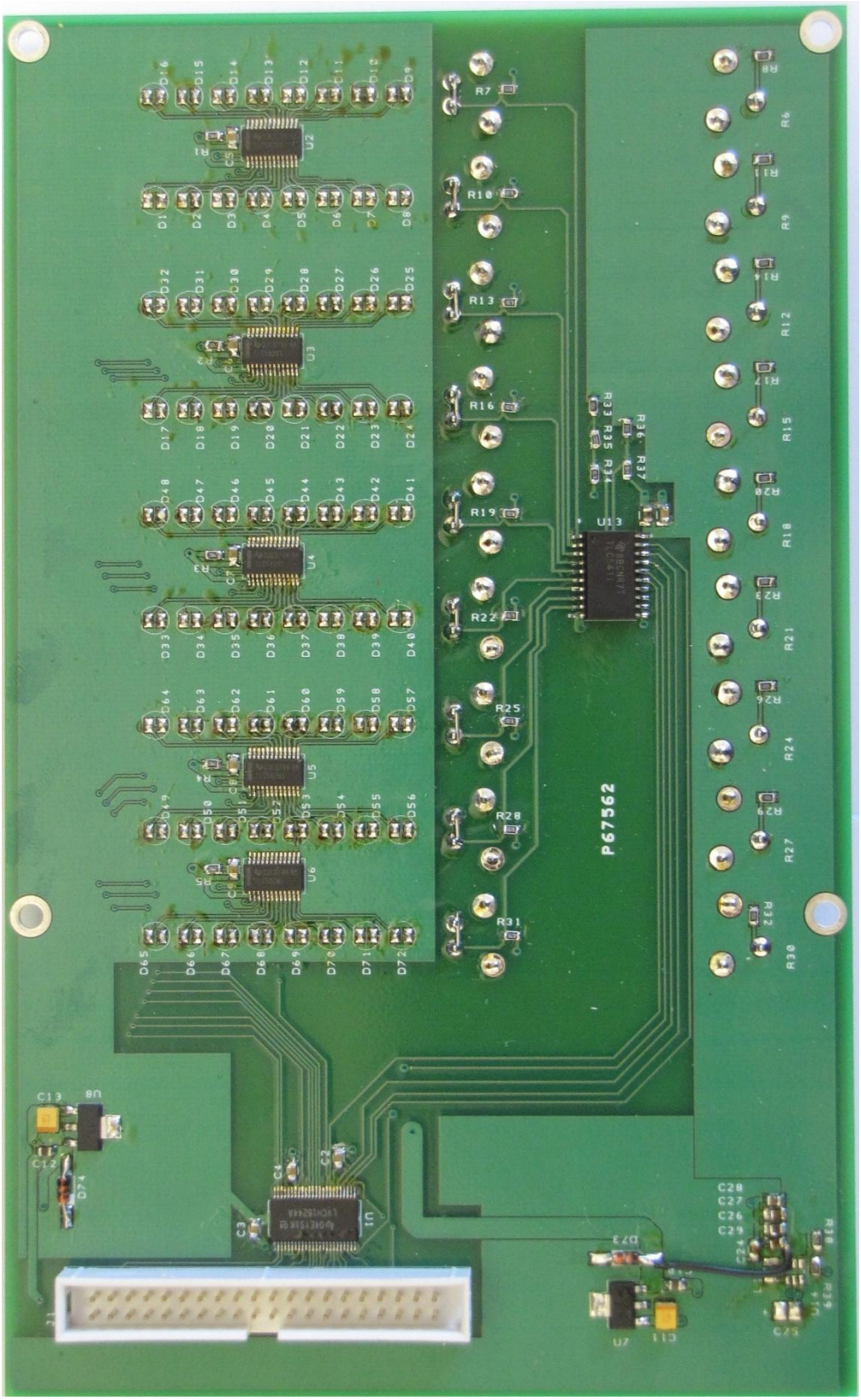

Figure 22: Peripheral board, fully populated, bottom view

Atlys FPGA Development Board

Specification

The Spartan-6 FPGA was chosen as the FPGA for this project, mainly because of its low cost and rich feature set. The most important feature of the Spartan-6 is it's DSP48A1 slices, which provide DSP building blocks to make the implementation of DSP systems much more efficient. At an academic price of $199, the Digilent Atlys was the least expensive Spartan-6 development board available, and yet still has a multitude of useful features, but was found to have some drawbacks.

The following specifications were required of the chosen FPGA development board:

- Spartan-6 FPGA with at least 54 DSP48A1 slices. The FIR filters used in this project require between 2 and 6 DSP48A1 slices each, depending on their configuration [12]. 54 DSP48A1 slices should provide sufficient resources to implement the required 9 filters.
- External connector to connect to the peripheral interface board, with a minimum of 20 GPIO pins, ground pins, and power supply pins.
- Audio Codec, or ADC and DAC with precise sampling rate of at least 40kHz.
- Flash ROM memory, to store FPGA programming file when the power is turned off.

Since the Digilent Atlys FPGA development board met the above specifications, it was chosen for this project.

Implementation Issues

There were some implementation issues with the Atlys FPGA development board, all concerning the LM4550 audio codec. The first problem with the board was noise present at a power supply pin of the LM4550, and the second was output noise on the line-out channel of theLM4550.

LM4550 Power Supply Noise

The digital supply pin VDD2 (pin 9) of the LM4550 was found to have almost $1V_{Pk-Pk}$ noise, at approximately 25MHz, shown in Figure 23. This is very significant, since the supply voltage is only 3.3V. This noise was caused by a low-pass LC filter, present on pin 9 [9]. The resonant frequency of this LC filter is approximately 5.8MHz, which is not the same as the 25MHz noise, but is within the same order of magnitude. To correct this, L1 was removed from the Atlys FPGA development board, and was replaced with a solder bridge, shown in Figure 24. An extra 0.01uF ceramic decoupling capacitor was also added in parallel to C27. After this rework, the noise on pin 9 was approximately $10mV_{Pk-Pk}$, a significant improvement.

LM4550 Line-Out Noise

When attempting to use the LM4550 to play audio or arbitrary waveforms, there is an unwanted noise source that is added to the line-out signal. This noise is approximately $800mV_{Pk-Pk}$, or 121mVRMS (using a σ=6.6.) Figure 25 shows the default Atlys test program (from Digilent) outputting a constant PCM code of 0 to the LM4550. Channel 2 (green) shows this output is actually very noisy, and the purple trace shows the FFT of channel 2. The noise frequency is approximately 300kHz. Figure 26 shows a loopback test of the LM4550. Channel 1 (orange) is connected to the line-in port, and shows a clean input sine wave. Channel 2 (green) shows the line-out port, with noise present. The purple trace shows the FFT of channel 2, indicating noise at about 340kHz. This noise is likely due to improper layout of the LM4550 on the Atlys FPGA development board and not improper LM4550 configuration/communication, since the test program provided by Digilent shows the same noise, and the noise is much higher than the Nyquist frequency. A remedy to this problem is outside

of the scope of this project. Thankfully, the noise is far above the range of human hearing, and does not appear to affect audio quality.

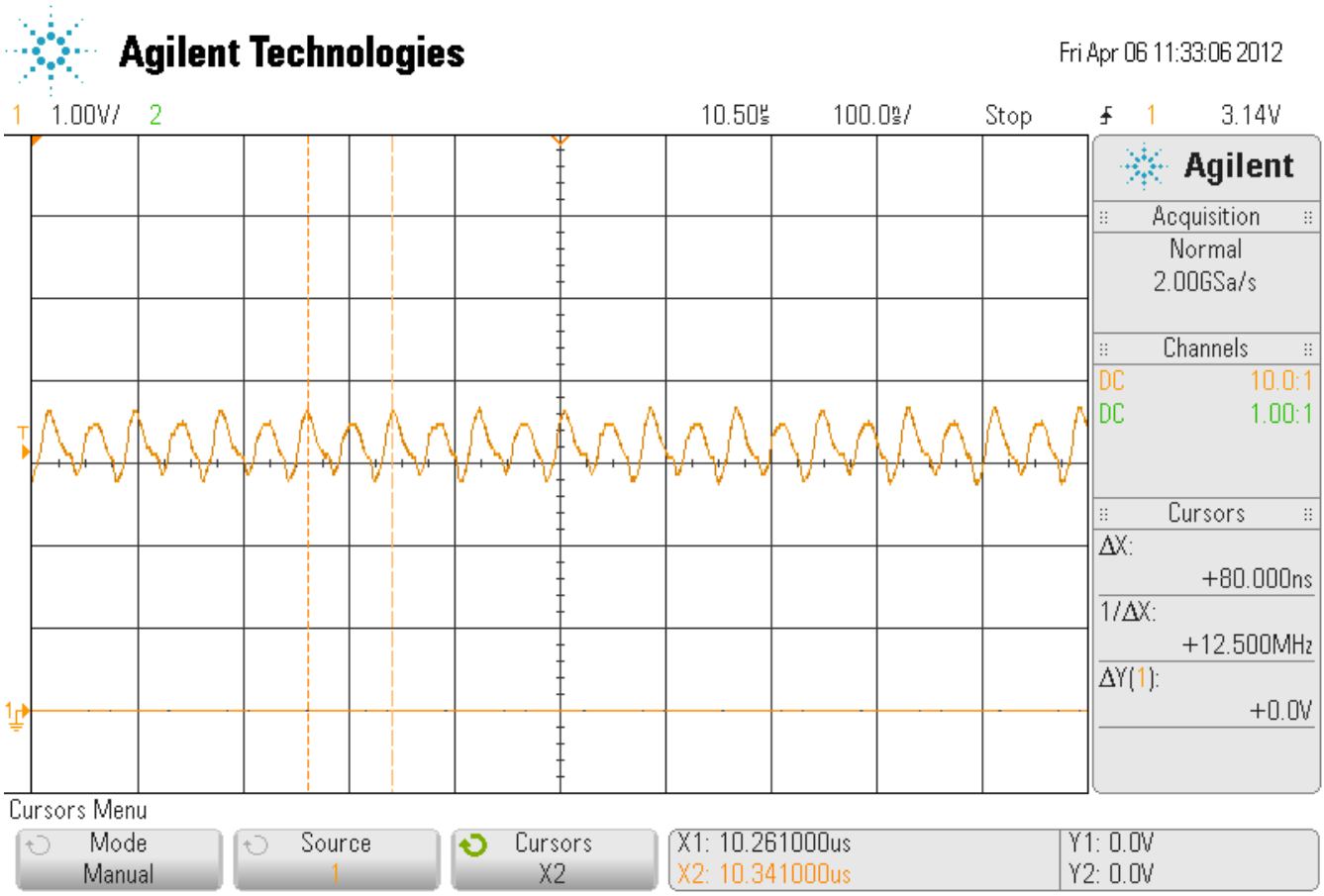

Figure 23: Supply noise on LM4550, pin 9, VDD2. CH1 connected to pin 9 of LM4550.

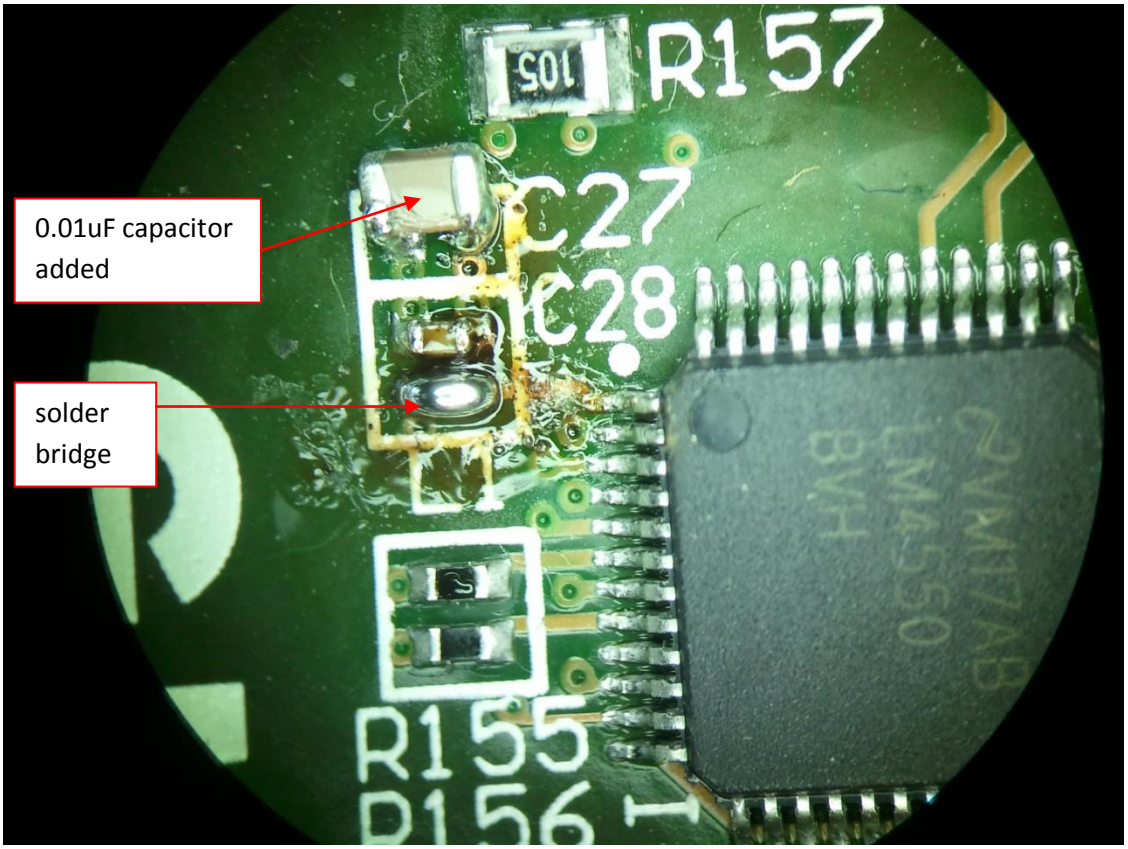

Figure 24: Rework #3: L1 replaced by solder bridge, and additional capacitor added in parallel to C27

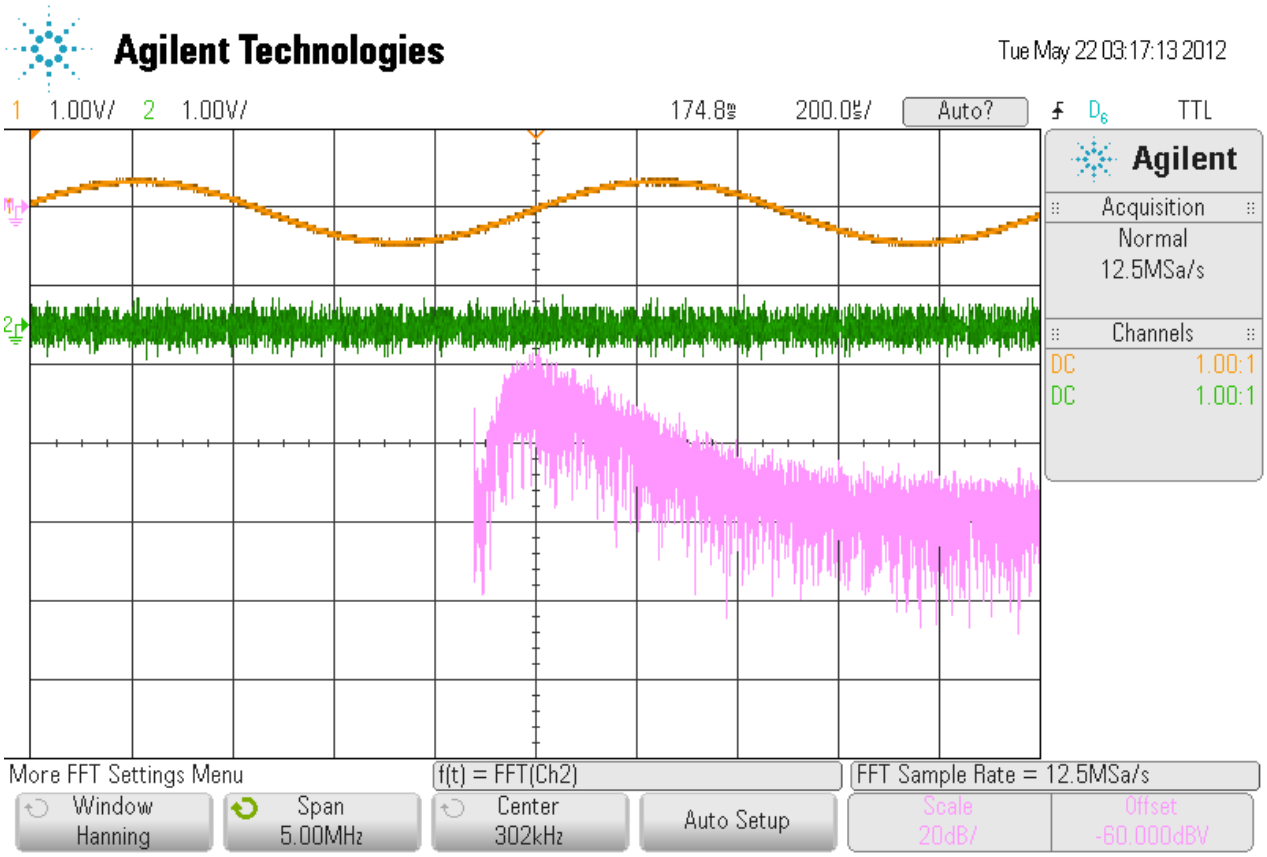

Figure 25: Output of Atlys test program, showing 300kHz noise on LM4550 line-out port. CH1: irrelevant, CH2: LM4550 line-out, M: FFT of CH2

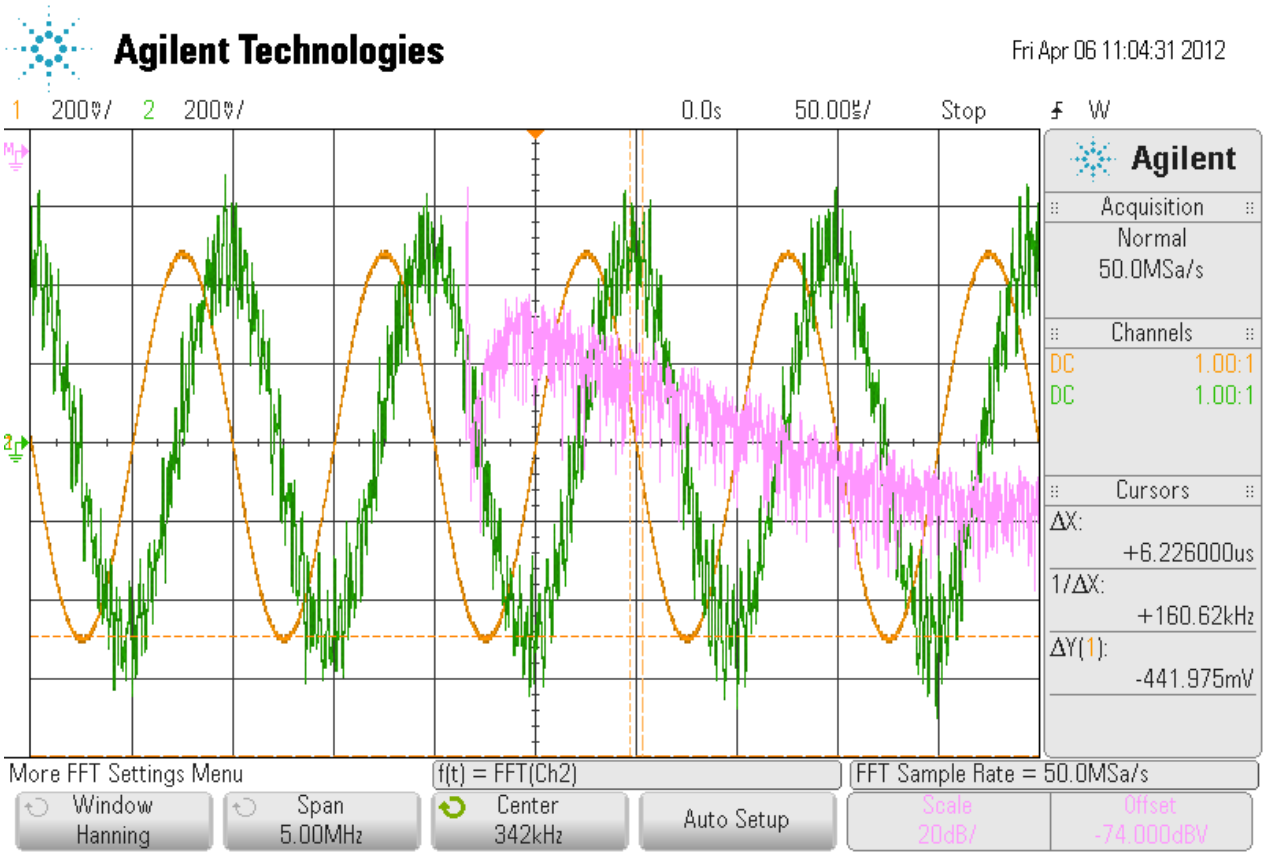

Figure 26: Loopback test of LM4550, showing 340kHz noise on line-out port. CH1: Line-in. CH2: line-out. M: FFT of CH2

Power Estimation

The main areas of concern for power estimation are the amount of power drawn through the four +5V supply pins of the interface cable, and the amount of power supplied on the peripheral board by the TLV1117-33. Table 5 shows an estimate of the maximum current drawn from the TLV1117-33, and Table 6 shows an estimate of the current drawn from the four +5V supply pins of the interface cable. The peripheral board 3.3V supply only draws a maximum of 345mA, which is less than half of its rated output current of 800mA. The total amount of current drawn through the four +5V supply pins of the interface cable is 354mA, and it is capable of supplying 1A.

Table 5: 3.3V current estimation for peripheral board

Part	***Current @ +3.3V***
LEDs, D1-D73	2mA * 72
LED drivers, U2-U6	40mA * 5
Buffer, U1	500uA
Total:	345mA

Table 6: 5V current estimation for peripheral board

Part	***Current @ +5V***
TLV1117-33, U8	345mA
Potentiometer resistor dividers, analog references	543uA * 11
ADC, U13	3mA
Total:	354mA

Chapter 5: Firmware Design

Firmware Development Toolchain

Firmware for this project is written entirely in VHDL. It is written in a very modular fashion, so that all of the individual components can be tested, and reused for another purpose if necessary. A combination of software tools were used for firmware development. The design flow is shown in Figure 27, and details are given in Table 7.

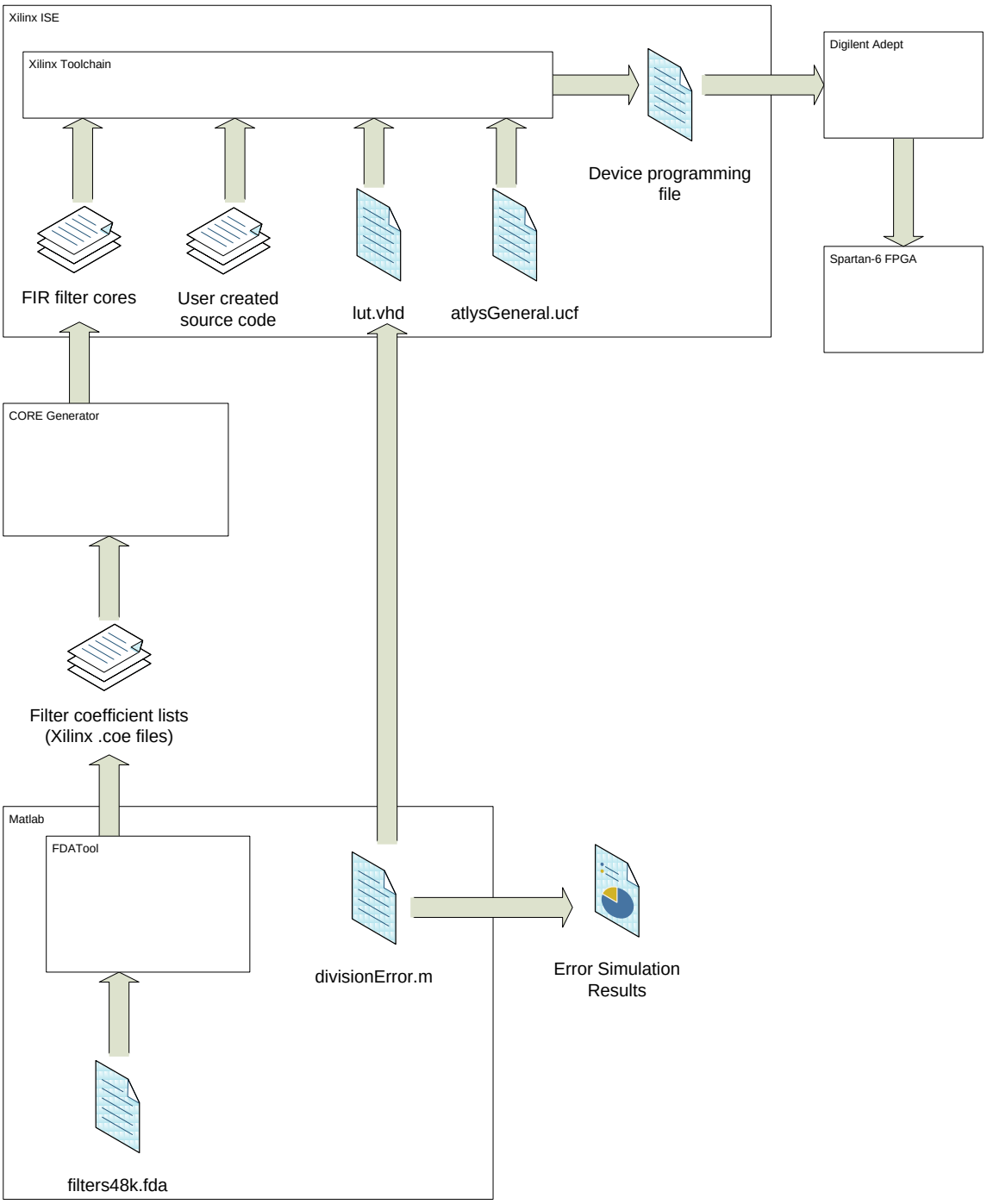

Figure 27: Firmware development design flow

Table 7: Software used for the firmware design flow

Software	Version	Vendor	Description and Use
Xilinx ISE	13.4 (64-bit)	Xilinx	The Xilinx ISE software suite is used to perform all of the VHDL synthesis, placement, routing, and programming file generation. In order to target the Spartan-6 device, the free Webpack license is insufficient. For this project, the System Edition Evaluation license was used. In order to properly use this license, the Trial License (which is installed by default with ISE) must be removed.
Matlab	R2010a, 7.10.0.499 (64-bit)	Mathworks	Matlab was used to run FDATool. The Matlab script divisionError.m also generates a lookup table for use in the firmware, and analyzes the error associated with that lookup table.
CORE Generator	13.4 (64-bit)	Xilinx	The core generator was used to run the Xilinx FIR Compiler 5.0, which uses the coefficient files generated with FDATool to generate filters for VHDL.
FDATool	4.7	Mathworks	FDATool is a component of the Matlab Signal Processing Toolkit and is used to generate FIR filter coefficients from filter specifications.
Digilent Adept	2.4.2	Digilent	The Digilent Adept program is used in conjunction with the Atlys hardware to configure the FPGA. It is an alternative to direct JTAG programming with the Xilinx system cable.

Top level Firmware Overview

Top Level Firmware Block Diagram

Figure 28 shows how all of the individual firmware modules are arranged within the top level firmware module, audioProcessor. For more information on how the firmware interfaces with the peripheral board, see Figure 2. Signals and busses colored in blue represent the audio signal processing chain, from end to end. Audio is sampled by the LM4550 at a rate of 48KHz. Audio samples from the LM4550 are sent to the audDriver module, which communicates with the LM4550 using the AC-Link serial bus. Once the audio input samples are read from the serial bus, they are sent to all nine FIR filters. The filters also sample at 48KHz, and read in a new sample shortly after a sample is placed on their input by the audDriver module. Each filter separates out a unique section of the spectrum, which is then sent to the corresponding gainAtten module. The gainAtten module provides ±12dB of gain/boost based on the slider switch positions, which are decoded form the TLC541 serial bus by the interfaceDriver module. All of the outputs from each gainAtten module, which are still individual sections of the full spectrum, are recombined into a broad band signal in the summer9x2ch module. the resulting signal is sent back to the audDriver module, which places the output audio samples on the AC-Link serial bus, to be sent back to the LM4550 and ultimately to the line-out port.

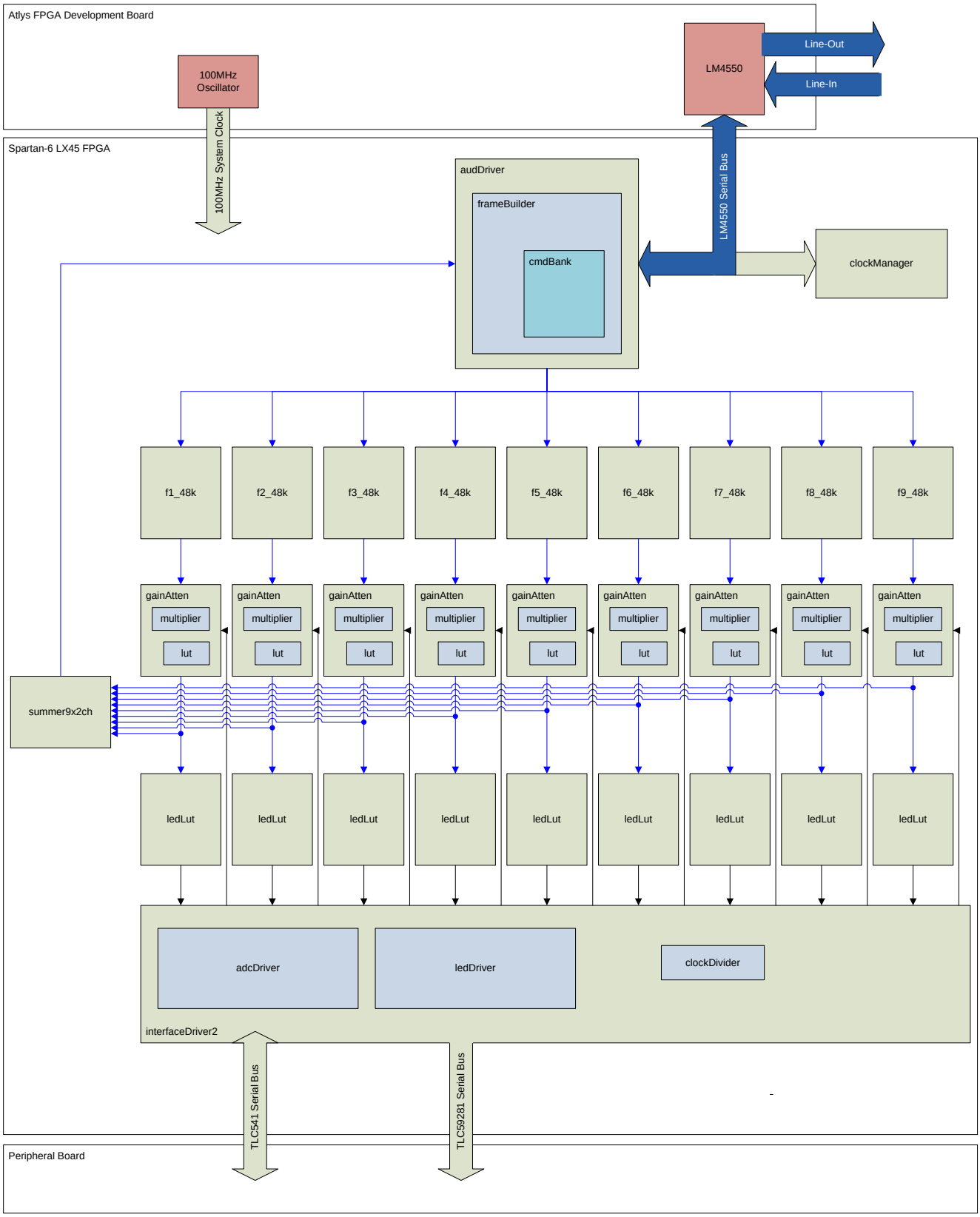

Figure 28: Top level block diagram of firmware

Top level timing diagram

Figure 29 shows the timing diagram that a right channel audio signal would follow, as it is filtered by FIR filter #1. The entire process is synchronous to the AUDSYNC signal, which has a frequency of 48KHz, which is the sampling rate. Once the audDriver module has extracted a new audio sample from the AC-Link bus, it asserts the codecDataOk signal, which causes the filters (in the case of this example, filter 1) to begin processing. When the filter is done processing, it's output is placed on the signal pregam1_r, which is the input to the gain/attenuation module, which provides gain or attenuation to the signal. The signal is then summed together with the signals from all other frequency bands, and is placed on the audioOut_r signal. On the next rising edge of AUDSYNC, the audDriver module latches in the value of audioOut_r, sends it to the LM4550 over the AC-Link bus, and the process begins all over again.

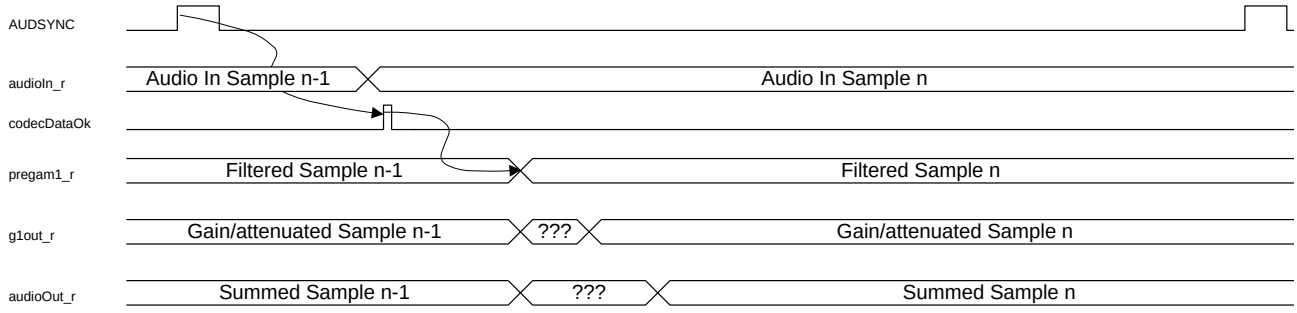

Figure 29: Top level timing diagram

System Level Clock Distribution

The entire system uses the 12.288MHz signal BITCLK, which is generated from the LM4550, as a clock. This ensures that both the LM4550 and the firmware on the FPGA are running on exactly the same clock. Early implementations used a divided-down version of the 100MHz system clock to run the FPGA firmware, and there were problems with phase drift between the LM4550 clock and the FPGA clock. The problem with this system, however, is that the 12.288MHz clock is not available when the LM4550 is in the reset state. Thus, it cannot be used as a counter to create a long reset signal. To get around this, the 100MHz system clock is used to drive a counter, which generates a long reset at system startup.

Firmware Modules

audioProcessor

The audioProcessor module is the top level VHDL module, and contains all of the other modules in this design. It's black box diagram is shown in Figure 30.

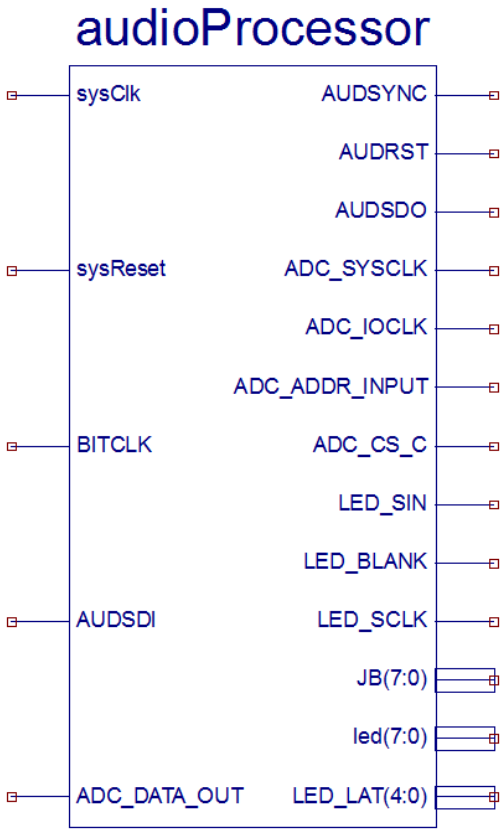

Figure 30: audioProcessor black box

ledDriver

The ledDriver module, shown in Figure 31, communicates with each TLC59281 LED driver IC using the timing diagram shown in Figure 32, which was adapted from the TLC59281 datasheet from Texas Instruments. It is controlled by the interfaceDriver2 module.

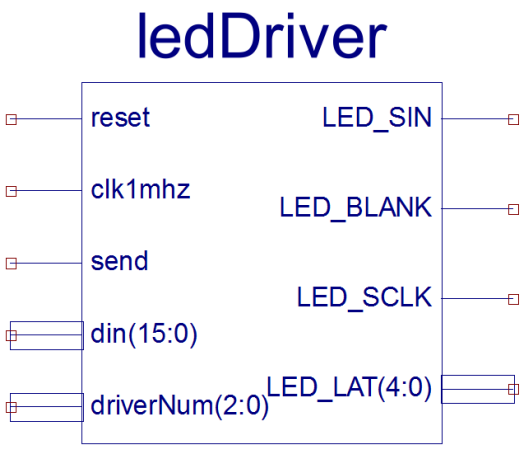

Figure 31: ledDriver black box

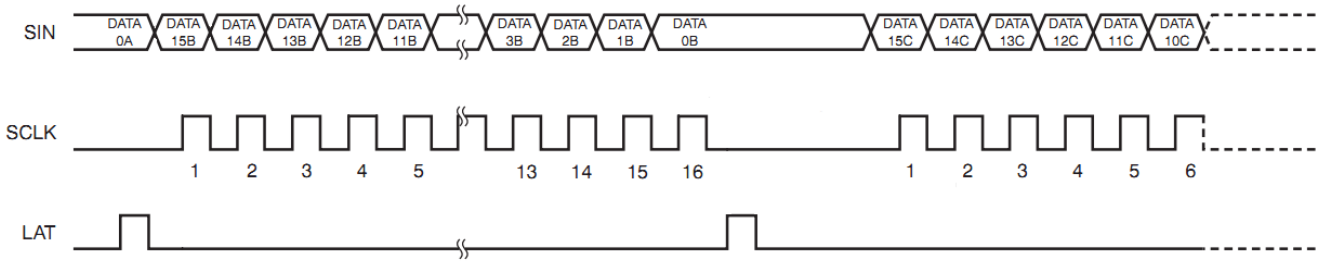

Figure 32: Timing diagram for the TLC59281

adcDriver

the adcDriver module, shown in Figure 33, communicates with the TLC541 ADC, using the timing diagram shown in Figure 34, which is adopted from the TLC541 datasheet from Texas Instruments. When the CS line is asserted low, data transmission begins, and 16 clock cycles are sent on the SCLK line. Every communication consists of sending a 4-bit address, and receiving an 8-bit PCM value. The received PCM value corresponds to the ADC channel belonging to the address sent during the previous communication.

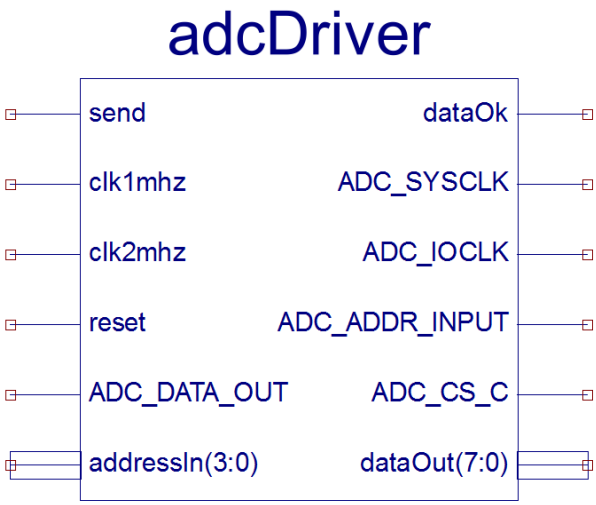

Figure 33: adcDriver black box

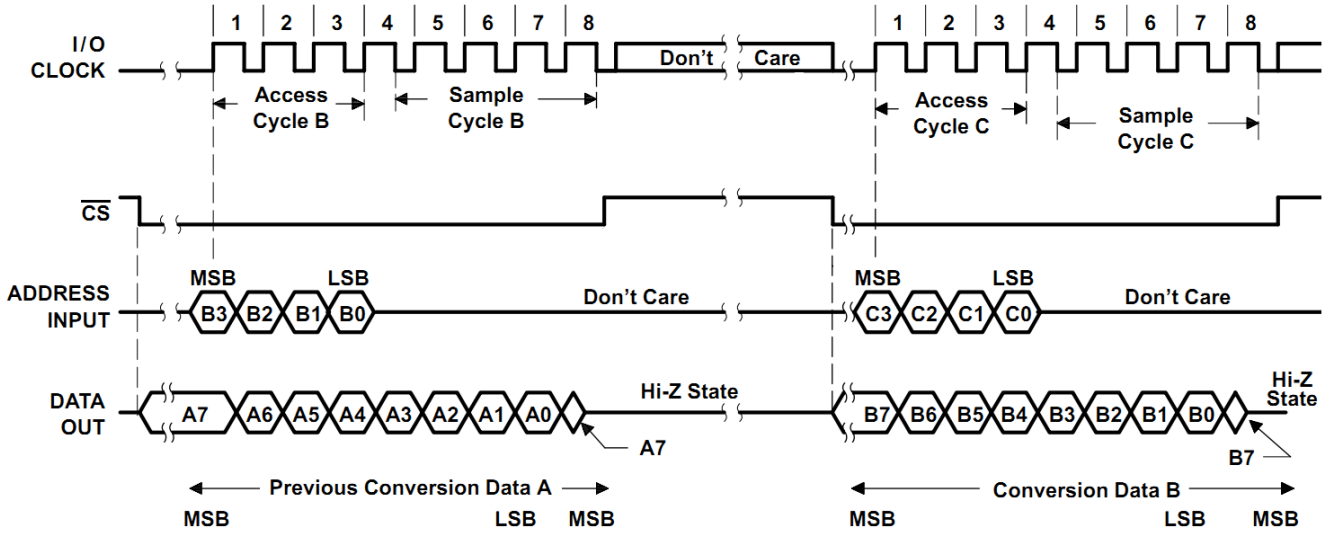

Figure 34: Timing diagram for the TLC541

clockDivider

The clockDivider module, shown in Figure 35, is a simple clock divider that is used in the interfaceDriver2 module. It is necessary because the slowest clock that the Xilinx Clock Management Tile can generate is not low enough for the needs of the interfaceDriver2 module [13]. The clockDivider module takes in a 2.048MHz clock, and divides it down to a 1.024MHz clock, and a 4KHz clock.

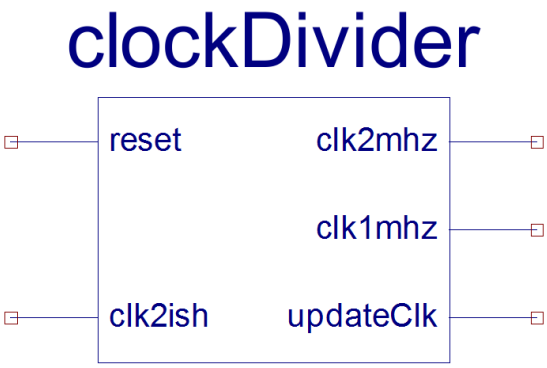

Figure 35: clockDivider black box

interfaceDriver2

The interfaceDriver2 module, shown in Figure 36, controls the adcDriver and ledDriver modules. The adcDriver module can only communicate with one ADC channel at a time, and the ledDriver module can only communicate with one TLC59281 at a time. The interfaceDriver2 module employs a time-division multiplexing scheme to interface with each TLC59281 and ADC channel every 4KHz.

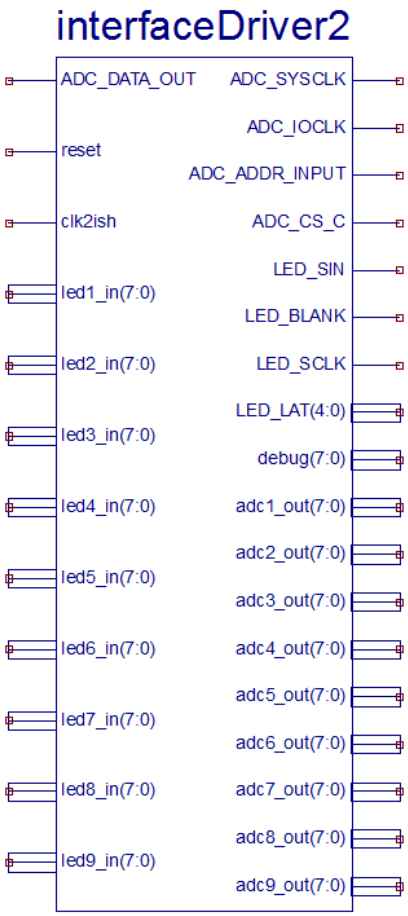

Figure 36: interfaceDriver2 black box

cmdBank

The cmdBank module, shown in Figure 37, is a basic lookup table, which stores configuration data to configure the LM4550. This data is used by the frameBuilder module, to assemble data packets for the AC-Link serial bus.

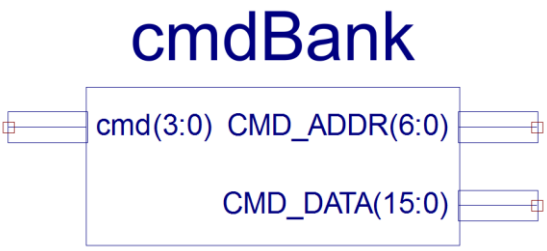

Figure 37: cmdBank black box

frameBuilder

The frameBuilder module, shown in Figure 38, uses data from the cmdBank module to assemble data packets to place on the AC-Link serial bus.

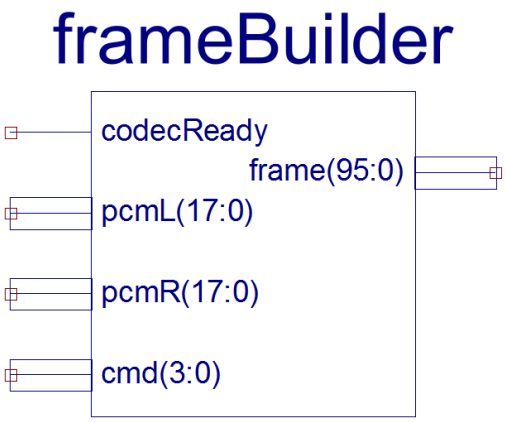

Figure 38: frameBuilder black box

codecDriver

The codecDriver module, shown in Figure 39, is a device driver for the LM4550 audio codec. It communicates with the LM4550 using the timing diagram shown in Figure 40. The communication protocol is the AC-Link serial interface protocol, defined by Intel [14]. It is a bidirectional serial bus, which simultaneously sends and receives configuration information and audio input/output data. Each data packet is made up of 13 time slots. 12 of the time slots are 20 clock cycles long, and the first time slot is 16 clock cycles long. These data packets are assembled by the frameBuilder module.

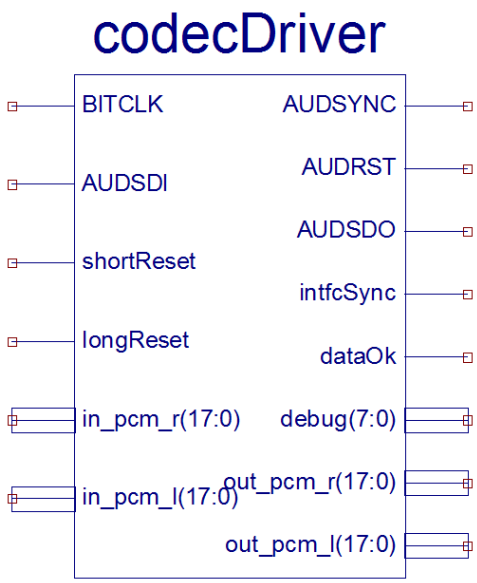

Figure 39: codecDriver black box

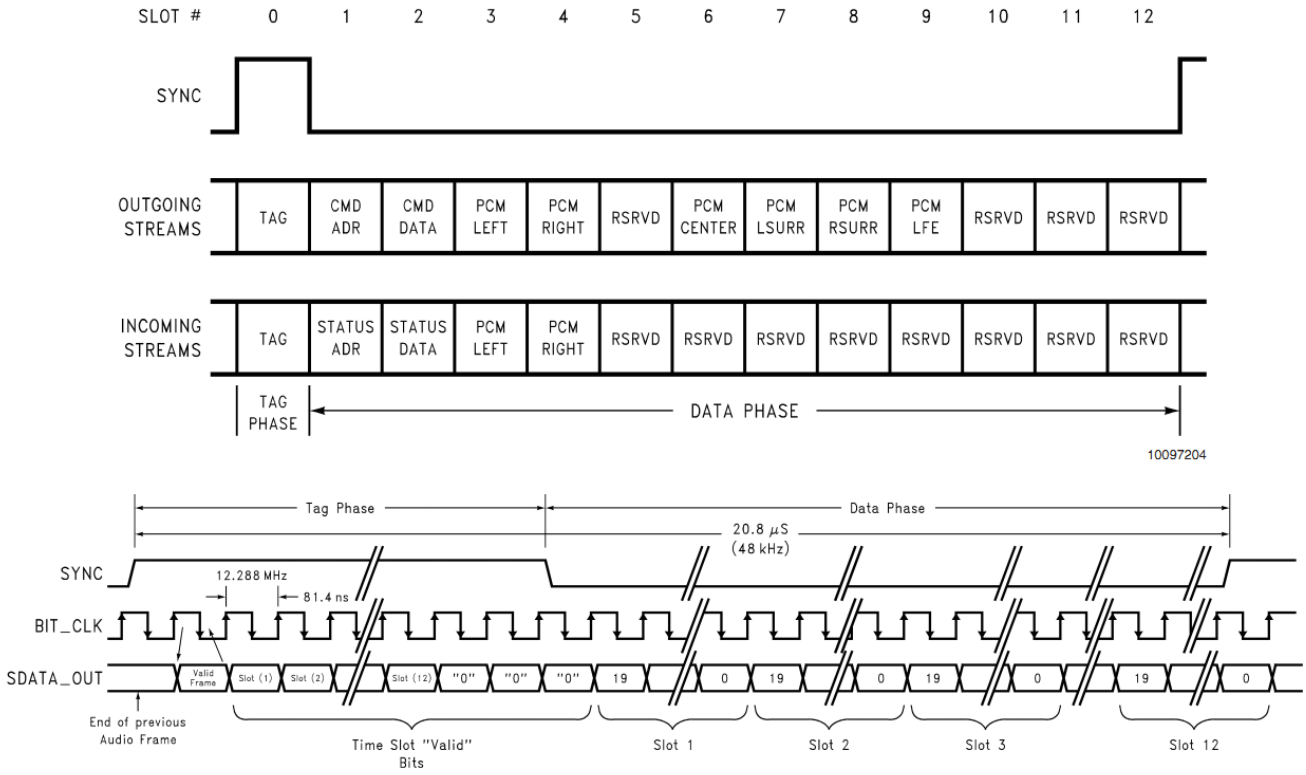

Figure 40: Timing diagram for LM4550

The LM4550 has an analog signal chain with adjustable gain/attenuation amplifiers and analog multiplexers to direct the signal flow through the device. Figure 41 shows the block diagram of the LM4550, which is adapted from the datasheet from National Semiconductor. The signal path used is shown in yellow, and registers programmed at configuration have their addresses shown in red.

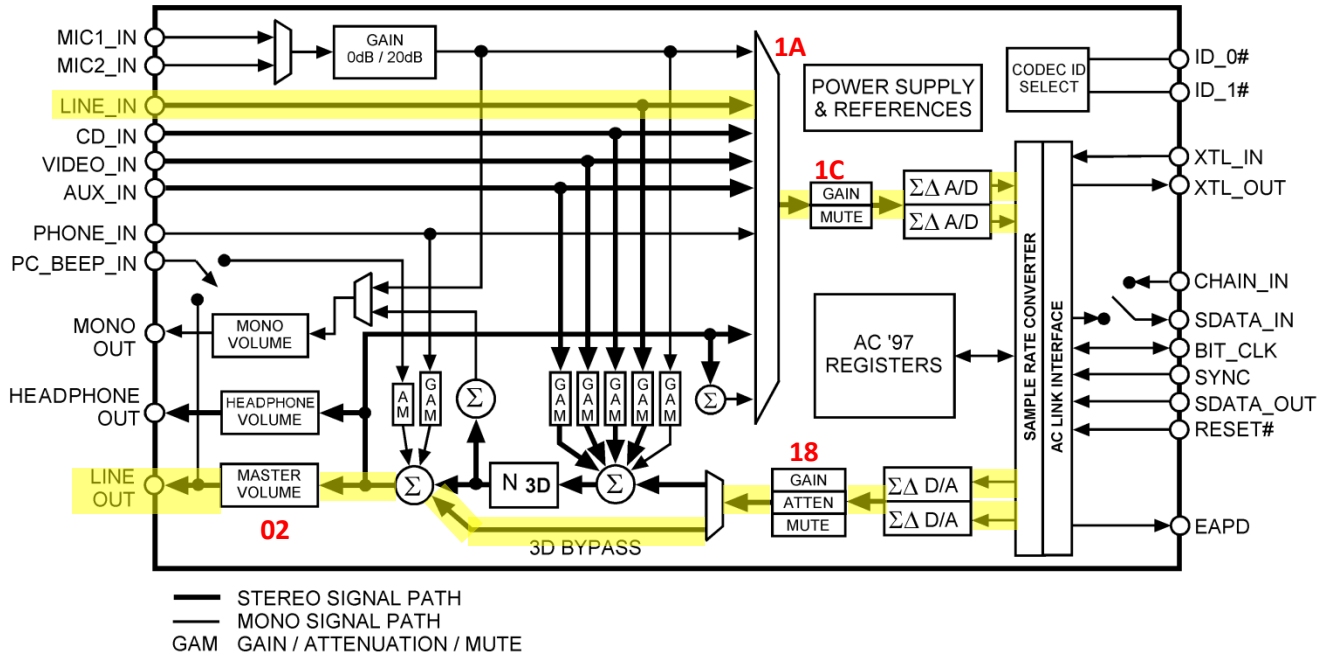

Figure 41: Block diagram of LM4550

summer9x2ch

The summer9x2ch module, shown in Figure 42, is a 9-input, two channel, twos-compliment summation block. Its purpose is to recombine all of the signals after they have been separated into individual frequency bands and adjusted for gain or attenuation.

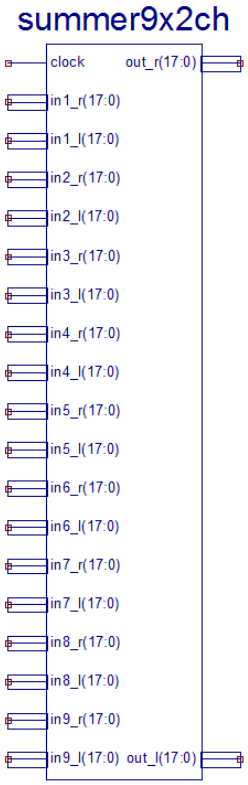

Figure 42: summer9x2ch black box

gainAtten

the gainAtten module, shown in Figure 43, is a lookup table implementation of scaled integer multiplication and division. In order for the gainAtten module to be able to provide both gain and attenuation, it must be able to perform multiplication and division operations, and these operations must be preformed entirely with integer math. The initial problem with this is that multiplying/dividing by integers has a finite resolution. For example, you can only multiply and divide by 1, 2, 3, etc. Multiplication by 1.2, for example, is not possible. In order to achieve a finer resolution, the product of the multiplication/division containing the decimal value is scaled by a power of two, so that it can be represented as an integer. It is then multiplied by the data signal, and the result is divided by the same scaling factor used before. This process, shown in Figure 44, allows a single multiplier module to perform both multiplication and division. The lut module, which is a lookup table, performs both scale factor multiplication, and conversion between a linear input scale and logarithmic input scale. More information on this module is given in the following lut section, along with rounding error analysis.

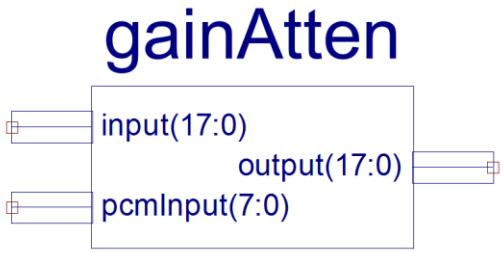

Figure 43: gainAtten black box

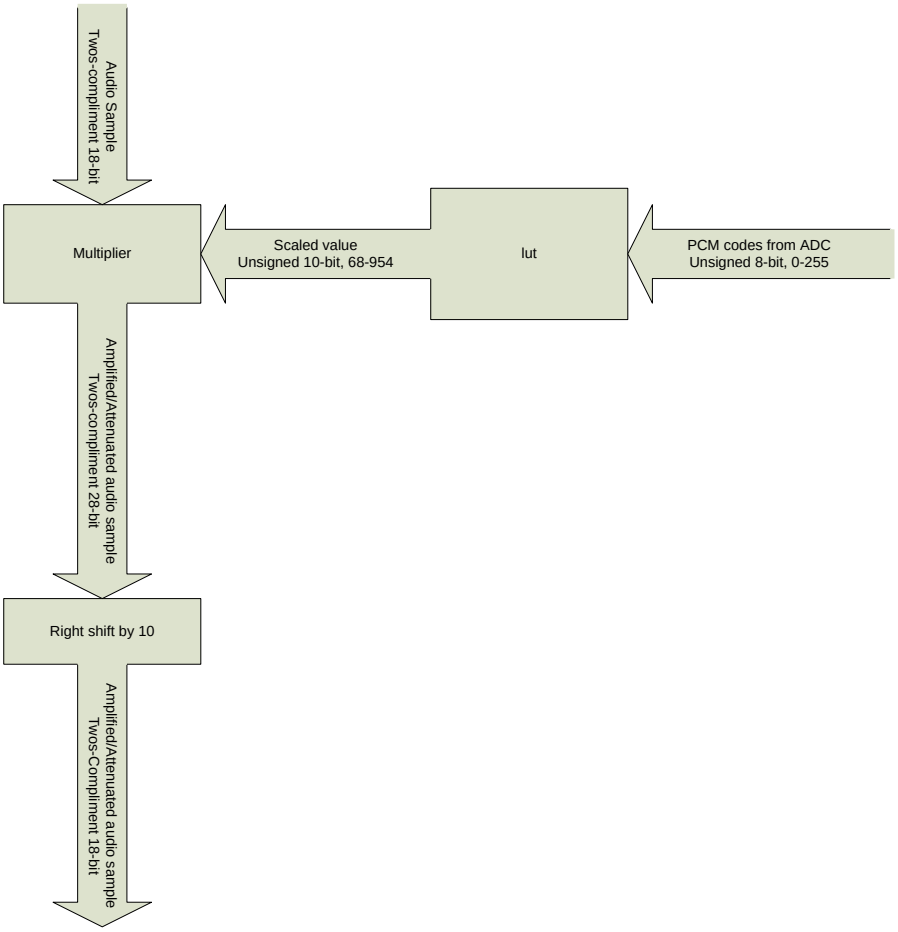

Figure 44: Scaled integer multiplication process

lut

The lut module, shown in Figure 45, is a lookup table which converts 8-bit PCM values from the ADCs to 10-bit values which are both multiplied by a scale factor and converted to a logarithmic scale. This lookup table is produced by the script divisionError.m, which is listed in Appendix B. The transfer function for the lut module is shown in Figure 46, which is produced from divisionError.m. Figure 47 is also produced from divisionError.m, and shows the maximum error when the multiplication/division operation is preformed, for all input PCM values. The percent error for very small values is high because these values are very close to zero. However, the number of high error values is very small compared to the input range of values. Figure 48 is similar to Figure 47, and shows all possible error values. Once again, the largest error occurs at small PCM values, but most of the possible values have small error associated with them.

Figure 45: lut black box

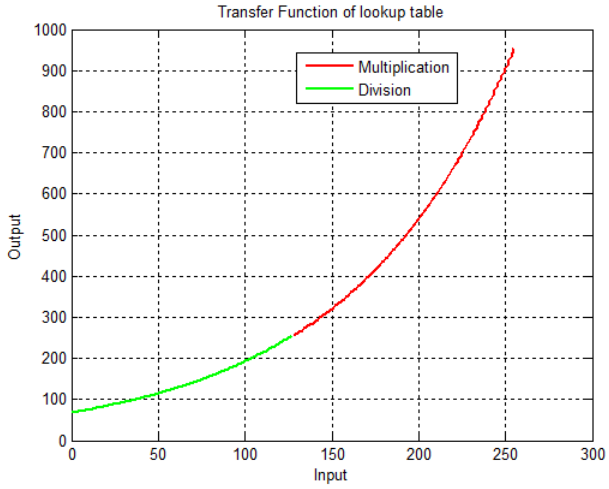

Figure 46: lut module transfer function

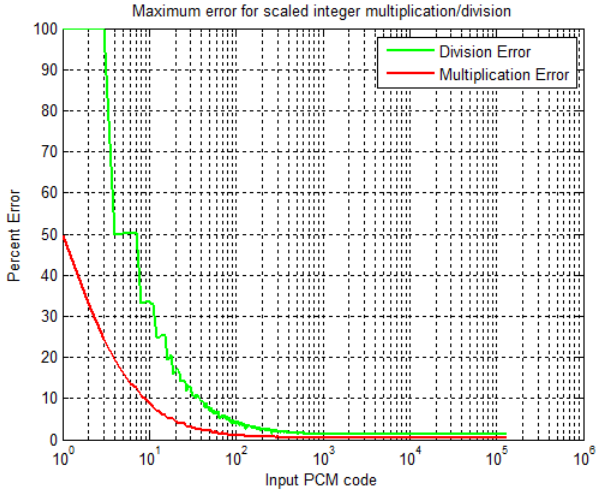

Figure 47: scaled integer multiplication/division error analysis

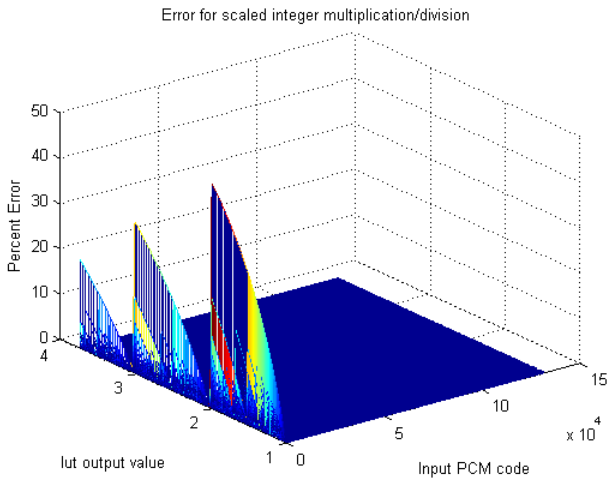

Figure 48: error for scaled integer multiplication/division vs. two inputs

ledLut

the ledLut module, shown in Figure 49, is a lookup table to convert the twos-compliment PCM codes of each frequency band to an encoding scheme that can be displayed on the LED banks, representing the approximate magnitude of the PCM value. It essentially takes the absolute value of the signed PCM code, and assigns one of eight amplitude values, which are displayed on the LED banks.

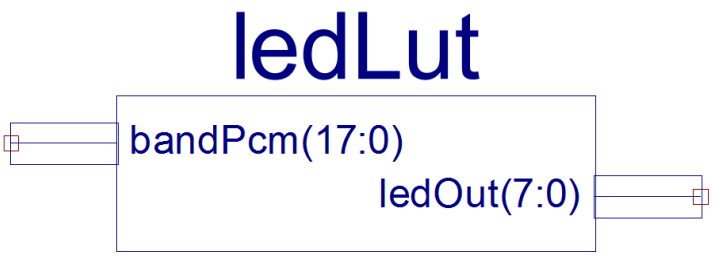

Figure 49: ledLut black box

multiplier

The multiplier module, shown in Figure 50, is generated using the Xilinx Multiplier IP CORE. It uses combinational logic to produce the product of one signed and one unsigned bus. The implementation details are listed in Table 8. Each multiplier uses approximately 188 LUT6 blocks on the FPGA.

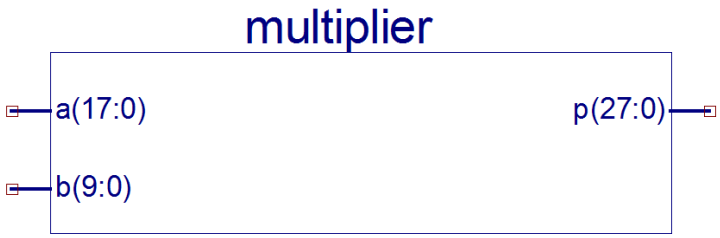

Figure 50: multiplier black box

Table 8: multiplier IP CORE configuration information

Specification	*Value*
IP CORE version	Xilinx.com:ip:mult_gen:11.2
Multiplier Type	Parallel Multiplier
Port A Data Type	Signed
Port A Data Width	18
Port B Data Type	Unsigned
Port B Data Width	10
Multiplier Construction	Use LUTs
Optimization Options	Speed Optimized
Use Custom Output Width	Unchecked
Pipeline Stages	0

clockManager

The clockManager module, shown in Figure 51, is an implementation of the Xilinx Clocking Wizard IP CORE. It is used to convert the BITCLK signal, from the LM4550, from 12.288MHz to 24.576MHz and 2.048MHz. Important configuration information is shown in Table 9. The Clocking Wizard IP CORE is used because it can take advantage of FPGA hardware resources to multiply and divide clocks with minimum clock jitter, making it a better alternative to using combinational logic.

Figure 51: clockManager black box

Table 9: clockManager IP CORE configuration information

Specification	*Value*
IP CORE version	Xilinx.com:ip:clk_wiz_3.3
Frequency synthesis	Checked
Phase Alignment	Checked
Minimize Power	Unchecked
Dynamic Phase Shift	Unchecked
Mode	Auto Selection
Jitter Optimization	Balanced
Input Jitter Unit	UI
Input Clock Input Frequency	12.288MHz
Input Jitter	0.010
Source	Single ended clock capable pin
CLK_OUT1 Output Frequency	12.288MHz
CLK_OUT1 Phase	0.000
CLK_OUT1 Duty Cycle (%)	50.000
CLK_OUT1 Drives	BUFG
CLK_OUT2 Output Frequency	24.576MHz

CLK_OUT2 Phase	0.000
CLK_OUT2 Duty Cycle (%)	50.000
CLK_OUT2 Drives	BUFG
CLK_OUT3 Output Frequency	2.048MHz
CLK_OUT3 Phase	0.000
CLK_OUT3 Duty Cycle (%)	50.000
CLK_OUT3 Drives	BUFG
Reset	Checked
Locked	Unchecked
Input_clk_stopped	Unchecked
Status	Unchecked
Clk_valid	Checked
Clock feedback source	Automatic control on-chip
Allow override mode	Unchecked

f1_48K through f9_48K

The FIR filters represent the heart of the signal processing chain. Nine FIR filters are used in parallel, and are named f1-f9. All of the filters were designed using FDATool, employing the equiripple method. The filter coefficients created using FDATool were exported both to Xilinx .coe files, and to the Matlab file filterCoe.mat. The .coe files were used by the FIR Compiler 5.0 Xilinx IP CORE generator to create synthesizable FIR filters. Each filter has the same inputs and outputs, shown in Figure 52. The .coe files used to generate the filters are listed in Table 11, and the FIR Compiler 5.0 configuration settings are listed in Table 10. Figure 53 was generated from the Matlab script filterPlotter.m, and shows the individual magnitude response of each FIR filter, and the combined response of all of the filters together.

The FIR Compiler is used here because it takes advantage of the DSP48A1 hardware slices contained on the Spartan-6 FPGA [12]. This allows efficient filter computation.

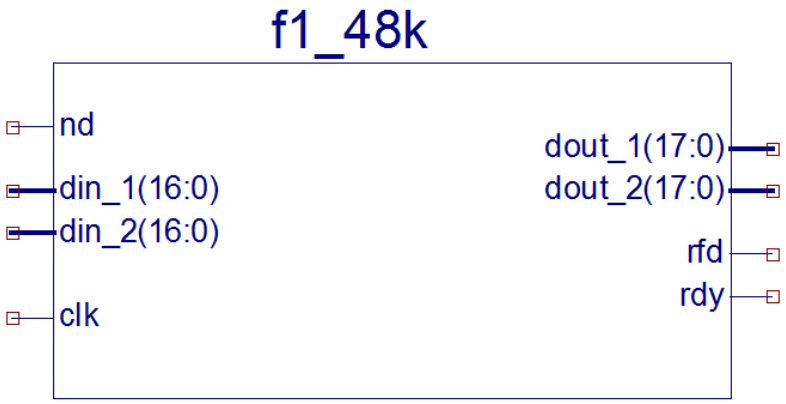

Figure 52: f1-48k through f9-48k black box

Table 10: FIR Filter specifications

Specification	***Value***
IP CORE version	Xilinx.com:ip:fir_complier:5.0
Select source	COE File

Coefficient file	Various, see Table 11
Number of coefficient sets	1
Filter type	Single rate
Number of channels	1
Select format	Frequency specification
Input sampling frequency	0.048MHz
Clock frequency	24.576 for f1-f3, 12.288 for f4-f9
Filter architecture	Systolic multiply accumulate
Use reloadable coefficients	Unchecked
Coefficient structure	Inferred
Coefficient type	Signed
Coefficient width	16
Number of data paths	2
Input data type	Signed
Input data width	17
Input data fractional bits	0
Output rounding mode	Non-symmetric rounding down
Output width	18
Registered output	checked
Optimization goal	Area
SCLR	Unchecked
ND	Checked
CE	unchecked
Data buffer type, coefficient buffer type, input buffer type, output buffer type, performance for other storage	automatic

Table 11: FIR filter coefficient naming scheme

Filter Name	***Coefficient file***	***Number of Coefficients***
f1_48K	f1_450pt.coe	451
f2_48K	f2_460pt.coe	461
f3_48K	f3_400pt.coe	401
f4_48K	f4_200pts.coe	201
f5_48K	f5_101pt.coe	101
f6_48K	f6_101pt.coe	101
f7_48K	f7_101pt.coe	101
f8_48K	f8_101pt.coe	101
f9_48K	f9_101pt.coe	101

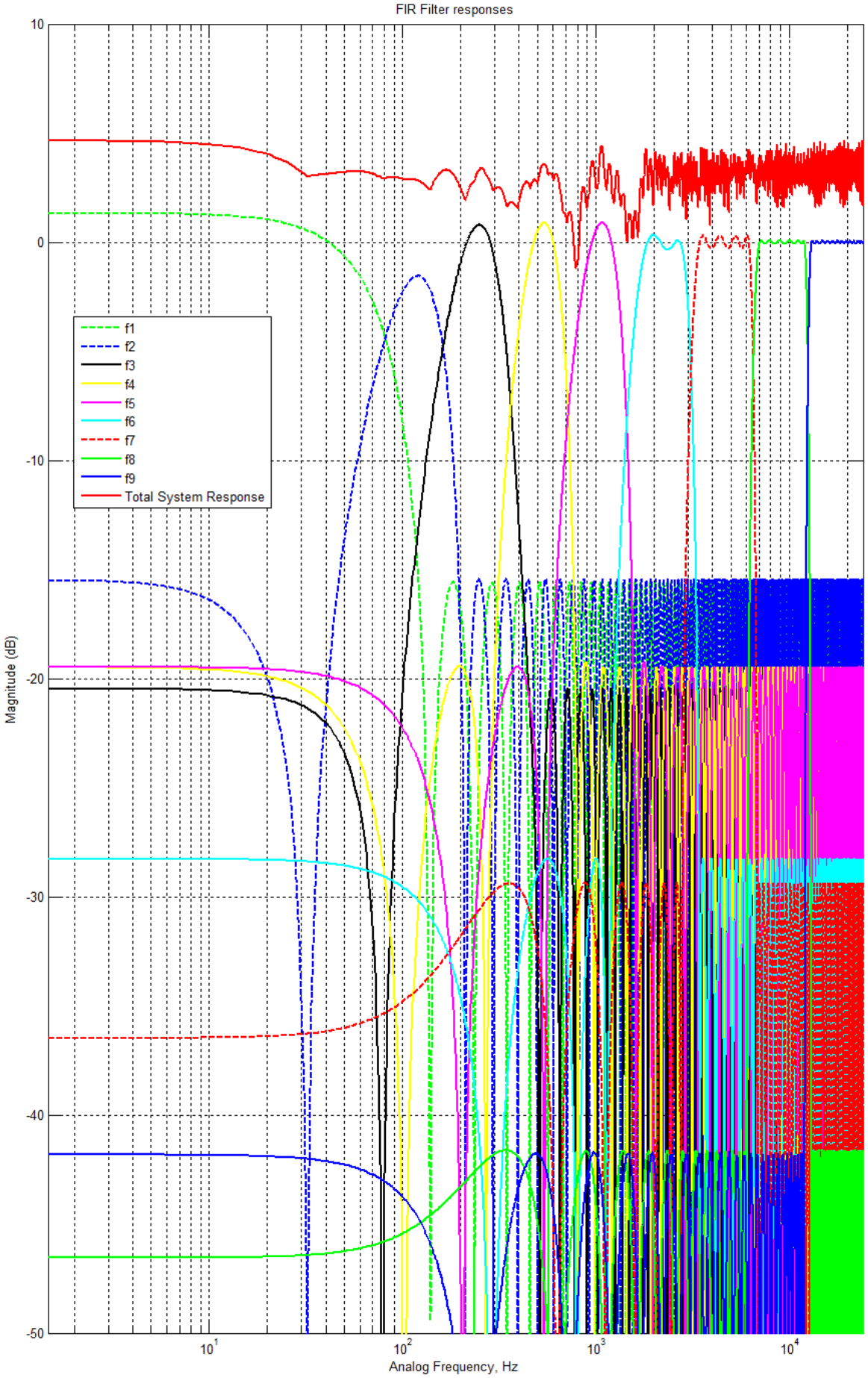

Figure 53: Magnitude response of FIR filters

Appendix A: VHDL Source Code

audioProcessor.vhd

```
1  ----------------------------------------------------------------------------
2  -- audioProcessor.vhd ---------------------------------------------------------
3  -- Author: Anthony Giardina                                                --
4  -- For: Graphic Audio EQ Senior Project                                    --
5  -- Last Modified: 5/29/2012                                                --
6  -- Purpose: This is the top level module for the audio equalizer project. It --
7  --  connects all other modules in the project togheter, and interfaces      --
8  --  all of the nessecary signals.                                          --
9  ----------------------------------------------------------------------------
10 ----------------------------------------------------------------------------
11
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 use IEEE.NUMERIC_STD.ALL;
15 use Ieee.std_logic_unsigned.all;
16
17 entity audioProcessor is
18    Port (
19         -- System clock from onboard oscillator, 100MHz.
20         sysClk : in  STD_LOGIC;
21         -- System reset from switch. Active low.
22         sysReset: in STD_LOGIC;
23         -- Debug connector on Atlys board
24         JB : out std_logic_vector (7 downto 0);
25         -- Debug LEDs on Atlys board
26         led: out std_logic_vector(7 downto 0);
27
28         -- Signals to LM4550
29         BITCLK: in std_logic;
30         AUDSYNC: out std_logic;
31         AUDRST: out std_logic;
32         AUDSDI: in std_logic;
33         AUDSDO: out std_logic;
34
35         -- Signals to peripheral board
36         ADC_DATA_OUT : in std_logic;
37         ADC_SYSCLK : out std_logic;
38         ADC_IOCLK : out std_logic;
39         ADC_ADDR_INPUT : out std_logic;
40         ADC_CS_C : out std_logic;
41         LED_SIN : out std_logic;
42         LED_BLANK : out std_logic;
43         LED_SCLK : out std_logic;
44         LED_LAT : out std_logic_vector (4 downto 0)
45     );
46 end audioProcessor;
47
48 architecture Behavioral of audioProcessor is
49     -- resetGenerator is a counter used for generating a long reset
50     -- at system startup, or when the reset button is pressed.
51     signal resetGenerator: std_logic_vector (23 downto 0);
52     -- Reset signals, which are toggled at system startup:
53     signal shortReset: std_logic;
54     signal longReset: std_logic;
55     signal notSysReset: std_logic;
56     -- CLK_VALID signal indicates that the clock management tile is
57     -- functioning properly.
58     signal CLK_VALID: std_logic;
59     -- System clocks:
60     signal clk12_288M: std_logic;
61     signal clk12_288M_buf: std_logic;
62     signal clk2_048M: std_logic;
63     signal clk24_576M: std_logic;
64     -- Debug signal between the audioProcessor and interface modules:
65     signal intDebug: std_logic_vector (7 downto 0);
66     -- Debug signal between the audioProcessor and the audio module:
```

```
67      signal audDebug : std_logic_vector ( 7 downto 0);
68      -- Buffer signals for internal use
69      signal bufAUDSYNC: std_logic;
70      signal bufAUDSDO: std_logic;
71      signal bufAUDRST: std_logic;
72      -- Syncronization signals:
73      signal codecDataOk: std_logic;
74      signal codecIntfcSync: std_logic;
75      -- PCM signals, left and right, coming from the line-in port
76      signal audioIn_r : std_logic_vector(17 downto 0);
77      signal audioIn_l : std_logic_vector(17 downto 0);
78      -- PCM signals, left and right, going to the line-out port.
79      signal audioOut_r : std_logic_vector(17 downto 0);
80      signal audioOut_l : std_logic_vector(17 downto 0);
81      -- Signals that connect the outputs from the gain/atten modules
82      -- to the summation module.
83      signal g1out_r : std_logic_vector(17 downto 0);
84      signal g1out_l : std_logic_vector(17 downto 0);
85      signal g2out_r : std_logic_vector(17 downto 0);
86      signal g2out_l : std_logic_vector(17 downto 0);
87      signal g3out_r : std_logic_vector(17 downto 0);
88      signal g3out_l : std_logic_vector(17 downto 0);
89      signal g4out_r : std_logic_vector(17 downto 0);
90      signal g4out_l : std_logic_vector(17 downto 0);
91      signal g5out_r : std_logic_vector(17 downto 0);
92      signal g5out_l : std_logic_vector(17 downto 0);
93      signal g6out_r : std_logic_vector(17 downto 0);
94      signal g6out_l : std_logic_vector(17 downto 0);
95      signal g7out_r : std_logic_vector(17 downto 0);
96      signal g7out_l : std_logic_vector(17 downto 0);
97      signal g8out_r : std_logic_vector(17 downto 0);
98      signal g8out_l : std_logic_vector(17 downto 0);
99      signal g9out_r : std_logic_vector(17 downto 0);
100     signal g9out_l : std_logic_vector(17 downto 0);
101     -- Ready, Ready For Data, and Need Data signals. These are useful
102     -- for debugging filter timing, but they are not nessesary for
103     -- the operation of the system.
104     signal rfd1 :  STD_LOGIC;
105     signal rdy1 :  STD_LOGIC;
106     signal nd1 : std_logic;
107     signal rfd2 :  STD_LOGIC;
108     signal rdy2 :  STD_LOGIC;
109     signal nd2 : std_logic;
110     signal rfd3 :  STD_LOGIC;
111     signal rdy3 :  STD_LOGIC;
112     signal nd3 : std_logic;
113     signal rfd4 :  STD_LOGIC;
114     signal rdy4 :  STD_LOGIC;
115     signal nd4 : std_logic;
116     signal rfd5 :  STD_LOGIC;
117     signal rdy5 :  STD_LOGIC;
118     signal nd5 : std_logic;
119     signal rfd6 :  STD_LOGIC;
120     signal rdy6 :  STD_LOGIC;
121     signal nd6 : std_logic;
122     signal rfd7 :  STD_LOGIC;
123     signal rdy7 :  STD_LOGIC;
124     signal nd7 : std_logic;
125     signal rfd8 :  STD_LOGIC;
126     signal rdy8 :  STD_LOGIC;
127     signal nd8 : std_logic;
128     signal rfd9 :  STD_LOGIC;
129     signal rdy9 :  STD_LOGIC;
130     signal nd9 : std_logic;
131     -- LED output signals. Each signal corrosponds to a bank of LEDs:
132     signal led1 : std_logic_vector (7 downto 0);
133     signal led2 : std_logic_vector (7 downto 0);
134     signal led3 : std_logic_vector (7 downto 0);
135     signal led4 : std_logic_vector (7 downto 0);
```

```
    signal led5 : std_logic_vector (7 downto 0);
    signal led6 : std_logic_vector (7 downto 0);
    signal led7 : std_logic_vector (7 downto 0);
    signal led8 : std_logic_vector (7 downto 0);
    signal led9 : std_logic_vector (7 downto 0);
    -- ADC input signals. Each signal corrosponds to a slide pot:
    signal adc1 : std_logic_vector (7 downto 0);
    signal adc2 : std_logic_vector (7 downto 0);
    signal adc3 : std_logic_vector (7 downto 0);
    signal adc4 : std_logic_vector (7 downto 0);
    signal adc5 : std_logic_vector (7 downto 0);
    signal adc6 : std_logic_vector (7 downto 0);
    signal adc7 : std_logic_vector (7 downto 0);
    signal adc8 : std_logic_vector (7 downto 0);
    signal adc9 : std_logic_vector (7 downto 0);
    -- These signals are used to connect the FIR filter outputs to the
    -- gain/atten module inputs.
    signal pregam9_l: std_logic_vector(17 downto 0);
    signal pregam9_r: std_logic_vector(17 downto 0);
    signal pregam8_l: std_logic_vector(17 downto 0);
    signal pregam8_r: std_logic_vector(17 downto 0);
    signal pregam7_l: std_logic_vector(17 downto 0);
    signal pregam7_r: std_logic_vector(17 downto 0);
    signal pregam6_l: std_logic_vector(17 downto 0);
    signal pregam6_r: std_logic_vector(17 downto 0);
    signal pregam5_l: std_logic_vector(17 downto 0);
    signal pregam5_r: std_logic_vector(17 downto 0);
    signal pregam4_l: std_logic_vector(17 downto 0);
    signal pregam4_r: std_logic_vector(17 downto 0);
    signal pregam3_l: std_logic_vector(17 downto 0);
    signal pregam3_r: std_logic_vector(17 downto 0);
    signal pregam2_l: std_logic_vector(17 downto 0);
    signal pregam2_r: std_logic_vector(17 downto 0);
    signal pregam1_l: std_logic_vector(17 downto 0);
    signal pregam1_r: std_logic_vector(17 downto 0);

    -- The rest of the architecture statement is simply components. Their
   -- function is described in further detail where they are instantiated.
    COMPONENT ledLut
    PORT(
        bandPcm : IN std_logic_vector(17 downto 0);
        ledOut : OUT std_logic_vector(7 downto 0)
        );
    END COMPONENT;

    COMPONENT interfaceDriver2
    PORT(
        ADC_DATA_OUT : IN std_logic;
        reset : IN std_logic;
        clk2ish : IN std_logic;
        debug: out std_logic_vector (7 downto 0);
        led1_in : IN std_logic_vector(7 downto 0);
        led2_in : IN std_logic_vector(7 downto 0);
        led3_in : IN std_logic_vector(7 downto 0);
        led4_in : IN std_logic_vector(7 downto 0);
        led5_in : IN std_logic_vector(7 downto 0);
        led6_in : IN std_logic_vector(7 downto 0);
        led7_in : IN std_logic_vector(7 downto 0);
        led8_in : IN std_logic_vector(7 downto 0);
        led9_in : IN std_logic_vector(7 downto 0);
        ADC_SYSCLK : OUT std_logic;
        ADC_IOCLK : OUT std_logic;
        ADC_ADDR_INPUT : OUT std_logic;
        ADC_CS_C : OUT std_logic;
        LED_SIN : OUT std_logic;
        LED_BLANK : OUT std_logic;
        LED_SCLK : OUT std_logic;
        LED_LAT : OUT std_logic_vector(4 downto 0);
        adc1_out : OUT std_logic_vector(7 downto 0);
```

```
205         adc2_out : OUT std_logic_vector(7 downto 0);
206         adc3_out : OUT std_logic_vector(7 downto 0);
207         adc4_out : OUT std_logic_vector(7 downto 0);
208         adc5_out : OUT std_logic_vector(7 downto 0);
209         adc6_out : OUT std_logic_vector(7 downto 0);
210         adc7_out : OUT std_logic_vector(7 downto 0);
211         adc8_out : OUT std_logic_vector(7 downto 0);
212         adc9_out : OUT std_logic_vector(7 downto 0)
213         );
214     END COMPONENT;
215 
216     COMPONENT codecDriver
217     PORT(
218         BITCLK : IN std_logic;
219         AUDSDI : IN std_logic;
220         in_pcm_r : IN std_logic_vector(17 downto 0);
221         in_pcm_l : IN std_logic_vector(17 downto 0);
222         shortReset : in std_logic;
223         longReset: in std_logic;
224         debug : out std_logic_vector (7 downto 0);
225         AUDSYNC : OUT std_logic;
226         AUDRST : OUT std_logic;
227         AUDSDO : OUT std_logic;
228         out_pcm_r : OUT std_logic_vector(17 downto 0);
229         out_pcm_l : OUT std_logic_vector(17 downto 0);
230         intfcSync : OUT std_logic;
231         dataOk : OUT std_logic
232         );
233     END COMPONENT;
234 
235     component clockManager
236     port
237      (-- Clock in ports
238       CLK_IN1           : in     std_logic;
239       -- Clock out ports
240       CLK_OUT1          : out    std_logic;
241       CLK_OUT2          : out    std_logic;
242       CLK_OUT3          : out    std_logic;
243       RESET             : in     std_logic;
244       CLK_VALID         : out    std_logic
245      );
246     end component;
247 
248     COMPONENT summer9x2ch
249     PORT(
250         clock : IN std_logic;
251         in1_r : IN std_logic_vector(17 downto 0);
252         in1_l : IN std_logic_vector(17 downto 0);
253         in2_r : IN std_logic_vector(17 downto 0);
254         in2_l : IN std_logic_vector(17 downto 0);
255         in3_r : IN std_logic_vector(17 downto 0);
256         in3_l : IN std_logic_vector(17 downto 0);
257         in4_r : IN std_logic_vector(17 downto 0);
258         in4_l : IN std_logic_vector(17 downto 0);
259         in5_r : IN std_logic_vector(17 downto 0);
260         in5_l : IN std_logic_vector(17 downto 0);
261         in6_r : IN std_logic_vector(17 downto 0);
262         in6_l : IN std_logic_vector(17 downto 0);
263         in7_r : IN std_logic_vector(17 downto 0);
264         in7_l : IN std_logic_vector(17 downto 0);
265         in8_r : IN std_logic_vector(17 downto 0);
266         in8_l : IN std_logic_vector(17 downto 0);
267         in9_r : IN std_logic_vector(17 downto 0);
268         in9_l : IN std_logic_vector(17 downto 0);
269         out_r : OUT std_logic_vector(17 downto 0);
270         out_l : OUT std_logic_vector(17 downto 0)
271         );
272     END COMPONENT;
273 
```

```
274     component f1_48k
275         port (
276         clk: in std_logic;
277         nd: in std_logic;
278         rfd: out std_logic;
279         rdy: out std_logic;
280         din_1: in std_logic_vector(16 downto 0);
281         din_2: in std_logic_vector(16 downto 0);
282         dout_1: out std_logic_vector(17 downto 0);
283         dout_2: out std_logic_vector(17 downto 0));
284     end component;
285     --
286     component f2_48k
287         port (
288         clk: in std_logic;
289         nd: in std_logic;
290         rfd: out std_logic;
291         rdy: out std_logic;
292         din_1: in std_logic_vector(16 downto 0);
293         din_2: in std_logic_vector(16 downto 0);
294         dout_1: out std_logic_vector(17 downto 0);
295         dout_2: out std_logic_vector(17 downto 0));
296     end component;
297
298     component f3_48k
299         port (
300         clk: in std_logic;
301         nd: in std_logic;
302         rfd: out std_logic;
303         rdy: out std_logic;
304         din_1: in std_logic_vector(16 downto 0);
305         din_2: in std_logic_vector(16 downto 0);
306         dout_1: out std_logic_vector(17 downto 0);
307         dout_2: out std_logic_vector(17 downto 0));
308     end component;
309
310     component f4_48k
311         port (
312         clk: in std_logic;
313         nd: in std_logic;
314         rfd: out std_logic;
315         rdy: out std_logic;
316         din_1: in std_logic_vector(16 downto 0);
317         din_2: in std_logic_vector(16 downto 0);
318         dout_1: out std_logic_vector(17 downto 0);
319         dout_2: out std_logic_vector(17 downto 0));
320     end component;
321
322     component f5_48k
323         port (
324         clk: in std_logic;
325         nd: in std_logic;
326         rfd: out std_logic;
327         rdy: out std_logic;
328         din_1: in std_logic_vector(16 downto 0);
329         din_2: in std_logic_vector(16 downto 0);
330         dout_1: out std_logic_vector(17 downto 0);
331         dout_2: out std_logic_vector(17 downto 0));
332     end component;
333
334     component f6_48k
335         port (
336         clk: in std_logic;
337         nd: in std_logic;
338         rfd: out std_logic;
339         rdy: out std_logic;
340         din_1: in std_logic_vector(16 downto 0);
341         din_2: in std_logic_vector(16 downto 0);
342         dout_1: out std_logic_vector(17 downto 0);
```

```
343         dout_2: out std_logic_vector(17 downto 0));
344     end component;
345 
346     component f7_48k
347         port (
348         clk: in std_logic;
349         nd: in std_logic;
350         rfd: out std_logic;
351         rdy: out std_logic;
352         din_1: in std_logic_vector(16 downto 0);
353         din_2: in std_logic_vector(16 downto 0);
354         dout_1: out std_logic_vector(17 downto 0);
355         dout_2: out std_logic_vector(17 downto 0));
356     end component;
357 
358     component f8_48k
359         port (
360         clk: in std_logic;
361         nd: in std_logic;
362         rfd: out std_logic;
363         rdy: out std_logic;
364         din_1: in std_logic_vector(16 downto 0);
365         din_2: in std_logic_vector(16 downto 0);
366         dout_1: out std_logic_vector(17 downto 0);
367         dout_2: out std_logic_vector(17 downto 0));
368     end component;
369 
370     component f9_48k
371         port (
372         clk: in std_logic;
373         nd: in std_logic;
374         rfd: out std_logic;
375         rdy: out std_logic;
376         din_1: in std_logic_vector(16 downto 0);
377         din_2: in std_logic_vector(16 downto 0);
378         dout_1: out std_logic_vector(17 downto 0);
379         dout_2: out std_logic_vector(17 downto 0));
380     end component;
381 
382     COMPONENT gainAtten
383     PORT(
384         input : IN std_logic_vector(17 downto 0);
385         pcmInput : IN std_logic_vector(7 downto 0);
386         output : OUT std_logic_vector(17 downto 0)
387         );
388     END COMPONENT;
389 
390 begin
391 
392     -- Instantiate clock manager. This IP CORE generates and distributes
393     -- the clocks required by the system from the BITCLK signal, which
394     -- comes from the LM4550.
395     masterClock : clockManager
396       port map
397         (-- Clock in ports
398          CLK_IN1 => BITCLK,
399          -- Clock out ports
400          CLK_OUT1 => clk12_288M,
401          CLK_OUT2 => clk24_576M,
402          CLK_OUT3 => clk2_048M,
403          RESET => notSysReset,
404          CLK_VALID => CLK_VALID);
405 
406     -- Instantiate the LM4550 driver.
407     audDriver: codecDriver
408     port map(
409         BITCLK => clk12_288M,
410         AUDSYNC => bufAUDSYNC,
411         AUDRST => bufAUDRST,
```

```
412         AUDSDI => AUDSDI,
413         AUDSDO => bufAUDSDO,
414         debug => audDebug,
415         in_pcm_r =>  audioOut_r,
416         in_pcm_l =>  audioOut_l,
417         out_pcm_r => audioIn_r,
418         out_pcm_l => audioIn_l,
419         shortReset => shortReset,
420         longReset => longReset,
421         dataOk => codecDataOk,
422         intfcSync => codecIntfcSync
423     );
424     AUDSYNC <= bufAUDSYNC;
425     AUDSDO <= bufAUDSDO;
426     AUDRST <= bufAUDRST;
427 
428     -- Instantiate the interfaceDriver, which controls the LED and ADC drivers.
429     Inst_interfaceDriver2: interfaceDriver2 PORT MAP(
430         ADC_DATA_OUT => ADC_DATA_OUT,
431         ADC_SYSCLK => ADC_SYSCLK,
432         ADC_IOCLK => ADC_IOCLK,
433         ADC_ADDR_INPUT => ADC_ADDR_INPUT,
434         ADC_CS_C => ADC_CS_C,
435         LED_SIN => LED_SIN,
436         LED_BLANK => LED_BLANK,
437         LED_SCLK => LED_SCLK,
438         LED_LAT => LED_LAT,
439         reset => longReset,
440         clk2ish => clk2_048M,
441         debug => intDebug,
442         adc1_out => adc1,
443         adc2_out => adc2,
444         adc3_out => adc3,
445         adc4_out => adc4,
446         adc5_out => adc5,
447         adc6_out => adc6,
448         adc7_out => adc7,
449         adc8_out => adc8,
450         adc9_out => adc9,
451         led1_in => led1,
452         led2_in => led2,
453         led3_in => led3,
454         led4_in => led4,
455         led5_in => led5,
456         led6_in => led6,
457         led7_in => led7,
458         led8_in => led8,
459         led9_in => led9
460     );
461 
462     -- Instantiate all 9 of the FIR filters used for audio processing. All of
463     -- these filters are Xilinx FIR Compiler 5.0 Cores.
464     filter1 : f1_48k
465             port map (
466                 clk => clk24_576M,
467                 nd => nd1,
468                 rfd => rfd1,
469                 rdy => rdy1,
470                 din_1 => audioIn_l(17 downto 1),
471                 din_2 => audioIn_r(17 downto 1),
472                 dout_1 => pregam1_l,
473                 dout_2 => pregam1_r);
474 
475     filter2 : f2_48k
476             port map (
477                 clk => clk24_576M,
478                 nd => nd2,
479                 rfd => rfd2,
480                 rdy => rdy2,
```

```
481                 din_1 => audioIn_l(17 downto 1),
482                 din_2 => audioIn_r(17 downto 1),
483                 dout_1 => pregam2_l,
484                 dout_2 => pregam2_r);
485
486     filter3 : f3_48k
487             port map (
488                 clk => clk24_576M,
489                 nd => nd3,
490                 rfd => rfd3,
491                 rdy => rdy3,
492                 din_1 => audioIn_l(17 downto 1),
493                 din_2 => audioIn_r(17 downto 1),
494                 dout_1 => pregam3_l,
495                 dout_2 => pregam3_r);
496
497     filter4 : f4_48k
498             port map (
499                 clk => clk12_288M,
500                 nd => nd4,
501                 rfd => rfd4,
502                 rdy => rdy4,
503                 din_1 => audioIn_l(17 downto 1),
504                 din_2 => audioIn_r(17 downto 1),
505                 dout_1 => pregam4_l,
506                 dout_2 => pregam4_r);
507
508     filter5 : f5_48k
509             port map (
510                 clk => clk12_288M,
511                 nd => nd5,
512                 rfd => rfd5,
513                 rdy => rdy5,
514                 din_1 => audioIn_l(17 downto 1),
515                 din_2 => audioIn_r(17 downto 1),
516                 dout_1 => pregam5_l,
517                 dout_2 => pregam5_r);
518
519     filter6 : f6_48k
520             port map (
521                 clk => clk12_288M,
522                 nd => nd6,
523                 rfd => rfd6,
524                 rdy => rdy6,
525                 din_1 => audioIn_l(17 downto 1),
526                 din_2 => audioIn_r(17 downto 1),
527                 dout_1 => pregam6_l,
528                 dout_2 => pregam6_r);
529
530     filter7 : f7_48k
531             port map (
532                 clk => clk12_288M,
533                 nd => nd7,
534                 rfd => rfd7,
535                 rdy => rdy7,
536                 din_1 => audioIn_l(17 downto 1),
537                 din_2 => audioIn_r(17 downto 1),
538                 dout_1 => pregam7_l,
539                 dout_2 => pregam7_r);
540
541     filter8 : f8_48k
542             port map (
543                 clk => clk12_288M,
544                 nd => nd8,
545                 rfd => rfd8,
546                 rdy => rdy8,
547                 din_1 => audioIn_l(17 downto 1),
548                 din_2 => audioIn_r(17 downto 1),
549                 dout_1 => pregam8_l,
```

```
550                 dout_2 => pregam8_r);
551
552     filter9 : f9_48k
553             port map (
554                 clk => clk12_288M,
555                 nd => nd9,
556                 rfd => rfd9,
557                 rdy => rdy9,
558                 din_1 => audioIn_l(17 downto 1),
559                 din_2 => audioIn_r(17 downto 1),
560                 dout_1 => pregam9_l,
561                 dout_2 => pregam9_r);
562
563     -- Assign all of the "New Data" pins for each FIR filter to the codecDataOk
564     -- signal. This ensures that all of the FIR filters will be syncronized, as
565     -- they will all begin to process data on the first rising clock edge after
566     -- the codecDataOk signal is asserted.
567     nd1 <= codecDataOk;
568     nd2 <= codecDataOk;
569     nd3 <= codecDataOk;
570     nd4 <= codecDataOk;
571     nd5 <= codecDataOk;
572     nd6 <= codecDataOk;
573     nd7 <= codecDataOk;
574     nd8 <= codecDataOk;
575     nd9 <= codecDataOk;
576
577
578     -- Instantiate all of the gain/attenuation modules. These modules adjust the
579     -- gain and attenuation of each audio signal coming out of every FIR filter,
580     -- based on the data received from the ADC.
581     gam1r: gainAtten PORT MAP(
582         input => pregam1_r ,
583         pcmInput => adc1,
584         output => g1out_r
585     );
586
587     gam1l: gainAtten PORT MAP(
588         input => pregam1_l,
589         pcmInput => adc1,
590         output => g1out_l
591     );
592
593     gam2r: gainAtten PORT MAP(
594         input => pregam2_r ,
595         pcmInput => adc2,
596         output => g2out_r
597     );
598
599     gam2l: gainAtten PORT MAP(
600         input => pregam2_l,
601         pcmInput => adc2,
602         output => g2out_l
603     );
604
605     gam3r: gainAtten PORT MAP(
606         input => pregam3_r ,
607         pcmInput => adc3,
608         output => g3out_r
609     );
610
611     gam3l: gainAtten PORT MAP(
612         input => pregam3_l,
613         pcmInput => adc3,
614         output => g3out_l
615     );
616
617     gam4r: gainAtten PORT MAP(
618         input => pregam4_r ,
```

```
619         pcmInput => adc4,
620         output => g4out_r
621     );
622 
623     gam4l: gainAtten PORT MAP(
624         input => pregam4_l,
625         pcmInput => adc4,
626         output => g4out_l
627     );
628 
629     gam5r: gainAtten PORT MAP(
630         input => pregam5_r ,
631         pcmInput => adc5,
632         output => g5out_r
633     );
634 
635     gam5l: gainAtten PORT MAP(
636         input => pregam5_l,
637         pcmInput => adc5,
638         output => g5out_l
639     );
640 
641 
642     gam6r: gainAtten PORT MAP(
643         input => pregam6_r ,
644         pcmInput => adc6,
645         output => g6out_r
646     );
647 
648     gam6l: gainAtten PORT MAP(
649         input => pregam6_l,
650         pcmInput => adc6,
651         output => g6out_l
652     );
653 
654 
655     gam7r: gainAtten PORT MAP(
656         input => pregam7_r ,
657         pcmInput => adc7,
658         output => g7out_r
659     );
660 
661     gam7l: gainAtten PORT MAP(
662         input => pregam7_l,
663         pcmInput => adc7,
664         output => g7out_l
665     );
666 
667 
668     gam8r: gainAtten PORT MAP(
669         input => pregam8_r ,
670         pcmInput => adc8,
671         output => g8out_r
672     );
673 
674     gam8l: gainAtten PORT MAP(
675         input => pregam8_l,
676         pcmInput => adc8,
677         output => g8out_l
678     );
679 
680     gam9r: gainAtten PORT MAP(
681         input =>pregam9_r ,
682         pcmInput => adc9,
683         output => g9out_r
684     );
685 
686     gam9l: gainAtten PORT MAP(
687         input => pregam9_l,
```

```
        pcmInput => adc9,
        output => g9out_l
    );

    -- Instantiate the summation block, which sums all of the FIR filter outputs
    -- together into one signal, which is the filtered audio output.
    Inst_summer9x2ch: summer9x2ch PORT MAP(
        clock => clk12_288M,
        in1_r => g1out_r,
        in1_l => g1out_l,
        in2_r => g2out_r,
        in2_l => g2out_l,
        in3_r => g3out_r,
        in3_l => g3out_l,
        in4_r => g4out_r,
        in4_l => g4out_l,
        in5_r => g5out_r,
        in5_l => g5out_l,
        in6_r => g6out_r,
        in6_l => g6out_l,
        in7_r => g7out_r,
        in7_l => g7out_l,
        in8_r => g8out_r,
        in8_l => g8out_l,
        in9_r => g9out_r,
        in9_l => g9out_l,
        out_r => audioOut_r,
        out_l => audioOut_l
    );

    -- Instantiate all of the LED lookup tables. One lookup table for each
    -- frequency band. The lookup tables assign an output to the LEDs, based
    -- on the PCM balues of that band.
    ledLut1: ledLut PORT MAP(
        bandPcm =>g1out_r ,
        ledOut => led1
    );
    ledLut2: ledLut PORT MAP(
        bandPcm =>g2out_r ,
        ledOut => led2
    );

    ledLut3: ledLut PORT MAP(
        bandPcm =>g3out_r ,
        ledOut => led3
    );
    ledLut4: ledLut PORT MAP(
        bandPcm =>g4out_r ,
        ledOut => led4
    );
    ledLut5: ledLut PORT MAP(
        bandPcm =>g5out_r ,
        ledOut => led5
    );
    ledLut6: ledLut PORT MAP(
        bandPcm =>g6out_r ,
        ledOut => led6
    );
    ledLut7: ledLut PORT MAP(
        bandPcm =>g7out_r ,
        ledOut => led7
    );
    ledLut8: ledLut PORT MAP(
        bandPcm =>g8out_r ,
        ledOut => led8
    );
    ledLut9: ledLut PORT MAP(
        bandPcm =>g9out_r ,
        ledOut => led9
```

```
    );

    -- Reset timer generator ----------------------------------------------------
    -- This is to generate a long reset on system startup, without relying on
    -- BITCLK from the LM4550. The longReset and shortReset signals both begin
    -- at the same time, but the longReset signal lasts longer. The purpose of
    -- this is to give the LM4550 to set up (which is reset with shortReset)
    -- before the rest of the system is active (which is reset with longReset).
    -------------------------------------------------------------------------------
    process (sysClk, sysReset) begin
        if rising_edge(sysClk) then
            if sysReset = '0' then
                resetGenerator <= "000000000000000000000000";
                longReset <= '0';
                shortReset <= '0';
            elsif resetGenerator = "000000000000000000100000" then
                shortReset <= '1';
                resetGenerator <="000000000000000000100001";
            elsif resetGenerator = "111111111111111111111111" then
                longReset <= '1';
            else
                resetGenerator <= resetGenerator +"000000000000000000000001";
            end if;
        end if;
    end process;
    notSysReset <= not(shortReset);
    -------------------------------------------------------------------------------

    -- Debug ports, for system development. These control the outputs to the 8
    -- LEDs and the 8-pin PMOD connector on the Atlys board.
    led <= "00000000";
    JB(0) <= '0';
    JB(1) <= '0';
    JB(2) <= '0';
    JB(3) <= '0';
    JB(4) <= '0';
    JB(5) <= '0';
    JB(6) <= '0';
    JB(7) <= '0';

end Behavioral;
```

codecDriver.vhd

```
1  ----------------------------------------------------------------------------
2  -- audDriver.vhd ----------------------------------------------------------
3  -- Author: Anthony Giardina                                               --
4  -- For: Graphic Audio EQ Senior Project                                   --
5  -- Last Modified: 5/29/2012                                               --
6  -- Purpose: This module is a driver for the National Semiconductor LM4550    --
7  --  audio codec. Specifically, the LM4550BVH. It uses the Intel AC-LINK      --
8  --  protocol. It configures registers in the LM4550, and then reads and      --
9  --  writes 18-bit PCM values to the device at 48kHz.                        --
10 ----------------------------------------------------------------------------
11 ----------------------------------------------------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 use Ieee.std_logic_unsigned.all;
15 use IEEE.NUMERIC_STD.ALL;
16
17 entity codecDriver is
18     Port (
19         --Communication signals for physical AC-Link connection to codec:
20         BITCLK : in  STD_LOGIC;
21         AUDSYNC : out STD_LOGIC;
22         AUDRST : out STD_LOGIC;
23         AUDSDI : in  STD_LOGIC;
24         AUDSDO : out  STD_LOGIC;
25         debug : out std_logic_vector(7 downto 0);
26         --stereo PCM data stream between codec driver and audio processor:
27         in_pcm_r : in std_logic_vector (17 downto 0);
28         in_pcm_l : in std_logic_vector (17 downto 0);
29         out_pcm_r: out std_logic_vector (17 downto 0);
30         out_pcm_l: out std_logic_vector (17 downto 0);
31         -- Signals for system control and syncronization:
32         shortReset : in std_logic;
33         longReset: in std_logic;
34         intfcSync: out std_logic;
35         dataOk: out std_logic
36     );
37 end codecDriver;
38
39 architecture Behavioral of codecDriver is
40
41     COMPONENT frameBuilder
42     PORT(
43         pcmL : IN std_logic_vector(17 downto 0);
44         pcmR : IN std_logic_vector(17 downto 0);
45         codecReady : IN std_logic;
46         cmd : IN std_logic_vector(3 downto 0);
47         frame : OUT std_logic_vector(95 downto 0)
48         );
49     END COMPONENT;
50
51     -- The codecReady bit is set when the LM4550 has signaled that it
52     -- it's power supplies have stabalized, it is internally ready, and
53     -- can accept new data.
54     signal codecReady : std_logic;
55     -- The following two signals are the PCM values which are sent to the
56     -- LM4550, once they are assembled into complete data packets.
57     signal PCM_R : std_logic_vector (17 downto 0);
58     signal PCM_L : std_logic_vector (17 downto 0);
59     -- The following two signals are the PCM values that have been read
60     -- in from the LM4550, from the line-in port.
61     signal PCM_R_IN : std_logic_vector (17 downto 0);
62     signal PCM_L_IN : std_logic_vector (17 downto 0);
63    -- the cmd signal tells the frameBuilder module what frame it should build.
64     -- see where this signal is used for more info:
65     signal cmd: std_logic_vector (3 downto 0);
```

```
66      -- the frame signal is the assembled data packet that is sent to the LM4550:
67     signal frame: std_logic_vector (95 downto 0);
68      -- bitCounter is a state machine variable that keeps track of how many clock
69      -- cycles have occured since the beginning of each communication frame
70      -- with the LM4550.
71      signal bitCounter: std_logic_vector (7 downto 0);
72      -- operation counter is a state machine variable that determines the
73      -- value of cmd.
74      signal operationCounter: std_logic_vector (5 downto 0);
75      -- readBackAddr is the address value that is read back from the LM4550 after
76      -- a register is programed. The readBackAddr must match the address that was
77      -- previously written. Same concept for readBackData.
78      signal readBackAddr: std_logic_vector (6 downto 0);
79      signal readBackData: std_logic_vector (15 downto 0);
80      -- lastAddr keeps track of the last address that was written
81     -- for readback verification. Same concept for lastData and lastCodecReady.
82     signal lastAddr : std_logic_vector (6 downto 0);
83      signal lastData: std_logic_vector (15 downto 0);
84     signal lastCodecReady: std_logic;
85      -- This signal exists to complete a case statement, and has no real purpose:
86     signal a: std_logic;
87      -- paddedFrame is the frame signal, with some zeroes padded to the front,
88      -- so that the timing diagram matches properly.
89      signal paddedFrame: std_logic_vector (97 downto 0);
90     -- internal buffer for the AUDSYNC signal:
91      signal sync: std_logic;
92
93 begin
94
95      -- Reset the LM4550. Using the shortReset instead of longReset ensures
96      -- that the LM4550 BITCLK signal, which is required for proper operation
97      -- of most of the firmware, is up and running properly before the rest of
98      -- the firmware comes out of the reset state, since it all uses the
99      -- longReset signal.
100     AUDRST <= shortReset;
101
102     -- Debug connection, for communicating to higher level modules.
103     debug(0) <= '0';
104     debug(1) <= '0';
105     debug(2) <= '0';
106     debug(3) <= '0';
107     debug(4) <= '0';
108     debug(5) <= '0';
109     debug(6) <= '0';
110     debug(7) <= '0';
111
112    -- Update the PCM values syncronously, so that they are not accidentally
113     -- read/written when they are being updated/used.
114     process (sync) begin
115    if rising_edge(sync) then
116
117         PCM_R <= in_pcm_r;
118         PCM_L <= in_pcm_l;
119
120         out_pcm_r <= PCM_R_IN;
121         out_pcm_l <= PCM_L_IN;
122
123     end if;
124    end process;
125
126
127    -- Instantiate the frameBuilter module, which assembles the data packets,
128     -- which are then sent to the LM4550.
129     Inst_frameBuilder: frameBuilder PORT MAP(
130         pcmL => PCM_L,
131         pcmR => PCM_R,
132         codecReady => codecReady,
133         cmd => cmd,
134         frame => frame
```

```
135     );
136
137     -- Assign values to the AUDSYNC signal and it's buffer:
138     sync <= '1' when (bitCounter >= "00000000" and bitCounter <= "00001111")  else
139                 '0';
140      AUDSYNC <= sync; -- buffer so we can use the signal internally.
141
142     -- Assign cmd values, based on the operationCounter state machine.
143     with operationCounter select
144         cmd <= "0000" when  "000000", --nothing
145                "0000" when  "000001", --nothing
146                  "0100" when  "000010", --write
147                  "1100" when  "000011", --request read
148                  "0000" when  "000100", --read and verify
149                  "0110" when  "000101", --write
150                  "1110" when  "000110", --request read
151                  "0000" when  "000111", --read and verify
152                  "0010" when  "001000", --write
153                  "1010" when  "001001", --read
154                  "0000" when  "001010", --verify
155                  "0011" when  "001011", --write
156                  "1011" when  "001100", --read
157                  "0000" when  "001101", --verify
158                  "0000" when others;
159
160      -- This statement places the proper bit of the current frame
161      -- on the AUDSDO pin.
162     AUDSDO <= '0' when longReset = '0' else
163               paddedFrame (96 - to_integer(unsigned(bitCounter)))
164                     when bitCounter <= "01011111" else
165              '0';
166
167      -- Adjust the frame, so that it starts at the proper time.
168      paddedFrame <= "00" & frame;
169
170     -- This process controls the large state machine, which controls the entire
171     -- driver.
172      process (longReset, BITCLK) begin
173          if rising_edge(BITCLK) then
174              if longReset = '0' then
175                 -- Syncronous reset:
176                  bitCounter <= "11111111";
177                  operationCounter <= "000000";
178                  lastData <= "0000000000000000";
179                  lastAddr <= "0000000";
180            elsif bitCounter = "11111111" then
181                 -- If in the final state, reset the state machine.
182                  bitCounter <= "00000000";
183                  -- Latch in the value of codecReady, so we know what to do
184              --  with the PCM values on the next frame.
185                  codecReady <= lastCodecReady;
186                  if (operationCounter = "000010" or operationCounter = "000101"
187                         or operationCounter = "001000"
188                           or operationCounter = "001011" ) then
189                           -- these are the write operations.
190                           -- Proceed to the read/verify operation.
191                      lastAddr <= frame(78 downto 72);
192                      lastData <= frame(59 downto 44);
193                      operationCounter <= operationCounter + "000001";
194                  elsif (operationCounter = "000011" or operationCounter = "000110"
195                           or operationCounter = "001001"
196                              or operationCounter = "001100" ) then
197                              -- Requesting a readback of the previously written
198                              -- register. Move along to verification.
199                     operationCounter <= operationCounter + "000001";
200                 elsif (operationCounter = "000100" or operationCounter = "000111"
201                           or operationCounter = "001010"
202                              or operationCounter = "001101") then
203                              -- Reading back the previously requested information.
```

```
                            -- Make a decision about about the next state, based
                            -- on the validity of the information read back.
                    if (lastAddr = readBackAddr and lastData = readBackData) then
                        -- if the data and address written match the data and address
                   -- read, proceed to the next operation
                         operationCounter <= operationCounter + "000001";
                     else
                        -- if the read/write data does not match, go to the
                         -- previous state and try again.
                        operationCounter <= operationCounter - "000010";
                     end if;
                elsif (operationCounter = "000000" or operationCounter = "000001" )
                            then
                            -- These are the do nothing operations.
                            -- Just advance to the next operation.
                        operationCounter <= operationCounter + "000001";
                end if;
            else
                bitCounter <= bitCounter + "00000001";
          end if;
       end if;
    end process;

   -- This process reads in bits from the AUDSDI pin of rht LM4550
    -- at the proper time, and also generates syncronization signals
    -- which are used to syncronize the audio processing filters and
    -- the interface module.
   process (longReset, BITCLK) begin
       if falling_edge(BITCLK) then
         case bitCounter is
               --when "00000000" => ;
               when "00000001" => lastCodecReady <= AUDSDI;
               --when "00000010" => ;
               --when "00000011" => ;
               --when "00000100" => ;
               when "00000101" => intfcSync <= '1';
               when "00000110" => intfcSync <= '0';
               --when "00000111" => ;
               --when "00001000" => ;
               --when "00001001" => ;
               --when "00001010" => ;
               --when "00001011" => ;
               --when "00001100" => ;
               --when "00001101" => ;
               --when "00001110" => ;
               --when "00001111" => ;
               --when "00010000" => ;
               --when "00010001" => ;
               when "00010010" => readBackAddr(6) <= AUDSDI;
               when "00010011" => readBackAddr(5) <= AUDSDI;
               when "00010100" => readBackAddr(4) <= AUDSDI;
               when "00010101" => readBackAddr(3) <= AUDSDI;
               when "00010110" => readBackAddr(2) <= AUDSDI;
               when "00010111" => readBackAddr(1) <= AUDSDI;
               when "00011000" => readBackAddr(0) <= AUDSDI;
               --when "00011001" => ;
               --when "00011010" => ;
               --when "00011011" => ;
               --when "00011100" => ;
               --when "00011101" => ;
               --when "00011110" => ;
               --when "00011111" => ;
               --when "00100000" => ;
               --when "00100001" => ;
               --when "00100010" => ;
               --when "00100011" => ;
               --when "00100100" => ;
               when "00100101" => readBackData(15) <= AUDSDI;
               when "00100110" => readBackData(14) <= AUDSDI;
```

```
                when "00100111" => readBackData(13) <= AUDSDI;
                when "00101000" => readBackData(12) <= AUDSDI;
                when "00101001" => readBackData(11) <= AUDSDI;
                when "00101010" => readBackData(10) <= AUDSDI;
                when "00101011" => readBackData(9) <= AUDSDI;
                when "00101100" => readBackData(8) <= AUDSDI;
                when "00101101" => readBackData(7) <= AUDSDI;
                when "00101110" => readBackData(6) <= AUDSDI;
                when "00101111" => readBackData(5) <= AUDSDI;
                when "00110000" => readBackData(4) <= AUDSDI;
                when "00110001" => readBackData(3) <= AUDSDI;
                when "00110010" => readBackData(2) <= AUDSDI;
                when "00110011" => readBackData(1) <= AUDSDI;
                when "00110100" => readBackData(0) <= AUDSDI;
                --when "00110101" => ;
                --when "00110110" => ;
                --when "00110111" => ;
                --when "00111000" => ;
                when "00111001" => PCM_L_IN(17) <= AUDSDI;
                 when "00111010" => PCM_L_IN(16) <= AUDSDI;
                 when "00111011" => PCM_L_IN(15) <= AUDSDI;
                 when "00111100" => PCM_L_IN(14) <= AUDSDI;
                 when "00111101" => PCM_L_IN(13) <= AUDSDI;
                 when "00111110" => PCM_L_IN(12) <= AUDSDI;
                 when "00111111" => PCM_L_IN(11) <= AUDSDI;
                 when "01000000" => PCM_L_IN(10) <= AUDSDI;
                 when "01000001" => PCM_L_IN(9) <= AUDSDI;
                 when "01000010" => PCM_L_IN(8) <= AUDSDI;
                 when "01000011" => PCM_L_IN(7) <= AUDSDI;
                 when "01000100" => PCM_L_IN(6) <= AUDSDI;
                 when "01000101" => PCM_L_IN(5) <= AUDSDI;
                 when "01000110" => PCM_L_IN(4) <= AUDSDI;
                 when "01000111" => PCM_L_IN(3) <= AUDSDI;
                 when "01001000" => PCM_L_IN(2) <= AUDSDI;
                 when "01001001" => PCM_L_IN(1) <= AUDSDI;
                 when "01001010" => PCM_L_IN(0) <= AUDSDI;
                 --when "01001011" => ;
                 --when "01001100" => ;
                 when "01001101" => PCM_R_IN(17) <= AUDSDI;
                 when "01001110" => PCM_R_IN(16) <= AUDSDI;
                 when "01001111" => PCM_R_IN(15) <= AUDSDI;
                 when "01010000" => PCM_R_IN(14) <= AUDSDI;
                 when "01010001" => PCM_R_IN(13) <= AUDSDI;
                 when "01010010" => PCM_R_IN(12) <= AUDSDI;
                 when "01010011" => PCM_R_IN(11) <= AUDSDI;
                 when "01010100" => PCM_R_IN(10) <= AUDSDI;
                 when "01010101" => PCM_R_IN(9) <= AUDSDI;
                 when "01010110" => PCM_R_IN(8) <= AUDSDI;
                 when "01010111" => PCM_R_IN(7) <= AUDSDI;
                 when "01011000" => PCM_R_IN(6) <= AUDSDI;
                 when "01011001" => PCM_R_IN(5) <= AUDSDI;
                 when "01011010" => PCM_R_IN(4) <= AUDSDI;
                 when "01011011" => PCM_R_IN(3) <= AUDSDI;
                 when "01011100" => PCM_R_IN(2) <= AUDSDI;
                 when "01011101" => PCM_R_IN(1) <= AUDSDI;
                 when "01011110" => PCM_R_IN(0) <= AUDSDI;
                 --when "01011111" => ;
                 -- Assert a signal so that he filters know when to begin processing
                 -- data. dataOk is asserted once the PCM values are finished
                 -- being read in from the LM4550.
                 when "01100010" => dataOk <= '1';
                 when "01100011" => dataOk <= '0';
                 when others => a <= a;
            end case;
        end if;
    end process;

end Behavioral;
```

frameBuilder.vhd

```
1  ----------------------------------------------------------------------------
2  -- frameBuilder.vhd -------------------------------------------------------
3  -- Author: Anthony Giardina                                               --
4  -- For: Graphic Audio EQ Senior Project                                   --
5  -- Last Modified: 5/29/2012                                               --
6  -- Purpose: This module assembles data packets, which are sent to the LM4550.--
7  --  It constructs these packets based on the current requested command,   --
8  --  which could be writing or reading a register, and the desired PCM     --
9  --  values to be sent to the LM4550 for output.                           --
10 ----------------------------------------------------------------------------
11 ----------------------------------------------------------------------------
12
13 library IEEE;
14 use IEEE.STD_LOGIC_1164.ALL;
15
16 entity frameBuilder is
17     Port (
18         -- Audio output PCM signals, which are sent to the LM4550:
19         pcmL : in  STD_LOGIC_VECTOR (17 downto 0);
20         pcmR : in  STD_LOGIC_VECTOR (17 downto 0);
21         -- Status of the LM4550, ready or not:
22         codecReady : in  STD_LOGIC;
23         -- Desired command:
24         cmd : in STD_LOGIC_VECTOR (3 downto 0);
25         -- Assembled output frame:
26         frame : out  STD_LOGIC_VECTOR (95 downto 0)
27     );
28 end frameBuilder;
29
30 architecture Behavioral of frameBuilder is
31
32
33    COMPONENT cmdBank
34        PORT(
35            cmd : IN std_logic_vector(3 downto 0);
36            CMD_ADDR : OUT std_logic_vector(6 downto 0);
37            CMD_DATA : OUT std_logic_vector(15 downto 0)
38            );
39        END COMPONENT;
40
41     -- address slot of the communication frame:
42    signal address: std_logic_vector (6 downto 0);
43     -- command slot of the communication frame:
44     signal command: std_logic_vector (15 downto 0);
45
46 begin
47
48    -- Instantiate the command back, which is essentially a lookup table,
49     -- creating the ADDRESS and DATA slots of the communication frame.
50     Inst_cmdBank: cmdBank PORT MAP(
51         cmd => cmd,
52         CMD_ADDR => address,
53         CMD_DATA => command
54     );
55
56
57     -- Tag Slot ----------------------------------------------------------
58     -- Indicate that the frame contains valid data, if the codec is ready
59     -- to accept data.
60     frame (95) <= codecReady;
61     -- Set the control address valid flag:
62     frame (94) <= '0' when codecReady = '0' else -- 1: control address is valid
63                   '1' when cmd(3) = '1' or cmd(2) = '1' or cmd(1) = '1'
64                           or cmd(0) = '1' else
65                       '0';
66     -- Set the control data valid flag:
67     frame (93) <= '0' when codecReady = '0' else -- 1: control data is valid
68                                                  -- (this is vaild on a write,
```

```
                                                    -- invaild on a read, invalid
                                                    -- if no cmd at all)
                    '1' when cmd(3) = '0' and (cmd(2) = '1' or cmd(1) = '1'
                           or cmd(0) = '1') else
                        '0';
    -- Set the left and right PCM data valid flags:
    frame (92) <= '0' when codecReady = '0' else -- 1: left PCM data is valid.
                                                    -- Always send PCM data if
                                                                -- codec is ready.
                    '1';
    frame (91) <= '0' when codecReady = '0' else -- 1: Right PCM data is valid.
                                                    -- Always send PCM data if
                                                                -- codec is ready.
                    '1';
    -- This is valid slot data for channels that we don't use, so it's always 0:
    frame (90 downto 84) <= "0000000";
    frame (83 downto 80) <= "0000"; -- unused bits stuffed with zeroes.

    -- Control Address slot ------------------------------------------------------
    frame (79) <= '0' when codecReady = '0' else -- read or write
                  cmd(3);
    -- address of register that's the target of the read/write operation
    frame (78 downto 72) <= "0000000"  when codecReady = '0' else
                            address;
    frame (71 downto 60) <= "000000000000"; -- Unused bits

   -- Control data slot ---------------------------------------------------------
    -- Data to be written to target register (all zeroes if read operation)
    frame (59 downto 44) <= "0000000000000000"  when codecReady = '0' else
                            command;
    frame (43 downto 40) <= "0000"; -- unused bits

   -- PCM left slot -------------------------------------------------------------
    -- Send PCM data only when the codec is ready
    frame (39 downto 22) <= "00000000000000000000" when codecReady = '0'  else
                            pcmL;
    frame (21 downto 20) <= "00"; -- unused bits

    -- PCM right slot ------------------------------------------------------------
    -- send PCM data only when the codec is ready
    frame (19 downto 2)  <= "00000000000000000000" when codecReady = '0'  else
                            pcmR;
    frame (1 downto 0) <= "00"; -- unused bits

end Behavioral;
```

cmdBank.vhd

```
1  ----------------------------------------------------------------------------
2  -- cmdBank.vhd      ------------------------------------------------------
3  -- Author: Anthony Giardina                                              --
4  -- For: Graphic Audio EQ Senior Project                                   --
5  -- Last Modified: 5/29/2012                                               --
6  -- Purpose: the cmdBank module stores register values, which are used to  --
7  --  configure the LM4550 on startup. It is essentially a large lookup table. --
8  ----------------------------------------------------------------------------
9  ----------------------------------------------------------------------------
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 use IEEE.NUMERIC_STD.ALL;
13
14 entity cmdBank is
15     Port (
16         -- Command input:
17         cmd : in  STD_LOGIC_VECTOR (3 downto 0);
18         -- Address and Data outputs:
19         CMD_ADDR : out  STD_LOGIC_VECTOR (6 downto 0);
20         CMD_DATA : out  STD_LOGIC_VECTOR (15 downto 0)
21     );
22 end cmdBank;
23
24 architecture Behavioral of cmdBank is
25
26    -- All of these signals are used to store register configureation values.
27     -- They are only read, never written.
28     signal micGain: std_logic_vector (6 downto 0);
29     signal sourceSelect: std_logic;
30     signal recGainMute: std_logic_vector(4 downto 0);
31     signal pcmOutGAM: std_logic_vector (5 downto 0);
32     signal headphoneVol: std_logic_vector (5 downto 0);
33     signal masterVol: std_logic_vector (5 downto 0);
34
35 begin
36
37    -- First bit sets mute (1) or unmute (0).
38     -- second bit Sets mic gain to 0dB (0) or +20dB (1).
39     -- Last 5 bits set gain or attn. 00000 -> +12dB.
40     --                               01000 -> 0dB gain.
41     --                               11111 -> 34.5dB gain.
42     micGain <= "0001000";
43
44     -- Select recording source: mic (0) or line in (1)
45     sourceSelect <= '1';
46
47     -- Select recording source gain.
48     -- First bit is unmute (0) or mute(1).
49     -- Last 4 bits are gain. 0000 -> 0dB.
50     --                       1111 -> +22.5dB
51     recGainMute <= "00100";
52
53     -- PCM output from D->A converter gain.
54     -- first bit is unmute (0) or mute(1).
55     -- last 4 bits are 00000 -> +12dB gain.
56     --                 01000 -> 0dB gain.
57     --                 11111 -> -34.5dB gain.
58     pcmOutGAM <= "000111";
59
60     -- Headphone vol. First bit is unmute (0) or mute (1).
61     -- Last 5 bits are attenuation. 00000 -> 0dB.
62     --                              11111 -> -46.5dB atten.
63     headphoneVol <= "000000";
64
65     -- Master volume. same as headphone volmue as far as the bits are concerned.
```

```
66      masterVol <= "000000";
67
68      -- CMD:
69      -- 1001: Read Mic Gain (either 0dB or +20dB)
70      -- 1010: Read Record Select (source = 0: mic. Source = 1: line in.)
71      -- 1011: Read Record Gain
72      -- 1100: Read PCM output gain/attn/mute
73      -- 1101: Read Headphone volume
74      -- 1110: Read Master volume
75      -- 0001: Write Mic Gain (either 0dB or +20dB)
76      -- 0010: Write Record Select (source = 0: mic. Source = 1: line in.)
77      -- 0011: Write Record Gain
78      -- 0100: Write PCM output gain/attn/mute
79      -- 0101: Write Headphone volume
80      -- 0110: Write Master volume
81      -- cmd(3) is the leading bit, and determines a write (0)
82      -- or a read (1) operation. If it's a read operation, data field is blank.
83
84      -- Set the register address, based on this lookup table:
85      CMD_ADDR(6 downto 0) <=    -- Mic Gain:
86                                 "0001110" when cmd(2 downto 0) = "001" else
87                                         -- Rec source:
88                                 "0011010" when cmd(2 downto 0) = "010" else
89                                         -- Rec Gain:
90                                         "0011100" when cmd(2 downto 0) = "011" else
91                                         -- PCM out Gain/attn/mute:
92                                         "0011000" when cmd(2 downto 0) = "100" else
93                                         -- Headphone Volume:
94                                         "0000100" when cmd(2 downto 0) = "101" else
95                                         -- Master volume:
96                                         "0000010" when cmd(2 downto 0) = "110" else
97                                         "0000000";
98
99     -- Set the register data, based on a lookup table:
100     CMD_DATA(15 downto 0) <= "0000000000000000"
101                              when cmd(3) = '1' else
102
103                                     -- Mic Gain:
104                                     micGain(6) &
105                                     "00000000" &
106                                     micGain(5) &
107                                     '0' &
108                                     micGain(4 downto 0)
109                                     when cmd(2 downto 0) = "001" else
110
111                                     -- Rec source:
112                                     "00000" &
113                                     sourceSelect &
114                                     "0000000" &
115                                     sourceSelect &
116                                     "00"
117                                     when cmd(2 downto 0) = "010" else
118
119                                     -- Rec Gain:
120                                     recGainMute(4) &
121                                     "000" &
122                                     recGainMute(3 downto 0) &
123                                     "0000" &
124                                     recGainMute(3 downto 0)
125                                     when cmd(2 downto 0) = "011" else
126
127                                     -- PCM out Gain/attn/mute:
128                                     pcmOutGAM(5) &
129                                     "00" &
130                                     pcmOutGAM(4 downto 0) &
131                                     "000" &
132                                     pcmOutGAM(4 downto 0)
133                                     when cmd(2 downto 0) = "100" else
134
```

```
135                                     -- Headphone Volume:
136                                     headphoneVol(5) &
137                                     "00" &
138                                     headphoneVol (4 downto 0) &
139                                     "000" &
140                                     headphoneVol(4 downto 0)
141                                     when cmd(2 downto 0) = "101" else
142
143                                     -- Master volume:
144                                     masterVol(5) &
145                                     "00" &
146                                     masterVol(4 downto 0) &
147                                     "000" &
148                                     masterVol(4 downto 0)
149                                     when cmd(2 downto 0) = "110" else
150
151                                     "0000000000000000";
152
153 end Behavioral;
```

interfaceDriver2.vhd

```
1  ----------------------------------------------------------------------------
2  -- interfaceDriver2.vhd -------------------------------------------------------
3  -- Author: Anthony Giardina                                                --
4  -- For: Graphic Audio EQ Senior Project                                    --
5  -- Last Modified: 5/29/2012                                                --
6  -- Purpose: This is a top level driver for the peripheral board. It controls --
7  --  the LED driver and the ADC driver.                                     --
8  ----------------------------------------------------------------------------
9  ----------------------------------------------------------------------------
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 use Ieee.std_logic_unsigned.all;
13
14 entity interfaceDriver2 is
15    Port (
16         -- ADC interface signals
17         ADC_DATA_OUT : IN std_logic;
18         ADC_SYSCLK : OUT std_logic;
19         ADC_IOCLK : OUT std_logic;
20         ADC_ADDR_INPUT : OUT std_logic;
21         ADC_CS_C : OUT std_logic;
22         -- LED driver signals.
23         LED_SIN : out  STD_LOGIC;
24         LED_BLANK : out  STD_LOGIC;
25         LED_SCLK : out  STD_LOGIC;
26         LED_LAT : out  STD_LOGIC_VECTOR (4 downto 0);
27         -- Driver reset signal:
28         reset : in  STD_LOGIC;
29         -- 2.048MHZ clock:
30         clk2ish : in  STD_LOGIC;
31         -- port used for debugging:
32         debug: out std_logic_vector (7 downto 0);
33         -- Data from the ADC
34         adc1_out : out std_logic_vector (7 downto 0);
35         adc2_out : out std_logic_vector (7 downto 0);
36         adc3_out : out std_logic_vector (7 downto 0);
37         adc4_out : out std_logic_vector (7 downto 0);
38         adc5_out : out std_logic_vector (7 downto 0);
39         adc6_out : out std_logic_vector (7 downto 0);
40         adc7_out : out std_logic_vector (7 downto 0);
41         adc8_out : out std_logic_vector (7 downto 0);
42         adc9_out : out std_logic_vector (7 downto 0);
43         -- Data sent to the LEDs
44         led1_in : in std_logic_vector (7 downto 0);
45         led2_in : in std_logic_vector (7 downto 0);
46         led3_in : in std_logic_vector (7 downto 0);
47         led4_in : in std_logic_vector (7 downto 0);
48         led5_in : in std_logic_vector (7 downto 0);
49         led6_in : in std_logic_vector (7 downto 0);
50         led7_in : in std_logic_vector (7 downto 0);
51         led8_in : in std_logic_vector (7 downto 0);
52         led9_in : in std_logic_vector (7 downto 0)
53    );
54 end interfaceDriver2;
55
56 architecture Behavioral of interfaceDriver2 is
57
58     COMPONENT ledDriver
59     PORT(
60
61         reset : IN std_logic;
62         clk1mhz : IN std_logic;
63         send : IN std_logic;
64         din : IN std_logic_vector(15 downto 0);
65         driverNum : IN std_logic_vector(2 downto 0);
```

```
66          LED_SIN : OUT std_logic;
67          LED_BLANK : OUT std_logic;
68          LED_SCLK : OUT std_logic;
69          LED_LAT : OUT std_logic_vector(4 downto 0)
70          );
71      END COMPONENT;
72
73      COMPONENT clockDivider
74      PORT(
75          reset : IN std_logic;
76          clk2ish : IN std_logic;
77          clk2mhz : OUT std_logic;
78          clk1mhz : OUT std_logic;
79          updateClk : OUT std_logic
80          );
81      END COMPONENT;
82
83      COMPONENT adcDriver
84      PORT(
85          addressIn : IN std_logic_vector(3 downto 0);
86          send : IN std_logic;
87          clk1mhz : IN std_logic;
88          clk2mhz : IN std_logic;
89          reset : IN std_logic;
90          ADC_DATA_OUT : IN std_logic;
91          dataOut : OUT std_logic_vector(7 downto 0);
92          dataOk : OUT std_logic;
93          ADC_SYSCLK : OUT std_logic;
94          ADC_IOCLK : OUT std_logic;
95          ADC_ADDR_INPUT : OUT std_logic;
96          ADC_CS_C : OUT std_logic
97          );
98      END COMPONENT;
99
100     -- 1.024MHz clock:
101     signal clk1mhz : std_logic;
102     -- 2.048MHz clock:
103     signal clk2mhz: std_logic;
104     -- Clock which defines the period on which all of the ADC/LED Driver
105     -- devices are updated on:
106     signal updateClk: std_logic;
107     -- Control signal used for LED driver:
108     signal led_send : std_logic;
109     -- Data being sent to an individual LED driver:
110     signal led_din : std_logic_vector (15 downto 0);
111     -- Selects which driver the data is being sent to"
112     signal led_driverNum : std_logic_vector  (2 downto 0);
113     -- State variables:
114     signal state : std_logic_vector (3 downto 0);
115     signal count : std_logic_vector (1 downto 0);
116     -- Signals used for rising edge detection:
117     signal lastUclk : std_logic;
118     signal lastUclk2 : std_logic;
119     -- Signals send to the LEDs:
120     signal led1 : std_logic_vector (7 downto 0);
121     signal led2 : std_logic_vector (7 downto 0);
122     signal led3 : std_logic_vector (7 downto 0);
123     signal led4 : std_logic_vector (7 downto 0);
124     signal led5 : std_logic_vector (7 downto 0);
125     signal led6 : std_logic_vector (7 downto 0);
126     signal led7 : std_logic_vector (7 downto 0);
127     signal led8 : std_logic_vector (7 downto 0);
128     signal led9 : std_logic_vector (7 downto 0);
129    -- Signals read back from the ADCs:
130     signal adc1 : std_logic_vector (7 downto 0);
131     signal adc2 : std_logic_vector (7 downto 0);
132     signal adc3 : std_logic_vector (7 downto 0);
133     signal adc4 : std_logic_vector (7 downto 0);
134     signal adc5 : std_logic_vector (7 downto 0);
```

```
135     signal adc6 : std_logic_vector (7 downto 0);
136     signal adc7 : std_logic_vector (7 downto 0);
137     signal adc8 : std_logic_vector (7 downto 0);
138     signal adc9 : std_logic_vector (7 downto 0);
139     -- Control signals for the ADC driver:
140     signal adc_addressIn : std_logic_vector(3 downto 0);
141     signal adc_dataOut : std_logic_vector (7 downto 0);
142     signal adc_dataOk : std_logic;
143     signal adc_send : std_logic;
144     -- Used for rising edge detection:
145     signal lastOk : std_logic;
146 
147 begin
148 
149     -- Optional signals for debugging:
150     debug(0) <= '0';
151     debug(1) <= '0';
152     debug(2) <= '0';
153     debug(3) <= '0';
154     debug(4) <= '0';
155     debug(5) <= '0';
156     debug(6) <= '0';
157     debug(7) <= '0';
158 
159     -- LED Driver Controller ---------------------------------------------------
160     ----------------------------------------------------------------------------
161     -- Set the led data in port to the proper LED values, based on which driver
162     -- is being written to, determined by the state variable:
163     led_din <= not(led1(0) & led1(1) & led1(2) & led1(3) & led1(4) &
164                    led1(5) & led1(6) & led1(7) & led2)
165                         when state = "0000" else
166                    not(led3(0) & led3(1) & led3(2) & led3(3) & led3(4) &
167                        led3(5) & led3(6) & led3(7) & led4)
168                         when state = "0001" else
169                    not(led5(0) & led5(1) & led5(2) & led5(3) & led5(4) &
170                        led5(5) & led5(6) & led5(7) & led6)
171                         when state = "0010" else
172                    not(led7(0) & led7(1) & led7(2) & led7(3) & led7(4) &
173                        led7(5) & led7(6) & led7(7) & led8)
174                         when state = "0011" else
175                    not(("11111111" & led9)) when state = "0100" else
176                    "0000000000000000";
177 
178     -- Select the driver to write data to, based on the state variable:
179     led_driverNum <= "000" when state = "0000" else
180                      "001" when state = "0001" else
181                            "010" when state = "0010" else
182                            "011" when state = "0011" else
183                            "100" when state = "0100" else
184                            "111"; -- Invalid driver num, so we don't accidentally
185                                   -- do something we don't want to do.
186 
187     -- Toggle the send pulse, once data has been placed on the output lines:
188     led_send <= '1' when count = "10" else
189                 '0';
190     ----------------------------------------------------------------------------
191     ----------------------------------------------------------------------------
192 
193     -- ADC Driver Controller ---------------------------------------------------
194     ----------------------------------------------------------------------------
195     -- Select which ADC address to query for data:
196     adc_addressIn <= "0000" when state = "0000" else
197                              "0001" when state = "0001" else
198                              "0010" when state = "0010" else
199                              "0011" when state = "0011" else
200                              "0100" when state = "0100" else
201                              "0101" when state = "0101" else
202                              "0110" when state = "0110" else
203                              "0111" when state = "0111" else
```

```
                        "1000" when state = "1000" else
                        "0000";

    -- Toggle the send signal when data has been placed on the output lines:
     adc_send <= '1' when count = "10" else
                 '0';

     -- This process reads the data from the ADC, which belongs to the address
     -- which was last written to the ADC:
     process (clk1mhz, adc_dataOk, lastOk, adc_dataOut) begin
         if rising_edge(clk1mhz) then
             if ((adc_dataOk = '1') and (lastOk = '0')) then
                 -- Rising edge of dataOk
                 if (state = "0000") then
                     adc9 <= adc_dataOut;
                 elsif (state = "0001") then
                     adc1 <= adc_dataOut;
                 elsif (state = "0010") then
                     adc2 <= adc_dataOut;
                 elsif (state = "0011") then
                     adc3 <= adc_dataOut;
                 elsif (state = "0100") then
                     adc4 <= adc_dataOut;
                 elsif (state = "0101") then
                     adc5 <= adc_dataOut;
                 elsif (state = "0110") then
                     adc6 <= adc_dataOut;
                 elsif (state = "0111") then
                     adc7 <= adc_dataOut;
                 elsif (state = "1000") then
                     adc8 <= adc_dataOut;
                 end if;

                 lastOk <= adc_dataOk;
             else
                 lastOk <= adc_dataOk;
             end if;
         end if;
     end process;
     ----------------------------------------------------------------------------
     ----------------------------------------------------------------------------

    -- This process detects a rising edge of update clock syncronously to the
     -- clk1mhz clock, and sets the "count" state machine into motion, which
     -- controls toggling of the "send" signals to the ADC and LED drivers.
     process (updateClk, count, clk1mhz, reset, lastUclk) begin
         if (rising_edge(clk1mhz)) then
             if (reset = '0') then
                 count <= "11";
                 lastUclk <= '0';
             else
                 if updateClk = '1' and lastUclk = '0' then
                     count <= "00";
                     lastUclk <= updateClk;

                 else
                     lastUclk <= updateClk;
                     if count = "00" then
                         count <= "01";
                     elsif count = "01" then
                         count <= "10";
                     elsif count = "11" then
                         count <= "11";
                     end if;
                 end if;
             end if;
         end if;
     end process;
```

```
    -- Process to control the state variable:
    process (updateClk, state, reset) begin
        if (reset = '0') then
            state <= "0000";
        elsif (rising_edge(updateClk)) then
            if (state = "1000") then
                state <= "0000";
            else
                state <= state + "0001";
            end if;
        end if;
    end process;

    -- Instantiate ADC driver:
    Inst_adcDriver: adcDriver PORT MAP(
        addressIn => adc_addressIn,
        dataOut => adc_dataOut,
        dataOk => adc_dataOk,
        send => adc_send,
        clk1mhz => clk1mhz,
        clk2mhz => clk2mhz,
        reset => reset,
        ADC_SYSCLK => ADC_SYSCLK,
        ADC_IOCLK => ADC_IOCLK,
        ADC_ADDR_INPUT => ADC_ADDR_INPUT,
        ADC_DATA_OUT => ADC_DATA_OUT,
        ADC_CS_C => ADC_CS_C
    );

    -- Instantiate LED driver:
    Inst_ledDriver: ledDriver PORT MAP(
        reset => reset,
        LED_SIN => LED_SIN,
        LED_BLANK => LED_BLANK,
        LED_SCLK => LED_SCLK,
        LED_LAT => LED_LAT,
        clk1mhz => clk1mhz,
        send => led_send,
        din => led_din,
        driverNum => led_driverNum
    );

    -- Instantiate clock divider, to divide down clocks:
    Inst_clockDivider: clockDivider PORT MAP(
        reset => reset,
        clk2ish => clk2ish,    --2.048MHz input clock
        clk2mhz => clk2mhz,    --2.048MHz output clock
        clk1mhz => clk1mhz,    --1.024MHz output clock
        updateClk => updateClk -- update syncronizing output clock
    );

    -- Connect signals together:
    led1 <= led1_in;
    led2 <= led2_in;
    led3 <= led3_in;
    led4 <= led4_in;
    led5 <= led5_in;
    led6 <= led6_in;
    led7 <= led7_in;
    led8 <= led8_in;
    led9 <= led9_in;

   -- Connect signals together:
    adc1_out <= adc1;
    adc2_out <= adc2;
    adc3_out <= adc3;
    adc4_out <= adc4;
    adc5_out <= adc5;
    adc6_out <= adc6;
```

```
342      adc7_out <= adc7;
343      adc8_out <= adc8;
344      adc9_out <= adc9;
345
346  end Behavioral;
```

adcDriver.vhd

```
1  ----------------------------------------------------------------------------
2  -- adcDriver.vhd -----------------------------------------------------------
3  -- Author: Anthony Giardina                                                --
4  -- For: Graphic Audio EQ Senior Project                                    --
5  -- Last Modified: 5/29/2012                                                --
6  -- Purpose: This module controls the TLC541L ADC at the lowest "bit bang"  --
7  --  level. Itsends ADC addressses to the ADC, and reads back data from the --
8  --  previous address sent.                                                 --
9  ----------------------------------------------------------------------------
10 ----------------------------------------------------------------------------
11
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 use Ieee.std_logic_unsigned.all;
15 use IEEE.NUMERIC_STD.ALL;
16
17 entity adcDriver is
18     Port (
19         -- addressIn is the ADC analog channel that will be read on the next cycle
20         addressIn : in  STD_LOGIC_VECTOR (3 downto 0);
21         -- dataOut is the ADC converison data that belongs to the previously
22         -- send ADC analog channel.
23         dataOut : out  STD_LOGIC_VECTOR (7 downto 0);
24         -- dataOk goes high when the conversion data on the output
25         -- bus is valid.
26         dataOk : out  STD_LOGIC;
27         -- on the rising edge of send, the driver begins serial
28         -- communication with the device:
29         send : in  STD_LOGIC;
30         -- System input clocks:
31         clk1mhz : in  STD_LOGIC;
32         clk2mhz : in  STD_LOGIC;
33         -- System Reset:
34         reset : in  STD_LOGIC;
35         -- Hardware input/output signals for the serial bus:
36         ADC_SYSCLK : out  STD_LOGIC;
37         ADC_IOCLK : out  STD_LOGIC;
38         ADC_ADDR_INPUT : out  STD_LOGIC;
39         ADC_DATA_OUT : in  STD_LOGIC;
40         ADC_CS_C : out  STD_LOGIC
41     );
42 end adcDriver;
43
44 architecture Behavioral of adcDriver is
45
46     -- state machine variable:
47     signal state: std_logic_vector (4 downto 0);
48     -- latch the input address, so it doesn't get changed durring
49     -- device communication:
50     signal latchAddressIn: std_logic_vector (3 downto 0);
51     -- ADC output data is assembled on this signal before it
52     -- is latched onto the output register, so that incomplete data
53     -- is not placed on the output:
54     signal dataOutInt: std_logic_vector (7 downto 0);
55     -- dummy signal:
56     signal a: std_logic;
57     -- signal used for syncronouse detection of the send signal:
58     signal lastSend: std_logic;
59
60 begin
61
62     -- 2mhz clock always goes to the ADC, unless in a reset state.
63     ADC_SYSCLK <= clk2mhz when reset = '1' else
64                   '0';
65     -- 1mhz clock goes to the ADC for 8 cycles of data communication:
```

```
    ADC_IOCLK <= not clk1mhz when state >= "00011" and state <= "01010" else
                    '0';
    -- drive the chip select signal:
    ADC_CS_C <= '0' when state >= "00001" and state <= "01010" else
                   '1';
    -- place the address bits serially on the serial bus:
    ADC_ADDR_INPUT <= latchAddressIn(3) when state = "00011" else
                          latchAddressIn(2) when state = "00100" else
                          latchAddressIn(1) when state = "00101" else
                          latchAddressIn(0) when state = "00110" else
                          '0';

    -- This process detects the a rising edge of the state signal,
    -- syncronously to the clk1mhz signal, and controls operation
    -- of the state machine:
    process (clk1mhz,reset,send, state, ADC_DATA_OUT) begin
        if rising_edge(clk1mhz) then
            if (send = '1' and lastSend = '0') then
                latchAddressIn <= addressIn;
                state <= "00001";
            end if;
            lastSend <= send;
            if state = "01100" then
                dataOut <= dataOutInt;
                dataOK <= '1';
            end if;
            if state = "00001" then
                dataOK <= '0';
            end if;
            if state = "00000" then
            elsif state = "11111" then
                state <= "00000";
            else
                state <= state + "00001";
            end if;
        end if;
    end process;

    -- This process reads the data coming from the device over the
    -- serial bus on the falling edge of the 1MHz clock:
    process(clk1mhz, state, ADC_DATA_OUT) begin
        if falling_edge(clk1mhz) then
            if (state = "00011" ) then
                dataOutInt(7) <= ADC_DATA_OUT;
            elsif (state = "00100") then
                dataOutInt(6) <= ADC_DATA_OUT;
            elsif (state = "00101") then
                dataOutInt(5) <= ADC_DATA_OUT;
            elsif (state = "00110") then
                dataOutInt(4) <= ADC_DATA_OUT;
            elsif (state = "00111") then
                dataOutInt(3) <= ADC_DATA_OUT;
            elsif (state = "01000") then
                dataOutInt(2) <= ADC_DATA_OUT;
            elsif (state = "01001") then
                dataOutInt(1) <= ADC_DATA_OUT;
            elsif (state = "01010") then
                dataOutInt(0) <= ADC_DATA_OUT;
            end if;
        end if;
    end process;

end Behavioral;
```

ledDriver.vhd

```
1  ----------------------------------------------------------------------------
2  -- ledDriver.vhd ----------------------------------------------------------
3  -- Author: Anthony Giardina                                               --
4  -- For: Graphic Audio EQ Senior Project                                   --
5  -- Last Modified: 5/29/2012                                               --
6  -- Purpose: This is the low level bit banging driver module for the TLC59281 --
7  --  LED driving IC. It is designed to be used on a serial bus with 5 devices --
8  --  and uses a TDM technique to control all of the drivers on the same bus.  --
9  ----------------------------------------------------------------------------
10 ----------------------------------------------------------------------------
11
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 use Ieee.std_logic_unsigned.all;
15 use IEEE.NUMERIC_STD.ALL;
16
17
18 entity ledDriver is
19     Port (
20         -- System reset:
21         reset: in std_logic;
22         -- Hardware serial bus signals:
23         LED_SIN : out  STD_LOGIC;
24         LED_BLANK : out  STD_LOGIC;
25         LED_SCLK : out  STD_LOGIC;
26         LED_LAT : out  STD_LOGIC_VECTOR (4 downto 0);
27         -- Input clock:
28         clk1mhz : in  STD_LOGIC;
29         -- Control signals:
30         -- A rising edge of send signals to the ledDriver module to
31         -- begin sending data on the serial bus.
32         send : in  STD_LOGIC;
33         -- din is the data that will be sent to the device.
34         din : in  STD_LOGIC_VECTOR (15 downto 0);
35         -- driverNum dictates which driver the data contained on the
36         -- din signal will go to. Can have values of 000 to 100.
37         driverNum : in  STD_LOGIC_VECTOR (2 downto 0)
38     );
39 end ledDriver;
40
41 architecture Behavioral of ledDriver is
42
43     -- State variable to control operation of the driver:
44     signal state : std_logic_vector (4 downto 0);
45     -- Internal register, to latch the incoming data on the rising edge of
46     -- send, so that it does not get corrupted durring sending:
47     signal latchDin : std_logic_vector (15 downto 0);
48     -- Decoded "one hot" version of driverNum input signal:
49     signal bankSelect : std_logic_vector (4 downto 0);
50     -- signal used for send signal edge detection:
51     signal lastSend : std_logic;
52
53 begin
54
55     -- Active low reset coming in, device has active high reset.
56     LED_BLANK <= not reset;
57
58     -- Device serial clock is active for 16 cycles:
59     LED_SCLK <= clk1mhz when (state >= "00010" and state <= "10001") else
60                 '0';
61
62     -- Decode the driverNum signal into a one-hot version, used to control
63     -- the LATCH signal on every device:
64     bankSelect <= "00001" when driverNum = "000" else
65                   "00010" when driverNum = "001" else
```

```
                    "00100" when driverNum = "010" else
                    "01000" when driverNum = "011" else
                    "10000" when driverNum = "100" else
                    "00000";
    -- Drive the LATCH signal with the bankSelect signal, unless it's a reset:
    LED_LAT <= "00000" when reset = '0' else
                bankSelect when state = "10011" else
                "00000";
    -- This process controls the state machine, and also puts the data bits on
    -- on the serial bus at the correct time. The state machine is set in
    -- motion by a rising edge of send, which is syncronously detected on the
    -- clk1mhz clock.
    process (reset, clk1mhz,send, state, din, lastSend) begin
        if (falling_edge(clk1mhz)) then
            if (send = '1' and lastSend = '0') then
                latchDin <= din;
                state <= "00001";
            end if;

            lastSend <= send;

            case state is
                when "00001" => LED_SIN <= latchDin(15);
                when "00010" => LED_SIN <= latchDin(14);
                when "00011" => LED_SIN <= latchDin(13);
                when "00100" => LED_SIN <= latchDin(12);
                when "00101" => LED_SIN <= latchDin(11);
                when "00110" => LED_SIN <= latchDin(10);
                when "00111" => LED_SIN <= latchDin(9);
                when "01000" => LED_SIN <= latchDin(8);
                when "01001" => LED_SIN <= latchDin(7);
                when "01010" => LED_SIN <= latchDin(6);
                when "01011" => LED_SIN <= latchDin(5);
                when "01100" => LED_SIN <= latchDin(4);
                when "01101" => LED_SIN <= latchDin(3);
                when "01110" => LED_SIN <= latchDin(2);
                when "01111" => LED_SIN <= latchDin(1);
                when "10000" => LED_SIN <= latchDin(0);
                when  others => LED_SIN <= '0';
            end case;

            if state >= "00001" and state <= "10011" then
                state <= state + "00001";
            elsif state >= "10100" then
                state <= "00000";
            end if;

        end if;
    end process;

end Behavioral;
```

clockDivider.vhd

```
1  ----------------------------------------------------------------------------
2  -- clockDivider.vhd -------------------------------------------------------
3  -- Author: Anthony Giardina                                               --
4  -- For: Graphic Audio EQ Senior Project                                   --
5  -- Last Modified: 5/29/2012                                               --
6  -- Purpose: This module takes in a 2.048 50% duty cycle clock, and produces  --
7  --  the following 3 output clocks:                                        --
8  --  > 2.048MHz, 50% duty cycle                                            --
9  --  > 1.024MHz, 50% duty cycle                                            --
10 --  > 4.000KHz, 50% duty cycle                                            --
11 ----------------------------------------------------------------------------
12 ----------------------------------------------------------------------------
13
14 library IEEE;
15 use IEEE.STD_LOGIC_1164.ALL;
16 use Ieee.std_logic_unsigned.all;
17 use IEEE.NUMERIC_STD.ALL;
18
19 entity clockDivider is
20     Port (
21         -- System reset:
22         reset : in STD_LOGIC;
23         -- Input clock:
24         clk2ish : in  STD_LOGIC;
25         -- Output clocks:
26         clk2mhz : out  STD_LOGIC;
27         clk1mhz : out  STD_LOGIC;
28         updateClk : out  STD_LOGIC
29     );
30 end clockDivider;
31
32 architecture Behavioral of clockDivider is
33
34 -- State is a large counter that is used as a clock divider.
35 signal state : std_logic_vector (9 downto 0);
36
37 begin
38
39 -- Counter to increment the state signal by one on every rising edge
40 -- of the 2MHz clock. Only reset once all of the bits are 1, so that
41 -- the duty cycle of every bit is 50%.
42 process (clk2ish, reset) begin
43     if (reset = '0') then
44         state <= "0000000000";
45     elsif (rising_edge(clk2ish)) then
46         if state = "1111111111" then
47             state <= "0000000000";
48         else
49             state <= state + "0000000001";
50         end if;
51     end if;
52 end process;
53
54 -- Assign clocks to bits of the counter.
55 updateClk <= state(8);
56 clk1mhz <= state(0);
57 clk2mhz <= clk2ish;
58
59 end Behavioral;
```

gainAtten.vhd

```
1  ----------------------------------------------------------------------------
2  -- gainAtten.vhd -----------------------------------------------------------
3  -- Author: Anthony Giardina                                               --
4  -- For: Graphic Audio EQ Senior Project                                   --
5  -- Last Modified: 5/29/2012                                               --
6  -- Purpose: This module provides gain or attenuation to the twos-compliment  --
7  --  PCM codes that come out of each of the band filters, based on the input  --
8  --  PCM value on the pcmInput signal. The values are scaled based on the     --
9  --  lut.vhd lookup table.                                                  --
10 ----------------------------------------------------------------------------
11 ----------------------------------------------------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14
15 entity gainAtten is
16     Port (
17         -- Input PCM value, which will be attenuated or amplified:
18         input : in  STD_LOGIC_VECTOR (17 downto 0);
19         -- Factor by which the input signal will be scaled:
20         pcmInput : in  STD_LOGIC_VECTOR (7 downto 0);
21         -- scaled output:
22         output : out  STD_LOGIC_VECTOR (17 downto 0)
23     );
24 end gainAtten;
25
26 architecture Behavioral of gainAtten is
27
28     -- output of the lookup table:
29     signal lutValue : std_logic_vector (9 downto 0);
30     -- output of the multiplier:
31     signal multOutput : std_logic_vector (27 downto 0);
32
33
34     COMPONENT multiplier
35       PORT (
36          a : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
37          b : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
38          p : OUT STD_LOGIC_VECTOR(27 DOWNTO 0)
39       );
40     END COMPONENT;
41
42
43     COMPONENT lut
44     PORT(
45         din : IN std_logic_vector(7 downto 0);
46         dout : OUT std_logic_vector(9 downto 0)
47         );
48     END COMPONENT;
49
50 begin
51
52     -- Instantiate multiplier, used for multiplying the input
53     -- value and the scaling value together:
54     your_instance_name : multiplier
55       PORT MAP (
56          a => input,
57          b => lutValue,
58          p => multOutput
59       );
60
61     -- Instantiate lookup table. The lookup table takes in  an 8
62     -- bit value , and outputs scaled integer values on a logarithmic
63     -- scale, which will be used for multiplication.
64     Inst_lut: lut PORT MAP(
65         din => pcmInput,
```

```
66          dout => lutValue
67      );
68
69     -- Shift over by 8 to undo scale factor
70      output(17 downto 0) <= multOutput (27 downto 10);
71
72  end Behavioral;
```

lut.vhd

```
-- Generated by matlab script divisionError.m
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lut is
    Port ( din : in  STD_LOGIC_VECTOR (7 downto 0);
           dout : out  STD_LOGIC_VECTOR (9 downto 0));
end lut;

architecture Behavioral of lut is

begin

dout <=
        "0001000100" when din = "11111111" else
        "0001000101" when din = "11111110" else
        "0001000110" when din = "11111101" else
        "0001000110" when din = "11111100" else
        "0001000111" when din = "11111011" else
        "0001001000" when din = "11111010" else
        "0001001001" when din = "11111001" else
        "0001001001" when din = "11111000" else
        "0001001010" when din = "11110111" else
        "0001001011" when din = "11110110" else
        "0001001100" when din = "11110101" else
        "0001001100" when din = "11110100" else
        "0001001101" when din = "11110011" else
        "0001001110" when din = "11110010" else
        "0001001111" when din = "11110001" else
        "0001010000" when din = "11110000" else
        "0001010001" when din = "11101111" else
        "0001010001" when din = "11101110" else
        "0001010010" when din = "11101101" else
        "0001010011" when din = "11101100" else
        "0001010100" when din = "11101011" else
        "0001010101" when din = "11101010" else
        "0001010110" when din = "11101001" else
        "0001010111" when din = "11101000" else
        "0001011000" when din = "11100111" else
        "0001011000" when din = "11100110" else
        "0001011001" when din = "11100101" else
        "0001011010" when din = "11100100" else
        "0001011011" when din = "11100011" else
        "0001011100" when din = "11100010" else
        "0001011101" when din = "11100001" else
        "0001011110" when din = "11100000" else
        "0001011111" when din = "11011111" else
        "0001100000" when din = "11011110" else
        "0001100001" when din = "11011101" else
        "0001100010" when din = "11011100" else
        "0001100011" when din = "11011011" else
        "0001100100" when din = "11011010" else
        "0001100101" when din = "11011001" else
        "0001100110" when din = "11011000" else
        "0001100111" when din = "11010111" else
        "0001101001" when din = "11010110" else
        "0001101010" when din = "11010101" else
        "0001101011" when din = "11010100" else
        "0001101100" when din = "11010011" else
        "0001101101" when din = "11010010" else
        "0001101110" when din = "11010001" else
        "0001101111" when din = "11010000" else
        "0001110000" when din = "11001111" else
        "0001110010" when din = "11001110" else
        "0001110011" when din = "11001101" else
```

```
66          "0001110100" when din = "11001100" else
67          "0001110101" when din = "11001011" else
68          "0001110110" when din = "11001010" else
69          "0001111000" when din = "11001001" else
70          "0001111001" when din = "11001000" else
71          "0001111010" when din = "11000111" else
72          "0001111011" when din = "11000110" else
73          "0001111101" when din = "11000101" else
74          "0001111110" when din = "11000100" else
75          "0001111111" when din = "11000011" else
76          "0010000001" when din = "11000010" else
77          "0010000010" when din = "11000001" else
78          "0010000011" when din = "11000000" else
79          "0010000101" when din = "10111111" else
80          "0010000110" when din = "10111110" else
81          "0010001000" when din = "10111101" else
82          "0010001001" when din = "10111100" else
83          "0010001010" when din = "10111011" else
84          "0010001100" when din = "10111010" else
85          "0010001101" when din = "10111001" else
86          "0010001111" when din = "10111000" else
87          "0010010000" when din = "10110111" else
88          "0010010010" when din = "10110110" else
89          "0010010011" when din = "10110101" else
90          "0010010101" when din = "10110100" else
91          "0010010110" when din = "10110011" else
92          "0010011000" when din = "10110010" else
93          "0010011010" when din = "10110001" else
94          "0010011011" when din = "10110000" else
95          "0010011101" when din = "10101111" else
96          "0010011110" when din = "10101110" else
97          "0010100000" when din = "10101101" else
98          "0010100010" when din = "10101100" else
99          "0010100011" when din = "10101011" else
100         "0010100101" when din = "10101010" else
101         "0010100111" when din = "10101001" else
102         "0010101001" when din = "10101000" else
103         "0010101010" when din = "10100111" else
104         "0010101100" when din = "10100110" else
105         "0010101110" when din = "10100101" else
106         "0010110000" when din = "10100100" else
107         "0010110010" when din = "10100011" else
108         "0010110011" when din = "10100010" else
109         "0010110101" when din = "10100001" else
110         "0010110111" when din = "10100000" else
111         "0010111001" when din = "10011111" else
112         "0010111011" when din = "10011110" else
113         "0010111101" when din = "10011101" else
114         "0010111111" when din = "10011100" else
115         "0011000001" when din = "10011011" else
116         "0011000011" when din = "10011010" else
117         "0011000101" when din = "10011001" else
118         "0011000111" when din = "10011000" else
119         "0011001001" when din = "10010111" else
120         "0011001011" when din = "10010110" else
121         "0011001101" when din = "10010101" else
122         "0011010000" when din = "10010100" else
123         "0011010010" when din = "10010011" else
124         "0011010100" when din = "10010010" else
125         "0011010110" when din = "10010001" else
126         "0011011000" when din = "10010000" else
127         "0011011011" when din = "10001111" else
128         "0011011101" when din = "10001110" else
129         "0011011111" when din = "10001101" else
130         "0011100010" when din = "10001100" else
131         "0011100100" when din = "10001011" else
132         "0011100110" when din = "10001010" else
133         "0011101001" when din = "10001001" else
134         "0011101011" when din = "10001000" else
```

```
135         "0011101110" when din = "10000111" else
136         "0011110000" when din = "10000110" else
137         "0011110011" when din = "10000101" else
138         "0011110101" when din = "10000100" else
139         "0011111000" when din = "10000011" else
140         "0011111010" when din = "10000010" else
141         "0011111101" when din = "10000001" else
142         "0100000000" when din = "10000000" else
143
144         "0100000000" when din = "01111111" else
145         "0100000010" when din = "01111110" else
146         "0100000101" when din = "01111101" else
147         "0100001000" when din = "01111100" else
148         "0100001010" when din = "01111011" else
149         "0100001101" when din = "01111010" else
150         "0100010000" when din = "01111001" else
151         "0100010011" when din = "01111000" else
152         "0100010110" when din = "01110111" else
153         "0100011001" when din = "01110110" else
154         "0100011011" when din = "01110101" else
155         "0100011110" when din = "01110100" else
156         "0100100001" when din = "01110011" else
157         "0100100100" when din = "01110010" else
158         "0100100111" when din = "01110001" else
159         "0100101011" when din = "01110000" else
160         "0100101110" when din = "01101111" else
161         "0100110001" when din = "01101110" else
162         "0100110100" when din = "01101101" else
163         "0100110111" when din = "01101100" else
164         "0100111010" when din = "01101011" else
165         "0100111110" when din = "01101010" else
166         "0101000001" when din = "01101001" else
167         "0101000100" when din = "01101000" else
168         "0101001000" when din = "01100111" else
169         "0101001011" when din = "01100110" else
170         "0101001111" when din = "01100101" else
171         "0101010010" when din = "01100100" else
172         "0101010110" when din = "01100011" else
173         "0101011001" when din = "01100010" else
174         "0101011101" when din = "01100001" else
175         "0101100000" when din = "01100000" else
176         "0101100100" when din = "01011111" else
177         "0101101000" when din = "01011110" else
178         "0101101100" when din = "01011101" else
179         "0101101111" when din = "01011100" else
180         "0101110011" when din = "01011011" else
181         "0101110111" when din = "01011010" else
182         "0101111011" when din = "01011001" else
183         "0101111111" when din = "01011000" else
184         "0110000011" when din = "01010111" else
185         "0110000111" when din = "01010110" else
186         "0110001011" when din = "01010101" else
187         "0110001111" when din = "01010100" else
188         "0110010011" when din = "01010011" else
189         "0110011000" when din = "01010010" else
190         "0110011100" when din = "01010001" else
191         "0110100000" when din = "01010000" else
192         "0110100100" when din = "01001111" else
193         "0110101001" when din = "01001110" else
194         "0110101101" when din = "01001101" else
195         "0110110010" when din = "01001100" else
196         "0110110110" when din = "01001011" else
197         "0110111011" when din = "01001010" else
198         "0110111111" when din = "01001001" else
199         "0111000100" when din = "01001000" else
200         "0111001001" when din = "01000111" else
201         "0111001110" when din = "01000110" else
202         "0111010010" when din = "01000101" else
203         "0111010111" when din = "01000100" else
```

```
        "0111011100" when din = "01000011" else
        "0111100001" when din = "01000010" else
        "0111100110" when din = "01000001" else
        "0111101011" when din = "01000000" else
        "0111110000" when din = "00111111" else
        "0111110110" when din = "00111110" else
        "0111111011" when din = "00111101" else
        "1000000000" when din = "00111100" else
        "1000000101" when din = "00111011" else
        "1000001011" when din = "00111010" else
        "1000010000" when din = "00111001" else
        "1000010110" when din = "00111000" else
        "1000011011" when din = "00110111" else
        "1000100001" when din = "00110110" else
        "1000100111" when din = "00110101" else
        "1000101100" when din = "00110100" else
        "1000110010" when din = "00110011" else
        "1000111000" when din = "00110010" else
        "1000111110" when din = "00110001" else
        "1001000100" when din = "00110000" else
        "1001001010" when din = "00101111" else
        "1001010000" when din = "00101110" else
        "1001010110" when din = "00101101" else
        "1001011100" when din = "00101100" else
        "1001100011" when din = "00101011" else
        "1001101001" when din = "00101010" else
        "1001110000" when din = "00101001" else
        "1001110110" when din = "00101000" else
        "1001111101" when din = "00100111" else
        "1010000011" when din = "00100110" else
        "1010001010" when din = "00100101" else
        "1010010001" when din = "00100100" else
        "1010011000" when din = "00100011" else
        "1010011111" when din = "00100010" else
        "1010100110" when din = "00100001" else
        "1010101101" when din = "00100000" else
        "1010110100" when din = "00011111" else
        "1010111011" when din = "00011110" else
        "1011000010" when din = "00011101" else
        "1011001010" when din = "00011100" else
        "1011010001" when din = "00011011" else
        "1011011001" when din = "00011010" else
        "1011100000" when din = "00011001" else
        "1011101000" when din = "00011000" else
        "1011110000" when din = "00010111" else
        "1011110111" when din = "00010110" else
        "1011111111" when din = "00010101" else
        "1100000111" when din = "00010100" else
        "1100001111" when din = "00010011" else
        "1100011000" when din = "00010010" else
        "1100100000" when din = "00010001" else
        "1100101000" when din = "00010000" else
        "1100110001" when din = "00001111" else
        "1100111001" when din = "00001110" else
        "1101000010" when din = "00001101" else
        "1101001010" when din = "00001100" else
        "1101010011" when din = "00001011" else
        "1101011100" when din = "00001010" else
        "1101100101" when din = "00001001" else
        "1101101110" when din = "00001000" else
        "1101110111" when din = "00000111" else
        "1110000000" when din = "00000110" else
        "1110001010" when din = "00000101" else
        "1110010011" when din = "00000100" else
        "1110011101" when din = "00000011" else
        "1110100110" when din = "00000010" else
        "1110110000" when din = "00000001" else
        "1110111010" when din = "00000000" else
        "0100000000";
```

```
273
274 end Behavioral;
```

summer9x2ch.vhd

```
1  ----------------------------------------------------------------------------
2  -- summer9x2ch.vhd ------------------------------------------------------------
3  -- Author: Anthony Giardina                                                --
4  -- For: Graphic Audio EQ Senior Project                                     --
5  -- Last Modified: 5/29/2012                                                 --
6  -- Purpose: This module takes in 18 twos compliment signals (9 per channel)  --
7  --  and generates two sums (one per channel).                               --
8  ----------------------------------------------------------------------------
9  ----------------------------------------------------------------------------
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12
13 entity summer9x2ch is
14     Port ( clock : in  STD_LOGIC;
15                in1_r : in  STD_LOGIC_VECTOR (17 downto 0);
16            in1_l : in  STD_LOGIC_VECTOR (17 downto 0);
17            in2_r : in  STD_LOGIC_VECTOR (17 downto 0);
18            in2_l : in  STD_LOGIC_VECTOR (17 downto 0);
19            in3_r : in  STD_LOGIC_VECTOR (17 downto 0);
20            in3_l : in  STD_LOGIC_VECTOR (17 downto 0);
21            in4_r : in  STD_LOGIC_VECTOR (17 downto 0);
22            in4_l : in  STD_LOGIC_VECTOR (17 downto 0);
23            in5_r : in  STD_LOGIC_VECTOR (17 downto 0);
24            in5_l : in  STD_LOGIC_VECTOR (17 downto 0);
25            in6_r : in  STD_LOGIC_VECTOR (17 downto 0);
26            in6_l : in  STD_LOGIC_VECTOR (17 downto 0);
27            in7_r : in  STD_LOGIC_VECTOR (17 downto 0);
28            in7_l : in  STD_LOGIC_VECTOR (17 downto 0);
29            in8_r : in  STD_LOGIC_VECTOR (17 downto 0);
30            in8_l : in  STD_LOGIC_VECTOR (17 downto 0);
31            in9_r : in  STD_LOGIC_VECTOR (17 downto 0);
32            in9_l : in  STD_LOGIC_VECTOR (17 downto 0);
33            out_r : out  STD_LOGIC_VECTOR (17 downto 0);
34            out_l : out  STD_LOGIC_VECTOR (17 downto 0));
35 end summer9x2ch;
36
37 architecture Behavioral of summer9x2ch is
38
39     -- Intermediate signals used to connect everything:
40     signal s10_r : std_logic_vector(17 downto 0);
41     signal s11_r : std_logic_vector(17 downto 0);
42     signal s12_r : std_logic_vector(17 downto 0);
43     signal s13_r : std_logic_vector(17 downto 0);
44     signal s14_r : std_logic_vector(17 downto 0);
45     signal s15_r : std_logic_vector(17 downto 0);
46     signal s16_r : std_logic_vector(17 downto 0);
47
48     signal s10_l : std_logic_vector(17 downto 0);
49     signal s11_l : std_logic_vector(17 downto 0);
50     signal s12_l : std_logic_vector(17 downto 0);
51     signal s13_l : std_logic_vector(17 downto 0);
52     signal s14_l : std_logic_vector(17 downto 0);
53     signal s15_l : std_logic_vector(17 downto 0);
54     signal s16_l : std_logic_vector(17 downto 0);
55
56 COMPONENT adder
57   PORT (
58     a : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
59     b : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
60     clk : IN STD_LOGIC;
61     s : OUT STD_LOGIC_VECTOR(17 DOWNTO 0)
62   );
63 END COMPONENT;
64
```

```
65      signal r : std_logic_vector (17 downto 0);
66      signal l : std_logic_vector (17 downto 0);
67
68
69  begin
70
71      -- Outputs are right shifted by 3:
72      out_r <= r(17) &  r(13) & r(12) & r(11) & r(10) & r(9) & r(8) & r(7) &
73               r(6) & r(5) & r(4) & r(3) & r(2) & r(1) & r(0) & r(0) & r(0) &
74                   r(0);
75      out_l <= l(17) &  l(13) & l(12) & l(11) & l(10) & l(9) & l(8) & l(7) &
76               l(6) & l(5) & l(4) & l(3) & l(2) & l(1) & l(0) & l(0) & l(0) &
77                   l(0);
78
79     -- This giant pile of 2-input adders sum all of the input signals together:
80      s1_l : adder
81        PORT MAP (
82           a => in1_l,
83           b => in2_l,
84           clk => clock,
85           s => s10_l
86        );
87
88      s1_r : adder
89        PORT MAP (
90           a => in1_r,
91           b => in2_r,
92           clk => clock,
93           s => s10_r
94        );
95     --
96      s2_l : adder
97        PORT MAP (
98           a => in3_l,
99           b => in4_l,
100          clk => clock,
101          s => s11_l
102       );
103
104     s2_r : adder
105       PORT MAP (
106          a => in3_r,
107          b => in4_r,
108          clk => clock,
109          s => s11_r
110       );
111     --
112     s3_l : adder
113       PORT MAP (
114          a => in5_l,
115          b => in6_l,
116          clk => clock,
117          s => s12_l
118       );
119
120     s3_r : adder
121       PORT MAP (
122          a => in5_r,
123          b => in6_r,
124          clk => clock,
125          s => s12_r
126       );
127    --
128     s4_l : adder
129       PORT MAP (
130          a => in7_l,
131          b => in8_l,
132          clk => clock,
133          s => s13_l
```

```
134       );
135
136     s4_r : adder
137       PORT MAP (
138          a => in7_r,
139          b => in8_r,
140          clk => clock,
141          s => s13_r
142       );
143     --
144     s5_l : adder
145       PORT MAP (
146          a => s10_l,
147          b => s11_l,
148          clk => clock,
149          s => s14_l
150       );
151
152     s5_r : adder
153       PORT MAP (
154          a => s10_r,
155          b => s11_r,
156          clk => clock,
157          s => s14_r
158       );
159     --
160     s6_l : adder
161       PORT MAP (
162          a => s12_l,
163          b => s13_l,
164          clk => clock,
165          s => s15_l
166       );
167
168     s6_r : adder
169       PORT MAP (
170          a => s12_r,
171          b => s13_r,
172          clk => clock,
173          s => s15_r
174       );
175     --
176     s7_l : adder
177       PORT MAP (
178          a => s14_l,
179          b => s15_l,
180          clk => clock,
181          s => s16_l
182       );
183
184     s7_r : adder
185       PORT MAP (
186          a => s14_r,
187          b => s15_r,
188          clk => clock,
189          s => s16_r
190       );
191     --
192     s8_l : adder
193       PORT MAP (
194          a => s16_l,
195          b => in9_l,
196          clk => clock,
197          s => l
198       );
199
200     s8_r : adder
201       PORT MAP (
202          a => s16_r,
```

```
203             b => in9_r,
204             clk => clock,
205             s => r
206         );
207 
208 end Behavioral;
```

ledLut.vhd

```
1  ----------------------------------------------------------------------------------
2  -- ledLut.vhd -----------------------------------------------------------------
3  -- Author: Anthony Giardina                                                   --
4  -- For: Graphic Audio EQ Senior Project                                       --
5  -- Last Modified: 5/29/2012                                                   --
6  -- Purpose: This module converts 18-bit twos-compliment values and converts  --
7  --  them to values which can be displayed on the LED banks, representing     --
8  --  the amplitude of the input value.                                        --
9  ----------------------------------------------------------------------------------
10 ----------------------------------------------------------------------------------
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13 use Ieee.std_logic_unsigned.all;
14 use IEEE.NUMERIC_STD.ALL;
15
16 entity ledLut is
17     Port ( bandPcm : in  STD_LOGIC_VECTOR (17 downto 0);
18            ledOut : out  STD_LOGIC_VECTOR (7 downto 0));
19 end ledLut;
20
21 architecture Behavioral of ledLut is
22
23     -- Sign bit of input twos-compliment number:
24     signal s : std_logic;
25     -- the rest of the bits, without the sign bit:
26     signal d : std_logic_vector (16 downto 0);
27
28 begin
29     -- Sign bit of twos compliment PCM value
30     s <= bandPcm (17);
31     -- left shift by 2 to get some additional range on the LEDs
32     d <=  bandPcm(14) & bandPcm(13) & bandPcm(12) & bandPcm(11) & bandPcm(10) &
33             bandPcm(9) & bandPcm(8) & bandPcm(7) & bandPcm(6) & bandPcm(5) &
34             bandPcm(4) & bandPcm(3) & bandPcm(2) & bandPcm(1) & bandPcm(0) &
35             bandPcm(0) & bandPcm(0);
36
37     -- This is a lookup table. The values were more or less experimentally
38     -- determined for best appearance on the display:
39     ledOut <= "00000000" when ((s = '1') and ( d <= "00000000000000011"))
40                                 or ((s = '0') and ( d >= "10000000000000000"))
41                                 else
42               "00000001" when ((s = '1') and ( d <= "00000000000000011"))
43                                 or ((s = '0') and ((d <  "10000000000000000")
44                                 and (d >= "00100000000000000")))
45                                 else
46               "00000011" when ((s = '1') and ((d >  "00000000000000011")
47                                 and (d <= "00000000000011111"))) or ((s = '0')
48                                 and ((d <  "00100000000000000")
49                                 and (d >= "00001000000000000")))
50                                 else
51               "00000111" when ((s = '1') and ((d >  "00000000000011111")
52                                 and (d <= "00000000001111111"))) or ((s = '0')
53                                 and ((d <  "00001000000000000")
54                                 and (d >= "00000010000000000")))
55                                 else
56               "00001111" when ((s = '1') and ((d >  "00000000001111111")
57                                 and (d <= "00000001111111111"))) or ((s = '0')
58                                 and ((d <  "00000010000000000")
59                                 and (d >= "00000000100000000")))
60                                 else
61               "00011111" when ((s = '1') and ((d >  "00000001111111111")
62                                 and (d <= "00000111111111111"))) or ((s = '0')
63                                 and ((d <  "00000000100000000")
64                                 and (d >= "00000000001000000")))
65                                 else
66               "00111111" when ((s = '1') and ((d >  "00000111111111111")
67                                 and (d <= "00011111111111111"))) or ((s = '0')
68                                 and ((d <  "00000000001000000")
```

```
69                                      and (d >= "0000000000010000")))
70                                      else
71                      "01111111" when ((s = '1') and ((d >  "0001111111111111")
72                                      and (d <= "0111111111111111"))) or ((s = '0')
73                                      and ((d <  "0000000000010000")
74                                      and (d >= "0000000000000100"))) else
75                      "11111111" when ((s = '1') and ( d >  "0111111111111111"))
76                                      or ((s = '0') and ( d <  "0000000000000100"))
77                                      else
78                      "01010101"; -- obviously invalid display output, for debugging
79
80  end Behavioral;
```

AtlysGeneral.ucf

```
# AtlysGeneral.ucf
# Adapted from the AtlysGeneral.ucf file, available
# from the Digilent website
 PIN "masterClock/clkout3_buf.O" CLOCK_DEDICATED_ROUTE = FALSE;
 PIN "masterClock/clkout1_buf.O" CLOCK_DEDICATED_ROUTE = FALSE;
# clock pin for Atlys rev C board
 NET "sysClk"   LOC = "L15";

# onBoard Leds
 NET "led<0>" LOC = "U18";
 NET "led<1>" LOC = "M14";
 NET "led<2>" LOC = "N14";
 NET "led<3>" LOC = "L14";
 NET "led<4>" LOC = "M13";
 NET "led<5>" LOC = "D4";
 NET "led<6>" LOC = "P16";
 NET "led<7>" LOC = "N12";

# onBoard SWITCHES
 NET "sysReset" LOC = "A10";
 NET "mode" LOC = "D14";
 NET "interfaceLoopback" LOC = "C14";


# Audio
 NET "BITCLK"   LOC = "L13";
 NET "AUDSDI"   LOC = "T18";
 NET "AUDSDO"   LOC = "N16";
 NET "AUDSYNC"  LOC = "U17";
 NET "AUDRST"   LOC = "T17";

# PMOD Connector
 NET "JB<0>" LOC = "T3";
 NET "JB<1>"  LOC = "R3";
 NET "JB<2>"  LOC = "P6";
 NET "JB<3>"  LOC = "N5";
 NET "JB<4>"  LOC = "V9";
 NET "JB<5>"  LOC = "T9";
 NET "JB<6>"  LOC = "V4";
 NET "JB<7>"  LOC = "T4";

# onboard VHDCI
 NET "LED_SIN"  LOC = "U15";
 NET "LED_BLANK"  LOC = "M11";
 NET "LED_LAT<1>"  LOC = "T12";
 NET "LED_LAT<3>" LOC = "R8";
 NET "ADC_CS_C" LOC = "U8";
 NET "ADC_IOCLK" LOC = "N7";
 NET "ADC_DATA_OUT" LOC = "R7";
 NET "LED_SCLK"  LOC = "V15";
 NET "LED_LAT<0>"  LOC = "N11";
 NET "LED_LAT<2>"  LOC = "T10";
 NET "LED_LAT<4>" LOC = "T8";
 NET "ADC_ADDR_INPUT" LOC = "V8";
 NET "ADC_SYSCLK" LOC = "P8";
```

Appendix B: Matlab Source Code

divisionError.m

```
% divisionError.m
% This Matlab script creates the lookup table lut.vhd. The purpose of
% the lookup table is described in the report. The script also outputs
% figures which show the transfer function of the lookup table, and
% the error associated with the scaled integer arithmitic that it
% preforms.

clear

mindbStep = 0.09;

adjDB = zeros(1,128);

shiftFactor = 256;

for p=1:length(adjDB)
    adjDB(p) = (p-1) * mindbStep;
end

adjLIN = zeros(size(adjDB));

for p=1: length(adjDB)
    adjLIN(p) = 10^(adjDB(p)/20);
end

 adjLinScaled = fix (adjLIN.* shiftFactor);

pcmCodes = linspace(0,2^17-1,2^17);

resultScaledUp = zeros(length(adjDB),length(pcmCodes));

for p=1:length(adjLIN)
    for q=1:length(pcmCodes)
        resultScaledUp(p,q) = adjLinScaled(p) * pcmCodes(q);
    end
end

resultIdeal = zeros(length(adjDB),length(pcmCodes));

for p=1:length(adjLIN)
    for q=1:length(pcmCodes)
        resultIdeal(p,q) = adjLIN(p) * pcmCodes(q);
    end
end

resultScaledDown = fix(resultScaledUp / shiftFactor);

error = zeros(length(adjDB),length(pcmCodes));

for p=1:length(adjLIN)
    for q=1:length(pcmCodes)
        error(p,q) = ((resultIdeal(p,q) - resultScaledDown(p,q))/ ...
            resultIdeal(p,q))*100;
    end
end

%Warning: This will bring your computer to it's knees, but looks good.
% figure(3)
% mesh(pcmCodes,adjLIN,error);
% title ('Error for scaled integer multiplication/division');
% xlabel('Input PCM code')
% ylabel('lut output value')
% zlabel('Percent Error')

```

```
maxError = max(error,[],1);

adjLINDiv = 1./adjLIN;
adjLinDivScaled = fix ( adjLINDiv .* shiftFactor);
resultScaledUpDiv = zeros(length(adjDB),length(pcmCodes));

for p=1:length(adjLINDiv)
    for q=1:length(pcmCodes)
        resultScaledUpDiv(p,q) = adjLinDivScaled(p) * pcmCodes(q);
    end
end

resultIdealDiv = zeros(length(adjDB),length(pcmCodes));
for p=1:length(adjLIN)
    for q=1:length(pcmCodes)
        resultIdealDiv(p,q) = adjLINDiv(p) * pcmCodes(q);
    end
end

resultScaledDownDiv = fix(resultScaledUpDiv / shiftFactor);

errorDiv = zeros(length(adjDB),length(pcmCodes));

for p=1:length(adjLIN)
    for q=1:length(pcmCodes)
        errorDiv(p,q) = ((resultIdealDiv(p,q) - ...
            resultScaledDownDiv(p,q))/resultIdealDiv(p,q))*100;
    end
end

maxErrorDiv = max(errorDiv,[],1);

figure (2)
grid on
plot(linspace(128,255,128),adjLinScaled,'-r','LineWidth',2);
hold on
plot(linspace(0,127,128),fliplr(adjLinDivScaled),'-g','LineWidth',2);
grid on
legend('Multiplication','Division');
title('Transfer Function of lookup table');
xlabel('Input')
ylabel('Output')

figure(11)

semilogx (linspace(0,length(maxErrorDiv)-1,length(maxErrorDiv)), ...
    maxErrorDiv,'-g','LineWidth',2);
hold on
semilogx (linspace(0,length(maxError)-1,length(maxError)),maxError, ...
    '-r','LineWidth',2);
legend('Division Error','Multiplication Error');
grid on
title('Maximum error for scaled integer multiplication/division');
xlabel('Input PCM code')
ylabel('Percent Error')

FILE = fopen('lut.vhd','w')
fprintf(FILE, '-- Generated by matlab script divisionError.m \n');
fprintf(FILE, 'library IEEE;\n');
fprintf(FILE, 'use IEEE.STD_LOGIC_1164.ALL;\n');
fprintf(FILE, ' \n');
fprintf(FILE, 'entity lut is\n');
fprintf(FILE, '    Port ( din : in  STD_LOGIC_VECTOR (7 downto 0);\n');
fprintf(FILE, '           dout : out  STD_LOGIC_VECTOR (9 downto 0));\n');
fprintf(FILE, 'end lut;\n');
fprintf(FILE, ' \n');
fprintf(FILE, 'architecture Behavioral of lut is\n');
fprintf(FILE, ' \n');
fprintf(FILE, 'begin\n');
```

```
fprintf(FILE, ' \n');
fprintf(FILE, 'dout <= \n');
for p = 1:128
   fprintf(FILE,'\t\t"');
   fprintf(FILE,dec2bin(adjLinDivScaled(129-p),10));
   fprintf(FILE,'" when din = "');
   fprintf(FILE,dec2bin(256-p,8));
   fprintf(FILE,'" else \n');
end

fprintf(FILE, ' \n');

for p = 1:128
   fprintf(FILE,'\t\t"');
   fprintf(FILE,dec2bin(adjLinScaled(p),10));
   fprintf(FILE,'" when din = "');
   fprintf(FILE,dec2bin(128-p,8));
   fprintf(FILE,'" else \n');
end

   fprintf(FILE,'\t\t"');
   fprintf(FILE,dec2bin(adjLinScaled(1),10));
   fprintf(FILE,'";\n');

fprintf(FILE, ' \n');
fprintf(FILE, 'end Behavioral;\n');
fclose(FILE);
```

filterPlotter.m

```
% filterPlotter.m
% This script plots the magnitude response of 9 paralell FIR filters,
% which are stored in the file filterCoe.mat. Plots 1-9 are the
% individual filter responses, and plot 10 is the combined
% system response.

clear
% Load filters from file where they are stored:
load filterCoe.mat

p = 2^15;

% Preform a padded FFT on all of the time-domain filters.
f1_spect = fft(f1,p);
f2_spect = fft(f2,p);
f3_spect = fft(f3,p);
f4_spect = fft(f4,p);
f5_spect = fft(f5,p);
f6_spect = fft(f6,p);
f7_spect = fft(f7,p);
f8_spect = fft(f8,p);
f9_spect = fft(f9,p);

% x axis, frequency in Hz
x = linspace(0,48000,p);

figure(1)

% plot all of the filter responses
semilogx(x,20*log10(abs(f1_spect)),'g--','LineWidth',2);
hold on
semilogx(x,20*log10(abs(f2_spect)),'b--','LineWidth',2);
semilogx(x,20*log10(abs(f3_spect)),'k-','LineWidth',2);
semilogx(x,20*log10(abs(f4_spect)),'y-','LineWidth',2);
semilogx(x,20*log10(abs(f5_spect)),'m-','LineWidth',2);
semilogx(x,20*log10(abs(f6_spect)),'c-','LineWidth',2);
semilogx(x,20*log10(abs(f7_spect)),'r--','LineWidth',2);
semilogx(x,20*log10(abs(f8_spect)),'g-','LineWidth',2);
semilogx(x,20*log10(abs(f9_spect)),'b-','LineWidth',2);

% sum up all of the responses
sum = zeros(1,p);
for q = 1:p
    sum(q) = abs(f1_spect(q)) + ...
             abs(f2_spect(q)) + ...
             abs(f3_spect(q)) + ...
             abs(f4_spect(q)) + ...
             abs(f5_spect(q)) + ...
             abs(f6_spect(q)) + ...
             abs(f7_spect(q)) + ...
             abs(f8_spect(q)) + ...
             abs(f9_spect(q));
end
semilogx(x,20*log10(sum),'r-','LineWidth',2);

% trim the axis so that nothing above nyquist frequency is displayed:
axis([0 24000 -50 10])

% labels and such:
grid on
title('FIR Filter responses')
xlabel('Analog Frequency, Hz')
ylabel('Magnitude (dB)')
legend ('f1','f2','f3','f4','f5','f6','f7','f8','f9', ...
    'Total System Response');
```

Appendix C: FIR Filter Coefficients

f1_450pt.coe

```
;
; XILINX CORE
Generator(tm)Distributed
Arithmetic FIR filter
coefficient (.COE) File
; Generated by MATLAB(R)
7.10 and the Filter
Design Toolbox 4.7.
;
; Generated on: 04-Apr-
2012 10:55:57
;
Radix = 16;
Coefficient_Width = 16;
CoefData = ace1,
03e6,
03d0,
03ba,
03a6,
0391,
037f,
036b,
035b,
0349,
033a,
032a,
031c,
030d,
0300,
02f3,
02e7,
02db,
02d1,
02c5,
02bc,
02b1,
02a9,
029f,
0298,
028f,
0289,
0281,
027c,
0274,
0270,
0269,
0265,
025f,
025c,
0256,
0255,
024e,
024f,
0245,
0245,
0257,
0239,
023a,
0238,
023c,
023c,
023e,
023d,
023c,
023a,
0239,
0237,
0237,
0236,
0236,
0236,
0237,
0238,
0239,
023b,
023d,
023f,
0240,
0243,
0244,
0247,
0248,
024a,
024c,
024e,
024f,
0252,
0253,
0256,
0258,
025b,
025d,
0261,
0261,
0267,
0268,
0267,
0271,
0273,
0276,
0278,
027b,
027e,
0282,
0286,
028b,
028e,
0292,
0295,
0298,
029c,
029f,
02a2,
02a6,
02a9,
02ad,
02b1,
02b4,
02b8,
02bb,
02bf,
02c3,
02c6,
02ca,
02cd,
02d0,
02d4,
02d7,
02dc,
02df,
02e3,
02e6,
02eb,
02ee,
02f3,
02f5,
02fa,
02ff,
0301,
0304,
0309,
030d,
0311,
0314,
0317,
031b,
031e,
0322,
0325,
0329,
032d,
0330,
0333,
0336,
033a,
033d,
0340,
0344,
0347,
034a,
034e,
0351,
0354,
0357,
035a,
035d,
0360,
0363,
0366,
0369,
036c,
036e,
0372,
0374,
0377,
0379,
037d,
037f,
0382,
0385,
0388,
038a,
038c,
038f,
0391,
0394,
0397,
0399,
039b,
```

```
039d,
03a0,
03a2,
03a5,
03a7,
03aa,
03ac,
03ae,
03b0,
03b2,
03b3,
03b5,
03b7,
03b9,
03ba,
03bc,
03bd,
03bf,
03c0,
03c2,
03c3,
03c5,
03c6,
03c7,
03c8,
03c9,
03ca,
03cc,
03cc,
03ce,
03ce,
03d0,
03d0,
03d1,
03d2,
03d3,
03d3,
03d4,
03d4,
03d4,
03d5,
03d5,
03d6,
03d6,
03d6,
03d6,
03d6,
03d7,
03d7,
03d7,
03d7,
03d7,
03d7,
03d6,
03d6,
03d6,
03d6,
03d6,
03d5,
03d5,
03d4,
03d4,
03d4,
03d3,
03d3,
03d2,
03d1,
03d0,
03d0,
03ce,
03cc,
03cc,
03ca,
03c9,
03c8,
03c7,
03c6,
03c5,
03c3,
03c2,
03c0,
03bf,
03bd,
03bc,
03ba,
03b9,
03b7,
03b5,
03b3,
03b2,
03b0,
03ae,
03ac,
03aa,
03a7,
03a5,
03a2,
03a0,
039d,
039b,
0399,
0397,
0394,
0391,
038f,
038c,
038a,
0388,
0385,
0382,
037f,
037d,
0379,
0377,
0374,
0372,
036e,
036c,
0369,
0366,
0363,
0360,
035d,
035a,
0357,
0354,
0351,
034e,
034a,
0347,
0344,
0340,
033d,
033a,
0336,
0333,
0330,
032d,
0329,
0325,
0322,
031e,
031b,
0317,
0314,
0311,
030d,
0309,
0304,
0301,
02ff,
02fa,
02f5,
02f3,
02ee,
02eb,
02e6,
02e3,
02df,
02dc,
02d7,
02d4,
02d0,
02cd,
02ca,
02c6,
02c3,
02bf,
02bb,
02b8,
02b4,
02b1,
02ad,
02a9,
02a6,
02a2,
029f,
029c,
0298,
0295,
0292,
028e,
028b,
0286,
0282,
027e,
027b,
0278,
0276,
0273,
0271,
0267,
0268,
0267,
0261,
0261,
025d,
025b,
0258,
0256,
0253,
0252,
024f,
024e,
024c,
```

```
024a,
0248,
0247,
0244,
0243,
0240,
023f,
023d,
023b,
0239,
0238,
0237,
0236,
0236,
0236,
0237,
0237,
0239,
023a,
023c,
023d,
023e,
023c,
023c,
0238,
023a,
0239,
0257,
0245,
0245,
024f,
024e,
0255,
0256,
025c,
025f,
0265,
0269,
0270,
0274,
027c,
0281,
0289,
028f,
0298,
029f,
02a9,
02b1,
02bc,
02c5,
02d1,
02db,
02e7,
02f3,
0300,
030d,
031c,
032a,
033a,
0349,
035b,
036b,
037f,
0391,
03a6,
03ba,
03d0,
03e6,
ace1;
```

f2_460pt.coe

```
;
; XILINX CORE
Generator(tm)Distribut
ed Arithmetic FIR
filter coefficient
(.COE) File
; Generated by
MATLAB(R) 7.10 and the
Filter Design Toolbox
4.7.
;
; Generated on: 04-
Apr-2012 11:14:06
;
Radix = 16;
Coefficient_Width =
16;
CoefData = 42af,
dcf6,
e4d2,
ead0,
ef61,
f2dc,
f582,
f787,
f90f,
fa3a,
fb1c,
fbc9,
fc4a,
fcad,
fcf6,
fd2e,
fd56,
fd75,
fd8b,
fd9c,
fda6,
fdaf,
fdb4,
fdb7,
fdb8,
fdba,
fdb9,
fdb9,
fdb7,
fdb6,
fdb3,
fdb2,
fdb0,
fdaf,
fdac,
fdab,
fda8,
fda8,
fda5,
fda5,
fda3,
fda3,
fda1,
fda1,
fda2,
fd9b,
fdab,
fd95,
fda3,
fda8,
fd9e,
fd9d,
fda6,
fdaa,
fda6,
fda3,
fda6,
fdad,
fdb1,
fdb1,
fdaf,
fdb1,
fdb5,
fdbc,
fdc0,
fdc3,
fdc3,
fdc5,
fdc9,
fdcf,
fdd5,
fddb,
fddf,
fde3,
fde6,
fdea,
fdf0,
fdf7,
fdfe,
fe05,
fe0b,
fe12,
fe16,
fe1c,
fe22,
fe2a,
fe31,
fe3a,
fe42,
fe4b,
fe53,
fe58,
fe63,
fe66,
fe71,
fe7a,
fe81,
fe8b,
fe97,
fea0,
fea8,
feb4,
febe,
fec7,
fece,
fed9,
fee4,
feef,
fef8,
ff03,
ff0f,
ff1c,
ff27,
ff32,
ff3c,
ff47,
ff53,
ff5e,
ff69,
ff74,
ff7f,
ff8c,
ff99,
ffa6,
ffb2,
ffbf,
ffcb,
ffd8,
ffe5,
fff2,
fffd,
000a,
0015,
0022,
002e,
003b,
0049,
0055,
0064,
006f,
007d,
008b,
0097,
00a6,
00b3,
00c0,
00cb,
00d9,
00e5,
00f1,
00fe,
010c,
0118,
0125,
0131,
013e,
014b,
0157,
0164,
0172,
017f,
018c,
0197,
01a4,
01af,
01bc,
01c7,
01d2,
01de,
01ea,
01f5,
0201,
020b,
0217,
0221,
022d,
0238,
0243,
024e,
0259,
0263,
026e,
0279,
0282,
028d,
0295,
029f,
```

```
205 02a9,
206 02b1,
207 02ba,
208 02c2,
209 02cb,
210 02d2,
211 02dc,
212 02e3,
213 02eb,
214 02f2,
215 02fa,
216 0301,
217 0309,
218 0310,
219 0318,
220 031e,
221 0325,
222 032b,
223 0331,
224 0337,
225 033c,
226 0341,
227 0347,
228 034b,
229 0350,
230 0354,
231 0359,
232 035c,
233 0360,
234 0363,
235 0367,
236 0369,
237 036c,
238 036d,
239 036f,
240 036f,
241 0372,
242 0373,
243 0375,
244 0375,
245 0376,
246 0377,
247 0377,
248 0378,
249 0377,
250 0377,
251 0376,
252 0375,
253 0375,
254 0373,
255 0372,
256 036f,
257 036f,
258 036d,
259 036c,
260 0369,
261 0367,
262 0363,
263 0360,
264 035c,
265 0359,
266 0354,
267 0350,
268 034b,
269 0347,
270 0341,
271 033c,
272 0337,
273 0331,
274 032b,
275 0325,
276 031e,
277 0318,
278 0310,
279 0309,
280 0301,
281 02fa,
282 02f2,
283 02eb,
284 02e3,
285 02dc,
286 02d2,
287 02cb,
288 02c2,
289 02ba,
290 02b1,
291 02a9,
292 029f,
293 0295,
294 028d,
295 0282,
296 0279,
297 026e,
298 0263,
299 0259,
300 024e,
301 0243,
302 0238,
303 022d,
304 0221,
305 0217,
306 020b,
307 0201,
308 01f5,
309 01ea,
310 01de,
311 01d2,
312 01c7,
313 01bc,
314 01af,
315 01a4,
316 0197,
317 018c,
318 017f,
319 0172,
320 0164,
321 0157,
322 014b,
323 013e,
324 0131,
325 0125,
326 0118,
327 010c,
328 00fe,
329 00f1,
330 00e5,
331 00d9,
332 00cb,
333 00c0,
334 00b3,
335 00a6,
336 0097,
337 008b,
338 007d,
339 006f,
340 0064,
341 0055,
342 0049,
343 003b,
344 002e,
345 0022,
346 0015,
347 000a,
348 fffd,
349 fff2,
350 ffe5,
351 ffd8,
352 ffcb,
353 ffbf,
354 ffb2,
355 ffa6,
356 ff99,
357 ff8c,
358 ff7f,
359 ff74,
360 ff69,
361 ff5e,
362 ff53,
363 ff47,
364 ff3c,
365 ff32,
366 ff27,
367 ff1c,
368 ff0f,
369 ff03,
370 fef8,
371 feef,
372 fee4,
373 fed9,
374 fece,
375 fec7,
376 febe,
377 feb4,
378 fea8,
379 fea0,
380 fe97,
381 fe8b,
382 fe81,
383 fe7a,
384 fe71,
385 fe66,
386 fe63,
387 fe58,
388 fe53,
389 fe4b,
390 fe42,
391 fe3a,
392 fe31,
393 fe2a,
394 fe22,
395 fe1c,
396 fe16,
397 fe12,
398 fe0b,
399 fe05,
400 fdfe,
401 fdf7,
402 fdf0,
403 fdea,
404 fde6,
405 fde3,
406 fddf,
407 fddb,
408 fdd5,
409 fdcf,
410 fdc9,
411 fdc5,
```

```
412 fdc3,
413 fdc3,
414 fdc0,
415 fdbc,
416 fdb5,
417 fdb1,
418 fdaf,
419 fdb1,
420 fdb1,
421 fdad,
422 fda6,
423 fda3,
424 fda6,
425 fdaa,
426 fda6,
427 fd9d,
428 fd9e,
429 fda8,
430 fda3,
431 fd95,
432 fdab,
433 fd9b,
434 fda2,
435 fda1,
436 fda1,
437 fda3,
438 fda3,
439 fda5,
440 fda5,
441 fda8,
442 fda8,
443 fdab,
444 fdac,
445 fdaf,
446 fdb0,
447 fdb2,
448 fdb3,
449 fdb6,
450 fdb7,
451 fdb9,
452 fdb9,
453 fdba,
454 fdb8,
455 fdb7,
456 fdb4,
457 fdaf,
458 fda6,
459 fd9c,
460 fd8b,
461 fd75,
462 fd56,
463 fd2e,
464 fcf6,
465 fcad,
466 fc4a,
467 fbc9,
468 fb1c,
469 fa3a,
470 f90f,
471 f787,
472 f582,
473 f2dc,
474 ef61,
475 ead0,
476 e4d2,
477 dcf6,
478 42af;
```

f3_400pt.coe

```
;
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File
; Generated by MATLAB(R) 7.10 and the Filter Design Toolbox 4.7.
;
; Generated on: 04-Apr-2012 10:35:18
;
Radix = 16;
Coefficient_Width = 16;
CoefData = 6119,
ffdd,
ffdf,
ffe2,
ffe7,
ffed,
fff5,
fffd,
0008,
0012,
001f,
002c,
003b,
0049,
0059,
006a,
007b,
008c,
009f,
00b1,
00c4,
00d6,
00e9,
00fb,
010e,
0120,
0132,
0142,
0153,
0161,
0171,
017c,
018a,
0193,
019e,
01a3,
01ac,
01ac,
01b4,
01a7,
01c9,
01bd,
01a9,
01a6,
0197,
018e,
017c,
016d,
0157,
0143,
0128,
010f,
00ef,
00d0,
00ac,
0087,
005e,
0034,
0006,
ffd6,
ffa3,
ff6f,
ff37,
fefe,
fec2,
fe84,
fe43,
fe01,
fdbd,
fd77,
fd2f,
fce6,
fc9b,
fc4f,
fc01,
fbb3,
fb63,
fb12,
fac2,
fa6d,
fa20,
f9d3,
f97b,
f92b,
f8d7,
f887,
f834,
f7e6,
f796,
f749,
f6fb,
f6b1,
f667,
f620,
f5da,
f597,
f555,
f517,
f4da,
f4a1,
f469,
f437,
f405,
f3d9,
f3ae,
f389,
f365,
f347,
f32b,
f315,
f301,
f2f3,
f2e8,
f2e2,
f2e0,
f2e3,
f2e9,
f2f5,
f305,
f319,
f331,
f354,
f374,
f39d,
f3c7,
f3f9,
f42d,
f467,
f4a5,
f4e8,
f52f,
f57b,
f5ca,
f61f,
f677,
f6d4,
f733,
f798,
f7ff,
f86b,
f8d9,
f94c,
f9c1,
fa3a,
fab4,
fb33,
fbb3,
fc37,
fcbb,
fd43,
fdcc,
fe57,
fee3,
ff71,
ffff,
008f,
011f,
01b1,
0241,
02d3,
0362,
03f6,
0486,
0517,
05a5,
0633,
06bf,
074b,
07d3,
085b,
08e0,
0963,
09e3,
0a62,
0adc,
0b55,
0bc9,
0c3b,
0ca9,
0d14,
0d79,
0ddc,
0e3a,
0e95,
0eea,
0f3b,
0f87,
0fcf,
1011,
1050,
```

```
1088,
10bc,
10ea,
1114,
1137,
1156,
116e,
1182,
118f,
1198,
1198,
1198,
118f,
1182,
116e,
1156,
1137,
1114,
10ea,
10bc,
1088,
1050,
1011,
0fcf,
0f87,
0f3b,
0eea,
0e95,
0e3a,
0ddc,
0d79,
0d14,
0ca9,
0c3b,
0bc9,
0b55,
0adc,
0a62,
09e3,
0963,
08e0,
085b,
07d3,
074b,
06bf,
0633,
05a5,
0517,
0486,
03f6,
0362,
02d3,
0241,
01b1,
011f,
008f,
ffff,
ff71,
fee3,
fe57,
fdcc,
fd43,
fcbb,
fc37,
fbb3,
fb33,
fab4,
fa3a,
f9c1,
f94c,
f8d9,
f86b,
f7ff,
f798,
f733,
f6d4,
f677,
f61f,
f5ca,
f57b,
f52f,
f4e8,
f4a5,
f467,
f42d,
f3f9,
f3c7,
f39d,
f374,
f354,
f331,
f319,
f305,
f2f5,
f2e9,
f2e3,
f2e0,
f2e2,
f2e8,
f2f3,
f301,
f315,
f32b,
f347,
f365,
f389,
f3ae,
f3d9,
f405,
f437,
f469,
f4a1,
f4da,
f517,
f555,
f597,
f5da,
f620,
f667,
f6b1,
f6fb,
f749,
f796,
f7e6,
f834,
f887,
f8d7,
f92b,
f97b,
f9d3,
fa20,
fa6d,
fac2,
fb12,
fb63,
fbb3,
fc01,
fc4f,
fc9b,
fce6,
fd2f,
fd77,
fdbd,
fe01,
fe43,
fe84,
fec2,
fefe,
ff37,
ff6f,
ffa3,
ffd6,
0006,
0034,
005e,
0087,
00ac,
00d0,
00ef,
010f,
0128,
0143,
0157,
016d,
017c,
018e,
0197,
01a6,
01a9,
01bd,
01c9,
01a7,
01b4,
01ac,
01ac,
01a3,
019e,
0193,
018a,
017c,
0171,
0161,
0153,
0142,
0132,
0120,
010e,
00fb,
00e9,
00d6,
00c4,
00b1,
009f,
008c,
007b,
006a,
0059,
0049,
003b,
002c,
001f,
0012,
0008,
fffd,
fff5,
ffed,
ffe7,
```

```
412 ffe2,
413 ffdf,
414 ffdd,
415 6119;
```

f4_200pts.coe

```
;
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File
; Generated by MATLAB(R) 7.10 and the Filter Design Toolbox 4.7.
;
; Generated on: 02-Apr-2012 20:07:53
;
Radix = 16;
Coefficient_Width = 16;
CoefData = ad37,
2e3a,
2435,
1d15,
1812,
1491,
1226,
107e,
0f64,
0ea9,
0e31,
0de2,
0db0,
0d88,
0d68,
0d43,
0d17,
0ce0,
0c9c,
0c47,
0be2,
0b68,
0add,
0a3c,
098c,
08c5,
07f0,
0704,
060c,
04ff,
03e7,
02bd,
018a,
0048,
ff00,
fdad,
fc5a,
fafe,
f9a5,
f845,
f6ef,
f594,
f44b,
f2fa,
f1c8,
f08b,
ef75,
ee68,
ed53,
ec81,
ebaa,
eae3,
ea3c,
e9c1,
e957,
e901,
e8ca,
e8bc,
e8cc,
e8fa,
e943,
e9a9,
ea34,
eae2,
ebaf,
ec9a,
ed9c,
eebc,
eff3,
f148,
f2b6,
f43d,
f5d5,
f77e,
f934,
faf8,
fcc8,
fea1,
0081,
0265,
0449,
062b,
0806,
09da,
0ba2,
0d5d,
0f07,
10a3,
122d,
13a1,
14fd,
1639,
1761,
1862,
194c,
1a0f,
1aaa,
1b2c,
1b8b,
1bbe,
1bcd,
1bbe,
1b8b,
1b2c,
1aaa,
1a0f,
194c,
1862,
1761,
1639,
14fd,
13a1,
122d,
10a3,
0f07,
0d5d,
0ba2,
09da,
0806,
062b,
0449,
0265,
0081,
fea1,
fcc8,
faf8,
f934,
f77e,
f5d5,
f43d,
f2b6,
f148,
eff3,
eebc,
ed9c,
ec9a,
ebaf,
eae2,
ea34,
e9a9,
e943,
e8fa,
e8cc,
e8bc,
e8ca,
e901,
e957,
e9c1,
ea3c,
eae3,
ebaa,
ec81,
ed53,
ee68,
ef75,
f08b,
f1c8,
f2fa,
f44b,
f594,
f6ef,
f845,
f9a5,
fafe,
fc5a,
fdad,
ff00,
0048,
018a,
02bd,
03e7,
04ff,
060c,
0704,
07f0,
08c5,
098c,
0a3c,
0add,
0b68,
0be2,
0c47,
0c9c,
0ce0,
0d17,
0d43,
0d68,
0d88,
0db0,
0de2,
```

```
0e31,
0ea9,
0f64,
107e,
1226,
1491,
1812,
1d15,
2435,
2e3a,
ad37;
```

f5_101pt.coe

```
;
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File
; Generated by MATLAB(R) 7.10 and the Filter Design Toolbox 4.7.
;
; Generated on: 02-Apr-2012 17:46:19
;
Radix = 16;
Coefficient_Width = 16;
CoefData = c1c8,
4bcc,
3183,
24cd,
1eef,
1c6a,
1b5c,
1ac7,
1a2b,
1931,
17be,
15b5,
1316,
0fdb,
0c1b,
07d0,
0319,
fe05,
f8bb,
f347,
edd9,
e88b,
e38b,
dee8,
dacd,
d74c,
d48e,
d29f,
d19d,
d18e,
d284,
d46e,
d753,
db26,
dfe6,
e56f,
eba9,
f263,
f98f,
0104,
0898,
1007,
1741,
1e20,
2470,
29f7,
2ebe,
329e,
3555,
371e,
379e,
371e,
3555,
329e,
2ebe,
29f7,
2470,
1e20,
1741,
1007,
0898,
0104,
f98f,
f263,
eba9,
e56f,
dfe6,
db26,
d753,
d46e,
d284,
d18e,
d19d,
d29f,
d48e,
d74c,
dacd,
dee8,
e38b,
e88b,
edd9,
f347,
f8bb,
fe05,
0319,
07d0,
0c1b,
0fdb,
1316,
15b5,
17be,
1931,
1a2b,
1ac7,
1b5c,
1c6a,
1eef,
24cd,
3183,
4bcc,
c1c8;
```

f6_101pt.coe

```
;
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File
; Generated by MATLAB(R) 7.10 and the Filter Design Toolbox 4.7.
;
; Generated on: 02-Apr-2012 17:49:20
;
Radix = 16;
Coefficient_Width = 16;
CoefData = 14b0,
0176,
006c,
fec0,
fca0,
fa57,
f837,
f694,
f5b1,
f5bb,
f6b9,
f88f,
fafd,
fdab,
0032,
0235,
036b,
03af,
030b,
01ba,
001d,
feb1,
fdf8,
fe60,
0032,
0374,
07e7,
0d09,
1219,
1636,
1879,
1820,
14a5,
0de8,
0430,
f841,
eb3f,
de98,
d3d4,
cc60,
c97b,
cbd6,
d39f,
e063,
f10d,
040d,
177a,
2950,
37a4,
40ec,
4422,
40ec,
37a4,
2950,
177a,
040d,
f10d,
e063,
d39f,
cbd6,
c97b,
cc60,
d3d4,
de98,
eb3f,
f841,
0430,
0de8,
14a5,
1820,
1879,
1636,
1219,
0d09,
07e7,
0374,
0032,
fe60,
fdf8,
feb1,
001d,
01ba,
030b,
03af,
036b,
0235,
0032,
fdab,
fafd,
f88f,
f6b9,
f5bb,
f5b1,
f694,
f837,
fa57,
fca0,
fec0,
006c,
0176,
14b0;
```

f7_101pt.coe

```
;
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File
; Generated by MATLAB(R) 7.10 and the Filter Design Toolbox 4.7.
;
; Generated on: 02-Apr-2012 17:43:52
;
Radix = 16;
Coefficient_Width = 16;
CoefData = f5a9,
fd41,
fef2,
0131,
02cb,
02f9,
01e9,
00a1,
0046,
013a,
02bf,
0363,
020b,
fed4,
fb3e,
f952,
fa51,
fdd1,
01e4,
044b,
03fe,
01df,
002b,
00d0,
03e5,
074f,
0802,
0427,
fcb1,
f52c,
f1bd,
f477,
fbce,
035f,
06cb,
04c0,
0024,
fe63,
037e,
0e8f,
1924,
1a70,
0cd4,
f282,
d608,
c5d1,
cc9b,
eafb,
15c8,
3ac1,
494c,
3ac1,
15c8,
eafb,
cc9b,
c5d1,
d608,
f282,
0cd4,
1a70,
1924,
0e8f,
037e,
fe63,
0024,
04c0,
06cb,
035f,
fbce,
f477,
f1bd,
f52c,
fcb1,
0427,
0802,
074f,
03e5,
00d0,
002b,
01df,
03fe,
044b,
01e4,
fdd1,
fa51,
f952,
fb3e,
fed4,
020b,
0363,
02bf,
013a,
0046,
00a1,
01e9,
02f9,
02cb,
0131,
fef2,
fd41,
f5a9
```

f8_101pt.coe

```
;
; XILINX CORE
Generator(tm)Distributed
Arithmetic FIR filter
coefficient (.COE) File
; Generated by MATLAB(R)
7.10 and the Filter
Design Toolbox 4.7.
;
; Generated on: 02-Apr-
2012 17:03:57
;
Radix = 16;
Coefficient_Width = 16;
CoefData = 0222,
ffe7,
0035,
00ab,
ff72,
fdb0,
fef9,
021d,
021f,
ff7a,
ff11,
0021,
fe84,
fd10,
009b,
0466,
01e3,
fde5,
fee6,
ffd0,
fc63,
fcd9,
04a0,
0758,
ffb4,
faff,
fec8,
fecb,
fa09,
ff9e,
0bbf,
088c,
f967,
f73d,
ff9c,
fcb1,
f84b,
0894,
174c,
0431,
ea59,
f297,
02f6,
f7d1,
f9c4,
2924,
3650,
dff6,
93e3,
dcec,
6760,
6760,
dcec,
93e3,
dff6,
3650,
2924,
f9c4,
f7d1,
02f6,
f297,
ea59,
0431,
174c,
0894,
f84b,
fcb1,
ff9c,
f73d,
f967,
088c,
0bbf,
ff9e,
fa09,
fecb,
fec8,
faff,
ffb4,
0758,
04a0,
fcd9,
fc63,
ffd0,
fee6,
fde5,
01e3,
0466,
009b,
fd10,
fe84,
0021,
ff11,
ff7a,
021f,
021d,
fef9,
fdb0,
ff72,
00ab,
0035,
ffe7,
0222;
```

f9_101pt.coe

```
;
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File
; Generated by MATLAB(R) 7.10 and the Filter Design Toolbox 4.7.
;
; Generated on: 02-Apr-2012 16:53:42
;
Radix = 16;
Coefficient_Width = 16;
CoefData = fff4,
0143,
0008,
ff86,
0012,
008f,
ffd7,
ff5d,
0047,
00b4,
ff94,
ff3e,
0099,
00cb,
ff33,
ff33,
0108,
00c6,
feb5,
ff4b,
0193,
0096,
fe1f,
ff98,
0234,
0028,
fd76,
002d,
02e1,
ff65,
fcc7,
0127,
038f,
fe2a,
fc1d,
02b3,
0431,
fc32,
fb87,
0540,
04b9,
f8c4,
fb10,
0a32,
051b,
f0c1,
fac5,
1a86,
054f,
aeba,
7aab,
aeba,
054f,
1a86,
fac5,
f0c1,
051b,
0a32,
fb10,
f8c4,
04b9,
0540,
fb87,
fc32,
0431,
02b3,
fc1d,
fe2a,
038f,
0127,
fcc7,
ff65,
02e1,
002d,
fd76,
0028,
0234,
ff98,
fe1f,
0096,
0193,
ff4b,
feb5,
00c6,
0108,
ff33,
ff33,
00cb,
0099,
ff3e,
ff94,
00b4,
0047,
ff5d,
ffd7,
008f,
0012,
ff86,
0008,
0143,
fff4;
```

Appendix D: Cost and Scheduling

Table 12: Bill of Materials and Total Cost

Component	Manufacturer	Distrib	Ref Name	Qty Req'd	Ordered	Extended Price
1N914A	FSC	none	D74 D73	2	2	$ -
3MM_LED_RED	Kingbright	Mouser	D1-D72	72	100	$ 9.00
AWHW40G-0202-T-R	Assmann WSW Components	Digi-Key	J1	1	1	$ 0.95
CB2518T4R7M	TaiyoYuden	Digi-Key	L1	1	3	$ 0.42
GRM21BF51A106ZE15L	Murata	Digi-Key	C23 C10 C12 C24 C26 C27 C28 C29	8	10	$ 4.78
GRM219R71C104KA01D	Murata	Digi-Key	C22 C4 C3 C1 C2 C5 C6 C7 C8 C9	10	20	$ 6.12
PS30-10PB10BR10K	TT Electronics/BI	Digi-Key	R6 R9 R12 R15 R18 R21 R24 R27 R30	9	10	$ 10.54
71600-040LF	FCI	Digi-Key		1	2	$ 4.22
RC0805FR-071KL_1	Yageo	Digi-Key	R8 R11 R14 R17 R20 R23 R26 R29 R32 R33 R36	11	20	$ 0.40
RC0805FR-071M62L_1	Yageo	Digi-Key	R38	1	10	$ 0.20
RC0805FR-072KL_1	Yageo	Digi-Key	R1 R2 R3 R4 R5	5	10	$ 0.20
RC0805FR-078K2L_1	Yageo	Digi-Key	R7 R10 R13 R16 R19 R22 R25 R28 R31 R34 R37	11	20	$ 0.40
RC0805FR-0710KL_1	Yageo	Digi-Key	R35	1	10	$ 0.20
RC0805FR-07180KL_1	Yageo	Digi-Key	R39	1	10	$ 0.20
SN74LVCH16244ADGGRG4	Texas Instruments	TI	U1	1	3	$ -
TLC541IDW	Texas Instruments	TI	U13	1	3	$ -
TLC59281DBQR	Texas Instruments	TI	U2 U3 U4 U5 U6	5	9	$ -
TLV1117-33CDCYR	Texas Instruments	TI	U7 U8	2	3	$ -
TPS61072DDCRG4	Texas Instruments	TI	U14	1	3	$ -
TPSB107K006R0250	Texas Instruments	Digi-Key	C11 C13 C25	3	6	$ 4.56
Digikey Shipping & Tax	USPS	Digi-Key		1	1	$ 5.61
Mouser Shipping & Tax	USPS	Mouser		1	1	$ 8.76
Custom PCB (incl ship and tax)	Advance Circuits	Advanced Circuits		1	1	$ 61.70
68-pin SCSI cable	Maddison Cable Corp	Halted Specialties		1	1	$ 5.00
Digilent Atlys (incl ship and tax)	Digilent	Halted Specialties		1	1	$ 210.45
Misc cables	Various	Halted Specialties		1	1	$ 15.00
Total						**$ 348.71**

Table 13: Gantt chart showing actual project progress

Month	April					May				June			
Week (starting Monday)	22	9	16	23	30	7	14	21	28	4	11	18	25
Component selection													
Firmware development													
Schematic design													
PCB Layout													
Hardware assembly and test													
Firmware test													
Design filters													
Final system integration													
Test and tweak													
Documentation													

Appendix E: References

[1] Ashok Ambardar, *Analog and Digital Signal Processing*, 2nd ed. Pacific Grove, California: Brooks/Cole Publishing Company, 1999.

[2] et al Kitazato, Digital Graphic Equalizer, Patent # 4,661,982, 1987.

[3] Digilent Incorporated. (2011, February) Atlys C2 Reference Manual. [Online]. www.digilentinc.com

[4] Kingbright. (2011, March) T-1 (3mm) Solid State Lamp WP710A10SRC/E, Datasheet. [Online]. www.kingbrightusa.com

[5] Texas Instruments, Incorporated. (2011, January) TLC59281 16-Channel, Constant-Current LED Driver, Datasheet. [Online]. www.ti.com

[6] Michael Leddige. (2006, December) Transmission Line Basics II - Class 6. [Online]. www.intel.com

[7] Texas Instruments, Incorporated. (2001, June) TLC541L 8-BIT Analog-to-Digital Converters with Serial Control and 11 Inputs, Datasheet. [Online]. www.ti.com

[8] TT Electronics. (2009, December) Model PSxx Slide Potentiometer, Datasheet. [Online]. www.bitechnologies.com

[9] Digilent, Incorporated. (2010, June) Digilent Incorporated Web Site. [Online]. www.digilentinc.com

[10] Xilinx, Incorporated. (2011, October) Spartan-6 FPGA Data Sheet: DC and Switching Characteristics, Datasheet. [Online]. www.xilinx.com

[11] Texas Instruments, Incorporated. (2010, August) TLV1117, Adjustable and Fixed Low-Dropout Voltage Regulator, Datasheet. [Online]. www.ti.com

[12] Xilinx, Incorporated. (2011, October) LogiCORE IP FIR Compiler v5.0, Datasheet. [Online]. www.xilinx.com

[13] Xilinx, Incorporated. (2011, May) Spartan-6 FPGA Clocking Resources. [Online]. www.xilinx.com

[14] Intel Corporation. (2000, september) Audio Codec '97, White Paper. [Online]. www.intel.com

[15] Texas Instruments, Incorporated. (2005, November) SN74LVCH16244A 16-bit Buffer/Driver with 3-state Outputs, Datasheet. [Online]. www.ti.com

[16] National Semiconductor Corporation. (2002, December) LM4550 AC '97 Rev 2.1 Multi-Channel Audio Codec with Stereo Headphone Amplifier, Sample Rate Conversion and National 3D Sound, Datasheet. [Online]. www.national.com