Single-Channel Audio Digitizer

ECE4873 Senior Design Project

Section sd20s01, Team Audio Synthesizers

Project Advisor, Dr. Whit Smith

Hardware Coordinator, Oliver Mattos

Joe Farnham, jfarnham3@gatech.edu, Webmaster

Moradeke Olumogba, molumogba3@gatech.edu

Submitted

August 2, 2020

Executive Summary

The goal of our project was to design a multi-channel audio digitizer at a cost that made the device affordable to individual hobbyists and developers. Our project was based on a similar project that was completed in 2014 - a link to the previous project is provided in the appendix. While devices that can sample audio data on multiple channels already exist in the market, they are often too expensive for individual developers to purchase. While we were confident in our ability to make a functioning, low-cost device, we found the project intriguing and knew that it would expose us to technologies and skills that could help prepare us for jobs in industry or academia. For these reasons, we decided to pursue this project to fulfill our senior design requirement.

Our design included low-cost, off-the-shelf components, and we have also made our source hardware and software designs easily accessible and open source. By estimating labor costs and overhead, we calculated the cost to produce this device to ~$8,041, with the components costing ~$201. The two main components in our design were an analog to digital converter (ADC) that was responsible for capturing audio data, and a USB chip that was responsible for converting the digitized audio data to something that could be transmitted via USB.

Unfortunately, while we proved aspects of both our hardware and software were functional, we were unable to read any data through our USB device given a labelling error on our schematic, which will be detailed later in this report. That being said, we were able to confirm that we could initialize all components of our design, see one channel of audio data output from the ADC, and affect the sampling rate of the hardware through software.

Table of Contents

Single-Channel Audio Digitizer

1. Introduction

Team Audio Synthesizers is requesting $201 in funding to develop a multi-channel audio digitizer. The funding would be used for hardware development and production.

1.1 Objective

The goal of this project was to develop a multi-channel audio digitizer that was cheaper than alternatives in the market. Our design included developing a functional printed circuit board (PCB) and software that could communicate with that board. Given our hardware and software designs are open source, hobbyists and developers can access those designs and change them to suit their specific needs.

1.2 Motivation

The consumers of this device want a technology that can sample multiple channels of audio simultaneously and output a .wav file that replicates what was heard through the stereo speakers [1]. Ideally, this device will be easy to use; the end consumer will not have to care about the internals of the device. The end user would need to tell the device to listen to audio for a certain length of time, and subsequently listen to the file produced by the C++ code. The device should be cheaper than the alternatives that already exist, which enables more people to be able to purchase this device.

1.3 Background

Our design used the CS5368 audio AD codec from Cirrus Logic. This device is capable of sampling audio at rates up to 192 kHz and can transmit eight channels of audio data within one serial data stream. [10] Analog Devices makes several AD converters that provide similar functionality. One such example is AD7264. This converter is a high-speed device that needs a 5V power supply to operate. The throughput on this chip goes up to 1Msps. [14] Texas Instruments also offers a similar ADC named ADS1274. This device is a 24-bit ADC that is a bit slower than the Cirrus Logic chip, operating with sampling rates up to 144 ksps. It also supports simultaneous sampling of four channels instead of eight. [15]

2. Project Description and Goals

Our group developed a device that has the necessary components to meet our goals. However, due to a schematic error, we were unable to sample audio data through the USB chip. Therefore, we could not demonstrate working functionality of any portions of the software that were downstream of that functionality. We were able to:

- Initialize all components of our design properly
- Verify the Cirrus Logic ADC could sample and transmit audio data
- Verify that we could affect the sampling rate of the ADC through software
- Sample data at rates up to 96 kHz
- Verify the audio data passed through the flip flop and made it to the USB chip

We were unable to:

- Read audio data from the USB chip due to a schematic error

- Verify we could sample multiple channels of audio simultaneously (although we have no reason to believe our design would not allow us to do so)
- Convert the audio data into a wave file (we have code in place to format data received from the USB chip, but we never tried outputting a wav file due to our focus on getting the USB chip to read)
- Sample audio at rates up to 192 kHz (although we have no reason to believe our design would not allow us to do so)

3. Technical Specifications & Verification

- USB interface from device to Windows PC
- Developed code in C++ instead of the proposed Python3

Multichannel Audio Synthesizer		
2020 August 2		
Technical Specifications		
	Proposed	**Quantitative**
Specification	**Quantitative**	**Goal**
	Goal	**Demonstrated**
Sample Rate	>= 192k Samples/Second	<= 96k Samples/Second
Channels	8	1
Bits/Sample	24 or 32	24
File Format	File must be a .wav file	Not Completed
Powered by USB	Yes	Yes

4. Design Approach and Details

4.1 Design Approach

Rather than create our own schematic, we decided to use the schematic that was created by the 2014 senior design cohort [17]. We decided to use their schematic for a number of reasons: 1) We knew from reading the previous group's final report that their design worked, 2) We walked through the schematic and verified to ourselves that it should work, 3) Our group had two people compared to the previous group's five people, and we were also operating with a shorter timeline, as the summer semester is shorter than the Fall and Spring semesters. By verifying the schematic should work, we saved ourselves time that was more valuable given the shortened semester.

The two main components of this design were the Cirrus Logic CS5398 codec chip, and the FTDI FT232H USB chip. Instead of directly purchasing the FT232H chip directly, we purchased the development board that had the FT232H chip on it. This development board is named UM232H and is sold by FTDI. To synchronize the timing between the ADC and USB chip, we inserted a D flip flop between the ADC and the USB. We also added an inverter to make the software more understandable. The reset pin on the ADC is active low, but we aimed to toggle this reset pin through a GPIO pin on the USB chip. Therefore, when we wanted to reset the ADC, we would simply set the GPIO pin high on the USB chip. This signal would then be inverted to match the active low nature of the ADC's reset pin and hold the ADC in reset. The software then always contained the logic value of the reset pin instead of the physical value. Speaking of GPIO pins, the UM232H chip had four GPIO pins connected to the ADC. One was a reset pin, as mentioned above, and the other three were for SPI communication. Those three GPIO pins were configured as a SPI clock, SPI data, and chip select. We were able to manually

toggle the three GPIO pins dedicated for SPI communication to transmit serial data to the ADC using the SPI protocol.

To test that we soldered our components correctly, and that the PCB was manufactured correctly, we used a multi-meter to verify connection integrity. We set the multi-meter to a mode that would beep when a connection was present between the two probes connected to the multi-meter. We completed this testing for every component of our design, including the codec, the flip-flop, the inverter, and resistors. Since we never received stereo jacks, we soldered a pair of earphones to an audio input on the ADC. We verified these earphones listen to the outside world by looking at the signals produced by those channels with the open source software PulseView [19]. When all the other components were placed on the PCB, the capacitors were probed by a multi-meter to determine whether it matched the estimated range given on the codec datasheet [10].

Moreover, in our design, we had to replace a 5.1 Ω resistor with a short since the resister never arrived in the mail. Seeing as our team was operating under both time constraints and COVID-19 constraints, we decided to proceed without this resistor. Considering the resistor was so small, at only 5.1 ohms, we do not think this had a significant effect on our project. Please find some pictures below that showcase different portions of our testing and building process.

Fig. 1. This figure is the test of the power line to be sure that the board is being provided with power.

The whole board was powered by the USB to Computer connection. The next figure shows the voltage across a capacitor.

Fig. 2. Voltage across the capacitor matched that which was supplied as expected.

See the following figures representing the testing process.

Fig. 3. Voltage across capacitor with the range of estimated expected values.

Fig. 4. Circuit assembled for first phase of testing. The extending cables are the stereo connector connection onto the surface.

The voltage across certain passive components was probed with a logic analyzer and seen that it was within the estimated voltage expected. The audio played through the stereo connector was also probed and captured by the logic analyzer. Please reference a video posted on our website showcasing this functionality.

After verifying that components had been soldered correctly, we then began testing our software written to interact with our device. Our software was written in C++ and used the .dll driver and API provided to us from FTDI to interact with the FT232H chip on the UM232H development board. The first responsibility of the software was to properly initialize all components of our design. In order to initialize the ADC, we needed to set the Global Control Register (GCTL) to tell the ADC to operate in Control Port Mode, which told the ADC to use the external crystal oscillator at its clock. Depending on the sampling rate we wished to sample at, we needed to flip certain clock division bits in that register. The software properly handled this logic.

To initialize the flip flop, we just needed to hold the flip flop in reset. A nice design in our schematic included tying the chip select pin of the ADC to the reset pin of the flip flop. Since the chip select pin on the ADC was active low, as was the reset pin on the flip flop, this would hold the flip flop in reset any time we wanted to reinitialize the ADC by modifying its GCTL.

Lastly, we needed to initialize the USB chip. This included but was not limited to both opening and resetting the FT232H chip, which can be done easily through the API provided to us. In addition, we set the FT232H chip to operate in FT1248 mode by programming the chip's EEPROM. The EEPROM holds configuration settings that the FT1248 loads upon initialization. Holding the flip flop in reset during initialization of the ADC allowed the data lines on the USB chip to be initialized with all zeros. When the USB chip is set to operate in FT1248 mode, that initialization step initializes FT1248 mode to operate in 8 bit mode, which was ideal since we

intended to sample eight channels of audio. In total, there were 5 pins connected from the flip flop to the USB chip. Four of the pins held serial data, while the fifth pin was the left-right clock. The left right clock differentiates between the two channels of data per pin. This is how we could connect four data pins to the USB chip instead of eight. Please reference the two screenshots below. The first one shows the output of the executable, and the second one shows the SPI data that we send to the ADC during initialization. The second picture shows the open-source software PulseView, which reads data from a logic analyzer that we had connected to the SPI pins of the USB chip.

Fig. 5. Output from the executable

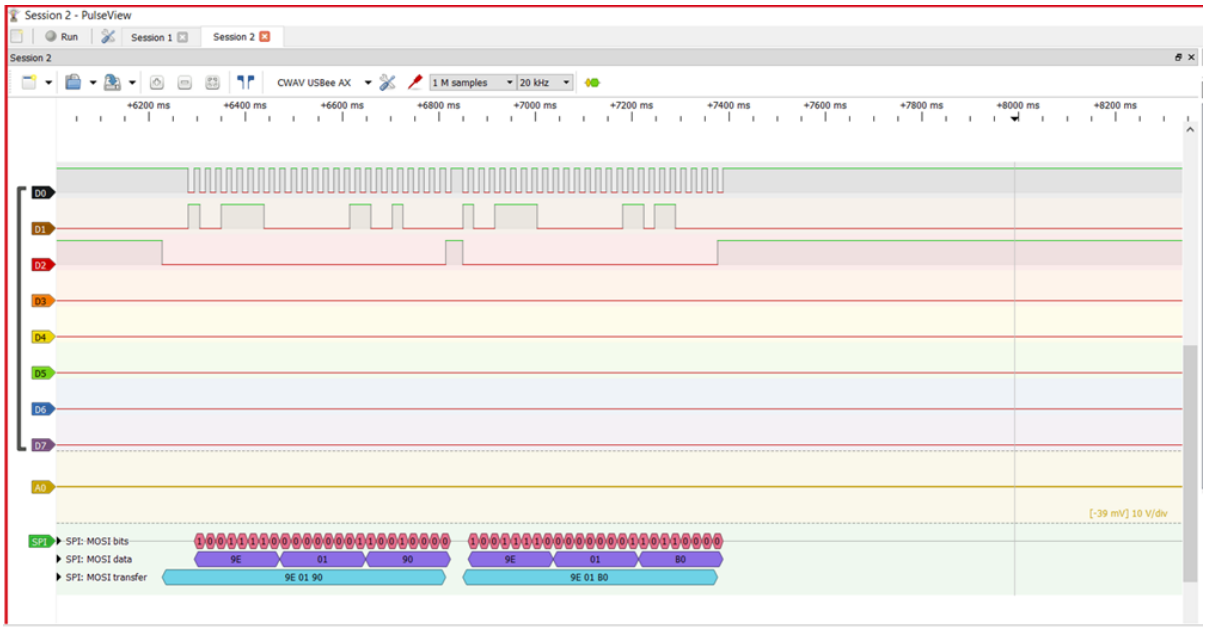

Fig. 6. (a) Software executable writing to the device. (b) Logic Analysis and SPI capture of the device. The data in both figures match. D0 represents SCLK, D1 represents the data, and D2 represents chip select.

Fig. 7. Noise, which should be around 0, was captured by probing the FTDI RX/TX pins. D0 represents LRCLK, D1 represents SCLK, and D2 represents the data.

In Figure 7 above, we are probing the FTDI chip's pins that contain both audio data and audio data clocks. You can also note in figure 8 and 9 below that we could double the sampling rate of the hardware. This was done through software by initializing the ADC via SPI communication. The following two pictures prove not only that SPI communication was working between the USB chip and the ADC, but that we could successfully and properly affect the sampling rate of the ADC through software.

Fig. 8. Frequency of LRCLK doubling verified by logic analysis. The period $_{changed}$ from 41 µs to 20.5 µs.

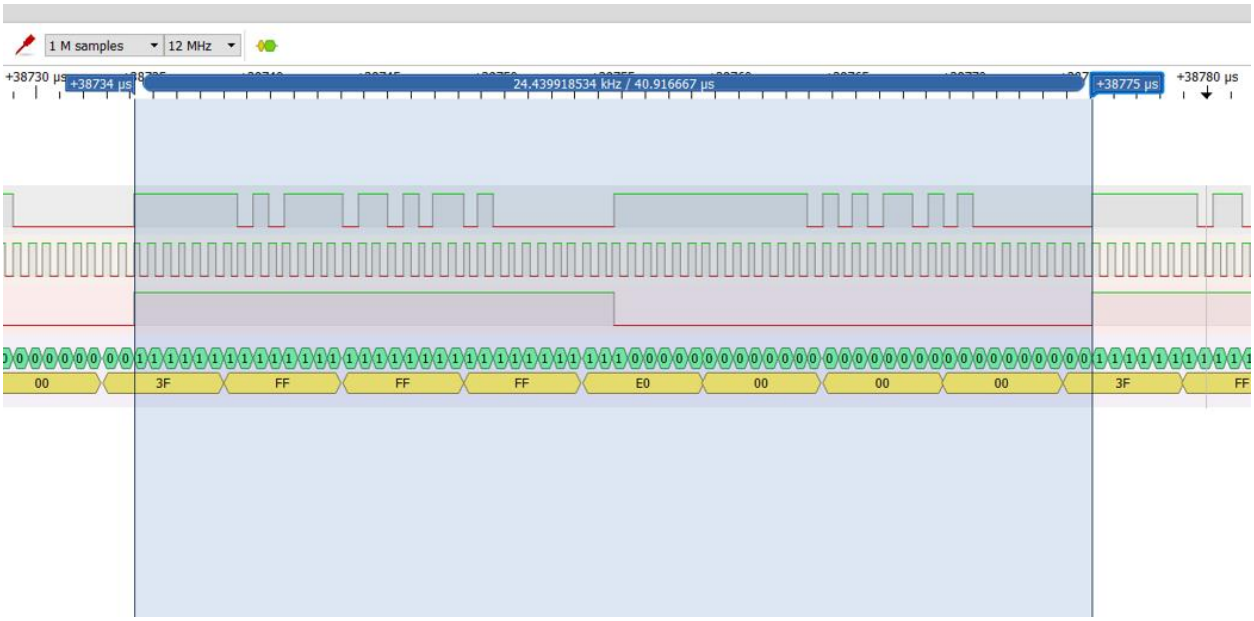

Fig. 9. Frequency of LRCLK used to be half that of figure 8. upon change by software and period used to be approximately 41 µs.

We have demonstrated what we accomplished; now, we would like to detail where our design fell short. As stated earlier, we were unable to sample data using FT1248 mode on the FT232H chip. We narrowed this down to one key error in our schematic and ultimately found two issues with our schematic (and therefore also the 2014 group's schematic). The first issue was that the

inverter was incorrectly labelled. The inverter was labelled to invert RST_bar to RST when it should have been inverting RST to RST_bar. This error made it impossible for us to hold the ADC in reset. The second issue was that the AC1 pin on the FTDI chip was labelled RST_bar. The pin should have been labelled RST. Because the inverter was backwards, we did not have any control over RST_bar. However, if we did, we still would have wanted to the AC1 pin to be RST instead of RST_bar. The AC1 pin on the UM232H device corresponded to the slave select pin (SSn) in FT1248 mode. Since we could not control the SSn pin, we could not activate FT1248 mode, which meant we could never read audio data into the FT232H chip. Even if we had been able to control the SSn pin, we would have wanted to set the pin to the value of RST, instead of RST_bar, since the SSn pin was active low. You can see in Figure 10 that we read zero bytes of data from the device, which is a result of these two labelling/wiring mistakes.

A future team could stand to correct these two mistakes we made or decide to rewire the UM232H device to use a different mode on the FTDI chip. We did not need to use FTt1248 mode but found it was suitable for our needs.

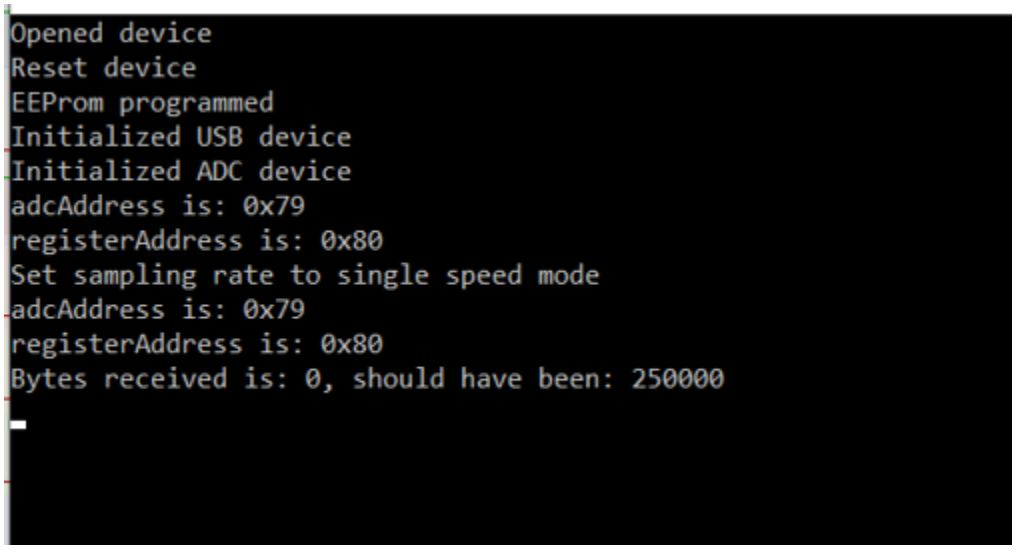

Fig. 10. Last statement informs user that RX transmission has not occurred indicating that data was not found to be read. However, logic analysis reveals existence of data.

4.2 Codes and Standards

The driver was written in C++ and complies to C++ standards [16]. We used the USB 2.0 protocol to transfer data over the USB cable. The entire USB protocol was handled on the FTDI chip, so that was abstracted away from us. We also used SPI to communicate with the ADC.

4.3 Constraints, Alternatives, and Tradeoffs

We did not need to use USB 2.0 to interface with the Windows computer. However, we chose to do so to increase the bandwidth of the connection to 480 Mbps. Other serial interfaces can be limited to 115 kbps. Also, almost every modern computer has a USB jack, which allows our device to work with almost any computer.

We also chose to use FT1248 mode. We could have chosen a different mode in our design. While we chose FT1248 mode, this mode limited us to use a single channel FT232H due to the pinout needed to operate FT1248 mode. If we had chosen a different mode, the chip could have read audio data while it was transmitting data via SPI. Instead, our device could only do one of those things at a time.

Our last design tradeoff was using the UM232H instead of the FT232H chip directly. Using the development board made the device more expensive, but it also made the device more easily configurable.

Lastly, the group members and professor were in three different time zones in two different countries and had to operate while COVID-19 safety protocols were in place. None of the three people involved in this project (Joe, Moradeke, and Dr. Smith) were in the same place at the

same time. Communication across multiple time zones proved difficult at times, and there were often delays in development since some of the development needed to be done in parallel. Joe also did not have access to a full hardware setup. Moradeke was the only one that had a PCB, so much of the software testing was time consuming and inefficient, and there was only so much testing that could be done without the representative device.

5. Schedule, Tasks, and Milestones

Joseph Farnham was the software developer. He was responsible for creating the software to initialize and communicate with the hardware. He wrote the C++ code that leveraged the FTDI driver and API. His software was Windows specific and was provided to the outside world in the form of an executable. Joe was also the Webmaster. Software risks included incorrectly configuring the chips, which could have bricked them.

Moradeke Olumogba was the hardware developer. She was responsible for verifying the previous team's schematic, ordering the physical parts, outsourcing manufacturing the PCB, and soldering components onto the PCB. She was also responsible for using the logic analyzer and PulseView to probe and decode various SPI and audio data. Oliver Mattos, mentioned on the title page, helped Moradeke with soldering the ADC, and provided the logic analyzer [18] that we used. He also helped Moradeke convert our schematic into Gerber files. Her risks included but were not limited to incorrectly soldering components onto the PCB and shorting two connections during testing which could have bricked a board.

6. Final Project Demonstration

We showcased four features of our design in the final demonstration.

- We can initialize the ADC and USB chips properly
    - During testing, we demonstrated this functionality through our ability to send SPI communication from the UM232H to the ADC chip
    - Seeing that we could affect the sampling rate of the hardware through software proved at a system level that the initialization process was working
- We can receive and transmit audio data from the ADC
    - We verified this with a logic analyzer and PulseView
- The audio data transmitted from the ADC makes its way through the flip flop to the USB pins
    - We verified this with a logic analyzer and PulseView
- We can affect the sampling rate of the ADC through software
    - We took two samples of audio data with the logic analyzer and PulseView. The first sample included one sampling rate, and the second sample included data doubled at twice the speed of the first sample. We then verified the frequency of the LRCLK matched the sampling rate we set in software for both samples.

Our Final Presentation includes our Final Demonstration and can be found on our website. The link to our website is in the Appendix.

7. Cost Analysis

A total of 38 virtual collaborative actual hours were undertaken via Skype. An approximate of 110 independent work hours were undertaken. An estimated 10 hours of meeting solicitations were undertaken with Dr. Smith. This yields an estimated total of 196 working hours. Labor costs came to $40/hr * 196 hrs = $7840

CS5368-CQZ (x2)	$45.64
SN74AHC273DWR (Flip flop, x5)	$2.25
UM232H (USB chip, x3)	$69.63
Inverter (x10)	$3.90
Resistors & Capacitors	~$13
Crystal	$0.69
PCB manufacturing	$21.44
USB cable	~$2
Breadboard	~$10
Labor	~$7,840
Tax Costs	$32.75
Total	$8,041.30

8. Conclusion

While we made progress with our design, we fell short in completing some of the main, overarching goals of our project. We were able to initialize the USB chip and ADC properly, affect the sampling rate of the ADC through software, and see that audio data was clearing the flip-flop and making it to the USB chip. We detailed the schematic changes in a previous section of the paper that would need to be made for this device to function as expected. We learned valuable lessons in hardware design, including how we need to understand every detail of the schematic before we ask for the schematic to be manufactured. If we had understood these details better, we would not have run into the hardware issues that we did.

Lastly, a future team could opt to use the FT232H chip directly instead of the UM232H development board. The device would be cheaper this way, and we have spent enough time with the UM232H to say that the development board, although nice to have for prototyping and developing, is not necessary for a marketable version of this device.

9. References

[1] A. Ward, Y. Lu, E. Patterson, M. Yang, F. Xie "Multichannel ADC Device with Simultaneous Conversion" Presented at ECE4011 Senior Design. [PDF]. Available: http://ece4012y202002.ece.gatech.edu/14spring/ECE4012L1A/ws2/

[2] W. Smith. (2020). Analog Devices (email of 5/29/2020)

[3] W. Smith (2020), AUVSI submarine vehicle competitions (email of 5/28/2020)

[4] W. Smith (2020), Initial Challenges (email of 5/27/2020)

[5] W. Smith (2020), Tasks and links to suppliers (email of 5/21/2020)

[6] W. Smith (2020), Prior Project's website (email of 5/19/2020 3:41 PM)

[7] W. Smith (2020), Suggestion of parts (email of 5/19/2020 1:38 AM)

[8] W. Smith (2020), Project description (email of 5/14/2020)

[9] W. Smith (2020), Expected serialized bits (email of 5/29/2020 2:13 PM)

[10] Cirrus Logic, 114 dB 192 kHz 8-Channel A/D Converter. Austin, TX. [Datasheet]. Available: on our website

[11] Future Technologies Devices International Ltd, Single Channel High-Speed USB to Multipurpose UART/FIFO IC. Glasgow, UK. [Datasheet]. Available: on our website

[12] Future Technologies Devices International Ltd, Single Channel High-Speed USB to Multipurpose UART/FIFO IC. Glasgow, UK. [Datasheet]. Available: on our website

[13] Future Technologies Devices International Ltd, Single Channel High-Speed USB to Multipurpose UART/FIFO IC. Glasgow, UK. [Programmer's guide]. Available: on our website

[14] Analog Devices, "1 MSPS, 14-Bit, Simultaneous Sampling SAR ADC with PGA and Four Comparators," AD7264 datasheet, 2012. Available: http://www.analog.com/static/imported-files/data_sheets/AD7264.pdf. [Accessed Dec. 2, 2013].

[15] Texas Instruments, "Quad/Octal, Simultaneous Sampling, 24-Bit Analog-to-Digital Converters," ADS1274 datasheet, 2011. Available: http://www.ti.com/lit/ds/symlink/ads1274.pdf. [Accessed Dec. 2, 2013].

[16] C++11, ISO 14882:2011, 2013.

[17] http://ece4012y202002.ece.gatech.edu/14spring/ECE4012L1A/ws2/misc/8ChannelADCtoUSB.zip [8ChannelADCtoUSB/design_spark/MultiADC_Schematic_2]

[18] LHT00SU1 Oscilloscope and Logic Analyzer

[19] PulseView – version 0.4.2

[20] Microsoft Visual Studio Code – version 1.47.3 for Windows 10

Appendix

Our website: http://ece4012y202002.ece.gatech.edu/sd20s01/