

THE ASPEN PLATFORM
TOOLS FOR MIXED SIGNAL ELECTRONICS, DIGITAL SIGNAL
PROCESSING, AND BIOMEDICAL ELECTRONICS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL
ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

William John Esposito
August 2018

© 2018 by William J Esposito. All Rights Reserved.
Re-distributed by Stanford University under license with the author.

This dissertation is online at: <http://purl.stanford.edu/nq977kw3386>

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Gregory Kovacs, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Juan Rivas-Davila

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Laurent Giovangrandi,

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Preface

This thesis describes the motivation for, development and testing of the ASPEN Platform. ASPEN, short for Advanced Signal Processing ENvironment is a suite of hardware and software tools to enhance and facilitate prototyping and teaching of mixed signal electronics, digital signal processing, and the design of biomedical instrumentation.

Acknowledgments

I would like to thank my parents for a lifetime of unflagging support and encouragement. I'd also like to thank my academic advisor, Greg Kovacs for his unflagging support and guidance, which had helped me become a better engineer and scientist, and whose vision has helped guide this project. I'd also like to thank Dr.'s Laurent Giovangrandi, Fernando Mujica, and Ron Schafer whose assistance has been critical in evaluating and improving these tools. Finally, I'd like to thank my fellow lab members, Corey McCall and Nathan Volman whose advice and assistance have been instrumental in the development of the system.

Contents

Preface	iv
Acknowledgments	v
1 Background	1
1.1 Motivations	1
1.2 Need	2
1.3 The Need For An open-source Approach	2
1.4 Requirements for Hardware	4
1.5 Abstractions	8
1.6 Lack of signal processing capabilities in the embedded processor space	9
1.7 A revolution in microcontrollers	9
1.8 No revolution in digital signal processors	12
1.9 Signal processing block diagram tools	15
1.10 Microcontroller tools do not support DSP	17
1.11 Embedded desktop systems	19
1.12 Lack of tools that meet the requirements	21
1.13 ASPEN architecture	21
1.14 Approach to the system	23
1.15 Conclusion	23
2 Hardware	24
2.1 System Design Approach	24

2.2	Analog Shield Design Overview	26
2.3	Analog Shield Breadboard and Power Supply	27
2.4	Analog Shield ADC and DAC Hardware	30
2.5	Characterization of the Analog Shield	32
2.6	DSP Shield	39
2.7	The Components of the DSP Shield System	42
2.8	DSP Shield CPU	42
2.9	Audio CODEC	45
2.10	Other DSP Shield Peripherals	48
2.11	DSP Shield Revisions	53
2.12	Conclusion	56
3	Software	57
3.1	Design Approach	57
3.2	System Architecture Overview	58
3.3	Analog Shield Software	59
3.4	Analog Shield Driver Architecture	59
3.5	Analog Shield Examples	61
3.6	DSP Shield Software Approach	63
3.7	DSP Shield Modes	65
3.8	DSP Shield Mode Application	68
3.9	DSP Shield Mailbox Library	73
3.10	DSP Shield IDE	74
3.11	DSP Shield Co-Processor and Standalone Modes	77
3.12	DSP Shield Bootloader	80
3.13	DSP Shield Hardware Abstraction Layer	82
3.14	DSP Shield Signal Processing Library	84
3.15	Conclusion	87
4	Mixed Signal Applications	88
4.1	Design Approach	88
4.2	Analog Shield “Instructables”	89

4.3	DDS Signal Generator Instructable	89
4.4	Spectrum Analyzer Instructable	94
4.5	Polyphonic Music Instructable	98
4.6	Lissajous Pattern Instructable	99
4.7	Network Analyzer Instructable	99
4.8	Kit use in EE122A	102
4.9	EE122A First Integration	102
4.10	EE122A Take Home Lab	104
4.11	Lab Exercises in EE122A	106
4.12	EE122A Conclusions	114
4.13	Kit Use in Other Mixed Signal Applications	114
4.14	Conclusion	116
5	Digital Signal Processing Applications	117
5.1	Overview	117
5.2	DSP Shield App	118
5.3	DSP Shield App Examples	119
5.4	Hello World for DSP	120
5.5	Student Projects Using the DSP Shield	122
5.6	Integration with EE264	123
5.7	Comparison with EE265	125
5.8	Tools Developed In Support of EE264	126
5.9	Labs Zero and One, Introduction, Hardware, and Software Setup . . .	128
5.10	Lab Two, Software Flow Control	129
5.11	Lab Three, fixed-point Arithmetic	130
5.12	Lab Four, Sample Rate Conversion	131
5.13	Lab Five, Hardware Accelerators	131
5.14	Lab Six, FFT	132
5.15	Final Project	132
5.16	OFDM Final Project	135
5.17	Conclusion	137

6	Biomedical Applications	138
6.1	Motivation	138
6.2	Background	138
6.3	Serving Scientific and Teaching Needs	140
6.4	The ASPEN platform for DC-coupled ECG	140
6.5	ECG Signal Background	141
6.6	Non-cardiac Sources of ECG Signal Corruption	144
6.7	Tradeoffs Between AC- and DC-Coupled Systems	146
6.8	Prototyping the system using ASPEN	150
6.9	Single Channel ASPEN DC-Coupled ECG	151
6.10	DC-Coupled ECG Performance	154
6.11	Multi-channel DC Coupled ECG	162
6.12	Filter Matching in a Multi-Channel System	163
6.13	Impact of Spectral Folding in the ASPEN Multi-Channel ECG	168
6.14	Digital Filters Implemented in the ASPEN Multi-Channel ECG . . .	170
6.15	Multi-Channel ECG Performance	172
6.16	DC Offset Removal in a DC-Coupled ECG	174
6.17	Implementing DC-Offset Removal in Software	175
6.18	Nonlinear Correction	180
6.19	Hybrid DC-AC ECG Performance	182
6.20	Offline Post-Processing Demonstrations	185
6.21	Conclusion	190
7	Conclusions	191
7.1	Overview	191
7.2	Future Work	194
A	Analog Shield ADC Test Code	195
B	Analog Shield Schematics and BOM	197
C	DSP Shield Schematics and BOM	207

D Analog Shield Manual	225
E DSP Shield HAL Key Functions	230
F Instructables	236
G Long-QT Detection Algorithm Code	271
Bibliography	276

List of Tables

1.1	Requirements for a system for use in teaching mixed-signal, biomedical electronics and signal processing.	6
2.1	A list of some of the most popular Arduino Shields electrically compatible with the DSP Shield.	40
2.2	List of major DSP Shield components.	43
3.1	Detail of inter-operation of Arduino Uno, Analog Shield and DSP Shield	67
3.2	Summary of functions supported in initial release of Shield App, detailing functionality shown in Figure 3.7.	72
3.3	OS platforms supported by ASPEN project. The Linux build of Energia-DSP is considered beta due to the limited number of test users. . . .	77
3.4	Listing of some of the Arduino HAL libraries ported to the DSP Shield.	84
4.1	Sample comparison table asked of students to evaluate the trade-offs between the various filter technologies they used in the experiment. . .	109
5.1	List of topics covered in lecture EE264 at Stanford, week by week. . .	124
5.2	List of topics covered in lecture EE264 at Stanford, week by week. . .	133
6.1	Comparison of performance specifications of ECG devices.	161
6.2	Comparison of performance specifications of ECG devices.	172
6.3	Comparison of performance specifications of ECG devices.	185

List of Figures

1.1	Example of a block diagram-based tool implementing a low-pass FIR Filter in Simulink.	16
1.2	Example of Teensy Signal Chain Development.	18
2.1	Top view of Arduino Uno highlighting discoverability of the design. .	25
2.2	Analog Shield as viewed from the top, without included breadboard (above) and with breadboard installed (below).	27
2.3	Photograph of the Analog Shield bottom, showing silkscreen. Top: ADC and DAC buffer. Bottom: Power supplies.	30
2.4	High-level ADC diagram, depicting channel multiplexer and selection between differential and single ended modes (after Burr-Brown, 2003). .	31
2.5	Comparison of Analog Shield, first (left) and final (right) revisions, top side of boards above, bottom side of boards below.	33
2.6	Plot of data sampled by ADC of the first revision of the Analog Shield, with a grounded input.	34
2.7	Partial listing of ADC data acquisition program run on Arduino for Analog Shield Characterization. Some setup and serial printing scripts omitted for brevity. The entire program is included as Appendix A. .	35
2.8	Analog Shield ADC Performance characterization, depicting a 1 kHz test signal input to both the Analog Shield and the Native ADC for an Arduino Uno. Each signal is an average of 250.	37
2.9	DAC performance characterization, depicting a generated 100 Hz sine wave with both the Analog Shield and the native Arduino PWM. . .	38

2.10	Overview of DSP Shield final revision, highlighting the location of some of the major components.	41
2.11	AIC3204 configurable FIR (top) and IIR (bottom) hardware filter block at the ADC, for one of two channels. A similar set of reconstruction filters exists at the output to the DAC before the Sigma-Delta node (after Texas-Instruments, 2012).	46
2.12	Characteristic spectrum of sigma-delta converter, as found in the ACI3204. This noise shaped signal has a low noise floor below 20 kHz but the noise level rises at higher frequencies. This is addressed by using an output reconstruction filter which attenuates high-frequency components outside the band of interest, in this case 20-20,000 Hz.	46
2.13	Unfiltered output with 100 mV noise envelope.	47
2.14	Comparison of headphone (top) and oscilloscope (bottom) outputs, showing the effect of the simple 1 st order low-pass reconstruction filter on the output spectrum of the DSP Shield.	49
2.15	Serial mux configuration.	50
2.16	DSP Shield to Arduino UNO reset control multiplexing scheme. These multiplexers enable the DSP Shield to use a GPIO pin to keep the Arduino in reset during an upload of user software to the DSP Shield, preventing bus contention. Once complete, they reconfigure the reset line so the Arduino and DSP Shield can be reset by the same signal.	50
2.17	DSP Shield OLED display, user DIP switches, and user LEDs.	51
2.18	The three versions of the DSP Shield, from left to right C5517 v1, C5535, C5517 v2.	54
3.1	Native Arduino analog interfaces and Analog Shield equivalents for reading and writing samples to the analog domain. The top two lines compare a write command to the native PWM and the Analog Shield DAC. The final two show read commands from the native ADC and the Analog Shield.	60
3.2	Additional software interfaces in the Analog Shield library	61

3.3	Drop down menu in Arduino IDE showing Analog Shield library examples.	62
3.4	DSP Shield operation modes.	65
3.5	Example shield mode application library call.	68
3.6	Application layers showing functional path to send data from an Arduino to the Shield App. The message passes from a function call to a predefined point in an application on the DSP Shield.	69
3.7	Shield Application signal chain block diagram.	70
3.8	Mailbox library example, a program which sends a message to the DSP Shield and prints the response received via the serial port.	73
3.9	New Energia sketch, as seen by a user at program launch.	76
3.10	DSP Shield mode selection drop down menu in Energia, showing how the user switches from Standalone to Co-Processor mode.	78
3.11	Process for a user to switch between standalone and co-processor modes.	79
3.12	Flow diagram showing how the DSP Shield receives a new program via a serial upload command, as well as how it can discriminate from an upload targeted at an Arduino Uno sharing the same serial bus.	82
4.1	DDS Function Generator using Analog Shield, Arduino, and character LCD.	90
4.2	DDS Function Generator software high level block diagram.	91
4.3	DDS Reconstruction Filter.	93
4.4	FFT spectrum visualizer, using Analog Shield and Adafruit LCD. Input is an electret microphone, out of frame.	94
4.5	Anti-aliasing filter.	95
4.6	FFT Visualizer anti-aliasing filter, using a single quad op-amp IC to implement an 8 th order Chebyshev filter with a 24 kHz cutoff.	96
4.7	Resistive summing network for polyphonic music Instructable.	97
4.8	Four voice MIDI playback using the Analog Shield, with output on an oscilloscope.	98

4.9	First generation prototype Analog Shield with Lissajous pattern generator code running, displayed on an oscilloscope.	99
4.11	Network analyzer noise source (top) and anti-aliasing filter (bottom).	100
4.10	Analog Shield network analyzer showing measurement of an LC band-pass filter.	101
4.12	Outputs of the quantization example on an oscilloscope. Students were asked to compare the output of the four channels of the Analog Shield DAC. These outputs were four copies of a single ADC input quantized at various levels.	103
4.13	EE122A lab kit from fall 2016. The kit contains an Arduino Uno, Analog Shield, Analog Discovery 2, breadboard, wire kit, TI My Parts Kit and miscellaneous additional components.	105
4.14	Four channel op-amp comparison board used in 2014 and 2015. Input at top, with the signal repeated and buffered by four op-amps of different types configured for unity gain.	107
4.15	The filter boards used in the EE122A filter comparison lab. From left to right, passive LC filter, op-amp filter using Sallen-Key topology, switched capacitor filter, DSP Shield.	108
4.16	ASCoT circuit schematic. Note that the device under test (DUT) is depicted as a bipolar transistor in this case.	111
4.17	Analog Shield curve tracer circuit, disconnected from the Analog Shield.	112
4.18	ASCoT PC application, showing the result of tracing a 2N222A NPN transistor.	113
5.1	DSP Shield App example that converts a stereo audio signal to a four-channel surround sound system.	119
5.2	Partial ASPEN stack with Arduino, DSP Shield, and Danger Shield. The controls on the Danger Shield are used to adjust the parameters of the filter running on the DSP Shield.	120
5.3	Slide potentiometer circuit diagram.	121

5.4	Device to allow for transmission of a mother's heartbeat to a newborn, using ASPEN components and a piezoelectric transducer to reproduce a heartbeat signal in a stuffed toy.	122
5.5	Example program for the DSP Shield using the serial communication library.	127
5.6	Audio loopback example showing the processAudio function.	128
5.7	EE264 lab two introduces students to the benefits of processing block-by-block to save context switching time (after Mujica, 2015).	129
5.8	EE264 Lab Three introduces students to the effects of truncation and rounding. Shown here is a comparison of a floating-point value and its value when rounded and when truncated.	130
5.9	Block diagram illustrating sample rate conversion.	131
5.10	Overview of CODEC hardware accelerator (HWA) architecture used by students in Lab Five.	131
5.11	Block diagram of OFDM system implemented by EE264 students (after Mujica, 2015).	136
6.1	The ECG waveform arises from electrical impulses in cardiac cells. The externally-recorded signal is the far-field sum of signals from groups of cells depolarizing in an organized spatio-temporal sequence (used with permission from Malmuivo, 1995).	142
6.2	ECG vector evolution in time and how its projection appears on standard ECG leads I, II and III (after Malmuivo, 1995).	143
6.3	ECG waveform, highlighting major segments (used with permission from Agateller, 2007).	144
6.4	Layers of material affecting the impedance between the electrode and the body.	147
6.5	A typical AC-coupled ECG. Note the inclusion of a high-pass filter for each lead.	148

6.6	Basic DC-coupled multichannel ECG System. Analog system complexity is reduced at the cost of higher digital performance requirements. In the last few years, this has become a viable approach and has appeared in some integrated ECG front-end ICs, such as the ADS-129x (Texas Instruments, Dallas, TX) and ADAS-1000 series (Analog Devices, Norwood, MA).	149
6.7	DC-coupled single channel ECG, demonstrating quick prototyping of a full ECG signal chain using minimal parts.	151
6.8	Typical single lead ECG circuit with offset cancellation.	151
6.9	67 tap FIR low-pass filter with an $F_c = 135$ Hz, used in DC-coupled ECG before downsampling.	153
6.10	DC Coupled single ECG performance, showing $77 \mu V_{pk-pk}$ of grounded input-referred noise (top), An SNR of 39 dB with a 2 Hz, 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.	156
6.11	EE122B teaching ECG circuit performance, showing $27 \mu V_{pk-pk}$ of grounded input referred noise (top), A SNR of 53 dB with a 2 Hz 4 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.	157
6.12	AD8232 ECG evaluation board, showing $67 \mu V_{pk-pk}$ of grounded input referred noise (top), An SNR of 42 dB with a 2 Hz 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.	158
6.13	CMRR of each of the systems, plotted from 1 Hz to 100 kHz. Roll-off above 150 Hz is due to the filtering schemes implemented in all three devices.	159
6.14	Comparison of the step response of an AD8232 and DC-coupled ECG to a 7 mV 100 Hz sinusoidal input. The AD8232 takes approximately 2.5 s to settle to within 5% of full scale of its final value while the DC-coupled ECG settles in milliseconds.	160

6.15	Multi-channel ECG system as implemented on the DSP Shield. Note how the topology allows IAs to be de-duplicated, as compared to the multi-channel system in Figures 6.5 and 6.6.	164
6.16	The CMRR of a mismatched differential RC-filter versus frequency for 0.1% matched resistors and 1% matched capacitors.	165
6.17	Sweep of cutoff frequency of mismatched RC filters with 0.1% resistor and 1% capacitor matching. Highlighted in red is the 150 Hz band of interest for ECG.	166
6.18	Plot of CMRR vs. frequency of the eight-channel ECG compared to the single channel commercial system.	168
6.20	PSD of 100 Hz test signal.	168
6.19	Plot of response to sinusoidal chirp signal fed into ECG system. The chirp was a 60 mV amplitude signal swept from 100 Hz to 9 kHz. Signal aliasing is visible in the 2.4, 4.8 and 7.2 kHz bands. The system digital filter causes rapid roll-off after 150 Hz.	169
6.21	Digital filter signal chain for each synthesized ECG lead.	170
6.22	single-line function call for applying one filter to an ECG channel, highlighted in yellow.	171
6.23	Multichannel ECG, showing 84 μ V of grounded input-referred noise (top), An SNR of 34 dB with a 2 Hz 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.	173
6.24	Block diagram of multi-channel ECG with individual channel mean removal feedback. A summing amplifier is used to provide additional gain and include nonlinear offset correction.	176
6.25	Example of test offset sinusoids being removed by digital post-processing removal algorithm.	177
6.26	Screenshot of eight-channel DC-coupled ECG real-time plotting software with DC-offset correction and gain of 1000, showing standard ECG leads I, II, III, and V3.	178

6.27	Plot of 4 channels, taken from the system with the gain set to 200. From top to bottom leads I, -II, III, V3.	179
6.28	ECG lead electrode motion offset with linear recovery.	180
6.29	System ADC input saturation due to lead - off condition. Nonlinear control is evident, with system responding in well under a single beat.	181
6.30	Eight-channel hybrid ECG, showing $54 \mu V_{pk-pk}$ of grounded input re- ferred noise (top), An SNR of 38 dB with a 2 Hz 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequen- cies of interest are DC to 150 Hz.	183
6.31	CMRR of 8Ch Multistage Hybrid ECG vs AD8232. As expected, there is some CMRR cost in the eight-channel system, versus the previously described systems and the reference AD8232. Nonetheless, overall per- formance is comparable between the systems, and as expected the mul- tistage system performs similarly to the single-stage eight-channel ver- sion.	184
6.32	Plot of ECG signal (top), detected QRS signal trigger events (third row), and extracted QRS complexes (bottom) for a lead I ECG signal.	186
6.33	Cardiac axis reconstruction via extracted, ensemble averaged QRS complexes (top). Einthoven Triangle (bottom).	188
6.34	(Top Row) Extracted QRS complex, (Second Row) ECG with QRS complex removed, (Third Row) Detected QRS and T events, (Fourth Row) S-T trigger events, (Fifth Row) Estimated ST intervals with mean and 1-standard deviation bars.	189
6.35	ST interval estimation algorithm. The algorithm removes the mean from the signal, then detects and removes the QRS and P complexes from the signal, storing the timing of the falling edge of the complex. The algorithm then detects the onset of the T-wave.	190

Chapter 1

Background

1.1 Motivations

The genesis of the Advanced Signal Processing Environment (ASPEN) project was a series of discussions in 2012 between Prof. Gregory Kovacs of Stanford, Dr. Martin Izzard (then VP of R&D systems at Texas Instruments) and Bill Esposito. These discussions focused on the success of new microcontroller tools, such as the rapidly growing Arduino platform. These new tools had made microcontrollers accessible to a much wider audience and had dramatically reduced the time required to set up prototype embedded systems, even finding use by engineers skilled in the field.

The discussions were informed by Prof. Kovacs experiences teaching lab courses in analog and biomedical electronics at Stanford. Dr. Izzard and Prof. Kovacs also had extensive experience with digital signal processing systems and saw the applicability of signal processing tools to teaching these concepts. Through these discussions it became clear that improved tools could be developed for teaching and prototyping of mixed-signal, digital signal processing and biomedical electronics. This would enable a cross-disciplinary approach encompassing all three topics, while allowing students and hobbyists to embed their work in devices outside the classroom.

1.2 Need

As will be discussed in the following sections, there was no existent platform to make these capabilities truly accessible to the masses. Creating such a tool would enable the teaching of a hands-on, mixed-signal-electronics or DSP course without a dedicated electronics laboratory. Previously, this was not feasible for several reasons: complexity of the tools, difficulty of setup, or over-abstraction introduced to avoid such challenges. Furthermore, from experience working with Dr. Laurent Giovangrandi to develop biomedical electronics course material at Stanford, Prof. Kovacs was aware that there was no viable open-source hardware and software suited to teaching biomedical electronics design.

The discussion of these needed tools lead naturally to questions about the requirements for teaching these topics without costly, high-quality test equipment and without compromising the material. An examination of the existing tools and consideration of the challenges that would have to be met resulted in a proposal for a system of hardware and software optimized to solve these problems and address those questions, later dubbed ASPEN.

In the following sections, an outline of the requirements for such a system and the technologies that helped inspire this approach are discussed. Additionally, some pre existing approaches to teaching signal processing are described. Funding for this project was graciously provided by Texas Instruments, Inc.

1.3 The Need For An open-source Approach

From the beginning of the project, it became clear that the hardware and software must all be “open” tools, meaning open-source software, and open hardware. open-source means that the underlying code and design files for all projects are released publicly, under a permissive license that allows sharing and reproduction of the designs without restriction of licensing fees.

There are several benefits to this approach. open-source software is easier to distribute globally, due to the public nature of its source code. Wide global distribution can in turn grow the pool of potential users, and a large group has enormous positive network effects in terms of development and community support for user issues. Therefore, the tools must be distributable globally, without export or licensing restriction such as the International Traffic in Arms Regulations (ITAR) [1] which require the vendor of software to have information about users they release it to. Such restrictions can even deter users and developers who are approved to use the software, both because they have to prove their eligibility for the software and because they in turn have to ask the same of others they pass the software on to. By making the software for the system open-source and inherently compliant, the process is dramatically simplified.

A second benefit of an open-source design approach is that the software is free for its users to use, distribute, modify and collaborate on [2], without undue burden from licensing fees or copyright restrictions. By choosing to use an open-source license for the software systems, as opposed to simply giving away proprietary software at no charge also provides certainty to the community. It guarantees users that the software cannot later disappear and that the developers cannot later monetize their previous work. It also provides members of the community the ability to improve the software without first obtaining the consent of the creators.

Similarly, as much of the hardware toolchain as possible should be “open hardware” as well. To be described as open hardware, the full detailed design schematics for the system need to be offered freely to anyone who might want to build their own version of the tools. This is congruent with the goals of a widely distributed project framework. It must be noted that there are some limitations to how far the open-hardware concept can be practically extended. high-performance, open hardware integrated circuits are virtually nonexistent and would be prohibitively expensive for users to replicate even with full designs. Therefore, a reasonable compromise is to choose integrated circuits expected to be available for several years at least and are not subject to export or licensing restrictions. As many components as possible should be available from multiple sources or at least expected to be available for an extended

period. Fortunately, unlike integrated circuits, the printed circuit board designs and layouts for the project are both practical to share, and for the community to replicate at reasonable cost. The globally successful and widely replicated Arduino platform, discussed later, is an example of how an open hardware platform can be built using proprietary components.

Therefore, the ideal open-source system is one that can be replicated freely by the community, ensuring availability and that the price of the system is driven as low as possible. In mass production the system should be comparable in price to a textbook and as readily available.

1.4 Requirements for Hardware

The hardware requirements for teaching analog electronics, signal processing, and biomedical electronics are driven by the needs of embedded circuit prototyping and of the disciplines of interest. The needs are multifaceted but fortunately overlap extensively. In addition to the more abstract considerations such as export, and duplicability, there are the more practical concerns, such as circuit design, performance, signal quality, and cost.

In order to expand learning of these disciplines outside of the classroom, and to meet the needs of embedded projects in the developing and developed world, the hardware must be physically small enough to be conveniently portable as well as sufficiently energy efficient to be powered by reasonably sized batteries.

To support teaching and building analog electronics, it must provide the tools necessary to build and analyze simple electronic circuits. A typical analog electronics workbench contains a number of tools for these tasks. Most prominent among those are a power supply, a multimeter, a signal generator, an oscilloscope and a breadboard.

Designing circuits for biomedical electronics often requires those same tools but also imposes additional restrictions. Because these signals are generally small and low-frequency AC signals, sometimes superimposed on large DC offsets, as in electrocardiography (ECG) and photoplethysmography (PPG), the system must be low noise and the signal-acquisition systems must have a large dynamic range. It also needs to capture the low-frequency components of these biomedical signals, meaning the system should work as near DC as is feasible. For safety reasons, the ability to electrically isolate the circuit when connected to a person is also critical. Commonly, this is achieved in low-cost systems by disconnecting communications and powering the device with a battery or using a USB isolator. These requirements are more restrictive than for a general low-cost mixed-signal-electronics platform, meaning that such a system can serve well for both fields.

In order to analyze and manipulate signals, signal processing tools are required. On a workbench this is often a combination of test instruments and a desktop computer. These tools include a wide range of functions, such as digitization, filtering and the computational performance to implement complex algorithms. To build a better (portable and embeddable) signal processing tool chain for all three fields, a similar capability must be provided. This requires sufficient performance in terms of signal to noise ratio (SNR) and effective number of bits (ENOB). This can enable analysis and manipulation of signals from DC for biomedical applications to 20 kHz for audio signal processing and possibly beyond at a reasonable quality.

As will be described later, many preexisting signal processing environments focus on audio signals because they are easy for students to grasp. Typical audio processing hardware has a 20 Hz high-pass filter at the input and output for circuit protection and biasing. This filter prevents these devices from being useful for analyzing most biomedical signals, which typically have critical frequency content well below 20 Hz. Therefore, a carefully selected analog-to-digital converter (ADC) and digital-to-analog converter (DAC) enabling both low and audio frequency signal analysis are needed. Additionally, support for multiple channels in the ADC and DAC allow for a broader range of applications, such as sensor fusion and reference signal acquisition.

Requirement	Detail
open-source software & hardware	The community should be able to replicate the system, creating trust and providing the ability for community improvement.
low-cost	Price point in large volumes ~\$100 US. Affordability enables use in the developing world and the device to be permanently installed in systems.
Small form factor	Similar size to an Arduino Uno (2.7" x 2.1") to enable embedding in projects.
Low power	Draws less than 50 mA at idle. This allows the device to run on four commodity AA batteries for approximately two days.
Galvanic isolation	Can be electrically isolated from earth for body connected biomedical systems
Large storage capacity	8 GB or more of storage for several hours of recorded data.
DC-coupled	Can capture low-frequency signals (below 1 Hz) for biomedical applications.
High ENOB	High ENOB (~15 bits) gives a large dynamic range on ADC and DAC to enable measurement of small signals with large offsets.
Simple user interface	Student can install and use the system without a mentor in the room.
Flexible design	System has simple interface for novice users, but can be used in more sophisticated ways by more sophisticated users.
Sufficient computing performance	Digital component has enough performance to filter, process, and store signals in the audio range (20-20,000 Hz) without extreme optimization.
Circuit prototyping support.	Easy way for user to construct circuits and interface them with the system.

Table 1.1: Requirements for a system for use in teaching mixed-signal, biomedical electronics and signal processing.

A processor that can acquire and manipulate such signals also presents an opportunity to take advantage of modern large and low-cost non-volatile storage. While many microcontroller platforms have some non volatile storage built-in, the size of this space is typically limited to a few kilobytes. This memory is generally used to store user programs and configuration data when system power is cycled. Larger memory capacity, sufficient to store and later process much larger sets of data can provide an opportunity to address new problems with a signal processing platform. In contrast, preexisting embedded signal processing platforms only have a moderate amount (several hundred kilobytes to a few megabytes) of high-speed memory in which to store signals. This memory is volatile, however, which prevents the embedding of software and pre-computed datasets. The continuing increase in the capacity of removable consumer flash memory devices has made a large non-volatile storage cheap enough to be included in an embedded platform. With a 32 GB storage device, many hours of audio or other signals can be recorded. Because biosignals are generally low bandwidth, even longer recordings can be feasibly made with such a system. A 2,000 sample/second recording has many times the required bandwidth to acquire a low-frequency biosignal such as ECG ($f_{\max} = 150$ Hz) or PPG ($f_{\max} = 5$ Hz), yet can be recorded for more than 90 days continuously with 32 GB of storage, making battery capacity and skin irritation due to electrodes the limiting factors. This storage capacity enables more novel applications than simple recording. A large number of pre-computed filters or other stored routines which require pre-computed data can be stored in memory. Importantly, vast datasets can be stored without meaningful impact to the amount of space available to the user for programs or data. The ability to pre-compute can be leveraged to allow novice students to implement complicated algorithms with simple function calls, as will be discussed in more detail in Chapter Three. Table 1.1 summarizes the hardware requirements defined for the system.

1.5 Abstractions

A major challenge in introducing students to signal processing tools can be the complexity of setting them up. Users in an online or remote teaching course will not necessarily have an expert on hand to resolve setup or configuration problems, and even in an environment where expert help is available, this can be a bottleneck. This means that the platform must be designed with carefully chosen abstractions in both hardware and software to avoid unnecessary complication. Poor design can create significant barriers to achieving the teaching goal of the system by discouraging users before they become familiar with it.

A trade must be made between flexibility and abstractions. More flexibility can require a user to make additional decisions before running the system. Contrarily, abstractions can limit flexibility by reducing the control parameters that users have easy access to. A less flexible system also has a smaller potential audience and reduced teaching and experimental potential, as it might be less easily reconfigured to explore new signals and applications. This can be seen in the application of existing audio CODEC (a device integrating an analog-to-digital enCODer and a digital to analog DECoder) devices which are unsuitable for non-audio applications. For example, medical or control systems applications may be challenging due to design decisions such as AC coupling, input voltage range, etc. Thoughtful hardware and software design choices can enable a system to serve the needs of many users across disciplines. Therefore, overall accessibility of the system must be considered throughout the design process. Good documentation and design choices in system components and configurations are key to this. As a whole, this requires ensuring that the behavior of the system can be understood by the user without excessive effort. Ideally, users learn about features of a system through examples, hardware labels and software comments. A user should only have to resort to reading documentation for less commonly used features. Sensible default configurations and optional parameters support this approach, balancing flexibility with abstraction. These abstractions can be designed to be flexible; intended to be stripped away progressively to reveal underlying

mechanisms and provide a natural progression for users. This model maximizes both flexibility and configurability.

1.6 Lack of signal processing capabilities in the embedded processor space

Unsurprisingly, a massive array of embedded processing platforms exists, each developed to address some specific needs in the global community of embedded systems users. These range from extensively customizable and powerful devices, such as the XILINX Spartan [3] series of FPGAs (field programmable gate arrays) to the tiny PICAXE [4] platform which marries an 8-pin microcontroller with an implementation of the BASIC programming language.

While there are many teaching tools that focus on topics of control, motivated by robotics applications and on computer systems design, signal processing teaching tools are comparatively rare. As such, there was a major need to move the state of the art forward - a teaching platform that would bring signal processing and its enormously diverse applications to the masses.

1.7 A revolution in microcontrollers

Microcontrollers have been fundamental to most new electronics developed since the early 1980s, and for decades mostly maintained a familiar form - an eight bit architecture with a rudimentary programming environment which demanded that the developer program in assembly language or perhaps a bare bones implementation of the C language. For academia this meant that teaching microcontrollers was often relegated to upper division undergraduate courses where mainly students already versed in programming and electronics would be exposed to them.

Over the course of the last decade, microcontrollers have undergone a revolution. Surprisingly, the most important change has not been performance of the CPUs, but instead the software and ecosystems that surround the microcontroller itself. Previously, devices were paired with a bare bones programming environment and a cumbersome flash programmer. Today, successful microcontrollers are paired with integrated development environments (IDEs) that offer high level hardware abstraction layers (HALs). These self-contained and affordable development boards allow single click programming via a serial interface, and developer sanctioned websites are designed to encourage user collaboration and sharing of supporting libraries and tutorials for the platform.

Arguably, the most prominent example of this change has been the Arduino and the associated ecosystem. The Arduino was originally developed in 2005 by Massimo Banzi (and collaborators) at the Interaction Design Institute Ivrea in Italy as a platform for teaching embedded systems to college students [5]. Prior to this point, many single-board controllers existed supporting a multitude of programming languages and IDEs. The Arduino team hit upon a highly successful combination of price/performance ratio, features, IDE-integrated demos, cross-platform support, simplified programming environment and streamlined setup. Since its inception as a tool for college students, the Arduino audience has broadened to include both more and less advanced users. Engineers have used it to rapidly prototype systems, but it has also steadily progressed down to less experienced users, allowing younger and younger students to use microcontrollers to improve the quality of systems they build. As a result, Arduino is the first microcontroller platform to have gained a reputation in the domain of artists as well as engineers and scientists.

The central element of the Arduino platform is the hardware. It includes a microcontroller on a printed circuit board with a standardized footprint [6]. This board usually contains an 8-bit AVR microcontroller (although alternative processors are now available), the necessary support circuitry to drive the microcontroller, and a USB interface to allow programming of the microcontroller.

Thanks to the standardized form factor of the circuit board, additional client hardware such as motor drivers or sensors can be integrated on mating boards. These boards share the same outline as the Arduino microcontrollers and use pass-through headers to stack on top of the microcontroller boards. These boards, referred to as “shields” in product literature simplify the process of interfacing hardware with the microcontroller by allowing them to be connected without any additional manual wiring. This allows even novices to quickly add functions without having to consult a datasheet. Plus, there is no risk of making frustrating or expensive wiring mistakes. Even for sophisticated users, it can be a significant convenience.

The hardware is complemented by a software platform. The Arduino IDE is a key factor in the success of the Arduino ecosystem. It is built on the “Processing” language, initially developed at the MIT Media Lab by Benjamin Fry and Casey Reas [7] to ease the adoption of Java by students. The Arduino platform uses a fork of the Processing IDE to support an enhanced version of C++ instead of Java, but the outward appearance of the application and ease of use remains similar. The IDE differs from common vendor-supplied toolchains in several key ways. As opposed to commercial tools such as Atmel Studio [8], or Code Composer Studio [9], it has never required the user to complete an online registration process. The Arduino IDE is extremely simple to install, in some cases requiring no action beyond decompressing the archive. On many systems it does not even require a driver installation to be able to upload a user program to an Arduino. With a double click of the executable, a student has access to a wide variety of built in libraries and protocol support that ships with the environment.

The IDE includes a software HAL which provides single-line, intuitive instructions to do many common tasks such as set a value on a digital pin, initiate pulse width modulation (PWM) on a pin, or read the current value of a timer. This type of abstraction is an extremely important improvement. It removes the tedious task of initializing hardware while not actually obfuscating the operation of the device (e.g., using a binary 1 to set a high voltage on a pin) from the user.

Recent versions of the IDE also support download and installation of libraries through an automated library manager, as well as via a simple drag-and-drop operation [10]. These libraries can provide new functionality or support additional hardware and client devices for the microcontroller. These libraries often provide single-line interfaces for new hardware in the same style as the HAL included with the IDE.

The current highly polished state of the Arduino ecosystem represents over a decade of effort by thousands of individuals. Since the first version of the Arduino, the design has been continually refined - its developers adding features small and large that seek to simplify and expand the capabilities of the platform. Some of these features have been immensely successful, such as the addition of a USB-to-serial converter on the board. On the other hand, some features have gained relatively little traction in the community, such as the Due board based around a faster 32-bit ARM type CPU [11], because it is not compatible with most Arduino libraries and shields.

Today, the key features that make the Arduino platform as a whole so immensely successful are its development environment, the ubiquity of its physical form factor, and the network effects resultant from its popularity. Ultimately, the Arduino ecosystem has evolved into a much broader development platform than its humble “simplified microcontroller” roots. The key part of the tool-chain is not the “Arduino” board itself, but instead, the ability to leverage the popularity and familiarity of the Arduino ecosystem to teach new concepts to students and the inquisitive public. With over one million official boards sold [11] and large numbers of clones and derivatives, it has made the microcontroller a tool accessible to millions of scientists, students, and artists who might have been unable to use it otherwise.

1.8 No revolution in digital signal processors

In spite of the changes brought about by the Arduino family and boards inspired by it, there has been little progress in bringing the digital signal processing to the same broad spectrum of users that microcontrollers have reached.

Microprocessors have been an instrumental tool for analyzing and manipulating signals throughout the history of their use. As their performance has increased, they have replaced larger and larger portions of analog signal manipulation hardware in systems. Specialized microprocessors called digital signal processors (DSPs) with instruction sets and hardware architecture optimized to improve the performance of programs handling the type of data typically found in discrete time signals have existed since the late 1970s [12].

DSPs have instruction sets and peripherals optimized to enable higher performance for real-time signal processing tasks at a given clock speed. Multiply and accumulate (MAC) operations are often quoted as the performance metric for a digital signal processor, as it is among the most repeated operation in several key signal manipulations, such as convolution and the Fast Fourier Transform (FFT) [13]. For example, the Texas Instruments C55xx family of fixed-point DSP chips include simultaneous dual-MAC instructions and saturation handling functionality (to address numerical overflow conditions). Optimizations such as these make specialized DSPs an important tool in analyzing and manipulating signals, even as clock speeds of embedded processors increase overall.

These digital signal processors (DSPs) have traditionally been high-performance devices with custom hardware and substantially different instruction sets from traditional microcontrollers. They have generally been high cost devices targeted at professional systems developers, and the cost has limited the size of the user base. The correspondingly small associated support communities reduces the usefulness of the systems for educators and sophisticated hobbyists.

The software that shipped with DSP ICs is usually designed for engineers implementing complete systems for specific applications. Such tools expose the developer to low level code without much streamlining. As such, the creators of these development tools have chosen to emphasize flexibility and performance over ease or speed of development. Thus, they have made few concessions to usability, opting for cumbersome and complicated, yet powerful development environments such as Eclipse [14].

These development environments provide important functionality for an advanced user but can turn away novice users due to the complexity of their initial setup and maintenance. Setup can require configuration of path variables, selection of arcane packages from a long list of similarly named devices, and downloading and installation of supporting drivers and utilities. This is exemplified by the 103-page “getting started guide” for TI’s Code Composer Studio (CCS) [15]. Users must perform repetitive and irrelevant hardware and software setup tasks that do not provide significant educational benefit. The complexity of initializing individual ports with several control commands serves little purpose in teaching, merely adding lines of esoteric code which have nothing to do with the core concepts of signal processing.

By requiring that users learn arcane terminology, install drivers to support step-through debugging, and configure a device build path, systems built around traditional signal processors have been of limited utility for novice or nontraditional users. Students who do not have the robust support network of an in person university course and a course assistant, are likely to be driven off by these complexities. Even with a project-driven need for signal processing tools, users are likely to find them too challenging to set up and learn for a single project.

There have been several examples of DSP hardware that have attempted to address this problem. One example, a development board called the eZdsp by Spectrum Digital, offered a low-cost DSP evaluation module [16], retailing for \$100, as compared to fully featured manufacturer evaluation modules (EVMs), which often cost several hundred dollars [13]. Unfortunately, while it did address the cost challenges of a DSP toolkit it did not make a substantial improvement in user experience, which is ultimately at least as significant barrier to entry for signal processing as cost. The eZdsp platform failed to address the shortcomings of the software environment that surrounds the hardware. It relies on CCS which, as described earlier is cumbersome to install and initialize. The tool also lacks a modern HAL, making configuration of peripherals complex. Additionally, these proprietary tools are often have limited compatibility with various operating systems. CCS and Atmel Studio for example, are incompatible with Macintosh, excluding a substantial minority of users.

1.9 Signal processing block diagram tools

There have been a number of educational tools intended to ease the teaching of signal processing through maximized abstraction. These opt for solutions in which users manipulate “block diagram” models. Signal manipulation is achieved by connecting signal blocks together mimicking signal flow diagrams instead of the code-based model used successfully in microcontroller environments like that of the Arduino. This conforms with an approach commonly taken in signal processing texts and lecture materials [17]. Figure 1.1 depicts a typical signal chain developed in a block diagram style tool, Simulink (The Math Works Inc., Natick, MA) [18], wherein a student constructs a system by graphically connecting predefined blocks. Unfortunately, these tools tend to abstract away the more nuanced challenges of practical implementation. In so doing, they do not provide users the ability to interact with the processes underlying these abstractions. Users of these systems may never manipulate an individual sample or data point. The cost of this abstraction is that since students do not manipulate samples directly, they will not gain an intuitive understanding of what a single “sample” actually represents, nor how it relates to a signal as a whole, even with explanation. That understanding underlines the real value of a hardware based approach to teaching signal processing concepts.

The teaching tools based on the block-diagram model can be broadly separated into two sub-categories. The first category is comprised of software applications for the PC. These applications use a full desktop PC to perform all signal processing tasks. They either manipulate simulated data, recorded data or data from simple signal acquisition systems. The second category is dedicated signal processing hardware that uses a PC to program a flow-style signal chain, but then performs signal processing on a standalone device. Generally, these devices process data acquired in real-time as opposed to simulated or recorded data.

The most popular PC-based signal flow diagram-based tools are the Simulink environment and Lab View (National Instruments, Austin, TX) software packages [19]. Other systems that are similarly structured exist, for example solidThinking Embed

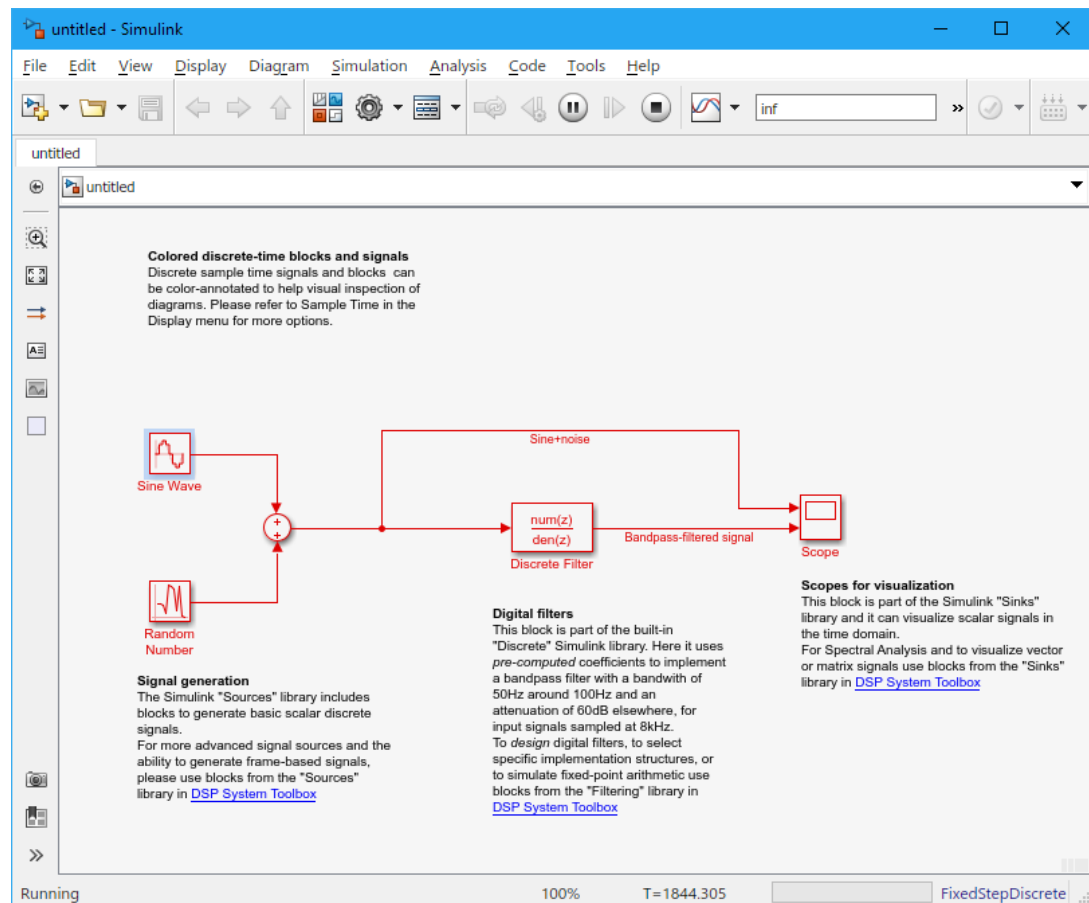


Figure 1.1: Example of a block diagram-based tool implementing a low-pass FIR Filter in Simulink.

(solidThinking, Inc., Troy, MI) [20]. These require a PC to operate and the result cannot be embedded in projects. This software-only approach has been one of the most successful options for teaching signal processing, but it provides a drastically different experience from the real systems that students later implement, and completely removes the real-time component of signal processing design. As a group, these systems provide opportunities to practice some skills developed in a signal processing theory class, but a gap remains between students and limitations of practical implementations.

In contrast, a hardware-based, signal-flow-diagram system is the SignalWizard developed by Prof. Patrick Gaydecki at the University of Manchester. This tool uses switchable signal blocks to produce a system that can allow a flow chain to be implemented in minutes, but the system is based on an audio CODEC, which is not capable of accessing signals below 20 Hz, and hence is unsuitable for biomedical applications. At \$699, it is prohibitively expensive for students to individually purchase for use in a course or to embed into projects. With two of its dimensions exceeding 16 cm [21], it is also too bulky to embed into most fieldable applications. Additionally, the need for a PC as a control device hinders its use as a component of an embedded system. Nevertheless, the ability to manipulate signals in real-time represents a significant improvement in the fidelity of the user experience, and this capability influenced the requirements for the ASPEN system.

1.10 Microcontroller tools do not support DSP

A new generation of 32-bit Microcontrollers such as the STMicroelectronics Nucleo and Particle (formerly SPARC Core) have begun to reach the market in the late 2010s. This new generation of processors have begun to have sufficient performance to process interesting signals without customized hardware. Only a handful of toolchains developed for traditional microcontrollers have even rudimentary support for signal processing.

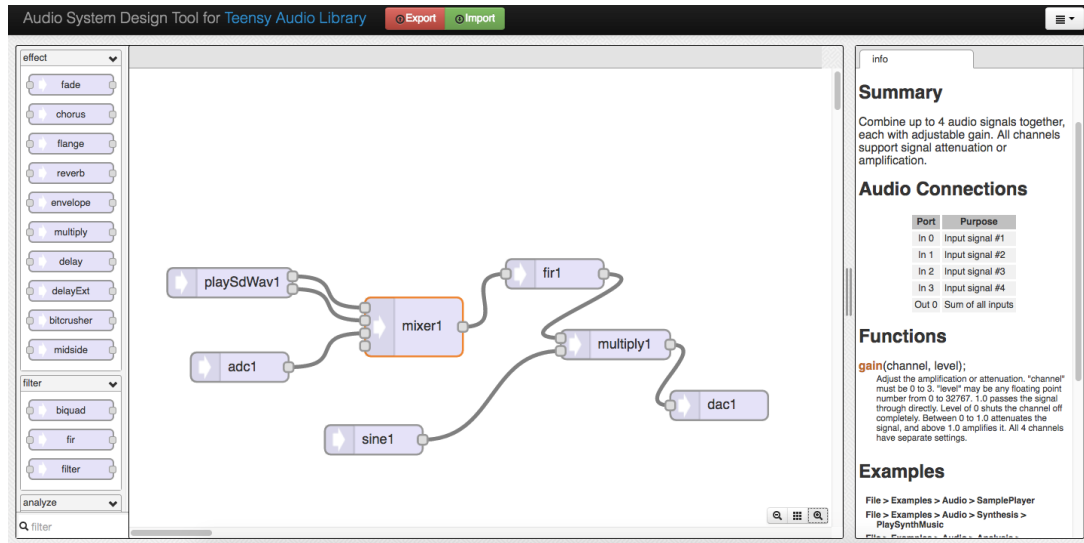


Figure 1.2: Example of Teensy Signal Chain Development [23].

A prominent recent example is the Teensy series. These compact development boards are built around ARM architecture-based microcontrollers. The most recent 3.6 revision has sufficient performance to perform basic filtering tasks on audio and most importantly the platform has a stream-based audio library available [22]. As with many Arduino-inspired spinoffs, the boards work with a patched version of the Arduino IDE. Unlike many other efforts however, the boards are not footprint compatible with standard Arduino microcontrollers. This means that the ecosystem of Arduino compatible shields cannot easily be directly interfaced with them, although adapter boards have been created.

An audio library shown in Figure 1.2, has been developed for the Teensy platform. It can be used with its onboard ADC and DAC signal chain, or with an add-on AC coupled CODEC board (described below, which acts as a sound card and provides additional resolution and stereo capabilities). The system however, does not make it straightforward for students or developers to implement custom signal blocks, and lacks the general-purpose capabilities needed for signal manipulation and education, as well as a convenient way to perform sample-by-sample signal manipulation,

which provides insight into practical challenges in implementing signal processing algorithms. Furthermore, the performance of the Teensy 3.6 is hampered by the (dual-channel) unipolar twelve-bit DAC outputs 0 to 3.3 V which somewhat limit the span of applications for the system. If a user needs higher performance, there is an additional adapter shield for the Teensy with an audio CODEC onboard [24]. It is possible that the sound shield may be able to be modified to support DC sampling, this has not been tested. As a whole, the Teensy platform represents the most meaningful alternative to the DSP Shield as a component of the ASPEN platform. The two groups have similar approaches to hardware, although the ASPEN approaches the software side of the challenge with more emphasis on the details of signal processing.

Another microcontroller based approach has used minimalist hardware augmentation, using an add-on board and software with simple hardware. One such example is the “Plain-DSP” package, which uses an Arduino Uno along with some software libraries, and a very simple shield using a quad op-amp, some resistors and capacitors. While such a device can be used for some very basic, low-frequency applications, the extremely limited performance of such a lightly augmented system become apparent within minutes. The Arduino Uno that the Plain DSP system uses has low computational performance which prevents all but the most basic algorithms from being implemented. Furthermore, the Uno lacks a conventional DAC and outputs labeled analog by the system are in fact pulse width modulated outputs, limited to about 100 Hz (without modification to the Arduino environment). With such low performance, the system abandons most of the learning potential, as well as all potential as a development tool for interesting applications.

1.11 Embedded desktop systems

Recently, single-board computers designed to be used like desktop computers have made a reappearance. Much like single-board computers of the 1970s and 1980s, these systems are self-contained computers of moderate performance which can run similar

software and interface with similar peripherals as desktop PCs. The Raspberry Pi is one of the earliest and most successful of this wave of systems. Its origins can be traced back to 2006 as a project to develop the system as a low-cost, high-performance educational platform for teaching computing and programming to children [25]. Due to the complexity and performance of the hardware, it usually runs a variant of the popular Linux operating system. This allows a wide variety of applications built for the operating system on other platforms to be compiled for the Raspberry Pi. This includes sophisticated mathematical and programming tools such as Matlab (The Math Works Inc, Natick, MA), Wolfram Alpha, and Python to be used directly on the Raspberry Pi. Since the Raspberry Pi is a single-board computer, these solutions can be embedded in projects and devices [25]. Although the Raspberry Pi arrived on the market after the ASPEN project had commenced, its capabilities have been evaluated. The major limitation of using an operating system designed for a desktop and server computers is that it is not designed for real-time operations. Due to background processes, tasks like sampling cannot be guaranteed to be uniformly spaced in time, which means that many signal processing algorithms will not work reliably.

An alternative operating system designed for real-time operations, referred to as an “RTOS” could be used to guarantee deterministic sampling, but these operating systems are not as popular and thus sacrifice the advantages of community support and a large pool of compatible software while adding the complexity of a full operating system. While an alternative real-time operating system or “bare metal” development environment could be created for the Raspberry Pi platform, it would require much of the same development undertaken for the ASPEN project and would lack key benefits of the ASPEN project design, such as Arduino shield compatibility.

Using the most popular Linux variant “Raspian” and “Pure-Data,” a block-style signal processing programming language [26], and an external sound card, it is possible to construct an audio signal processing toolchain. These tools, however, do not provide sample-by-sample signal manipulation and thus are subject to the same limitations as the block diagram systems described above. Furthermore, without modification,

the audio CODECs used by these systems are limited to a 20 Hz to 20 kHz band and thus are not optimal for all the signals of interest here.

Several other similar tools have arrived on the market, such as the oDroid and BeagleBone platforms, all of which suffer from the same limitations as the Raspberry Pi, but are further hampered by the challenge of smaller communities which leads to less online support and fewer examples.

1.12 Lack of tools that meet the requirements

At the outset of the ASPEN Project, none of the systems described above or found after research met the defined needs of teaching analog electronics. Nor did they enable the construction of a class of devices that opens signal processing and biomedical electronics to the masses. Some of these goals could be pursued with a hybrid of several tools but no existing tools meet all the needs defined here. Therefore, developing a new hardware and software system to address them has potential for a substantial improvement in utility and accessibility.

1.13 ASPEN architecture

The result of the consideration of the needs of the project and the existing solutions at the time was a proposal for a new system to achieve these goals. The system leverages the immense popularity and simplicity of the Arduino platform. It consists of a hardware stack with three boards. First is an ordinary Arduino Uno, the most common version of the Arduino board. Next is a new DSP based board, called the DSP Shield. This device can act either as a standalone development system or as a shield supporting applications on the Arduino. The final board in the stack is a support shield called the Analog Shield which can be used in conjunction with either of the other two boards. This board provides an analog front end for signal acquisition

and support infrastructure for building circuits, while allowing for a flexible system configuration, allowing the DSP Shield to be excluded in applications where it is not necessary.

Most critical to the system, however, is the software. For the Analog Shield, a library with a simple interface that enables function calls similar to existing Arduino interfaces was conceived.

Envisioned for the DSP shield was an Arduino-derived development environment that leverages software interfaces and paradigms to extend the Arduino approach to signal processing and biomedical electronics. Achieving this simplicity requires implementation of several familiar systems, including a HAL, serial bootloader, IDE, and compiler tool-chain. Furthermore, all these systems must be designed so that when they must differ from existing Arduino paradigms, their default operation is intuitive and meets the needs of as many users as possible. This is especially critical in the implementation of the core innovation the DSP shield provides - the signal processing libraries. These must be written so they can be quickly grasped through example and that the user can see how to utilize them to provide ever greater functionality.

To address the hardware needs, the Analog Shield has a quality DC-coupled CODEC to allow for low-frequency signals in biomedical systems and frequencies above the top end of the audio range, matching or exceeding the bandwidth of on-board CODECs in microcontrollers. Furthermore, biomedical signals demand a high dynamic range to measure signal deviations on the order of tens of microvolts, something that common microcontrollers do not achieve with inbuilt converters, which typically have an ENOB of 8-10 bits. Higher ENOB dramatically simplifies acquiring the small signals in this range. Adding multichannel capability - at least two but preferentially four or more, can allow for fusion of multiple signals.

1.14 Approach to the system

Having defined the need and an overall approach to designing a platform for teaching and prototyping, development began. The next chapter includes an outline of the system hardware design and trade-offs. The third chapter includes a discussion of the ASPEN software architecture and design. Applications developed to demonstrate the utility of the system in mixed-signal education and development are discussed in Chapter Four. Application of the platform to digital signal processing is described in Chapter Five. In Chapter Six, applications of biomedical electronics using the system are detailed, including a novel ECG topology developed with it. Finally, conclusions and future work are covered in chapter seven.

1.15 Conclusion

As a whole, ASPEN represents a concept for a new system that can address the needs of building and teaching analog electronics, signal processing, and biomedical instrumentation, both in the lab, and in fieldable applications. It can also create a path to clinical-quality, open-source and highly functional medical devices for learning and cost-constrained clinical use, in the developed and developing world. A successful result will enable the open-source community to solve real problems in medicine using safe and inexpensive hardware. The preceding review of existing and emerging systems revealed that there is a gap between the challenges in addressing these tasks and the tools previously available.

Chapter 2

Hardware

2.1 System Design Approach

In order to meet the needs defined in Chapter One, new hardware was needed. As discussed, it was necessary that it be low-cost, flexible, maintainable, and capable of being understood by users without continually consulting documentation. This is termed “discoverability,” meaning that the user can learn about the system through use instead of reading dry technical documentation [27]. To achieve this, the system was conceptualized as a stack of three devices. This physical separation makes it easier for the components of the system to be individually understood by the user. Practically, it also enables users to select subsets of the ASPEN system for downstream applications, enabling cost and complexity to be tailored to application needs.

The Arduino UNO is an excellent example of a discoverable hardware system. The printed circuit board has clearly silkscreened descriptions of its inputs and outputs. The headers are keyed ensuring shields are connected correctly. Naming conventions are chosen to be self-explanatory. Figure 2.1 depicts a top-view of an Arduino UNO board of the type used in the ASPEN project. Examples of good discoverability can be seen in touches such as the “ ” marks next to PWM-enabled pins and the

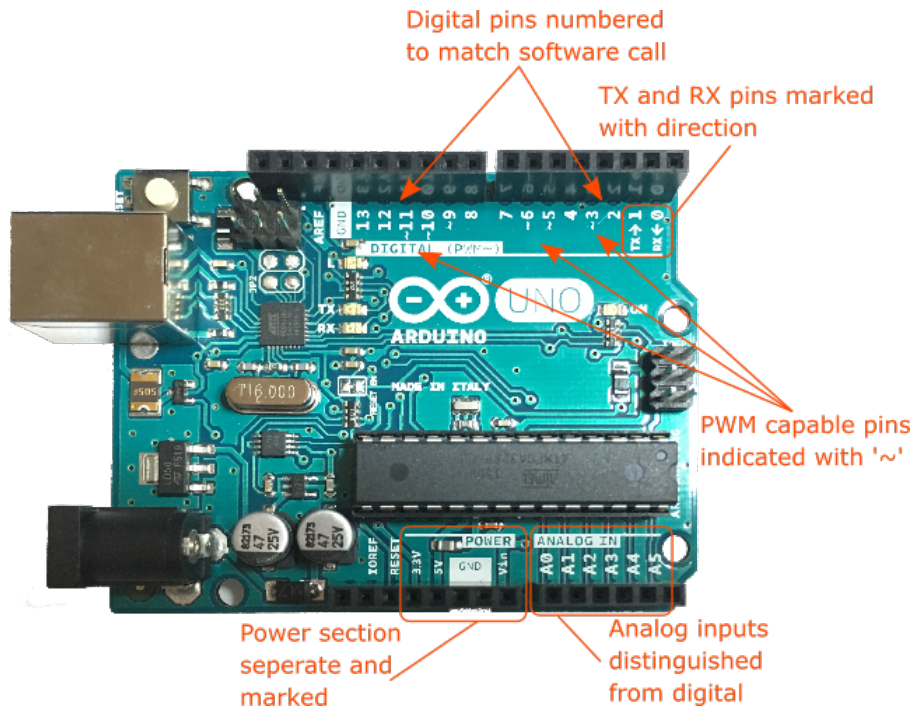


Figure 2.1: Top view of Arduino Uno highlighting discoverability of the design.

serial port being marked “TX” and “RX.” These may initially seem like small design flourishes, but together they dramatically streamline the user experience, making the system more approachable.

The two boards developed for the ASPEN project extend the capabilities of the Arduino ecosystem. The Analog Shield provides high-quality infrastructure for building and interfacing with analog circuits and the DSP Shield provides signal processing capabilities which allow the user to manipulate data in ways that could not be achieved by processing signals on the Arduino alone.

The following sections cover trade-offs and decisions made in the design of these boards as well as some of the testing performed to validate their functionality.

2.2 Analog Shield Design Overview

The Analog Shield, depicted in Figure 2.2, contains the support infrastructure for bridging the analog and digital domains. It contains a convenient interface for a multi-channel, high-fidelity ADC as well as a matching DAC. These are supported by software methods to connect to the digital component of the platform (discussed in Chapter Three).

There are built-in bipolar power supplies, which can substantially simplify circuit construction. The system is also designed to provide a place to actually construct the circuits in a compact manner, in the form of a small solderless breadboard (optional).

As opposed to a more general “breakout board” form factor, the Analog Shield uses the Arduino Shield footprint. This easy-to-use form factor is familiar to many students. It includes standard pass-through headers, allowing the shield to be interfaced with the Arduino and stack with shields above. This form factor also provides a structurally stable platform for circuit construction and avoids the need for a student to run wires to a separate board or to power supplies and converters.

An alternative design might use a standard footprint and interfaced via cable to microcontrollers of any form factor. The Arduino footprint however, benefits from the broad popularity of the ecosystem and wide variety of Arduino-like devices. A number of alternative microcontrollers, from approved ones such as the Arduino Yn [28] to less widely known competitors like the Arch Pro [29], have been developed to use the same footprint. Like many other ubiquitous connectors, the Arduino Uno footprint is likely to outlive the Uno itself as a standard for microcontroller daughter boards by a significant margin.

2.3 Analog Shield Breadboard and Power Supply

To facilitate users constructing circuits and interfacing them with the Analog Shield, the decision was made to allow for the optional placement of a solderless breadboard in the top center of the board. This layout is similar to the design used in the Adafruit ProtoShield [30], which has proven to be both popular and extremely useful in educational settings.

The final version of the design can be seen in Figure 2.2. The included small (seventeen row) breadboard is provided with adhesive tape backing and can be placed in the open area in the center of the Analog Shield by the user. To enable the use of the breadboard, many active components of the power supplies and converters were placed on the bottom side of the shield, but some top side components must be covered by the breadboard. A manually adjustable power supply was also included in order to facilitate compact circuit construction as many circuits need voltages other than the +5 V and +3.3V available on the Arduino UNO platform.

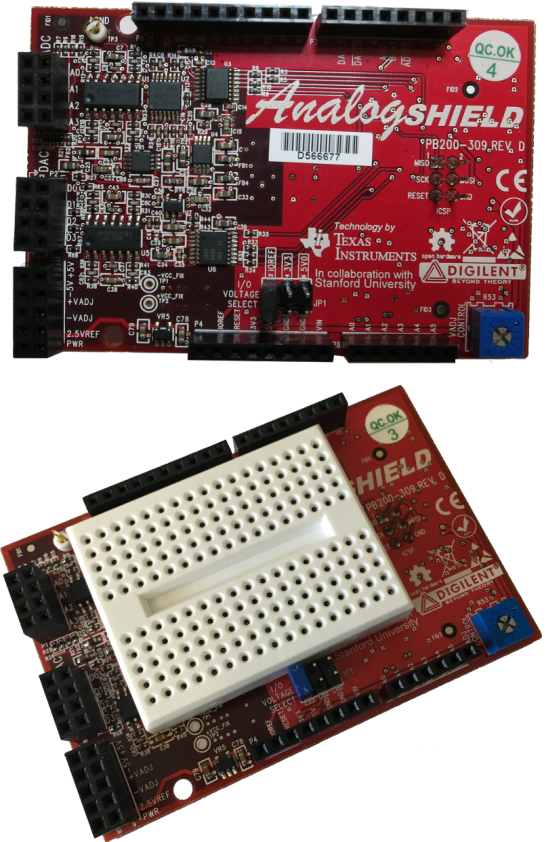


Figure 2.2: Analog Shield as viewed from the top, without included breadboard (above) and with breadboard installed (below).

Since many signals are centered around 0V (ground), measurement circuits often are most naturally implemented with both positive and negative supply voltages. This need typically presents circuit designers with a challenge. They can use an external supply to provide positive and negative voltages or design their circuit for single-supply operation using the UNOs built-in +5 or +3.3V supply voltages. Such single-supply design generally results in increased circuit complexity, and requires additional capacitors and bias resistors to offset an AC signal to the UNOs input range between zero and five volts. This also often necessitates constructing a virtual ground system which requires placing an extra operational amplifier and several more resistors and capacitors on the board. In a portable system, the large through-hole components commonly used by students and makers would mean that the single-rail power supply alone occupies most of the room on the included breadboard. This would severely limit the utility of the system.

To address this, two bipolar power supplies are included on the shield. The first is a fixed ± 5 V supply and the second is an adjustable ± 7.5 V supply. An important design choice made when developing the power supply was the user interface, so the final design uses a single-board-mounted potentiometer to set the voltage of both the positive and negative supply rails. Digital control of the supply voltage was considered, and did offer some advantages. Perhaps surprisingly, digital control would have been lower cost than adding a physical potentiometer to the board. While it would have allowed user-written software that sequenced or modulated the output voltage at low frequencies, a digital interface has significant downsides. Firstly, it increases complexity in managing power up and power down of circuits. Secondly, digital control would have reduced the discoverability of the supply interface itself, as there would have been no obvious way to adjust the supply without downloading software and uploading a program to an Arduino. Finally, it would mean that the power supply had software performance cost and the support library for the board would have to be invoked for every circuit using the board, not merely circuits using the ADC and DAC on the shield.

Another factor influencing the design of the power supply was compatibility with the Analog Shield ADC and DAC. The nominal input voltage range of these devices is ± 5 V. To ensure ease of building circuits that can use the full dynamic range of these inputs and outputs, the power supply was designed with a maximum output range of ± 7.5 V. This extra capacity allows for a wide variety of components to be used, including older low-cost (non rail-to-rail) operational amplifiers with a large difference between their supply and output voltage ranges.

The core component of the supply is a TPS61093 (Texas Instruments, Dallas TX) switch-mode converter. As designed, the system offers overcurrent protection at 300mA. Because the applications envisioned are primarily interface and sensing circuits, a maximum supply current of a few hundred milliamps is sufficient. Unlike a typical benchtop power supply, there is no constant-current limiting. Instead, the device uses a self-resetting polyfuse to disconnect the output from the input in case of an overcurrent event, shutting off power to the load. To reset the system, the user removes and reapplies power to the device. This can potentially lead to user confusion but protects a student from destroying what is potentially the only variable power supply available to them. While a constant-current mode might be considered for a future revision, it would have required several additional components. These include a relatively costly current set potentiometer, a current limit indicator, the current control circuitry and more complicated explanatory graphics. This would have driven up cost and board area without substantially improving user experience. Thus, current-limiting functionality was omitted from the design.

Experience in teaching courses using the Analog Shield has shown that the power supply has been trouble free and has not been a source of student confusion, providing evidence that the choice of a simple design was successful.

2.4 Analog Shield ADC and DAC Hardware

For the Analog Shield ADC and DAC, the decision was made to use a minimally integrated solution. There are numerous and popular single-chip solutions, called CODECs. These ICs handle conversion to and from the digital domain. For the Analog Shield, there are several advantages to avoiding an integrated solution. First, such a design conflicts with the teaching mission, obscuring the sampling process with configuration routines and data buffering memory. Additionally, as discussed in Chapter One, CODECs typically contain input and output 20 Hz high-pass filters, making them unsuitable for biomedical applications envisioned for the system.

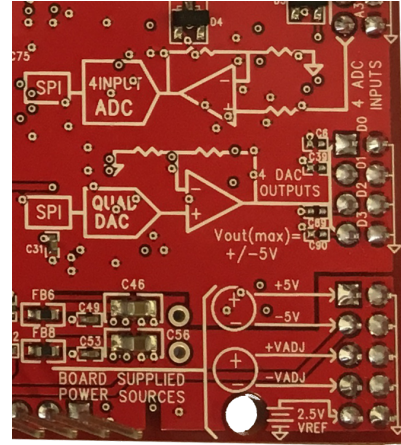


Figure 2.3: Photograph of the Analog Shield bottom, showing silkscreen. Top: ADC and DAC buffer. Bottom: Power supplies.

Thus, the Analog Shield was designed with discrete ADC and DAC devices. Simple op-amp buffers were added for circuit protection and to allow support for a wide range of input impedances and output loads. This non-integrated design also allowed for simplified circuit schematics to be printed on the bottom of the board, providing the user with information on the behavior and design of the system without the need for a manual. Figure 2.3 shows a detailed view of the art silkscreened on the bottom of the Analog Shield. It shows a high-level diagram the ADC, DAC, input signal conditioning circuitry, and power supply interfaces.

The ADC used in the Analog Shield is a four-channel successive approximation (SAR) converter, part number ADS8343 (Analog Devices Inc, Norwood MA). It has a serial peripheral interface (SPI), one analog-to-digital conversion circuit and a four-channel

multiplexer. The converter is specified as having a 16-bit resolution with a maximum sample rate of 100 ks/s (given a sufficiently fast microprocessor to sample). It can be configured as either a two-channel differential or four-channel single-ended converter which samples inputs successively, as shown in Figure 2.4. If all four channels are in use, each individual channel can be sampled at 25 ks/s. With a single channel in use it can be sampled at the full 100 ks/s rate. At higher input frequencies (over 10 kHz) a 90 dB spur free dynamic range (SFDR) limits the ENOB to 14.7 bits, decreasing further as frequency increases, resulting in an overall specified ENOB of 13.6 bits [31]. When used directly with the Arduino Uno platform, the ADC has been observed to have a maximum sample rate of only 70 ks/s, when acquiring a burst of samples. This is due to the performance limitations of the Arduino itself. When used with a higher performance processor such as the DSP Shield or the ChipKit Uno32 (Digilent, Pullman WA), the maximum 100 ks/s sample rate has been demonstrated.

The DAC used in the Analog Shield is a four-channel string DAC type device, part number DAC8564 (Texas Instruments, Dallas TX). As with the ADC, the DAC has four channels and a nominal sixteen-bit resolution. It has a settling time of 10 μ s, providing a 100 ks/s sample rate, which matches the performance of the ADC. Like the ADC, the DAC is also a serial device, sharing the SPI communication bus. As with the ADC, the choice of a simple part means that there is minimal configuration necessary.

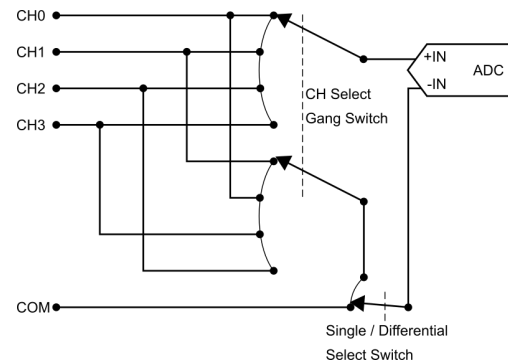


Figure 2.4: High-level ADC diagram, depicting channel multiplexer and selection between differential and single ended modes (after Burr-Brown, 2003).

The choice of the SPI bus for both devices was a measured one. Higher bandwidth could have been achieved using another bus, such as the high-speed I2S standard, but less commonly-used standards would limit device compatibility with microcontrollers

commonly used by students and experimenters. Choosing SPI devices represented a balance between speed, pin count, simplicity of implementation, and popularity of support across platforms.

The hardware choices made in designing the Analog Shield reflect the design philosophy and goals of the ASPEN system. The straightforward design contributes significantly to the overall goal of building a system that a student can understand and extend as needed. It also ensures that the system can continue to be maintained by the community with minimal modification even if a specific component reaches the end of its production life.

2.5 Characterization of the Analog Shield

To validate the functionality and performance of the Analog Shield, it was tested as an entire system to validate the functionality of the Analog Shield as designed.

In the process of developing and testing the Analog Shield, four revisions were created, with the final released version named “Rev. D.” Seen at the left of Figure 2.5, the initial version of the Analog shield was a highly-optimized design from an aesthetic and user interface perspective. It contained no integrated circuits on the top side of the board, leaving that area entirely clear for placement of the included breadboard, as discussed in Section 2.3.

Unfortunately, initial testing of the first revision of the Analog Shield revealed a large amount of noise. This was the result of several design issues, including interference from the stacked Arduino Uno but primarily from proximity of the ADC and DAC to the onboard switched-mode power supply. The noise was present with the shield stacked, but was still substantial with the shield remotely connected with extended wires and RF shielding. With a DC input, the ADC count varied by more than 50 mV, meaning the effective number of bits (ENOB) was less than 7.1, as shown in Figure 2.6. The noise floor was only improved to 7.9 bits with the Uno separated from the

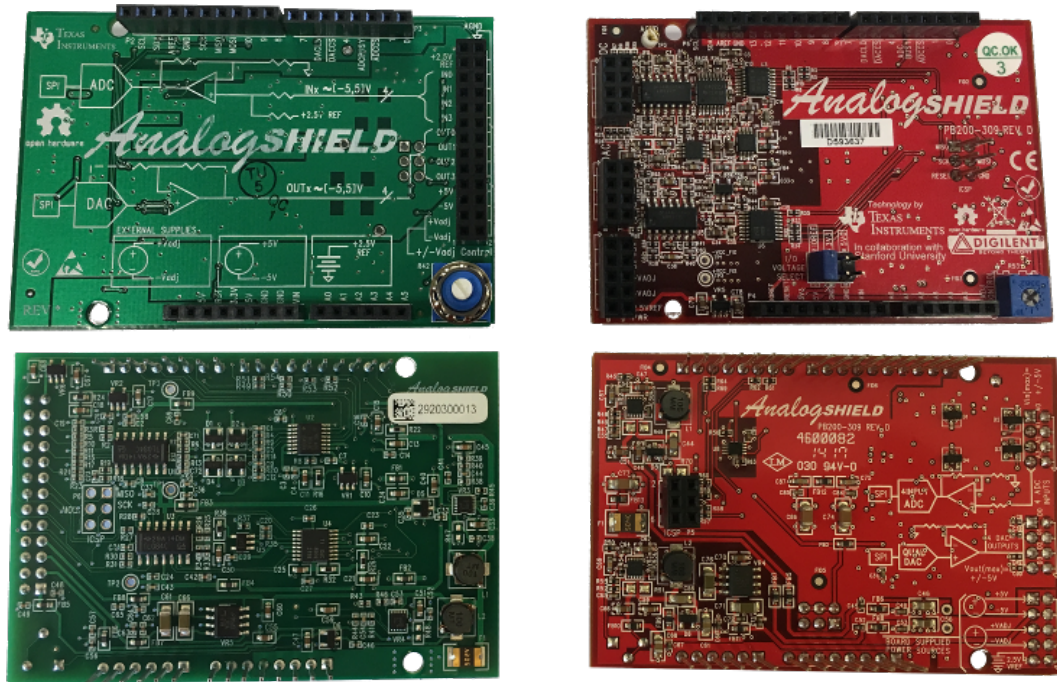


Figure 2.5: Comparison of Analog Shield, first (left) and final (right) revisions, top side of boards above, bottom side of boards below.

shield by extended wires. This is far worse than the performance of the 10-bit rated performance of the ADC built into the Arduino Uno - rendering this version of the Analog Shield effectively useless. Given the demonstrably poor performance ceiling of the ADC, a detailed evaluation of this version of the shield was not performed.

To address this problem a design compromise was made. While keeping costs reasonable as well as maintaining the general desired form factor, a slightly more expensive and complex four-layer

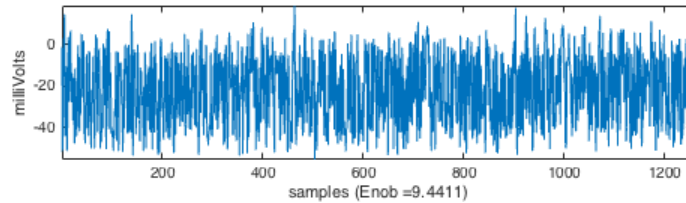


Figure 2.6: Plot of data sampled by ADC of the first revision of the Analog Shield, with a grounded input.

board design was chosen. Components were moved to the top side of the board, as can be seen at the right-hand side of Figure 2.5. This allows the ground and power planes within the board to shield the converters from noise. The design also features increased spacing between the converts and power supplies to reduce signal coupling. The components in the “clear” area on top of the board are low profile, and uniform in height. Even with these design compromises, a solderless breadboard can be adhered on top of the components using foam-backed double-sided tape, albeit with a slight tilt. A secondary result of this redesign was that the silkscreen information describing the inputs and outputs of the shield was moved to the bottom of the board. While these graphics are smaller than the original version, they have the advantage of being visible to the user even after the breadboard is permanently attached.

To evaluate the performance of the Analog Shield, the ADC and DAC were each tested. This procedure was more detailed than the testing performed on the initial batch of shields. In these tests, the Analog Shield was tested stacked with an Arduino Uno, to evaluate real-world performance. A SR1+ Audio Analyzer (Stanford Research Systems, Sunnyvale, CA) was used as both the sinusoidal source used for ADC testing

```

unsigned int vect_in[LEN] = {0};
void setup() {
    initialize();
    Serial.begin(115200);
    while (!Serial.available() || (Serial.available() == 1 && Serial.read() != '1'))
    {};
    while (Serial.available())
    {
        Serial.read();
    }
}

void initialize()
{
    cli();    // Disable global interrupts while we make our changes.

    TIMSK0 &= ~(1 << TOIE0);    // Disable Timer 0.
    TIMSK2 &= ~(1 << TOIE2);    // Disable Timer 2.

    |
    sei();    // Enable global interrupts.
}

void loop() {
    cli(); // UDRE interrupt slows this way down on arduino1.0
    for (int i = 0 ; i < LEN ; i++) { // save 256 samples
        vect_in[i] = analog.signedRead(0); // put real data into bins
        //delayMicroseconds(50);
    }
    sei();
    printHex(vect_in[i]);
    while (!Serial.available() || (Serial.available() == 1 && Serial.read() != '1'))
    {};
    while (Serial.available())
    {
        Serial.read();
    }
}

```

Figure 2.7: Partial listing of ADC data acquisition program run on Arduino for Analog Shield Characterization. Some setup and serial printing scripts omitted for brevity. The entire program is included as Appendix A.

and FFT analyzer for DAC testing, unless otherwise noted. The SR1+ sine source has a -112 dB THD+N when used in the 1 kHz configuration used for these tests [32], making it suitable for characterizing the Analog Shield ADC which has a theoretical -86 dB SINAD [31].

To evaluate the Analog Shield ADC, a sinusoidal signal with a fixed 1 kHz, 4.5 V amplitude, 0 V mean signal was input (unterminated) to the ADC under test. For each test, a Matlab script on a PC requests a frame of sample data from a program on the Arduino. In response, the Arduino program (depicted in Figure 2.7) disables interrupts to reduce sample jitter and captures a burst of 801 points of sample data at the maximum possible rate. For the Analog Shield, this was 63 ks/s and required no special configuration, verified by sweeping an input signal and measuring the nyquist frequency. The native Arduino `AnalogRead()` routine had to be bypassed with direct register calls to obtain a 36 ks/s rate with the on-board Arduino ADC (using `AnalogRead()` directly further limits the rate to 9 ks/s). To prevent interference from serial transmission, the program transmits the frame to the PC after acquisition is complete.

The Matlab script windows the received frame using an 801-point Hann window, transforms it to the frequency domain and stores it in a buffer. The process repeats for 250 frames and the resulting frequency domain signals are averaged. Power spectral density, SNR, SINAD, SFDR, and more are computed from this data.

For comparison, these tests were run on the Analog Shield and the Arduino Unos on board ADC. In all cases, the input sinusoid was 1 kHz. Figure 2.8 shows the result of this comparison. The redesigned shield substantially improves on the performance of the Arduino Uno, with an SNR of 81 dB as opposed to the 59 dB seen on the Arduino. This results in a 13.0 bit ENOB, within 0.6 bits of the ADS8343s specified ENOB of 13.6 bits. It is an improvement over the measured 9.3 bit ENOB of the Arduino. This 22 dB improvement in SNR is substantial, and a 13-bit ENOB is sufficient for many biomedical applications, when leveraging oversampling techniques.

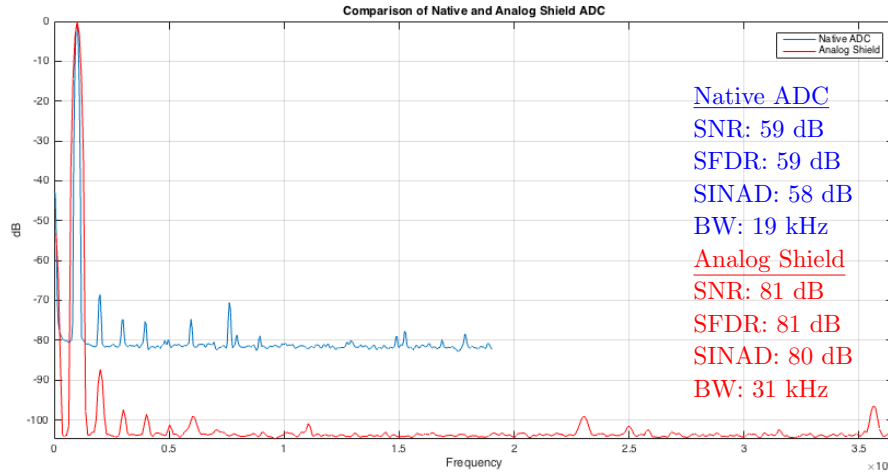


Figure 2.8: Analog Shield ADC Performance characterization, depicting a 1 kHz test signal input to both the Analog Shield and the Native ADC for an Arduino Uno. Each signal is an average of 250.

While at first glance a 3.7 bit improvement may seem trivial, it is critical for the acquisition of biomedical signals as it greatly relaxes the need for DC offset and drift compensation circuitry in the design of systems such as electrocardiograms, as will be shown in Chapter Six.

To evaluate the Analog Shield DAC, a stored 512-point, sixteen-bit sinusoid was output via the Analog Shield DAC. The signal had an amplitude of 4.5 V and a sample rate of 66 ks/s, producing a 130 Hz sinewave. As with the ADC, the DAC signal was captured using an SR1 audio analyzer, in high-resolution mode producing a 3200-point FFT with a span of 1.5 kHz. These results can be seen in Figure 2.9. The experiment showed an SNR of 79.5 dB for the DAC, a THD of -111.4 dB and a SINAD of 79.5 dB. This results in an ENOB for the Analog Shield DAC of 12.9 bits.

For the Arduino onboard PWM output, a Direct Digital Synthesis (DDS) routine by Martin Nawrath of the Academy of Media Arts Cologne [33] was used. This DDS routine generates a sinusoidal signal using a PWM output with a 31 kHz switching

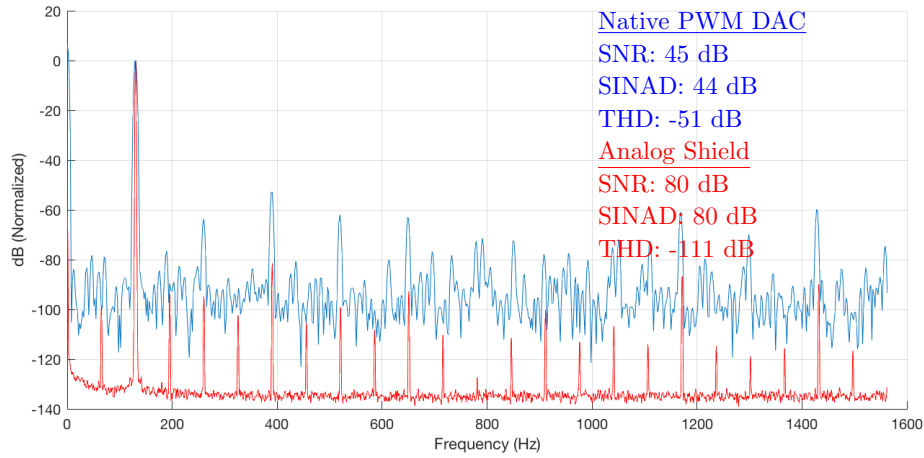


Figure 2.9: DAC performance characterization, depicting a generated 100 Hz sine wave with both the Analog Shield and the native Arduino PWM.

frequency. A robust DDS implementation like the one used here represents a good standard to compare against, as it uses a much higher switching frequency than the Arduinos standard `AnalogWrite()` routine, which uses a 980 Hz PWM frequency [34], too low for practical signal generation. This bandwidth improvement is not of interest in this test, however. This approach offered best-case test conditions for the built-in Arduino PWM output.

Measurements for the Arduino PWM were made using a SR780 (Stanford Research Systems, Sunnyvale, CA), generating a 801 point FFT with a 1.6 kHz span and with a SR650 (Stanford Research Systems, Sunnyvale, CA) filter at the input. The SR650 was configured as a 1.6 kHz low-pass filter, with a high-impedance input (1 M Ω). The result of these tests showed that the Arduino PWM output has a 44.8 dB SNR, 50.9 dB THD and 43.9 dB SINAD performance. The Analog Shield DAC offered a 35 dB improvement in SNR and 34 dB improvement in SINAD as shown in Figure 2.9. This means that the ENOB of the Analog Shield is 12.9 bits as opposed to the 7.0 bit ENOB using the native PWM output with external filtering.

Overall, the characterization of the Analog Shield demonstrated that the system significantly improves on the capabilities built into the Arduino Uno board. As expected they do not quite match the theoretical specifications of the converters used on the board, but the performance is reasonably good. The overall feature set of the system, coupled with measured performance represents a substantial base of capabilities for ASPEN as a signal acquisition and generation system. After validation, the open-source hardware design of the final version of the Analog Shield was made available commercially by Digilent [35], and has been used in courses at Stanford and elsewhere, as will be described later. Appendix B contains a bill of materials and schematic for the Analog Shield.

2.6 DSP Shield

As a part of the ASPEN system, the DSP Shield is designed to address the signal processing needs outlined in Chapter One - specifically to facilitate embedding and teaching of signal processing, analog and biomedical electronics. As discussed, a key part of the designs and interfaces for the project revolve around compatibility with the Arduino platform and design metaphors, which are familiar to much of the intended audience. As a part of this mission, the DSP Shield design needed to both be accessible to new users and powerful enough to serve the needs of advanced users. The design emphasizes flexibility, ease of use, performance and cost. It can be used both as a master and slave device, coupled with, or independent of the other components of ASPEN. Furthermore, the Arduino Shield form factor allows it to inter-operate with other third-party shields that use the same footprint (and do not have electrical pin conflicts). Table 2.1 lists some of the most popular and useful shields that are electrically compatible with the DSP Shield (although drivers have not necessarily been created for them) [36] [37] [38]. Figure 2.10 shows the DSP Shield, and highlights some of the components chosen to address these goals.

Shield	Description	Pins Used
Analog Shield	analog-to-digital prototyping.	2,3,5,6
Arduino Ethernet Shield	Wired network commnuications.	10-13
Arduino WiFi Shield	Wireless network commnuica- tions.	3,4,9-13
Arduino Motor Shield	Motor driver.	1-13
Arduino USB Host Shield	Control specific USB devices	7-13
Gameduino 2	Graphic LCD & touch	2,8,9,11-13
Batsocks TellyMate TV	Composite video output.	0,1
Adafruit RGB LCD Shield	Character LCD.	A4-A5
Adafruit 2.8 TFT Touch Shield	LCD display with touch screen.	4, 8-13

Table 2.1: A list of some of the most popular Arduino Shields electrically compatible with the DSP Shield.

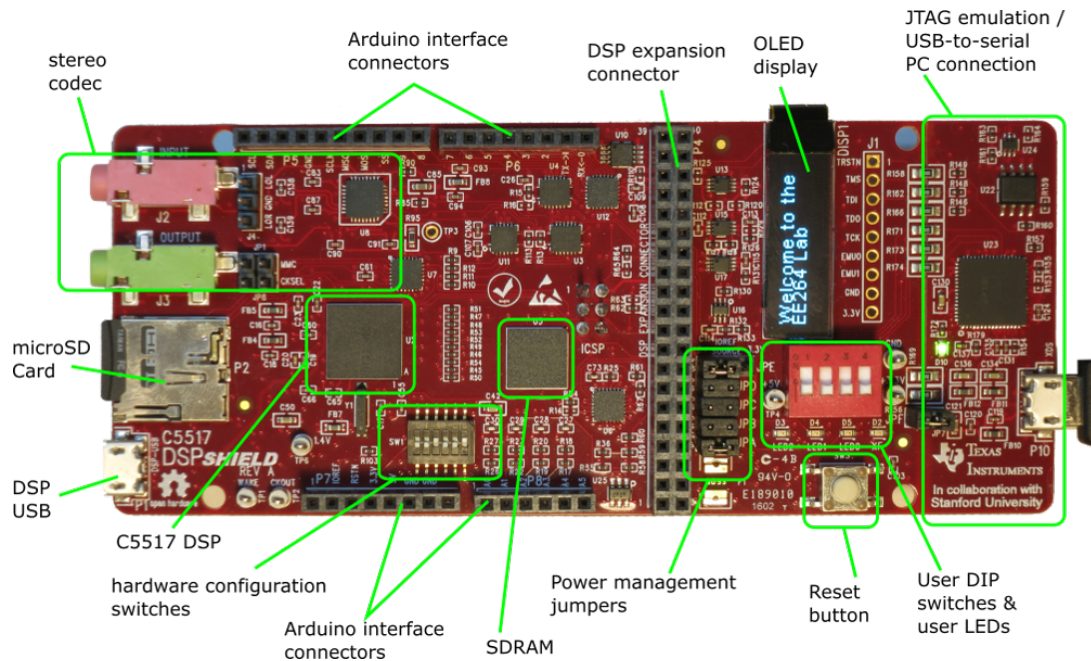


Figure 2.10: Overview of DSP Shield final revision, highlighting the location of some of the major components.

Some of these peripherals were chosen to enable independent and flexible operation, including onboard storage, multiplexers, and USB-to-serial converter. Together, they allow the DSP shield to be used with or without another microcontroller present. This allows the Arduino to be used as a means of introducing the platform in a user-friendly way while not limiting the user to the low performance of the Arduino Uno microcontroller.

To capture and perform non-trivial processing of signals used in DSP, analog electronics and biomedical applications, the DSP Shield needed to have reasonably high-performances and high flexibility. This requires a combination of well-chosen interfaces, peripherals and a flexible design paradigm.

2.7 The Components of the DSP Shield System

The DSP Shield is comprised of hundreds of components (bill of materials and schematics attached as Appendix C.). These can be broken down into some key subsystems. The core of the DSP Shield is the CPU, which is a TMS320C5517 (C5517) sixteen-bit fixed-point Digital Signal Processor (Texas Instruments, Dallas TX). Other noteworthy peripherals include an AIC3204 audio CODEC (Texas Instruments, Dallas TX) for signal acquisition and generation. Also on the board are a 96×16 pixel monochrome OLED display, GPIO buffers for connecting to the Arduino interface, a high-speed USB port tied to the DSP core, a serial to USB bridge tied to a second USB port, a microSD card reader, a real-time clock crystal, and a switching power supply to provide the various voltages necessary to power the shield. Table 2.1 lists these major components, and they are discussed in more detail in the next few sections.

2.8 DSP Shield CPU

The C5517 is a fixed-point CPU with a sixteen-bit word size. On the DSP Shield, it operates at its maximum frequency of 200 MHz and can perform up to 450 million multiply and accumulate (MAC) operations per second. It also contains 320 kB of on-chip memory.

Using a dedicated DSP allows for a higher performance level than is achievable with most conventional microcontrollers for common signal processing applications such as high-complexity filters. These can require hundreds of MAC operations to be performed on every sample of data, at tens of kilo-samples per second. Using a DSP optimized for these computations greatly relaxes the demand for algorithmic optimization on the part of the end user, key for ASPENs accessibility. While it does not allow completely careless algorithm implementation, it alleviates the need to optimize performance on a per-cycle basis. Practical applications have included

Component	Description
TMC320C5517 DSP	200 MHz sixteen-bit fixed-point DSP.
AIC3204	192 ks/s sixteen-bit sigma-delta stereo audio CODEC.
MicroSD Card	8 GB (default) flash memory for storage.
Arduino Headers	I2C GPIO expanders for 5 V Arduino Uno compatible interface.
OLED Display	96×16 pixel monochrome LED display.
User Dip Switches	4 user input dip switches.
User LEDs	4 indication LEDs.
SDRAM	8 MB of off chip SDRAM.
FT2232H	FTDI USB-to-serial bridge.
32 kHz crystal	real-time clock using 32.768 kHz crystal.
Peripheral Multiplexers	Several ICs used to allow the DSP Shield to be reconfigured to communicate in various ways.

Table 2.2: List of major DSP Shield components.

1024 point FIR filters processing two channels of data in real-time with a sample rate of 48 ks/s, without the need for code optimization.

The decision to use a fixed-point CPU as opposed to a similarly performing floating-point device was driven by development team familiarity, but has provided an opportunity to teach fixed-point signal processing concepts (as discussed later). The C5517 satisfied the requirements of the DSP Shield specification, whereas floating-point DSPs considered were more expensive and consumed more power. The slight disadvantage to choosing a fixed-point device is that it makes algorithm development more difficult for new users, who have to be cognizant of limitations to fixed-point processing. This does, however, provide a teaching opportunity as will be discussed in Chapter Five.

The C5517 contains several integrated peripheral interfaces, including the popular I2C, I2S and SPI buses. Used to connect to the peripherals included on the DSP Shield, these interfaces can also be connected to an Arduino by reconfiguring the on-board multiplexers. Extending these buses to the standard location on the Arduino headers allows a wide variety of shields and peripherals to directly interface with the DSP Shield.

Additionally, the C5517 supports external SDRAM, a USB interface, a real-time clock, an FFT accelerator and other peripherals. Importantly, the C5517 can access most of these peripherals through an integrated direct memory access (DMA) controller. This means that they can be operated asynchronously without the DSP core waiting for data transfer or sampling operations. This ability to acquire data asynchronously further increases the available performance of the DSP for signal processing tasks.

The C5517 DSP has the right set of peripherals, interfaces, performance and features to enable the DSP Shield design to meet the requirements set out for the project.

2.9 Audio CODEC

At a glance, the most noticeable components on the DSP Shield are arguably the 3.5mm “headphone” jacks. These are inputs and outputs to the on-board AIC3204 audio CODEC. This device functions much like the sound card on a desktop computer, allowing for “CD quality” (44.1 ks/s, sixteen-bit) recording and playback of audio files, with higher rates - up to 32-bit sampling at up to 192 ks/s [39]. The CODEC is connected via a combination of an I2C port for control signals and a high-speed I2S port for audio data. This data is streamed to the C5517s DMA controller which allows it to be asynchronously processed as buffered blocks.

In addition to streaming raw data, the AIC3204 has a selectable 25-coefficient finite impulse response (FIR) or 10th order infinite impulse response (IIR) filter at both the inputs and outputs [40]. These filters operate independently from the C5517 core, and require no CPU operations while in use. For the ADC, the samples pass through the filters before being sent to the DSP, as depicted in Figure 2.11. Similarly, before being sent to the DAC, the samples pass through the output filters after they are received from the CPU. Although they are relatively low complexity filters, their implementation would otherwise be directly computed on the DSP, freeing cycles for other tasks.

The AIC 3204 also contains digitally-gain-controlled input and output amplifiers, as well as power-up anti-pop protection. The analog input to the CODEC is protected by a 20 Hz to 20 kHz bandpass filter, which makes it unsuitable for biomedical applications, but well suited to audio and many classic DSP education applications.

The ADC and DAC are Sigma-Delta type [39] converters. These converters provide high fidelity by using noise shaping, which results in very low noise levels in the frequency band of the device. The tradeoff is that noise increases out-of-band (as much as 50 dB), as depicted in Figure 2.12. Since physical speakers cannot produce these high-frequency components, they act as nearly ideal reconstruction filters and produce high quality, low distortion outputs.

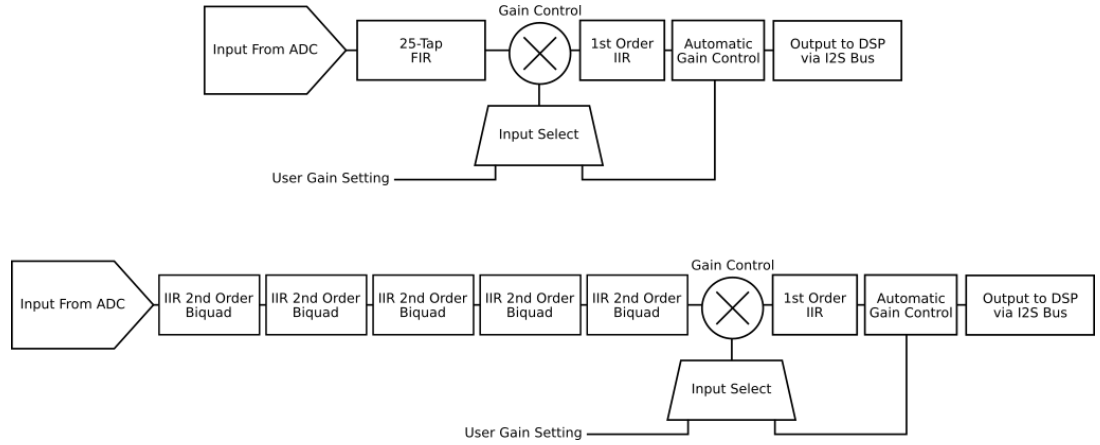


Figure 2.11: AIC3204 configurable FIR (top) and IIR (bottom) hardware filter block at the ADC, for one of two channels. A similar set of reconstruction filters exists at the output to the DAC before the Sigma-Delta node (after Texas-Instruments, 2012).

Unfortunately, this significant out-of-band noise is visible when observed on a wide bandwidth device such as an oscilloscope, which can reproduce it faithfully. This appears as broad-band noise enveloped around the output signal - as much as 100 mV on the DSP shield. This noise can be seen in Figure 2.13, measured at the headphone output of the DSP Shield using a Tektronix 2440 oscilloscope (Tektronix, Beaverton OR). With high level ($>1V_{pk-pk}$) signals, this is not a significant problem, but the AIC3204 can interface with signals much smaller than 100 mV, which would be lost in the noise floor when observed using a test instrument.

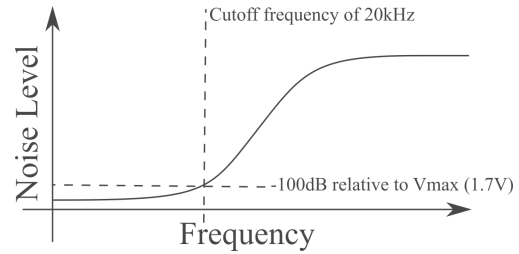


Figure 2.12: Characteristic spectrum of sigma-delta converter, as found in the AIC3204. This noise shaped signal has a low noise floor below 20 kHz but the noise level rises at higher frequencies. This is addressed by using an output reconstruction filter which attenuates high-frequency components outside the band of interest, in this case 20-20,000 Hz.

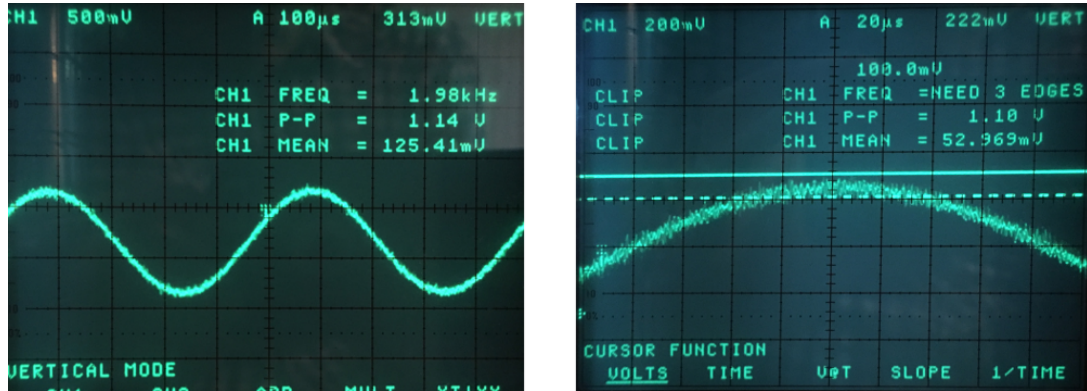


Figure 2.13: Unfiltered output with 100 mV noise envelope.

Although the detailed operation of sigma-delta type converters are a topic generally covered separately a more advanced signal processing or electronics design courses, the use of such a converter provides a teaching opportunity. With ASPEN, a signal output from the DSP Shield CODEC can be compared to the low noise output of the string-DAC on the Analog Shield.

To mitigate the out-of-band noise problem, secondary DAC outputs are fed through a simple low-pass RC filter with a cutoff frequency of 100 kHz to an “oscilloscope output,” a set of three 0.1” spaced header pins. This does not attenuate signals at the full 96 kHz bandwidth of the device, but is low enough to reduce the noise produced by the noise shaping characteristic of the CODEC. The result is an improvement in out-of-band noise from approximately 100 mV_{pk-pk} seen in Figure 2.13 to approximately 15 mV_{pk-pk}. Figure 2.14 shows a comparison of the two outputs, with a 12 dBm reduction in peak out-of-band noise, as expected. This was measured using an HP89410A (Keysight Technologies, Santa Rosa CA) spectrum analyzer. In future builds of the board, these values may be changed to a lower cutoff frequency to trade bandwidth for noise.

As the secondary output provides the benefit of standard 0.1” spaced pin headers, a user can simply hook an oscilloscope probe onto them. This simplifies connections

to outside systems - alleviating the need for an adapter or breakout board from headphone jack outputs.

There are limitations to the AIC3204 that prevent it from being the sole ADC/DAC in ASPEN. As described earlier, the 20 Hz high-pass filter at the inputs and outputs makes the AIC3204 unsuitable for some biomedical applications. Additionally, it uses a buffered block processing approach, which is more complicated than a simple direct sample request model. The Analog Shield compliments these capabilities with DC-coupled input and output that can be used with the DSP Shield to address all the needs of the project.

2.10 Other DSP Shield Peripherals

As previously mentioned, in addition to the audio CODEC, the DSP Shield includes a number of on-board peripherals which facilitate its flexibility and usefulness. The key peripherals include a USB-to-serial bridge, a number of multiplexers, two General Purpose Input / Output (GPIO) expanders, an OLED display, MicroSD card socket, SDRAM, and a high-speed USB Port.

The DSP Shield contains a dual-port serial interface. Using multiplexers, the Universal Asynchronous Receiver Transmitter (UART), a serial communication port, can be selectively connected in several ways, shown in Figure 2.15. It can be connected to the pins on the Arduino header designated for serial communication, or to an on-board FTDI FT2232H USB-to-serial converter (Future Technology Devices International Limited, Glasgow, UK). When connected to the FTDI device, the DSP shield can directly communicate with a host PC over USB without the need for additional hardware. When stacked with an Arduino, the DSP shield shares a USB connection to the PC. Additionally, the multiplexers enable the transmit and receive lines to be cross-connected, allowing the DSP shield to communicate directly with the stacked Arduino via serial (in addition to the SPI and I2C paths discussed earlier).

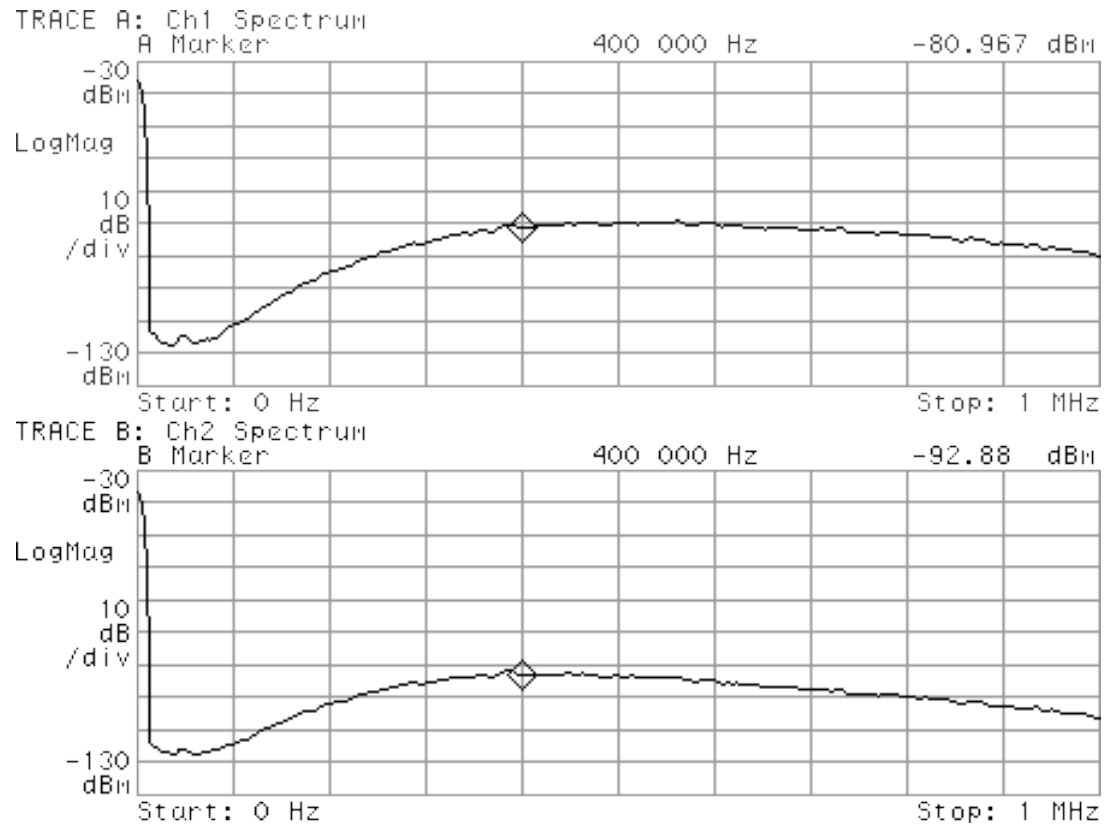


Figure 2.14: Comparison of headphone (top) and oscilloscope (bottom) outputs, showing the effect of the simple 1st order low-pass reconstruction filter on the output spectrum of the DSP Shield.

Several other multiplexers were included on the DSP Shield to enable multi-mode operation of the system. Two of the multiplexers enable the selective connection of the DSP Shield I2C and SPI buses. These buses can be connected solely to the peripherals on the DSP Shield or extended to the external Arduino shield headers on pins ten through seventeen.

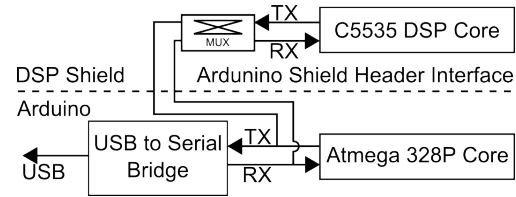


Figure 2.15: Serial mux configuration.

Additional multiplexers enable the reset lines of the DSP Shield and Arduino to be controlled by toggling the 'RTS' line on the USB serial port. This makes it possible to have a shared reset signal for both boards as well as enabling the reset line of the Arduino to be controlled independently by the DSP Shield as shown in Figure 2.16.

After a reset, the reset multiplexers are used by the bootloader software to enable the DSP Shield to be programmed independently of an attached Arduino using the same serial line. In normal operation, they enable the DSP Shield to reset in synchrony with the Arduino. Together with the serial multiplexers described above, they enable the DSP Shield to be programmed by the user while stacked with an Arduino UNO or used as an independent system. This functionality is described in more detail in the software section in Chapter Three.

Another key subsystem is the I2C connected GPIO expanders. These two TCA6416A (Texas Instruments, Dallas,

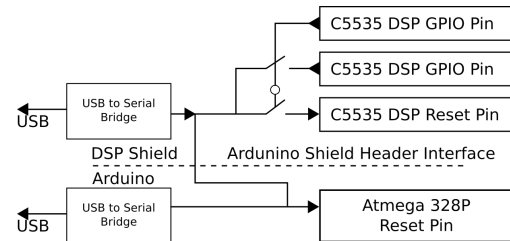


Figure 2.16: DSP Shield to Arduino UNO reset control multiplexing scheme. These multiplexers enable the DSP Shield to use a GPIO pin to keep the Arduino in reset during an upload of user software to the DSP Shield, preventing bus contention. Once complete, they reconfigure the reset line so the Arduino and DSP Shield can be reset by the same signal.

TX) devices are connected to the software bus and provide a total of thirty-two general purpose I/O (GPIO) lines [41]. Ten lines are connected directly to the Arduino headers and provide connections for the DSP shield interface with Arduino Shields, including the Analog Shield. The remaining ports are used to control the previously-described multiplexers, interface with the onboard user switches and LEDs, or connect to the DSP expansion header.

The DSP Shield includes an OLED display, an OSD9616 (OSD Displays, Orlando, FL). Shown in Figure 2.17, this 96×16 pixel monochrome display is also connected to the I2C bus on the shield. It is normally connected only to the DSP Shield, but can also be connected to a stacked Arduino when the I2C buses are joined by the multiplexer subsystem. With a relatively low resolution and a full screen update rate of about 3 frames per second, it is slow but sufficient to display user data and can even be used to display the spectrum of a signal being processed in real-time.

Adjacent to the OLED are four DIP-switches use as inputs in user code. To the left of the user switches are three LED indicators connected to the GPIO expanders. Due to the overhead imposed by the I2C serial datapath of the expanders, these LEDs can take $\frac{1}{10}^{\text{th}}$ of a millisecond to switch, which can be a limitation in time sensitive interrupt routines.



Figure 2.17: DSP Shield OLED display, user DIP switches, and user LEDs.

There is a fourth LED labeled “XF.” This LED is directly connected to the DSP shield C5517 core and can be accessed at high-speed with a single register write, making it a useful debugging tool. Additionally, because this LED does not require configuration of the GPIO expanders, it is used during the boot load sequence to provide a “heartbeat” signal, blinking as data is uploaded to indicate to users that data is being transferred.

As discussed in Chapter One, a design goal for the DSP Shield was the ability to store long signals and large data sets, as well as to have non-volatile program storage for embedded applications. Many DSP evaluation systems contain little or no non-volatile storage, requiring that the user re-upload code at each power-up.

Since the C5517 is similar to most DSPs in that it does not have on chip non-volatile storage, an 8 GB microSD card, such as those found in commodity smart phones and cameras, included with the DSP Shield. These cards retail for less than seven dollars today [42] and provide many times the capacity of common non-volatile storage ICs. This is the approach used in the successful Raspberry Pi platform to store the full Linux operating system. Additionally, the choice of an SD card as the primary storage system for the DSP Shield allows the user to easily insert the card into a PC to correct a corrupted upload, offload data generated by a program, or add large amounts of data for a DSP Shield user program to manipulate. The DSP Shield, “Shield Mode Application” discussed in a later chapter takes advantage of this capability. Addressing the need for moderately fast, high capacity non-volatile storage that is treated as an integral part of the platform is one of the key improvements that the DSP Shield provides to the teaching environment.

In addition to non-volatile flash storage, the DSP Shield also contains an 8 MB SDRAM component. This dramatically increases the available working memory of the system over the 320 kB of on-chip RAM, allowing for larger buffering and processing blocks. RAM supplements the capability of the SD card. Even fast SD cards are not quick enough to be used in some real-time signal processing applications. This is especially true when rapid and repeated manipulation of large amounts of data is necessary.

The final peripheral discussed here is a second USB 2.0 port connected directly to the CPU core. Separate from the USB-to-serial bridge, this high-speed port can enable the DSP Shield to be used as a high-speed device. It has sufficient bandwidth to enable the shield to enable applications such as a USB sound card for a host PC. The port has been shown to work with the hardware as a proof of concept, but full software support has not yet been fully developed for the DSP Shield.

The suite of peripherals selected for the DSP Shield enable it to fulfill a multitude of tasks. It can be used both for digital signal processing as both a teaching and development device, with capabilities not found together elsewhere. Simultaneously, the design allows it to work in concert with a host microcontroller as a peripheral, without compromise - this can be better seen in the “Shield App” discussed in Chapter Three.

2.11 DSP Shield Revisions

In the course of developing the DSP Shield, three hardware revisions were created, shown in Figure 2.18. The first revision of the DSP Shield (Figure 2.18, left) was developed during the summer of 2012. It was based around a C5517 core, and had most of the peripherals that the current revision contains, but lacked some of the design refinements that will be discussed below. Most critically, at that time, the C5517 DSP was not yet in mass production, and was provided as engineering samples by Texas Instruments. At the time, this seemed to be a small risk as the components were slated to go into full-scale production later that year. Unfortunately, in the late fall of 2012, Texas Instruments customer needs changed and the company elected to not put the chip into full-scale production immediately. This proved a significant hurdle to moving forward. As some of the software and the overall hardware architecture had been developed, it was decided to move forward with a second revision of the DSP Shield, using the already available TMS320C5535 part (C5535).

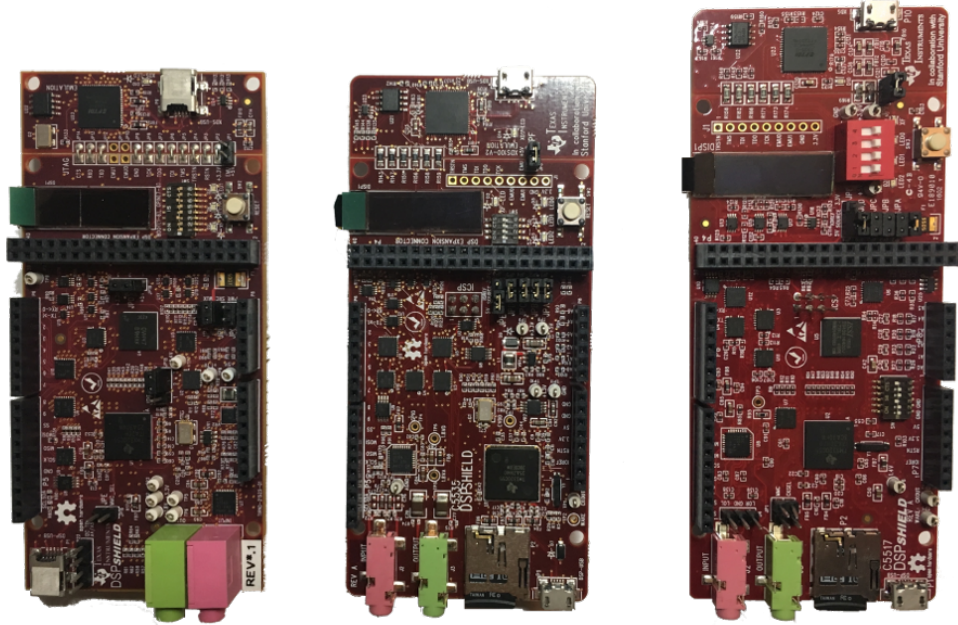


Figure 2.18: The three versions of the DSP Shield, from left to right C5517 v1, C5535, C5517 v2.

The C5535 revision of the DSP shield (Figure 2.18, center) contains several improvements to the original design. The board layout was modified to improve user friendliness, placing the SD Card on the top of the board. The spacing between input and output 3.5mm jacks was increased, and the corners of the board were rounded.

Additionally, the power supply was redesigned to remove a failure condition wherein a user could brush a finger across it, creating an electrical path (10 k Ω or more) causing it to permanently enter a latching condition. After this event, the supply would heat up, and draw more than 300 mA of current, even after being power cycled, although the DSP Shield would otherwise continue to function. This represented an increase in power draw on the order of ten times.

Unfortunately, the C5535 variant had several drawbacks when compared to the C5517 core. Most notable was the lack of a slave mode for the SPI bus, meaning that the DSP Shield could only act as a client to an Arduino via a complicated software workaround. Another key feature lost in the transition was the lack of an SDRAM port, meaning that the DSP Shield was limited to 320 kB of working memory. Finally, the C5535 had a maximum clock frequency of 100 MHz as opposed to the 200 MHz frequency of the C5517.

By 2015, Texas Instruments had changed plans again, and released the C5517 part to production. With superior hardware available, the task of revising the design for the C5517 was deemed reasonable. A third board was designed, using the C5517 part (Figure 2.18, right), completed in February of 2016. This final revision of the DSP Shield took advantage of the experience gained in using the previous two revisions of the board, adding a few additional refinements, such as larger user-friendly dip switches and the oscilloscope output for the audio CODEC.

While development of three revisions of the DSP Shield was a suboptimal path, it provided an opportunity to produce a much-refined system that better serves the needs of the end users.

2.12 Conclusion

The Analog Shield and the DSP Shield together represent the core hardware developed to drastically improve the teaching of signal processing at all skill levels and enable the teaching of electronics laboratory courses and biomedical instrumentation outside of traditional lab environments. Through careful design and several design iterations, these flexible devices can be used in a wide variety of applications and are novel solutions to existing problems.

Chapter 3

Software

3.1 Design Approach

As discussed in the first two chapters, the key insight that drove the development of ASPEN was that the Arduino hardware and software interfaces could be extended to mixed-signal, DSP, and biomedical electronics. A well-designed system with the right level of abstraction can provide complex functionality while remaining intuitive and accessible to new users.

As a whole, careful software design is absolutely critical to the success of any system with a software component. Identifying the needs, expectations, and capabilities of the end user allows a software system to be developed that can drastically improve the utility of the overall system. As computational power has improved and costs decreased, developers have allotted increasing amounts of computational resources to human factors of software design. A good system design takes into account users time spent configuring and operating the final system and makes a measured trade between raw performance and development complexity. As discussed in Chapter One, the shift towards more user-friendly programming environments has only recently reached the field of microcontrollers and has not yet penetrated the digital signal processing space.

As with the hardware, the software component of the ASPEN system was designed with user experience in mind. Various groups of users, from the high school to the graduate level were considered. As a result, the design of the software component of the system balances project development time, complexity and conceptual utility. As a result the Arduino inspired software suite presented here is less highly abstracted than some earlier alternatives which will be discussed. It is designed to be less daunting to new users by leveraging techniques like the choice of sensible “smart” defaults and the introduction of components in a progressive manner. By designing the system to function as independent pieces, as well as a unified whole, the tools allow users to discover various parts of the system as they gain experience, enabling the Analog Shield and the DSP Shield to be used either separately or together, as will be described in more detail below. There are many software components to ASPEN, including a development environment, drivers, supporting applications and tutorial examples.

3.2 System Architecture Overview

As discussed, the ASPEN platform is designed to be used in several ways. This is in order to serve both the needs of various user groups and of individual users as they gain experience.

The three boards can work in various physical combinations. All three boards are inter-operable and the software supporting their use is designed to support this. The driver library for the Analog Shield, described below, supports operation with both the Arduino and DSP Shield. The DSP Shield software systems enable it to function in a variety of ways. In one mode, it can be used with simplified capabilities as a client device controlled by an Arduino. In another mode, it can serve as a co-processor, running custom DSP software while communicating with an Arduino. Finally, it can be used wholly independently, with no Arduino present. In the latter two cases, the Analog Shield can be used as a signal acquisition and support system, controlled by

either the Arduino or DSP Shield. Additionally, support for the Analog Shield is planned for the first mode. The various configurations and supporting software are described in much greater detail later in this chapter, starting with the DSP Shield Modes section.

3.3 Analog Shield Software

The Analog Shield software can be divided into two parts, the driver library and the examples. The driver library contains the code to interface the Analog Shield ADC and DAC to a microcontroller or DSP. It has been ported to provide the same interface and compatibility across several microcontroller platforms. Initially developed to support the 8-bit Arduino Uno, it has been extended to the DSP Shield, the 32-bit PIC microcontroller based ChipKit (Digilent, Pullman, WA) platform and the ARM based Arduino DUE platform (Arduino AG, Mainz, DE) among others. The library also includes a suite of tutorial and explanatory examples which augment the instruction manual and serve as teaching materials. Because of the open-source nature of the ASPEN platform, the code (published on GitHub.com at github.com/wespo/AnalogShield) can be modified by the community to add support for the Analog Shield to many additional microcontrollers.

3.4 Analog Shield Driver Architecture

The Analog Shield library is designed with a similar interface style to the Arduino HAL to leverage user familiarity. For example, a user wishing to output an analog value to one of the GPIO pins on a standard Arduino Uno calls the “`analogWrite()`” function as shown on the second line in Figure 3.1. Similarly, to set an output value with the Analog Shield DAC, the user calls “`analog.write()`” as shown on the third line. The ADC input routine is analogous, as shown on the fifth and six lines. Because of the library design, the only setup required for this functionality is the

```
//native Arduino and Analog Shield example DAC output calls  
  
analogWrite(pin, value); //native PWM Write  
  
analog.write(channel, value); //Analog Shield DAC write  
  
//native Arduino and Analog Shield example ADC input calls  
  
value = analogRead(pin); //Native ADC Read  
  
value = analog.read(channel); //Analog Shield Read
```

Figure 3.1: Native Arduino analog interfaces and Analog Shield equivalents for reading and writing samples to the analog domain. The top two lines compare a write command to the native PWM and the Analog Shield DAC. The final two show read commands from the native ADC and the Analog Shield.

following include statement: `#include "analogShield.h"` at the top of the program, and `"analog.begin()"` in the setup function.

Additionally, the Analog Shield ADC hardware produces a signed integer output while the DAC accepts an unsigned input. To address this, the functions in the library convert ADC output values to unsigned integers. This means that when a value is produced by measuring an input voltage with the ADC, writing that result directly to the DAC with an `"analog.write()"` call will produce the same output voltage.

In addition to basic read and write functionality, support for additional hardware features has been included in the library as shown in Figure 3.2. One feature is the simultaneous update capability of the DAC. By calling the `"analog.write()"` function with multiple values, the user can load up to four channels of the DAC at the same time. Additionally, the library supports the ADC's differential input capability, allowing for pairs of converters (0 and 1 or 2 and 3) to be measured as a differential pair. The ADC can also be called via `"analog.signedRead()"` to return a signed value, reflecting the raw output of the converter, although this value cannot be directly output to the DAC.

```

unsigned int read(int channel, bool mode = false);
int signedRead(int channel, bool mode = false);
void write(int channel, unsigned int value); //any one channel
void write(unsigned int value0, unsigned int value1, bool simul); //channels 0 and 1
void write(unsigned int value0, unsigned int value1, unsigned int value2, bool simul); //channels 0-3
void write(unsigned int value0, unsigned int value1, unsigned int value2, unsigned int value3, bool simul); //all channels

```

Figure 3.2: Additional software interfaces in the Analog Shield library

While the interfaces themselves are not remarkable in their design, this simple approach was key to the success of the project. Emphasis was placed on ensuring the system was accessible by emulating the existing (and familiar) Arduino user interface paradigm and extending it to support additional hardware.

3.5 Analog Shield Examples

Included with the Analog Shield library are several examples. The examples were developed as teaching tools, but the simplest ones were ultimately included with the library as demonstrations of the boards functionality, providing a form of self-documentation. Once the user has installed the Analog Shield library, these examples appear in a drop-down menu along with the standard examples included with the Arduino itself, as shown in Figure 3.3. This is standard functionality included with most Arduino shields, and users who have been introduced to the Arduino platform are generally familiar with this approach.

With only a few lines of code, these examples demonstrate each of the interfaces described above, providing simple pass-through functionality as well as a demonstration of the effects of quantization and sampling rate on an acquired signal. While the libraries included with many Arduino shields attempt to use their examples as the sole documentation, a manual was also created for the Analog Shield. This manual covers basics, introduces novice users to the hardware, details the setup process and shows users how to upload example software for the Analog Shield. The Analog Shield manual is attached as Appendix Appendix D.

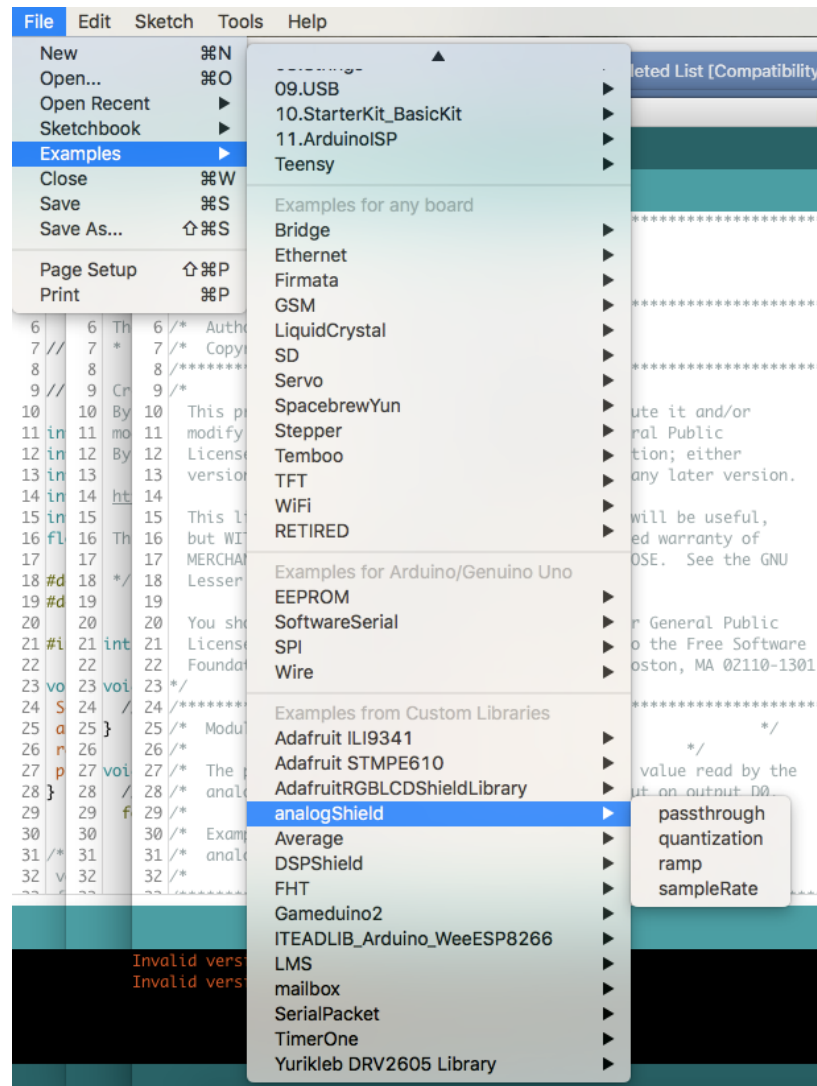


Figure 3.3: Drop down menu in Arduino IDE showing Analog Shield library examples.

In addition to these simple examples, a series of more complicated tutorials have been developed as teaching tools. Inspired by the popular Instructable format but with a more academic tone. These are distributed separately from the library itself but include code and appendices. They comprise analog circuits designed to be constructed using a breadboard, detailed conceptual tutorials, and additional software to support these circuits, on the Arduino and in some cases, on a supporting PC. The examples are outlined in more detail in Chapter Four, as some of the basic examples as well as the longer tutorials are intended to serve as teaching examples.

The Analog Shield library is as important a part of improving the student user experience and enabling ASPENs analog component as the hardware itself is.

3.6 DSP Shield Software Approach

The effort to use carefully designed software interfaces to make complex systems approachable played a fundamental role of the design choices made in the development of hardware and software for the DSP Shield. As discussed in Chapter One, a major goal in the design of the DSP Shield was to extend the Arduino paradigm to digital signal processing.

To do this effectively, thoughtful choices in the design of abstractions were required. Things like initial configurations were to be made automatic but kept adjustable by using sensible default parameters. At the same time, core intellectual concepts were not to be abstracted away.

Another aspect influencing the software design was that users are not expected to have much experience developing software. This means that syntax abstractions must be made to meet user expectations and match the interfaces of the familiar Arduino environment. A user should not have to perform arcane tasks such as typecasting an “int” to a “uint_16t” to access a user-facing function. This may seem like common sense, but non-intuitive names and parameters are often found in examples and sample

code for DSP development systems. This is in contrast to the Arduino environment. The Arduino HAL is a simple abstraction layer implemented in the C++ language. It is often mistaken by students for a language in and of itself because it has removed the need for use of many tedious C++ conventions. This is evidenced by the fact that a Google search for “is Arduino a language” yields 1.23 million results. One of the top results is the Arduino frequently asked questions page, which addresses this question [43]. The success of the Arduino environment has provided a path which shows that developing code for the DSP Shield platform can be an exercise that a novice can approach and, with only a few clicks, have code that works.

To address the needs of various groups, the shield is designed to work in several modes. These modes will be described in more detail in the next section but the following is a basic progression. Novice users will use the DSP Shield as a signal processing shield, using a library on the Arduino to control the configuration of a general purpose signal processing chain running on the DSP Shield. Upon needing more advanced capabilities, users can “graduate” to a friendly Arduino-like programming environment that enables direct programming of the DSP Shield. Programs written for the DSP Shield in this environment can be built to work in conjunction with an Arduino-like microcontroller, but programs can also be developed with no Arduino present.

The software to support the DSP Shield represents a much larger development effort than that required for the Analog Shield, and arguably represents the largest time investment of the entire ASPEN project. The complete software system for the DSP Shield includes a large number of components, including an integrated development environment (IDE), a PC bootloader application, dozens of code examples, a library for companion microcontrollers, communication software for desktop PCs, a HAL, a signal processing library, and many additional supporting subsystems. As a whole, the approach facilitates user exploration during learning and avoids abstractions that hide the real sample-based behavior of the underlying system.

3.7 DSP Shield Modes

The software architecture chosen for the DSP Shield was a consequence of the desired user paradigm. The earliest proposal for the DSP Shield envisioned a device which functioned purely as a co-processor for an Arduino Uno, providing novice-friendly signal processing functionality and allowing the user to specify a set of standardized signal processing functions to apply to signals. As the design of the system evolved, it became clear that a much richer tool-set could be developed using the same hardware if it was sufficiently flexible. Eventually, the system design crystallized into a three mode paradigm. Figure 3.4 depicts these three modes: shield mode, co-processor mode, and standalone mode.

The first mode, “shield mode” is much like the very earliest idea for a DSP Shield that sparked the start of the ASPEN project. In this mode the DSP Shield is stacked with an Arduino Uno or other microcontroller and functions as a conventional Arduino Shield. The “Shield App” is the open-source application that enables this functionality. Developed using the Energia-DSP environment (described later) and compiled before distribution to run on the DSP Shield.

This pre-compiled shield app implements a signal processing chain controlled by a master microcontroller (e.g. the Arduino Uno) via the SPI bus. Individual blocks in the signal chain on the DSP Shield can be enabled, disabled, and configured as requested by the master microcontroller, using a library for the Arduino IDE. In

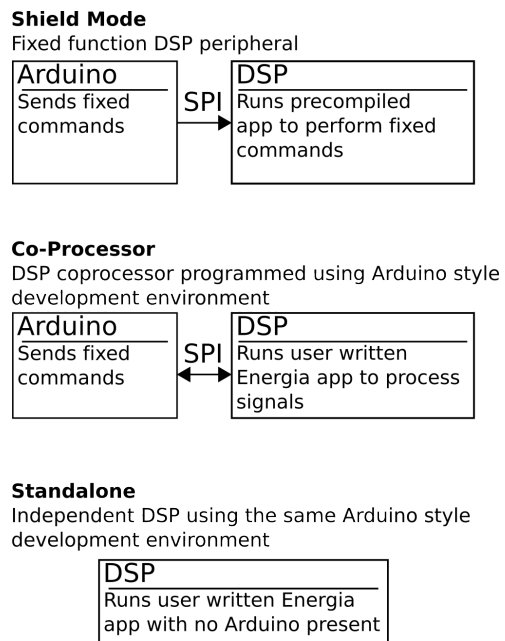


Figure 3.4: DSP Shield operation modes.

this mode, the DSP shield functions much like the block diagram tools discussed in Chapter One and has the associated limitations. The advantage is that it requires virtually no effort from the end user to begin manipulating signals, enabling its use as an introduction for novice users before they begin to develop algorithms for the DSP Shield itself.

The second mode in which the DSP Shield can function is as a general-purpose co-processor. In this mode, the DSP Shield is again stacked with an Arduino. It is in turn programmed with an independently uploaded user created program using the DSP Shield IDE, Energia-DSP. These programs can use the shields SPI, Serial and I2C communication interfaces to share data or control signals with the host microcontroller, which is still programmed via the Arduino IDE.

The third mode is standalone mode. In this mode, the DSP Shield acts as an independent development system running user written programs without the need for an Arduino. The user uploads Energia-DSP programs directly to the DSP Shield as they would with any Arduino style microcontroller.

The relationship between the Analog Shield and the DSP Shield requires explanation. As there is a driver library for the Analog Shield for both the DSP Shield and the Arduino Uno, the Analog Shield can be used to some extent in all three modes. In shield mode, the utility of the Analog Shield is currently somewhat limited, as the Shield App has no support for the Analog Shield, relying instead only on the AIC3206 codec built onto the DSP Shield. This could be changed in the future, as it is merely a matter of integrating a block providing support for the Analog Shield driver library to the existing Shield App. In co-processor mode, the Analog Shield can be used by the user application on either the DSP Shield or the Arduino Uno with the Analog Shield library. In standalone mode, the DSP Shield can be used with the Analog Shield library, just as with a standalone Arduino Uno. Table 3.1 shows the relationship between the devices in more detail.

Hardware Present	DSP Shield Mode	Device Controlling Analog Shield	Detail
Arduino Uno Analog Shield	N/A	Arduino Uno	User program controls Analog Shield.
Arduino Uno Analog Shield DSP Shield	Shield Mode	Arduino Uno	User program controls Analog Shield.
Arduino Uno Analog Shield DSP Shield	Shield Mode	DSP Shield	No support, future feature.
Arduino Uno Analog Shield DSP Shield	Co-processor mode	DSP Shield or Analog Shield	Either device can control Analog Shield.
Analog Shield DSP Shield	Standalone Mode	DSP Shield	User program controls Analog Shield.

Table 3.1: Detail of inter-operation of Arduino Uno, Analog Shield and DSP Shield

3.8 DSP Shield Mode Application

As discussed in Chapter One, providing an introduction to real world signal processing that could be approached by novices (such as students at the high school or first year university level) was a major goal of the project. The “shield mode” addresses this need.

Because many users of the ASPEN stack are expected to be novices interested in access to DSP functionality without the skill set to immediately begin developing DSP applications or code, pre-compiled applications can be installed on the DSP Shield and interfaced to the Arduino (or other compatible microcontroller platforms). These applications can provide DSP tools with only simple configuration commands on the part of the user.

```
#include <SPI.h>
#include <mailbox.h>
#include <DSPShield.h>

void setup() {
  //initialize
  DSPShield.init();

  //start shield audio loopback
  DSPShield.startLoopback();

  //load a 201 coeff 3kHz LPF
  DSPShield.setFIRFilter(CHAN_BOTH, LOW_PASS, 201, 3000);
}

void loop() {}
```

Figure 3.5: Example shield mode application library call.

To address the needs of these novice users, a high-level general purpose signal processing application has been developed. This application - named Shield App is at the heart of the “shield mode” described above. It effectively runs as a layer on top of the “co processor mode” and includes pre-developed signal processing functionality. Users can control the Shield App using an associated library developed for the Arduino Uno that exposes several major signal processing algorithms running on the DSP as single-line function calls. These calls are sent from a library running on a stacked Arduino as seen in Figure 3.5.

Each function call is sent by the DSP Shield library via a communication library and the SPI (Serial Peripheral Interface) connection, as seen in Figure 3.6.

To facilitate this, a bidirectional “mailbox” communication library has been developed that enables bidirectional control of the SPI bus between the DSP Shield and Arduino. The library ensures that data is sent without error and provides positive confirmation of message receipt. This library is described in greater detail below.

The Shield App supports several signal processing operations, including DDS function generation, Chirp, FIR filtering, IIR filtering, spectrum analysis, control of the OLED display, and control of the hardware audio CODEC. These are implemented as blocks in a predefined signal chain, seen in Figure 3.7. Each block is switchable and configurable via calls to the DSP Shield library on the master Arduino.

The Shield App takes advantage of the unique large storage space provided by the presence of an SD card on the DSP Shield. One example of this is the inclusion of

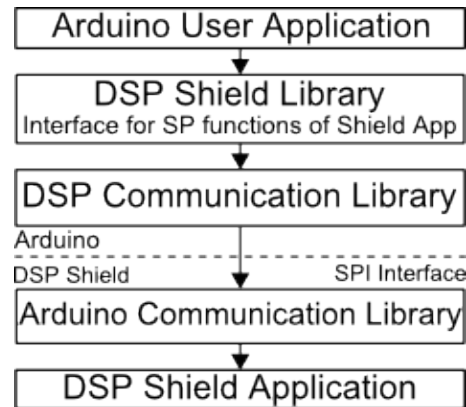


Figure 3.6: Application layers showing functional path to send data from an Arduino to the Shield App. The message passes from a function call to a predefined point in an application on the DSP Shield.

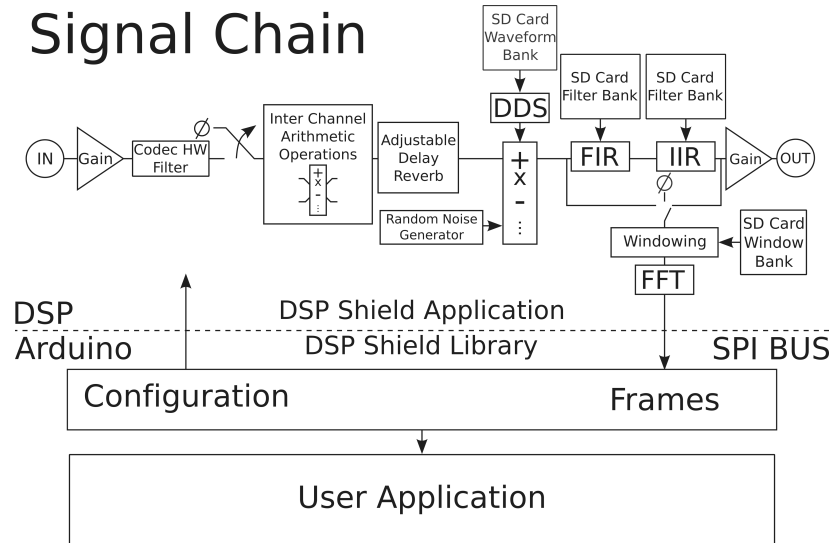


Figure 3.7: Shield Application signal chain block diagram.

tens of thousands of pre-computed digital filters. These filters include high- and low pass FIR filters in 1 Hz steps from 9 Hz to 20 kHz (slightly wider than the -3 dB bandwidth of the CODEC) and ranging from 4 to 511 coefficients. This enables a user to implement a given filter from the Arduino simply by specifying the cutoff and order desired in a single function call. Additionally, the DSP Shield library can accept manually uploaded filter coefficients transmitted from the Arduino, allowing completely custom FIR filters designed by a student to be used as needed.

The same functionality has been implemented for IIR filtering, using an IIR algorithm described in more detail later. Included on the SD card with the Shield App are Butterworth, Bessel, Chebyshev, and Elliptical high- and low pass filters of 2nd to 14th order in 1 Hz increments from 9 Hz to 20 kHz, although the minimum (low-pass) and maximum (high-pass) cutoffs are dependent on the order and type of filter. This is due to numerical effects that appear in higher order filters on fixed-point systems with cutoffs at small fractions of the sample frequency (1% or less). When the requested IIR filter is not available, the Shield App loads the nearest value. A future revision of the software will further clarify this limitation by using a pre-listed enumerated data

type in the Arduino library for filter cutoffs. This will provide a compiler warning when non-supported filter cutoffs are called for by the user. Additionally, a future revision may implement an optional full 32-bit IIR filter signal chain to improve noise performance at the cost of CPU cycles.

The Shield App can also be configured to periodically return the spectrum of a signal passed through the DSP Shield. This spectrum can be windowed on the DSP Shield using an arbitrary window vector or one of several predefined vectors stored within the shield app on the SD card.

The Shield App also includes a DDS function generator subsystem, which can be used to generate an arbitrary waveform from a single sampled period. A selection of such waveforms are stored on the SD card as 8,192 sample long raw PCM data. This gives the DDS routine a theoretical 80 dB SNR. Frequency can be set from DC to 22 kHz. This is the Nyquist rate of the system as configured in the Shield App, although frequencies below 20 Hz will be attenuated by the high pass filter of the DSP Shield AIC3206 CODEC. New waveforms can also be added to the system by the user, by changing the files stored on the SD card. The DDS block can be used in conjunction with full functionality filtering and even as an input to one or both channels of the predefined signal chain. Included in this functionality is a sine sweep register allowing for linear sweep chirps of arbitrary rates. This enables swept-sine-style network analysis applications to be developed. This contrasts with the previously described DDS using the Analog Shield alone, which is limited in bandwidth, cannot change waveform or frequency at run time, and cannot perform frequency sweeps. Additional functionality has been developed which allows for amplitude or frequency modulation of a generated signal, although this has not yet been included in the published version of the Shield App.

Table 3.2 summarizes the functions and specifications of the initial release of the Shield App. Continuing development of the Shield App will bring more functionality to shield mode, and the DSP Shield library is designed to be correspondingly extensible. The source code for the Shield App has been released publicly and was developed in

Function	Description	Specification
Loop back	Audio passthrough.	Connects or disconnects the audio input signal from the signal processing chain.
Input Gain	Gain of input amplifier.	Pre-amplifier gain for CODEC input from 0 dB to 47.5 dB in 0.5 dB steps.
DDS	Function generator.	Plays back a waveform at a set frequency from DC to 22 kHz. Supports sinusoid, square, triangle, sawtooth, white noise, and user defined waveforms. Support for chirp and fixed frequency output. Signal can be mixed with input from audio codec.
Noise	Generates random noise.	White noise signal created using a software random number generation routine.
FIR	Finite Impulse Response Filter bank	Load an FIR filter by either sending coefficients from Arduino Uno or by loading pre-computed values from SD card. Cutoffs from 9 Hz to 20 kHz in 1 Hz steps. Order 41-tap to 511-tap, high pass or low pass.
IIR	Infinite Impulse Response Filter Bank.	Load an IIR filter by either sending second-order section coefficients from Arduino Uno or loading pre-computed values from SD-card. Order 2-14. Butterworth, Bessel, Chebyshev, Elliptical types. Cutoffs from 9 Hz to 20 kHz in 1 Hz steps, high pass or low pass.
Window	Apply windowing function.	Windowing function is applied to a set number of samples, to pass to the FFT function. Coefficients can be passed from and Arduino or loaded from SD Card. Pre-computed filters are of Blackman, Hamming, Hann types with length 32 to 1024 points.
FFT	Return FFT of acquired samples to Arduino.	Spectrum can be complex or real valued (magnitude). Length 32 to 1024 points.
Reverb	Audio delay line.	Creates an echo effect by adding a delayed version of the signal to itself.
Remove Voice	Removes signals present in both stereo channels.	By subtracting the channel A from channel B, common components of a stereo signal are removed, creating a mono signal. As the human voice is centered in most recordings, this has the effect of voice removal.
OLED Print	Text Display.	Display of characters sent from Arduino UNO
Output Volume	Gain of output amplifier.	Percentage (0-100%) of full scale system output (line level, 1.4 V), floating-point.

Table 3.2: Summary of functions supported in initial release of Shield App, detailing functionality shown in Figure 3.7.

Energia-DSP, to encourage contribution to the project by the community. It serves as a set of training wheels to introduce signal processing to students who are not ready to program the full DSP Shield on their own. These students will benefit from quick success in manipulating signals, which can encourage them to take further steps, facilitated by hardware that they already have and are familiar with.

3.9 DSP Shield Mailbox Library

The mailbox library was developed in conjunction with the Shield App in order to facilitate communication between the DSP Shield and a host Arduino. By abstracting the process of sending data between the DSP Shield and the Arduino, the Mailbox library serves as a tool to facilitate use of the co-processor mode of the DSP Shield, as well as providing key infrastructure for the development of the Shield App. As with the Shield App itself, the mailbox library is in fact two pieces of software, a communications library on the host Arduino and one on the DSP Shield.

The mailbox library works using the SPI interface and a GPIO pin on both devices, with the DSP Shield acting as the SPI master device. When the Arduino needs to send data to the DSP Shield, a call is made to the transmit function, and the Arduino uses a GPIO pin as a flag to indicate that service is needed. The DSP Shield then reads the data from the Arduino. When

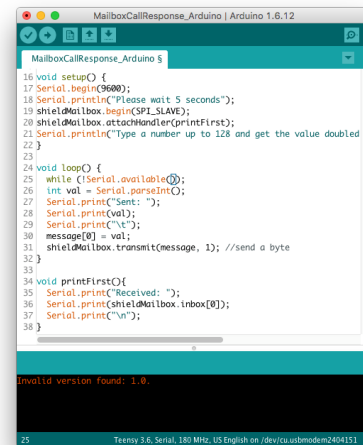


Figure 3.8: Mailbox library example, a program which sends a message to the DSP Shield and prints the response received via the serial port.

the DSP Shield needs to send data to the Arduino, it initiates a “transmit” call as well. In this case, the DSP Shield asserts the slave select GPIO pin and the Arduino executes an interrupt, calling a preset handler function to process the received message. An example function call from the Arduino can be seen on line 31 in Figure 3.8.

As with many parts of the DSP Shield, the Mailbox library represents a small improvement in the user experience for deploying DSP as a tool in student and hobbyist facing embedded environments, but it is a key part of making these systems more approachable to novice users.

3.10 DSP Shield IDE

A hurdle in the transition from teaching signal processing theory using a simulated environment such as Matlab to DSP is the usability of the programming environment itself. This problem has been approached in the past with high-level block diagram type solutions such as Simulink. These tools can be modified to directly generate C code for a DSP and upload to a development board [44]. This has the advantage of removing frustrations revolving around implementation details, but it abstracts away many important real-world design challenges, such as block processing and handling of frame edges, which are critical problems when developing real-time, continuous signal processing applications.

The most popular approach to teaching signal processing education uses PC based software tools (such as Matlab) to prototype an algorithm and then asks students to develop their algorithm a second time in the embedded environment. The DSP Shield can be used with this approach, but simplifies the process of implementing an algorithm on the DSP. Because of the ease of development on the DSP Shield, it is often possible to skip the simulation step entirely, and simply begin development of code directly in the embedded environment.

The improved development environment for the DSP shield is called Energia-DSP. It is a fork of the Energia IDE, itself forked from the Arduino IDE to add support for the MSP430 microcontroller and other Texas Instruments processors [45]. As discussed in Chapter One, that IDE is based on work done at the MIT Media Lab for the Processing programming language, which is an extension of the Java programming language [46]. There has been discussion of renaming the DSP fork of the Energia project to clarify the distinction between the core version of Energia and the DSP fork, but that is a task that has not yet been undertaken. This document will refer to Energia DSP, but the two should not be confused as their development has gone in different directions since the start of the project.

The choice of forking the an Arduino related environment instead of developing a new application or using another existing IDE, such as CCS was driven by the original design requirements for the ASPEN project. As discussed in Chapter One, the Arduino environment includes fixtures for accessible built-in examples that can be customized for each board. Additionally, the familiar Arduino paradigm and interface makes the system appealing for the large cohort of users already familiar with Arduino. Almost as important is easy distribution of the system. The simple installation requires the user to traverse only a few mouse clicks to go from having never used the environment to successfully uploading a program to the DSP Shield. This is in stark contrast to existing environments (also discussed in Chapter One), which can take hours to install, and hours more to configure to the point where an example is functional.

Because Energia is related to the Arduino environment, it offers the same usability and familiarity benefits. At the outset of the project Energia was chosen because its use presented a simpler development path than a direct fork of the Arduino IDE. Additionally, the possibility of merging the new features and support back into the Energia IDE, already targeted at Texas Instruments microcontrollers, was promising.

As with the Arduino IDE, Energia DSP includes an abstracted C++ HAL Application Programming Interface (API), which gives development of applications for the DSP Shield similar syntax to Arduino applications and the Processing language.

The Energia-DSP environment provides an extremely straightforward user interface. On launch, the user is presented with a simple text editor that includes the framework of a working program, as shown in Figure 3.9. The user selects the platform they are targeting from a drop-down menu, usually one of the variants of the DSP Shield. When the user is ready to test their code, they simply click an arrow shaped upload button in the upper left corner to upload it to the DSP Shield.



Once a user clicks the upload button, Energia-DSP gathers the libraries and support files invoked by the user script and by the board selected from the drop-down, compiles, and links the files. A copy of the generated binary is saved to the users script directory, allowing for distribution or backup. Finally, Energia-DSP uses the built-in “Pydude” serial uploader program store the user binary to the DSP Shield.

Figure 3.9: New Energia sketch, as seen by a user at program launch.

To support the various user modes of the DSP Shield, Energia-DSP provides a separate listing for each version of the DSP Shield in shield and standalone modes. Selecting the mode determines how the binary created by Energia-DSP configures the multiplexers for the Arduino interfaces on the DSP Shield at startup. This provides a straightforward means for the user to configure their shield and switch program modes. In shield or co-processor mode, Energia-DSP can be used in parallel with the Arduino IDE and upload code to the DSP Shield through the same USB connection used by the Arduino Uno with which it is stacked. In standalone mode, DSP Shield binaries are uploaded through the DSP Shields onboard serial port.

Device	Platform	Compiler Support for OS
Arduino Uno	Windows	Pre-existing
Arduino Uno	Mac	Pre-existing
Arduino Uno	Linux	Pre-existing
DSP-Shield	Windows	Pre-existing
DSP-Shield	Mac	New for ASPEN project
DSP-Shield	Linux	Pre-existing, IDE beta

Table 3.3: OS platforms supported by ASPEN project. The Linux build of Energia-DSP is considered beta due to the limited number of test users.

One challenge in developing Energia-DSP, was the requirement that the DSP Shield be compatible across the three major personal computing platforms, Windows, Linux and Mac OS. At the time C55x compilers were only available as a part of the Code Composer Studio release for Windows and Linux and there was no publicly available compiler tool-set for Mac OS. Working with Texas Instruments, the team was able to get the compilers for Windows and Linux released as a separate download. Additionally, a Mac OS version of the compilers was developed, tested, and released publicly [47]. This made it possible to release Energia-DSP as a fully cross platform development environment. Table 3.3 shows the supported platforms of the TI Compilers.

3.11 DSP Shield Co-Processor and Standalone Modes

In both standalone mode and co-processor mode, user programs are developed using Energia-IDE. From a software perspective, both modes are very similar. This is in contrast to shield mode, where the DSP Shield behaves as a fixed function signal

processor and the user sends commands from an attached Arduino.

The behavior of both of these modes could be recreated by a user with another development environment, such as Code Composer Studio, or even by simply compiling a binary from the command line. Doing so, however, would require that the user manually define the desired DSP Shield operation mode in the program, as well as import the libraries included with Energia-DSP.

The Energia-DSP IDE, however, has been specifically built to support both modes. It compiles programs to automatically configure the multiplexers (described in Chapter Two) for the selected mode modes. Figure 3.10 shows how an Energia-DSP user selects between modes and models of the DSP Shield using a drop down board selection menu.

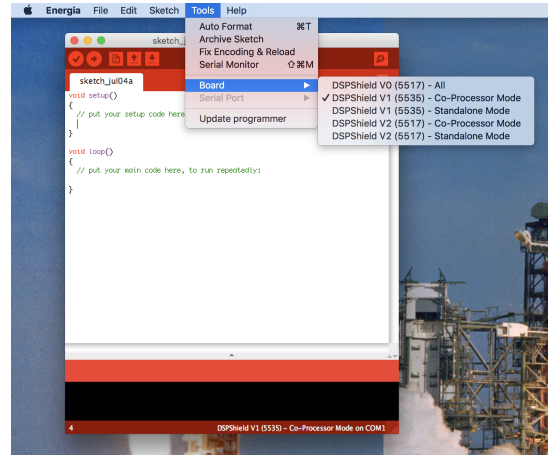
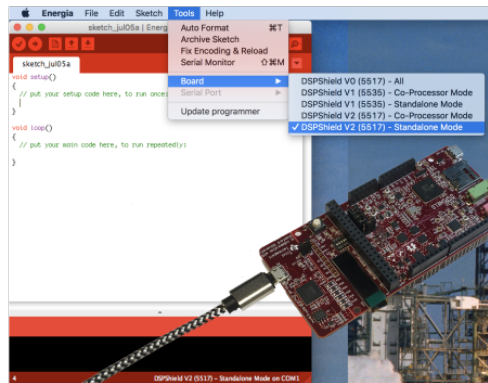


Figure 3.10: DSP Shield mode selection drop down menu in Energia, showing how the user switches from Standalone to Co-Processor mode.

Once the user selects a mode for the DSP Shield and uploads changed code, the DSP Shield will reboot in the new mode. This requires the user to switch USB cables to the correct device. That is, when the user switches from standalone mode to shield mode uploads new code, they must then move the USB connection from the DSP Shield to the Arduino. Conversely, switching from standalone mode to shield mode will require moving the USB connection from the DSP Shield to the Arduino after the new program is uploaded. Figure3.11 shows this process in more detail.

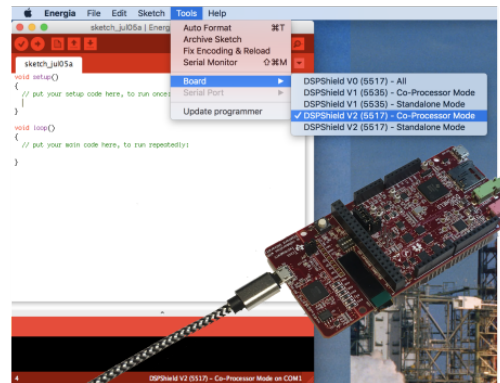
Because the IDE handles all the configuration and communication steps for switching the DSP Shield between standalone and co-processor modes, the user can generally ignore the presence and application of the multiplexers, and simply develop C++

Switching the DSP Shield from standalone mode to co-processor mode:



Step 1.

- DSP Shield is already in standalone mode.
- DSP Shield is connected via its own USB.



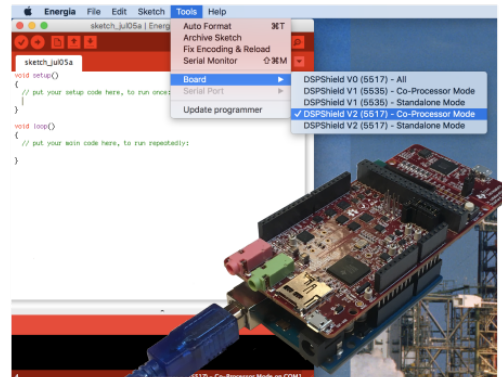
Step 2.

- Select co-processor mode as board.
- DSP Shield still connected via its own USB.



Step 3.

- Upload code with arrow button at top-left of Energia IDE.
- DSP Shield still connected via its own USB.



Step 4.

- After upload, stack with Arduino and switch to Arduino USB.
- Continue to develop code using Co-processor mode, upload as normal

Figure 3.11: Process for a user to switch between standalone and co-processor modes.

code to address higher level algorithmic tasks. This is an example of how the design choices made in developing the DSP Shield enable the ASPEN system to meet the goal of abstracting tedious setup steps without obscuring fundamental concepts.

3.12 DSP Shield Bootloader

One of the major challenges identified at the outset of the ASPEN project was that existing DSP evaluation modules and teaching platforms lack a simple way for users to upload their program code to the target device and keep it there. Devices such as the C5535 eZdsp have an in-circuit serial programming (ICSP) interface [48]. The eZdsp can be configured to boot from a file stored on an SD Card, but lacks an on board USB-to-serial converter. More importantly, none of the DSP development and teaching systems evaluated at the project outset had a means of automatically and permanently storing code uploaded by a user locally on a mass storage device (e.g. a Micro SD card). Because a major goal of the ASPEN project has been to lower the barriers of entry to using signal processing tools, single click serial upload of user programs was identified as a critical requirement for the system.

The concept of a serial bootloader is far from novel, having become one of the defining features of modern microcontroller development environments. As mentioned in Chapter One, this simple serial upload process is a compelling feature of the Arduino system and has contributed to the success of that platform.

In lieu of the conventional ICSP-based upload to DSP RAM, the DSP Shield code upload process functions on the model used by the Arduino. The device receives a serial upload of code from the user and automatically stores that user program on a microSD card. This allows for user program persistence from boot to boot, which enables use of the device as a standalone embedded system. Choosing to always store a user program on mass storage allows the program developer to rely on the availability of mass storage. This also enables the design of interesting applications using stored datasets.

Given the success of the Arduino serial bootloader implementation, the initial approach to development of the DSP Shield bootloader was to create a fork (variant) of the existing “AVRDUDE” serial uploader application [49] with support for the DSP Shield. AVRDUDE is used by the Arduino project for serial uploading of code to the Arduino Uno and several other supported boards. This fork built for the project required the user to install the open-source “libusb” library and quickly fell out of sync with the main version of AVRDUDE used for other microcontrollers. To remedy this, a new application was developed named Pydude, which implements the protocol without a need for additional library installation and without unnecessary support for additional platforms - Energia-DSP is able to use the main trunk variant of AVRDUDE to support those platforms without the need for additional libraries. Communication with the PC is facilitated on the DSP Shield by a bootloader code section that runs before user program code is executed. The bootloader checks for a new upload request on the serial port. If there is new code available, the bootloader code handles communication with the Pydude application and stores the code locally. This bootloader application is where the reset and serial multiplexers are configured to select among the various DSP Shield modes.

To achieve this type of transparent upload, the DSP Shield bootloader follows the flow sequence depicted in Figure 3.12. It first monitors the shared serial bus for a handshake message after restarting. If the first bytes received are a unique handshake for the DSP, it will hold the Arduino in reset and accept a new user upload via the serial port. Once the upload is complete it validates the code with a checksum and then stores the new program as a binary on the SD card. If the serial upload is not intended for the DSP Shield (i.e., an upload for a stacked Arduino Uno sharing the serial bus in co-processor mode), or no upload is detected, the system loads the pre-existing binary from the SD card.

The DSP Shield bootloader brings the concept of a single click serial upload process to DSP teaching tools. It both makes the device easier to use and enables the use of the system as an embedded device in addition to as a teaching tool.

3.13 DSP Shield Hardware Abstraction Layer

Many ASPEN users will spend most of their time writing code. This means that simplicity and understandability of the libraries they use are two of the most critical components in ensuring that the system is approachable and useful as a development tool. Given that the software metaphor presented to the end user is that of the Arduino environment, it is critical that the software interface be as Arduino-like as possible. As discussed above, the underlying structure of the Arduino code developed by users is simply an implementation of the C/C++ language with a series of extensions automatically included as libraries to simplify many of the routine functions that microcontroller users utilize. As discussed earlier, many users are completely unaware that they are writing C code, in-

stead thinking of “Arduino” as a programming language wholly unto itself [43]. In order to mimic that interface, the HAL used by the Arduino must be re-implemented in Energia-DSP. One of the most prominent features of this is a “setup()” and “loop()” functions for user code. These run once and continually respectively, and are simply called from a hidden “main()” function familiar to more traditional C variants. The API also implements Arduino style function calls for I/O, variable type names, and syntactic conventions.

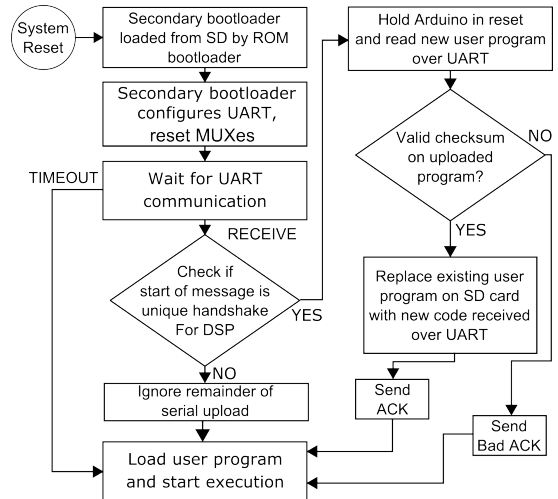


Figure 3.12: Flow diagram showing how the DSP Shield receives a new program via a serial upload command, as well as how it can discriminate from an upload targeted at an Arduino Uno sharing the same serial bus.

To enable this functionality, the existing Texas Instruments Chip Support Library (CSL) [50] was leveraged. This library is syntactically very different from the Arduino HAL, following traditional C naming conventions more closely. For example, the Arduino HAL uses “int” for 16-bit integers while the CSL would instead use “Uint16_t” [51]. In order to address this within the limited development time available, a subset of the most important standard Arduino functions, types, and conventions were identified in a design document. These interfaces to be implemented were prioritized for frequency of use, utility to the user, and appearance in basic examples. These functions were then re-implemented on the DSP Shield. Using the CSL to speed development where possible, many functions were simple wrappers for interface calls. Others, however, required additional development due to the structure of the DSP Shield. Things like GPIO read and write events depended on the I2C GPIO expander hardware and thus had to be implemented from scratch, and carefully optimized to ensure that performance approached what naive users would expect. In the case of the GPIO functionality, this required optimizing the read and write command sequence to prevent unnecessary verification read/write instructions on the I2C bus. It also required optimization of the I2C library itself, to increase the read and write bit rate and dramatically reduce timing delays. Together, these improvements increased the performance of the GPIO library from the initial version capable of few dozen read/write operations per second to the current version which is capable of over 10,000.

The result of this effort is that when a user familiar with the Arduino ecosystem begins to write code for the DSP Shield, most of the functions operate as expected without the need for additional instruction. This also means that many explanations and code examples that exist online can be used for the DSP Shield. Ultimately, a well implemented HAL is the cornerstone of a good user experience. A complete listing of the implemented libraries from the HAL would not fit in this space and is attached as Appendix Appendix E. Table 3.4 shows a selection of the most prominent ones.

Function	Purpose
General purpose I/O	digitalRead() / digitalWrite() functionality.
SPI	SPI communication protocol, for peripherals.
I2C (Wire)	I2C communication protocol, for peripherals.
Interrupt	Standardized interrupt support.
SD	Reading / writing files to microSD card.
AnalogRead	Input from DSP Shield analog pins.
Timers	General purpose timers.
Serial	Serial communication bus.

Table 3.4: Listing of some of the Arduino HAL libraries ported to the DSP Shield.

3.14 DSP Shield Signal Processing Library

In addition to re-implementing the of the Arduino environment’s HAL on the DSP Shield, the Energia-DSP environment includes a Signal Processing Library (SPL) which extends these conventions. A primary goal in developing the DSP Shield has been making the implementation of signal processing algorithms more accessible and understandable. The SPL is the component of the DSP Shield software package that addresses this need. It is designed to simplify signal processing tasks while retaining direct access to raw data as was initially envisioned.

One key component of the SPL is the way in which data is presented to the user. As discussed earlier, stream processing was deemed inappropriate as it hides samples from the user. Providing the data one sample at a time to the end user however, would result in a significant performance penalty for little benefit. As a result, a block processing approach was chosen. Using this architecture, a device fills a buffer

with a number of acquired samples before interrupting the user program to process the block of data stored in the buffer. This has the advantage of significantly reducing the frequency at which the main program is interrupted, but increases the complexity of buffer management, generally using a so-called “ping-pong” buffer scheme. In this scheme, the system fills one buffer and then flags it for processing while beginning to fill a second, switching alternatively between the two.

To handle this buffering in an intuitive way, the part of the SPL which supports the AIC3204 audio CODEC was implemented with a function named `processAudio()`. If the user initializes the CODEC by including the header file, `Audio.h` the compiler will require that the user to fill in the `processAudio()` function. This works similarly to the familiar `setup()` and `loop()` functions in a conventional Arduino program. When a buffer is full, the library will interrupt the loop function and automatically handle all management of the ping-pong buffers. It then calls the user generated `processAudio()` function performing whatever signal processing tasks the user has defined. Finally, the system will perform the required post processing buffer management. With this scheme, the user does not have to handle swapping between multiple buffers, but still is provided full access to samples. Furthermore, the audio processing workflow is similar to the already familiar main loop functionality, which is intended to make the process more understandable to novice users.

The SPL also supports several core signal processing algorithms, designed to integrate with the audio CODEC component as well as to process data sampled by other means (e.g. use with the Analog Shield). Many of these functions are abstractions or wrappers built on the pre-existing Texas Instruments DSPLIB functions which provide core signal processing functionality such as trigonometric functions or Fast Fourier Transform (FFT). These include a direct form FIR filter, which was abstracted for ease of use and tested with filters of arbitrary length ranging from 3 to 1024 coefficients. This structure lends itself perfectly to the implementation of audio effects filters due to their well-behaved overflow characteristics and linear phase. The FIR implementation also lends itself well to the implementation and teaching of adaptive filtering algorithms for teaching applications such as noise cancellation.

Another signal processing algorithm implemented for the DSP Shield API was an Infinite Impulse Response (IIR) block-filtering algorithm. As the DSPLIB IIR algorithm was found unsuitable due to a fixed coefficient in the b_0 position, an alternative implementation was developed. The IIR algorithm is implemented as a cascade of direct form I (DF-I) biquad blocks. Developed from a previous algorithm by Prof. Mujica, it was modified for use on the DSP Shield. The transfer function of each biquad in the basic implementation is given in Equation 3.1.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{2^{a_0} + a_1 z^{-1} + a_2 z^{-2}} \quad (3.1)$$

In a sixteen-bit system such as the DSP Shield, the effect of quantization creates audible noise in IIR filters with low cutoff frequencies. To mitigate this, a technique called fraction saving can be applied [52]. This technique was added to the IIR filtering routine for the DSP shield. In this implementation, the bottom sixteen-bits of the 32-bit summing node of the DF I implementation are stored before the result is quantized. These bits are then added back as an input to the next sample, approximating a 32-bit signal chain without full 32-bit operations. The resulting algorithm with fraction saving can be seen in equation 3.2. This approach saves memory and clock cycles, freeing them up for use in other operations.

$$\begin{aligned} y_0 &= x_0 b_0 + x_1 b_1 + x_2 b_2 - y_0 a_1 - y_2 a_2 \\ y_0 &= y_0 + \text{frac} \\ \text{frac} &= y_0 \& ((1 \ll a_0) - 1) \\ y_0 &= y_0 \gg a_0 \end{aligned} \quad (3.2)$$

This IIR implementation proved satisfactory for applications a wide range of filter orders and cutoffs as a part of the DSP Shield libraries was used in the Shield App, described earlier.

Also included in the IDE is support for PCM WAV playback and recording, extending the Audio and SD card components of the HAL API. This implementation provides a wide range of sample and bit rates, from 48 ks/s to below 1 ks/s. While the high sample rate seems more interesting initially, as it is useful for audio applications, lower sample rates are useful as well. A low sample rate signal, say 1 ks/s provides more than sufficient bandwidth for a biomedical signal, yet the 8 GB capacity of DSP Shield would enable a continuous recording for more than 45 days. Extremely low-frequency signals such as ECG may be impossible to acquire accurately with the CODEC, but useful information is present in audio signals such as the phonocardiogram.

Additional functionality has been included in the SPL. Among these are DDS with frequency and amplitude modulation, power spectral density and several other commonly used signal processing functions. Additionally, the original unmodified TI DSPLIB library can be accessed by advanced users if needed, providing useful functionality such as adaptive filtering and routines for trigonometric vector operations. Future development will wrap more of these functions to conform to the syntactic standards of the DSP Shield HAL, making them more accessible to all users.

3.15 Conclusion

The software architecture of ASPEN represents the core of the user experience with the system. Through a number of thoughtful design choices, a system has been realized which ensures that user frustration is reduced by creating a short path to a users first successful demonstration. The system is supported by APIs that leverage smart defaults, code examples as a form of self-documentation, and extension of an existing metaphor for software interfaces that meet the needs of users previously not exposed to analog electronics or signal processing.

As a whole, this system can increase the approachability of these tools and reduce the need for in-person support to teach these concepts, enabling the teaching of lab classes away from a laboratory setting.

Chapter 4

Mixed Signal Applications

4.1 Design Approach

In order to demonstrate the utility of the platform as a tool for mixed-signal prototyping and education, several projects were undertaken. As a part of this effort, the Analog Shield was released to the public as open-source hardware. To coincide with this release, several Instructable-style teaching examples were developed. These are based on the annotated project-walkthrough style of the Instructables.com website. These rapid development and simple design of these examples demonstrate the utility of the platform as a tool for mixed-signal prototyping and have value as educational tools.

After release, the Analog Shield was integrated into the curriculum of EE122A - Analog Circuits Laboratory at Stanford, taught by Professor Kovacs for several decades, to demonstrate the viability of the platform for use as a tool in a real-world educational environment. Independently, Professor Mark Wickert at the University of Colorado developed a course curriculum using the Analog Shield and a compatible microcontroller [53]. Some of the basic educational examples have been included with the Analog Shield and DSP Shield driver libraries while others have been packaged

as more complete tutorials. The second category, DSP applications will be discussed in Chapter Five.

4.2 Analog Shield “Instructables”

With the release of the Analog Shield, a suite of four tutorials was developed. A well-developed Instructable contains step by step instructions to recreate a project. For the projects released with the Analog Shield, these include photos, diagrams, an explanation of theory, and code. Each example provides students with a practical demonstration of a device as well as an explanation of the underlying theory, and factors motivating design decisions. They were created to both demonstrate how the platform facilitates the construction of circuits and its application to the teaching of mixed-signal topics not commonly approached with an embedded system by modern students or hobbyists.

Because the projects describe simplified versions of systems that one might find on a laboratory bench, they also help demystify the monolithic bench-top test instrument. They demonstrated that the tools students use in the lab may be complicated but are not fundamentally different from circuits they build on a breadboard. All the Instructables discussed here are included as Appendix F.

4.3 DDS Signal Generator Instructable

The first tutorial uses an Arduino, Analog Shield and basic character LCD to produce a (relatively) low bandwidth but well documented function generator using the DDS technique, which is often used in modern professional test equipment.

The finished device, depicted in Figure 4.1 can output a reproduced waveform at frequencies from 0.01 Hz to approximately 10 kHz with step sizes of 0.01 Hz. It can output four waveform types: sine, square, sawtooth and triangle. The sine waveform

can be replaced by the user with an arbitrary waveform of their own design by copying the data into the definition at the top of the code and re-uploading. This allows for more complicated waves and could be used to simulate biological signals such as ECG or PPG.

A DDS function generator works by computing the phase offset of the next sample of a repeating waveform, given sample rate and the frequency of the output waveform. This phase offset is then applied to a lookup table containing a single full period of the desired waveform. The resulting sample is output via a DAC and passed through a reconstruction filter to interpolate the sampled output in continuous time.

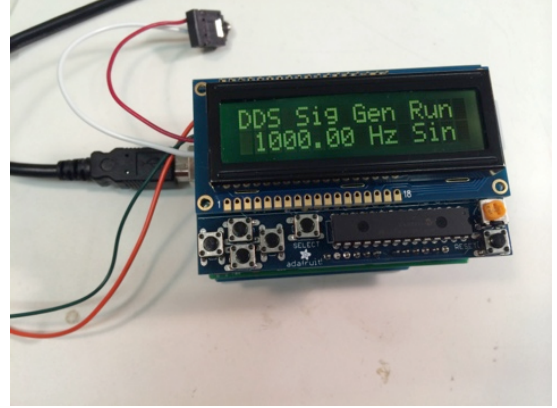


Figure 4.1: DDS Function Generator using Analog Shield, Arduino, and character LCD.

one were to produce a 1 Hz output, then output of every tenth sample (stride of ten) would produce a 10 Hz output. By calculating how many samples to skip, a DDS algorithm can precisely produce any frequency, up to the Nyquist rate of the system [54]. Equation 4.1 shows how the step size, M (also referred to here as “tuning word”), is computed for a given frequency and table length.

$$f_0 = \frac{M \cdot f_c}{2^n} \quad (4.1)$$

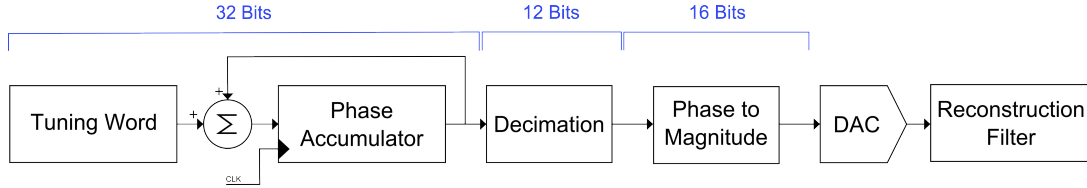


Figure 4.2: DDS Function Generator software high level block diagram.

Figure 4.2 shows a block diagram of the system as used in the Analog Shield DDS Instructable. This implementation uses a twelve bit (4096 point) lookup table with a 32-bit phase accumulator ($n=32$) and an f_c of 41.9 kHz.

After the phase is computed, a decimation operation equal to the length of the output lookup table is applied to the index. In this case, only the top twelve bits are preserved, for the aforementioned 4096 point (twelve bit long) lookup table. Because the computed phase wraps back to the beginning of the table when striding past the end, the algorithm simulates an endless signal of any frequency with only a single stored period of the waveform.

The twelve bit sample position is applied to the lookup table which contains sample values (in this case sixteen-bits). The result of this lookup is then output via the Analog Shield DAC.

The detailed algorithm used in the DDS Instructable is given in equation 4.2. In this implementation, the phase accumulator is incremented by the tuning word variable, which is the pre-computed value of M from Equation 4.1 for the desired frequency (e.g. 1 kHz). The 32-bit phase accumulator is right shifted twenty bits to keep only the top twelve bits. The result is used to look up a sixteen-bit sample value from the 4096 point table. Finally, the sixteen bit sample is output via the DAC.

$$\begin{aligned}
phaseAccumulator &= phaseAccumulator + tuningWord \\
tableIndex &= phaseAccumulator >> 20 \\
value &= lookupTable[tableIndex] \\
analog.write(0, value)
\end{aligned}
\tag{4.2}$$

For signals containing significant harmonic frequencies above the fundamental (e.g. square or sawtooth waves), the practical maximum frequency in any DDS system is well below the Nyquist limit. This is due to a phenomenon called “edge jitter” wherein the sharp edges of a waveform with high-frequency components (such as a square or sawtooth wave) fall between two sample points, and thus appear to jitter back and forth between the two instants in time. To address this, the design uses a trapezoidal ramp as a band-limited approximation. This means that multiple samples fall along the (less steeply) sloped portion of the edge, reducing jitter with only a small reduction in sharpness of the apparent waveform. The extra harmonic components of such waveforms does still impose limitations in DDS systems however, as these signals must be oversampled. In the case of the Analog Shield DDS, the maximum output of a square wave is on the order of 2-3 kHz before significant distortion appears, as opposed to 16 kHz for the sine wave.

While there are many commercial implementations of DDS systems, including low-cost integrated circuits such as the AD9838 (Analog Devices Inc, Norwood MA) which might be suited to a dedicated device, the DDS Instructable focuses on an exploration of the theory and design choices made in the development of a DDS system, as opposed to offering maximum performance.

The complete DDS software is moderately complex as it supports multiple waveforms and controls the LCD and menu system. Due to memory constraints on the Arduino Uno, waveforms for square, sawtooth and triangle signals could not be stored as full lookup tables. Instead, these waveforms are computed directly in real-time, but the DDS algorithm uses the resulting data as if it was stored in ROM memory. The core

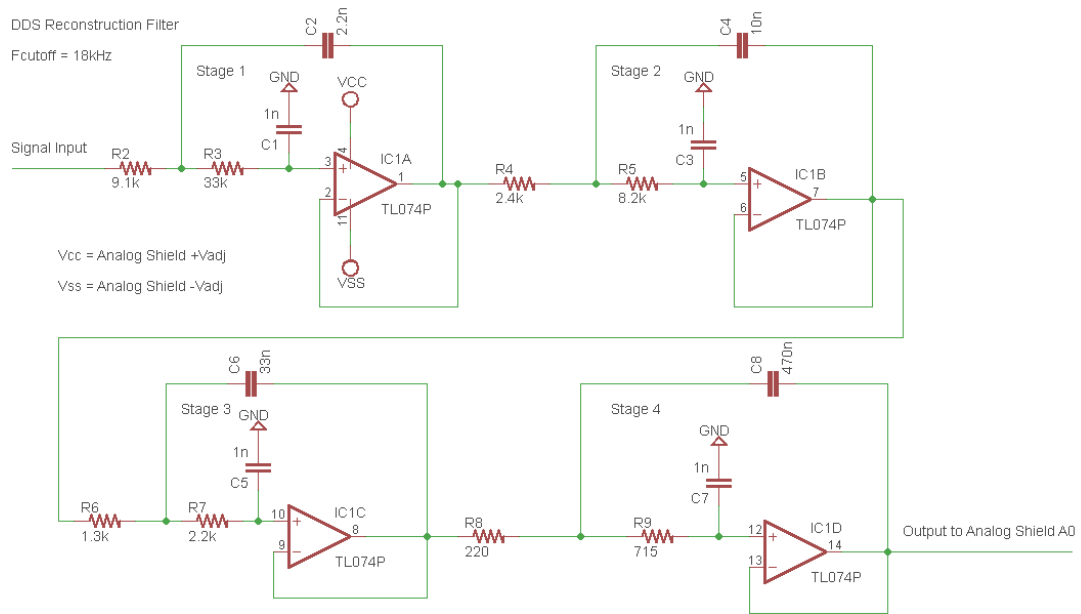


Figure 4.3: DDS Reconstruction Filter.

code of this algorithm, described above, is only a few lines long and is embedded in-line in the Instructable, while a full code listing has been made available as an open-source program on github.com/wespo.

The Instructable also goes into detail on need for (and design of) a reconstruction filter, shown in Figure 4.3. It compares the behavior of the system with no reconstruction filter, a simple resistor-capacitor filter, and an 8th order Sallen-Key topology opamp filter. It encourages the student to build the latter, providing a real-world demonstration of a key signal processing concept. The exercise provides analog electronics design experience, showing students how a filter can drastically improve the performance of a sample system.

This fusion of analog filter design and digital signal generation demonstrates the mixed-signal utility of ASPEN. It demonstrates how the analog and digital domains can merge in a design to produce powerful tools that could not be realized as easily in a purely analog or digital system.

4.4 Spectrum Analyzer Instructable

The spectrum analyzer Instructable, shown completed in Figure 4.4, uses an Arduino, Analog Shield, and an LCD touch screen shield, with a display similar to those commonly found in older cell phones. The Instructable outlines a device that displays the power spectral density of a signal in real-time, although the signal is discontinuously sampled. The resulting system can be configured while running to output on a linear or logarithmic scale and the user can select several bandwidth spans between zero and 1 kHz to 35 kHz.

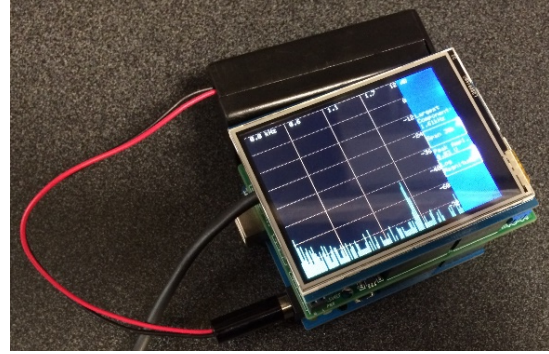


Figure 4.4: FFT spectrum visualizer, using Analog Shield and Adafruit LCD. Input is an electret microphone, out of frame.

Referred to in the Instructable as the “FFT Spectrum Visualizer” the program has a straightforward flow. With each iteration of the main loop, it acquires an array containing 256 samples of sixteen-bit data from the Analog Shield at the specified sample rate. Once the array has been acquired, the signal is multiplied with a Hamming window function. The result is then passed to a Fast Hartley Transform (FHT) implementation, developed by Open Music Labs [55]. The FHT is a sibling of the Fast Fourier Transform that saves computation time for real-valued (as opposed to complex) input signals. Finally, the resultant spectrum is scaled (logarithmically or linearly, as appropriate) for display on the 320×240 pixel LCD. Updating the buffered LCD (using an efficient delta update scheme), as well as detecting touchscreen input and minor housekeeping takes a few tens of milliseconds. The process then repeats, sampling the next frame of data, and producing a reasonably quick refresh rate of approximately 15 frames / second. As mentioned, the system has a user adjustable

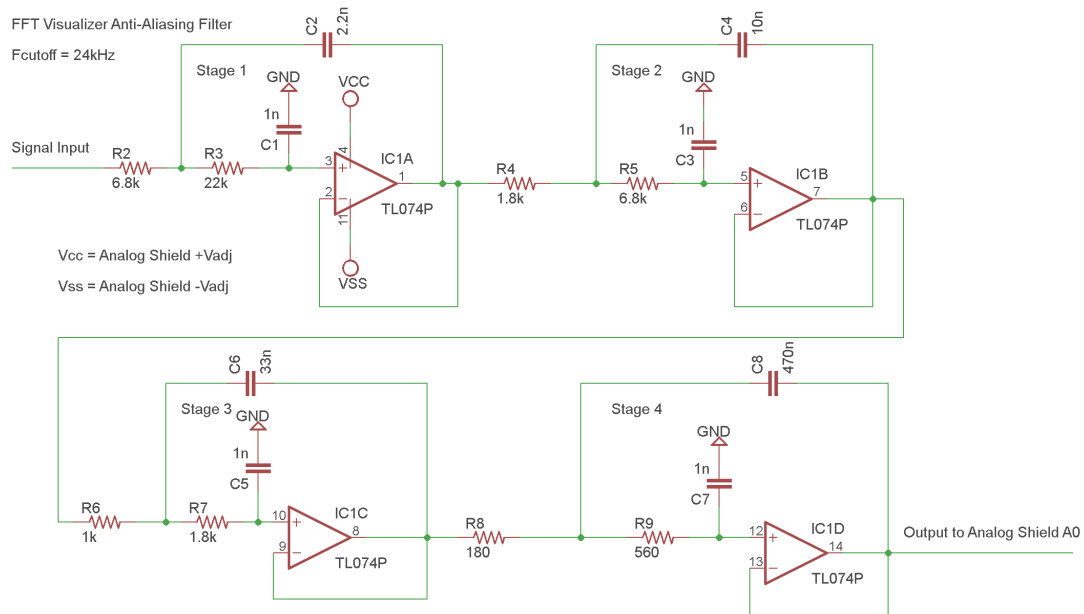


Figure 4.5: Anti-aliasing filter.

frequency span, affecting the system sample rate. For slow sample rates (with a frequency span close to 1 kHz, and longer acquisition times) the screen update delay is a fairly small part of the overall display frame rate (a few frames per second). At high sample rates, the screen refresh time is the dominant factor and the display output appears to be continuous and video-like.

Included in line in the Instructable is a section of the code not meant to convey the entire program in detail, but merely illustrate the flow of the main program loop. The entire program has many helper functions and takes up several hundred lines, not including the invoked libraries, and is provided as an attachment to the Instructable, as well as shared on github.com/wespo.

Complimentary to the section in the signal generator tutorial, the spectrum analyzer tutorial includes a discussion of the need for and design of an anti-aliasing filter. This filter is an 8th order 24 kHz low-pass Chebyshev filter, using a single quad op-amp chip, shown in Figure 4.5.

This design was chosen as it enables the implementation of a filter with a relatively steep roll-off in a compact form factor, using a single IC. Making use of the Analog Shields built-in bipolar power supply enables the entire circuit to fit on the breadboard included with the Analog Shield, as shown in Figure 4.6. As with the signal generator, it demonstrates how mixed-signal design is fundamental to developing a fully realized system that interfaces with the real world. In addition to the filter, a circuit is included to drive an electret microphone as an input to the spectrum analyzer, allowing the student to investigate ambient sound as a resource. Amusingly, in the process of developing the tutorial, the device revealed a tone at approximately 30 kHz being emitted by the ventilation system in the Paul Allen building at Stanford.

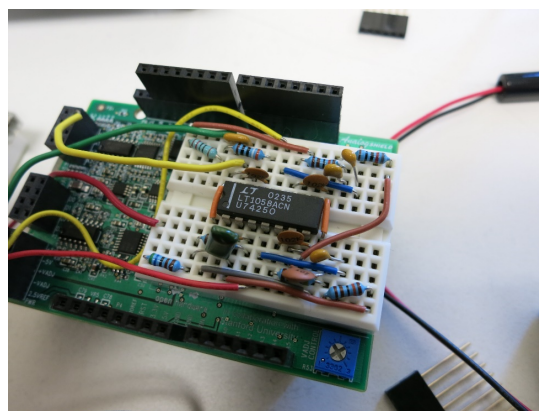


Figure 4.6: FFT Visualizer anti-aliasing filter, using a single quad op-amp IC to implement an 8th order Chebyshev filter with a 24 kHz cutoff.

Conceptually, this Instructable serves to demonstrate the utility of the Fourier transform itself, as well as the crucial role of an anti-aliasing filter. More importantly, it can be a powerful teaching tool when the filter is removed, allowing a hands-on demonstration of spectral folding. Using an adjustable function generator for sinusoidal input, the student can observe spectral folding - watching the peak move from the left of the spectrum display to the right and fold back again as they increase input frequency. Allowing the student to control the frequency and observe spectral folding hands-on provides a much clearer demonstration of the phenomenon than an explanation in a lecture can. As a whole it demonstrates how the Analog Shield can be used as a teaching tool. It both enables demonstration of a complicated phenomenon and provides basic functionality that otherwise requires expensive test equipment.

The polyphonic music Instructable is based on a modified form of the DDS signal generator code. Instead of maximizing the bandwidth of a single output channel, the system developed in this Instructable uses all four channels of the Analog Shield DAC simultaneously. These channels are mixed using a simple resistive summing network to demonstrate additive synthesis of a polyphonic sound from pure sinusoidal sources, showing students how each generated sinusoidal source can be summed to generate a more complex waveform in real-time. The summing operation is performed by a simple resistor summer circuit, shown in Figure 4.7.

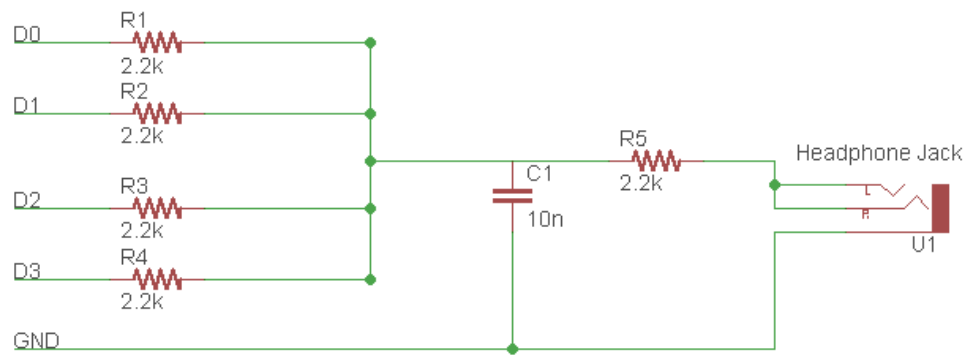


Figure 4.7: Resistive summing network for polyphonic music Instructable.

4.5 Polyphonic Music Instructable

Using a program provided for the PC, the user can convert any standard MIDI file into an array, which they can paste into their Arduino code for playback via the Analog Shield. The example song included an approximately six-minute sample of Toccata and Fugue in D minor (Johan Sebastian Bach, c. 1730), which can be output to headphones to demonstrate how multiple waveforms can be summed to create complex sounds. The output can also be input to an oscilloscope to demonstrate the principal visually, as can be seen in Figure 4.8.

As with the previously discussed tutorials, the polyphonic music tutorial demonstrates both the utility of the Analog Shield as a part of the ASPEN stack while teaching directly identifiable principles of both signal processing and analog electronics. Unlike those tutorials, the circuit construction is quicker and more rudimentary, but the result is more likely tangible to a novice.

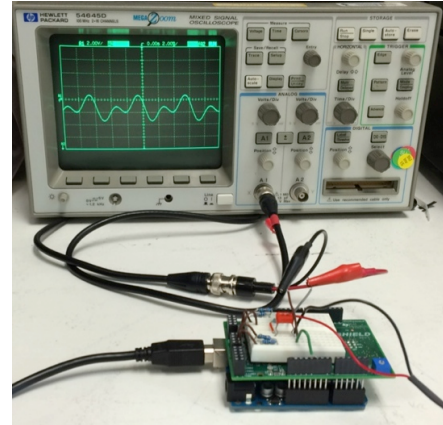
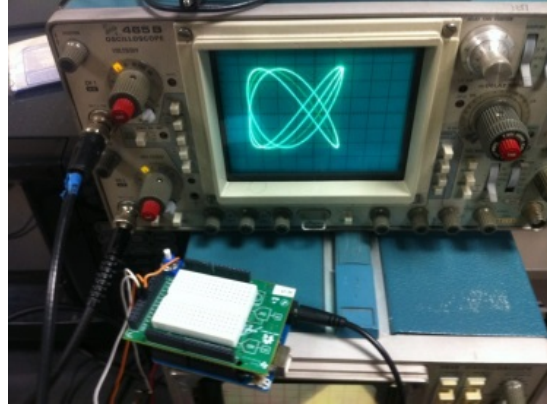


Figure 4.8: Four voice MIDI playback using the Analog Shield, with output on an oscilloscope.

4.6 Lissajous Pattern Instructable

The fourth Instructable example released at the Analog Shield launch was an X-Y Lissajous Curve generator. A Lissajous pattern is the curve generated by two sinusoidal signals plotted against each other with one on each axis [56]. The ratio of the frequencies of the two waveforms, as well as the phase between them controls the number and shape of the lobes plotted.



Using DDS on two channels of the Analog Shield DAC, the tutorial does require an X-Y capable oscilloscope or display for visualization. The code included outputs a sine wave at a fixed frequency on one channel and repeatedly plays a chirp on the second, creating a changing pattern displayed on screen, as seen in Figure 4.9.

Figure 4.9: First generation prototype Analog Shield with Lissajous pattern generator code running, displayed on an oscilloscope.

4.7 Network Analyzer Instructable

Shortly after the release of the Analog Shield, a fifth Instructable was developed. This scalar network analyzer, is shown in Figure 4.10. The network analyzer provides a characteristic curve of the transfer function of a two-port device, typically used for applications such as filter analysis. While a scalar network analyzer is not lab grade as it does not provide the user with the phase information that a more complex vector network analyzer does, it has utility for basic component characterization and as a concrete demonstration tool.

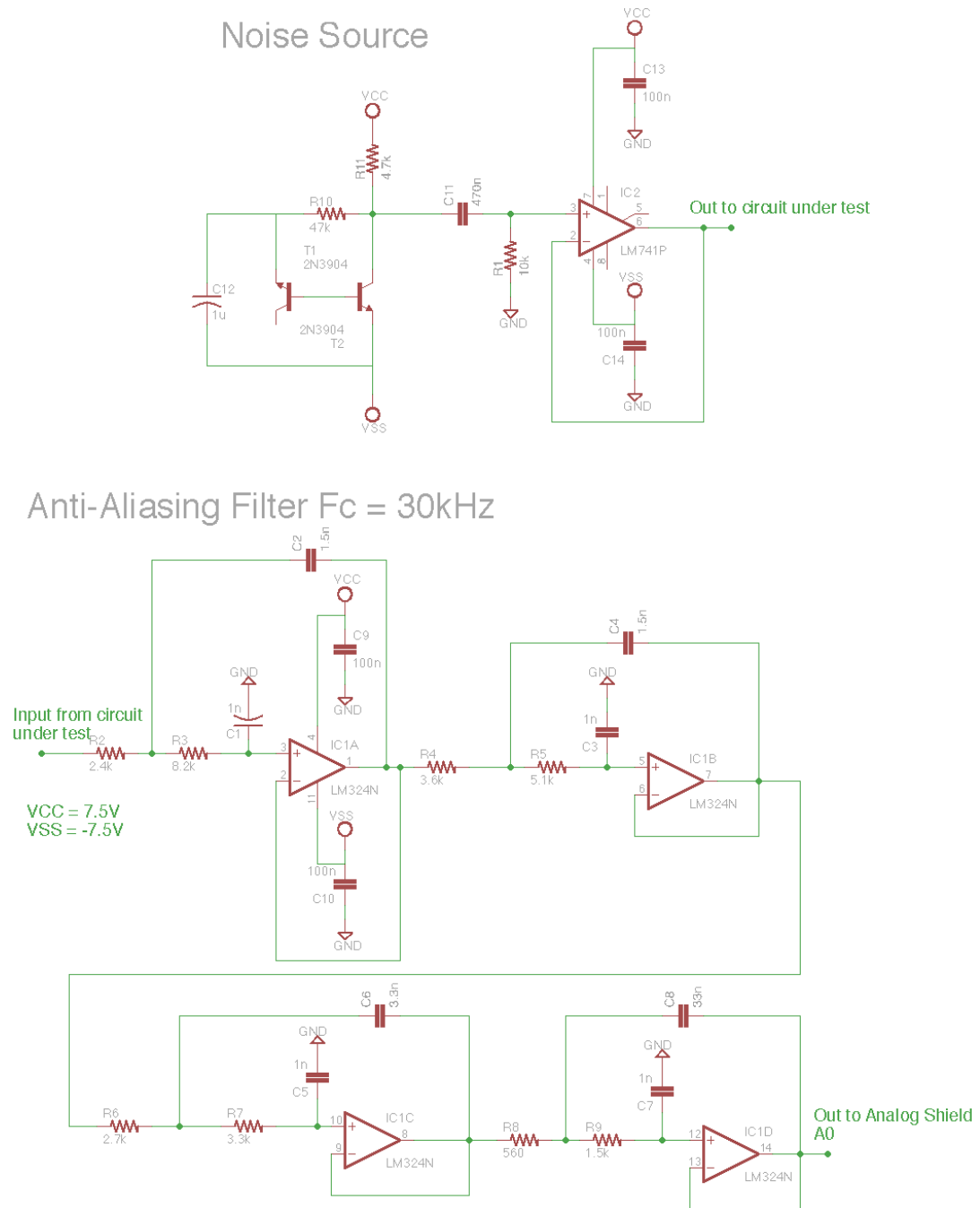


Figure 4.11: Network analyzer noise source (top) and anti-aliasing filter (bottom).

An FFT-based analyzer, it uses a white noise circuit as an input to the device under test and averages the computed power spectral density over several sampled periods. The resulting transfer function is then plotted on an attached LCD touch screen (of the same type as used in the spectrum analyzer demonstration). The noise source (shown at the top of Figure 4.11 is a student-

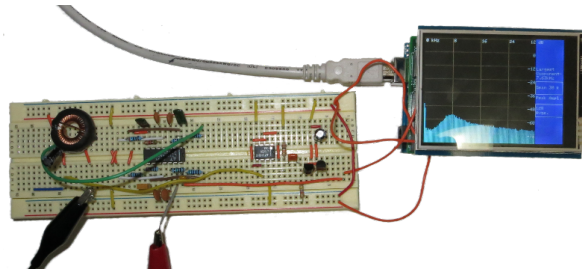


Figure 4.10: Analog Shield network analyzer showing measurement of an LC bandpass filter.

built circuit using a bipolar transistor in avalanche breakdown to generate white noise needed for the system. This circuit is enabled by the 15 V range of the Analog Shield power supply, as the 5 V available on common microcontroller circuits could not drive the bipolar transistors used in the circuit into the breakdown region. Additionally, students are asked to build an 8th order anti-aliasing filter, similar to the one from the spectrum analyzer Instructable (Figure 4.11, bottom).

This Instructable demonstrates several signal processing concepts. It demonstrates the utility of white noise as an actual practical tool. Additionally, it shows a practical application of the Fourier transform as a tool to understand the behavior of a system. As a demonstration of the Analog Shield itself, it shows how the power supplies and interfaces simplify the process of building circuits that might otherwise be difficult for students to implement. It also shows how these tools can be leveraged to make accessible more complicated concepts than those traditionally found in hobbyist electronics kits. These Instructables would also enable a student to construct basic versions of the key instruments for an audio or biomedical lab workbench.

4.8 Kit use in EE122A

Stanfords Analog Circuits Laboratory course, EE122A was a hands-on electronics course, covering commonly taught undergraduate electronics topics such as operational amplifiers, oscillators, filters, opto-electronics, and more [57]. The first five to six weeks of the course are comprised of lectures and lab exercises covering these topics. The second portion of the course involves the students working individually or in groups, developing a unique, self-defined electronic project. The project is proposed by the students and refined with staff before being approved by the professor.

4.9 EE122A First Integration

In fall 2014, and 2015 the course format was similar to previous years - a traditional laboratory course, with components of ASPEN added as a part of some of the laboratory exercises. Historically, this course was taught in a laboratory setting, with a teaching assistant supervising student activities in conjunction with the professor. The staff would provide in-person support and feedback as the students worked through the labs and final project. Prof. Kovacs experience in teaching this course had been informative in defining the needs of the ASPEN system as a whole, so it was natural to integrate the tools developed as a part of the project into the course material. This version of the course combined the Analog and DSP Shield with the (then) new Electronics Explorer (Digilent, Pullman WA) and the existing high-performance test equipment that had been previously used in the lab. The Electronics Explorer is a USB oscilloscope, power supply and function generator embedded in a breadboard form factor.

The Analog Shield was introduced to the students as a part of the first lab exercise, paired with the Electronics Explorer. Students worked with two of the tutorial programs provided with the Analog Shield. As part of the experiment, students were introduced to the effects of quantization on a signal. In this exercise, students pass a

low-frequency triangle wave into the first channel of the ADC on the Analog Shield and observe the four channel outputs of the DAC. The system samples input on the ADC, and immediately reproduces it on all four output DAC channels of the Analog Shield at varying levels of fidelity. Channel one repeats the signal at full sixteen-bit resolution. Channel 2 shows a 6-bit output, channel 3 a 4-bit output and channel 4 at 2-bits (channels 3 and 4 shown in Figure 4.12). This is achieved by masking out the lower bits of the sampled data before sending the signal to the DAC. Students were asked to record the signals on all four channels overlaid. Students also used a spectrum analyzer embedded in their oscilloscope or Electronics Explorer to observe and comment on the spectral and harmonic content of the signal, as well as the impact of quantization on that content.

The second example that students were asked to run demonstrated sample rate effects. Again, a signal was sampled on the first channel of the Analog Shield ADC and reproduced on the four channels of the output. As with the previous example, the channels had different versions of the input waveform. Channel one had the input reproduced at the maximum sample rate of the system, channel two updated the output every second cycle, channel three every fourth, and channel four every eighth cycle, using a software sample and hold approach. Students were then asked to vary the frequency of the input signal and observe how the varying sample frequency defined the Nyquist rate of their system. Students observed aliasing at different frequencies based on the differing Nyquist frequencies of the various channels.

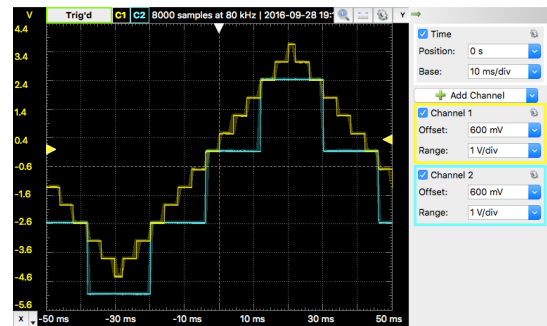


Figure 4.12: Outputs of the quantization example on an oscilloscope. Students were asked to compare the output of the four channels of the Analog Shield DAC. These outputs were four copies of a single ADC input quantized at various levels.

These are the first two interactions students had with the Analog Shield, and in a few minutes, students are exposed to pitfalls of a sampled digital system and how changes in the sample resolution and frequency can drastically affect how the signal they are trying to observe or measure is reproduced.

4.10 EE122A Take Home Lab

In fall 2016, EE122A was substantially revised to move the course outside the laboratory. In this version of the course, students used a provided tool kit to perform their lab work outside the laboratory space and on their own schedule. This revision demonstrates how ASPEN can enable a massively online open course (MOOC) style laboratory course. To facilitate this, each student was provided with a Lab-In-A-Box kit, shown in Figure 4.13. The kit included a breadboard, Arduino UNO, Analog Shield, DSP Shield (distributed separately as a part of the third lab assignment), and a Digilent/TI My Parts Kit (which includes resistors, capacitors, integrated circuits, and discrete components). Students were also issued a breadboard wire kit, additional lab relevant discrete components. Also included was the Analog Discovery 2 (Digilent Inc., Pullman WA) a USB oscilloscope, logic analyzer and function generator. Prior version of the device by the same manufacturer, the Analog Discovery and the Analog Explorer had been used in the lab version of the class as a companion to conventional full-size test equipment.

Application of a higher performance microcontroller platform promises the opportunity to both process and stream signals from a single device, which would facilitate the omission of the Analog Discovery. A similar cost, higher performance ARM microprocessor such as the Teensy 3.6 represents a potential candidate to replace the Arduino Uno and will be considered in future implementations and might enable the use of the Analog Shield as both the circuit prototyping platform and as a low-cost test instrument for the course, replacing the Analog Discovery 2.

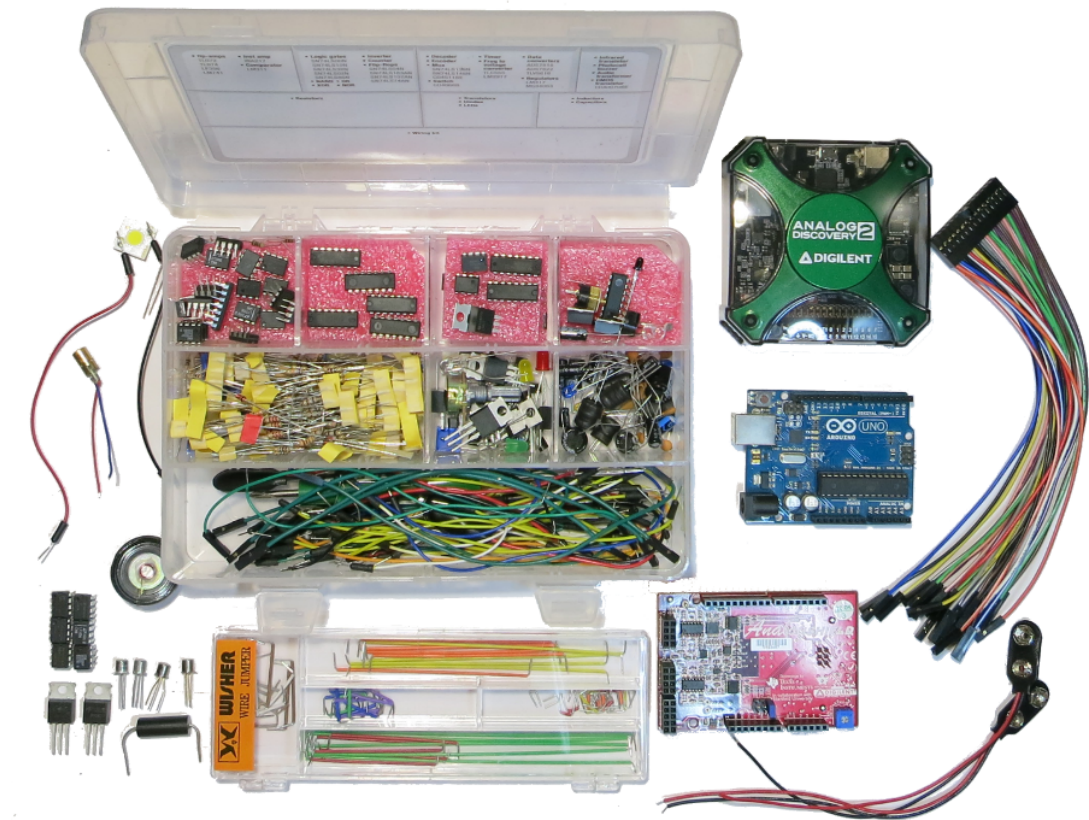


Figure 4.13: EE122A lab kit from fall 2016. The kit contains an Arduino Uno, Analog Shield, Analog Discovery 2, breadboard, wire kit, TI My Parts Kit and miscellaneous additional components.

The interaction model used in the course followed several steps. The week before performing a lab exercise, students were asked to perform a pre-lab. This assignment generally included conceptual questions, theoretical calculations, design work, and SPICE simulation of a designed circuit. The following week, students were expected to perform the assigned experiments, in parallel with beginning the next pre-lab. Students would use the provided kit to build their designed circuit and use the Analog Discovery to collect data and validate that their experiment matched their simulations. Additionally, virtual office hours were held several times per week, to demonstrate the viability of supporting lab exercises in a virtual environment. Stanfords canvas course support website includes support for Big Blue Button, (BigBlueButton Inc., Ottawa, Ontario, CA) online conference software tailored for teaching applications. The feature set used for EE122A was desktop sharing, video conferencing, voice conferencing, and text chat. All of these features are supported by alternative free conferencing software options, such as Skype (Microsoft Inc., Redmond, WA) or Google Hangouts (Google Inc., Mountain View, CA). With a PC based oscilloscope (e.g., the Analog Discovery 2) and conferencing software, a student can be asked to probe a physical point in their circuit and the resulting signal can be viewed live by an instructor from anywhere in the world.

4.11 Lab Exercises in EE122A

The first lab focused on familiarizing students with their tools and understanding signal acquisition. The exercise included the Analog Shield examples from previous years as described earlier, with students familiarizing themselves with the tools, learning about quantization and sample rate, as well as designing and characterizing a simple RC filter. The second lab exercise focused on signal manipulation and the effects of the limitations of amplifiers in a chain on a signal. It included comparison of the bandwidth and slew rate of four different types of operational amplifier. In previous years, students had used a pre-built board which allowed the comparison of four amplifiers simultaneously, shown in Figure 4.14. In 2016, this was replaced with

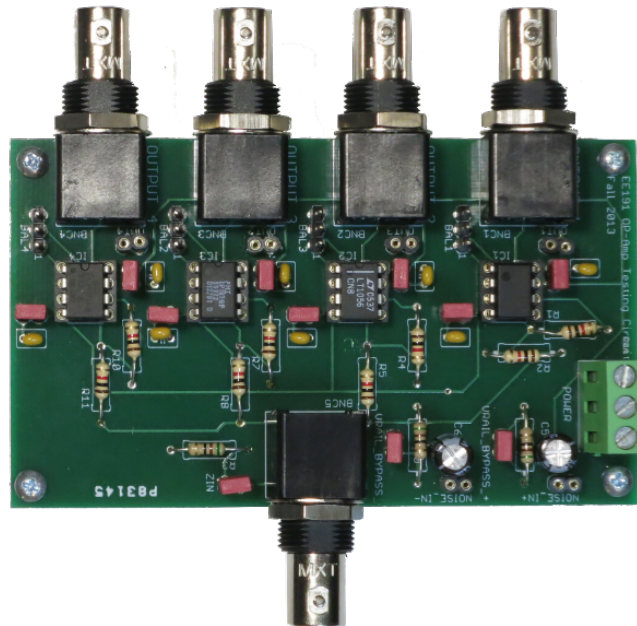


Figure 4.14: Four channel op-amp comparison board used in 2014 and 2015. Input at top, with the signal repeated and buffered by four op-amps of different types configured for unity gain.

application of the Analog Shield and breadboard (as a platform for circuit construction). As in many other applications, the power supply of the Analog Shield enabled the use of a wider range of amplifiers than would have been possible even with the Analog Discovery 2.

Additionally, the experiment asked students to develop and characterize integrator, differentiator, and relaxation oscillator circuits. The Analog Discovery 2 was the primary measurement tool with the Analog Shield providing the support infrastructure to build the circuits.

The third lab exercise focused on filters. Students used the DSP Shield and three additional filter boards expressly developed for the class by Prof. Kovacs, with layout by William Esposito. These are depicted in Figure 4.15. The purpose of the lab was to demonstrate to students how several completely different technologies could be used to implement a filter while maintaining the same characteristic response, as well as

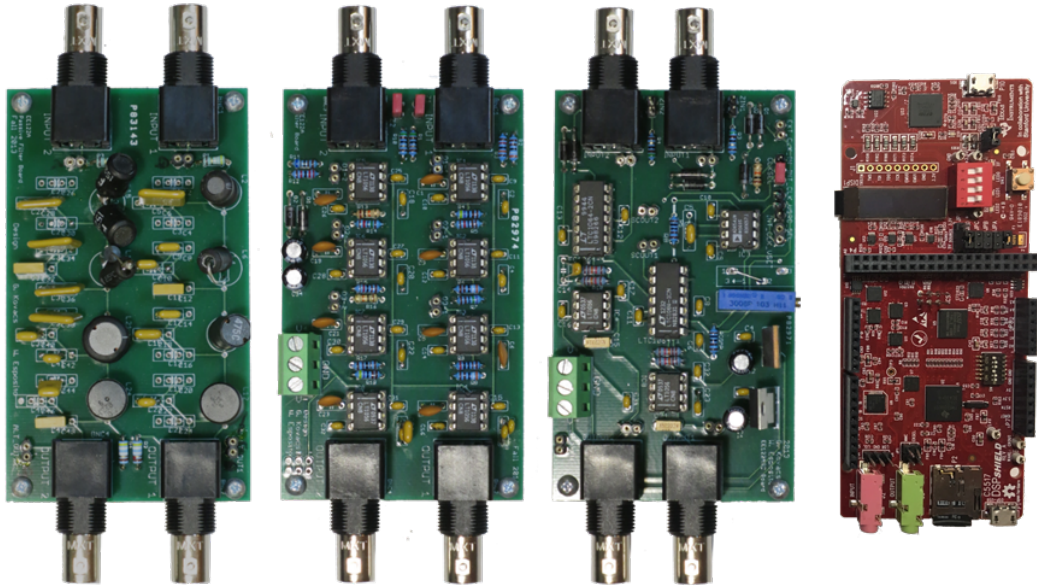


Figure 4.15: The filter boards used in the EE122A filter comparison lab. From left to right, passive LC filter, op-amp filter using Sallen-Key topology, switched capacitor filter, DSP Shield.

demonstrating the various trade-offs between the technologies. In this experiment, in a single sitting, students were asked to characterize and compare the behaviors of eight different 1 kHz low-pass filters, two of each of four technologies. These technologies were passive LC (inductor capacitor), Sallen-Key op-amp, switched capacitor, and DSP. The first three types of filter were implemented on three custom designed circuit boards, to save the students several hours which would be spent building and debugging each circuit.

This experiment is the first known example of a lab where a single student has been able to apply and characterize such a wide array of filters at once and provides an example of how these concepts can be demonstrated to students rapidly with the right tools. The fact that this could all be done using a portable lab kit, far from the physical lab space demonstrated the viability of the tool-chain again as a teaching system. In addition to detailed characterization of the various filters, as a part of their

Filter Type	Power Consumption	Degree of Programmability	Maximum Signal Input w/o Distortion
L-C	0 mW	None (requires soldering)	>25 V (breakdown voltage of components)
Sallen-Key	260 mW	None	7.5 V (range of op-amp with headroom)
Switched Capacitor	225 mW	Adjustable frequency with potentiometer.	4 V (range of SC chip with headroom)
DSP	320 mW	Full programmability of frequency and type.	1.5 V (range of DSP Codec)

Table 4.1: Sample comparison table asked of students to evaluate the trade-offs between the various filter technologies they used in the experiment.

report students were asked to produce a tabulation of the various trade-offs among the types of filter architectures, as shown in Table 4.1. These tables included a comparison of metrics such as noise performance, flexibility, cost, power, and complexity of implementation. These observations were based on students detailed empirical observations of each filter type.

The filter boards seen in Figure 4.15 were designed for the earlier in-class version of the course, and as a result the cost of these boards is prohibitive - on the order of \$50 each. Additionally, each is as large as the Analog Discovery 2 that students used to evaluate them. This makes them a prohibitively expensive and bulky addition

to the ASPEN stack. In order address this, a future revision of the boards using smaller, cheaper surface mount components and connectors has been proposed. This need not compromise circuit performance or simplicity - and initial design work has already confirmed its feasibility. The earlier design was largely driven by the choice of BNC connectors (for interface with larger bench-top equipment) and through-hole components (simplifying hand assembly).

As a whole, this experiment has proven to be one of the most memorable experiments in EE122A and demonstrates the utility of the approach taken in the ASPEN project to integrate systems for teaching while keeping them accessible for investigation.

The fourth lab focused on amplifiers and oscillators. The experiment had students use their lab kit to assemble a Wein-Bridge oscillator, a class-AB amplifier, and two types of class-D amplifiers. Again, students designed and simulated first, then used the lab kit to provide a signal source and power supplies. Students then acquired and characterized the response of the various circuits developed. Students were asked to compare parameters such as dynamic range and THD for each circuit, and tested the effects of various guided design improvements to their class AB amplifier, characterizing as they went.

The fifth lab was arguably the most complex application of the Analog Shield in any of the experiments. The exercises focused on opto-electronics, and device characterization. The first part of the lab had students build a curve tracer circuit using the Analog Shield.

The curve tracer is an instrument which produces a family of current / voltage curves of two- or three-terminal devices by sweeping the voltages at two of the terminals and measuring the current through the third terminal. The fancifully named Analog Shield Curve-o-Tracer (ASCoT), seen in Figure 4.16 was designed to demonstrate to students the feasibility of building useful test equipment from a handful of components and using those tools to a practical end. The final device that the students built is shown in Figure 4.17. It has a low power supply voltage of 18V, provided by two 9V batteries in series (unlike most commercial curve tracers which have maximum

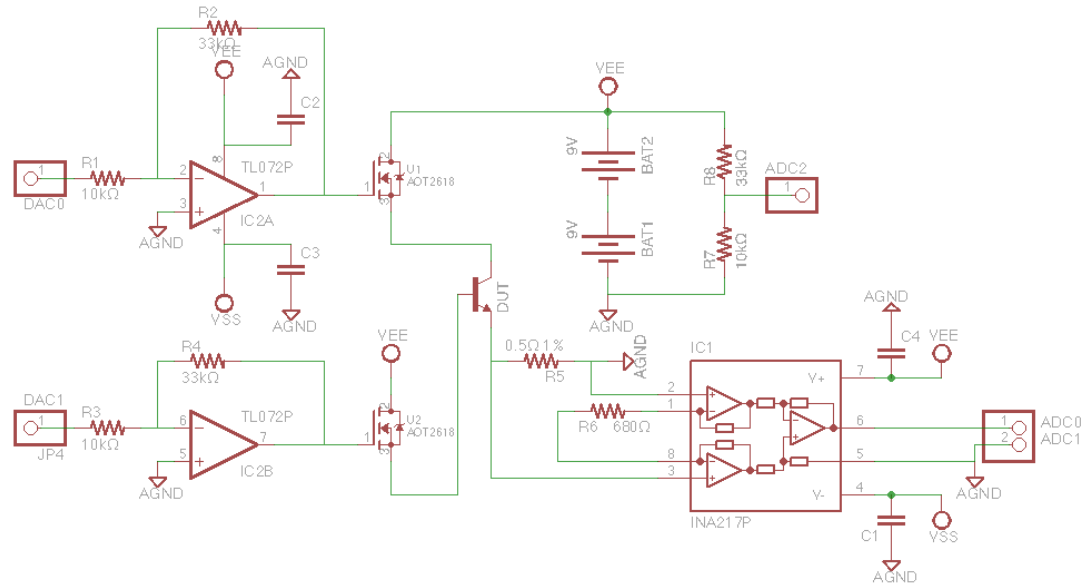


Figure 4.16: ASCoT circuit schematic. Note that the device under test (DUT) is depicted as a bipolar transistor in this case.

voltage ranges of several hundred volts). Limiting the voltage of the ASCoT ensured student safety and simplified the design by removing the need for a high voltage power supply. The low 18 V supply voltage necessitated use of a $0.5\ \Omega$ series current measurement resistor, due to the inability of the circuit to support a large burden voltage for the current sense resistor without substantially impacting overall dynamic range. As a result, the measured voltage across the sense resistor was on the order of a few hundred microvolts for low current values - which required the sensitivity of the Analog Shield to capture accurately. This constraint challenged students, requiring careful layout of their implementation to avoid excessive measurement noise. To control the ASCoT and plot or export the resulting data, the lab included a specially written PC application. It is depicted in Figure 4.18 showing the resulting trace of an NPN bipolar transistor such as those given to students. Because this was a basic demonstration for educational purposes, the transistor was controlled with a limited voltage range applied to the base instead of using proper current control.

Once the students had completed the circuit, they were asked to characterize and produce plots for older bipolar and Darlington transistors in their kit, chosen because their gain was sufficiently low to allow for several characteristic curves to be plotted within the dynamic range and step size constraints of the system.

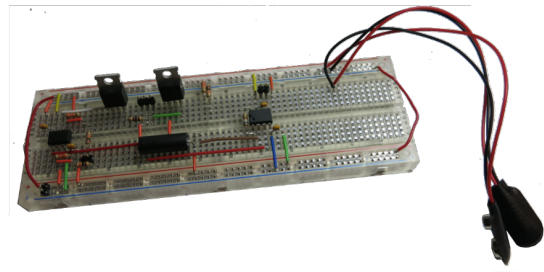


Figure 4.17: Analog Shield curve tracer circuit, disconnected from the Analog Shield.

Next, students were given a black box system, to analyze. This device, was an LED brake light array from an automobile. Using the curve tracer (or other method of their choice) the students were instructed to deduce the internal structure of the simple circuit, which was encased in black tape obscuring the connections. With some prompting, and using the ASCoT for measurement, students were able to successfully reconstruct the correct circuit diagram, which had multiple LEDs in series, a series current limiting resistor and a parallel bypass resistor. Students also deduced the values of the components and the number of LEDs in series, by evaluating the shape of the current / voltage curve they produced. Evaluating the slope of the curve, breakpoint of the forward turn on voltage of the LEDs and using their knowledge of diode behavior, students were able to determine number of diodes in series and parallel, the series current limiting resistor value and the parallel load resistor embedded in the circuit.

In the next part of the experiment, students were provided a high power white LED chip and asked to use their Arduino and a cadmium sulfide (CdS) photo-resistor in their kit to develop a constant brightness light source, which would dim the LED based on incident light.

The final exercise in the experiment was a laser transceiver, which used a provided red laser diode, photo-diode, and voltage to frequency (V-F) converter IC. Students were

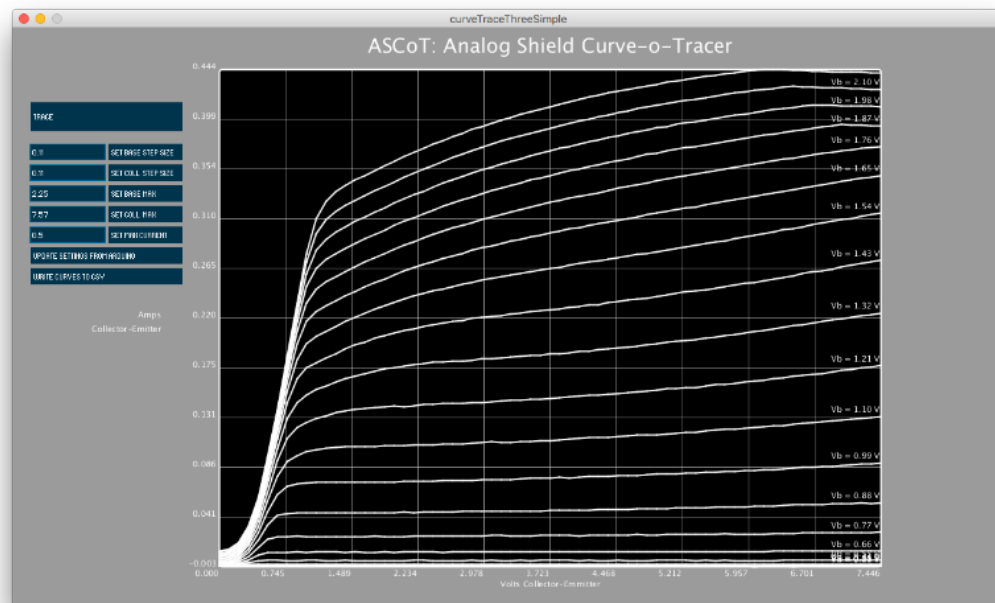


Figure 4.18: ASCoT PC application, showing the result of tracing a 2N222A NPN transistor.

asked to design a system using the V-F IC to modulate a current source they design to drive the laser. Students were also asked to develop a transimpedance amplifier circuit to amplify and demodulate signal received at the photo-diode.

4.12 EE122A Conclusions

As a whole, the use of ASPEN in EE122A demonstrated the feasibility of teaching analog electronics robustly, using portable, relatively low-cost tools to replace several thousand dollars of lab equipment. These tools, when brought together as a complete system, enable the teaching of EE122A outside the lab building without compromising the course content or teaching material - indeed, the tools have enabled the addition of new topics not previously covered due to time constraints. There is additional development work to be done as a result of this first take home version. Course material might be modified to allow omission of the Analog Discovery 2, and some experiments can be refined. Nevertheless, the first trial demonstrates the feasibility of the concept and the utility of the ASPEN platform as a tool for analog electronics education, both inside and outside of the lab.

4.13 Kit Use in Other Mixed Signal Applications

Because the Analog Shield has been released commercially by Digilent Technologies, it has seen a number of applications by users who have had no prior interaction with the ASPEN project or team (although Digilent stopped selling the product in 2018, the team plans to re-release the open-source hardware through other venues). These have included hobbyists performing simple experiments with the Analog Shield [58], and more complicated examples such as a synthesizer [59].

The most noteworthy example of independent application of the ASPEN tools has been the work of Prof. Mark Wickert at the University of Colorado [53]. In the paper,

he describes a hardware stack very similar to the Lab-In-A-Box kit used in EE122A, as well as its application to teaching signal processing concepts to students. This stack includes an Arduino, a first generation Analog Discovery (Digilent, Pullman WA), and the Analog Shield. Additionally, a higher performance, Arduino compatible microcontroller the ChipKit Uno32 (Digilent, Pullman WA) was used. The course material described in his paper falls somewhere between the signal processing focused material described in the next chapter and the mixed-signal material covered in EE122A.

In his paper, he characterizes the Analog Shield tool-chain with a maximum sample rate of 55.4 ks/s. This is comparable to that measured by the ASPEN team using a ChipKit Uno32. Rather than using the ADC and DAC separately, however, he uses the Uno32 as the core of a lower performance signal processing tool-chain with a maximum sample rate of approximately 24 ks/s. Much like the DSP Shield environment, his students use a frame based signal processing approach with asynchronous buffer filling. In this case, the system uses a timer interrupt rather than fully asynchronous DMA buffering as found on the DSP Shield. Nonetheless, he was able to demonstrate application of a 31-tap FIR filter in his signal chain. This does not match the performance of the 1000+ tap FIR filters seen on the DSP Shield, but is more than sufficient for teaching the fundamental mixed-signal concepts described in his paper.

He describes several DSP experiments using this ASPEN based system. First is a more detailed explanation of sampling theory. Much like the experiment described in the discussion of EE122A at Stanford, students pass a 1 kHz sine wave through the Analog Shield ADC and DAC, and then observe the image frequencies created by the sample and hold behavior of the system. Using the Analog Discovery, students compare the spectral folding they observe to a theoretical model they were introduced to using scientific tools from the Python programming environment. Through this experiment students observe that the behavior of the real-world Analog Shield system matches sampling theory precisely.

Another experiment describes using both the human voice and a smartphone as a source for a 12 kHz sine wave. Using a serial terminal, input sample rate is progressively decreased until aliasing is noticeable. Using a 16 ks/s sample rate students then introduce a 4th order Butterworth anti-aliasing filter and observe how the sine wave, previously aliased to 4 kHz is attenuated strongly by the filter.

A third experiment described uses an FIR filter and white noise generator. Students generate coefficients of a 31-tap FIR low-pass filter and compare the spectral output of filtered white noise with the theoretical response of the filter. As with the previous exercise, students compare their result with theoretical calculations.

A fourth experiment introduced students to multi-rate processing. Up-sampling of a signal was demonstrated with students up-sampling of an input 500 Hz sinusoid, and then observing the results with and without a 7-tap FIR interpolation filter.

The final experiment described in his paper was a demonstration of deliberate under-sampling. This demonstration used an input signal on an AM carrier at 1.02 MHz with 1 kHz sidebands at 60% depth. With a sample rate of 0.016 Ms/s, a 1 kHz signal modulated on a 4 kHz carrier was received as expected.

As a collection, Mark Wickert's excellent work at the University of Colorado has provided a masterful demonstration of the power and utility of the tools designed for the ASPEN project. The Analog Shield portion of ASPEN has already enabled educators and hobbyists to experiment with and teach analog electronics and signal processing concepts, validating the utility of the system as a whole.

4.14 Conclusion

Components of ASPEN have already been successfully used extensively for mixed-signal education and there are many more teaching applications to which it could be applied. The examples described here show that ASPEN is useful for teaching and prototyping such designs both inside and outside lab environments.

Chapter 5

Digital Signal Processing Applications

5.1 Overview

To demonstrate the utility of the ASPEN system for signal processing prototyping and education, a similar approach to that taken for mixed-signal systems was used. Several tools and examples were developed to demonstrate the utility of the system and make it more useful for both novice and advanced users.

As part of the development of the DSP Shield App and Shield Mode (described in Chapters Two and Three) several examples were developed using the App and library. These examples serve several purposes. They demonstrate functionality, signal processing concepts and how the Shield App can be applied practically. The Shield App was also leveraged as part of an effort to address the need to create an impactful first introduction example for signal processing, comparable to hello world in programming.

To demonstrate the system as an educational tool, the DSP Shield was deployed as a part of a graduate level digital signal processing course at Stanford. Previously,

the graduate level signal processing curriculum at Stanford had been bifurcated into two courses. The first, “EE264: Digital Signal Processing”, focused on theory. The second, “EE265: Digital Signal Processing Laboratory” had a strong emphasis on practical application. Using the DSP Shield, an optional one-credit laboratory component was added to EE264. These labs added a comparable hands-on experience, but with a substantially reduced overall time commitment for students.

By developing a continuum of DSP tools for a wide range of signal processing approaches, the utility of the Arduino-like approach taken with the ASPEN platform was demonstrated.

5.2 DSP Shield App

In Chapter Three, the shield mode and associated Shield App were described. They comprise a configurable signal processing chain that can manipulate real world signals without complicated setup. With them a student or hobbyist can perform basic real-time signal processing tasks with minimal knowledge of low level tools. An example application of the Shield App might include waveform generation, mixing with input audio, filtering the result, and output. Using the shield app and library, any one of these can be done in less than ten lines of code.

While the Shield App does not offer the hands-on signal processing algorithm implementation experience found developing directly in Energia-DSP, it serves as an introduction to the field. With it, novice users gain a new capability (signal manipulation). This hands-on success can increase confidence in, and excitement for signal processing. Building on these successes, with hardware in hand, they can transition to using the DSP Shield in standalone mode.

The fact that the Shield App is open-source and written in Energia-DSP environment also means that the code is not simply a black box. Over time these tools can be extended, modified, and improved by users in the (potentially global) community.

5.3 DSP Shield App Examples

When using the Shield App, the DSP Shield is used with an Arduino much like any other shield. To support this, the Arduino DSP Shield library interfaces with the Shield App and controls the signal chain. Like the Analog Shield library, the DSP Shield library comes with examples of applications for the system, using the Shield App blocks described in Chapter Three. These examples include both simple functionality tests like audio loopback and more complicated examples. These include custom FIR and IIR filters, time varying filters, a reverb engine, spectrum analysis, voice removal from music, and more.

It is demonstrative of the power of this tool that these signal processing tasks can be implemented with a handful of lines of code. For example, Figure 5.1 shows an example included with the library which uses three lines of code to generate a rear speaker signal for a surround sound audio system from a stereo input. It does this by first removing the common signal (centered component) from the two channels of the stereo signal, removing most of the voice signal. The signal is then slightly attenuated, and finally a moderate amount of reverb (3 ms) is applied to simulate a concert hall echo effect. These examples as a whole show how powerful of a tool the Shield App can be, enabling quick prototyping and demonstration of the capabilities of signal processing systems.

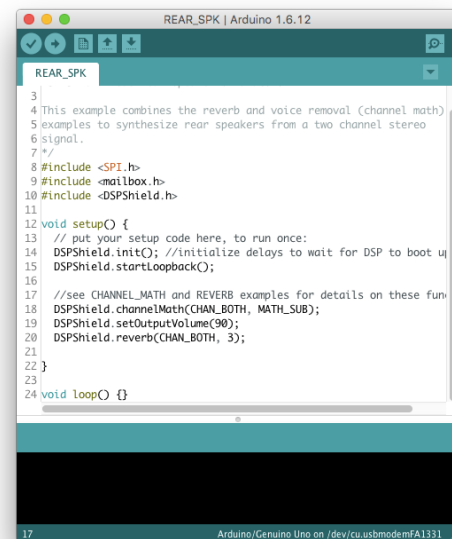


Figure 5.1: DSP Shield App example that converts a stereo audio signal to a four-channel surround sound system.

5.4 Hello World for DSP

The first computer program that most students are asked to create is Hello World - a program that simply prints that phrase to the screen repeatedly. This task allows students to validate that their tools are functional and simultaneously gives them a successful outcome which can build confidence. This simple program has become so ubiquitous that the words Hello World can be shorthand for an introductory teaching task. Similarly, the microcontroller revolution has brought about its own version of Hello World: Blink. This program toggles an LED attached to a microcontroller on and off, usually at 1 Hz. As with Hello World, Blink shows an experienced user that their tools are working and (by simple example) can build confidence in a new student.

The development of ASPEN raised an important question: What should the first introduction to signal processing be for students? That is, the equivalent of Hello World. Several candidates were considered, from the complex (spectrum analyzers, guitar effect, etc.) to the simplest (audio loopback). After several iterations, and discussions with students who had minimal experience with signal processing, a proposed answer was arrived at.

The result was a user adjustable bandpass filter. Using three slide potentiometers, the user can set the gain of the system, the width of the bandpass region, and its center frequency. Because of the flexibility in setting the frequency limits of the filter, it can become a low- or high- pass filter as well.

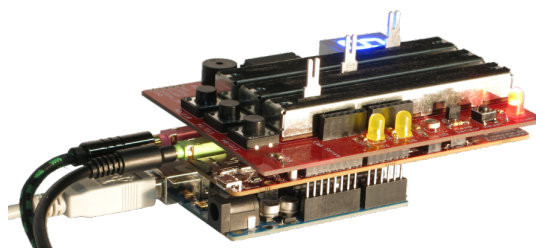


Figure 5.2: Partial ASPEN stack with Arduino, DSP Shield, and Danger Shield. The controls on the Danger Shield are used to adjust the parameters of the filter running on the DSP Shield.

Connecting a cell phone or MP3 player and a pair of headphones to the DSP Shield, the student can change the filter characteristics in real-time and hear the effects on audio. It can extract vocals or bassline and show how a low- or high- pass filter can make a song muddy or tinny. This gives the immediate and visceral exposure to the application and manipulation of filters, and their effect.

The system uses a Danger Shield (Spark-Fun Electronics, Boulder, CO) as a breakout board for three slide potentiometers, shown in Figure 5.2 stacked with a DSP Shield and Arduino Uno. The DSP Shield is running the Shield App and uses the included FIR filters loaded from the microSD card. The Arduino commands the Shield App to con-

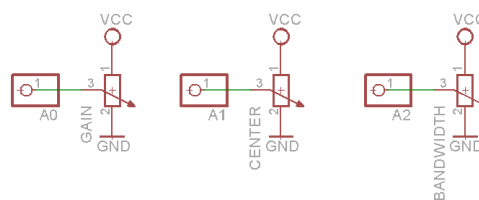


Figure 5.3: Slide potentiometer circuit diagram.

figure its filter chain according to input from the Danger Shield, allowing for changes to the filters in real-time. The utility of the Arduino here is in the familiarity of the device, allowing the most novice of users to not have to develop code for a new device (the DSP Shield). The Danger Shield was chosen to speed setup and make the user interface more robust and compelling, but the same demonstration could be achieved with a simple breadboard and standard rotary potentiometers. Figure 5.3 shows the simple schematic for connecting these potentiometers.

Ultimately, the concept of a real-time, user-adjustable bandpass filter may or may not be the conclusive answer to the question of Blink for DSP. Nonetheless, it provides a compelling example that addresses the question. It provides a short path to student success, a tangible experience for the student and a clear demonstration of the power of the ASPEN approach.

5.5 Student Projects Using the DSP Shield

A few groups of undergraduate students at Stanford have taken advantage of the DSP Shield to develop projects for coursework. One example was a neonatal care device as a part of a class project for EE153 at Stanford. This system was designed to demonstrate the possibility of remotely reproducing a mother's heartbeat (demonstrated to be beneficial to newborns) as a part of neonatal care [60]. The project was a collaboration between the author and Tushar Gupta, then an MSEE student. This system was designed to remotely recreate the pulse of a mother in real-time inside a stuffed animal which could be placed next to a newborn infant.

The DSP Shield and Analog Shield were used to acquire a PPG, a measure of the heartbeat, obtained optically. The analog signal-conditioning circuitry was simple, using only a dual op-amp as a transimpedance amplifier and part of a high-pass filter. The DSP Shield provided most of the signal conditioning, including automatic gain control and an 800-tap FIR filter for bandpass filtering. The resulting PPG signal was output to a high voltage piezoelectric driver, (DRV2667, Texas Instruments, Dallas TX) placed inside a child friendly stuffed toy, as shown in Figure 5.4. Because the flex of piezoelectric film transducer is not palpable on the surface of the toy at low-frequency, the PPG was input to AM and FM blocks. Using a 100 Hz carrier, the combined modulation scheme was chosen to maximize the tactile sensation generated

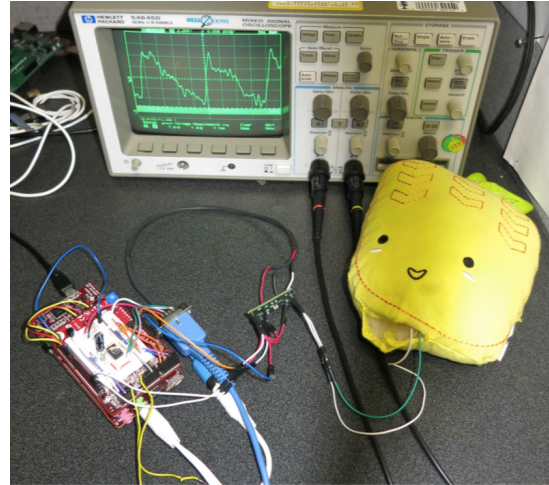


Figure 5.4: Device to allow for transmission of a mother's heartbeat to a newborn, using ASPEN components and a piezoelectric transducer to reproduce a heartbeat signal in a stuffed toy.

by the transducer to recreate the heartbeat signal. ASPEN allowed the signal processing portion of the project to be rapidly and simply developed. The entire project was completed with only two students working part time over the course of a few weeks.

Another example of undergraduate students using the DSP Shield for a project was an automatic headphone equalization system, by a group of EE122A students in fall 2015. The purpose of the system was to improve the quality of low-cost headphones and make their output more closely resemble that of more expensive headphones, which generally have flatter transfer functions. To do this, the students proposed to automatically generate an equalization filter to emphasize frequencies weakly reproduced by the low-end headphones and deemphasize the strongest frequencies.

The project used a DSP Shield, a white noise source circuit, microphones and speakers. The students developed a program that used the DSP Shield to measure the frequency characteristics of the noise output from the test headphones. This spectral data was used to generate a filter in Matlab with inverse frequency response characteristics to the headphones and the resulting filter run on the DSP Shield to improve the performance of the headphones. The DSP Shield enabled a group of three undergraduate students with no hands-on signal processing experience to take this project from concept to completion in less than five weeks.

5.6 Integration with EE264

The deployment of the DSP Shield in EE264 was motivated by a previously identified need to increase hands on, practical experience in coursework at Stanford. Prof. Ronald Shafer teamed up with Prof. Fernando Mujica to add a hands-on real-time signal processing component to his EE264 course. The theory portion of the class focuses fundamentals of signal processing - some of the major topics covered in the course are listed in Table 5.1.

Topic
Discrete-time (D-T) random signals
Sampling, reconstruction, D-T filtering, multi-rate systems.
Quantization in analog-to-digital conversion, and oversampling.
Properties of linear time invariant (LTI) systems.
Quantization effects in fixed-point implementations of filters.
Digital filter design.
Discrete Fourier Transform (DFT) and FFT.
Spectrum analysis using the DFT.
Parametric signal modeling.

Table 5.1: List of topics covered in lecture EE264 at Stanford, week by week.

Having been involved with the DSP Shield project during his time as Director of System Architectures Lab at Texas Instruments, Dr. Mujica was supportive of its application as the hardware component of the new lab portion of the course. The decision was informed by several factors, including student familiarity with the Arduino, simplicity of setup and deployment and portability. This allows for labs to be performed at home and brought to office hours with questions. In contrast, the previous DSP lab course at Stanford, EE265 was a dedicated 3-5 credit course where students spent tens of hours per week with a DSP evaluation module tethered to a desktop computer.

By incorporating a lab component into EE264, professors Schafer and Mujica sought to introduce students to the challenges of practical implementation of real-time signal processing algorithms, with as little impact to student workload as possible [61].

5.7 Comparison with EE265

In 2013, EE265 had been put on hiatus due to the retirement of its professor, Dr. Teresa H. Meng, leaving a gap in the hands-on component of the Stanford signal processing curriculum. EE265 focused heavily on the implementation of DSP algorithms on a DSP evaluation board as opposed to the more theoretical focus of EE264. The differences are clear in the structure of the labs. For example, students in the EE264 lab use Arduino style C++ to program their DSP while students in EE265 primarily wrote in assembly, with support routines in ANSI C. Furthermore, the grading rubric of EE264 generally does take code efficient into account, but EE265 had assembly cycle count targets for implementations of algorithms (e.g., convolution of a signal with a symmetric FIR filter) driving grading. Meeting these targets required that students manage tasks like loop unrolling and manually using processor specific dual-multiply and accumulate (“Dual-MAC”) instructions to parallelize individual calculations. In contrast, EE264 focused on high-level learnings, abstracting away optimization work that would arguably be better suited to an embedded systems course.

These optimizations are automatically handled by modern compilers (including C++ compiler for the DSP Shield). This means that DSP algorithm implementations generally no longer require such aggressive manual optimization, especially in the educational and prototyping space. While there is value in teaching such concepts and optimization, in a ten-week course such as EE264 or 265, they can serve as a distraction from the core material rather than an enhancement to learning.

The material of the EE265 labs covers several of the same topics as EE264 including quantization, frame processing, time- and frequency-domain filtering. The most noteworthy parallel is found in the courses final project. In both cases, students implement an Orthogonal Frequency Division Multiplexing (OFDM) communication system, described in more detail below. Ultimately, A-B educational comparison studies were outside the scope of this project, and would have been problematic given the retirement of EE265. The metric for success was the performance assessment of Prof's Schafer and Mujica given the reduced time commitment on the part of students when compared to the previous offering (one credit hour versus four)

5.8 Tools Developed In Support of EE264

In preparing to deploy the DSP Shield in EE264, there were several places where existing libraries for the ASPEN system did not yet meet the needs of the course. As a result, libraries and examples were developed or improved. The parts of these libraries not specific to the exact structure of EE264 have since been included integrally in Energia-DSP (yet another benefit of the open-source structure used in the project).

The most prominent tool developed for EE264 was the Matlab Serial library. Initially developed by Reggie Wilcox, a teaching assistant for the course. The library consists of two parts which together allow for simple, packetized communication between a PC and the DSP Shield. At first glance, this may seem a superfluous addition to the existing support for serial communication, but it fits well with the ASPEN ecosystem. It abstracts and automates memory allocation and reconstruction of transferred vector

data, as well as guaranteeing positive confirmation of receipt - an otherwise tedious task unrelated to signal processing. The library also provides a robust interface to define commands and pass arrays of data back and forth between Matlab and the DSP Shield. This simplifies transmission of control information and data with a PC. While a student could develop an alternative communication protocol using the DSP Shield, it would likely be less robust than the library developed and the wasted time would serve none of the educational purposes that the system is designed to address.

The library has a simple procedural interface [61], with data transferred via function calls:

- `serial_connect(baudRate)` establishes a serial connection at the specified baud rate.
- `serial_rcv_array(data, length)` receives a data array of length elements.
- `serial_send_array(data, length)` sends a data array of length elements.

The library also includes a command class to facilitate sending commands to be executed on the DSP Shield. With this library, a user can transfer a vector of data to be processed between a PC and the DSP with a total of three lines of code, including the library invocation. Figure 5.5 shows a sample program which receives a gain, offset and array. It returns a modified version of that array, in eleven functional lines of code.

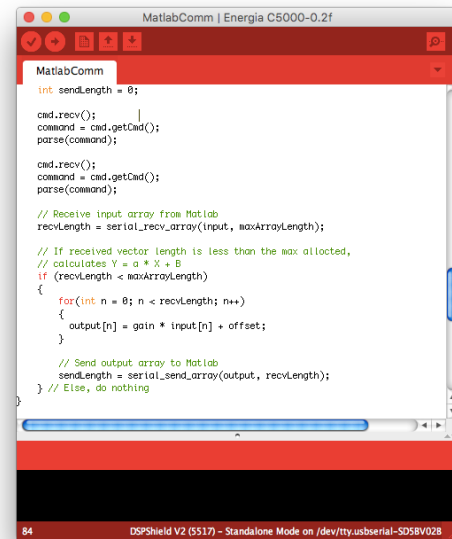


Figure 5.5: Example program for the DSP Shield using the serial communication library.

Having a robust communication path to a scientific toolbox like Matlab allows for algorithms to be and verified against a gold standard model developed in Matlab. It also allows for piecemeal development, with successive portions being transferred from the PC side of the link to the DSP as they are verified. The code for the library has been shared as a part of the ASPEN project and the Matlab portion can be easily ported for use with other scientific toolkits and languages like Python and SciPy, Java, or Lua.

5.9 Labs Zero and One, Introduction, Hardware, and Software Setup

In the ten week EE264 course, students complete six lab exercises, followed by a final project. Experiments are assigned weekly, until the final project which is allotted four weeks. These exercises have been chosen to correspond with materials taught in the theory portion of the class and build sequentially upon one another.

Labs Zero and One are assigned together. The Lab Zero portion walks students through setup of the DSP Shield and has them verify that they can upload and run some of the examples provided with Energia-DSP. Lab One introduces students to the features of the DSP Shield. They are asked to control the LEDs, send and receive data from the shield via serial link, read the DIP switches, use the Matlab Serial library and use the MicroSD card. These experiments comprise an introduction

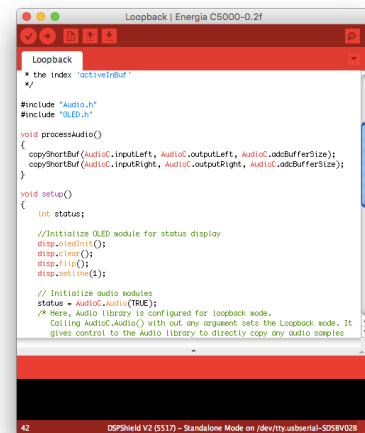


Figure 5.6: Audio loopback example showing the processAudio function.

to the hardware and its capabilities, to ensure that students are able to use the tools for the following labs.

5.10 Lab Two, Software Flow Control

Lab Two introduces students to flow control, interrupts, and ping-pong buffer management.

The DSP Shield audio library, discussed in Chapter Three automatically handles buffer management. In addition to the `setup()` and `loop()` functions, The user is provided a `processAudio()` function to fill in. As seen in Figure 5.6, this function has access to variables containing current ready data as well as an empty output buffer.

This lab covers the interrupt-based architecture of the DSP Shield audio library, introducing the DMA mechanism used by the DSP Shield audio CODEC to asynchronously fill a buffer in RAM on the DSP, without CPU involvement. When enough samples are collected, this triggers an interrupt event, calling the `processAudio()` function. Students see the tradeoffs in block-based processing as opposed to processing every sample as it arrives. Figure 5.7 [61] shows the tradeoff that students are introduced to with block based processing. While the logic of the application can be more complex than sample-by-sample processing, substantial context switching time is saved because the interrupt service routine is executed less often. The experiment also discusses ping-pong buffer management, used by the library to process incoming data while a second buffer is filled by hardware using DMA in the background.

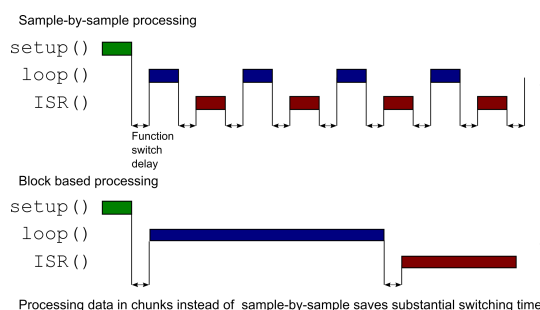


Figure 5.7: EE264 lab two introduces students to the benefits of processing block-by-block to save context switching time (after Mujica, 2015).

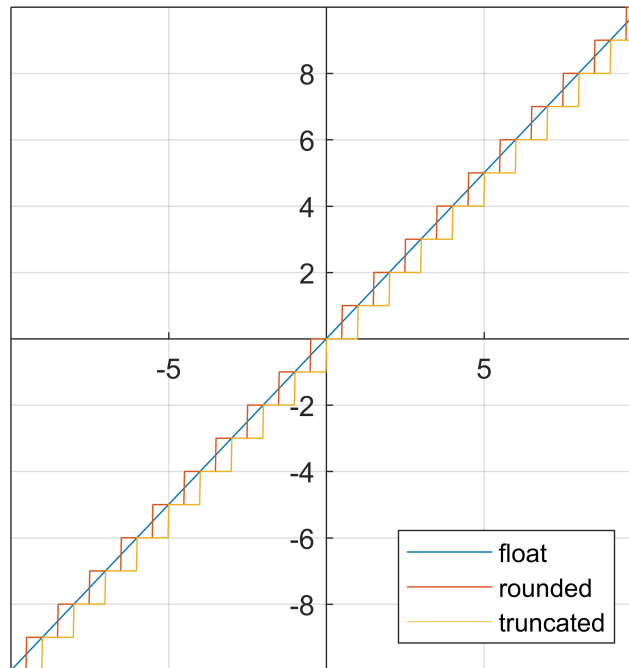


Figure 5.8: EE264 Lab Three introduces students to the effects of truncation and rounding. Shown here is a comparison of a floating-point value and its value when rounded and when truncated.

5.11 Lab Three, fixed-point Arithmetic

Lab Three focuses on fixed-point arithmetic. It is designed to complement lecture discussion of round off, quantization, and overflow. As such, it introduces students to the specific sixteen-bit architecture of the DSP Shield. Implementation issues that arise in fixed-point implementations are discussed. These include using twos complement arithmetic, rounding, quantization, saturation, and overflow handling are discussed, with specific reference made to relevant IEEE standards. Figure 5.8 shows a comparison of the effects of truncation and rounding of a number when converting to fixed-point.

5.12 Lab Four, Sample Rate Conversion

Lab Four teaches rate conversion, asking students to implement a simple karaoke system. Students up-convert audio data stored on the microSD card from 32 ks/s to 48 ks/s. They also used the audio CODEC to capture audio at 16 ks/s and up-convert it to 48 ks/s. These two streams, now at the same sample rate are mixed and output on the CODEC at 48 ks/s. Students are asked to configure the DSP Shield audio CODEC, develop their own FIR interpolation and decimation filters, and implement a rate-conversion function from scratch. Figure 5.9 shows a block diagram of the rate conversion system students are asked to implement.

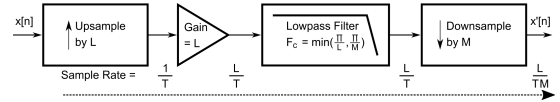


Figure 5.9: Block diagram illustrating sample rate conversion.

5.13 Lab Five, Hardware Accelerators

Lab Five uses the hardware FIR / IIR filters built-in to the AIC3204 audio CODEC on the DSP Shield. Students first demonstrate the ability to set up FIR and IIR filters on the CODEC. They then build an algorithm to adapt the filter coefficients on a block-by-block basis, coefficients provided by a pre-built PC based Matlab script. Figure 5.10 details the blocks of the audio signal chain on the DSP Shield, including the hardware accelerated (HWA) digital filters built into the audio codec. One filter is after the ADC and another is before the DAC.

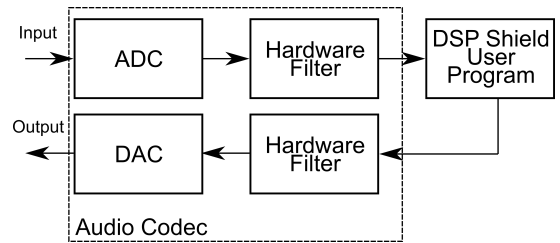


Figure 5.10: Overview of CODEC hardware accelerator (HWA) architecture used by students in Lab Five.

Students see how these accelerators can be used to implement additional filters with no computational cost to the DSP core itself.

5.14 Lab Six, FFT

In Lab Six students are introduced to the FFT support included with the DSP Shield. In this lab, students gain hands-on experience with FFT and inverse FFT functions. They experiment with the effects of frame size, frame edges and scaling. They implement periodic and linear convolution using the FFT and inverse FFT and perform basic spectral analysis. They are then introduced to optimized routines for real signals and implement circular convolution using the FFT. Using these tools, they are asked to perform basic spectral analysis of an input signal and compare their results with spectral analysis of the same signal performed in Matlab.

5.15 Final Project

In the first year of the EE264 lab, students finished the lab portion of the course by undertaking a project of their choice. A list of approved suggestions was included, as seen in Table 5.2. These projects focused on a broad array of topics, but all span topics covered in the course and stretch students knowledge. In addition to the predefined projects, students could submit their own proposals for projects subject to staff approval.

Two example projects were focused on audio. The first, was a soft-clipping fuzz-box by Woody Herman [62]. This project implemented a non-linear sigmoid function to emulate the soft-clipping effect of vacuum tubes as well as a custom filter to create a similar frequency response to a Vox ac30 guitar amplifier (Vox Amplification Ltd., Dartford, Kent, UK).

List of Final Project Topics Suggested to
EE264 Students in 2014

Old-phone ring tone decoder

OFDM digital communication receiver

Adaptive filtering demonstration

Phase vocoder (aka, a pitch shifter)

Heart rate detector

Programmable audio equalizer

Tree-structured sub-band coding of audio
(MP3)

Time/frequency analyzer (spectrogram)

Digital FM/AM communications

Acoustic noise suppression

Active echo cancellation

Table 5.2: List of topics covered in lecture EE264 at Stanford, week by week.

To address the creation of harmonics by this non-linear function, the signal was up-sampled by eight times before the sigmoid distortion function was applied. The signal was then filtered and down-sampled for output.

The impact of clipping with both a single-rate (non-up-sampled) hard clipping function and his soft-clipping up- and down-sampled sigmoid function, was measured using an input chirp function. This showed the harmonics introduced by the hard clipping function as opposed to the soft-clipping up-sampled function. He also implemented an 8th order IIR filter using the hardware IIR blocks in the audio CODEC. This filter was developed with Matlab tools to produce similar frequency and phase characteristics to the measured impulse response of the Vox ac30. The result was a successful high-fidelity single function guitar effects pedal which behaves like a classic vacuum tube device. This approach, however, could easily be modified to emulate other devices. At first glance, this project might seem straightforward, but it clearly demonstrates the power of the DSP Shield as a tool for understanding and manipulating the behavior of real-world signals.

Leveraging the publicly available Physionet database, students were able to experiment with a large body of pre-recorded ECG signals. In the project by Matthew Hu, a wavelet transform was developed based on available literature for ECG heart rate estimation [63]. The student implemented the transform using a filtering and down-sampling bank, first in Matlab with a floating-point implementation and then on the DSP Shield with a fixed-point implementation. The student successfully estimated the heart rate of streamed datasets from the PC and displayed it on the DSP Shield OLED display.

One project, by Terry Kong, implemented a phase vocoder system. Using the fast lifting wavelet transform (FLWT) algorithm to estimate pitch, and a time-domain, pitch synchronous overlap and add algorithm for pitch correction [64]. The FLWT algorithm required the student to implement a high- and low- pass block, and then down-sample the resulting signal to generate each successive set of wavelet coefficients. The resulting wavelet transform was analyzed for the maximum valued bin and used

to estimate pitch of the signal. This pitch estimate was used to control an overlap and add algorithm to correct the pitch to a target. A similar project by Hanchel Cheng and Sevy Harris [65] modified the tempo of a song using a beat estimation algorithm from E.D. Scheirer as the basis for a tempo modification algorithm [66]. Again, using an overlap and add methodology, the students changed the rate of songs, and demonstrated the efficacy of their algorithm using a metronome.

As a group, these projects demonstrate that students, following a few weeks of introduction to the DSP Shield were able to build complex and useful systems. Their diversity shows how the DSP Shield was useful for fairly wide range of tasks and produced impressive results in a limited amount of time.

5.16 OFDM Final Project

Since the second year of EE264 students have been encouraged to pursue a more structured final project, an Orthogonal Frequency Division Multiplexing (OFDM) decoder. This project is a more fleshed out version of a suggested topic from the first year. Prioritizing a single project provided logistical simplicity for the teaching and grading staff while offering a more consistent emphasis of topics covered, albeit at the expense of creativity.

In the OFDM algorithm, a signal is encoded by being split across several sub-carrier signals in a given band, spaced evenly such that they do not interfere with each other. The individual signals can be encoded by any one of a number of methods, although quadrature amplitude modulation (QAM) is typically used. The inverse FFT is then used to encode the signal and it is transmitted as a series of encoded frames. This signal is transmitted along with a frame-start block, which contains a known signal, providing synchronization and channel estimation [67].

Implementing an OFDM receiver requires students to develop several fairly challenging and complex sub-components. Figure 5.11 shows a block diagram of the major

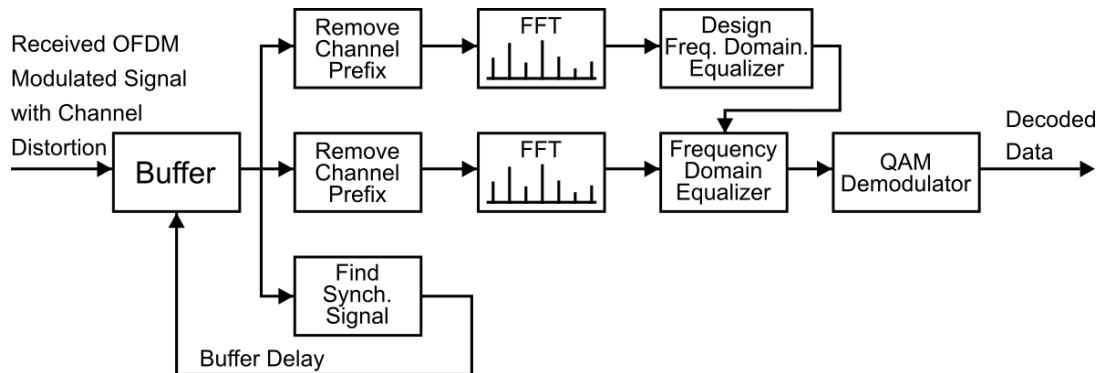


Figure 5.11: Block diagram of OFDM system implemented by EE264 students (after Mujica, 2015).

parts of the system in the EE264 OFDM project. Students were asked to develop a state machine, implement a channel prefix (CP) before each frame for alignment, and frame synchronization using the CP. This was followed by Fourier transform and frequency domain equalization of the channels using channel characterization information recovered from the transmitted data. Finally, the equalized frequency domain symbol data was passed to a QAM demodulation block to recover transmitted data.

The encoded data-stream was a text file sent as a baseband audio stream (20 Hz - 20 kHz) originated via headphone cable using the sound card of an attached PC. The stream was created by encoding a text file into WAV format using a provided Matlab OFDM encoder script.

Bonus points were awarded to students who implemented a channel estimation algorithm using the CP and successfully transmitted data from the PC to the DSP Shield using speakers and a standard electret microphone, as opposed to a clean wired audio channel.

The OFDM project raised the minimum expectation of student project quality in EE264 as compared to the first year the lab was offered. The success of most students in undertaking this project in a three-week window validates the utility of the DSP Shield for teaching DSP concepts in a course.

It is also striking how similar this project is to the final project from the EE265 curriculum. Both projects require students to implement a baseband OFDM decoder to process and return a block of text encoded a Matlab application running on the PC. Additionally, both courses require students to handle channel prefix, address and block processing issues, use QAM to encode transmitted symbols, and have an optional component for channel estimation and correction. The parallels between the two projects and success in teaching the same topic with a smaller time commitment using the DSP Shield demonstrates the power of the ASPEN approach to signal processing prototyping and education.

5.17 Conclusion

The combination of the introductory Shield App for novices and Energia-DSP for more experienced students helps demonstrate the utility of the DSP Shield as a tool for students of several skill levels. The use of the DSP Shield in Stanfords EE264 is of particular note. Profs. Schafer and Mujica have been successful in using the tools to bridge the gap between a theoretical and practical course with a reduced time commitment when compared to the previous EE265 course. Students in EE264 have been able to practically implement DSP algorithms in a real-time standalone system, including the complex OFDM algorithm, the same final project as required in EE265. Not only have students successfully covered the material of the course, but several teams have taken the opportunity to branch out and apply DSP tools to topics of their own interest, flexibility that was not feasible with the previous approach to teaching DSP. This suggests that the approach taken with the ASPEN platform can help students and teachers recapture time spent on low-learning value work.

Chapter 6

Biomedical Applications

6.1 Motivation

One of the major motivations for designing ASPEN platform was the need for tools to ease the development of biomedical instruments. A system that can easily acquire and manipulate biomedical signals in real-time serves two main purposes. First, it improves the tools available to teach biomedical instrumentation. Second, it increases the pool of people who can build systems. This allows students, scientists and others to safely acquire and manipulate biomedical signals without involving an engineering team, encouraging experimentation that might not otherwise happen. This is similar to the way the Arduino platform has been applied to general embedded systems tasks by scientists and engineers.

6.2 Background

As described in prior chapters, the Arduino platform is the most successful example of a revolutionary design shift that has made embedded platforms more accessible. A handful of tools for biomedical sensing development have been developed along these

lines. Among these are the Arduino derived Bitalino (Plux, Lisbon Portugal)[68] and the MySignals (Libelium, Zaragoza, Spain)[69] which uses the Raspberry Pi platform. They differ fundamentally from the ASPEN approach, however. These systems are built to simplify connecting pre-built sensors on breakout boards to a microcontroller. They are not optimized for real-time signal processing, nor do they facilitate sensor design. Additionally, these systems are expensive, priced from hundreds to thousands of dollars depending on model and configuration. The high prices limit their market appeal and excludes many students. While these platforms can make accessing biomedical signals simpler, the approach hides the details of signal acquisition circuitry inside a black box.

Using ASPEN for biomedical instrumentation serves a fundamentally different purpose. It facilitates the development of novel circuits for acquiring and manipulating these signals. The Analog Shield offers a high-ENOB and high-SNR. Additionally, the close coupling of the DSP Shield with this acquisition system enables control, filtering and analysis not feasible using off-board post-processing. This makes ASPEN well suited to developing new architectures for biomedical signal acquisition and to teaching the principles and pitfalls underlying biomedical signal acquisition. This is in contrast to the pre-existing Arduino & breakout board style approach.

Additionally, because the design of ASPEN is well suited for a broad range of signals, it is useful for a larger audience. This could enable lower costs (due to manufacturing volume) and more robust tools (due to community involvement) than previous systems for biosignals.

One might develop a similar tool working with new ARM-based embedded system prototyping boards, but current general signal-processing libraries are sparse on these platforms. Furthermore, none of these systems have been optimized for acquiring and manipulating biomedical signals - the intersection of these capabilities is new.

6.3 Serving Scientific and Teaching Needs

To demonstrate the utility of ASPEN both as a platform to simplify the construction of biomedical circuits, two projects were undertaken. The first was a single channel ECG circuit to demonstrate its utility as a tool for teaching biomedical circuit design. This system was compared to the reference circuit used for teaching at Stanford as well as a commercial off-the-shelf ECG system.

The second project demonstrated the utility of ASPEN for developing unique biomedical systems. A novel DC-coupled ECG-signal-acquisition system was developed which addresses the standing need for a fast-settling, high-gain ECG front-end. This design was made practical by the capabilities of the DSP and Analog Shields.

6.4 The ASPEN platform for DC-coupled ECG

Most current clinical-grade diagnostic ECG systems are AC-coupled with a high-pass filter at 0.05 Hz, as recommended by the Association for the Advancement of Medical Instrumentation (AAMI) [70], while ambulatory ECGs can have a high-pass of 0.5 Hz. These filters are the cause of several challenges in ECG systems. The time-constant of the AC-coupling filter leads to long signal-settling times, which can result in missed beats after large voltage excursions (e.g. lead-off events or patient motion). Choosing a higher cutoff frequency for the AC-coupled filter to alleviate this problem can lead to distortion of ECG waveform features, which can mask clinically relevant problems such as S-T segment elevation [71]. The advent of low-cost, high-performance ADCs which provide sufficient dynamic range make an alternative approach - DC-coupled ECG - viable. As discussed below, some recent, highly integrated front ends could be used as DC-coupled devices. Evidence of use in this manner is lacking however, due in part to the long development cycles of commercial ECG systems and the lack of visibility of problems with settling times, because clinicians are used to these limitations.

Additionally, the DC-coupled design of ASPEN makes the system useful for multiple types of biomedical signals, all of which have components at frequencies below the 20 Hz cutoff of traditional audio systems. For example, a PPG signal generally has dominant components in the range of 0.5 to 5 Hz [72], respiration is 0.05 to 1.5 Hz [73], and seismocardiograms have frequencies in the 0.6 to 100 Hz range [74].

The proposed architecture provides higher gain than previous DC-coupled ECGs, while digital control allows for much shorter settling times than an AC-coupled system. This can help address the longstanding need for fast settling ECG systems, especially for the emerging mobile health market [75].

6.5 ECG Signal Background

Understanding the design of an ECG system requires a high level grasp of the ECG signal and how it is acquired. The electrocardiographic signal arises from the repetitive, sequential electrical discharge of action potentials in several billion myocardial cells. This causes contraction of the chambers of the heart in a specific sequence, in order to circulate blood throughout the body. In the process of firing, each cell retransmits the contraction signal to its nearest neighbors. It then experiences a short refractory period following this transmission, preventing the signal from propagating backwards to cells already triggered in that cardiac cycle (under normal circumstances). The resulting ECG waveform consists of a series of defined complex shapes and intervals, reflecting the excitation of various segments of the heart. These parts of the ECG waveform, shown in Figure 6.1 arise from the physical structure of hearts conduction system and musculature.

Because the signal is physically created within a non-homogeneous volume conductor, it cannot be modeled using point-charge concepts familiar to most electrical engineers. Observations of the ECG signal made from differing placements of electrodes produces various projections of the underlying cardiac vector. This vector can be modeled as a voltage gradient that rotates in space over time. These different placements produce

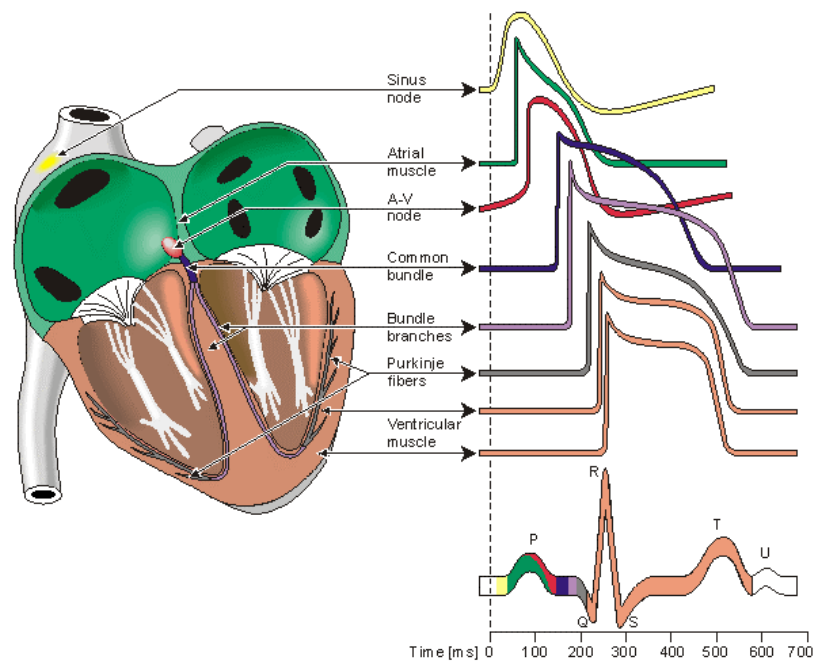


Figure 6.1: The ECG waveform arises from electrical impulses in cardiac cells. The externally-recorded signal is the far-field sum of signals from groups of cells depolarizing in an organized spatio-temporal sequence (used with permission from Malmuivo, 1995).

signals that emphasize different portions of the heart as illustrated in Figure 6.2 [76] and can aid the identification of cardiac conditions.

Observed at the surface of the torso, the signal ranges from 0.5-4 mV, and has frequency components that are generally in the 0.05 to 150 Hz range [70], arising from the combination of slowly varying components like the P and T segments as well as higher frequency components like the sharp QRS complex, seen in Figure 6.3 [77].

These factors inform the sensitivity and bandwidth requirements of any ECG acquisition system, as well as the filtering trade-offs that must be made in such designs. Non-clinical designs, such as simple heart rate monitors may employ much higher cutoffs for the high-pass filter and lower cutoffs for the low-pass. The AAMI specification for diagnostic ECG devices (EC11) from 0.05 Hz to 500 Hz, with limited attenuation (<30%) of test signals between 0.67 and 150 Hz [70].

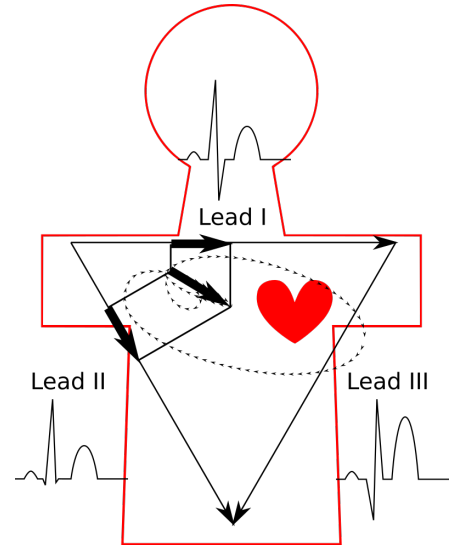


Figure 6.2: ECG vector evolution in time and how its projection appears on standard ECG leads I, II and III (after Malmuivo, 1995).

6.6 Non-cardiac Sources of ECG Signal Corruption

Typically overlaid on the ECG are several in or near-band corrupting signals. These include other physiologic signals arising from muscle movement and electrical activity in other organs. Pickup of these in-band signals is typically minimized through a combination of electrode placement and restriction of patient movement. There are non-biological sources of interference as well, including powerline noise and DC offset.

The first, 50 or 60 Hz powerline frequency is present in nearly all modern environments. There are several common approaches to mitigating this signal. It is fortunately generally uniform across a subjects body, appearing at all inputs nearly equally. It can therefore be

addressed by designing the ECG input circuitry with a high common-mode-rejection-ratio (CMRR). The most critical technique is the use of an instrumentation amplifier (IA) with a high CMRR as the first gain stage of the system, as well as careful matching of any differential passive input components. This results in dramatic attenuation of the common mode signal. An IA such as the LT1167 (Linear Technology, Milpitas, CA) used in all three circuits described here has a minimum 90 dB of CMRR [78]. Additional techniques can be used to improve this rejection ratio. A right-leg-drive system uses an op-amp to track the common mode voltage of an ECG lead, and uses another op-amp to drive the body itself to partially cancel the common mode signal.

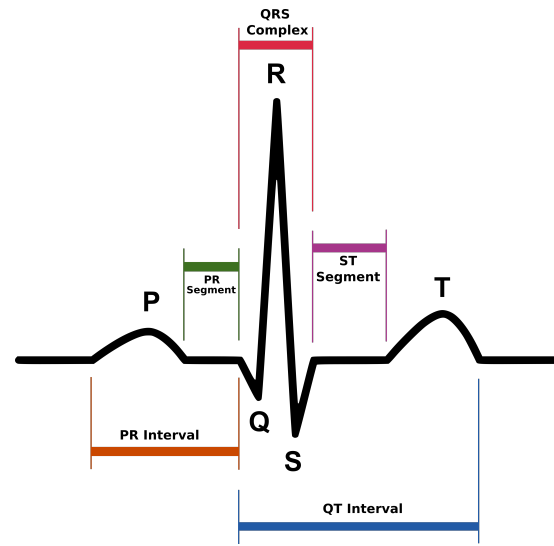


Figure 6.3: ECG waveform, highlighting major segments (used with permission from Agateller, 2007).

Additionally, a sharp 50/60 Hz notch or comb filter can be applied. Designed to remove the 50/60 Hz signal as well as the first few associated harmonics, such a filter can greatly attenuate the component of the common mode signal not attenuated by the CMRR of the system.

A second less predictable problem is DC offset induced the interface between electrodes and the body. As described in detail in [79], this complex interface can be modeled as the sum of several offsets, as seen in Equation 6.1.

$$V_p = E^0 + V_r + V_c + V_a \quad (6.1)$$

The first term, E^0 , is the half-cell potential formed by the interface between the conductive electrode and the electrolyte gel on the skin. This value is determined by the chemistry of the electrode material used. This voltage is defined by the Nernst potential, given in Equation 6.2, where a_1 and a_2 are the activities of the ions on each side of the membrane. For typical Ag/AgCl chemistries used in most of these experiments and typically in clinical disposable electrodes, this is 0.2 V.

$$E = E^0 - \frac{RT}{nF} \ln \frac{a_1}{a_2} \quad (6.2)$$

The next term V_r is the ohmic overpotential, defined by the resistance of the electrolyte. The third term, V_c is the concentration overpotential, due to the changes in ion concentration near the electrode/electrolyte interface. The final term, V_a is the activation overpotential is due to the activation energy of the chemical reaction at the membrane boundary. Furthermore, offset is affected by the electrode-body interface as shown in Figure 6.4. This results in a variable impedance affected by a myriad of factors including physical contact area, pressure, perspiration, and prior skin abrasion. Electrode-to-electrode differences in these factors can affect the offset voltage.

This offset will differ when observed across different electrode pairs (leads), due to differences in each of the aforementioned parameters due to skin conditions and chemical conditions at the interface. Additionally, the offset can shift over time as a patient physically moves, modifying the contact surface of the attached electrode - creating so-called motion artifacts. The overall offset can be as much as 300 mV [70], but in the past a paper chart recorder or cathode ray tube) reaches its maximum input voltage. As a result, most existing ECG systems include a high-pass filtering node at some point between the circuit electrode and the analog-to-digital converter to remove the DC offset.

6.7 Tradeoffs Between AC- and DC-Coupled Systems

Because the ECG signal arises from the electrical activity of the cardiac cells themselves, the precise shape of this waveform as well as the relationship between waveform cycles has great clinical value [80]. Therefore, preserving the shape and amplitude of the complex shape is valuable in a medical-grade ECG system.

To address the problem offset in ECG leads, the most common approach has historically been to incorporate a high pass filter in the signal chain [81]. The high-pass filter has several effects. The key function is the removal of the DC offset of the ECG signal, allowing for much higher gain in the circuit. The low cutoff frequency, however, can mean long settling times for any large change in signal offset. Additionally, as discussed above, the lack of true DC measurement of the signal can distort certain important markers of impending cardiac events [71]

If a high-pass filter is placed after the switching point (multiplexer) in a multi-channel system using a single ADC, it dramatically limits the speed at which the inputs to the signal chain may be switched. This is due to the settling time of the filter, which must settle to the mean voltage of a new lead. A worst-case step function in ECG DC offset could be as much as 600 mV lead-to-lead.

Using Equation 6.3 from [82], we can calculate the number of time constants required for an ADC to settle to within one-half least significant bit. For a 15-bit ENOB converter like the Analog Shield this means a delay of 11τ before acquisition.

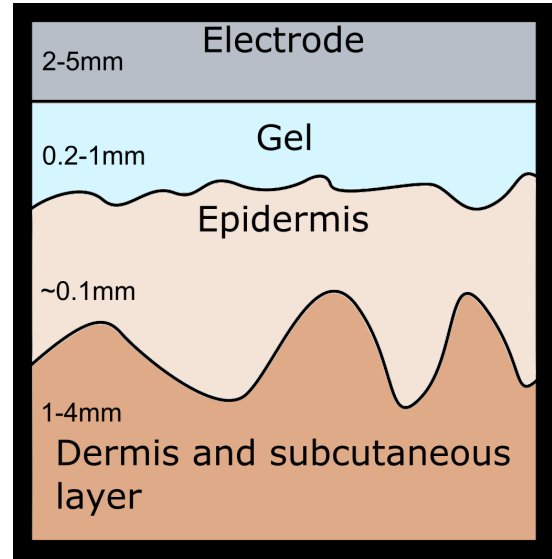


Figure 6.4: Layers of material affecting the impedance between the electrode and the body.

$$\#\tau = -\ln\left(\frac{\#least\ significant\ bits}{2^N}\right) \quad (6.3)$$

To resolve a 1 mV ECG signal with a clinical grade 0.05 Hz cutoff high pass filter, the RC time constant would be $\tau = 3.2\ s$. This means that such a high pass filter would impose a settling delay of more than 33 seconds after each channel switch. Such a long delay prevents the design of a practical system that uses a single front-end to poll several electrode combinations in rapid succession (e.g., polling several channels sequentially at a Nyquist rate of 300 s/s for the 150 Hz bandwidth required for an ECG system) to create quasi-simultaneous sampling. Instead, a system which records one lead at a time for a span of several minutes might be developed, but such a system would be cumbersome to use and would not allow multiple simultaneous projections of the cardiac vector to be recorded. Additionally, current clinical practice assumes

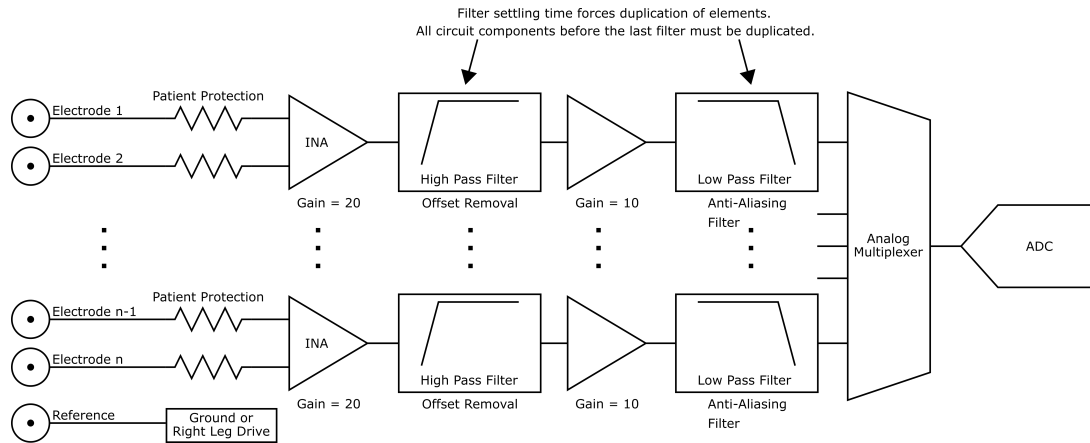


Figure 6.5: A typical AC-coupled ECG. Note the inclusion of a high-pass filter for each lead.

that simultaneous projections are recorded [80].

To address this limitation two approaches to building a multi-channel ECG can be taken. Most commonly, the signal chain has been duplicated for each lead. This includes at least one IA and high pass filter per channel, allowing each HPF to stay settled between samples. Additionally, the capacitors of these high-pass filters present a challenge to integration in a monolithic CMOS device. This duplication results in increased complexity and cost, however. Additionally, it can limit flexibility, as shown in the multi-channel ECG system diagram in Figure 6.5. In such a system, electrode combinations cannot be changed when the system is operating, although some designs have addressed this limitation by including an additional set of signal steering switches before the IAs. The problem of circuit duplication is more pronounced in electroencephalogram (EEG) systems, which can have hundreds of electrodes [83]. Commercial EEG front-ends such as the ADS1299 (Texas Instruments, Dallas, TX) have been designed using this approach, although a full EEG system may still require many of these ICs to acquire all channels.

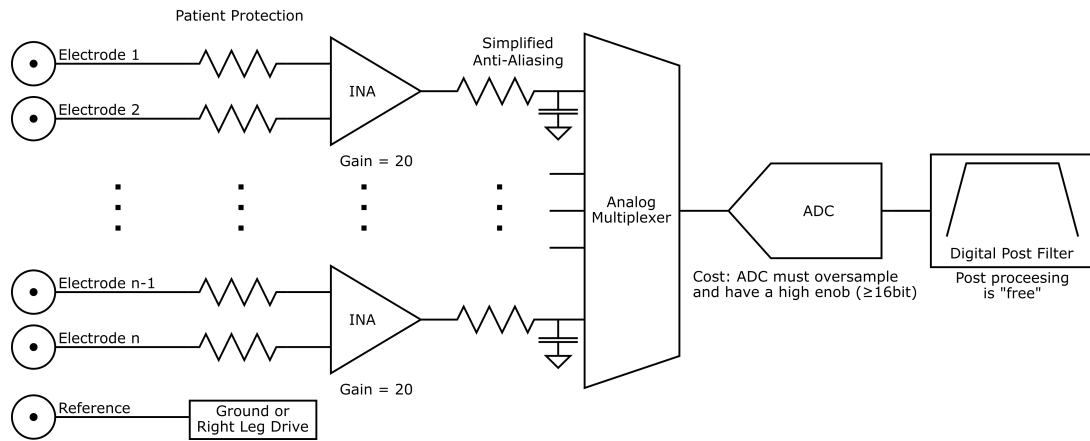


Figure 6.6: Basic DC-coupled multichannel ECG System. Analog system complexity is reduced at the cost of higher digital performance requirements. In the last few years, this has become a viable approach and has appeared in some integrated ECG front-end ICs, such as the ADS-129x (Texas Instruments, Dallas, TX) and ADAS-1000 series (Analog Devices, Norwood, MA).

An alternative approach is to remove the high-pass filter entirely. This dramatically reduces circuit complexity for multi-channel ECG systems. The cost to this choice is that DC offset of the signal is not removed. This offset is amplified by the system front end and is much larger than the signal of interest. Therefore, to prevent saturation of the amplifiers or ADC (with maximum voltage ranges dictated by the system power supply rails), the gain of the system must be much lower. For a system with $\pm 5\text{ V}$ of dynamic range, this is typically on the order of 10-20 instead of 200-500. Figure 6.6 shows a general example of a common DC-coupled circuit topology. The advent of low-cost, low-power ADCs, such as those of the sigma-delta type has made such designs commercially feasible.

As a result, the smaller AC component of a less amplified ECG signal is overlaid on the large amplified DC-offsets to the signal. Thus, the ADC in a DC-coupled system must have a large dynamic range [84]. Equation 6.4 shows that resolving an (unamplified) ECG waveform of 1 mV peak-peak with 100 steps (producing 10 μ V resolution), and have the ADC not saturate with 300 mV of offset, an ADC with at least 14.9-bit effective resolution must be used. In practice, signal processing techniques such as oversampling can be used to relax this requirement slightly.

$$\# \tau = -\log_2 \left(\frac{\text{maximum offset voltage}}{\text{minimum step voltage}} \right) \quad (6.4)$$

6.8 Prototyping the system using ASPEN

As discussed, the ASPEN platform is well suited to the task of ECG systems development. It has an appropriate ADC and DAC and the DSP Shield enables the rapid implementation of filters, rate converters, and control systems with only a handful of lines of code. All of this might be out of reach for a biomedical researcher without the powerful abstractions implemented here.

The task of prototyping an advanced DC-coupled ECG system was undertaken as a staged development process, with validation steps along the way. First, a low-parts count single channel system for education was developed, followed by a state-of-the-art multi-channel system and finally a novel, experimental high-gain multi-channel system with nonlinear filtering was created. This approach resulted in three reference designs of increasing complexity, demonstrating the utility of the ASPEN system as a tool for both teaching and for research. For these experiments, the DSP Shield was used in standalone mode, and was coupled with the Analog Shield as a front-end, power supply, and prototyping platform.

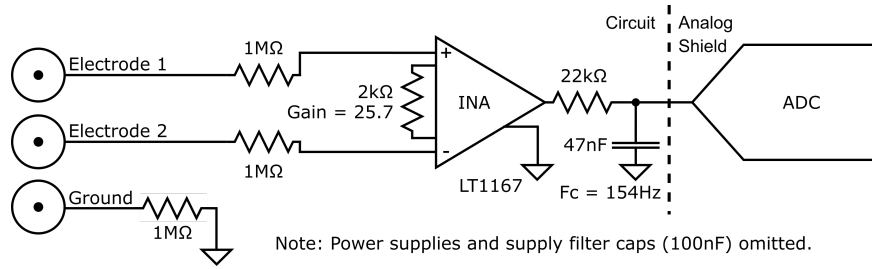


Figure 6.7: DC-coupled single channel ECG, demonstrating quick prototyping of a full ECG signal chain using minimal parts.

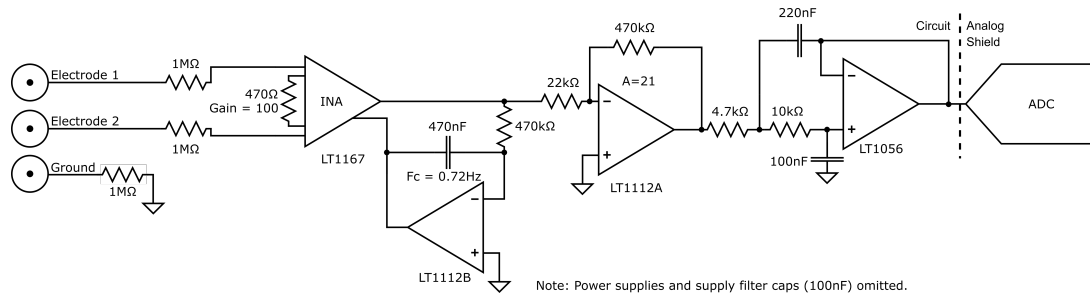


Figure 6.8: Typical single lead ECG circuit with offset cancellation.

6.9 Single Channel ASPEN DC-Coupled ECG

The first ECG system developed using ASPEN was an educational, low part-count single-channel DC-coupled ECG, shown in Figure 6.7. Estimating an optimistic maximum electrode offset of approximately ± 200 mV, a gain of 25.7 was selected. Because the R-peak (the largest amplitude component of the ECG waveform) is only a few millivolts, the DC offset can be considered as the primary component of the ECG waveform for dynamic range considerations. By choosing a gain of 25.7, the ADC will see approximately 5 V at its input with the maximum offset, meaning that an ECG signal can be digitized without saturating the DC-coupled front-end in all cases.

Using the Analog Shield as a signal acquisition system, the front end circuit is comprised of a single LT1167 INA (Linear Technology, Milpitas, CA) for amplification and an RC low-pass filter with a cutoff of approximately 154 Hz for anti-aliasing.

With a single active component, the circuit is a robust, simple to construct single lead ECG system. It provides similar performance to, but is more educationally useful and open than an off-the-shelf single channel ECG evaluation kit. The LT1167 has an input voltage range of $V_{supply-} + 2.1$ V to $V_{supply+} - 1.4$ V, and can be used with the ± 7.5 V supply of the Analog Shield for designs with a ± 5 V range.

For comparison, Figure 6.8 shows a circuit diagram for a single channel ECG circuit. This specific circuit has historically been introduced to students in an undergraduate introductory course on biomedical electronics (EE122B) at Stanford University as a canonical ECG circuit, typically taking students several hours to design, construct, and test. It uses an op-amp as part of an integrator circuit to provide DC-offset correction. This effectively AC-couples the circuit through feedback input to the reference pin of the LT1167, removing the DC component of the signal. As a result, this design still suffers from common mode-range limitation issues of other DC-coupled designs. The offset corrected signal is fed to a second gain stage then to a 2nd order Sallen-Key low-pass filter. While offering good performance for a settled signal and high gain, the circuit requires design and construction of four amplifier stages.

In contrast, the simplified ASPEN DC-coupled design moves most of the filtering into the digital domain. This decreases the amount of time required to prototype the system, while simultaneously allowing for implementation of a higher order filter. Before output via serial for display on a PC, the acquired signal is pre-processed on the DSP Shield with a 67-tap FIR low-pass filter with a cutoff of 135 Hz, shown in Figure 6.9. This filter was implemented using the DSP Shield SPL described in Chapter Three. Realization of the filter using the SPL requires only three lines of code: definition of coefficients, initialization, and a call to process a buffer.

By moving the filtering to the digital domain, not only can a steeper high-frequency cutoff be achieved, the filter design itself becomes more flexible, as students can experiment with the impact of different filter designs more easily than in a traditional system. Additionally, the reduced rejection of out of band signals resulting from the

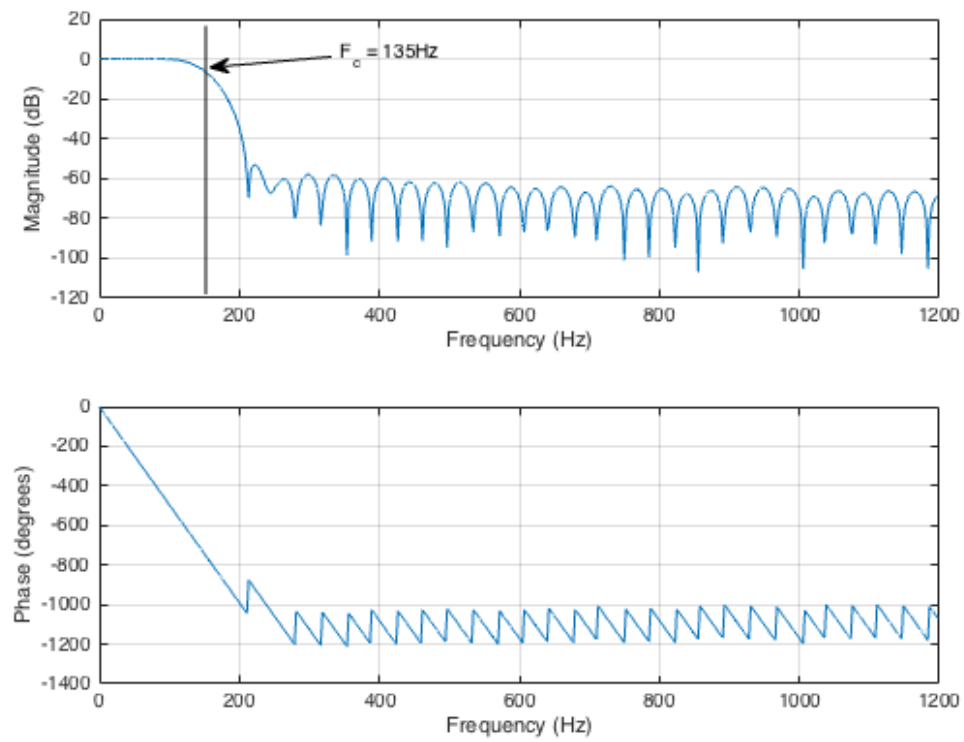


Figure 6.9: 67 tap FIR low-pass filter with an $F_c = 135\text{ Hz}$, used in DC-coupled ECG before downsampling.

use of a first-order anti-aliasing filter presents a teaching opportunity. Demonstrations of high-frequency out of band noise sources like an LED lamp with a switched-mode driver could be used to demonstrate how a digital post-processing filter cannot filter out aliased noise, which higher order anti-aliasing filters can reject.

6.10 DC-Coupled ECG Performance

To evaluate the performance of the ASPEN DC-coupled ECG circuit, it was compared to both the EE122B teaching circuit and an off-the-shelf AD8232 single-lead heart rate monitor evaluation board (SparkFun Electronics, Niwot, CO), using a commercial, off the shelf AD8232 IC (Analog Devices, Norwood, MA). This highly integrated IC is a representative component might be found in a high-end consumer or non-clinical medical device, such as a wearable cardiac monitor. Several aspects of each of these circuits were characterized, including input referred noise, SINAD, and CMRR. Additionally, for each device, a real ECG signal was also recorded.

Data acquisition from all circuits was performed using an Analog Shield and a DSP Shield. In the case of the reference circuits no additional filtering was applied and a sampling rate of 480 s/s was used with a 154 Hz anti-aliasing filter. For the ASPEN DC-coupled circuit (as well as later circuits developed), a sample rate of 2400 s/s was used, which was down-sampled to 480 s/s after applying the aforementioned 135 Hz FIR filter described above as a rate-conversion filter.

Figure 6.10 shows some key performance parameters of the ASPEN DC-coupled circuit. For comparison, Figure 6.11 shows the same parameters in the previously used teaching circuit and Figure 6.12 shows the reference commercial AD8232 system, in the form of a SparkFun Electronics Single Lead Heart Rate Monitor (SparkFun Electronics, Boulder, CO). The AD8232 is configured on this board with a bandwidth from 0.05 Hz to 45 Hz and a gain of 100. In all cases, the reference sinusoidal input was generated using an SR780 dynamic signal analyzer as a signal generator. The output was $4 V_{peak-peak}$ passed through a passive resistive divider network, with a

ratio of 1000:1, and connected to the ECG lead input terminals of the circuit (no electrodes were used). The SR780 was then set to put out a $3 V_{peak-peak}$, 2 Hz signal (except for the EE122B reference ECG with a $4 V_{peak-peak}$ signal).

The grounded, input referred peak-to-peak noise of the 1 lead ECG is $77 \mu\text{V}$, comparable to the $67 \mu\text{V}$ of the AD8232, but worse than $27 \mu\text{V}$ of the EE122B circuit. This higher performance is expected, due to the higher gain in the first stage, which (as expected, per the device datasheet) reduces the effective input referred noise of the LT1167 [78].

The ASPEN DC-Coupled ECG has an SNR of 39 dB. While this is inferior to the 53 dB SNR of the high gain ($A = 2200$) EE122B circuit, it is comparable to the 42 dB SNR of the AD8232 system, and more than adequate for resolving an ECG signal, as seen in the bottom left plot of Figure 6.10.

Figure 6.13 compares the CMRR performance of all three systems from 1 Hz to 100 kHz. Measurements were performed using an SR780 as a signal source. At each sample frequency a 1 V amplitude common mode signal was input to a pair of lead inputs. The CMRR performance of all three systems is similar. As expected, the EE122B circuit has slightly better performance due to high gain and the DC-Coupled circuit has similar performance to the AD8232.

In all three cases, the CMRR rolls off from DC to the cutoff of the various system low-pass filters. In the case of the ASPEN DC-coupled ECG, the step closely matches the filter frequency response seen in Figure 6.9. Above this cutoff frequency, the CMRR again rolls off as frequency increases. This is expected behavior as the filters of each system effect measured CMRR outside of their pass-bands. As with the SINAD and input-referred noise, the DC-coupled ECG performs well, with similar performance to the integrated AD8232 and somewhat degraded performance compared to the more complex EE122B circuit.

Additionally, as expected, the DC-coupled system provides a dramatically improved settling time to major input excursions when compared to the AD8232, as configured

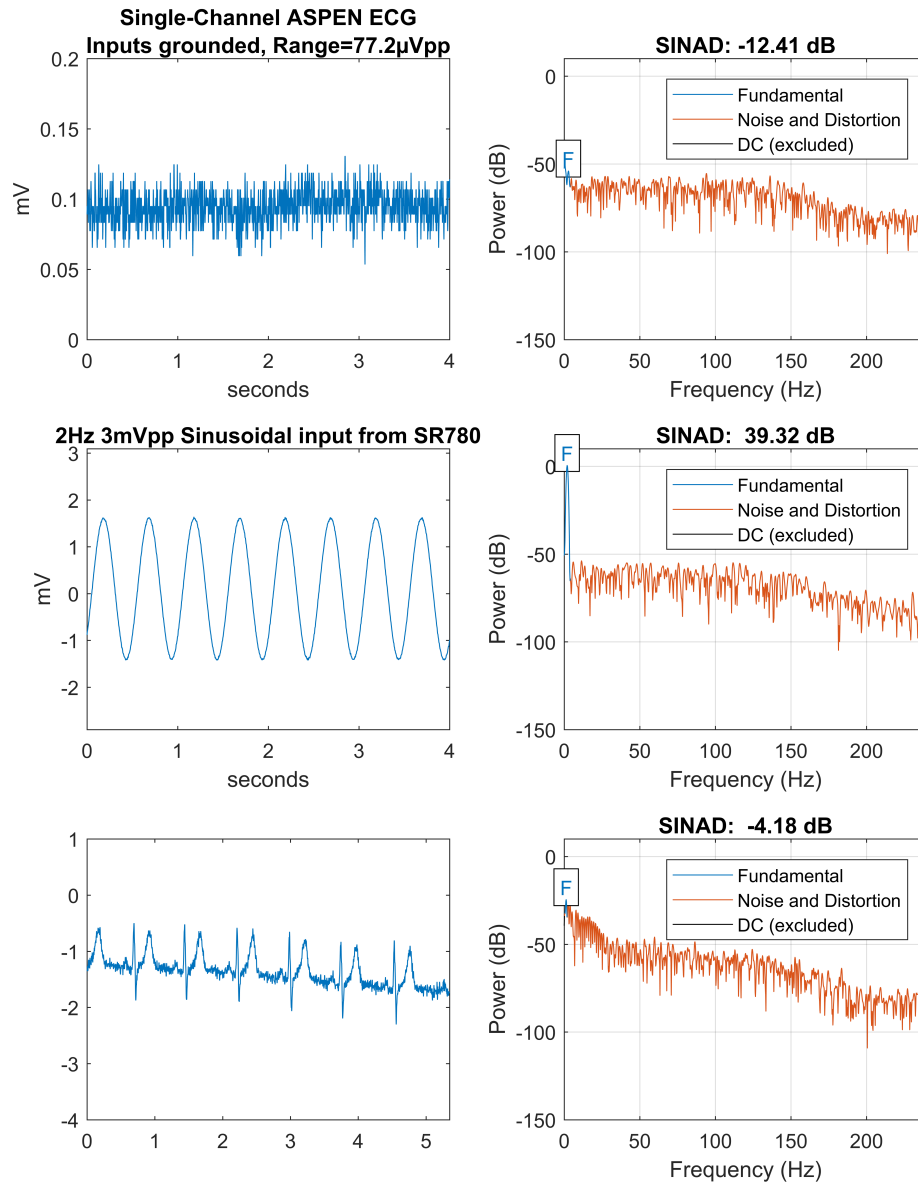


Figure 6.10: DC Coupled single ECG performance, showing $77 \mu V_{pk-pk}$ of grounded input-referred noise (top), An SNR of 39 dB with a 2 Hz, 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.

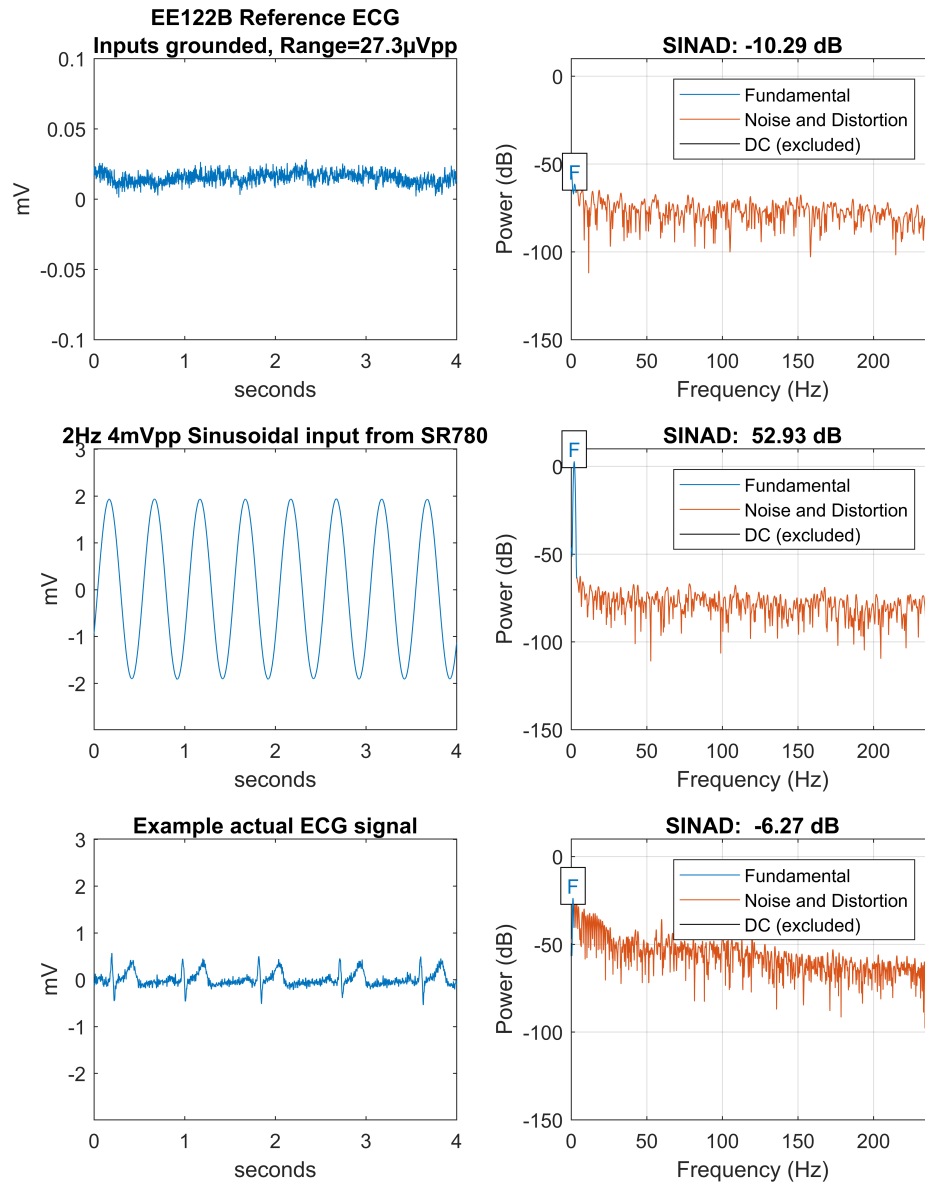


Figure 6.11: EE122B teaching ECG circuit performance, showing 27 μV_{pk-pk} of grounded input referred noise (top), A SNR of 53 dB with a 2 Hz 4 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.

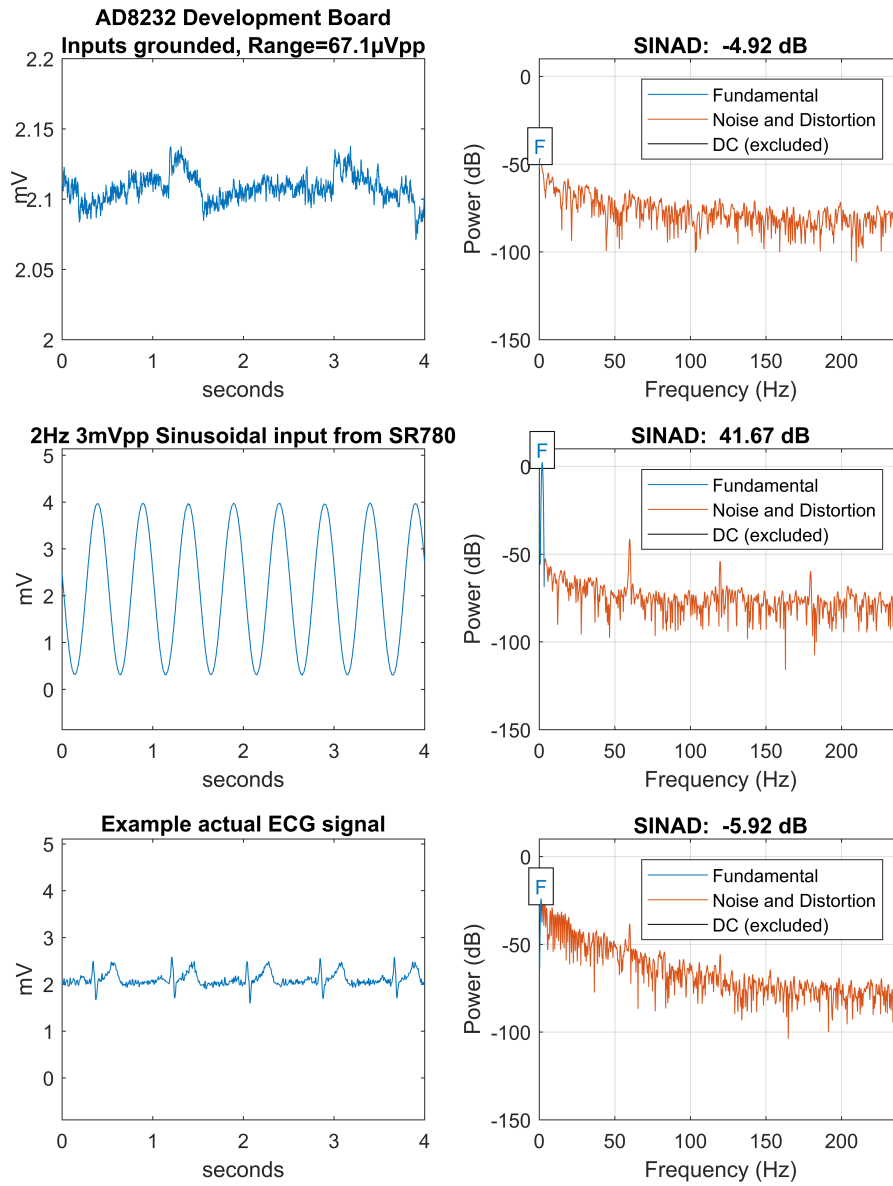


Figure 6.12: AD8232 ECG evaluation board, showing 67 μV_{pk-pk} of grounded input referred noise (top), An SNR of 42 dB with a 2 Hz 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.

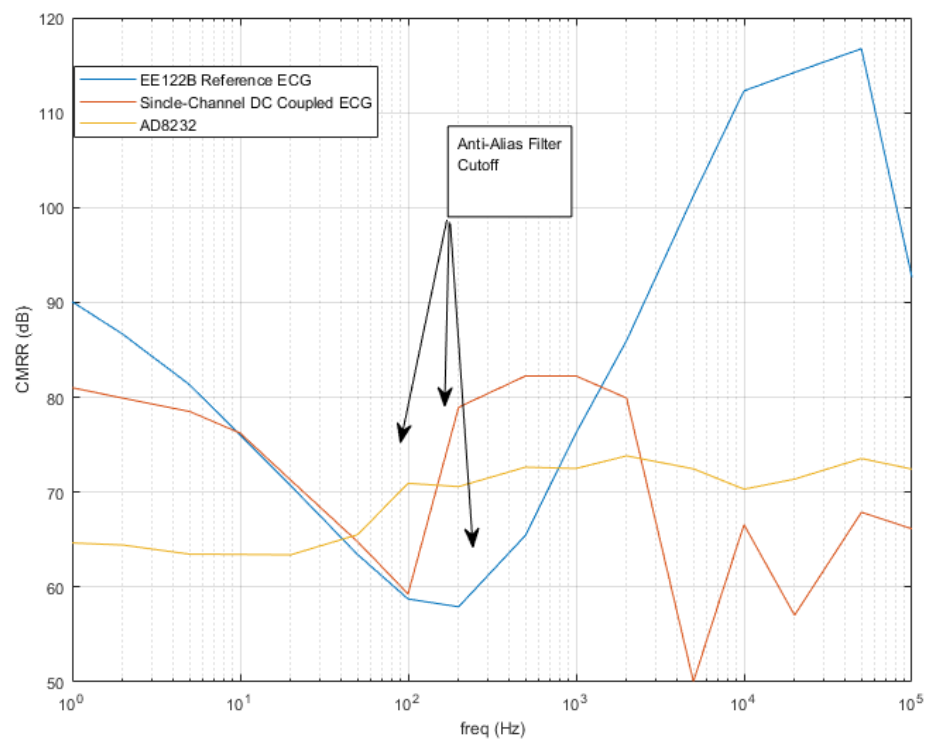


Figure 6.13: CMRR of each of the systems, plotted from 1 Hz to 100 kHz. Roll-off above 150 Hz is due to the filtering schemes implemented in all three devices.

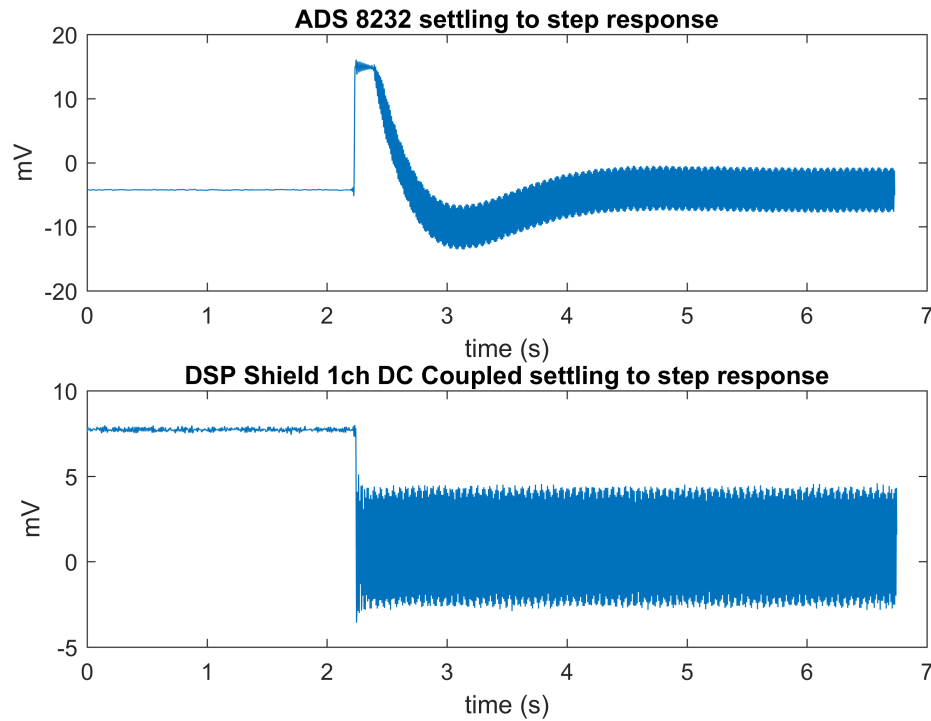


Figure 6.14: Comparison of the step response of an AD8232 and DC-coupled ECG to a 7 mV 100 Hz sinusoidal input. The AD8232 takes approximately 2.5 s to settle to within 5% of full scale of its final value while the DC-coupled ECG settles in milliseconds.

on the SparkFun evaluation board used here. Figure 6.14 shows the response of both the AD8232 and DC-coupled ECG to step inputs (power on of a 100 Hz sinusoidal signal). The DC-coupled system responds nearly instantaneously, while the AD8232 takes several seconds to settle. The EE122B circuit (not shown) performs similarly to the AD8232 circuit, taking several seconds to settle to a large step input.

Table 6.1 summarizes the performance of the three devices. Overall the performance of the DC-coupled ASPEN 1-channel ECG circuit is comparable to the AD8232 device. While the system has worse noise performance than the high-performance reference teaching circuit from EE122B, it represents a good trade in terms of design

Device	Input Referred Noise ($\mu V_{peak-peak}$)	SINAD (dB)	CMRR @ 5 Hz (dB)
AD8232	67	42	79
Old EE122B	27	53	81
New DC-Coupled	77	39	78

Table 6.1: Comparison of performance specifications of ECG devices.

and construction complexity for adequate signal quality. A single rail 3.3 V design, like the AD8232 would be limited to a gain of $\approx 5\times$ if DC-coupled due to the lower voltage supply rails, although such a design could easily be modified to reference the ADC to the appropriate supply voltage, mitigating the problem. Additionally, the techniques developed for the hybrid DC-AC circuit described later in this chapter address the challenge of offset in a DC-coupled ECG, and could be applied to a circuit with any supply voltage.

Additionally, the larger component count of the EE122B circuit offers many more opportunities for students to make errors during construction. It also makes correction of design errors (e.g., an improperly chosen filter cutoff) cumbersome as several components must be changed to adjust a filter. In contrast, the simplified single-channel circuit leaves less room for such errors and allows for quick correction of poor filter design choices in the digital domain. This facilitates the same real-world learnings in a shorter time.

Finally, the DC-coupled ECG has the advantage of fast step-input settling times, a capability that is more critical in circuit designs described later.

While the DSP Shield could be used to filter the outputs of either of these other circuits, the design is a positive demonstration that the system as a whole enables the construction of a simple ECG circuit with good performance.

6.11 Multi-channel DC Coupled ECG

The second design phase focused on a DC-coupled eight-input ECG circuit. The unusual architecture uses two 8:1 multiplexers connected in parallel to a set of eight electrodes. Multiplexing was moved as far forward in the circuit as possible to decrease system complexity as compared to conventional multi-channel ECG designs.

These multiplexers enable the system to be used to sample arbitrary electrode combinations in rapid succession. Eight software-defined combinations of electrodes are used to acquire various leads. This number is only a software limitation, the system could be configured to acquire any number of permutations of the eight electrodes with an adjusted sample rate. The leads are sampled sequentially, with the software configured to sample each of the eight leads at 2,400 samples/second for a total system sample rate of 19,200 samples/second.

A fully DC-coupled architecture for a multi-channel ECG has several benefits. Naturally, the lack of a high-pass filter reduces design complexity and component count. More importantly, without the long time constant induced by the AC-coupling capacitor in the high-pass filter there is no substantial settling delay when the electrodes connected to the IA are switched. This means that a single fast IA may be used for the entire system, as opposed to the need for one device per channel in an AC coupled system. Additionally, because the electrodes are DC-coupled, mismatched capacitors do not cause phase mismatch between channels, which can present a problem in high fidelity AC coupled systems.

Alleviating the need for multiple IAs in a multi-channel system is beneficial because a quality IA generally costs - typically several dollars [85]. Modern ECG front end ICs, such as the ADS1298 (Texas Instruments, Dallas, TX) [86] and the ADAS-1000 series (Analog Devices, Norwood, MA) [87], however, still use multiple IAs integrated onto the chip. The ADS1258 (Texas Instruments, Dallas, TX) [88] includes multiplexers before the IA and is an example of a recent device that could be used to implement a similar architecture for an ECG, although it does not appear that a multi-channelled,

DC-coupled, single-IA topology has been applied to an ECG acquisition system to date, with all device developers having chosen the more conservative approach of duplicating large segments of the signal chain.

In this new topology shown in Figure 6.15, the only components not multiplexed are the first order passive anti-aliasing filters. This is in contrast to the architecture seen in Figure 6.6, which uses one IA per channel.

6.12 Filter Matching in a Multi-Channel System

In the case of the ASPEN multi-channel ECG, the input user protection resistors, usually on the order of 1-5 M Ω are used as part of a simple 1st order RC anti-aliasing filter to reduce component count. In the experimental circuit, 1 M Ω resistors and 1 nF capacitors were used, creating a filter with a nominal 159 Hz cutoff, as seen to the left of the analog multiplexer in Figure 6.15. By dedicating a simple 1st order RC filter to each lead, but not dedicating a single amplifier per lead, an important tradeoff is made. While amplifiers are de-duplicated, the number of capacitors used in the anti-aliasing filters is doubled - and matching of these capacitors becomes a concern for maintaining circuit CMRR, as any component mismatch turns common mode signals into differential signals. This tradeoff presents a challenge, as matching can be problematic, especially when prototyping. In a mass-produced system the problem might be addressed in several ways, including use of laser trimmed components, trimmer components, or by integrating the components onto an IC version of this circuit. The largest challenge in integrating this design on an IC is the 1 nF capacitor in the anti-aliasing filter on each channel. While some design tricks might be applied, these capacitors are generally too large to be expected to be included on an integrated device.

To obtain tightly matched capacitors, a group of twenty nine $\pm 1\%$ capacitors was tested and from those, the most closely matched set of eight was chosen, resulting in $\pm 0.36\%$ matching, with the closest pair at better than 0.2%. The input protection

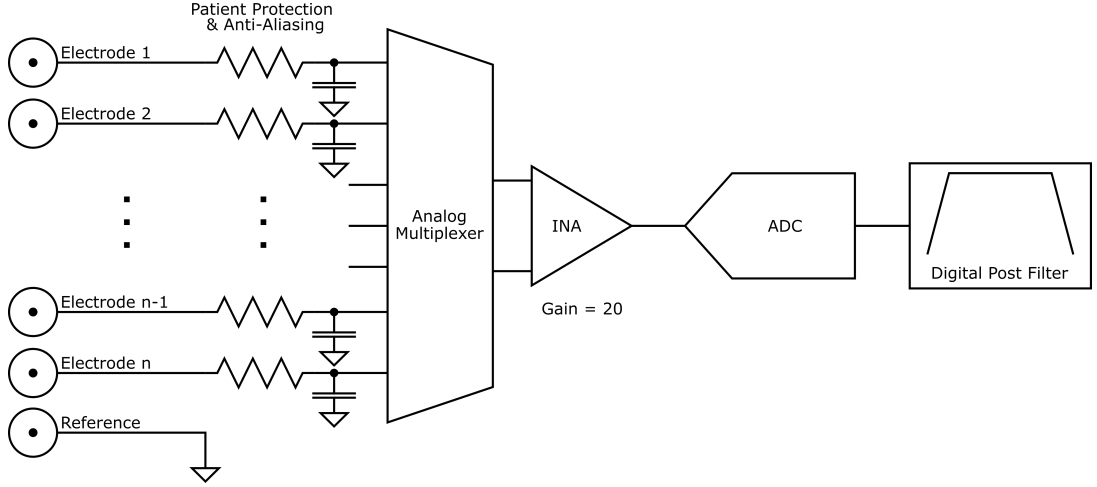


Figure 6.15: Multi-channel ECG system as implemented on the DSP Shield. Note how the topology allows IAs to be de-duplicated, as compared to the multi-channel system in Figures 6.5 and 6.6.

resistors were also similarly binned. A set of $35 \pm 1\%$ resistors resulted in $\pm 0.09\%$ matching. The binning process minimized the impact of mismatch on the system, but it is clearly still a concern. Resistors matched to $\pm 0.1\%$ and capacitors matched to $\pm 1\%$ are readily obtainable from commercial sources.

Given the filter Equation 6.5, we can estimate the worst-case mismatch between any differential pair of RC input filters. This will be the difference between the filter cutoffs using the worst-case value for each component. Using Equation 6.5 $\pm 0.1\%$ resistors and $\pm 1\%$ capacitors result in an $f_c = 2.86$ Hz.

$$f_c = \frac{1}{2\pi RC} \quad (6.5)$$

$$f_{c\Delta} = \frac{1}{2\pi R_{max}C_{min}} - \frac{1}{2\pi R_{min}C_{max}} \quad (6.6)$$

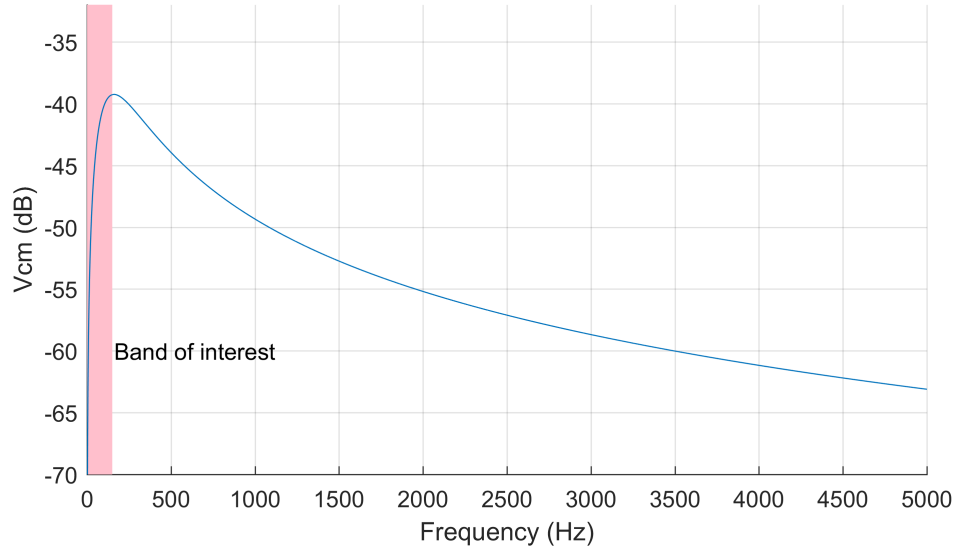


Figure 6.16: The CMRR of a mismatched differential RC-filter versus frequency for 0.1% matched resistors and 1% matched capacitors.

Mismatch turns common mode signals into differential ones and negatively affects the CMRR of the system. Equations 6.7 - 6.9 show how a common mode signal is converted to a differential signal via component mismatch.

$$Z_{Cmm\%} = 1 + \frac{2(\text{Capacitor Mismatch Range in } \%) }{100} \quad (6.7)$$

$$Z_{Rmm\%} = 1 + \frac{2(\text{Resistor Mismatch Range in } \%) }{100} \quad (6.8)$$

$$V_{diff_mismatch} = V_{cm} \left(\frac{Z_C Z_{Cmm\%}}{Z_R + Z_C Z_{Cmm\%}} - \frac{Z_C}{Z_R Z_{Rmm\%} + Z_C} \right) \quad (6.9)$$

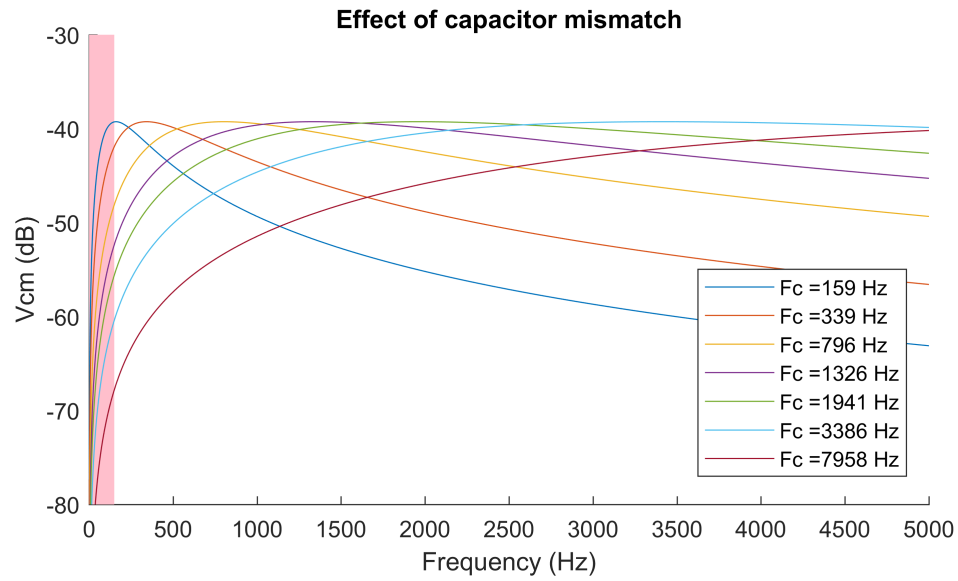


Figure 6.17: Sweep of cutoff frequency of mismatched RC filters with 0.1% resistor and 1% capacitor matching. Highlighted in red is the 150 Hz band of interest for ECG.

The mismatch of the R's and C's can be understood as a mismatched pair of frequency dependent voltage dividers. Observing the differential output of a combination of the most mismatched possible components (i.e. resistors maximally mismatched in one direction and capacitors in the other), we see the worst case filter CMRR. Because both halves of the differential filter attenuate frequencies well above the cutoff frequency and do not attenuate frequencies below the cutoff, the impact of this mismatch peaks there. Figure 6.16 shows the differential gain of a common mode signal for a worst case mismatch of easily obtained 0.1% resistors and 1% capacitors in a nominally 159 Hz low-pass differential RC-filter.

The mismatch results in a minimum 41 dB CMRR for the differential filter, around 159 Hz. At lower frequencies, the impact of mismatch is substantially less.

By changing the cutoff of the RC-filter, the frequency of the worst case CMRR can be moved. Figure 6.17 shows a family of 0.1% matched R, 1% matched C anti-aliasing

filters of various cutoff frequencies. The CMRR of these filters is plotted from DC to 5 kHz. Using a higher sample rate would allow the cutoff of the filter to be moved higher while still sufficiently preventing aliasing of the signal at the ADC. This would move the peak mismatch outside the band of interest of the system, improving overall CMRR at the cost of attenuation of aliased frequencies. Such a change would help improve rejection of 60 Hz line noise, which is the dominant common mode signal in most ECGs. There are other ways to mitigate the effect of line noise and its harmonics, like right-leg drive, digital notch or comb post-filtering. The tradeoff between these two parameters can be balanced depending on the needs of a specific design. In the case of the ASPEN ECGs, choosing a low cutoff frequency of 159 Hz provided sufficient common mode rejection and anti-aliasing filtering.

In the case of the ASPEN multi-channel ECGs, the analog switches also have the potential to negatively effect CMRR. For this reason, both high performance ADG508A (Analog Devices, Norwood, MA) and low cost CD4051B switches (Texas Instruments, Dallas, TX) were evaluated. The final version of the circuits evaluated use the ADG508A switches, but low cost CD4051B switches performed well in circumstances where CMRR is not critical.

For the experimental system, with hand-picked resistors and capacitors, even better performance was measured. Figure 6.18 shows a comparison of the CMRR of the DC-coupled multichannel system with the reference AD8232 system. As before, an SR780 was used to generate the test signal, with a $0.5 V_{peak}$ amplitude. As expected, CMRR rolls off as frequency increases, until the anti-aliasing filter cutoff frequency is reached. Nevertheless, performance is adequate for an experimental circuit, and could be improved via matching of the anti-aliasing filters using laser trimming of input resistors or adjustment potentiometers.

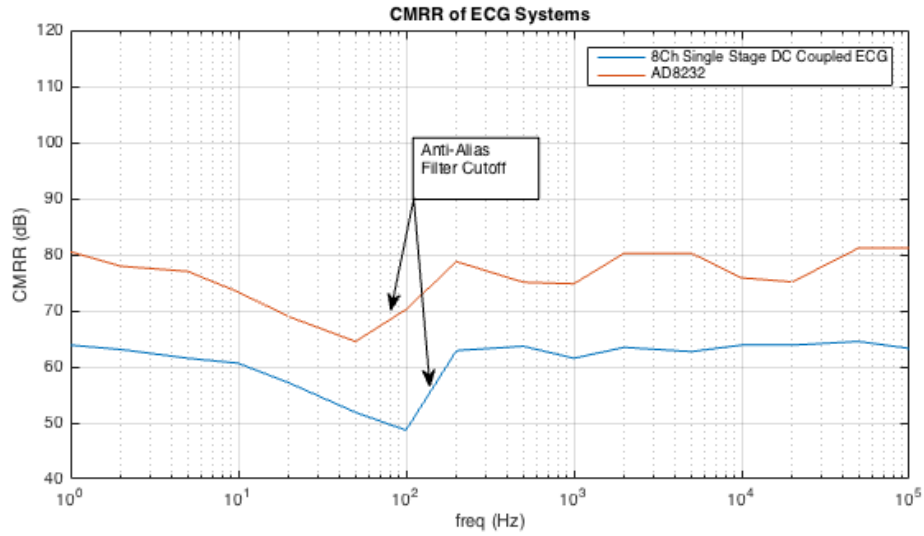


Figure 6.18: Plot of CMRR vs. frequency of the eight-channel ECG compared to the single channel commercial system.

6.13 Impact of Spectral Folding in the ASPEN Multi-Channel ECG

As discussed earlier, each of the eight combinations of electrodes is sampled at 2,400 s/s in a sequential fashion. The relatively low sample rate and first order anti-aliasing filter results in spectral folding of out-of-band frequencies. Figure 6.19 shows a sinusoidal chirp signal from 100 Hz to 9 kHz. The signal source was an HP8165A function generator (Hewlett Packard, Palo Alto, CA). As designed, below 150 Hz the signal is not attenuated. At 150 Hz, the roll-off

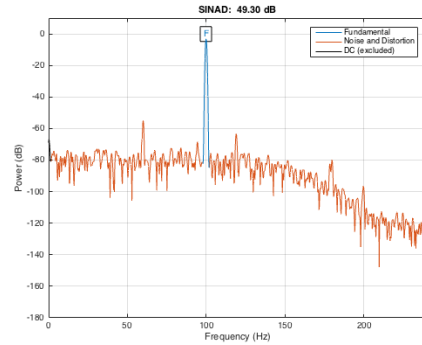


Figure 6.20: PSD of 100 Hz test signal.

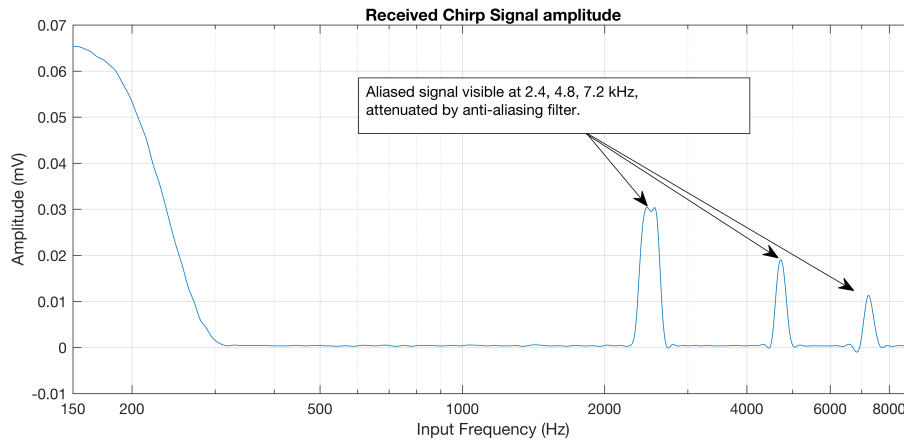


Figure 6.19: Plot of response to sinusoidal chirp signal fed into ECG system. The chirp was a 60 mV amplitude signal swept from 100 Hz to 9 kHz. Signal aliasing is visible in the 2.4, 4.8 and 7.2 kHz bands. The system digital filter causes rapid roll-off after 150 Hz.

of the 150 Hz FIR low-pass filter takes effect. Due to aliasing, the chirp reappears in ≈ 300 Hz wide bands centered around 2.4, 4.8 and 7.2 kHz. The first order RC anti-aliasing filter causes the observed roll-off of these aliased frequencies.

While aliasing can be problematic, the simple RC filter is expected to be sufficient in most normal conditions. The spectrum of an input signal was observed from DC to $F_{Nyquist} = 240$ Hz. The result is shown in Figure 6.20. The test signal was a 100 Hz, 5 mV sinusoidal input from an SR1 Audio Analyzer (Stanford Research Systems, Sunnyvale, CA). A digital low-pass filter for down-sampling at 150 Hz was applied as well. The result shows a SINAD of 49.3 dB. While there are interference sources at 60, 120, and 180 Hz, these are due to AC line noise (and harmonics) which can be digitally filtered. A production system could be easily modified to sample each channel at a much higher rate, further reducing the impact of spectral folding. For an experimental system, however, ASPEN is well suited to the task even with a modest sample rate.

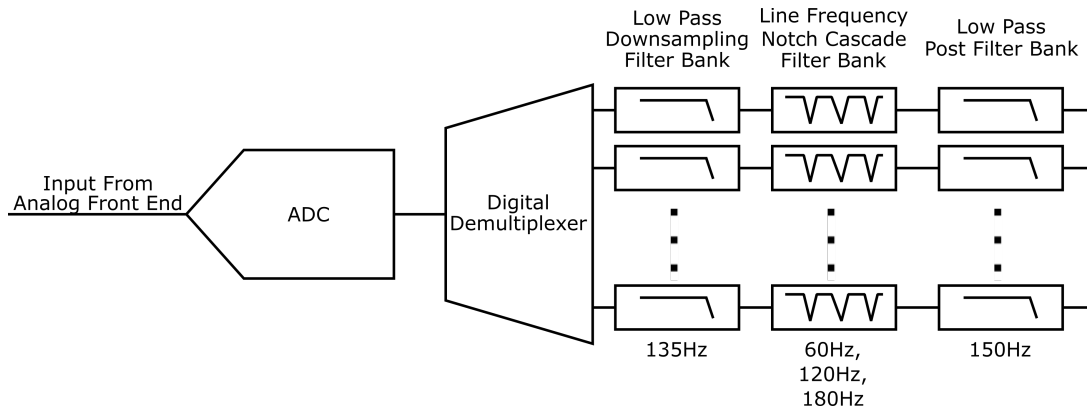


Figure 6.21: Digital filter signal chain for each synthesized ECG lead.

6.14 Digital Filters Implemented in the ASPEN Multi-Channel ECG

In addition to controlling the multiplexed sampling system, the DSP Shield was used to implement a bank of filters. This chain of low pass and notch filters was implemented digitally for each channel, shown in Figure 6.21.

As with the ASPEN DC-Coupled single channel design, each channel is down-sampled by the DSP after capture, by $5\times$ to 480 s/s. The final version of the system contains an FIR type down-sampling filter with 67 taps, and a cutoff of 135 Hz. An FIR filter was chosen for its linear phase characteristic, to minimize distortion of the captured ECG waveform. Since the computational complexity of the filtering is dependent on the number of samples processed, minimizing the length of the down-sampling filter allows for greater complexity in the later filtering stages.

The line frequency filter is composed a series of Butterworth 2nd order sections. It includes a 4th order 60 Hz notch, a 2nd order 120 Hz notch and a 2nd order 180 Hz notch filter, as shown in Figure 6.21. This is followed by an additional 150 Hz cutoff, 53-tap FIR low-pass filter. Applying an additional filter after the down-sampling filter is more computationally efficient than applying a higher order down-sampling filter

```

File Edit Sketch Tools Help
ECG_ms_iirshunt_FIRpost150_settlestop filters.h supportTools.h vectorTools.h
int* sampleVector = sampleBufferRaw[outChannel][sampleBufferReadIndex[outChannel]]; //set up pointer we use for io
int sampleVectorLength = SAMPLE_BUFFER_SIZE; //length of buffer
if(downsampleOn) //if we are downsampling
{
    FIRFDS[outChannel].filter(sampleVector, sampleVector, sampleVectorLength); //LFF for downsampling
    downsample(sampleVector, sampleBufferDown, M, sampleVectorLength); //downsample
    sampleVector = sampleBufferDown; //sampleVector now points at downsampled parameters
    sampleVectorLength /= M; //note that we shortened sample vector by the downsample ratio. (5)
}
if(meanRemOn && !settled) //if mean removal algorithm is on
{
    int meanVal = computeMean(frameSums[outChannel], frameSumsIndex[outChannel], sampleVector, sampleVectorLength);
    if(blankCount[currentChannel] == 0) //if fast slewing (set in interrupt because ADC is railing) ran recently, me
    {
        //if mean removal is on
    }
}

```

261 DSPShield V2 (5517) - Standalone Mode on COM11

Figure 6.22: single-line function call for applying one filter to an ECG channel, highlighted in yellow.

at the same cutoff frequency. Secondly, this filter serves to remove quantization noise introduced by the high-q IIR notch filter stage, which manifests as small oscillations near the Nyquist frequency.

As with the single channel ASPEN DC-coupled ECG, the filter chain takes advantage of the DSP signal processing library described in Chapter Three, allowing each filter to be initialized with one function call, and each block processed with one additional function call, as shown in Figure 6.22. This function call applies the filter with its state variables (FIRFDS[outChannel]) to the block of data being currently processed (sampleVector). To filter the received signals, the program passes the appropriate data to the correct filter block, and the result is stored in place.

Device	Input Referred Noise ($\mu V_{peak-peak}$)	SINAD (dB)	CMRR @ 5 Hz (dB)
AD8232	67	42	79
New DC-Coupled	77	39	78
Eight-Channel	84	34	62

Table 6.2: Comparison of performance specifications of ECG devices.

6.15 Multi-Channel ECG Performance

In addition to CMRR, the ASPEN multi-channel DC-coupled ECG system was tested for input-referred noise, SINAD and an ECG signal was obtained, as seen in Figure 6.23. As with the previous tests, an SR780 was used as a sinusoidal signal source. Expectedly, the circuit performs similarly to the ASPEN single channel design with similar gain using the same INA. The system has $84 \mu V$ peak-to-peak input-referred noise. This means that the analog switches (ADG508A, Analog Devices, Norwood, MA) and switching action do not dramatically degrade the noise performance of the system. The SINAD is somewhat degraded at 34 dB, but this is similar to the measured performance of the single channel system (39 dB), as well as the AD8232 (41 dB). Qualitatively, the observed ECG signal is clear, with visible P, R, and T waves (although due to electrode placement, the morphology in this example is slightly different).

Table 6.2 summarizes the performance of the eight channel DC-coupled system and compares it to the single channel device as well as the commercial AD8232 system. Ultimately, the eight channel DC-coupled system performs reasonably well and is comparable to the AD8232 while providing additional channels.

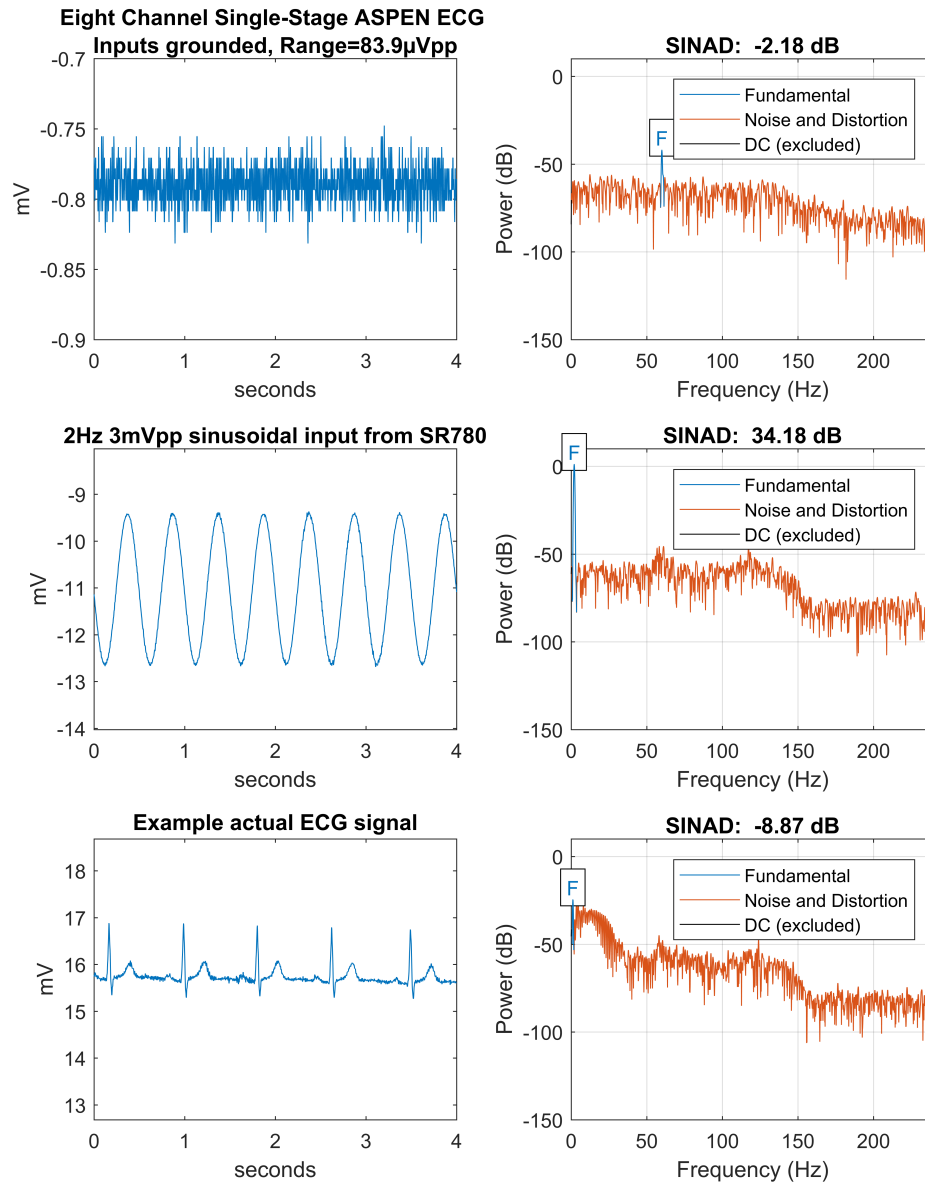


Figure 6.23: Multichannel ECG, showing 84 μ V of grounded input-referred noise (top), An SNR of 34 dB with a 2 Hz 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.

6.16 DC Offset Removal in a DC-Coupled ECG

As discussed above, one of the disadvantages of an AC-coupled system is the long settling time of the (typically 0.05 Hz) high-pass filter. This can be especially problematic when the signal goes far out of range (due to a disconnected lead, or patient motion) and takes several time constants (and consequently tens of seconds) to settle.

One mitigation strategy used with traditional AC-coupled ECGs, is to electrically short the AC coupling capacitors using switches [89], to reset the offset quickly without waiting for the filter to return to normal. For DC-coupled ECGs, there is not any long time-constant RC that needs to be reset, although this comes at the cost of gain (and thus, signal fidelity). To address the offsets and wander captured by a DC-coupled ECG circuit, a digital high pass filter can be applied in post processing. In the case of these digital filters, the problem of long settling time for out of range signals crops up again. As with hardware high pass filters, with a digital filter the buffer can be reset when the signal is determined to be out of range, producing the same effect as shorting an AC-coupling capacitor [89]. Applying a filter in post processing addresses many of the problems inherent in a DC-coupled ECG design, including baseline wander and large captured signal offset. It does not, however, address the limitation on front end gain due to offsets.

The ASPEN system presents an opportunity to address this limitation in a novel way, using the DSP to move the AC coupling filter to the digital domain. To achieve this, the circuit shown in Figure 6.24 includes a second gain stage between the instrumentation amplifier and ADC. This stage is configured as an inverting summing amplifier, one of the inputs to which is an output from the Analog Shield DAC. This allows the summing amplifier to be used as part of a mixed-domain feedback loop to create a hybrid non-linear high-pass filter.

Using the DAC on the Analog Shield to drive the mean correction signal to back into the circuit at this node, the DSP shield can cancel the offset of the ECG signal of any lead combination, independent of the offset in any other channel. Unlike a traditional

AC-coupling node, the DAC voltage (controlled by a digital low-pass filter) can be changed in microseconds. This approach means that multiple high pass filters can be implemented in software and thus for an arbitrary number of channels with no additional hardware, limited only by system sample rate. The offset removal means that additional gain (included in the summing stage) can be added to the DC-coupled system before signal acquisition, enabling overall system gain comparable to other AC-coupled systems, while preserving the other advantages of DC-coupled systems such as reduced circuit complexity and settling time when switching between channels. Additionally, as discussed below, it enables nonlinear offset correction, something not feasible in the analog domain without a DSP.

6.17 Implementing DC-Offset Removal in Software

Before developing the full system, a post-processing only system was built for testing the mean removal algorithm. The moving average of each input signal over three seconds was tracked. This mean was then subtracted from the measured signal before the data was output for display, effectively implementing a 0.05 Hz cutoff high-pass filter, similar to diagnostic-grade instruments. The algorithm included hysteresis, disabling the filter when the mean of the output signal was within a dead band around zero. Figure 6.25 shows a set of test sinusoids from 2 to 100 Hz. Each sinusoid was created digitally and input as if they were sampled from the ADC. All have a DC offset of approximately 2.5 V which the algorithm removes and settles around 0 V.

Implementing the summing amplifier and DAC drive, several gain settings for the second stage were tested. Figure 6.26 shows a screenshot of the real-time plotting application with a system gain of $A = 1000$. In this plot, all ECG signals have a mean near zero and exhibit no clipping, despite the fact that the ECG is DC-coupled and the acquired signals show approximately 15,000 counts, or about the full dynamic range of the Analog Shield ADC. Figure 6.27 shows a lower gain $A = 200$ ECG test in more detail.

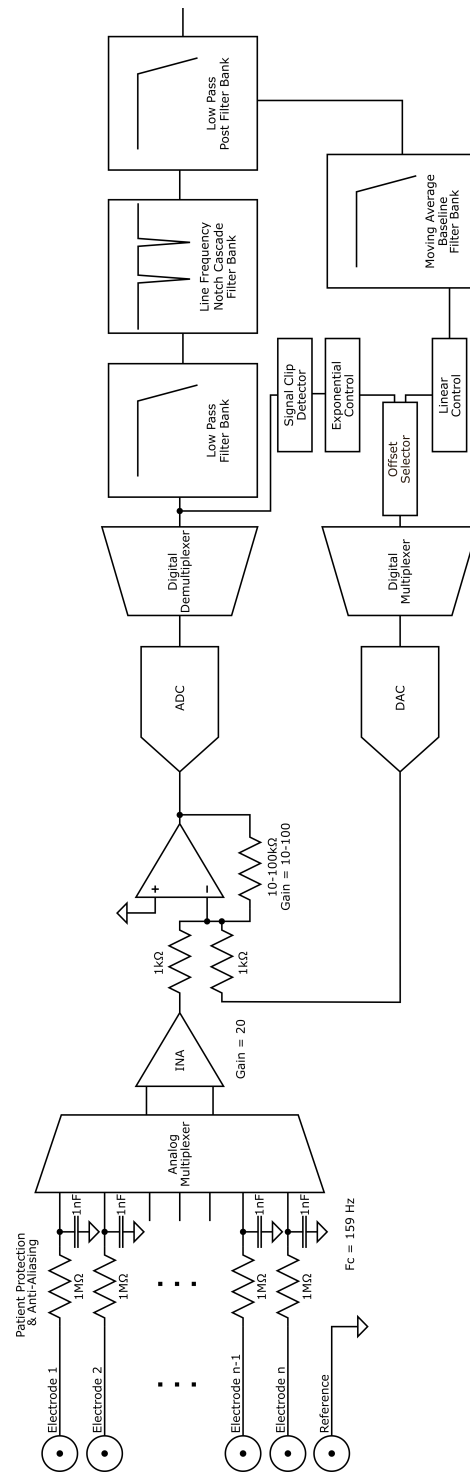


Figure 6.24: Block diagram of multi-channel ECG with individual channel mean removal feedback. A summing amplifier is used to provide additional gain and include nonlinear offset correction.

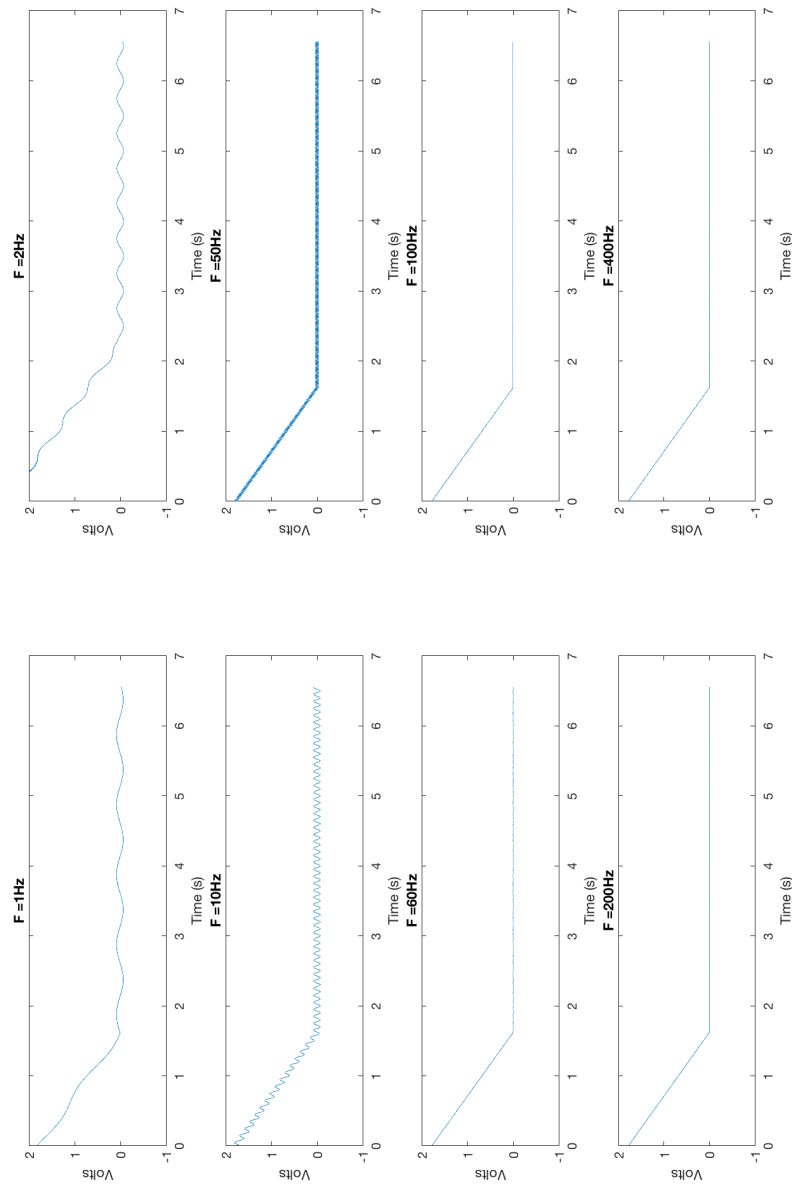


Figure 6.25: Example of test offset sinusoids being removed by digital post-processing removal algorithm.

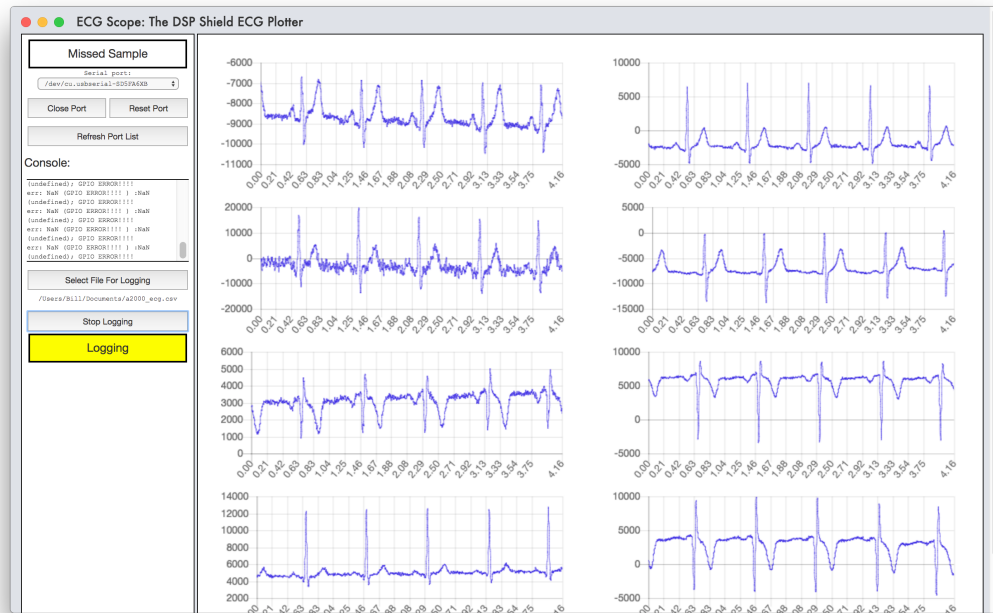


Figure 6.26: Screenshot of eight-channel DC-coupled ECG real-time plotting software with DC-offset correction and gain of 1000, showing standard ECG leads I, II, III, and V3.

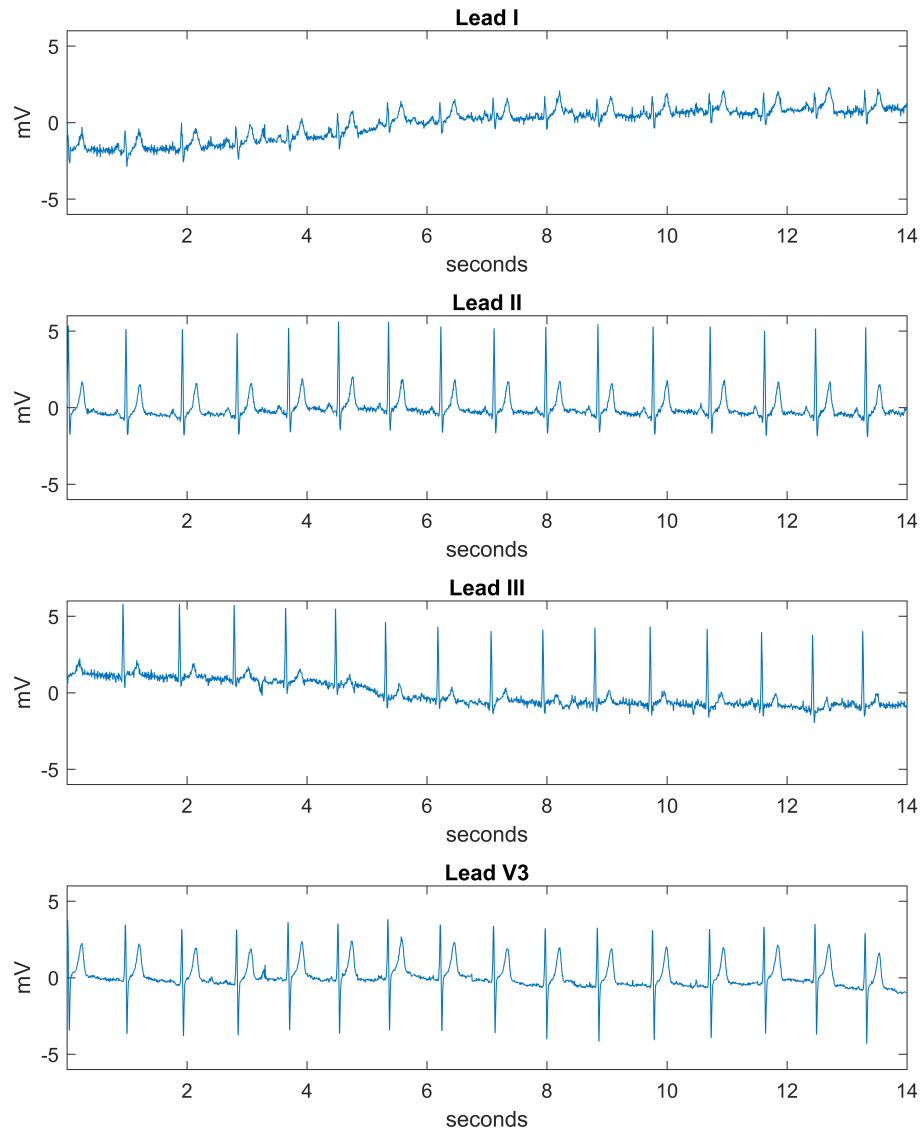


Figure 6.27: Plot of 4 channels, taken from the system with the gain set to 200. From top to bottom leads I, -II, III, V3.

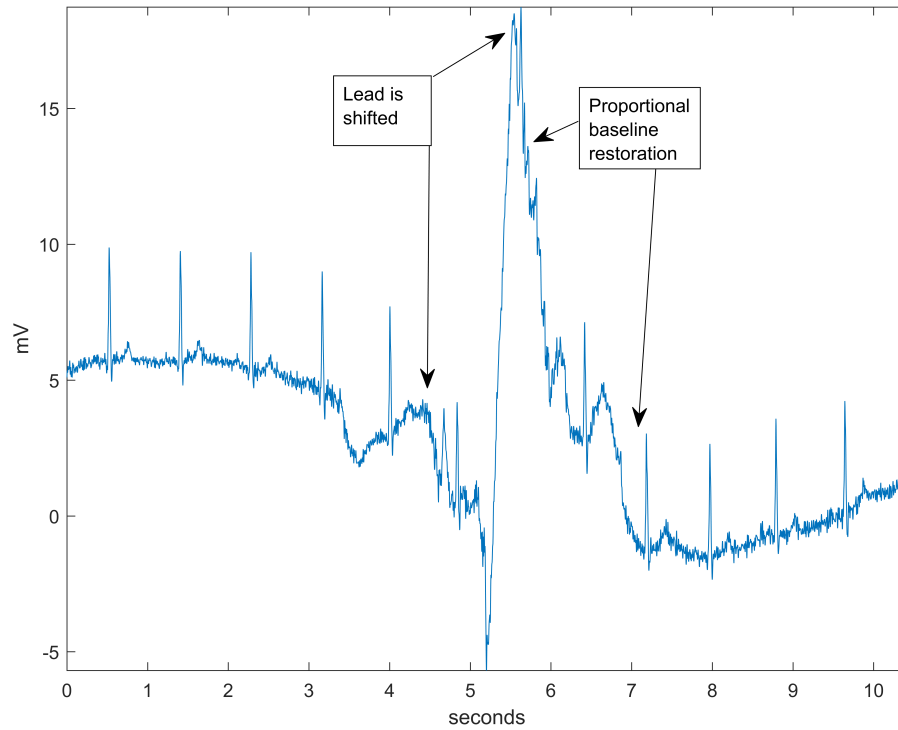


Figure 6.28: ECG lead electrode motion offset with linear recovery.

6.18 Nonlinear Correction

The most unique benefit of adding offset feedback is that it enables control loops to be applied to the signal offset. Figure 6.28 shows an ECG signal with an induced motion artifact. As described above, the filter is disabled as long as the signal remains within a defined dead band. When the signal wanders outside this voltage range a proportional control loop is enabled, driving the ECG signal to zero using the Analog Shield DAC.

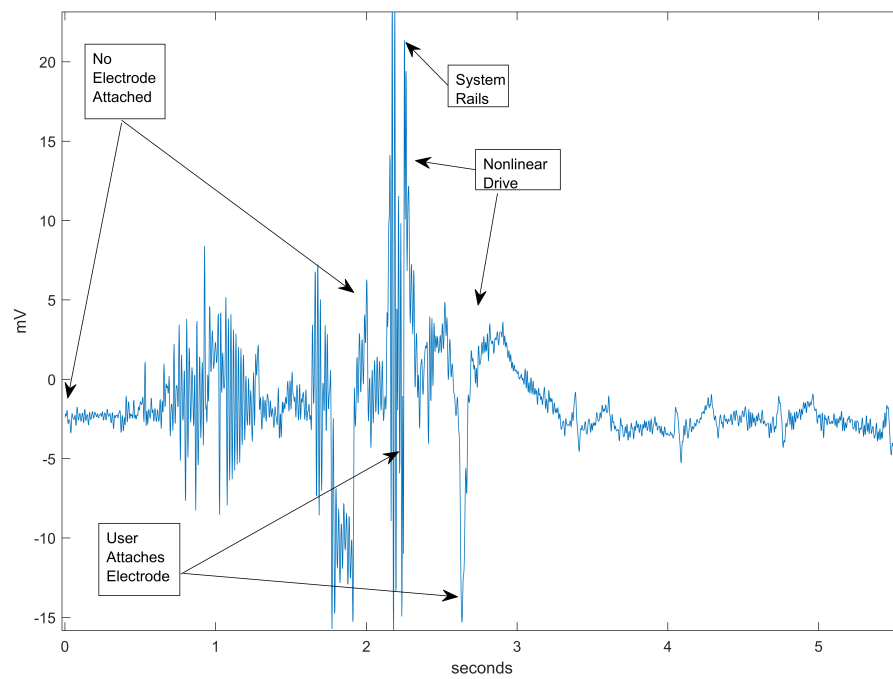


Figure 6.29: System ADC input saturation due to lead - off condition. Nonlinear control is evident, with system responding in well under a single beat.

The DSP Shield also enables the implementation of a second, high-speed control loop. If the signal measured at the ADC is sufficiently near the input limits to risk distortion ($>90\%$ of V_{ref}), the system deems the signal invalid and enables the nonlinear drive.

This algorithm checks if each sample is valid ($<90\%$ of $\pm V_{ref}$) and if not, drives the signal towards 0 V by 25% of the full ADC scale. Due to the gain of the final amplification stage, the DAC can change the offset by far more than the full scale of the ADC. Once the signal has been measured as valid for several cycles, normal proportional control is restored. This allows the system to recover from an out-of-range signal in about two milliseconds, as seen in Figure 6.29.

This response is dramatically faster than would be possible in a traditional system with a long time constant high-pass filter, which would be on the order of tens of seconds. While similar time performance would be possible in a DC-coupled system without this control loop, much lower gain would be possible.

6.19 Hybrid DC-AC ECG Performance

As with the previous ECG systems developed using the ASPEN platform, the hybrid eight-channel DC-AC system was characterized for input referred noise, SINAD, and a sample ECG was captured. Additionally, the system was characterized for CMRR performance from 1 Hz to 100 kHz.

As seen in Figure 6.30, the grounded, input referred noise is $47\ \mu\text{V}$, better than either the AD8232 or the single stage circuit. This means that the higher gain achieved by the AC-DC-coupled circuit improves the total input-referred noise. SINAD is improved to 38 dB, a relatively small improvement. Fully integrated analog switches may improve this further. The CMRR of the system was also broadly comparable to that of the single stage system, as seen in Figure 6.31. This is expected, with 80 dB of CMRR at low-frequency and decreasing to approximately 50 dB until the system filtering cutoff between 100 Hz and 200 Hz. Table 6.3 summarizes the performance

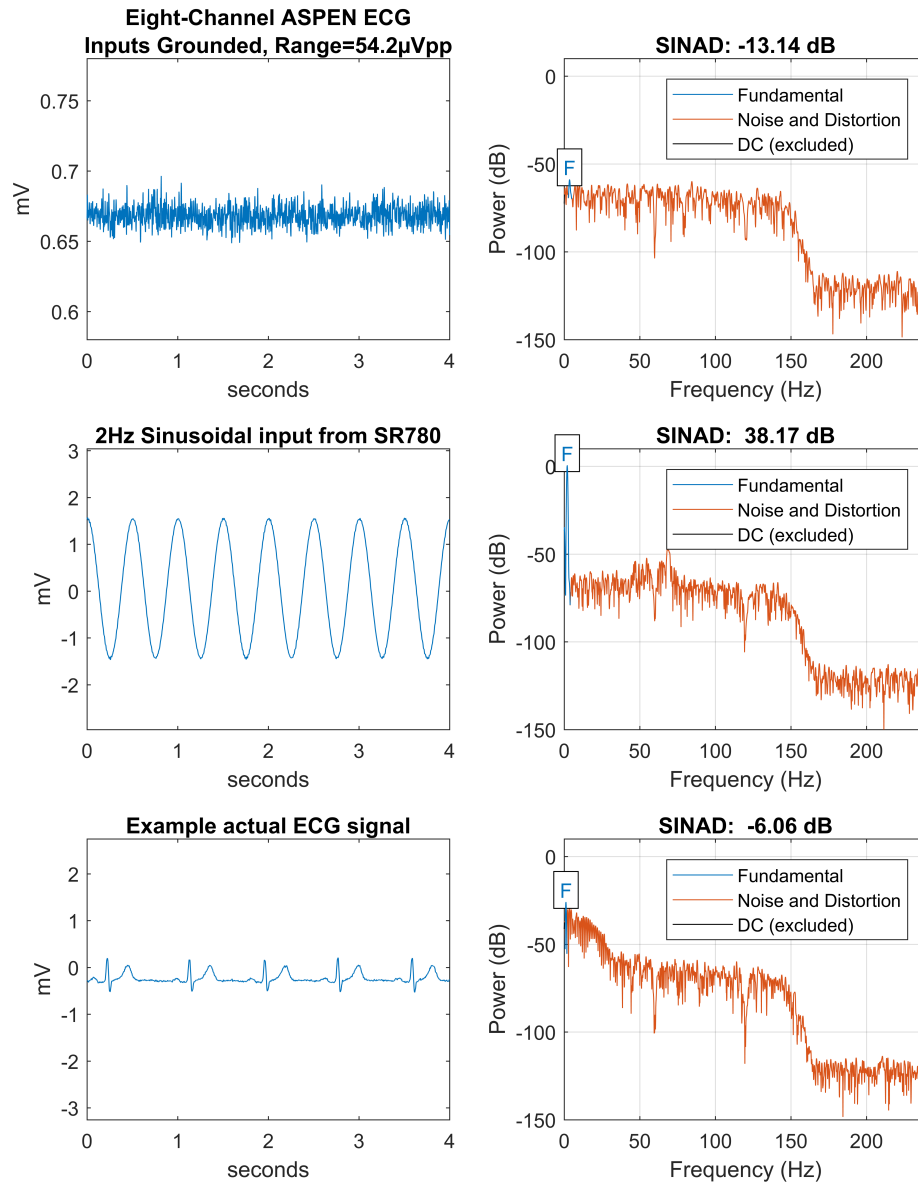


Figure 6.30: Eight-channel hybrid ECG, showing 54 μV_{pk-pk} of grounded input referred noise (top), An SNR of 38 dB with a 2 Hz 3 mV peak-to-peak sine wave (middle), and an actual recorded ECG (bottom). Frequencies of interest are DC to 150 Hz.

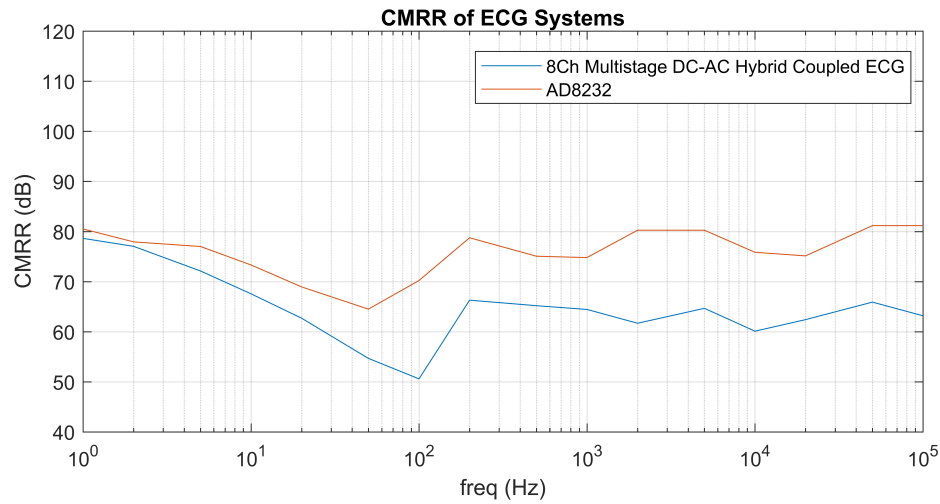


Figure 6.31: CMRR of 8Ch Multistage Hybrid ECG vs AD8232. As expected, there is some CMRR cost in the eight-channel system, versus the previously described systems and the reference AD8232. Nonetheless, overall performance is comparable between the systems, and as expected the multistage system performs similarly to the single-stage eight-channel version.

of the eight-channel DC-coupled system and compares it to the single stage device as well as the commercial AD8232 system.

The combination of DC-coupled ECG, with multiple channels, and a nonlinear digital high pass filter at the output produces a novel system with many of the benefits of both DC- and AC-coupled approaches. The rapidity with which it was prototyped, as well as the capabilities demonstrated clearly show the utility of the ASPEN platform as a powerful tool for prototyping of biomedical instrumentation.

Device	Input Referred Noise ($\mu V_{peak-peak}$)	SINAD (dB)	CMRR @ 5 Hz (dB)
AD8232	67	42	79
Eight-Channel Single-stage	84	34	62
Eight-Channel Multi-stage	47	38	73

Table 6.3: Comparison of performance specifications of ECG devices.

6.20 Offline Post-Processing Demonstrations

The single stage DC-coupled eight-channel ECG was used with standard wet electrodes to acquire real-world sample ECG signals for the leads I, II, and III. This data was used for several post processing algorithms developed using Matlab. These include a QRS complex extractor, shown in Figure 6.32. The source code for these experiments is in Appendix G.

Using the extracted QRS complex times as a trigger, an ensemble averaging block was created to generate normalized QRS complexes for each of the three leads, shown in Figure 6.33. The system was used to generate the QRS axis vector as seen in Figure 6.33. Because each of the leads in the Einthoven triangle represents a different spatial projection of the electrical signal of the heart, and the location of the leads is known, the dominant axis of the electrical activity of the heart can be determined. First, values for $Lead_I$, $Lead_{II}$ and $Lead_{III}$ are computed by measuring the average magnitude of the acquired QRS complex for each lead. With the positioning and

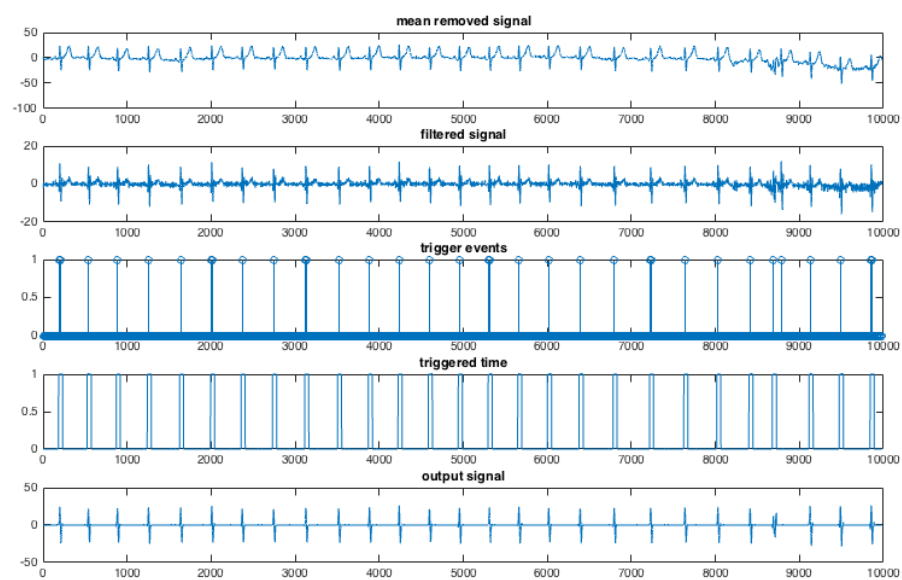


Figure 6.32: Plot of ECG signal (top), detected QRS signal trigger events (third row), and extracted QRS complexes (bottom) for a lead I ECG signal.

polarity of the leads, Equation 6.10 can be used to compute the QRS axis.

$$\begin{aligned}
 I_{II} &= Lead_I + Lead_{II} \cdot \frac{j\sqrt{3}}{2} \\
 II &= Lead_{II} \cdot \frac{1}{2} + Lead_{II} \cdot \frac{j\sqrt{3}}{2} \\
 III &= Lead_{III} \cdot \frac{-1}{2} + Lead_{III} \cdot \frac{j\sqrt{3}}{2} \\
 QRSVector &= \frac{I_{II} + II + III}{3}
 \end{aligned} \tag{6.10}$$

The QRS axis can be a useful tool in detecting cardiac abnormalities.

Finally, the QRS extractor block was combined with P- and T-wave extractors to build the system chain shown in Figure 6.35, producing an off-line QT-interval measurement system, the output of which can be seen in Figure 6.34. The QT-interval is a clinically important feature of the ECG. An elongated QT-interval, called long-QT syndrome is a serious health risk, and more than half of people who have untreated, inherited long-QT syndrome die within 10 years [90]. Importantly, some drugs can cause further elongation of the QT interval. In healthy individuals, this is not a problem, but because long-QT syndrome can go otherwise undetected, QT-interval measurement has become common when prescribing indicated drugs [91].

While algorithms of this sort are not fundamentally new [92], the ability to rapidly prototype a system to acquire such signals along with the possibility of later integrating such functionality directly into a prototype demonstrates the utility of the platform with a clinically useful outcome.

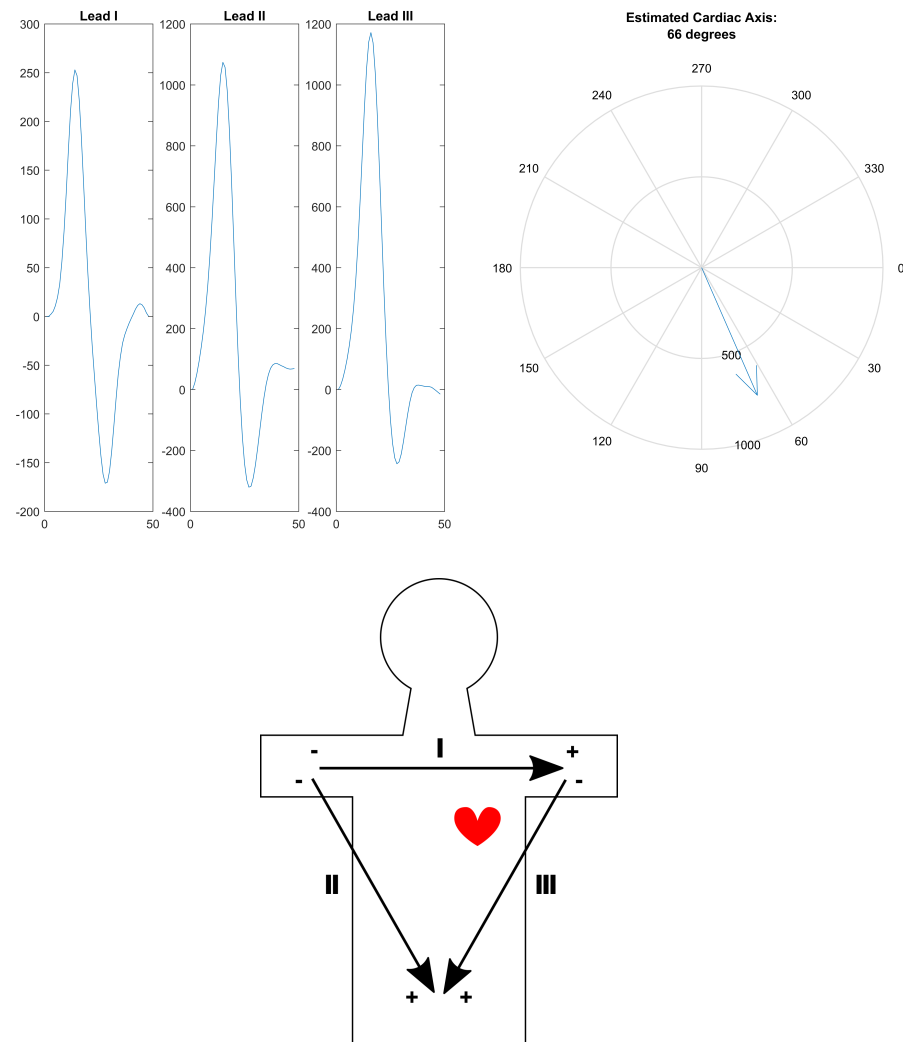


Figure 6.33: Cardiac axis reconstruction via extracted, ensemble averaged QRS complexes (top). Einthoven Triangle (bottom).

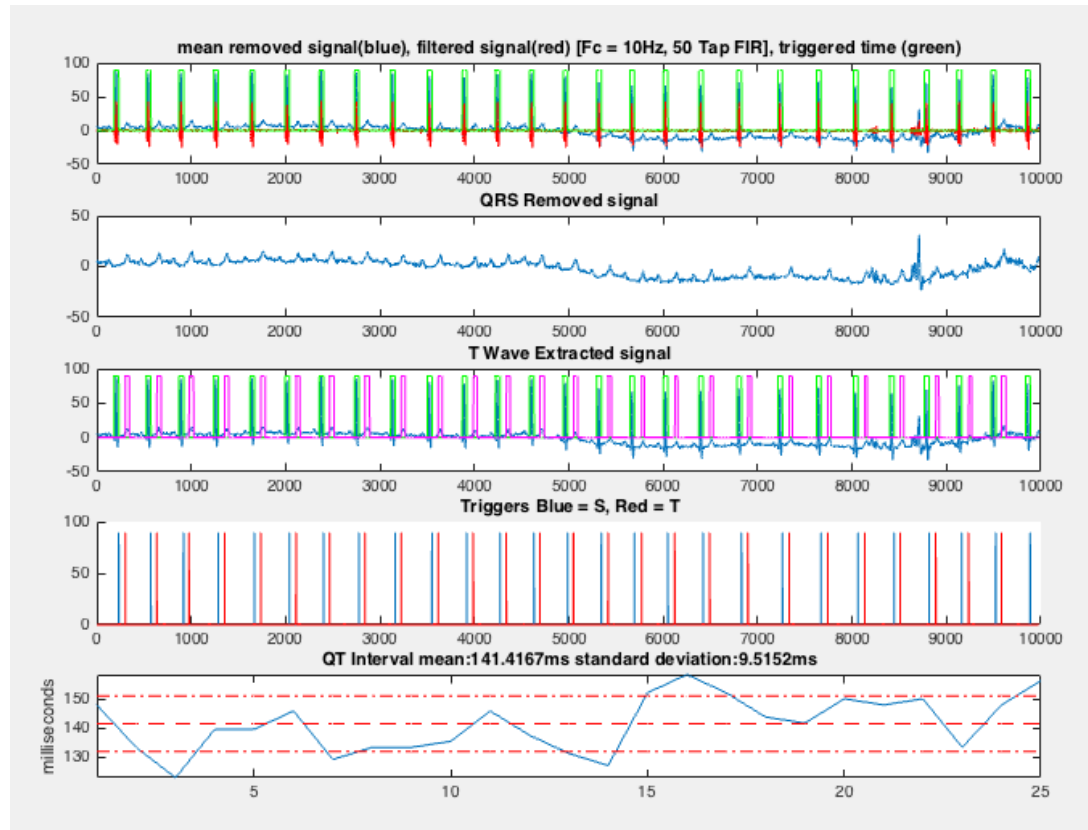


Figure 6.34: (Top Row) Extracted QRS complex, (Second Row) ECG with QRS complex removed, (Third Row) Detected QRS and T events, (Fourth Row) S-T trigger events, (Fifth Row) Estimated ST intervals with mean and 1-standard deviation bars.

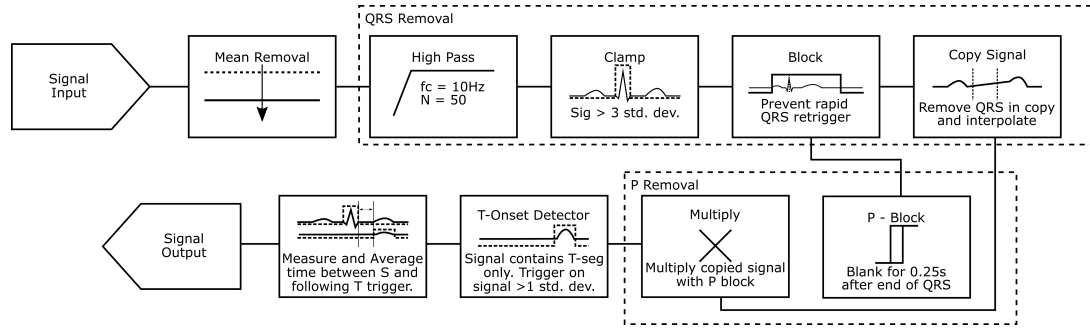


Figure 6.35: ST interval estimation algorithm. The algorithm removes the mean from the signal, then detects and removes the QRS and P complexes from the signal, storing the timing of the falling edge of the complex. The algorithm then detects the onset of the T-wave.

6.21 Conclusion

These experiments demonstrate the utility of a simple-to-use software platform coupled with the Analog Shield and DSP Shield for streamlining the development of educational biomedical systems and rapid prototyping. With ASPEN, we have demonstrated a novel ECG system, a DC-coupled multi-channel circuit with nonlinear offset correction. The resulting high-gain, multi-channel ECG system benefits from the best aspects of both existing AC- and less common DC- coupled systems. It combines high gain with fast settling times and reduced analog circuit complexity. It enables same-beat recovery after large upsets, faster than current AC-coupled systems, which can require as much as 30 seconds to fully settle. This could serve many clinical and non-clinical needs, potentially improving the performance of outpatient ECG systems (e.g. Holter monitors could have higher percentages of valid signal intervals, improving the reliability of outpatient stress tests, and other applications).

Chapter 7

Conclusions

7.1 Overview

The ASPEN project was developed to address a gap in the tools for several fields. It was created to support teaching and prototyping of mixed-signal-electronics, digital signal processing, and biomedical electronics. The Analog Shield enables building and interfacing analog circuits with digital systems. The DSP Shield extends the Arduino metaphor to signal processing concepts to address the need for powerful, easy to learn and embeddable signal processing tools. Finally, the platform as a whole can be leveraged to build useful tools for the teaching and prototyping of new biomedical sensors.

The Analog Shield is a Swiss Army Knife for converting between the analog and digital domains. With a multi-channel ADC and DAC, a prototyping area and bipolar power supplies, it simplifies the development of the DSP Shield is a unique, powerful signal processing tool. It is a modern, Arduino-style embeddable DSP that improves teaching and prototyping of real-time signal processing, without obscuring the details of signal manipulation.

As these boards were developed, their performance was characterized to confirm that they met the needs of the tasks at hand. Hardware revisions were made to both devices to improve their utility as teaching and development tools. In support of the hardware development, a large and complex software project was undertaken. It includes driver libraries, examples, tutorials, course material, development tools, and abstraction libraries.

The Analog Shield driver library was developed with support for several microcontroller platforms. Made publicly available as open-source software, it provides a simple interface for the Analog Shield, similar to pre-existing Arduino conventions. The library includes a handful of teaching examples to both aid the user in starting to develop with the device and to demonstrate basic signal processing concepts.

To demonstrate the utility of the platform for mixed-signal applications, examples were developed using the Analog Shield. These included a series of Instructables style documents which demonstrated the utility of the platform to rapidly prototype mixed-signal systems that could not be readily achieved using just an Arduino Uno. Among these were a function generator, spectrum analyzer, and network analyzer.

For the DSP Shield, the support tools were more complex. The DSP Shield can function in three modes. The first, shield mode is an introductory mode that allows the DSP Shield to work as a fixed function signal toolchain. The second, co-processor mode allows the user to write their own code for the DSP Shield to support an Arduino. In the third, standalone mode, the DSP Shield operates as a fully independent development platform.

To support the simplified shield mode, DSP Shield communication and support libraries were developed for the Arduino. These libraries allow the microcontroller to interface with the Shield App. This open-source program runs on the DSP Shield and presents a flexible processing toolchain for audio signals. It includes functions such as signal generators, banks of pre-computed filters and more. The DSP Shield library for the Arduino supports this application and includes example code that demonstrates the capabilities of the Shield App. These include signal generation,

filtering, voice removal from music, and more. With this library, a basic hello world example for signal processing was developed, providing an immediate, accessible tool to demonstrate how DSP can be used to manipulate a signal.

For more advanced users, a set of tools called Energia-DSP was developed. These tools allow users to develop programs directly for the DSP Shield, and are used for the co processor and standalone modes. The core of Energia-DSP is its eponymous development environment, derived from the Arduino IDE. To support the platform, software was developed to allow the user upload and store programs in non-volatile memory on the DSP Shield with a single click. To make the experience of programming for the DSP Shield similar to the Arduino, hardware abstraction libraries were created to re-implement the Arduino interface. A signal processing library was created to extend the metaphor to DSP tasks.

The platform was deployed for teaching of mixed-signal concepts in Stanfords EE122A analog electronics laboratory. Its use was integral to the conversion of the class to a take home lab format, with enhanced course material from the previous in-lab version. This included use of the Analog Shield in several lab experiments. One of these was the creation of a curve tracer test instrument. In also included use of the DSP Shield as part of an experiment to compare and contrast the implementation of filters using a variety of technologies. A similar toolchain including the Analog Shield was used outside of Stanford at the University of Colorado by Professor Mark Wickert, and taught similar material. This course culminated in using the Analog Shield for an AM demodulation lab.

The DSP Shield has not yet been sold commercially, and so has only seen use within courses at Stanford. Nevertheless, its use in EE264 enabled the addition of a hands on lab component to a theoretical course. Previously, Stanford had a four-credit DSP lab course. This course covered much of the same material but required that students carefully optimize assembly code for full credit. The new EE264 lab option was a shorter, one credit hour offering which covered similar fundamental topics at a higher level of abstraction. The utility of the approach is demonstrated most clearly by the fact that the final project of both courses is an implementation of an OFDM decoder.

Finally, the ASPEN toolbox has been used to demonstrate its utility as a platform for biomedical teaching and experimentations. To this end, DC-coupled electrocardiography was investigated. The circuits developed demonstrated the utility of the tool for enhancing the teaching of biomedical system design, showing similar performance to a previous textbook teaching circuit. The new circuit uses significantly fewer components, shortening construction time while requiring students to use their understanding make similar design decisions.

The system was then applied to the development of a novel DC-coupled, multi-channel ECG system. This system demonstrates the union of DC-coupled ECG with DSP processing power to create a digitally conditioned hybrid ECG. This ECG has reduced component count and mixes the benefits of a fast-settling DC-coupled system with the high gain of an AC-coupled system.

7.2 Future Work

Moving forward, there is extensive opportunity to continue to develop the platform. Hopefully, given the open nature of the platform, we will be able to find or create a manufacturer to produce the DSP Shield in quantity and release it to the world. If we succeed, this will allow a community of users to develop to extend and improve upon the hardware and software tools and reference materials already developed for the platform.

Appendix A

Analog Shield ADC Test Code

```
#include "analogShield.h"
#define LEN 801
unsigned int vect_in[LEN] = {0};
void setup() {
    initialize();
    Serial.begin(115200);
    while (!Serial.available() || (Serial.available() == 1 && Serial.read() == 0))
        while (Serial.available()) {
            Serial.read();
        }
}
void initialize()
{
    cli();    // Disable global interrupts while we make our changes.
    TIMSK0 &=~(1 << TOIE0);    // Disable Timer 0.
    TIMSK2 &=~(1 << TOIE2);    // Disable Timer 2.
    sei();    // Enable global interrupts.
}
void loop() {
```

```

cli(); // UDRE interrupt slows this way down on arduino1.0
for (int i = 0 ; i < LEN ; i++) { // save 256 samples
    vect_in[i] = analog.read(); // put real data into bins
}
sei();
printHex(vect_in);
while (!Serial.available() || //
(Serial.available() == 1 && Serial.read() != '1')){};
while (Serial.available()) {
    Serial.read();
}
}
void printHex(unsigned int val)
{
    for (int i = 0 ; i < LEN ; i++) { // save 256 samples
        if (vect_in[i] < 0x10){
            Serial.print('0');
            Serial.print('0');
            Serial.print('0');
        } else if (vect_in[i] < 0x100) {
            Serial.print('0');
            Serial.print('0');
        } else if (vect_in[i] < 0x1000) {
            Serial.print('0');
        }
        Serial.println(vect_in[i], HEX);
    }
}

```

Appendix B

Analog Shield Schematics and BOM

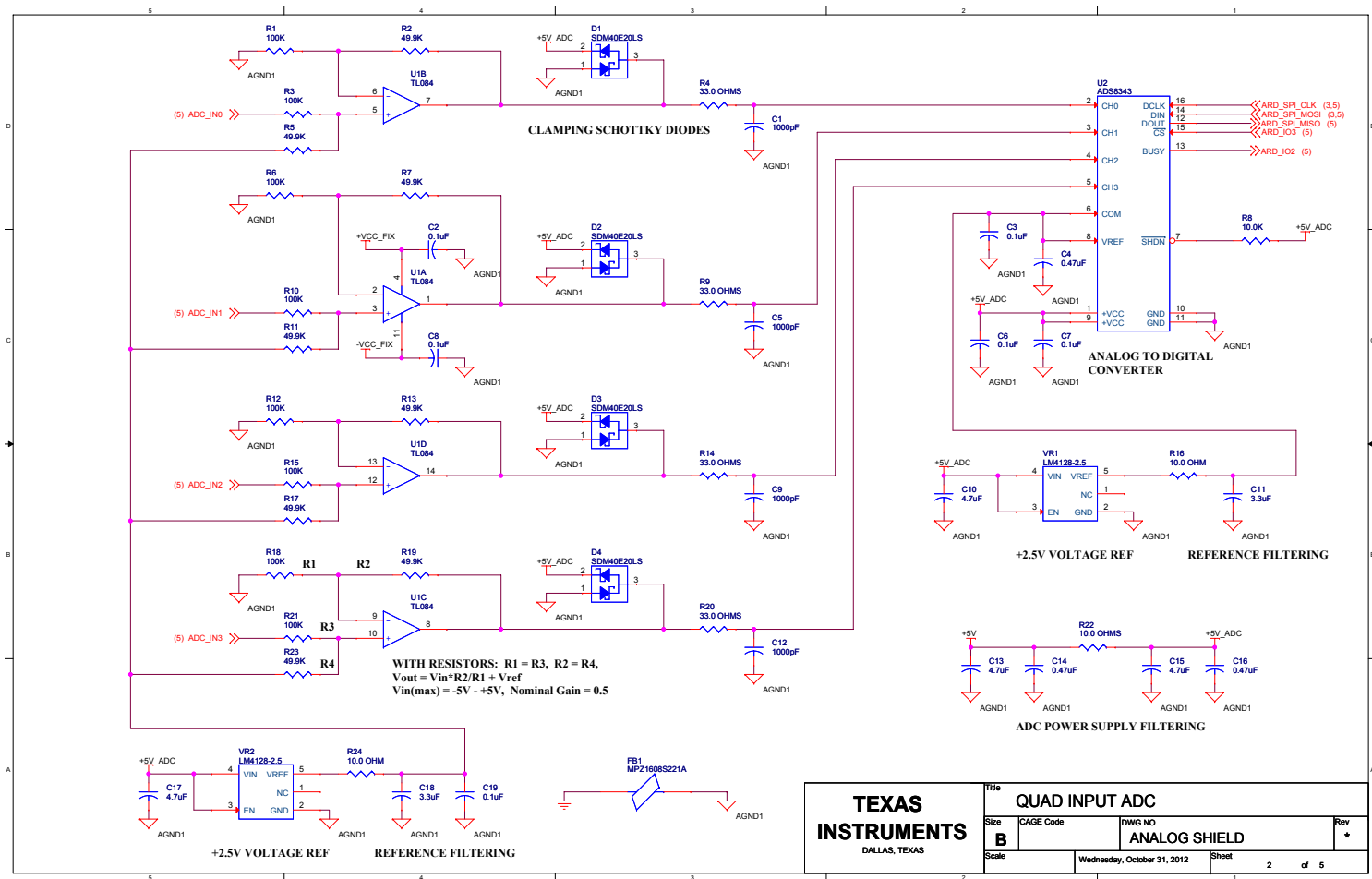
NOTES:

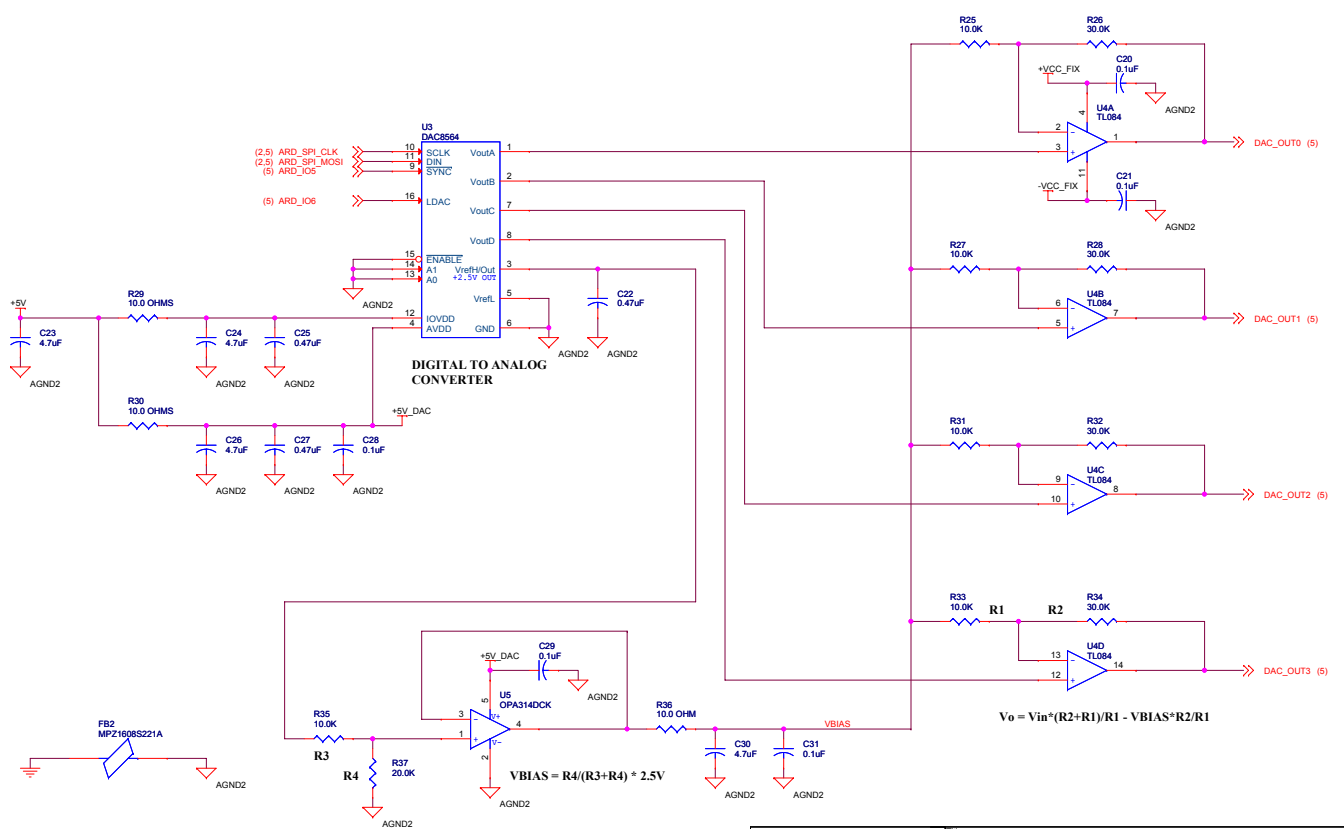
SHEET DESCRIPTIONS:

- 01: ESD TITLE PAGE
- 02: QUAD INPUT ADC
- 03: QUAD DAC
- 04: VOLTAGE REGULATORS
- 05: ARDUINO & EXPANSION HEADERS

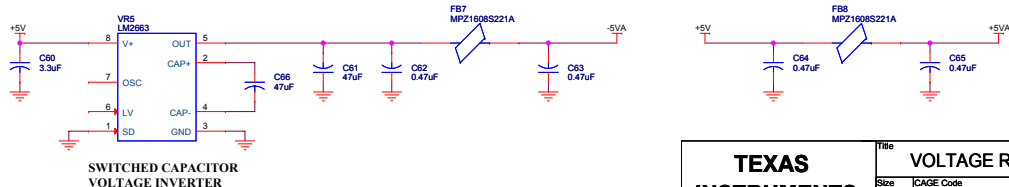
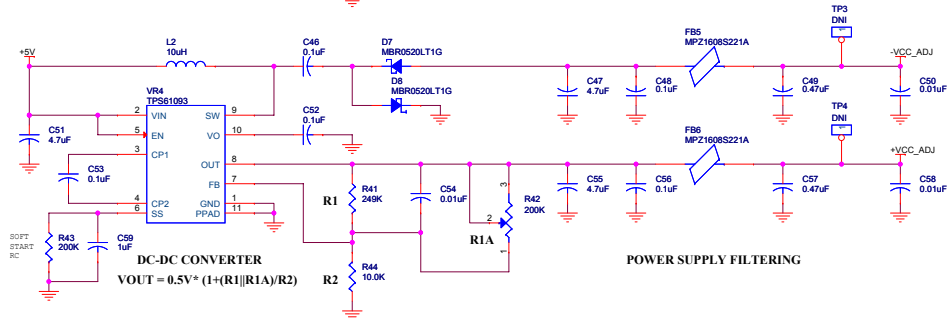
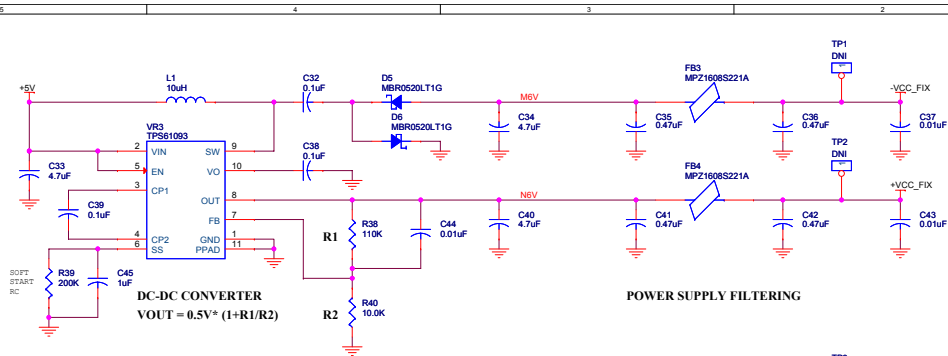
Version: Oct 31, 2012

TEXAS INSTRUMENTS DALLAS, TEXAS	SYSTEMS AND APPLICATIONS R&D CENTER SYSTEM ARCHITECTURES LAB 12500 TI Boulevard Dallas, TX 75243 MS 8849			
	ELECTRONIC SCHEMATIC DIAGRAM			
ENGR D. GARCIA	Size B	CAGE Code	DWG NO ANALOG SHIELD	Rev *
RIGHTS COMPLIANT	Scale	Wednesday, October 31, 2012		Sheet 1 of 5



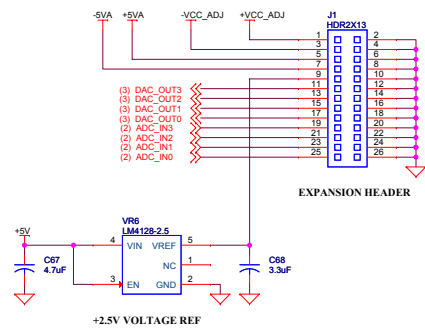
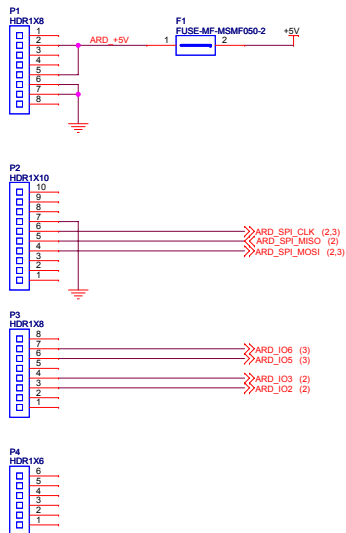


TEXAS INSTRUMENTS DALLAS, TEXAS		QUAD DAC		
		Size B	CAGE Code	DWG NO ANALOG SHIELD
Scale		Wednesday, October 31, 2012		Sheet 3 of 5



TEXAS INSTRUMENTS DALLAS, TEXAS		VOLTAGE REGULATORS		
		Size B	CAGE Code	DWG NO ANALOG SHIELD
Scale		Wednesday, October 31, 2012		Sheet 4 of 5

ARDUINO HEADERS



TEXAS INSTRUMENTS DALLAS, TEXAS		Title ARDUINO & EXPANSION HEADERS		
		Size B	CAGE Code	Rev *
		Scale	DWG NO DSP SHIELD	
		Scale	Wednesday, October 31, 2012	Sheet 5 of 5

Analog SHIELD

ELECTRONIC SCHEMATIC DIAGRAM Revised: Friday, March 28, 2014
ANALOG SHIELD Revision: D

TEXAS INSTRUMENTS
12500 TI BOULEVARD
DALLAS, TX 75243

IN COLLABORATION WITH STANFORD UNIVERSITY

Bill Of Materials March 28,2014 14:19:11

ITEM	QTY	REF DES	PART	DESCRIPTION	MFG	MFG P/N	SUPPLIER	SUPPLIER P/N	PACKAGE	Single QTY Price	Price in Volume	Volume Size	Notes 1	Notes 2
1	8	C1,C10,C12,C18,C80,C81,C82,C83	12pF	CAPACITOR, 50V NPO 5%	Yageo	CC0402JRNPO9BN120	DIGIKEY	311-1016-1-ND	402	\$ 0.10	\$ 0.00	1000		
2	9	C2,C13,C14,C41,C50,C54,C63,C64,C91	0.1uF	CAPACITOR, 25V CERAMIC X5R 10%	TDK Corporation	C1005X5R1E104K	DIGIKEY	445-4964-1-ND	402	\$ 0.10	\$ 0.01	1000		
3	8	C3,C21,C24,C25,C26,C28,C30,C34	10uF	CAPACITOR, 16V CERAMIC X5R 20%	Taiyo Yuden	EMK107BB1106MA-T	DIGIKEY	587-3238-1-ND	603	\$ 0.63	\$ 0.15	1000		
4	15	C4,C9,C20,C31,C35,C36,C37,C42,C49,C53,C55,C65,C84,C85,C88	0.01uF	CAPACITOR, 25V CERAMIC X7R 10%	TDK Corporation	C1005X7R1E103K	DIGIKEY	445-1260-1-ND	402	\$ 0.10	\$ 0.01	1000		
5	4	C5,C7,C38,C43	2.2uF	CAPACITOR, 25V CERAMIC X5R 10%	Taiyo Yuden	TMK107ABJ225KA-T	DIGIKEY	587-2909-1-ND	603	\$ 0.22	\$ 0.05	1000		
6	4	C6,C39,C89,C90	DNI	CAPACITOR					402				Do Not Install	
7	4	C8,C11,C16,C19	1000pF	CAPACITOR, 25V CERAMIC X7R 10%	TDK Corporation	C1005X7R1E102K	DIGIKEY	445-4924-1-ND	402	\$ 0.10	\$ 0.01	1000		
8	3	C15,C22,C32	2.2uF	CAPACITOR, 3 TERMINAL EMIFIL 20%	Murata Electronics North America	NFM18PC225B1A3D	DIGIKEY	490-6972-1-ND	CAP-3TERM_MURATA	\$ 0.30	\$ 0.12	1000		
9	13	C17,C29,C33,C40,C47,C48,C51,C59,C62,C66,C70,C78,C79	4.7uF	CAPACITOR, 25V CERAMIC X5R 10%	Samsung Electro-Mechanics America, Inc	CL10A475KA8NQNC	DIGIKEY	1276-1900-1-ND	603	\$ 0.33	\$ 0.09	1000		
10	6	C23,C27,C57,C69,C75,C87	1uF	CAPACITOR, 25V CERAMIC X5R 10%	TDK Corporation	C1608X5R1E105K	DIGIKEY	445-5146-1-ND	603	\$ 0.13	\$ 0.02	1000		

11	2	C44,C58	0.1uF	CAPACITOR, 25V CERAMIC X7R 10%	Kemet	C0805C104K3RACTU	DIGIKEY	399-1168-1-ND	805	\$ 0.10	\$ 0.01	1000		
12	6	C45,C52,C60,C67,C72,C73	0.47uF	CAPACITOR, 25V CERAMIC X5R 10%	TDK Corporation	C1005X5R1E474M	DIGIKEY	445-8021-1-ND	402	\$ 0.13	\$ 0.02	1000		
13	4	C46,C56,C61,C68	10uF	CAPACITOR, 25V CERAMIC X5R 10%	Samsung Electro-Mechanics America, Inc	CL21A106KAYNNNE	DIGIKEY	1276-2891-1-ND	805	\$ 0.25	\$ 0.05	1000		
14	2	C71,C76	47uF	CAPACITOR, 10V CERAMIC X5R 20%	TDK Corporation	C3216X5R1A476M	DIGIKEY	445-6006-1-ND	1206	\$ 1.13	\$ 0.41	1000		
15	3	C74,C77,C86	22uF	CAPACITOR, 25V CERAMIC X5R 10%	Samsung Electro-Mechanics America, Inc	CL31A226KAHNNNE	DIGIKEY	1276-3047-1-ND	1206	\$ 0.50	\$ 0.16	1000		
16	4	D1,D2,D3,D4	SDM40E20LS	DIODE, SCHOTTKY DUAL SERIES	Diodes Inc	SDM40E20LS-7-F	DIGIKEY	SDM40E20LS-FDICT-ND		\$ 0.44	\$ 0.16	1000		
17	2	D5,D6	BAT54S	DIODE, SCHOTTKY DUAL SERIES	Diodes Inc	BAT54S-7-F	DIGIKEY	BAT54S-FDICT-ND		\$ 0.44	\$ 0.16	1000		
18	13	FB1,FB2,FB3,FB4,FB5,FB6,FB7,FB8,FB9,FB10,FB11,FB12,FB13	MPZ1608S221A	INDUCTOR, FERRITE BEAD 220 OHM 2A	TDK Corporation	MPZ1608S221A	DIGIKEY	445-1565-1-ND	603	\$ 0.31	\$ 0.03	1000		
19	1	F1	FUSE-MF-- MSMF050-2	FUSE	Bourns Inc.	MF-MSMF050-2	DIGIKEY	MF-MSMF050-2CT-ND	1812	\$ 0.22	\$ 0.12	1000		
20	1	JP1	HDR2X3	CONNECTOR, 6-PIN HEADER	SAMTEC	TSW-103-07-T-D	AVNET	TSW-103-07-T-D	HDR2X3	\$ 0.22	\$ 0.19	1000		
21	2	L1,L2	10uH	INDUCTOR, POWER 1.23A	SUMIDA	CDH53NP-100LC	DIGIKEY	308-1421-1-ND	IND2-CDH53_SUM	\$ 0.64	\$ 0.36	1000		
22	2	P1,P2	HDR8-2x4-FEM	HEADER, 8-PIN 2X4 FEMALE	SAMTEC	SSW-104-01-T-D	AVNET	SSW-104-01-T-D	HDR8-2X4-FEM	\$ 0.66	\$ 0.66	1000		
23	1	P3	HDR10-2x5-FEM	HEADER, 10-PIN 2X5 FEMALE	SAMTEC	SSW-105-01-T-D	AVNET	SSW-105-01-T-D	HDR10-2x5-FEM	\$ 0.68	\$ 0.68	1000		
24	2	P4,P7	HDR1X8	HEADER, FEMALE 8POS .1"	SAMTEC	SSQ-108-04-T-S	AVNET	SSQ-108-04-T-S	HDR1X8	\$ 0.74	\$ 0.61	1000		Do not cut off wire-wrap tails
25	1	P5	HDR2X3-FEM	HEADER, FEMALE 6POS .1" 2ROW	SAMTEC	ESQ-103-24-T-D	AVNET	ESQ-103-24-T-D	HDR2X3-FEM	\$ 0.92	\$ 0.77	1000		Do not cut off wire-wrap tails
26	1	P6	HDR1X10	HEADER, FEMALE 10POS .1"	SAMTEC	SSQ-110-04-T-S	AVNET	SSQ-110-04-T-S	HDR10_SULL	\$ 0.92	\$ 0.76	1000		Do not cut off wire-wrap tails
27	1	P8	HDR1X6	HEADER, FEMALE 6POS .1"	SAMTEC	SSQ-106-04-T-S	AVNET	SSQ-106-04-T-S	HDR1X6	\$ 0.55	\$ 0.45	1000		Do not cut off wire-wrap tails
28	8	R1,R7,R11,R15,R17,R20,R22,R25	100K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF1003X	DIGIKEY	P100KLCT-ND	402	\$ 0.10	\$ 0.01	1000		

29	8	R2,R9,R12,R16,R18,R21, R23,R26	40.2K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF4022X	DIGIKEY	P40.2KLCT-ND	402	\$	0.10	\$	0.01	1000		
30	7	R3,R5,R8,R32,R33,R34,R35	27.0 OHM	RESISTOR, 1/10W JUMPER	Panasonic Electronic Components	ERJ-2RKF27R0X	DIGIKEY	P27.0LCT-ND	402	\$	0.10	\$	0.01	1000		
31	4	R4,R14,R19,R24	33.0 OHM	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF33R0X	DIGIKEY	P33.0LCT-ND	402	\$	0.10	\$	0.01	1000		
32	14	R6,R10,R57,R58,R59,R60, R61,R62,R64,R65,R66,R67, R68,R69	100 OHM	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF1000X	DIGIKEY	P100LCT-ND	402	\$	0.10	\$	0.01	1000		
33	10	R13,R30,R36,R38,R40,R43, R49,R55,R56,R63	10.0K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF1002X	DIGIKEY	P10.0KLCT-ND	402	\$	0.10	\$	0.01	1000		
34	3	R27,R29,R42	10.0 OHM	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF10R0X	DIGIKEY	P10.0LCT-ND	402	\$	0.10	\$	0.00	1000		
35	1	R28	10.0K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-3EKF1002V	DIGIKEY	P10.0KHCT-ND	603	\$	0.10	\$	0.01	1000		
36	4	R31,R37,R39,R41	30.0K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF3002X	DIGIKEY	P30.0KLCT-ND	402	\$	0.10	\$	0.01	1000		
37	1	R44	20.0K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF2002X	DIGIKEY	P20.0KLCT-ND	402	\$	0.78	\$	0.01	1000		
38	2	R45,R50	0.0 OHM	RESISTOR, 1/10W JUMPER	Panasonic Electronic Components	ERJ-2GE0R00X	DIGIKEY	P0.0JCT-ND	402	\$	0.05	\$	0.00	1000		
39	3	R46,R51,R70	DNI	RESISTOR, 1/8W JUMPER					805						Do Not Install	
40	1	R47	140K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF1403X	DIGIKEY	P140KLCT-ND	402	\$	0.10	\$	0.01	1000		
41	2	R48,R54	200K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF2003X	DIGIKEY	P200KLCT-ND	402	\$	0.10	\$	0.01	1000		
42	1	R52	470K	RESISTOR, 1/10W 1%	Panasonic Electronic Components	ERJ-2RKF4703X	DIGIKEY	P470KLCT-ND	402	\$	0.10	\$	0.01	1000		
43	1	R53	200K	POTENTIOMETE R	Bourns Inc.	3362P-1-204LF	DIGIKEY	3362P-204LF-ND			\$	0.98	\$	0.44	1000	
44	2	TP1,TP2	DNI	TERMINAL, TEST POINT	KEYSTONE	5002		5002K-ND							Do Not Install	
45	1	TP3	TEST POINT	TERMINAL, TEST POINT	KEYSTONE	5002	DIGIKEY	5002K-ND			\$	0.35	\$	0.15	1000	
46	2	U1,U5	LM837	I.C.,OP-AMP	Texas Instruments	LM837MX/NOPB	DIGIKEY	LM837MX/NOPBCT-ND				\$	1.44	\$	0.47	1000
47	1	U2	OPA4322	I.C.,OP-AMP	Texas Instruments	OPA4322AIPW	DIGIKEY	296-30337-5-ND			\$	2.90	\$	0.95	1000	
48	1	U3	ADS8343	I.C., 16-bit 4 Channel ADC	Texas Instruments	ADS8343EB	DIGIKEY	ADS8343EB-ND			\$	17.03	\$	8.95	1000	

49	1	U4	OPA2350EA	I.C., OP-AMP	Texas Instruments	OPA2350EA/250	DIGIKEY	OPA2350EACT-ND	SOIC8-DGK_TI	\$ 5.33	\$ 1.90	1000		
50	1	U6	DAC8564	I.C., 16-bit Quad DAC	Texas Instruments	DAC8564IAPW	DIGIKEY	296-22920-5-ND	SOIC16-PW_TI	\$ 14.16	\$ 5.95	1000		
51	1	U7	OPA314DBV	I.C., OP AMP	Texas Instruments	OPA314AIDBVR	DIGIKEY	296-36699-1-ND	SOIC5-DBV_TI	\$ 0.97	\$ 0.30	1000		
52	1	U8	SN74LVC2G125	I.C., DUAL TRI-STATE BUFFER	Texas Instruments	SN74LVC2G125DCUR	DIGIKEY	296-13265-1-ND	SOIC8-DCU_TI	\$ 0.70	\$ 0.28	1000		
53	1	VR1	REF5020	I.C., 2.048V VOLTAGE REFERENCE	Texas Instruments	REF5020AIDGKT	DIGIKEY	296-24496-1-ND	SOIC8-DGK_TI	\$ 4.02	\$ 1.55	1000		
54	2	VR2,VR3	TP561093	I.C., BOOST CONVERTER	Texas Instruments	TP561093DSKT	DIGIKEY	296-25245-1-ND	SOIC10-DSK_TI_TEMP	\$ 3.58	\$ 1.38	1000		
55	1	VR4	LM2663	I.C., SWITCHED CAP VOLT CONVERTER	Texas Instruments	LM2663M/NOPB	DIGIKEY	LM2663M-ND	SOIC8-D_TI	\$ 2.40	\$ 0.88	1000		
56	1	VR5	LM4128-2.5	I.C., VOLTAGE REFERENCE	Texas Instruments	LM4128DMF-2.5/NOPB	DIGIKEY	LM4128DMF-2.5CT-ND	SOIC5-DBV_TI	\$ 1.66	\$ 0.94	1000		
57	1		Breadboard	Solderless Breadboard	Parallex Inc.	700-00012	DIGIKEY	700-00012-ND		\$ 3.49	\$ 2.80		User Installed	
58	1		4 Layer PCB	Printed Circuit Board						?	?			

Appendix C

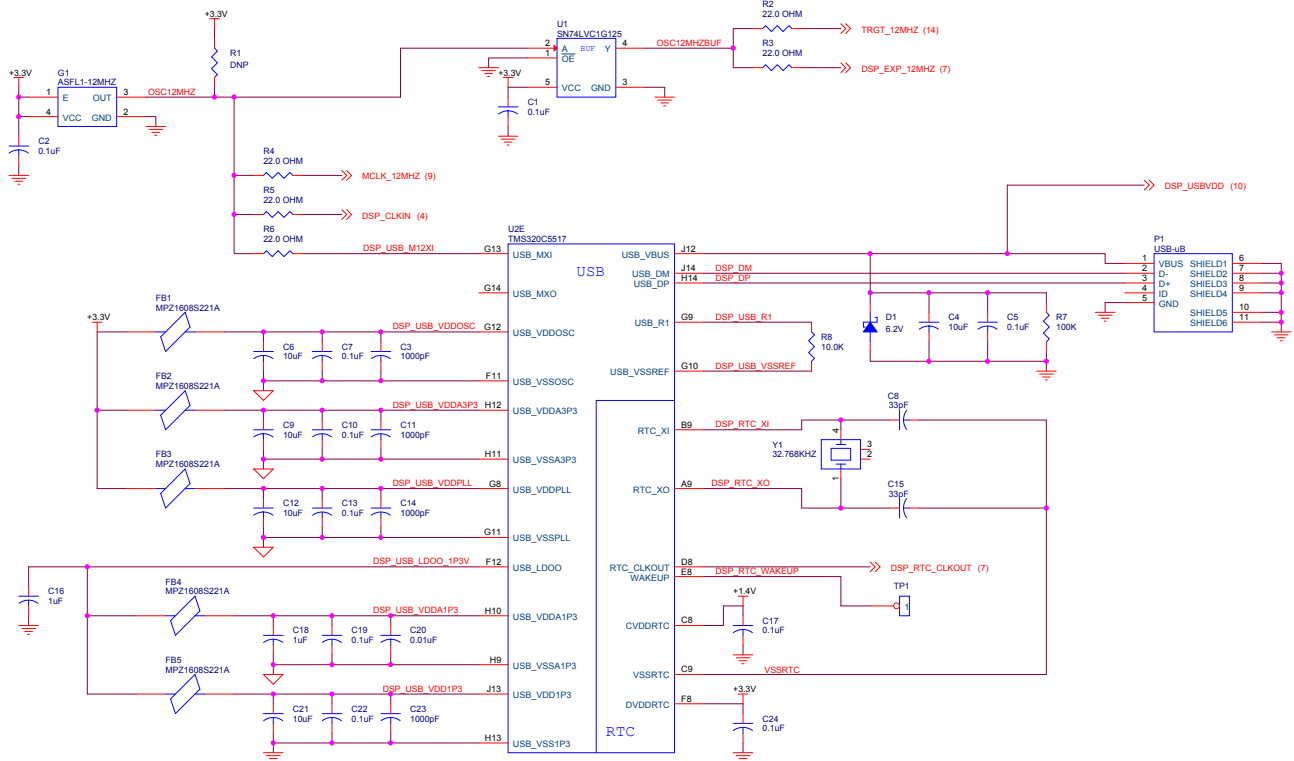
DSP Shield Schematics and BOM

NOTES:

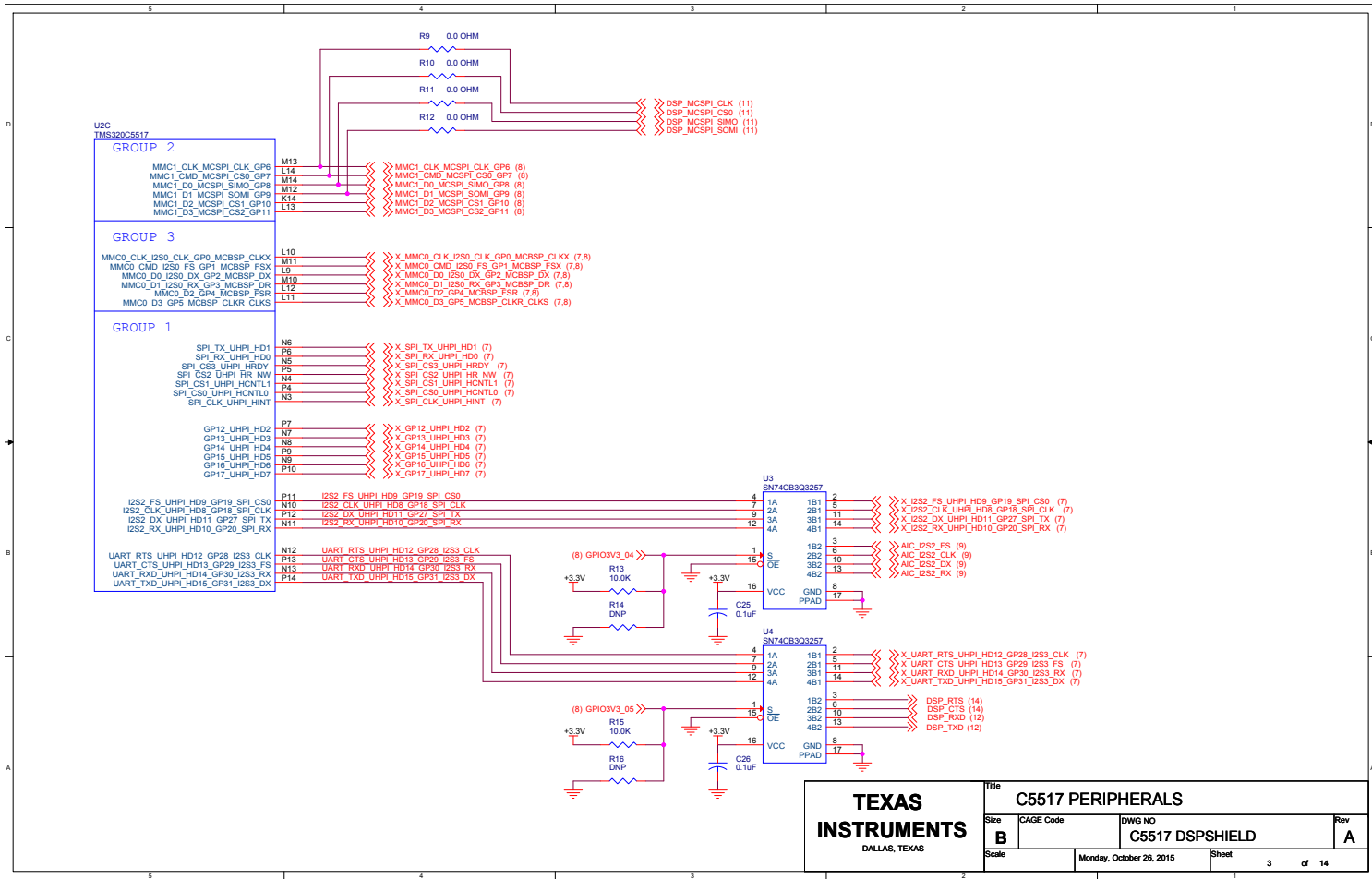
SHEET DESCRIPTIONS:

- 01: ESD TITLE PAGE
- 02: C5517 USB, RTC, OSC
- 03: C5517 PERIPHERALS
- 04: C5517 CLK,JTAG, I2C, GPAIN
- 05: C5517 EMIF, SDRAM, BOOTMODE
- 06: C5517 CORE AND IO POWER
- 07: DSP EXPANSION CONNECTOR
- 08: OLED, SDCARD, GPIO, DIPSW
- 09: TLV320AIC3204 CODEC
- 10: VOLTAGE REGULATORS, SVS
- 11: ARDUINO INTERFACE 1
- 12: ARDUINO INTERFACE 2
- 13: ARDUINO INTERFACE 3
- 14: XDS100-V2 INTERFACE

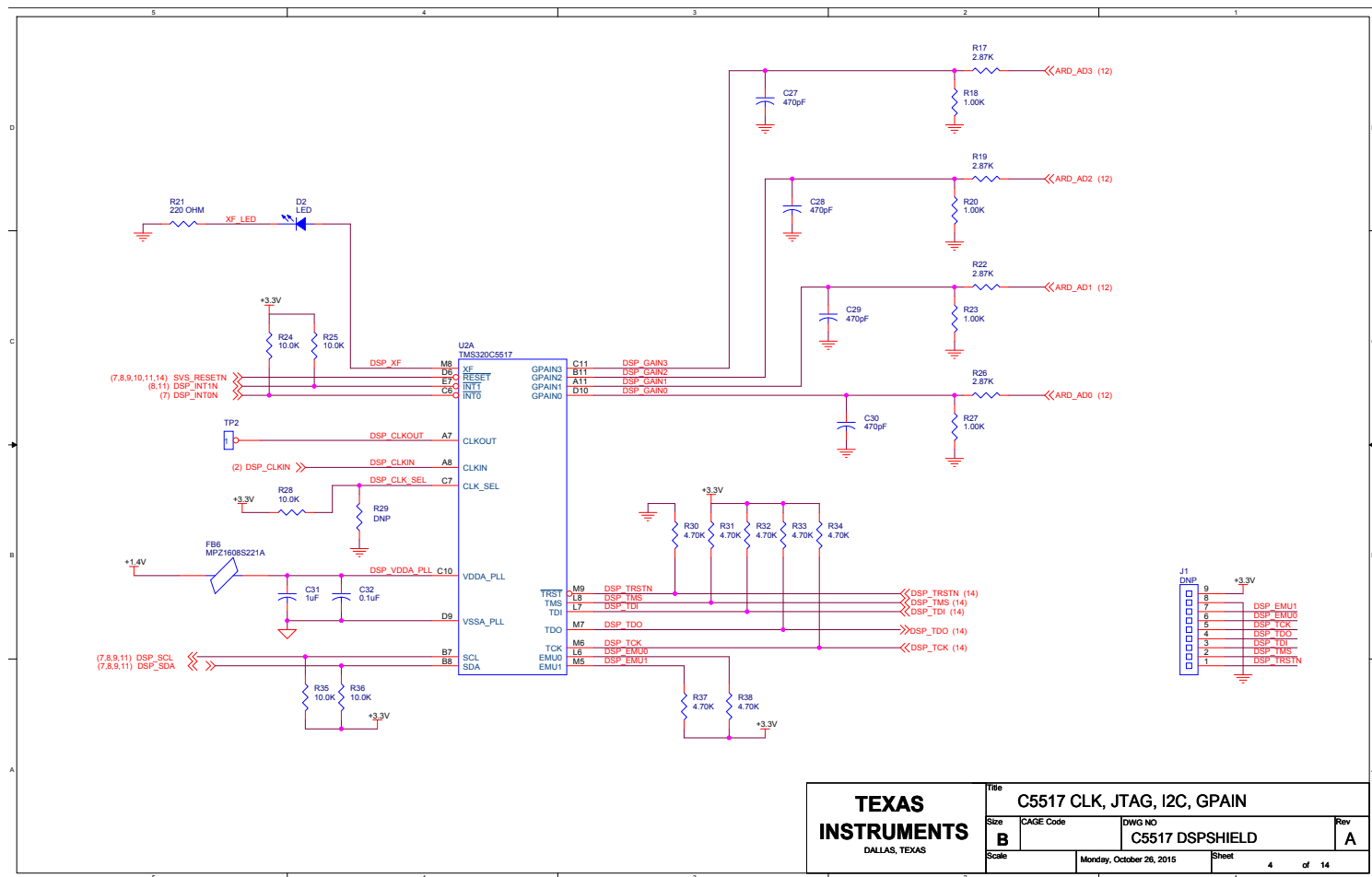
TEXAS INSTRUMENTS DALLAS, TEXAS	TEXAS INSTRUMENTS KILBY LABS 13550 North Central Expressway Dallas, TX 75243 MS 3747			
	ELECTRONIC SCHEMATIC DIAGRAM			
ENGR D. GARCIA	Size B	CAGE Code	DWG NO C5517 DSPSHIELD	Rev A
RIGHTS COMPLIANT YES	Scale	Monday, October 26, 2015		Sheet 1 of 14

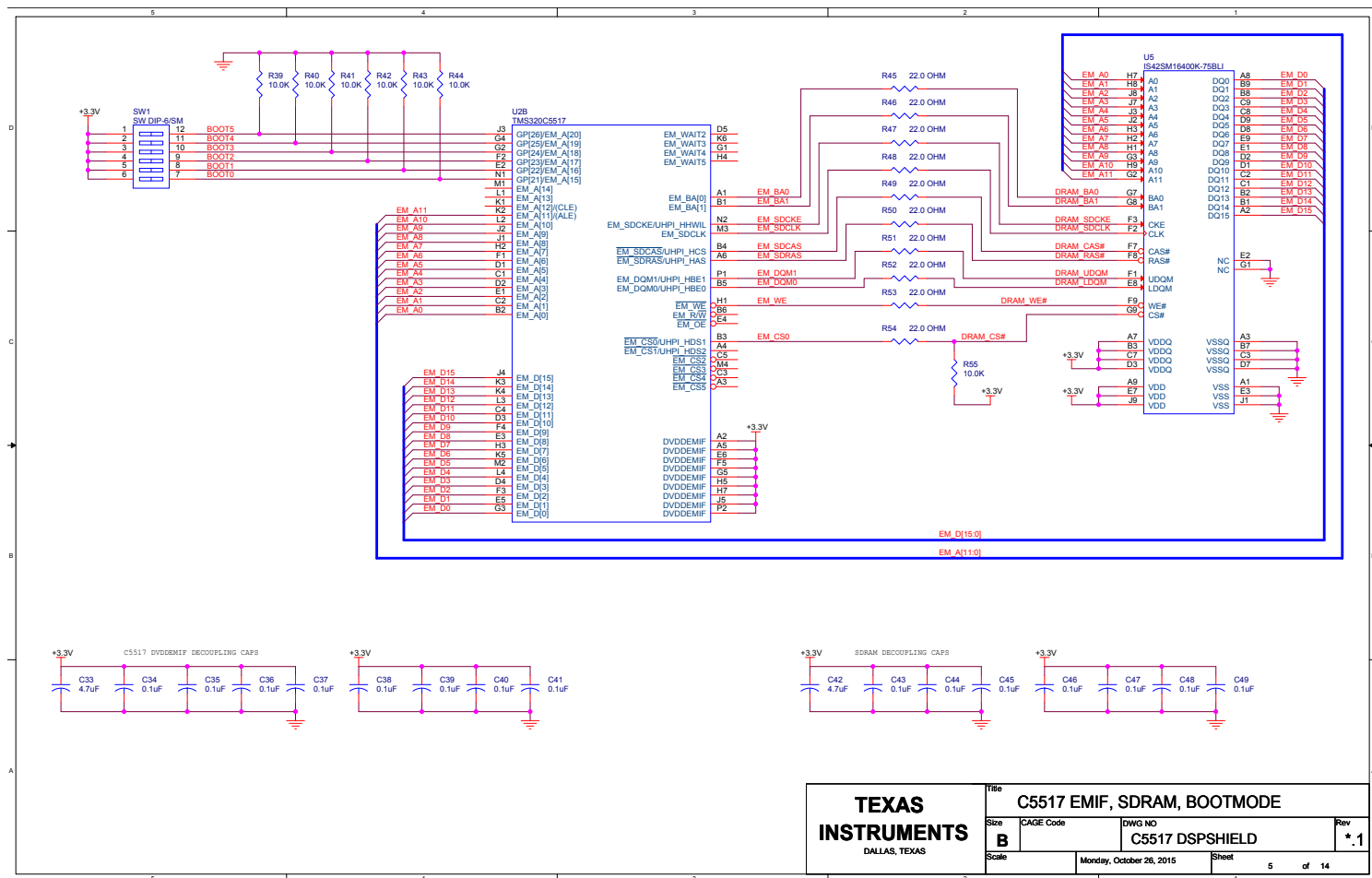


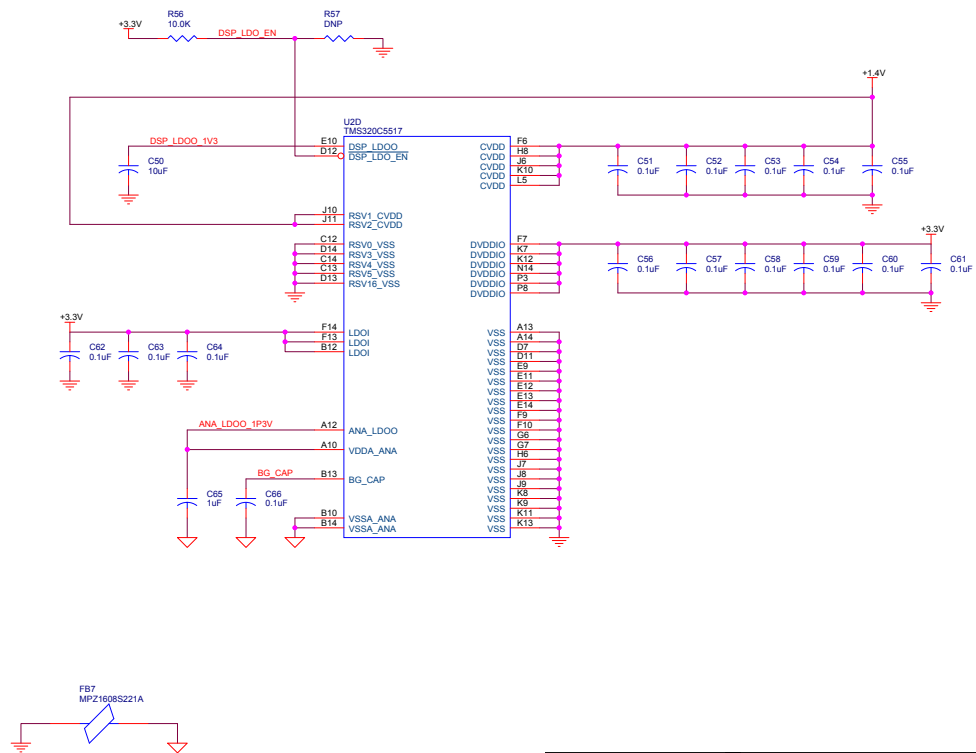
TEXAS INSTRUMENTS DALLAS, TEXAS		File C5517 USB, RTC, OSC		
		Size B	CAGE Code	DWG NO C5517 DSPSHIELD
		Scale	Monday, October 26, 2015	Sheet 2 of 14
				Rev A



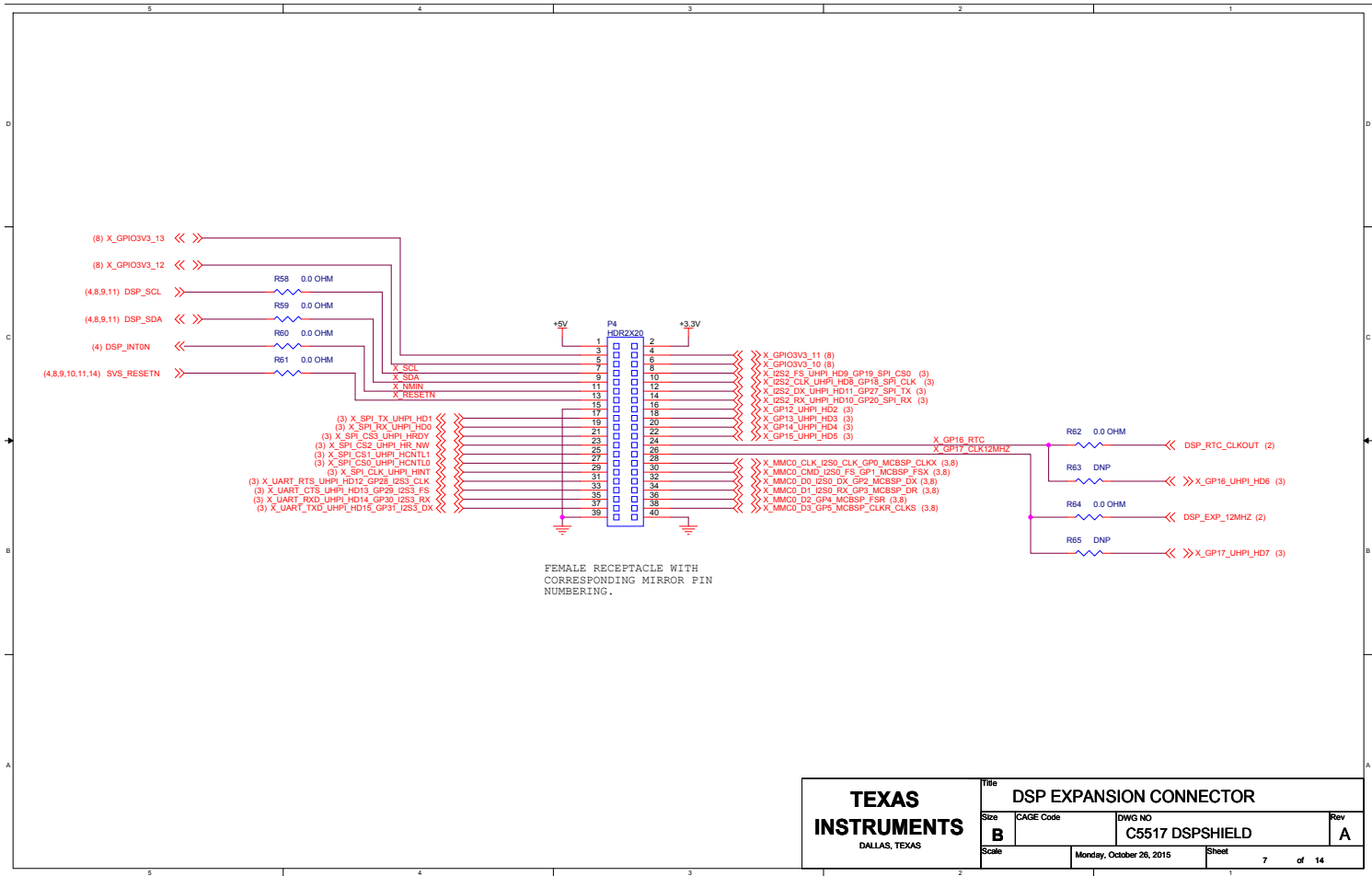
TEXAS INSTRUMENTS DALLAS, TEXAS		File C5517 PERIPHERALS		
		Size B	CAGE Code C5517 DSPSHIELD	Rev A
Scale		Monday, October 26, 2015		Sheet 3 of 14

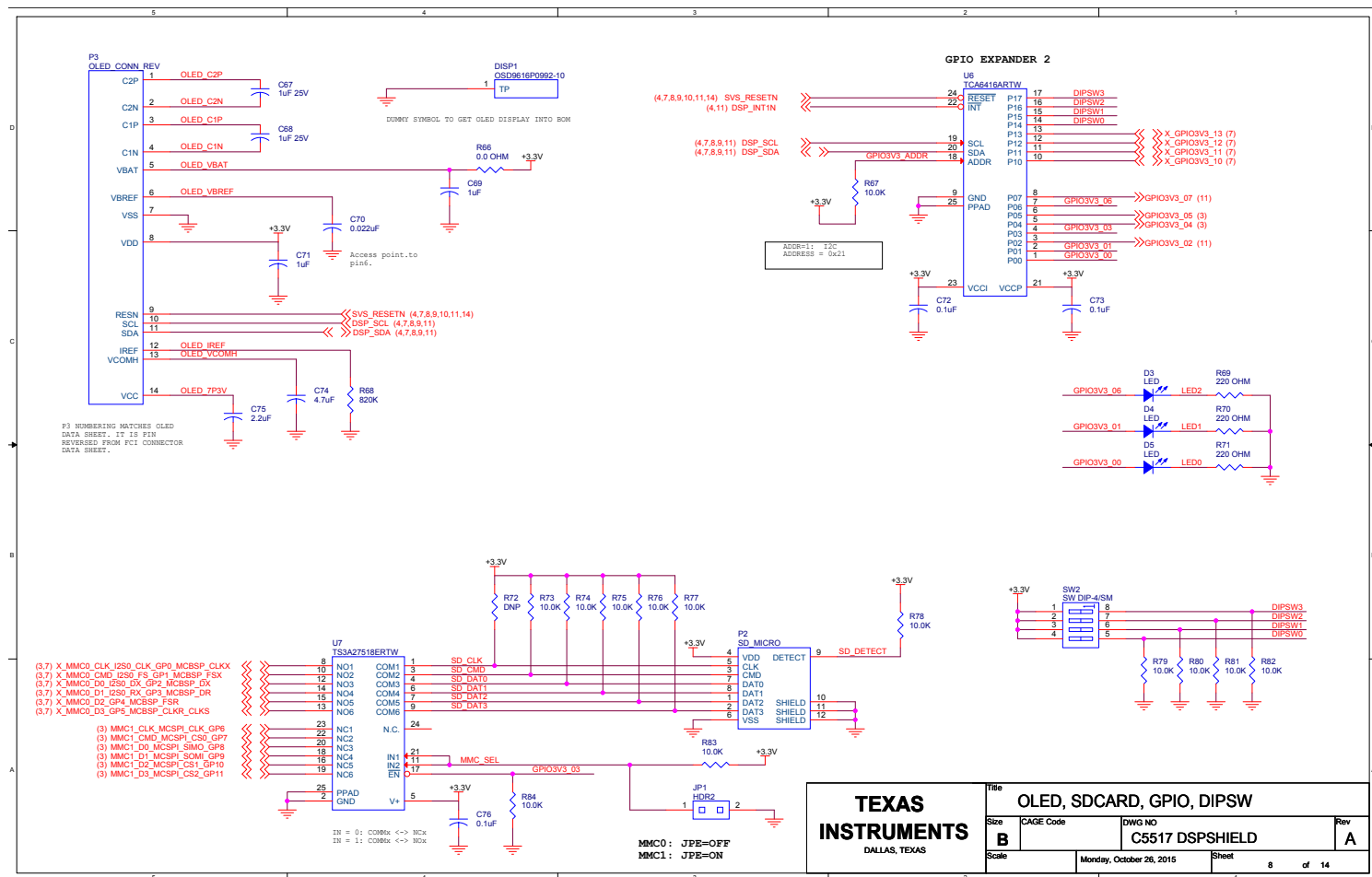


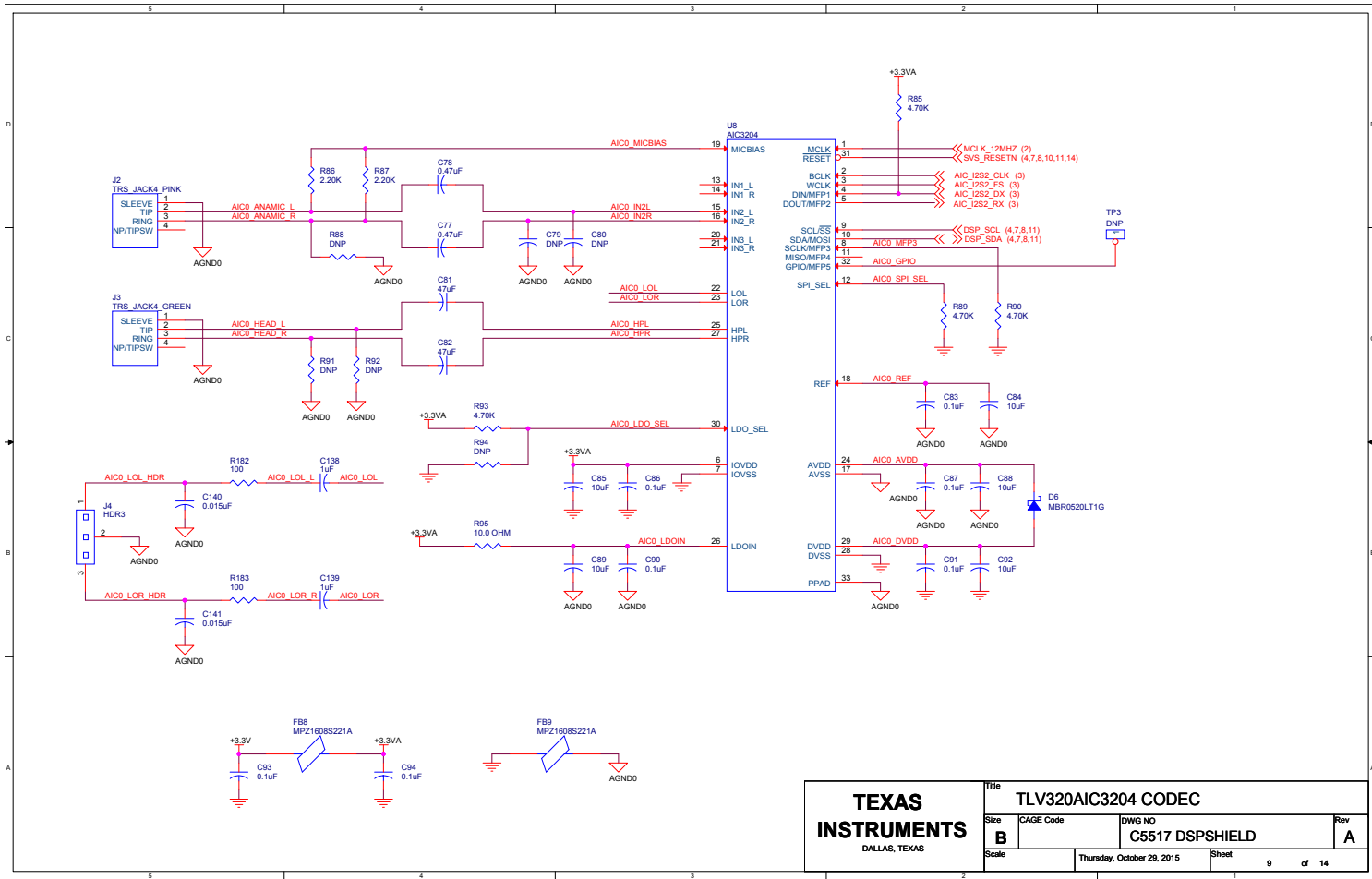




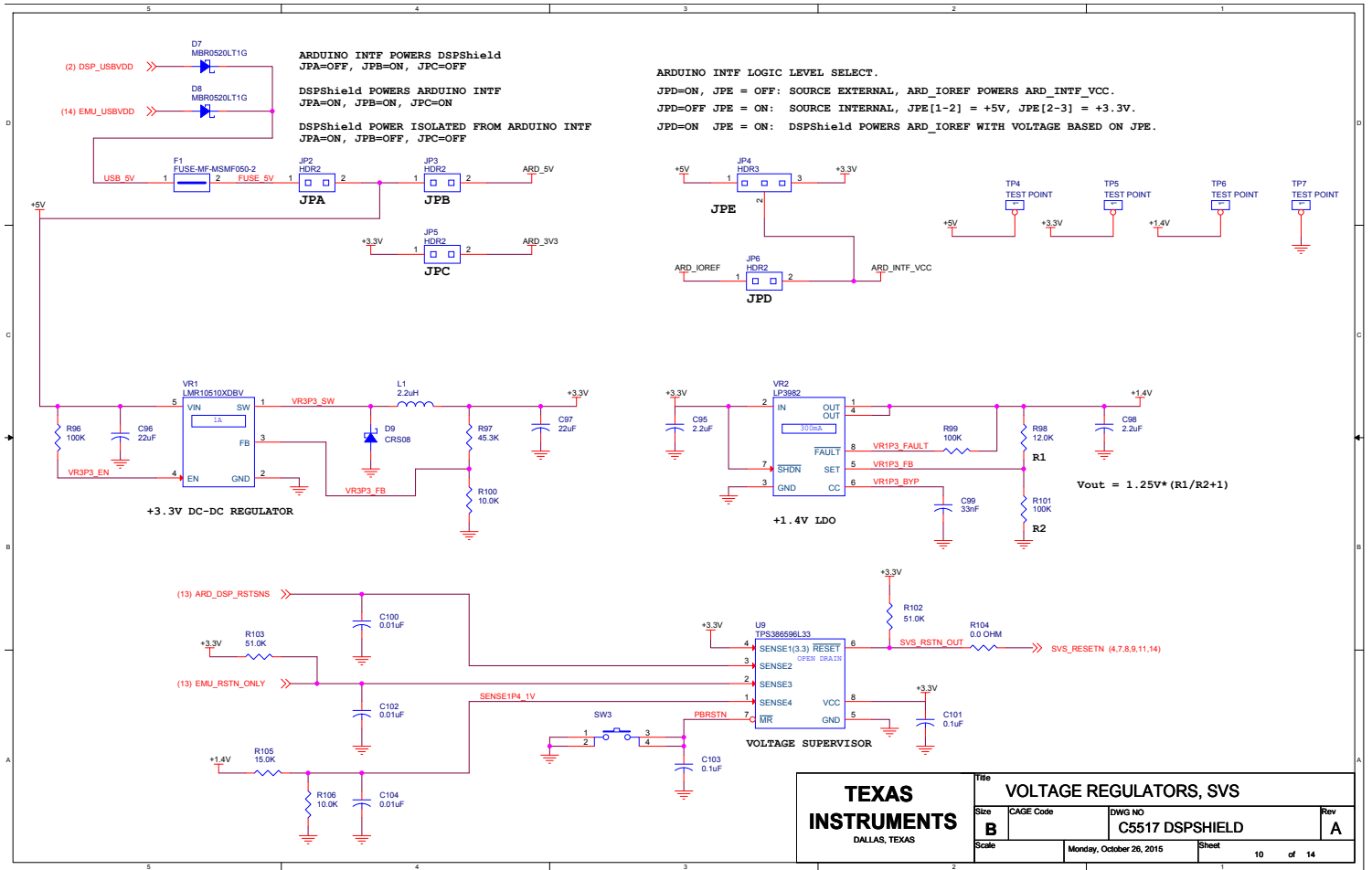
TEXAS INSTRUMENTS DALLAS, TEXAS	File C5517 CORE AND IO POWER		
	Size	CAGE Code	DWG NO
	B		C5517 DSPSHIELD
	Scale	Monday, October 26, 2015	Sheet 6 of 14

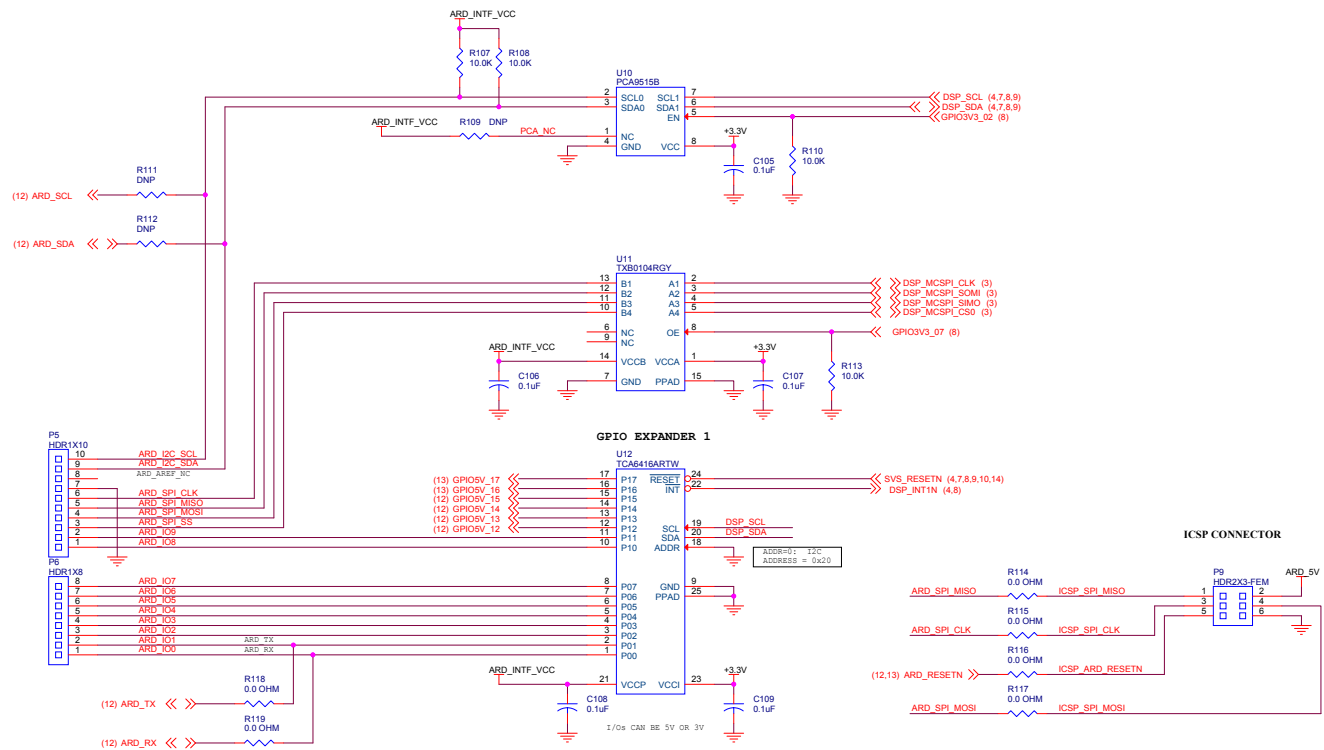




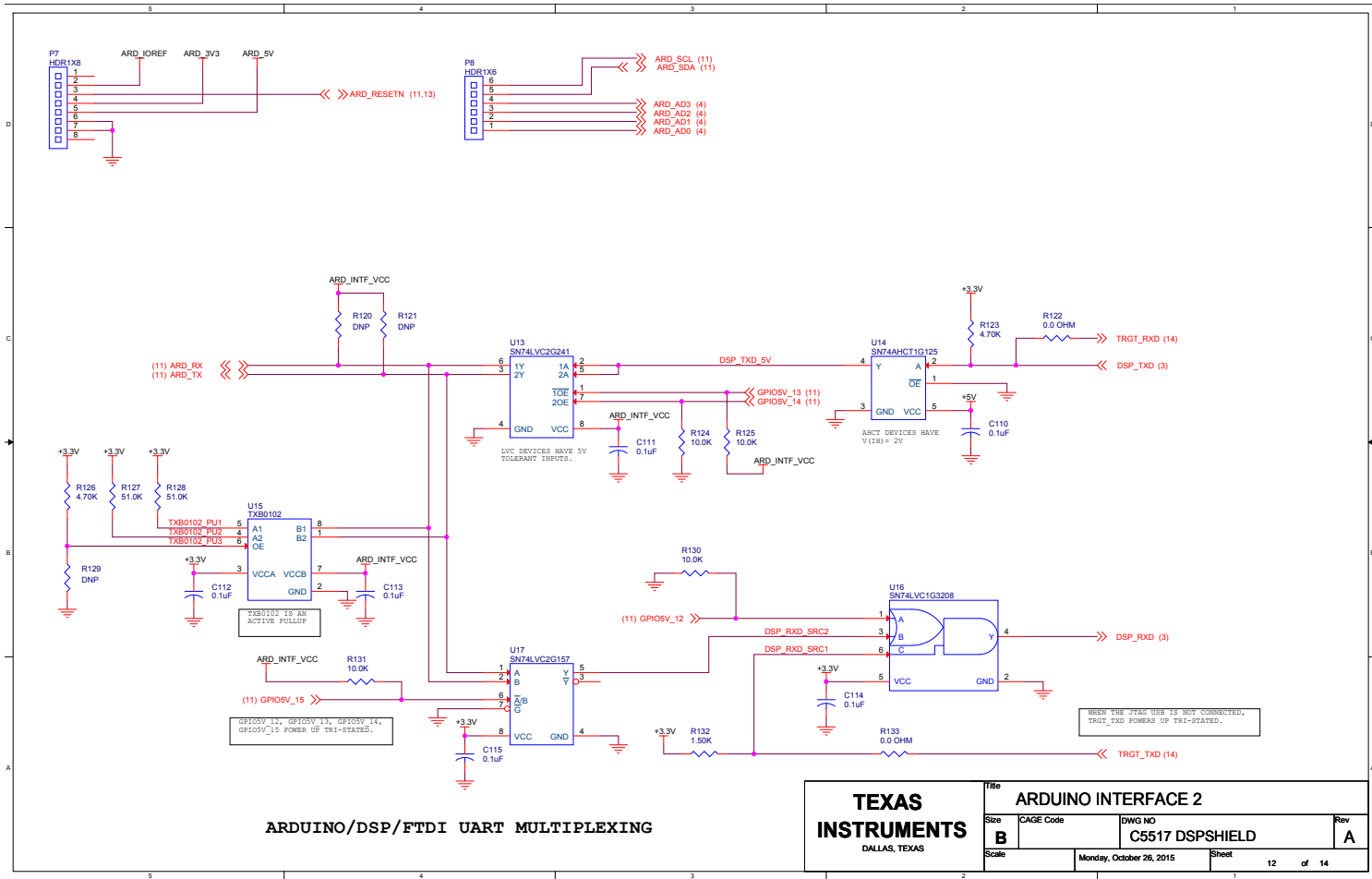


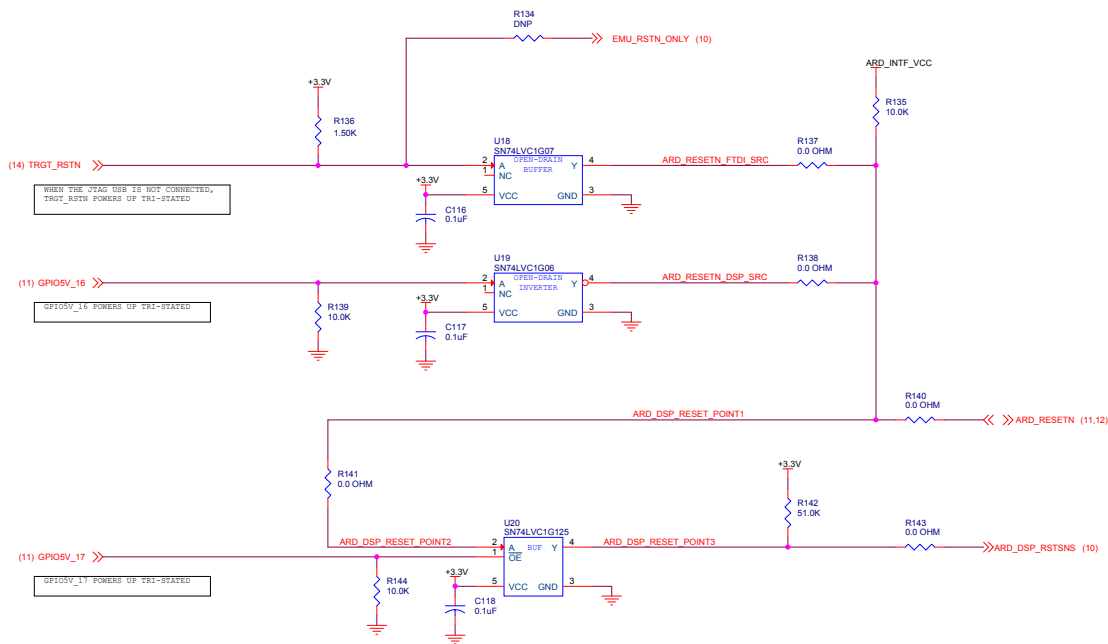
TEXAS INSTRUMENTS DALLAS, TEXAS		File TLV320AIC3204 CODEC		
		Size	CAGE Code	Rev
		B	C5517 DSPSHIELD	A
		Scale	Thursday, October 29, 2015	Sheet 9 of 14





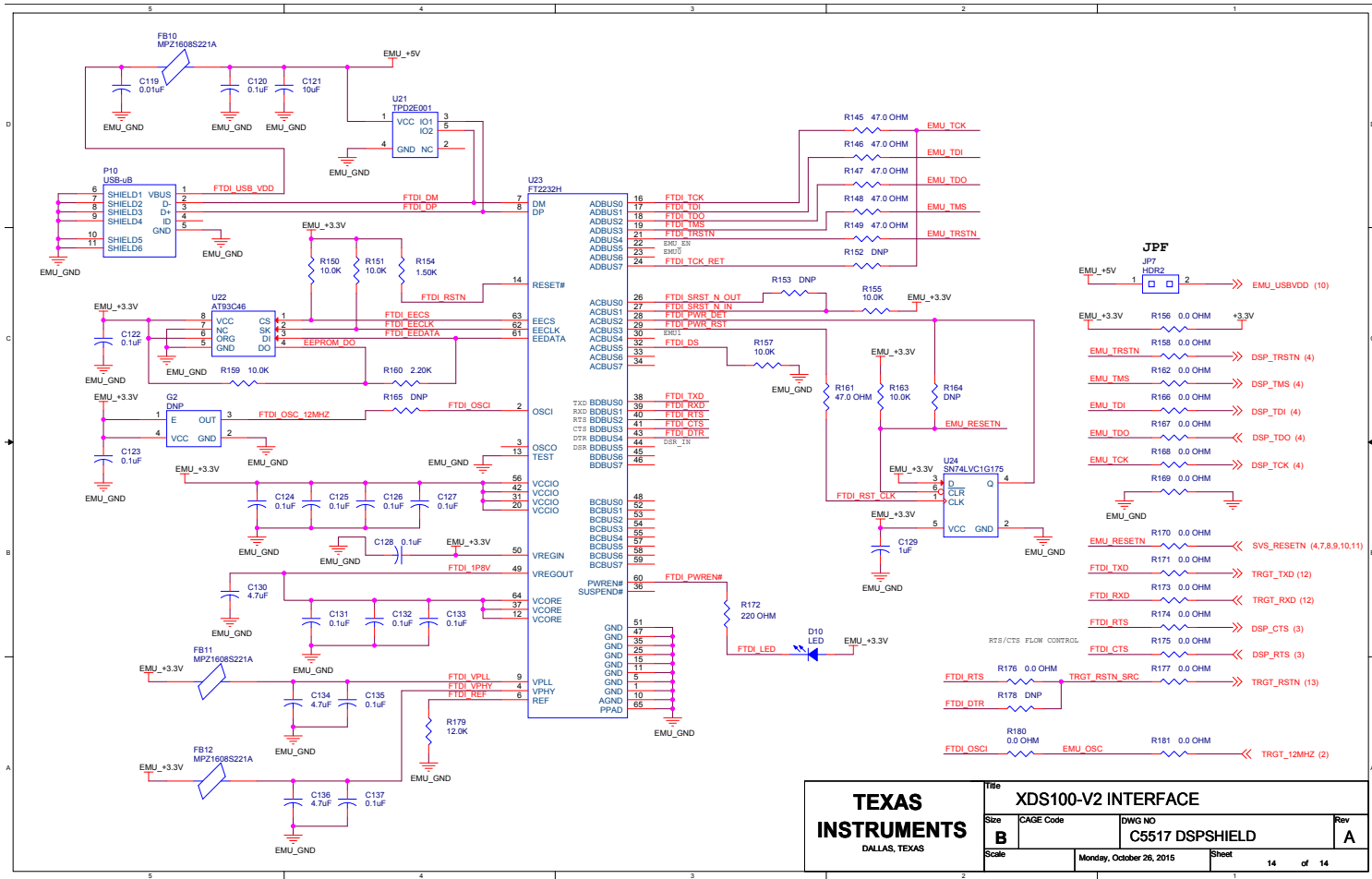
TEXAS INSTRUMENTS DALLAS, TEXAS		Title ARDUINO INTERFACE 1		
		Size B	CAGE Code	Rev A
		Scale	DWG NO C5517 DSPSHIELD	
		Monday, October 26, 2015	Sheet 11	of 14





ARDUINO/DSP/FTDI RESET

TEXAS INSTRUMENTS DALLAS, TEXAS	File ARDUINO INTERFACE 3			
	Size	CAGE Code	DWG NO	Rev
	B		C5517 DSPSHIELD	A
	Scale	Monday, October 26, 2015	Sheet	13 of 14



TEXAS INSTRUMENTS DALLAS, TEXAS		File XDS100-V2 INTERFACE	
		Size B	DWG NO C5517 DSPSHIELD
Scale 		Monday, October 26, 2015	Sheet 14 of 14

C5517 DSP *SHIELD*

ELECTRONIC SCHEMATIC DIAGRAM Revised: Thursday, November 05, 2015
C5517 DSPSHIELD Revision: A

TEXAS INSTRUMENTS
KILBY LABS
13560 North Central Expressway
Dallas, TX 75243
MS 3747

Bill Of Materials November 5, 2015 10:10:56

Ref #	Supplier PN	SUPPLIER	Des	Qty	PART	MFG	MFG P/N
1	445-7486-1-ND	DIGIKEY	CAPACITOR, 10V CERAMIC X5R 20% 603	13	10uF	TDK Corporation	C1608X5R1A106K
2	587-1994-2-ND	DIGIKEY	CAPACITOR, 16V CERAMIC X7R 10% 402	58	0.1uF	Taiyo Yuden	GMK105BJ104KV-F
3	399-7757-1-ND	DIGIKEY	CAPACITOR, 10V CERAMIC X7R 10% 402	3	1000pF	Kemet	C0402C102K8RACTU
4	445-1241-1-ND	DIGIKEY	CAPACITOR, 50V CERAMIC NPO 5% 402	2	33pF	TDK Corporation	C1005C0G1H330J050BA
5	587-1454-1-ND	DIGIKEY	CAPACITOR, 10V CERAMIC X5R 10% 402	16	1uF	Taiyo Yuden	LMK105BJ105KV-F
6	445-6843-1-ND	DIGIKEY	CAPACITOR, 10V CERAMIC X7R 10% 201	11	0.01uF	TDK Corporation	C0603X7R1A103K030BA
7	445-7318-1-ND	DIGIKEY	CAPACITOR, 10V CERAMIC X5R 10% 201	6	0.1uF	TDK Corporation	C0603X5R1A104K
8	587-1248-1-ND	DIGIKEY	CAPACITOR, 25V CERAMIC X5R 10% 603	2	1uF 25V	Taiyo Yuden	TMK107B1105KA-T
9	445-1261-1-ND	DIGIKEY	CAPACITOR, 25V CERAMIC, X7R 10% 402	1	0.022uF	TDK Corporation	C1005X7R1E223K
10	445-1385-1-ND	DIGIKEY	CAPACITOR, 16V CERAMIC 10% 1206	2	4.7uF	TDK Corporation	C3216X7R1C475K/1.60
11	490-1312-1-ND	DIGIKEY	CAPACITOR, 25V CERAMIC X7R 10% 402	5	0.01uF	Murata Electronics North America	GRM155R71E103KA01D
12	587-1250-1-ND	DIGIKEY	CAPACITOR, 16V CERAMIC X7R 10% 603	2	0.47uF	Taiyo Yuden	EMK107B7474KA-T
14	445-6711-1-ND	DIGIKEY	CAPACITOR, 10V CERAMIC X5R 20% 1210	2	47uF	TDK Corporation	C3225X5R1A476M
15	445-8190-1-ND	DIGIKEY	CAPACITOR, 10V CERAMIC X5R 10% 805	2	22uF	TDK Corporation	CGJ4J2X7R1A224K125AA
16	445-5166-1-ND	DIGIKEY	CAPACITOR, 6.3V CERAMIC X5R 10% 603	2	2.2uF	TDK Corporation	C1608X5R1A225K080AC
17	445-12238-1-ND	DIGIKEY	CAPACITOR, 16V CERAMIC X7R 10% 402	1	33nF	TDK Corporation	CGA2B2X7R1C333K050BA
18	490-5421-1-ND	DIGIKEY	CAPACITOR, 6.3V CERAMIC X5R 20% 603	3	4.7uF	Murata Electronics North America	GRM188R60J475ME19D
20	MMS25234B-FDICT-ND	DIGIKEY	DIODE ZENER 6.2V 500MW SOD-123 SOD-123-1N4148W_DIODES	1	6.2V	Diodes Inc	MMS25234B-7-F
21	160-1829-1-ND	DIGIKEY	DIODE, LED ORANGE 5mA 0603_D	5	LED	Lite-On Inc	LTST-C193FKT-5A
22	MBR0520LT1GOSCT-ND	DIGIKEY	DIODE, SCHOTTKY SOD123-MBR0520_ONSEMI	3	MBR0520LT1G	Semiconductor	MBR0520LT1G
23	CRS08QMCT-ND	DIGIKEY	DIODE, SCHOTTKY SOD123F_TOSHIBA	1	CRS08	Toshiba	CRS08(TER5LQ_M)
24	SDM40E20LS-FDICT-ND	DIGIKEY	DIODE, SCHOTTKY DUAL SERIES SOT-23-BAT54S_DIO	4	SDM40E20LS	Diodes Inc	SDM40E20LS-7-F
25	490-1014-1-ND	DIGIKEY	FERRITE BEAD 600 OHM 500mA 603	6	BLM18AG6015N1D	Murata Electronics North America	BLM18AG6015N1D
26	445-1565-1-ND	DIGIKEY	FERRITE BEAD 220 OHM 2A 603	5	MPZ160BS221A	TDK Corporation	MPZ160BS221A
27	MF-MSMF050-2CT-ND	DIGIKEY	FUSE, PTC RESETTABLE .50A 15V 1812	1	FUSE-MF-MSMF050-2	Bourns Inc	MF-MSMF050-2
28	535-9253-1-ND	DIGIKEY	OSCILLATOR, CRYSTAL CLOCK, 4-PIN CRY4-8002_SI	1	ASFL1-112MHZ	Abracon Corporation	ASFL1-12.000MHZ-EK-T
30	952-2262-ND	AVNET	HEADER, MALE 2POS .1" HDR2	5	HDR2	SAMTEC	TSW-102-07-T-S
31	TSW-103-07-T-S	AVNET	HEADER, MALE 3POS .1" HDR3	1	HDR3	SAMTEC	TSW-103-07-T-S
33	CP-3524SIPCT-ND	DIGIKEY	CONNECTOR, 3.5MM STEREO AUDIO JACK STEREOJACKDECAL	1	TRS JACK4_PINK	CUI Inc	SI-3524-SMT-PI
34	CP-3524SJGRCT-ND	DIGIKEY	CONNECTOR, 3.5MM STEREO AUDIO JACK STEREOJACKDECAL	1	TRS JACK4_GREEN	CUI Inc	SI-3524-SMT-GR
35	490-5114-1-ND	DIGIKEY	INDUCTOR, 1.3A 40MHz 20% IND2-1008CS_COIL	1	2.2uH	Murata Electronics North America	LQM2HPN2R2MGO1L
36	609-4613-1-ND	DIGIKEY	CONNECTOR, USB MICRO-B USB-UB-10118192_FCI	2	USB-uB	FCI	10118192-0001LF
37	101-00660-68-6-1-ND	DIGIKEY	CONNECTOR, SD MICRO CONN9-MICRO-SD-10100660XXX_AMP	1	SD_MICRO	Amphenol Commercial Products	101-00660-68-6
38	609-1237-1-ND	DIGIKEY	CONNECTOR, OLED DISPLAY CONN14-10051922_FCI_REV	1	OLED_CON14_REV	FCI	10051922-1410ELF
39	59200-ND	Avnet	HEADER, FEMALE 40 POS .1" 2ROW HDR40_REV_PINOUT	1	HDR2X20	SAMTEC	SSW-120-01-T-D
40	1528-1074-ND	Avnet	Arduino Header	1	HDR1X10	SAMTEC	SSQ-110-04-F-S
41	SSQ-108-04-F-S	Avnet	HEADER, FEMALE 8POS .1"	2	HDR1X8	SAMTEC	SSQ-108-04-T-S
42	SSQ-106-04-F-S	Avnet	HEADER, FEMALE 6POS .1"	1	HDR1X6	SAMTEC	SSQ-106-04-T-S
43	ESQ-103-24-T-D	AVNET	HEADER, FEMALE 6POS .1" 2ROW	1	HDR2X3_FEM	SAMTEC	ESQ-103-24-T-D
45	P0.0ICT-ND	DIGIKEY	RESISTOR, 1/10W JUMPER 402	32	0.0 OHM	Panasonic Electronic Components	ERJ-2GE0R00X
46	P10.0KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	43	10.0K	Panasonic Electronic Components	ERJ-2RKF1002X

47	P100KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	10	100K	Panasonic Electronic Components	ERJ-2RKF1003X
48	P4.70KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	11	4.70K	Panasonic Electronic Components	ERJ-2RKF4701X
49	P220LCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	5	220 OHM	Panasonic Electronic Components	ERJ-2RKF2200X
51	P0.0ACT-ND	DIGIKEY	RESISTOR, 1/8W 1% 805	15	0.0 OHM	Panasonic Electronic Components	ERJ-6GEY0R00V
53	P820KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	1	820K	Panasonic Electronic Components	ERJ-2RKF8203X
54	P100LCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	1	100 OHM	Panasonic Electronic Components	ERJ-2RKF1000X
55	P2.20KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	3	2.20K	Panasonic Electronic Components	ERJ-2RKF2201X
56	P10.0HCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 603	1	10.0 OHM	Panasonic Electronic Components	ERJ-3EF10R0V
57	P45.3KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	1	45.3K	Panasonic Electronic Components	ERJ-2RKF4532X
58	P4.02KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	1	4.02K	Panasonic Electronic Components	ERJ-2RKF4021X
59	P51.0KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	4	51.0K	Panasonic Electronic Components	ERJ-2RKF5102X
60	P15.0KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	1	15.0K	Panasonic Electronic Components	ERJ-2RKF1502X
61	P1.50KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	3	1.50K	Panasonic Electronic Components	ERJ-2RKF1501X
62	P47.0LCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	6	47.0 OHM	Panasonic Electronic Components	ERJ-2RKF4780X
63	P12.0KLCT-ND	DIGIKEY	RESISTOR, 1/10W 1% 402	1	12.0K	Panasonic Electronic Components	ERJ-2RKF1202X
64	CT2184PST-ND	DIGIKEY	SWITCH, DIP 4-POS SW 8PN	SW DIP-14/SM		Electrocomponents	218-4LPST
65	EG2527CT-ND	DIGIKEY	SWITCH, SPST MOMENTARY SW4-TL3301-GULL E-SW	SW MOMENTARY_ESWITCH		E-Switch	TL3301AF260QG
66	5002K-ND		TERMINAL, TEST POINT TP-5002_KEY	TEST POINT		Keystone Electronics	5002
85	AT93C46DN-SH-B-ND	DIGIKEY	I.C., 4K EEPROM SOIC8-8S1_ATM	1	AT93C46	Atmel	AT93C46DN-SH-B
86	768-1025-1-ND	DIGIKEY	I.C., USB TO UART BRIDGE PQFP64-QFN_FTDI	1	FT232H	FTDI, Future Technology Devices International	FT232HQ-REEL
90	728-1064-1-ND	DIGIKEY	CRYSTAL, CRYSTAL-SSPT7F_SII	1	32.768KHZ	Seiko Instruments	SSPT7F-12.5PF20-R
94	870-1425M16400K75BLI	Mouser	DRAM 64M 4Mx16 133Mhz 3.3V Mobile SDR	1	64MBIT Mobile SDRAM	ISSI	IS425M16400K-75BLI
68	296-39818-5-ND	DIGIKEY	IC DSP FIXED-POINT 196NFBGA	1	TMS320C5517	Texas Instruments	TMS320C5535AZHH10
69	296-12605-1-ND	DIGIKEY	I.C., 2 TO 1 MULTIPLEXER SOIC8-DCU_TI	2	SN74LVC2G157	Texas Instruments	SN74LVC2G157DCUR
70	296-16834-1-ND	DIGIKEY	I.C., 4-BIT 1 OF 2 MUX/DEMUX PQFP16-RGY_TI	2	SN74CB3Q3257	Texas Instruments	SN74CB3Q3257RGYR
71	296-24826-1-ND	DIGIKEY	I.C., 16-BIT I2C I/O EXPANDER QFN24-UF_LIN	2	TCA6416ARTW	Texas Instruments	TCA6416ARTWR
72	296-18012-1-ND	DIGIKEY	I.C., SINGLE BUFFER SOIC5-DRL_TI	1	DNI	Texas Instruments	SN74LVC1G125DRLR
73	296-23775-1-ND	DIGIKEY	I.C., STEREO AUDIO CODEC PQFP32-RHB_TI	1	AI3204	Texas Instruments	TLV320AIC3204RHB
74	296-27694-1-ND	DIGIKEY	I.C., QUAD RESET SUPERVISOR SOIC8-DGK_TI	1	TP5386596L33	Texas Instruments	TP5386596L33DGKT

75	296-30393-1-ND	DIGIKEY	I.C., I2C BUS REPEATER SOIC8-DGK_TI	1	PCA9515B	Texas Instruments	PCA9515BDGKR
76	296-21930-1-ND	DIGIKEY	I.C., 4-BIT VOLTAGE TRANSLATOR PQFP14-RGY_TI	1	TXB0104RGY	Texas Instruments	TXB0104RGYR
77	296-11936-1-ND	DIGIKEY	I.C., DUAL TRI-STATE BUFFER SOIC8-DCU_TI	1	SN74LVC2G241	Texas Instruments	SN74LVC2G241DCUR
78	296-19379-1-ND	DIGIKEY	I.C., SINGLE BUFFER SOIC5-DRL_TI	1	SN74AHCT1G125	Texas Instruments	SN74AHCT1G125DRLR
79	296-22862-1-ND	DIGIKEY	I.C., 2-BIT VOLTAGE TRANSLATOR SOIC8-DCU_TI	1	TXB0102	Texas Instruments	TXB0102DCUR
80	296-17845-1-ND	DIGIKEY	I.C., SINGLE 3-INPUT POSITIVE OR-AND GATE SOIC6-DCK_TI	1	SN74LVC1G3208	Texas Instruments	SN74LVC1G3208DCKR
81	296-18010-1-ND	DIGIKEY	I.C., SINGLE OPEN-DRAIN BUFFER SOIC5-DRL_TI	1	SN74LVC1G07	Texas Instruments	SN74LVC1G07DRLR
82	296-18009-1-ND	DIGIKEY	I.C., SINGLE OPEN-DRAIN INVERTER SOIC5-DRL_TI	1	SN74LVC1G06	Texas Instruments	SN74LVC1G06DRLR
83	296-18012-1-ND	DIGIKEY	I.C., SINGLE BUFFER SOIC5-DRL_TI	1	SN74LVC1G125	Texas Instruments	SN74LVC1G125DRLR
84	296-21883-1-ND	DIGIKEY	I.C., ESD PROTECTION SOIC5-DRL_TI	1	TPD2E001	Texas Instruments	TPD2E001DRLR
87	296-16998-1-ND	DIGIKEY	I.C., SINGLE D FLIP-FLOP SOIC6-DCK_TI	1	SN74LVC1G175	Texas Instruments	SN74LVC1G175DCKR
88	LMR10510XMF/NOPBCT-ND	DIGIKEY	I.C., 1A STEP-DOWN REGULATOR SOIC5-DBV_TI	1	LMR10510XDBV	Texas Instruments	LMR10510XMF/NOPB
89	LP3982IMM-ADJ/NOPBCT-ND	DIGIKEY	I.C., LDO LINEAR REG, 300mA SOIC8-DGK_TI	1	LP3982	Texas Instruments	LP3982IMM-ADJ/NOPB

Appendix D

Analog Shield Manual

Analog Shield: Introduction

By William J. Esposito

R0.3, 5/12/14

Introduction

The analog shield offers a high fidelity and easy way to connect your Arduino to analog circuits. In order to do this, the Digilent Analog Shield* provides:

- 4 Channel, 16 bit 100ks/s SAR ADC.
- 4 Channel 16 bit 100ks/s DAC.
- Variable +/-7.5V Supply.
- Fixed +/-5V Supply.
- A small area for a breadboard.
- Software read and write to the ADC/DAC with a single line of C.

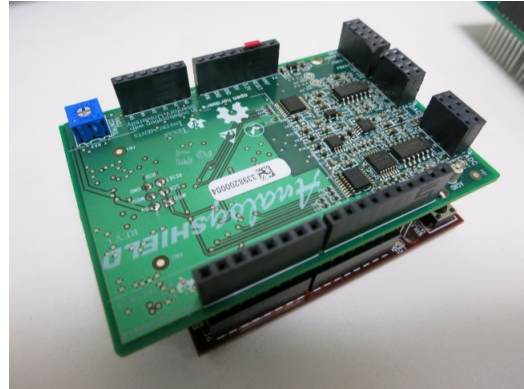


Figure 1: The Analog Shield.

In addition to providing many of the features available on other prototyping or breadboard shields, this shield provides substantially higher resolution inputs and outputs than an Arduino Uno does natively, as well as providing the bipolar inputs and supplies needed to drive many useful analog circuits.

The Analog shield communicates with the Arduino using the SPI protocol. The Analog-to-Digital input conversion (ADC) and Digital-to-Analog output conversion (DAC) both use the standard Arduino SPI bus pins and independent chip selects.

Applications

The Analog Shield allows access to many signals that cannot reliably be recorded or generated using the standard pins on the Arduino.

A few demos using the analog shield have been developed.

- Sine wave / function generator (using direct digital synthesis)
- FFT spectrum analyzer
- Polyphonic music generator
- 4 channel low bandwidth oscilloscope
- Lissajou diagram generator
- XY-analog scope display driver

These demos are each explained in their own associated documentation

*The Analog Shield was developed as a collaboration by Texas Instruments and the Kovacs / Giovangrandi lab at Stanford University.

Hardware

Expansion Header

The left hand edge of the Analog Shield contains a three pin dual row female headers. The first columns on the connectors contain inputs and outputs of all of the analog shield's peripherals. All of the connections on the second column are referenced to a shared ground.

Analog Inputs

The Analog Shield provides 4 channels, of 16 bit of Analog-to-Digital input conversion (ADC). As described below, to sample one of the analog input requires only a single line function call:

```
analog.read(channel);
```

Analog read takes a channel as an input and returns a 16 bit unsigned integer.

These inputs are buffered with an op-amp to protect the circuitry of the ADC, and a simplified circuit diagram of the inputs has been printed directly on the board.

Analog Outputs

As well as inputs, the Analog Shield has 4 channels of Digital-to-Analog output conversion (DAC) for analog output. The outputs of the Analog Shield function slightly differently from the Analog outputs of the Arduino itself. The standard Arduino `analogWrite(pin, value)` function initiates a Pulse Width Modulated square wave that only approximates the desired output value over time. While this behavior is suitable for many applications, there are some analog circuits where it is important to have an actual stable output value. The DAC on the Analog Shield serves this purpose nicely. Values written to these ports remain stable until a new value is written to the selected port. As described below, in order to write to one of the analog output channels on the expansion connector (labeled OUT0-OUT4), one simply has to call:

```
analog.write(pin, value);
```

The `analog.write()` function takes an unsigned 16-bit unsigned integer as an input.

As with the inputs, the outputs are buffered with an op-amp to protect the circuitry of the DAC, and a simplified circuit diagram of the inputs has been printed directly on the board.

Power Supply

In addition to the inputs and outputs, the Analog Shield also includes an integrated power supply. Unlike the Arduino's included power supply, the supply on the Analog shield is bipolar – meaning that it provides both positive and negative voltages.

The onboard power supply includes five outputs, +5V, -5V, +V_{adj}, and -V_{adj}, and 2.5V V_{ref}.

In addition, to the fixed +/-5V supply pins, The Analog Shield also includes adjustable +/-7.5V pins. The voltage of these pins is controlled using the potentiometer located in the lower right corner of the Analog Shield. Adjusting the potentiometer will adjust the voltage of the $+V_{\text{adj}}$ supply and the $-V_{\text{adj}}$ supply simultaneously, keeping the two supplies balanced at the same voltage (but with the opposite polarity). For example, one might adjust the potentiometer to produce +3.3V on the $+V_{\text{adj}}$ pin, which would produce an output of -3.3V on the $-V_{\text{adj}}$ pin and thus a potential between the two pins of 6.6V.

Finally, a +2.5V reference output is provided to offer a 'center' point between 0 and 5 volts, in case a single rail (+5V to 0V supply) circuit needs a center voltage reference point.

Prototype Area

The center of the board is intentionally left open. It includes a silkscreened drawing that provides a high level schematic of one channel of the ADC and one of the DAC, as mentioned above. This area has been left free of any top mounted components, and a miniature breadboard has been included with the shield, to be affixed in this area, for use in constructing circuits.

Software

Included with the Analog Shield is a simple driver library. It attempts to provide a similar interface to the existing Analog read and write functionality. To install the driver library see the document “**Analog Shield – 02 First Time Setup**”. To include the library in a sketch, once it is installed, it is simply added to a sketch with:

```
#include <analogShield.h>
```

Once the shield library is included, the sketch will have access to a variable called **analog**, which has access to all the methods associated with the shield. The following are the six most important methods associated with the shield.

```
unsigned int read(int channel, bool mode = false);
```

This method samples the selected analog input. 'mode' is an optional parameter. **False** (the default value) is normal single ended mode where each analog pin is referenced to ground. **True** enables 'differential' mode, where the value returned is the difference between the voltages on pairs of adjacent inputs (i.e. IN0-IN1 and IN2-IN3). Read returns an unsigned integer.

```
signedRead(int channel, bool mode = false);
```

Signed read behaves identically to read, but returns a signed integer.

```
write(int channel, unsigned int value);
write(unsigned int value0, unsigned int value1, true);
write(unsigned int value0, unsigned int value1, unsigned int value2, true);
write(unsigned int value0, unsigned int value1, unsigned int value2,
      unsigned int value3, true);
```

These methods allow writing to the Analog Shield's analog output. The first method writes to any arbitrary selected channel. The next three methods write to two or more channels simultaneously (but only write to specific channels, i.e. 0 and 1 instead of to any arbitrary pair of channels). The simultaneous update methods are useful for higher speed applications where a small delay between channel outputs will produce unwanted signals or timing issues. For example, to use the shield to drive the X-Y input of an oscilloscope to draw a circle, a delay between writing the X and the Y channels produces a ghost 'ellipse' drawn during the interim between the X and Y DAC updates.

The power supply is entirely analog and as such needs no software to operate, but the ADC (part number DAC8564) and DAC (part number ADS8343) are both SPI devices. Their datasheets are freely available from the manufacturers website and can be used to write your own software as need be, but this simple library should do the trick for most users.

Summary:

Unlike the native Analog input of the Arduino, the Analog shield provides a 16 bit ADC, (as opposed to 10 bits provided by the Arduino UNO) offering greater precision (about 25dB) than the Arduino does natively. Additionally, the ADC works as a bipolar input, meaning that sense signals in the range of +/-5V can be measured without any additional hardware.

The onboard DAC provides a similar improvement in precision. The Analog Shield provides 16 bits of precision, and offers as much as 25dB better signal to noise ratio when driving a sinusoidal output at low frequency.

The analog shield provides a compact integrated power supply with fixed and adjustable outputs to allow a variety of analog circuits to be assembled with a minimum of wasted space and complexity.

Special thanks to Gregory Kovacs, Fernando Mujica, and Clint Cole and their respective teams at Stanford, TI, and Digilent for supporting this project to completion, as well as Domingo Garcia for his amazing work on board layout and to Laurent Giovangrandi for his support in program oversight and debugging.

Appendix E

DSP Shield HAL Key Functions

DSP-Duino

Energia Based DSP programming API

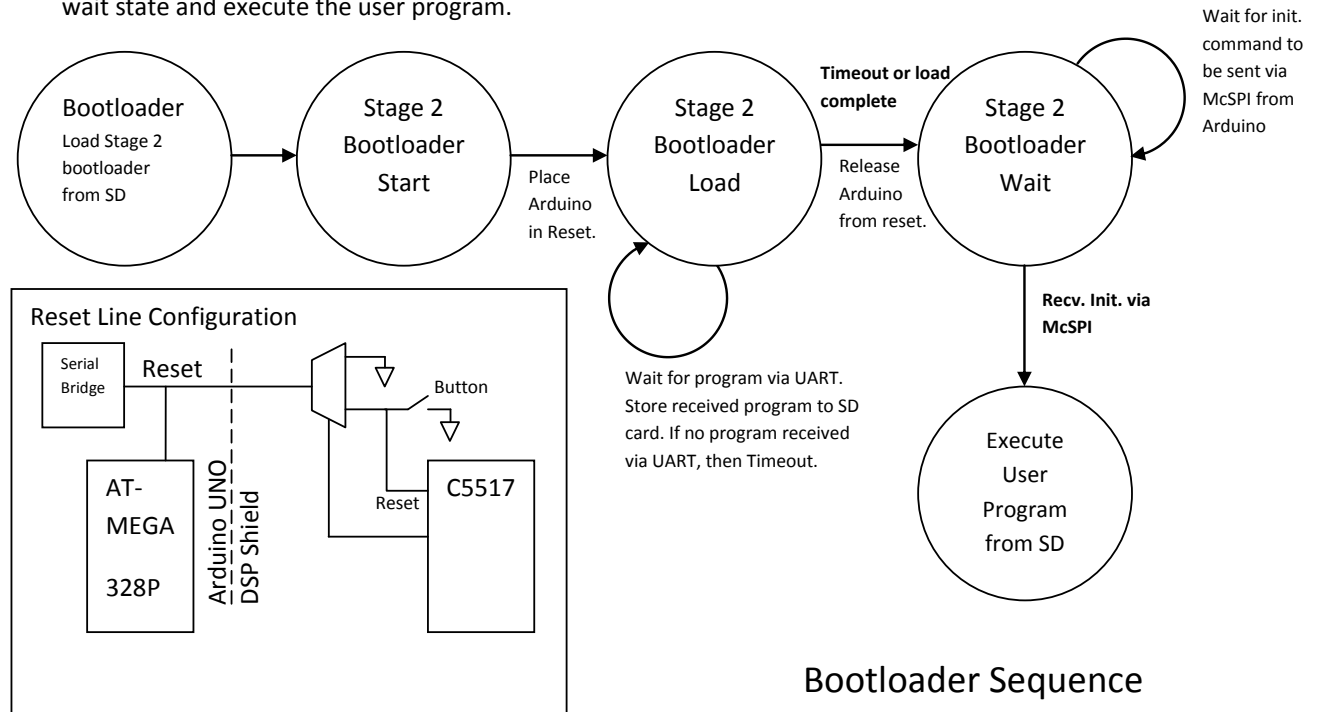
The goal for this project is to port the Energia IDE / “language” to the new C5517 based DSP shield. This shield will contain a number of peripherals that should be interfaced in a way that mimics the Arduino hardware as naturally and closely as feasible. To that end, the API will include a number of directives that wrap hardware specific functionality with simple APIs and mnemonics that simply mirror the pin names on the board itself. The API will additionally contain a number of higher level functions that allow for easy use of powerful signal processing methods, at the cost of some detailed control.

Bootloader:

The DSP Shield will boot directly from the SD card, in both “shield” mode and “stand-alone” mode, negating any need for bootloader dip switches.

On boot, the DSP Shield will load a “Stage 2” bootloader from the SD card. This bootloader will disconnect the 5517’s reset pin from the shared reset line. It will then keep the Arduino in reset and look for a handshake and new program to be transmitted via the UART, which it will then store to the SD card, overwriting any existing user program. Once the program has been loaded, or a timeout has been reached, the bootloader will finish and enter a wait state. At this time it will release the Arduino from reset and reconnect its reset pin to the shared reset line.

When the stage 2 bootloader receives an initialization signal from the Arduino via McSPI, it will leave the wait state and execute the user program.



Pin Names:

The pin names should correspond to the Arduino pin names as closely as possible. Pins 0-13 should be numbered as such. It is not clear if pins 10-13 will be accessible as they will be utilized for SPI, but their numbers must be reserved as if they were available. These pins will correspond to the pins on the I²C GPIO device attached to the shield, not to the pins on the DSP itself.

0-9: GPIO on I2C chip

10-13: Superseded by SPI, but reserved

D0-DXX: GPIO Pins on TI Header

A0-A6: Analog Input Pins corresponding to Arduino Header. Only 4 active at a time.

Low Level (Hardware Directives):

Included here is a preliminary list of functionality that should be available at launch for the HAL. These functions are either in the form of libraries invoked by the user or built directly in to the programming environment. Built in functionality is noted by an asterisk in parenthesis (*) after the section name.

Analog Read(*): Reads analog value from one of the '6' analog pins. Note that three of them are MUXed together, so only four pins can be read at the same time. Given that we are going to hide multi-pin reads from the end user (and allow only reading one pin at a time), this should not be a problem.

```
analogRead(pin);
```

Digital Read/Write(*): Reads or writes an appropriate digital value to the assigned pin. This should work for ANY GPIO pin on the final device.

```
digitalRead(pin);  
digitalWrite(pin);
```

Serial(*): Serial communication over the UART should work exactly like the arduino environment for the HW UART. A separate software serial library exists to allow serial communication over GPIO. Software serial can likely be left to be implemented by the community.

Importantly, hardware serial communication is integral to Arduino sketches, and as such must be available to users from the start.

<code>if (Serial);</code>	Returns '1' if the serial port is ready to be used, '0' otherwise.
<code>Serial.available();</code>	Returns the number of bytes available for reading on the UART.
<code>Serial.begin(baudrate);</code>	Initializes serial port.
<code>Serial.end();</code>	Closes serial port, tristates pins. Useful on Arduino so that pins can be reused. Since we will share pins with arduino GPIO, this holds on the shield as well.

Additional Serial Functions (descriptions on Arduino Playground):

Serial.find(); Serial.findUntil(); Serial.flush(); Serial.parseFloat();	Serial.parseInt(); Serial.peek(); Serial.print(); Serial.println();	Serial.read(); Serial.readBytes(); Serial.readBytesUntil(); Serial.setTimeout(); Serial.write();
--	--	--

Attach / Detach Interrupt(*): Wrappers to enable / disable interrupts for specific functionality with a single line of code. See Arduino Playground

```
attachInterrupt(interrupt, function, mode);
```

```
detachInterrupt(interrupt);
```

SD Library: the Arduino library for interfacing with an SD card. We should mimic the interface as closely as possible. The SD Library contains the SD class and the File class, for handling files on FAT16/32 formatted SD cards.

SD:

begin();	exists();	mkdir();
open();	remove();	rmdir();

Begin(cspin) – start SD library and communication with card. Cspin is an optional parameter that allows the user to select which GPIO pin is used for chip select on the SD card. This will be hard wired in our case, so the optional parameter can be unsupported.

Additional function descriptions on Arduino Playground.

File:

available()	close()	flush()	peek()
position()	print()	println()	seek()
size()	read()	write()	isDirectory()
openNextFile()	rewindDirectory()		

Timers: The Arduino Uno has one 16 bit timer available to the user, as well as two 8-bit timers. The Timer1 library is commonly used to handle timer interrupts. The following function descriptions are taken from Arduino Playground:

initialize(period) ;

You must call this method first to use any of the other methods. You can optionally specify the timer's period here (in microseconds), by default it is set at 1 second. Note that this breaks analogWrite() for digital pins 9 and 10 on Arduino.

setPeriod(period) ;

Sets the period in microseconds. The minimum period or highest frequency this library supports is 1 microsecond or 1 MHz. The

maximum period is 8388480 microseconds or about 8.3 seconds. Note that setting the period will change the attached interrupt and both pwm outputs' frequencies and duty cycles simultaneously.

pwm(pin, duty, period);

Generates a PWM waveform on the specified pin. Output pins for Timer1 are PORTB pins 1 and 2, so you have to choose between these two, anything else is ignored. On Arduino, these are digital pins 9 and 10, so those aliases also work. Output pins for Timer3 are from PORTE and correspond to 2,3 & 5 on the Arduino Mega. The duty cycle is specified as a 10 bit value, so anything between 0 and 1023. Note that you can optionally set the period with this function if you include a value in microseconds as the last parameter when you call it.

attachInterrupt(function, period);

Calls a function at the specified interval in microseconds. Be careful about trying to execute too complicated of an interrupt at too high of a frequency, or the CPU may never enter the main loop and your program will 'lock up'. Note that you can optionally set the period with this function if you include a value in microseconds as the last parameter when you call it.

setPwmDuty(pin, duty);

A fast shortcut for setting the pwm duty for a given pin if you have already set it up by calling pwm() earlier. This avoids the overhead of enabling pwm mode for the pin, setting the data direction register, checking for optional period adjustments etc. that are mandatory when you call pwm().

detachInterrupt();

Disables the attached interrupt.

disablePwm(pin);

Turns PWM off for the specified pin so you can use that pin for something else.

read();

Reads the time since last rollover in microseconds.

The method for interfacing with the timer is typically a traditional interrupt methodology and the timer1 library is likely to be reasonably portable. This is not an integral library for booting the board, but it will be a high profile / demand library.

Audio: This will be a new set of calls *defined by TI* to enable easy handling of data streams to and from either the analog pins or from the Audio codec. We can develop a few function calls based on an "Audio" class. Essentially the model will be:

init(): initialize 3204 codec / initialize analog pin.

attachInterrupt(): Interrupt on available frame.

detachInterrupt(): Cancel Interrupt on available frame.

read (): Read a stereo frame.

write (): Write a stereo frame.

~~frameSize()~~: Set the size of the frame, possibly. Alternative: Fixed size frame.

Codec configuration needs to be defined as well, but the calls are not yet proposed here.

SPI (Wire Library): Write to peripheral using dedicated pins on shield, as SPI master to emulate behavior of Arduino Wire library. From Arduino playground:

<code>begin();</code> <code>requestFrom();</code> <code>beginTransaction();</code>	<code>endTransmission();</code> <code>write();</code> <code>available();</code>	<code>read();</code> <code>onReceive();</code> <code>onRequest();</code>
--	---	--

SPI (Slave – to Arduino) (*): Since the DSP Shield is a client device for the Arduino, the system must be able to accept commands from the master as needed. As such, a routine must be established for interrupting on new input over the McSPI, and reacting to that data appropriately.

USB (Client Device): There is no equivalent standard library for Arduino. We will develop our own simplified system calls. This is a lower priority functionality that can be added in later or even left to the community (if they beat us to the punch).

OLED Screen (LCD):

<code>autoscroll();</code> <code>begin();</code> <code>clear();</code> <code>clear(int);</code> <code>display();</code>	<code>flip();</code> <code>init();</code> <code>noAutoscroll();</code> <code>noDisplay();</code> <code>OLED();</code>	<code>print(char[]);</code> <code>printchar(unsigned char);</code> <code>scrollDisplayLeft();</code> <code>scrollDisplayRight();</code>	<code>setline(int);</code> <code>setOrientation(int);</code> <code>setRolling(int, int);</code> <code>write(unsigned char);</code>
---	---	--	---

High Level (DSP Software):

FFT Class. (“Signal” class?):

Perform an FFT on an “Audio” stream and/or generic data array

`FFT();`

`IFFT();`

`filter(signal, taps);` Filter a signal based on filter and taps.

`rate(signal, rate_new);` Rate Conversion

More to come.

Appendix F

Instructables

Analog Shield Application Note 1: Direct Digital Synthesis Function Generator

By William J. Esposito, Doctoral Candidate and Gregory T.A. Kovacs, Professor of Electrical Engineering, Stanford University
R0.7, 5/29/14

Overview

The Analog Shield Direct Digital Synthesis (DDS) Function Generator is a compact, low-frequency function generator capable of producing sine, square, triangle or sawtooth waveforms (*figure 1*).

DDS works by means of “playing back” a prerecorded waveform stored in a lookup table at a precisely controlled rate. One full period of the waveform is stored in an array on the Arduino. To create a wave output, the Arduino sends a single sample of the stored waveform out via the analog shield. It moves through the table outputting samples until it reaches the last sample in the array. Once it reaches the end, the program returns to the beginning of the table and repeats. In order to put out a waveform at a higher frequency, the program skips steps in the table. For example, if a step of 1 produces a 1 Hz output, then outputting every 10th sample (a step of 10) would produce a 10 Hz output. By calculating how many samples it has to skip, it can precisely generate any frequency it needs, up to half the rate at which it puts out samples (the Nyquist rate, where there are a minimum of two samples per waveform period, which assumes a sine... more samples per period are necessary for more complex waveforms).

Hardware

The Analog Shield DDS is built around the Arduino and uses the TI/Stanford Analog Shield* (available from Digilent, *figure 2*) for signal output. The Analog Shield DDS also uses the Adafruit RGB LCD Shield kit for its user interface.

The Arduino UNO R3 Can be found at:

http://store.arduino.cc/index.php?main_page=product_info&products_id=195

The Adafruit RGB LCD Kit can be found at:

<https://www.adafruit.com/products/714>

The Analog Shield can be found at:

<http://www.digilentinc.com/analogshield>

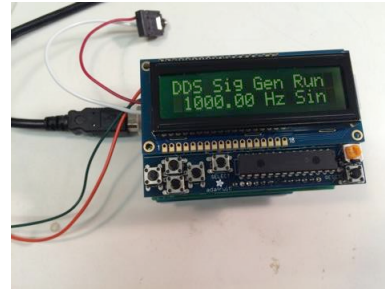


Figure 1: DDS Function Generator.

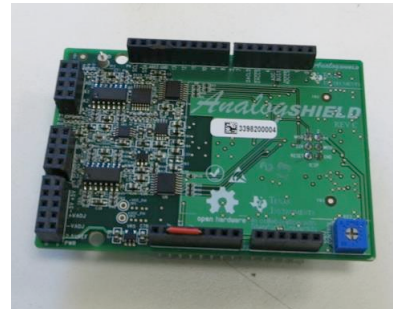


Figure 2: Analog Shield.

In order to get a meaningful signal at more than a

*The Analog Shield was developed as a collaboration between Texas Instruments and the Kovacs / Giovangrandi lab at Stanford University.

kilohertz, the DDS also needs a “reconstruction” filter on its output. The reconstruction filter is discussed in its own section, below, but the quality of the reconstruction filter will make a big difference in the quality of the signal produced

Building The Demo

In order to build the Analog Shield DDS, four nonstandard Arduino libraries are necessary. For a guide on Arduino libraries and how to install them, go to <http://arduino.cc/en/Guide/Libraries>

The libraries required for the DDS are:

The Adafruit RGB-LCD Shield and MCP23017 libraries (For the LCD)

<https://github.com/adafruit/Adafruit-RGB-LCD-Shield-Library>

The TimerOne library (for the DDS sample clock)

<https://code.google.com/p/arduino-timerone/>

The Analog Shield library (for analog input)

<http://github.com/wespo/analogshield>

In case these libraries move in the future, the Adafruit LCD libraries are linked from the Adafruit Touch LCD Shield tutorial at

<https://learn.adafruit.com/rgb-lcd-shield/>

The code for the DDS is available with this document at

<http://www.digilentinc.com/analogshield>.

A detailed discussion is included below, in the ‘How the Code Works’ section of this document.

Once these libraries are downloaded and unzipped in the Arduino/libraries folder, open the Arduino IDE and the “DDS_Generator” sketch. It is important to note that if the Arduino IDE was already open before the libraries were added, the IDE will need to be restarted to recognize the libraries.

Stack the LCD and Analog Shield on your Arduino. Next, click the “Upload” arrow in the top left to upload the DDS Function Generator to the Arduino.

Once the upload is complete, you will see a menu will appear on the LCD.

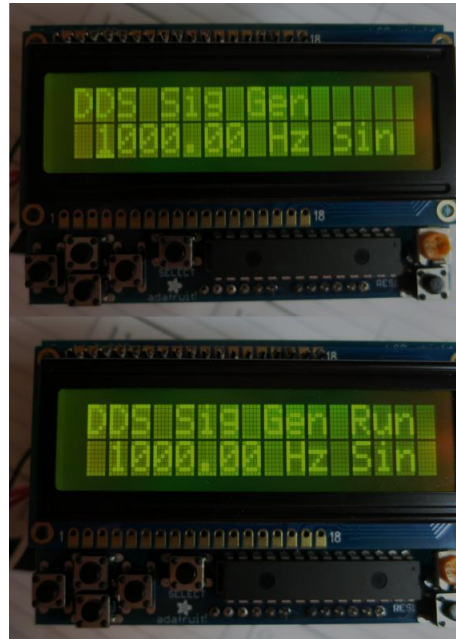


Figure 3: LCD with cursor and DDS running.

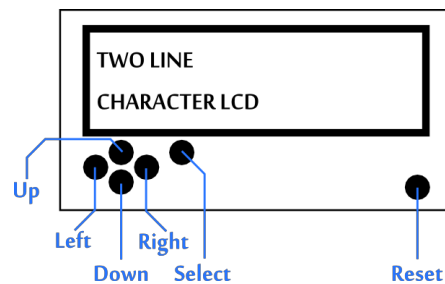


Figure 4: DDS Controls.

Controlling the Demo

At startup, the Arduino waits for the user to set the frequency and wave output type.

To change the frequency, press up and down. The left and right buttons move the cursor and selects which digit the up and down buttons effect (*figure 4*). Moving the cursor past the lowest digit allows selection of the wave shape (sine, square, sawtooth, triangle).

Once a timeout (5 seconds) occurs or the user presses 'select' the system will output the desired wave. When running, the word 'run' displays in the upper right corner of the LCD (*figure 3*).

To set a new waveform, press reset to return to the menu.

Output

Now that there is a signal coming out of D0 on the analog shield, the next step is to do something with the signal.

One option is to directly connect the output (D0) to an oscilloscope (as well as a ground from the shield to the ground on your oscilloscope probe, to complete the circuit).

With a direct output, an oscilloscope will show a fairly clear waveform at low frequencies (up to 1 kHz or so, *figure 5*), although it will have a somewhat stair-step nature. At higher frequencies, the waveform will break down. This is due to the nature of the DDS. At higher frequencies, the DDS skips more steps, and each step will therefore make a larger vertical jump. At a high enough frequency, the signal begins to look like a rolling square wave. To allow the DDS to work at higher frequencies a reconstruction filter is needed, which will be addressed below.

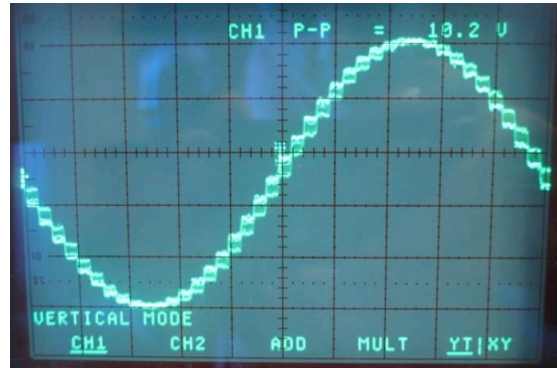


Figure 5: 1kHz sine wave with no reconstruction filter.

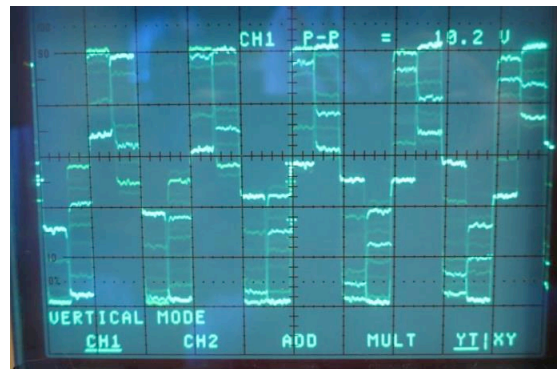


Figure 6: 10kHz sine wave with no reconstruction filter.

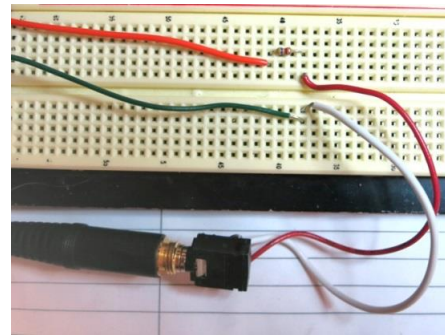


Figure 7: Headphone jack output. Orange wire (top) to D0, green wire (bottom) to the ground on the analog shield.

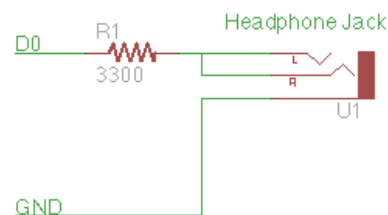


Figure 8: Headphone jack connector circuit.

If an oscilloscope isn't handy, the output can be heard with headphones. One advantage of this is that most headphones naturally act as a low pass filter with a cutoff at or below 20 kHz. To connect your headphones you'll need a **headphone jack and a resistor with a value of at least 3.3 k Ω** . The resistor is critical because it will limit the volume put out through the headphones, saving your ears and your headphones from damage. *Figures 7 and 8* show this circuit.

Reconstruction Filter

To improve the DDS, a reconstruction filter can be added.

The purpose of the reconstruction filter is to remove the stair-step nature of our sampled sine wave. *Figure 9* shows the relatively low frequency 1kHz output after passing through the high quality filter that will be discussed in a moment. Note the drastic improvement in this output when compared to the same wave seen without a filter in *figure 5*.

The reconstruction filter should be a low pass filter which removes the sharp (40 kHz high frequency) steps in the output without weakening the lower frequency (1 Hz – 16 kHz) signal that is the actual intended output.

The simplest filter that can be implemented is a passive RC (resistor-capacitor) low pass filter. Such a filter will improve the signal, but its simplicity means that it will only weakly attenuate higher frequencies. This means that the resulting signal is less of an improvement when compared to a better filter.

To make a simple RC Filter, connect a resistor of 2 k Ω between D0 on the analog shield and our output, and a capacitor of 22 nanofarads (nF) between our output and ground as shown in *figure 11*.

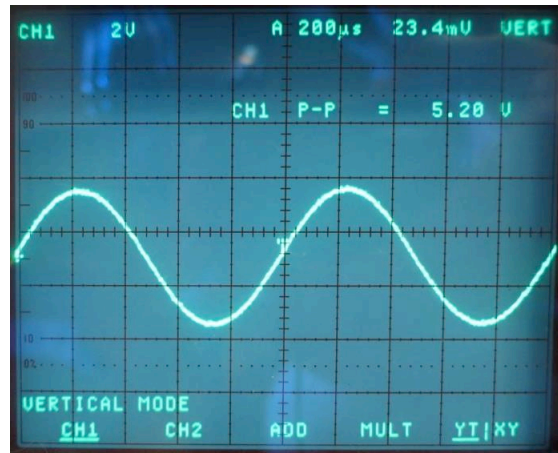


Figure 9: 1kHz sine through high quality filter.

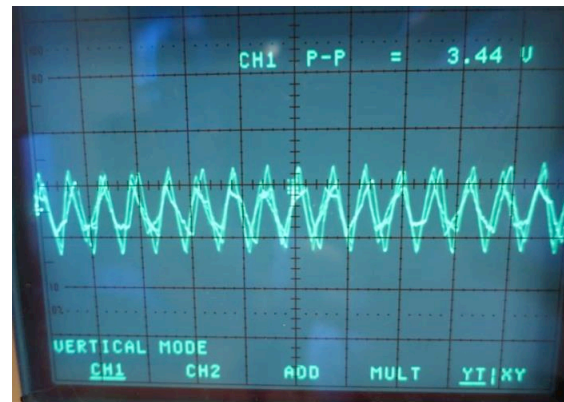


Figure 10: 10kHz signal through RC filter, with the same scale as figure 9.

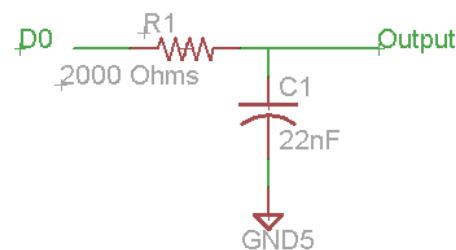


Figure 11: Simple RC filter.

This will create a filter with a 'cutoff' (half power) point at:

$$f_{cutoff} = \frac{1}{2 * \pi * 2000 * .000000022}$$

$$f_{cutoff} = 3617Hz$$

This will help our resulting signal by attenuating high frequency noise. It will not provide optimal performance, however, as the signal of interest is often above 10 kHz and will be attenuated as well. *Figure 10* shows the 10 kHz signal as observed through this RC filter. Note that it is closer to a sine wave than *figure 5* above, but is still a poor quality output.

To maximize the frequency that can be reproduced with the DDS, a complicated 8th order active low pass filter can be implemented. Such a filter uses operational amplifiers (op-amps) to allow numerous successive stages of high frequency removal without a loss in signal strength. *Figure 12* shows the same signal as *figure 10*, passed through one of these high quality filters.

A full filter design has been attached to this document for an 8th order filter with an 18 kHz cutoff that will allow for a clear DDS output up to a 16 kHz sine wave. The design for this filter has been included as figure 17 at the end of this document.

There is a caveat, however. Sometimes, the desired output waveform needs to have sharp edges. In an extreme example, the DDS can output a square wave, which has the sharpest possible edge. By passing that signal through the reconstruction filter, the output waveform would be reduced to an approximation of the sine wave output at 10 kHz (see *figures 14 and 15*). This is due to the nature of a square wave itself, whose sharp edges are the result of harmonics which are attenuated through the reconstruction filter.

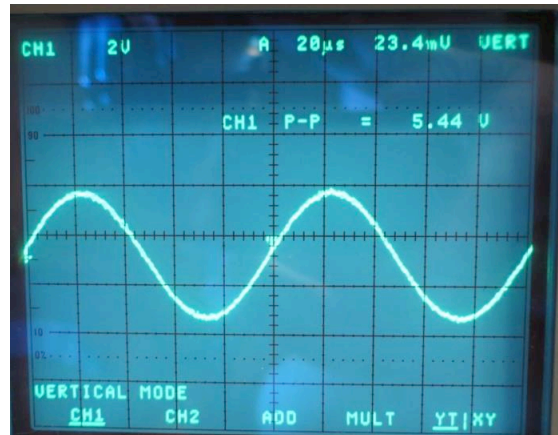


Figure 12: The same 10kHz sine wave through an 8th order reconstruction filter, with an 18kHz cutoff.

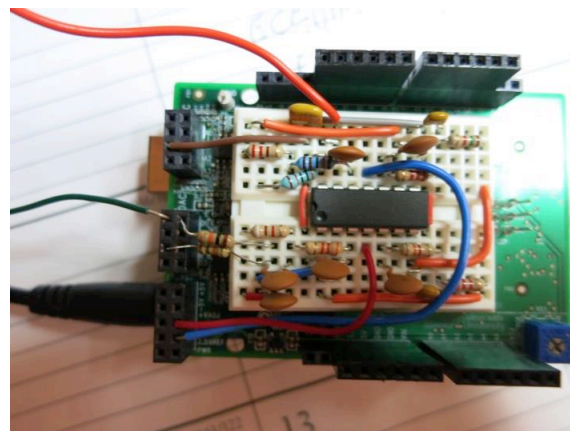


Figure 13: 8th order reconstruction filter with an 18kHz cutoff constructed on the prototyping area of the Analog Shield.



Figure 14: 10 kHz square wave with no reconstruction filter

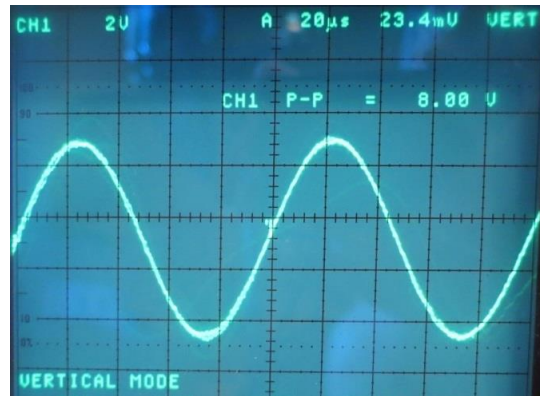


Figure 15: 10 kHz square wave through 8th order reconstruction filter with an 18kHz cutoff.

How the Code Works

On the Analog Shield, the DDS has been implemented in a very standard configuration.

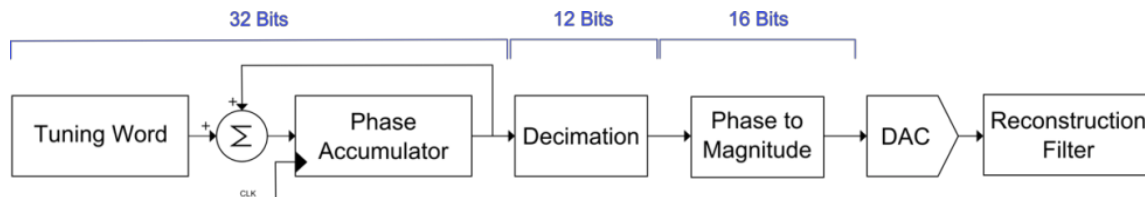


Figure 16: DDS Software block diagram

The DDS stores a phase offset, which represents a position in a hypothetical infinite waveform. Each sample cycle, the DDS routine increments that phase offset by its stride size (tuning word). Since only one period of the waveform is actually stored in memory, the DDS computes the offset with respect to a single wave cycle and uses that as a table lookup. The resulting value is then sent to the DAC for conversion to a voltage output. By storing the phase in a higher precision word than the table, the program avoids having to perform complicated carry offset operations every time it generates a sample (*figure 16*).

The actual DDS routine is moderately complex as it supports multiple waveforms and controls the LCD and menu system. A shortened version of the main code (attached below) shows how the block diagram can be simply implemented with careful coding.

```

/* Direct Digital Synthesis */
//12 bit long (4096 length) sine wave table
PROGMEM prog_uint16_t isinTable16[] = {<sine wave table omitted for space>};
#include <analogShield.h> //Analog Shield
#include <avr/pgmspace.h> //Sine Wave Storage
#include <TimerOne.h> //Timer Control

//DDS variables
volatile unsigned long tuningWord = 0;
volatile unsigned long phaseAccumulator = 0; //32 bits
bool toggle = false;
void setup() {}

void loop() { //takes user input and the goes into DDS forever.
    Timer1.initialize(16); //Setup the timer for DDS
    Timer1.attachInterrupt(dds); //start DDS
    while (1) {}; //wait forever. Reset Arduino for new frequency.
}

//DDS code
void dds() { //Direct Digital Synthesis Lives Here.
    unsigned int value; //value to output on DAC
    unsigned long tempPhase; //calculated phase for this sample

    //Increment the phase accumulator by the value stored in the tuning word.
    phaseAccumulator += tuningWord; //32 bits
    //use only top 12 bits of 32 bit phase accumulator
    tempPhase = (unsigned long)(phaseAccumulator >> 20);
    //Get Sine Value
    value = pgm_read_word(isinTable16 + tempPhase);
    //write the result to the output.
    analog.write(0, value);
}

```

Additional Caveats About the DDS

The waveform itself of a DDS is band limited by the sample clock used to drive the digital to analog converter. In the case of the Analog Shield DDS, the sample clock runs around 40,000 samples/second. This means that a sharper edge (as found in the square and sawtooth waveform) can't be accurately reproduced. This phenomenon results in apparent jitter on the sharp edges of the waveform as they appear to move back and forth in time by one sample.

Notes

[1] Maxim Semiconductor has produced a whitepaper, MT-085 which explains DDS much more rigorously. It can be found at:

<http://www.analog.com/static/imported-files/tutorials/MT-085.pdf>

Disclaimer:

This code and circuit was developed by William Esposito, Ph.D. Candidate in Electrical Engineering, Stanford University, in the Kovacs/Giovangrandi Laboratory in collaboration with Texas Instruments, Incorporated. All code herein is free and open source, but is provided as-is with no warranties implied or provided. Use of this code and the associated documentation is at the user's own risk.

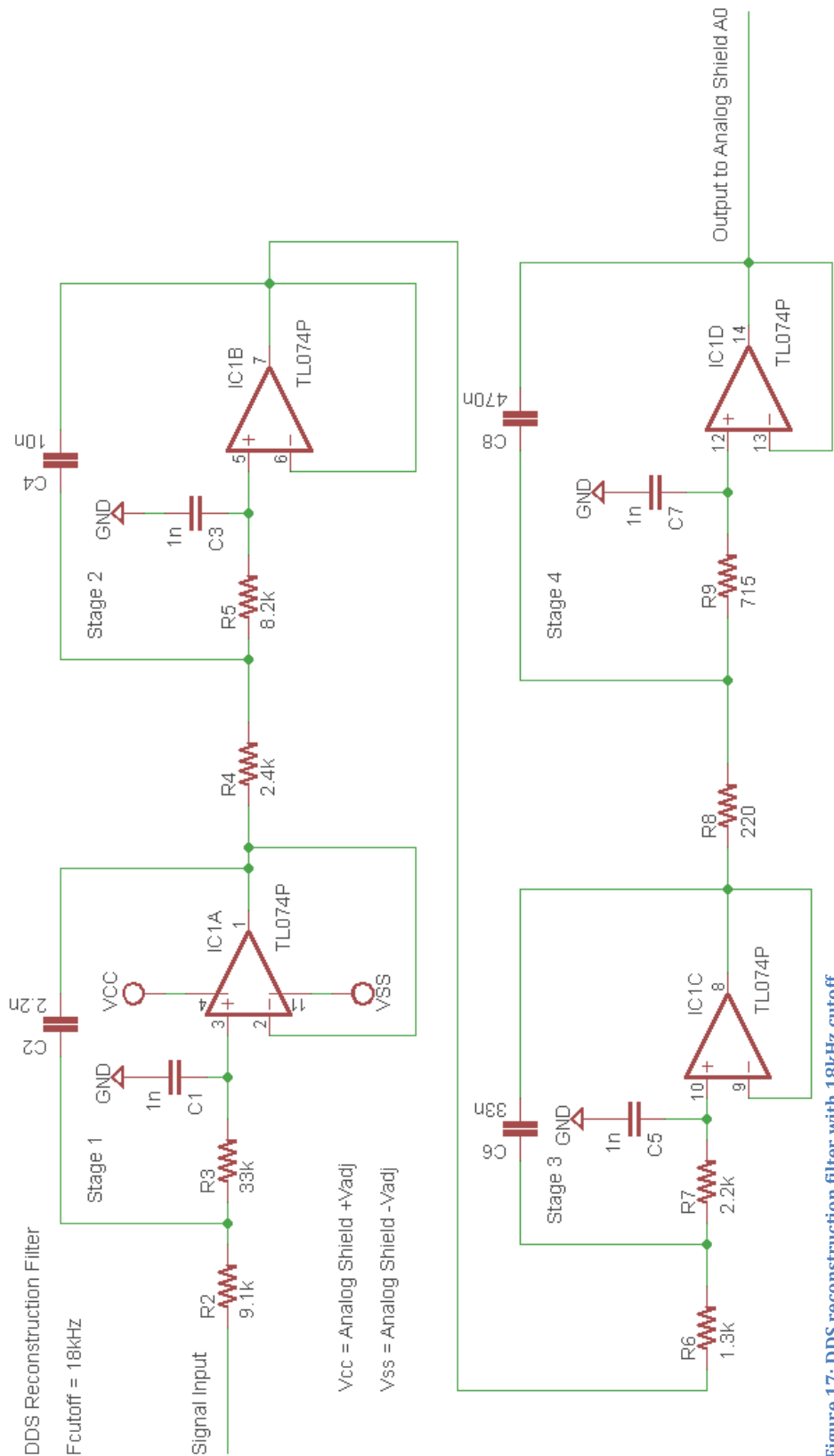


Figure 17: DDS reconstruction filter with 18kHz cutoff.

Analog Shield Application Note 2: FFT Spectrum Visualizer

By William J. Esposito, Doctoral Candidate and Gregory Kovacs, Professor of Electrical Engineering, Stanford University
R0.4, 5/9/14

Overview

The FFT Spectrum Visualizer takes in a signal (sound, electricity, noise) and shows the tones that make up that signal.

For example, in a musical piece, two musical instruments could be playing tones at different frequencies. The spectrum reveals exactly what tones make up that sound.

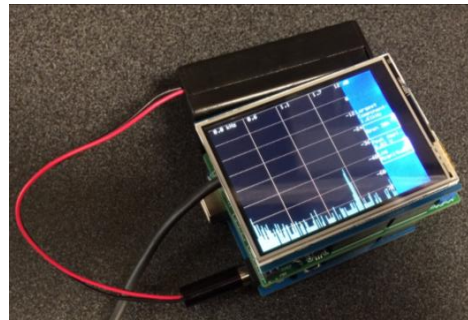


Figure 1: FFT spectrum visualizer.

It displays the spectrum of tones in a signal acquired with the Analog Shield and computes the spectrum using the Fast Hartley Transform (FHT) library from Open Music Labs (<http://wiki.openmusiclabs.com/wiki/ArduinoFHT>). The FHT function is a modified version of the Fast Fourier Transform optimized for strictly real input data [1].

The system offers touchscreen selectable log and linear scaling as well as selectable frequency span from 1 kHz to 30 kHz.

Hardware

The FFT Spectrum Visualizer is built around the Arduino and two shields, the Digilent Analog Shield*, and the Adafruit 2.8" TFT Touch Shield (v2), a 320x240 pixel touchscreen LCD. The Analog Shield is used for signal acquisition, and the Adafruit Touch LCD Shield is to display the spectrum.



Figure 2: Adafruit touch LCD.

The Arduino UNO R3 Can be found at:

<http://store.arduino.cc/>

The Adafruit RGB LCD Kit can be found at:

<http://www.adafruit.com/products/1651>

The Analog Shield can be found at:

<http://www.digilentinc.com/analogshield>



Figure 2: Analog Shield, showing space for prototyping breadboard.

*The Analog Shield was developed in cooperation with Texas Instruments and the Kovacs / Giovangrandi lab at Stanford University

Also recommended is some sort of input. At least a headphone jack or plug (see the “Input” section). Eventually, advanced users may want to build an anti-aliasing filter as well, although this is not necessary. It is discussed in a separate section for the interested reader.

To build the microphone input circuit, you will need:

- 1× LM741 Op Amp (or equivalent)
- 1× Electret Microphone
- 1× 1 kΩ Resistor
- 1× 10 kΩ Resistor
- 1× 200 kΩ Resistor
- 1× 100 nF Capacitor

Some solid core wire suited for breadboard use.

If you wish to use a line level input, you will also need a 3.5mm headphone jack.

Building the Demo

In order to build the Analog Shield FFT Visualizer, five nonstandard Arduino libraries are required. For a guide on Arduino libraries and how to install them, go to <http://arduino.cc/en/Guide/Libraries>

The libraries required for the visualizer are:

- The Open Music Labs FHT library
<http://wiki.openmusiclabs.com/wiki/ArduinoFHT>
- The Adafruit ILI9341 library (For the LCD)
http://github.com/adafruit/Adafruit_ILI9341
- The Adafruit GFX library (For the LCD)
<http://github.com/adafruit/Adafruit-GFX-Library>
- The Adafruit STMPE610 library (For Touch Functionality)
http://github.com/adafruit/Adafruit_STMPE610
- The Analog Shield library (for analog input)
<http://digilentinc.com/analogshield>

In case these libraries move in the future, the Adafruit LCD libraries are linked from the Adafruit Touch LCD Shield tutorial at <http://learn.adafruit.com/adafruit-2-8-tft-touch-shield-v2>

Once these libraries are downloaded and unzipped in the Arduino/libraries folder, be sure to re-open the Arduino IDE as it will not recognize libraries added while the program is open. Once the libraries are ready, stack the LCD and Analog Shield on the Arduino Uno. Next open the “FFT_Visualizer” sketch with the Arduino IDE and upload it to the Arduino UNO. Some data will appear on screen, although the visualizer does not yet have a signal. This data represents the inherent noise in the Spectrum Visualizer and the environment.

Input

To actually use the visualizer, some signal input is needed. Input from a signal generator is the obvious solution for an initial test, so any sort of handy sine wave generator will work. Hook the sine wave output between A0 and Ground and run. Be sure to not exceed the safe voltage of the Analog Shield, +/-5V.

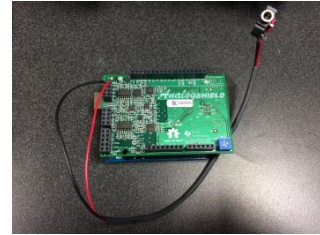


Figure 3: Headphone jack input.

A function generator is not a very satisfying test. Many hobbyists and students do not have convenient access to a good signal generator, and almost everyone will eventually want to investigate real world signals.

One option is to simply use line level (0-1.4V) audio as an input source. A standard 3.5mm (1/8") headphone jack soldered to breadboard wire can be connected to the headers of the Analog Shield. With this, the spectrum of playing music can easily be observed. Figure 4 shows the Spectrum Visualizer with a headphone jack attached and LCD removed.

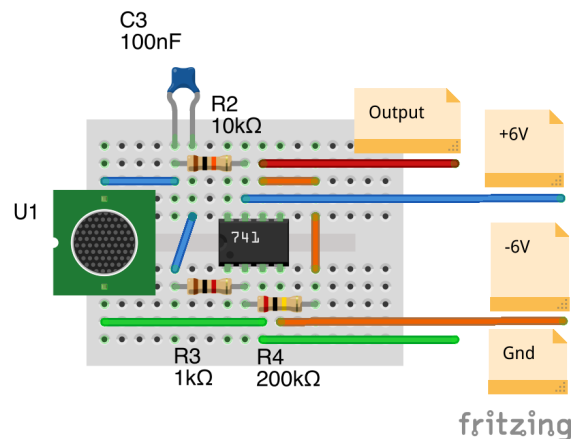


Figure 4: Microphone circuit layout.

Going one step farther, one can connect a microphone to the analog shield. After all, watching the spectrum move around as a person sings a tune or whistles, is fascinating.

A fairly straightforward solution is to obtain an electret microphone^[2] and an amplifier. Figures 5 and 6 show an example circuit using a commonly available electret microphone and an inexpensive LM741 Amplifier. For better performance, a higher performance amp, like an LF356 amplifier can be used. (U1 is the electret microphone).

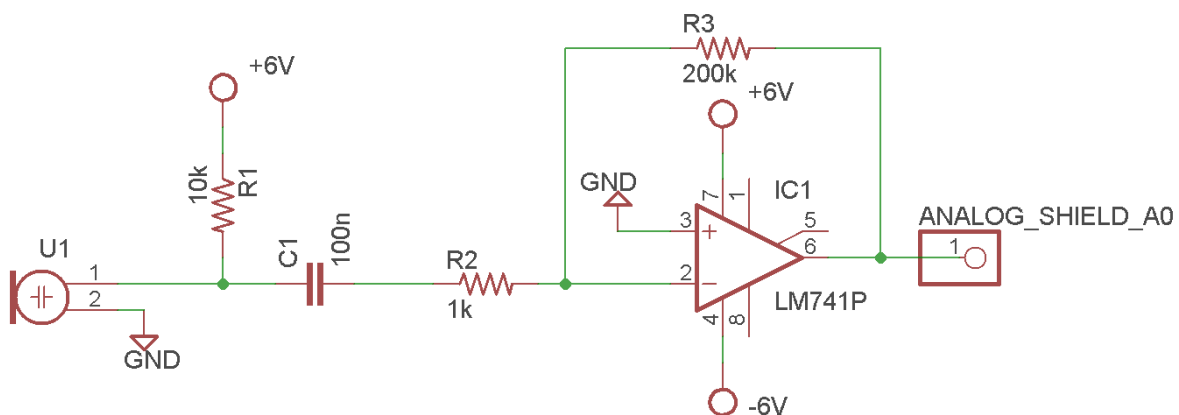


Figure 5: Electret microphone circuit schematic.

Using the Analyzer

With a signal in the analyzer, it is worth discussing how to use the analyzer itself.

The main display shows the spectrum of the signal received, which is a plot of the tones of the received signal. Increasing in frequency from left to right, the left most column reflects the power in the signal at 0Hz (DC) and the right most column the power in the signal at about 30 kHz. Each column reflects the power of a distinct frequency bin (band).

Tapping the main display (where the spectrum is displayed) will toggle the grid on and off. The grid shows the magnitude (strength) of a signal on the vertical axis and the frequency of the signal received on the horizontal axis. Toggling the grid off will allow for faster screen updates (a smoother output).

Tapping the 'span' button changes how wide of a frequency range the system samples. There are several steps from 1 kHz to 30 kHz. Switching to a smaller range means that individual frequency bins represent a smaller frequency band, but a smaller total frequency range is measured.

Finally, tapping the 'magnitude' button will switch between a logarithmic display and a linear output. A logarithmic output is generally more useful as it is more representative of how we perceive sounds, and makes small signals more visible.

How the Code Works

The FFT Spectrum Visualizer follows a fairly straightforward program flow. With each iteration of the main loop, it acquires an array containing 256 samples of 16-bit data from the Analog Shield at the specified sample rate. Once the array has been acquired, it passes the data to the FHT function, which transforms the time domain samples of a waveform in the array into a frequency spectrum. Finally, the resultant 16 bit spectrum is converted to an 8-bit range (on a logarithmic or linear y-axis scale) so that it can be reasonably displayed on our 240 pixel tall LCD.

Once the spectrum is ready to display, the system spends a few tens of milliseconds updating the LCD, as well as checking for touchscreen input and minor housekeeping. It then repeats the process. For slow sample rates (with a frequency span close to 1 kHz) the screen update delay is a fairly small part of the overall display frame rate (a few frames per second). At high sample rates, the screen refresh time is the dominant factor and the display output appears to be continuous.

The section of code below is not meant to convey the entire code, but merely the program flow of the main loop. The entire program has many helper functions and takes up several hundred lines, not including the invoked libraries. The full code can be found at

<http://www.digilentinc.com/analogshield>

```

void loop() {
    while(1) { //not leaving the loop function reduces jitter
        cli(); //stop all interrupts for acquisition.
        if(span == 30) //fastest frequency range, no sample delay
        {
            for (int i = 0 ; i < FHT_N ; i++) { // save 256 samples
                fht_input[i] = analog.signedRead(0); // put real data into bins
            }
        }
        else //any other frequency span.
        {
            for (int i = 0 ; i < FHT_N ; i++) { // save 256 samples
                fht_input[i] = analog.signedRead(0); // put real data into bins
                delayMicroseconds(spanDelay); //wait to lower sample rate
            }
        }

        //now, the fourier functions!
        fht_window(); // window the data for better frequency response
        fht_reorder(); // reorder the data before doing the fht
        fht_run(); // process the data in the fht

        if(linLog) //decide if output is going to be linear or logarithmic.
        {
            fht_mag_lin8(); // take the output of the fht
        }
        else
        {
            fht_mag_log(); // take the output of the fht
        }
        sei(); //start interrupts back up and build our display

        SPI.setDataMode(SPI_MODE0); //correct SPI Mode for the LCD

        if(linLog) //draw the display in the right mode
        {
            drawBins(fht_lin_out8); //draw the screen.
        }
        else
        {
            drawBins(fht_log_out); //draw the screen.
        }

        SPI.setDataMode(SPI_MODE3); // correct SPI mode for analog.read();
    }
}
    
```

Anti-Aliasing Filter

To improve the analyzer, an anti-aliasing filter can be added.

The process of sampling a digital signal into the analog domain causes frequencies too high to be accurately captured by the Analog to Digital conversion to falsely reappear as lower frequencies in the digitized signal^[2]. Signals seem to wrap around, or ‘bounce off’ the top of the spectrum plot.

The FFT spectrum visualizer can actually provide a fantastic demonstration of aliasing and the need for an anti-aliasing filter. If the FFT Visualizer is set to a 1 kHz span and given an input sine wave at 500 Hz (for example, using a headphone jack and an online tone generator), there will be sharp peak in the middle of the spectrum. If the input frequency is increased to 800 Hz, the peak will move to the right. Above 1 kHz (1000 Hz) the peak would begin to move ‘backwards’ (to the left), even as the frequency continues to increase. At 1.5 kHz, the peak would once again be in the center of the spectrum. At 2 kHz, the peak reaches the left hand side, and then starts moving rightward again! By 2.5 kHz it would once again return to the middle of the display, appearing for all intents and purposes identical to the 500 Hz signal! Clearly, this is a problem.

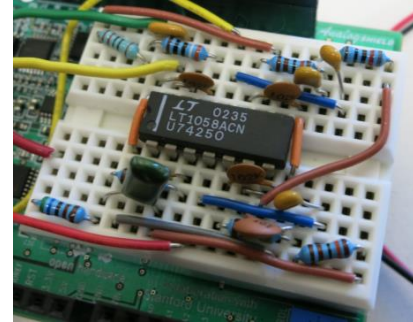


Figure 6: Anti-aliasing filter.

The solution to this is an analog ‘low pass’ filter between the input signal and the input to the Analog Shield. A low pass filter is used to remove the higher frequencies before they are digitized, passing only the frequencies of interest.

In an ideal world, this filter would act like a brick wall, perfectly passing frequencies up to the highest one that the Visualizer can capture and perfectly eliminating all others. Unfortunately, such filters can’t be made in the real world, as making a sharper edge requires a more complicated filter. A truly perfect cutoff would require an infinitely large filter!

For the 30 kHz sample mode, an example anti-aliasing filter design with a cutoff at 24 kHz and a stop band at 40kHz has been shared with this document, both as Figure 7 and in more detail as “Appendix A_Filter.pdf”. This filter is not perfect, as signals between 24 kHz and the highest frequency bin (around 30kHz) would appear weaker than they actually are. Furthermore, signals between 30 kHz and 40 kHz will still appear, as if they are actually signals below 30kHz, but these tradeoffs have to be made. Ideally, one would simply not show the bins between 24 and 30 kHz, but given the flexible nature of this project and the fact that it is already limited to 30kHz, those bins have been kept.

It stands to reason that if the FFT Visualizer is set to 1 kHz mode, a filter with a cutoff near 1kHz should be chosen, offering similar behaviors to the 30 kHz filter, but at lower frequencies. Such a design hasn’t been included here, but TI Webench (www.ti.com/webench), or many other free filter design tools can easily design such a filter.

For any active filter, the signal from outside should go into the input of the filter; output from the filter should be connected to the input (A0) of the Analog Shield. Power can be drawn from the Analog Shield's onboard adjustable supply.

Notes

[1] Cypress semiconductor has a whitepaper explaining the differences between the FFT and FHT, it can be found at <http://www.cypress.com/?docID=43614>

[2] An electret microphone is a simple microphone that is fairly easy to wire up, see the Wikipedia article at http://en.wikipedia.org/wiki/Electret_microphone. Sparkfun sells a standard electret <https://www.sparkfun.com/products/8635>, although it is identical to one you can get at Radio Shack or any commodity electronic components store.

[3] Maxim Integrated has put together a clear article describing the need for and benefit of an anti-aliasing filter which can be found at <http://www.maximintegrated.com/app-notes/index.mvp/id/928>.

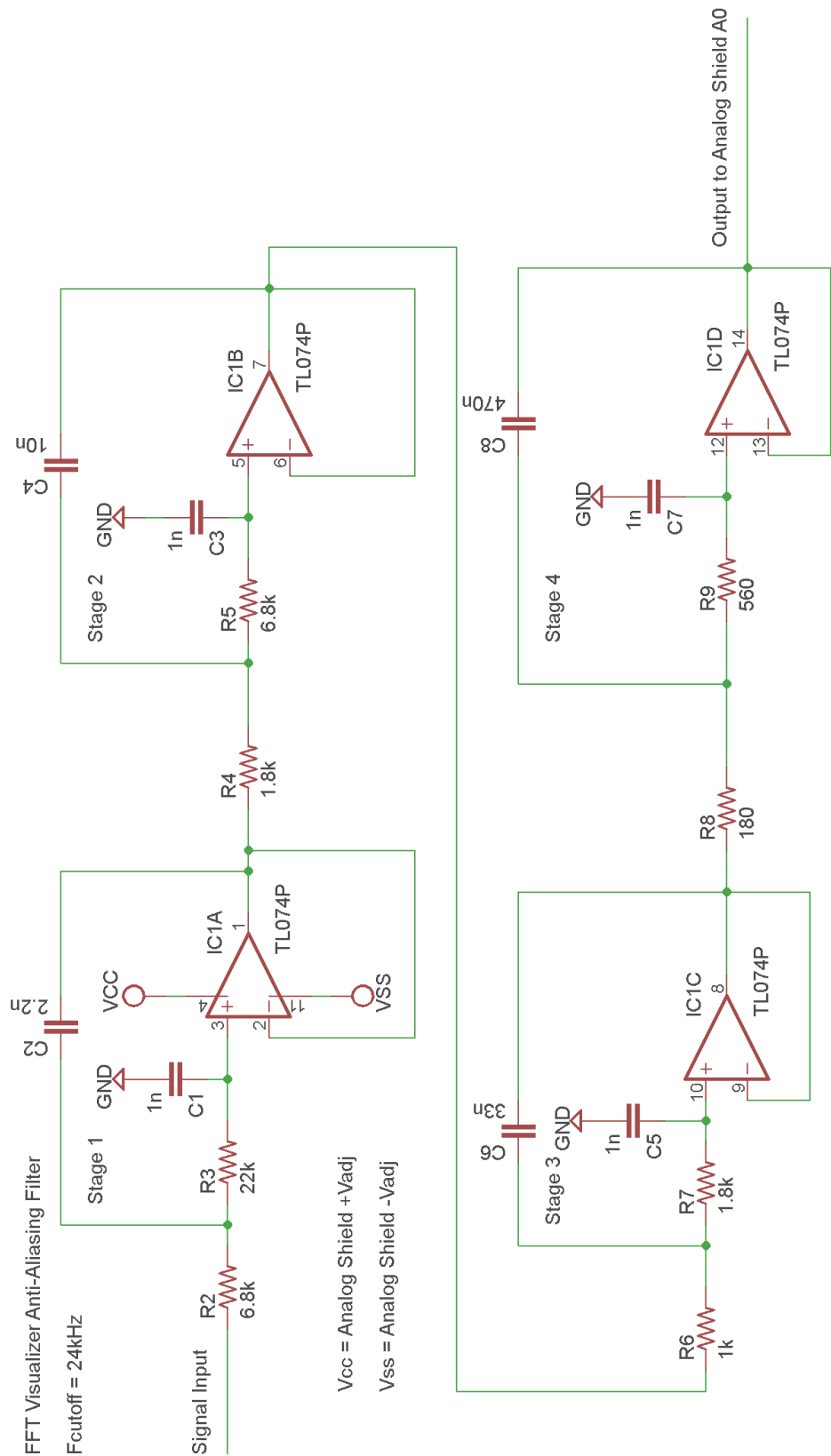


Figure 7: LPF with 24kHz Cutoff. A more detailed schematic is attached as "Appendix A_Filter.pdf".

Analog Shield Application Note 3: Polyphonic Music Player

By William J. Esposito, Doctoral Candidate and Gregory T.A. Kovacs, Professor of Electrical Engineering, Stanford University
R0.6, 5/29/14

Overview

The Analog Shield Polyphonic music Player plays a converted 4-voice music file through normal headphones or a stereo system.

Hardware

The Polyphonic Music Player is built around the Arduino and uses the Digilent Analog Shield* for signal output, combining the four channels with a simple resistive mixer.

The Arduino UNO R3 Can be found at:

<http://store.arduino.cc/>

The Analog Shield can be found at:

<http://www.digilentinc.com/analogshield>

A full description of the resistive mixer is found below, but the parts requirements include (for monaural output):

4× 330Ω Resistors

1× 3.3kΩ Resistor

1× 0.047μF Capacitor

1× 3.5mm Headphone Jack

Building the Demo

In order to build Polyphonic music Player, two nonstandard Arduino libraries are necessary. For a guide on Arduino libraries and how to install them, go to <http://arduino.cc/en/Guide/Libraries>

The libraries required for the visualizer are:

The TimerOne library (for the DDS sample clock)

<https://code.google.com/p/arduino-timerone/>

The Analog Shield library (for analog input)

<http://github.com/wespo/analogshield>

In case these libraries move in the future, the Adafruit LCD libraries are linked from the Adafruit Touch LCD Shield tutorial at

<https://learn.adafruit.com/rgb-lcd-shield/>

*The Analog Shield was developed as a collaboration between Texas Instruments and the Kovacs / Giovangrandi lab at Stanford University.

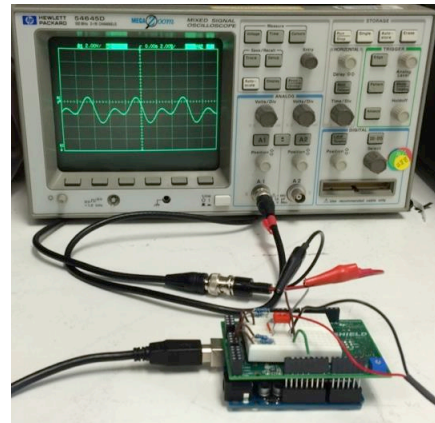


Figure 1: Polyphonic music Player with an oscilloscope.



Figure 2: Analog Shield, with breadboard not attached.

Once these libraries are downloaded and unzipped in four Arduino/libraries folder, be sure to re-open the Arduino IDE as it will not recognize libraries added while it is open. Once that is done, attach the Analog Shield to the Arduino Uno. Next, open the “Polyphonic_music” sketch with the Arduino IDE and upload it to the Arduino Uno. Once the demo is uploaded, one tone will be output from each of the four DAC outputs on the analog shield (D0-D3).

Included with the player is an example song, which plays when the sketch is uploaded.

At this point, tones can be heard directly at the outputs, by connecting a headphone (and a current limiting resistor of at least 330 Ω). To produce a combined sound output, a simple filter will have to be constructed, which will be described next.

Connecting the Output

The most straightforward way to output all four voices at one time is to use resistors to build a simple four input low-pass mixer / filter at the output of the Analog Shield’s DAC. It is crucial that all four independent resistors be used before the junction instead of one after, as the presence of the resistors ensures that there is no short circuit between outputs of the DAC. These circuits will work well for headphones, and a lower value or a potentiometer can be substituted for R5 (and R6) if adjustable volume is desired.

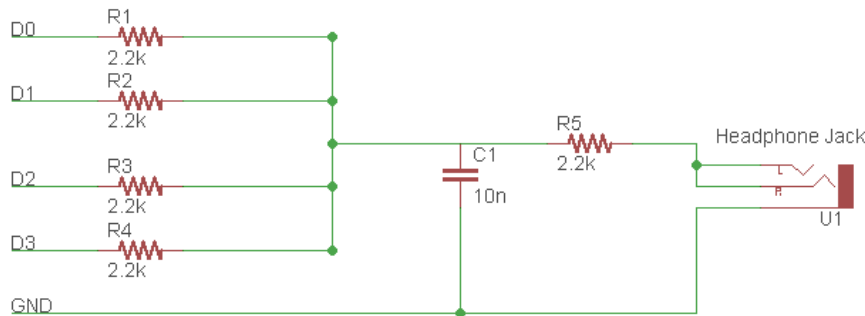


Figure 3: Monaural output mixer for headphones.

An alternative configuration allows stereo output, although the left and right channels are separated by voice, rather than arranged as the composer may have intended.

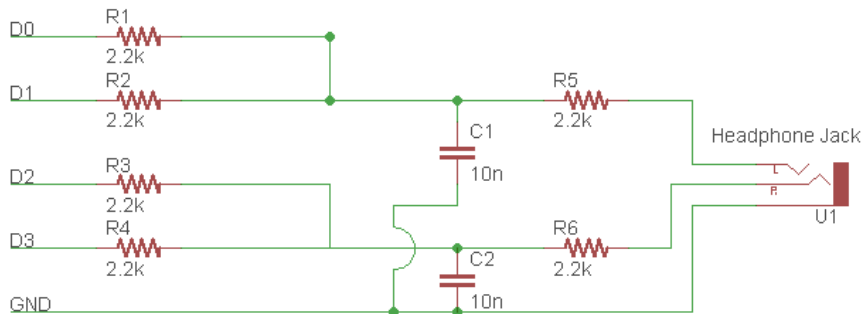


Figure 4: Pseudo-stereo output headphone mixer.

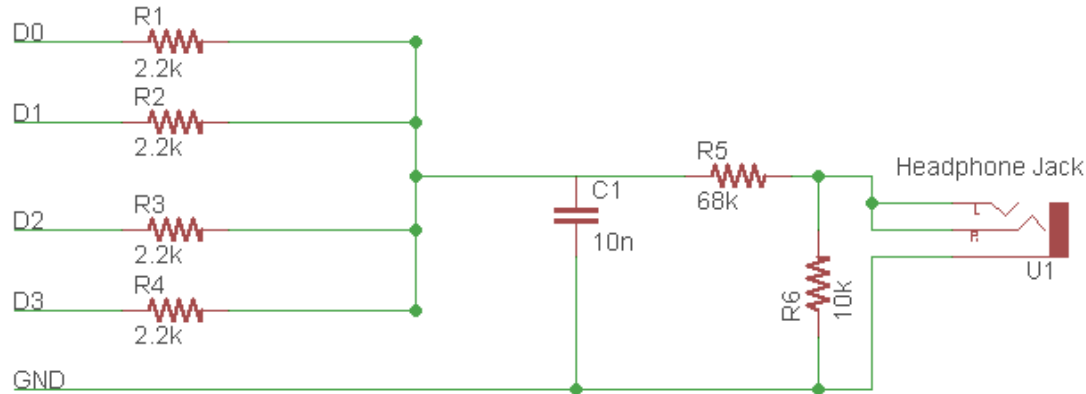


Figure 5: Mixer for driving line input on a home stereo.

For driving a stereo receiver, this output may deliver too much power. Be sure to turn the volume to the minimum setting and then adjust upward. If the primary output of the player is going to be a stereo, the circuit described in *figure 5* would be preferable.

Once a mixer has been connected to the music Player, a headphone or the line input of a stereo can be connected to the polyphonic demo and tested. Beware, that the input is above standard line input voltages (line level) and a larger resistor is recommended for R5/6

A simple resistive mixer is a low quality circuit and it is only due to the buffers on the outputs of the Analog Shield DAC that such a simple circuit is possible here. A more detailed discussion of mixer circuits (along with several higher quality mixer circuits that can be built with components found in most any parts kit) has been written by Rod Elliot and can be found at

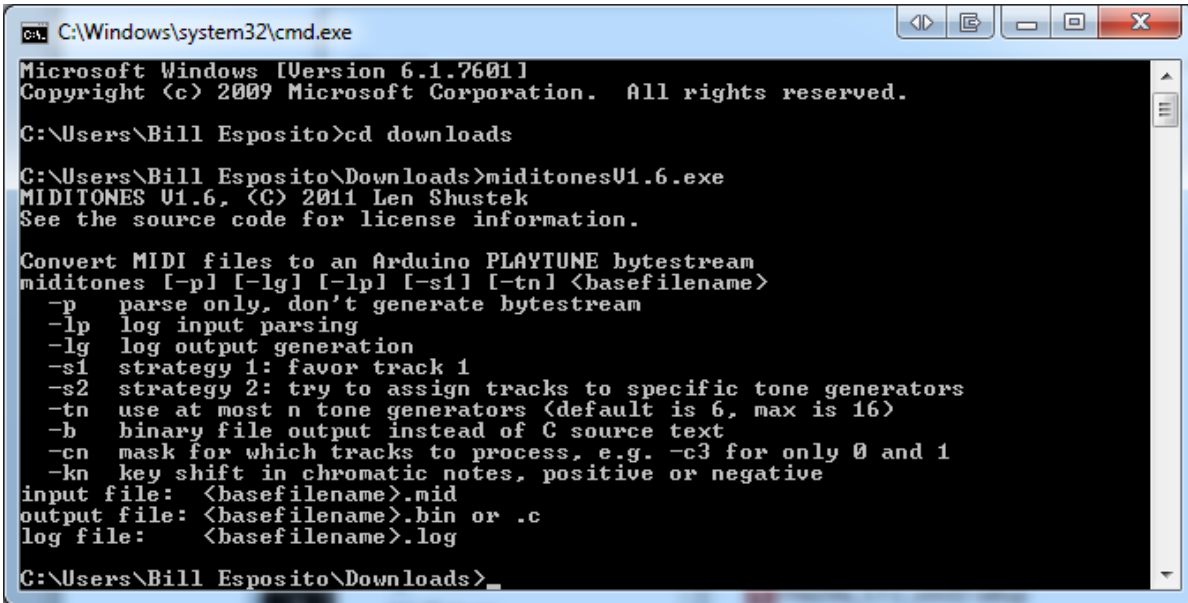
<http://sound.westhost.com/articles/audio-mixing.htm>.

How the Music Is Stored

The Polyphonic Music Player stores its output sound in the form of an array stored in program memory. Using this larger part of memory allows a much longer song to be stored but prevents the score to be modified in real time by the player itself. Changing the output song requires a new upload to the Arduino Uno. Using program memory is described in detail at <http://arduino.cc/en/Reference/PROGMEM>.

In order to create a new song, an array of commands and delays must be constructed. Instructions perform operations such as starting a tone generator, stopping a tone generator, ending the track, or restarting the track from the beginning. Between events, delays are inserted to pace the music.

The command to start a tone generator is of the format '0x9n' where 'n' is a number between 0 and 3 indicating which tone generator to start. This command must be followed by an integer indicating which MIDI note to start the generator at. There are 128 MIDI notes which span five octaves, from 8 Hz to 12.5 kHz. A full listing of MIDI notes can be found at <http://www.tonalsoft.com/pub/news/pitch-bend.aspx>.



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Bill Esposito>cd downloads
C:\Users\Bill Esposito\Downloads>miditonesV1.6.exe
MIDITONES V1.6, (C) 2011 Len Shustek
See the source code for license information.

Convert MIDI files to an Arduino PLAYTUNE bytestream
miditones [-p] [-lg] [-lp] [-s1] [-tn] <basefilename>
  -p parse only, don't generate bytestream
  -lp log input parsing
  -lg log output generation
  -s1 strategy 1: favor track 1
  -s2 strategy 2: try to assign tracks to specific tone generators
  -tn use at most n tone generators (default is 6, max is 16)
  -b binary file output instead of C source text
  -cn mask for which tracks to process, e.g. -c3 for only 0 and 1
  -kn key shift in chromatic notes, positive or negative
input file: <basefilename>.mid
output file: <basefilename>.bin or .c
log file: <basefilename>.log
C:\Users\Bill Esposito\Downloads>
  
```

Figure 6: Miditones command line interface.

{START GENERATOR 0, MIDI NOTE 20, DELAY 2000ms, START GENERATOR 1, MIDI NOTE 30, DELAY 1000ms, START GENERATOR 2, MIDI NOTE 40, DELAY 1500ms, STOP GENERATOR 0, DELAY 1500ms, END TRACK}

Converting a MIDI for the Player

A utility called Miditones exists which takes a MIDI file as input and outputs an array in the format used by the player. The program is open source and available on Google code at <https://code.google.com/p/miditones>. Unfortunately, the version on Google code is not compiled for OSX, so included with the polyphonic player download are Windows and Mac compiled binaries of Miditones along with the source code.

The primary limitation of the Polyphonic MIDI player is that it only has four voices. Although this is better than the best internal Arduino tone generator, there are many midi files with more than four voices. Be aware that complex tracks may have “missing” portions if too many voices are cut out in the conversion process.

Once a suitable midi file is ready, download the Miditones program from Google code. It is easiest to download Miditones to a default directory, such as “downloads” as it will be easier to find and use on the command line.

“Miditones” must be run from a command window. The command window can be found by searching for CMD on windows).

From the command window, navigate to the folder which contains miditones (if the command window opens in "C:\Users\<your username>", typing "cd Downloads" will work.

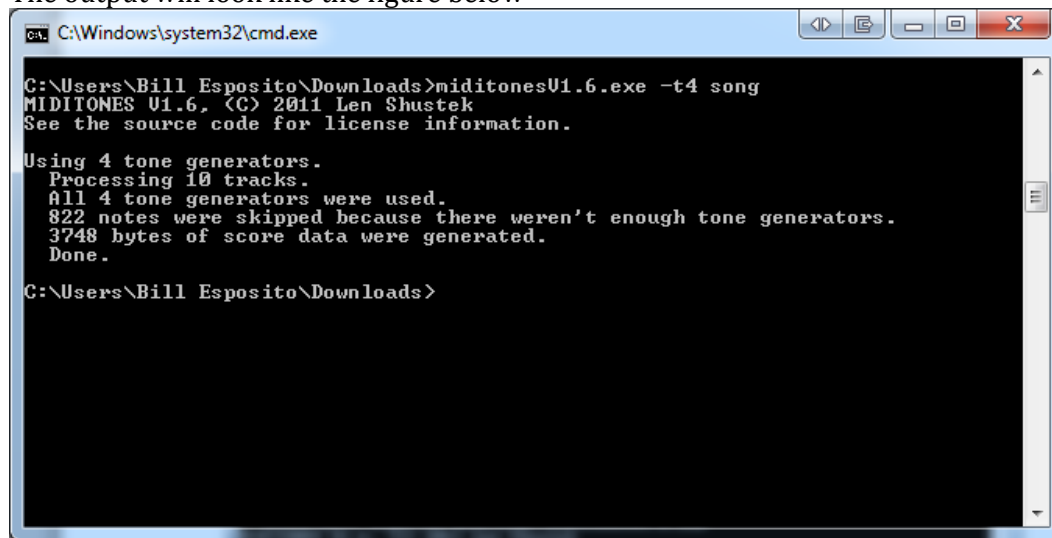
Try executing Miditones with no optional parameters by typing "miditonesV1.6.exe". Miditones will display a manual page which explains various useful options.

To convert a file, execute Miditones with the filename as a parameter. It is also useful to limit the number of output voices to four with the "-t4" option. Four is the maximum number of channels that the Midi Player will output, and additional channels will waste memory but not play back sound.

For example, for a file "song.mid", with a four voice limit, the command would be:
miditonesV1.6.exe -t4 song

Including the .mid extension to the file name will cause an error.

The output will look like the figure below



```
C:\Windows\system32\cmd.exe
C:\Users\Bill Esposito\Downloads>miditonesV1.6.exe -t4 song
MIDITONES V1.6, (C) 2011 Len Shustek
See the source code for license information.

Using 4 tone generators.
Processing 10 tracks.
All 4 tone generators were used.
822 notes were skipped because there weren't enough tone generators.
3748 bytes of score data were generated.
Done.

C:\Users\Bill Esposito\Downloads>
```

Figure 7: Executing Miditones.

Once Miditones reports that it has produced an output, it will be stored in a file with the same name as the input midi but a .c extension (in the example, "song.c"). This file can be opened with any text editor (notepad will do), and the text is ready to copy into the polyphonic MIDI player Arduino sketch.

Adding Music to the Player

To copy the song into the Arduino sketch, replace the array named 'score' that starts on line 17 of the sketch, just below the comment "PASTE YOUR SONG HERE" and above the comment "Analog Shield Library"

Arduino Analog Shield Polyphonic Music Player

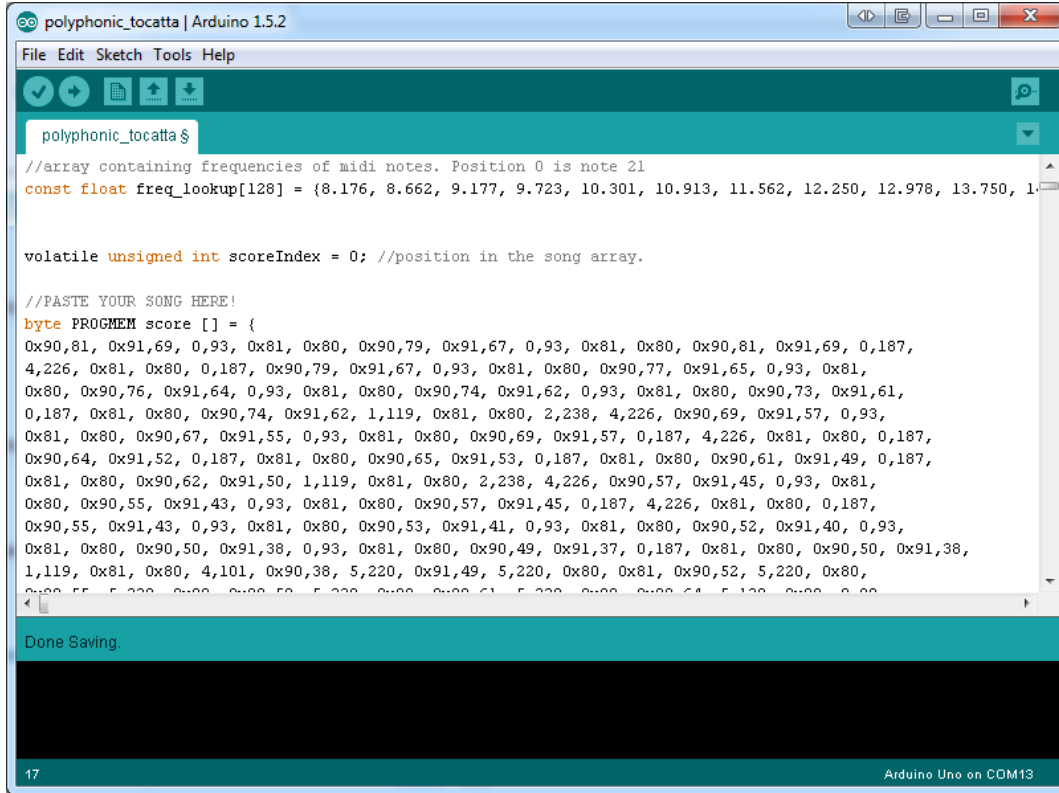


Figure 8: Insertion location for the song.

Once the new song is pasted into the sketch, it is ready to upload. Note, that a long or particularly complex music file can cause the Arduino to run out of memory and compilation to fail.

If the song is too long, array elements can be deleted, which will truncate the song, but save memory. To truncate the song, delete array elements to (but not including) an “0x80”, “0x81”, “0x82” or “0x83” and add an “0xf0” command. So long as the last element of the array is 0xf0, the song will end properly. These ‘0x##’ commands are hexadecimal instructions.

How the Code Works

The actual code of the Polyphonic music Synthesizer is based on the Analog Shield DDS routine described in Instructional materials available online at <http://digilentinc.com/analogshield>. The routine has been extended to support four channels of output, at the cost of some frequency precision and bandwidth.

Unlike the single channel DDS routine, the polyphonic music player can only generate sine waves; output becomes unreliable for notes above 8 kHz (which manifests in the form of tones modulated on the 20 kHz sample clock).

Arduino Analog Shield Polyphonic Music Player

With the DDS acting as a four channel tone generator, the main loop drives the DDS by reading the score array and parsing the commands therein which are a series of instructions to set a tone generator to a certain frequency, stop a tone generator, or wait a set amount of time.

When the main loop reaches the end of the array, it will either begin again or simply stop playback, as determined by the last element of the array (either 0xf0 or 0xe0).

The code for the polyphonic music player is attached here, and can be found at <http://www.digilentinc.com/analogshield>

```
/* Direct Digital Synthesis with LCD Output */
/* 16 bit polyphonic DDS for music generation.*/
//12 bit long sine table
PROGMEM prog_uint16_t isinTable16[] = { <full sine wave here> };

const float freq_lookup[128] = {<frequencies go here>}; //freqs of midi notes.
volatile unsigned int scoreIndex = 0; //position in the song array.
byte PROGMEM score [] = {<music goes here>}; //PASTE YOUR SONG HERE!

#include <analogShield.h> //Analog Shield Library
#include <avr/pgmspace.h> //for storing sine and song
#include <TimerOne.h> //Timer

//DDS variables
volatile unsigned int tuningWord[4] = {0};
volatile unsigned int phaseAccumulator[4] = {0}; //16 bits
volatile unsigned long sampleCount = 0;
volatile unsigned long nextEvent = 0;

void setup() {}

void loop()
//takes user input and the goes into DDS forever.
{
    Timer1.initialize(50); //Setup the timer for DDS
    Timer1.attachInterrupt(dds); //start DDS
    while (1) { //stay inside this loop to reduce jitter
        TIMSK0 = 0x00; //Turn off timers
        TIMSK2 = 0x00; //to reduce jitter
        //enough samples have passed time to do something
        if (sampleCount > nextEvent)
        {
            //first a byte from the array.
            //This will set your nextEvent delay
            byte event = pgm_read_word(score + scoreIndex);
            scoreIndex++;
            if (event < 0x80)
                //If the highest bit is zero.
                //It is a time and we want to get a delay.
            {
                unsigned int delayMS =
                    highByte(event) +
                    lowByte(pgm_read_word(score + scoreIndex));
            }
        }
    }
}
```

Arduino Analog Shield Polyphonic Music Player

```
        scoreIndex++;
        nextEvent += 20 * delayMS; //actually 19.770
    }
    else //some sort of command
    {
        if((event&0xF0) == 0x80) //start channel
        {
            event = event & 0x03;
            tuningWord[event] = 0;
        }
        else if((event&0xF0) == 0x90) //start channel
        {
            event = event & 0x03;
            //get midi Note
            byte note = pgm_read_word(score + scoreIndex);
            scoreIndex++; //increment table pointer
            //convert midi note into frequency
            float newFreq = freq_lookup[note];
            //set appropriate channel;
            tuningWord[event] = (unsigned int) (newFreq * 3.28);
        }
        else if(event == 0xF0) //end of song sentinel; Stop
        {
            tuningWord[0] = 0;
            tuningWord[1] = 0;
            tuningWord[2] = 0;
            tuningWord[3] = 0;
            while(1);
            {};
        }
        else if(event == 0xE0) //end of song sentinel; Repeat
        {
            scoreIndex = 0;
        }
    }
}
}; //wait forever.
}
//DDS code
void dds() //Direct Digital Synthesis Lives Here.
{
    unsigned int value[4];
    unsigned int tempPhase[4];
    //loop unrolled by hand because apparently the ATMEGA compiler
    //doesn't understand the word 'optimize'.

    //increment the phase accumulator by the value
    //stored in the tuning word.
    phaseAccumulator[0] += tuningWord[0]; //16 bits
    phaseAccumulator[1] += tuningWord[1]; //16 bits
    phaseAccumulator[2] += tuningWord[2]; //16 bits
    phaseAccumulator[3] += tuningWord[3]; //16 bits

    //top 12 bits of each accumulator
    tempPhase[0] = (unsigned int) (phaseAccumulator[0] >> 4);
    tempPhase[1] = (unsigned int) (phaseAccumulator[1] >> 4);
```

Arduino Analog Shield Polyphonic Music Player

```
tempPhase[2] = (unsigned int)(phaseAccumulator[2] >> 4);
tempPhase[3] = (unsigned int)(phaseAccumulator[3] >> 4);
//look up values
value[0] = pgm_read_word(isinTable16 + tempPhase[0]);
value[1] = pgm_read_word(isinTable16 + tempPhase[1]);
value[2] = pgm_read_word(isinTable16 + tempPhase[2]);
value[3] = pgm_read_word(isinTable16 + tempPhase[3]);

//write the result to the output.
analog.write(value[0], value[1], value[2], value[3], true);
//count samples
sampleCount++;
}
```

Potential Improvements

The music player is an extremely satisfying application of the Analog Shield. It would be worth additional investigation to attempt to connect the Arduino to an SD card, which would provide hours of playback. Another improvement worth investigating is “soft voices”. That is, implementing software multiplexing of two voices on each channel. This could allow for an 8 voice music player, although the cost would be a further reduction in bandwidth of the synthesized output.

Another interesting modification would be to use four amplifiers and four speakers to output each voice on its own physical channel.

Disclaimer:

This code and circuit was developed by William Esposito, Ph.D. Candidate in Electrical Engineering, Stanford University, in the Kovacs/Giovangrandi Laboratory in collaboration with Texas Instruments, Incorporated. All code herein is free and open source, but is provided as-is with no warranties implied or provided. Use of this code and the associated documentation is at the user’s own risk.

Analog Shield Application Note 4: Lissajous Diagrams

By William J. Esposito, Doctoral Candidate and Gregory T.A. Kovacs, Professor of Electrical Engineering, Stanford University
R0.3, 5/29/14

Overview

Using concepts from trigonometry, a circle can be created from a sine and a cosine function. More complex shapes can be made by varying phases and frequencies.

With the Analog Shield DAC, two waves are output to the x- and y- axis inputs of an oscilloscope. One is set at a fixed frequency and the second varies over time.

The ratio of the frequencies on the x- and y- axes determines shape of the waveform displayed on the screen. If the frequencies are equal, a single loop is displayed. At a ratio of 2:1, a figure eight shape with two lobes appears. At 3:1, three lobes appear, and so on.

Hardware

The Lissajous Pattern Generator is built around the Arduino and the Analog Shield*

The Arduino UNO R3 Can be found at:

<http://store.arduino.cc/>

The Analog Shield can be found at:

<http://www.digilentinc.com/analogshield>

The demo also requires a two-channel oscilloscope with x-y mode for output.

Building the Demo

In order to build Lissajous demo, two nonstandard Arduino libraries are necessary.

For a guide on Arduino libraries and how to install them, go to

<http://arduino.cc/en/Guide/Libraries>

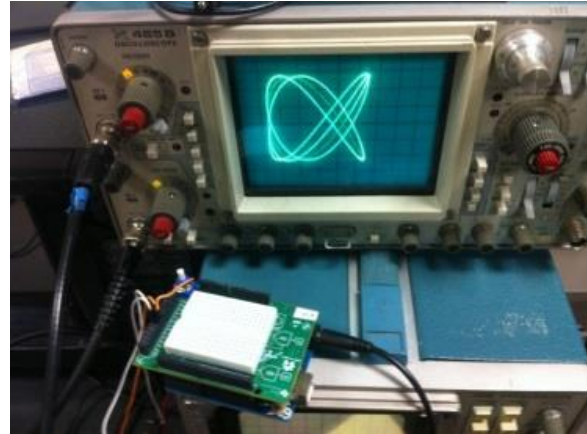


Figure 1: Prototype Analog Shield with Lissajous pattern generator with an oscilloscope.



Figure 2: Prototype Analog Shield, with breadboard not attached.

*The Analog Shield was developed as a collaboration between Texas Instruments and the Kovacs / Giovangrandi lab at Stanford University.

Arduino Analog Shield Lissajous Curve Generator

The libraries required for the demo are:

The TimerOne library (for the DDS sample clock)

<https://code.google.com/p/arduino-timerone/>

The Analog Shield library (for analog input)

<http://github.com/wespo/analogshield>

The code itself for the Lissajous demo can be found at <https://github.com/wespo> under 'Lissajous', as well as in this document.

Once these libraries are downloaded and unzipped in four Arduino/libraries folder, be sure to re-open the Arduino IDE as it will not recognize libraries added while it is open. Once that is done, attach the Analog Shield to the Arduino Uno. Next, open the "Lissajous_curve" sketch with the Arduino IDE and upload it to the Arduino Uno.

Connecting the Output

Next, connect the channel one input of an oscilloscope between D0 and Ground on the Analog Shield. Connect the channel two input of the oscilloscope between D1 and Ground. Switch the oscilloscope to X-Y input mode and the image on the scope should appear to shift and swirl in patterns similar to *figure 1*. Switching the oscilloscope to standard time mode will show two sine waves (one on each respective channel), one at a fixed frequency and one varying in frequency over time.

How the Code Works

The Lissajous curve generator is based on DDS code outlined at in the DDS tutorial at <http://www.digilentinc.com/analogshield>. It uses a recorded single period of a sine wave stored in a table to generate an output waveform. It can adjust the output frequency via software, by controlling the rate at which the program steps through the table. By outputting two sine waves simultaneously (one of them phase shifted by 90° to generate a cosine), a Lissajous curve is produced.

The code for the Lissajous curve generator is attached here, and can be found at

<http://www.digilentinc.com/analogshield>

```
/* Lissajous Curve */

//12 bit long sine table
PROGMEM prog_uint16_t isinTable16[] = { <sine table omitted for space> };

#include <analogShield.h>
#include <avr/pgmspace.h>
#include <TimerOne.h>

unsigned long frequencyIncrement = 40000; //32 bits
unsigned long frequencyIncrement2 = 40000; //32 bits
unsigned long phaseAccumulator = 0; //32 bits
unsigned long phaseAccumulator2 = 0; //32 bits

void setup() {
    Timer1.initialize(30); //100000
    Timer1.attachInterrupt(circle);
}
```


Arduino Analog Shield Lissajous Curve Generator

```
void loop(){ //Sweeps frequency
  frequencyIncrement2 = 20000;
  while(frequencyIncrement2 < 40000)
  {
    frequencyIncrement2 += 50;
    delay(50);
  }
  delay(10000);
}

void circle() //this is it.
{
  //increment the phase accumulator
  phaseAccumulator += frequencyIncrement;
  phaseAccumulator2 += frequencyIncrement2;

  //peel off top bits ch 1
  unsigned long tempPhase = (phaseAccumulator >> 12);
  unsigned int tablePosition = tempPhase & 0xFFFF;
  tablePosition = tablePosition & 0xFFFF;
  //peel off top bits ch 2
  unsigned long tempPhase2 = (phaseAccumulator2 >> 12);
  unsigned int tablePosition2 = tempPhase2 & 0xFFFF;
  tablePosition2 = (tablePosition2 + 1024) & 0xFFFF;

  //look up and output sine wave values
  unsigned int value = 0;
  unsigned int value2 = 0;
  value = pgm_read_word(isinTable16 + tablePosition);
  value2 = pgm_read_word(isinTable16 + tablePosition2);
  analog.write(value,value2, true);
}
```

Potential Improvements

The Lissajous curve generator could be used as a music visualizer by controlling the frequency variable with input from an Analog Shield ADC configured as a beat detector. It could also be used as a back end for a game where the user controls one of the frequencies with a potentiometer and the other is controlled via software, challenging the user to follow the frequency changes made by the software and maintain a circular output.

Disclaimer:

This code and circuit was developed by William Esposito, Ph.D. Candidate in Electrical Engineering, Stanford University, in the Kovacs/Giovangrandi Laboratory in collaboration with Texas Instruments, Incorporated. All code herein is free and open source, but is provided as-is with no warranties implied or provided. Use of this code and the associated documentation is at the user's own risk.

App Note 5: Network Analyzer

*By William J. Esposito, Doctoral Candidate and Gregory T.A. Kovacs, Professor of Electrical Engineering,
Stanford University
R0.1, 6/12/15*

Overview

The Network Analyzer project extends concepts from the FFT [Spectrum Analyzer](#) project to use the analog shield to generate the frequency domain magnitude transfer function of an electrical circuit, up to 30 kHz.

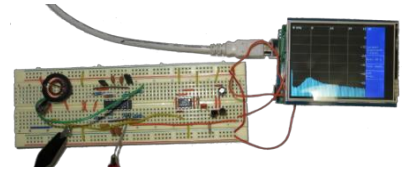


Figure 1 Network Analyzer.

Hardware

The Network Analyzer is built around the Arduino and two shields, the Digilent Analog Shield*, and the Adafruit 2.8" TFT Touch Shield (v2), a 320x240 pixel touchscreen LCD. The Analog Shield is used for signal acquisition, and the Adafruit Touch LCD Shield is to display the spectrum.

The Arduino UNO R3 Can be found at:

<http://store.arduino.cc/>

The Adafruit RGB LCD Kit can be found at:

<http://www.adafruit.com/products/1651>

The Analog Shield can be found at:

<http://www.digilentinc.com/analogshield>

Also needed are several analog components to build the noise source and Anti-aliasing filter (described in detail below). You will need:

1× solderless breadboard

1× LM741 Op Amp (or equivalent)

1× LM324 Quad Op-Amp (or equivalent)

2× 2N3904 Bipolar Junction Transistor (or equivalent, MOSFET Types will not work)

4× each 100n, 1nF capacitor

2× 1.5nF capacitor

1× each 1μ, 470n, 33n, 3.3nF capacitor

1× each 47k, 10k, 8.2k, 5.1k, 4.7k, 3.6k, 3.3k, 2.7k, 2.4k, 1.5k, 560Ω, resistors

Some solid core wire suited for breadboard use.

Additional components for optional, test filter:

1× each 10k, 150Ω resistors

1× each 47n, 470nF capacitors

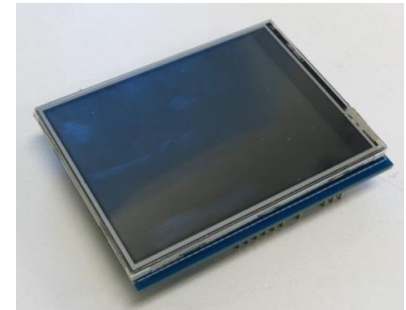


Figure 2: Adafruit Touch LCD.



Figure 3: Analog Shield without breadboard.

App Note 5: Network Analyzer

Building the Demo: Software

In order to build the Analog Shield Network Analyzer, five nonstandard Arduino libraries are required. For a guide on Arduino libraries and how to install them, go to <http://arduino.cc/en/Guide/Libraries>

The libraries required for the visualizer are. The Open Music Labs FHT library <http://wiki.openmusiclabs.com/wiki/ArduinoFHT>

The Adafruit ILI9341 library (For the LCD)

http://github.com/adafruit/Adafruit_ILI9341

The Adafruit GFX library (For the LCD)

<http://github.com/adafruit/Adafruit-GFX-Library>

The Adafruit STMPE610 library (For Touch Functionality)

http://github.com/adafruit/Adafruit_STMPE610

The Analog Shield library (for analog input)

<http://www.github.com/wespo/analogshield>

In case these libraries move in the future, the Adafruit LCD libraries are linked from the Adafruit Touch LCD Shield tutorial at

<http://learn.adafruit.com/adafruit-2-8-tft-touch-shield-v2>

Once these libraries are downloaded and unzipped in the Arduino/libraries folder, be sure to re-open the Arduino IDE as it will not recognize libraries added while the program is open. Once the libraries are ready, stack the LCD and Analog Shield on the Arduino Uno. Next open the “Network_Analyzer” sketch with the Arduino IDE and upload it to the Arduino UNO. Some data will appear on screen, although the analyzer does not yet have a signal. This data represents the inherent noise in the analyzer and the environment.

The Theory of the Network Analyzer

With the software out of the way, the next task is to construct the necessary hardware. First, however, a detour into theory is in order. This hardware consists of two major parts, the first is a “white noise” source. White noise sources are so named because, like light coming from a white lightbulb, their output contains all the frequencies (colors) of the audio spectrum. The output of this device sounds much like the “snow” effect heard when turning an old-fashioned analog television or radio to an unused channel. This source is key element to the Network Analyzer. When the white noise is passed through a circuit with an unknown frequency response, the output will contain all the frequencies that the circuit being tested can pass. By taking the FFT of the output signal, the Network Analyzer can show the frequency response of the circuit.

There is however, a catch. The white noise source, necessarily cannot put out every possible frequency in a given instant. Instead, what it does is put out many frequencies at once, so that over sufficient time, all frequencies will be output. Since the Network Analyzer’s FFT won’t show frequencies that are missing, the result would be an incorrect frequency response of the circuit. There’s a simple

App Note 5: Network Analyzer

workaround to this, however — the Network Analyzer records several periods of output and averages the resulting FFT response. A larger number of averages means frequencies are more uniformly represented at the input, producing a more accurate result at the output.

As with the FFT Spectrum Analyzer project, the Network Analyzer also requires an Anti-Aliasing filter before the input. With the output signal being a white noise source, this is substantially more critical than in the case of the spectrum analyzer, as the output signal is guaranteed to contain frequencies that would “fold” back and incorrectly appear to the analyzer as lower frequencies. The [Anti-Aliasing filter section](#) of the FFT Visualizer tutorial goes into the theory in greater detail. It is key to note though, that the folding phenomenon is real and can drastically affect the output of the system, as seen in the figure below. This pair of figures demonstrates how a notch filter appears to be nonfunctional when tested with the anti-aliasing filter disabled.

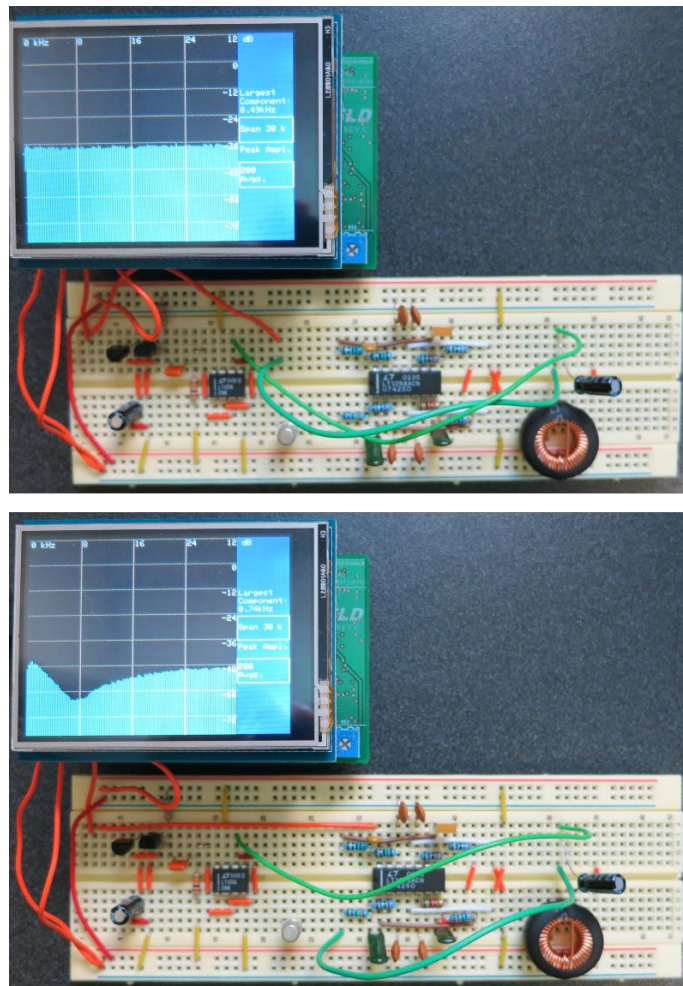


Figure 5: Network Analyzer with the anti-aliasing filter bypassed (top) and connected (bottom)

App Note 5: Network Analyzer

Building the Demo: Hardware

Now, with theory handled, it is time to assemble the circuits for the Noise Source and Anti-Aliasing filter as shown below. Note that VCC and VSS in the circuit diagrams are +Vadj and $\frac{1}{\sqrt{2}}V_{adj}$ on the Analog Shield, respectively. The variable supply on the Analog Shield should be set to its maximum value ($\pm 7.5V$) to ensure that the noise source generates output.

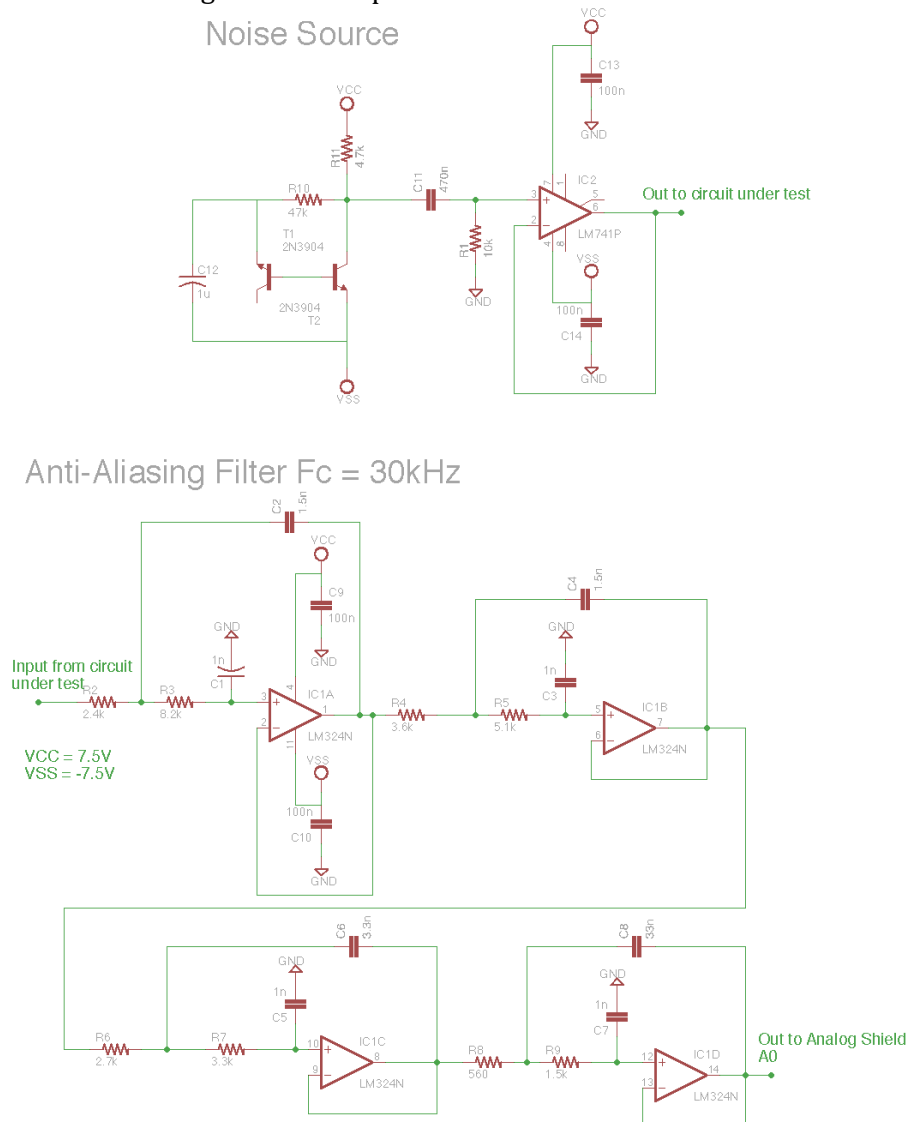


Figure 6: Noise source and anti-aliasing filter.

App Note 5: Network Analyzer

Connecting the Output

Now, the circuit is ready to go. A simple test is to design a simple 2nd order low-pass filter, using resistors and capacitors. The example filter below uses a 47 nF capacitor, a 470 nF capacitor, a 10 k Ω resistor and a 150 Ω resistor.

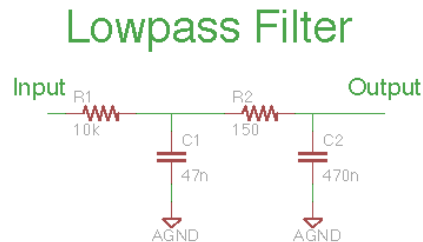


Figure 7: Test low-pass filter.

When the output of the noise source is connected to the input of the test filter, and the input of the anti-aliasing filter is connected to the output of the test filter, the network analyzer produces the output shown below.

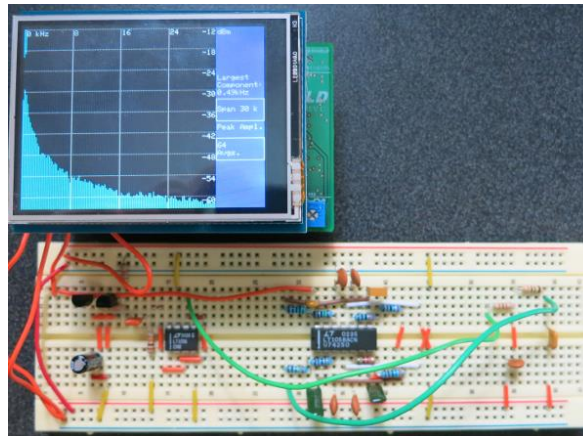


Figure 8: Network response of test low-pass filter.

Conclusion

The network analyzer provides a practical application of FFT spectrum plotting and provides a useful, if rudimentary version of a test instrument that normally retails for hundreds or thousands of dollars. Knowing the frequency response of a circuit under test is immensely useful for debugging circuits of all types and can provide insights into their behavior that are unobvious when observing the circuit purely in the time domain using an oscilloscope alone.

App Note 5: Network Analyzer

Disclaimer: This code and circuit was developed by William Esposito, Ph.D. Candidate in Electrical Engineering, Stanford University, in the Kovacs/Giovangrandi Laboratory in collaboration with Texas Instruments, Incorporated. All code herein is free and open source, but is provided as-is with no warranties implied or provided. Use of this code and the associated documentation is at the user's own risk.

Appendix G

Long-QT Detection Algorithm Code

```
function [outsig, triggers, avgsig] = TExtractor(signal, Fs, Tt, displayPlots)
% Check number of inputs.
if nargin > 4
    error('myfuncs:somefun2:TooManyInputs', ...
        'requires at most 3 optional inputs');
end

% Fill in unset optional values.
switch nargin
    case 2
        Tt = 0.25;
        displayPlots = false;
    case 3
        displayPlots = false;
end

%settings
numplots = 6;
skipTimeFactor = 5;

%Filde for QRS
Fc = 10;
b = fir1(50, 2/Fs * Fc, 'high');
%remove mean, plot
meanSig = mean(signal);
signal = (signal - meanSig);
if(displayPlots)
    subplot(numplots,1,1);
    plot(signal);
    title('mean removed signal');
end

fSignal = filtfilt(b,1,signal);
%fSignal = fSignal.^2;
if(displayPlots)
    subplot(numplots,1,1);
    hold on;
    plot(fSignal, 'red');
    title('mean removed signal, filtered signal');
    hold off;
end

% %some info
% max(signal)
% min(signal)
% range(signal)

%take derivative,
%diffsig = diff(fSignal);
%do not take derivative
diffsig = fSignal;
```

```

diffclamp = (diffsig > std(diffsig) * 3); %2.5 gets all three but misses beats

index = 1;
numBeats = 0;
outsig = signal;
triggers = zeros(1, length(diffsig));
windowLength = floor(Fs*0.05);
tLength = floor(Fs*Tt);
avgsig = zeros(1, 2*tLength);
while index < length(diffsig) %loop until end of array.
    if(diffclamp(index) == 1) %if triggered, set start index, set end index, copy data to
    ouptut array, move to end of copy period
        %compute start index
        startIndex = max([1, index - windowLength*3/4]); %compute start
        endIndex = min([index + windowLength*5/4, length(diffsig)]); %compute end
        outsig(startIndex : endIndex) = linspace(signal(startIndex), signal(endIndex),
        endIndex - startIndex + 1); %copy data
        triggers(startIndex : endIndex) = ones(1, endIndex - startIndex + 1);
        numBeats = numBeats + 1;
        beat = signal(endIndex : min([length(diffsig), endIndex+2*tLength-1]));
        if(length(beat) == length(avgsig))
            avgsig = avgsig + beat/numBeats;
        end
        index = index + windowLength * skipTimeFactor;
    else
        index = index + 1;
    end

end

maxsig = max(signal);
if(displayPlots)
    subplot(numplots,1,1);
    hold on;
    plot(triggers.*maxsig, 'green');
    title('mean removed signal(blue), filtered signal(red) [Fc = 10Hz, 50 Tap FIR],
    triggered time (green)');
    hold off;
    subplot(numplots,1,2);
    plot(outsig);
    title('QRS Removed signal');
end

% now we have QRS free signal. Extract T wave. Remove P wave

% now we have a PQRS free signal. extract T wave

%first high pass

```

```

Fc = 1.5;
b = fir1(250, 2/Fs * Fc, 'high');
fSignal = filtfilt(b,1,outsig);
Fc = 10;
b = fir1(250, 2/Fs * Fc);
fSignal = filtfilt(b,1,fSignal);

if(displayPlots)
    subplot(numplots,1,3);
    plot(fSignal);
    title('Filtered S signal');
end

% blank out all of time except for 250ms after end of QRS
difftriggers = diff(triggers);
difftriggersneg = (difftriggers < 0)*1.0;
tWindow = 250/1000; %(250ms)
windowPeriod = ones(1,tWindow * Fs);
diffperiod = conv(difftriggersneg,windowPeriod);
diffperiod = diffperiod(1:length(outsig));
tattack = fSignal .* diffperiod;

% now trigger
tclamp = (tattack > (std(tattack) * 1))*1.0; %2.5 gets all three but misses beats
diffTpos = (diff(tclamp) > 0)*1.0;
if(displayPlots)
    subplot(numplots,1,3);
    plot(signal);
    hold on;
    title('T Wave Extracted signal');
    subplot(numplots,1,3);
    hold on;
    plot(triggers.*maxsig, 'green');
    plot(tclamp*maxsig, 'magenta');
    hold off;
end

Spositions = find(difftriggersneg);
Tpositions = find(diffTpos);

if(Tpositions(1) < Spositions(1)) %if there's a T trigger before the first s, drop it.
    Tpositions = Tpositions(2:end);
    disp 'drop first for T before s'
end

if(Tpositions(end) < Spositions(end)) %if the last T comes before the last S, we have an
unfinished beat, drop it
    Spositions = Spositions(1:end-1);

```

```

    disp 'drop last for s after t'
end

if(displayPlots)
    subplot(numplots,1,4);
    hold on;
    title('Triggers Blue = S, Red = T');
    subplot(numplots,1,4);
    plot(difftriggersneg*maxsig);
    hold on;
    plot(diffTpos*maxsig, 'red');
    hold off;
end

intervals = [];
for beatIndex = 1:(length(Spositions)-1) %make sure each T is before the next S
    if(Spositions(beatIndex+1) > Tpositions(beatIndex)) %next QRS comes after next T
        intervals = [intervals Tpositions(beatIndex) - Spositions(beatIndex)];
    end %otherwise, skip to next S
end

intervals = intervals / Fs * 1000; %convert to ms

minintervals = ones(1, length(intervals)) * mean(intervals);

%first, drop a Q
if(displayPlots)
    subplot(numplots,1,5);
    plot(intervals);
    hold on;
    plot(minintervals, 'red--');
    plot(minintervals + std(intervals), 'red-.');
    plot(minintervals - std(intervals), 'red-.');
    title(strcat('QT Interval mean: ', num2str(mean(intervals)), 'ms standard deviation:✓',
    ', num2str(std(intervals)), 'ms'));
    ylabel('milliseconds');
    axis tight;
    hold off;
end

end

```

Bibliography

- [1] US Government. The International Traffic in Arms Regulations, 2014.
- [2] The Open Source Initiative. The Open Source Definition.
- [3] XILINX. Spartan-6 FPGA SP601 Evaluation Kit, 2017.
- [4] Picaxe. What is picaxe.
- [5] David Kushner. The Making Of Arduino, 2011.
- [6] Sparkfun. What is an Arduino.
- [7] Ben Fry. computational information design. Technical report, Massachusetts Institute of Technology, Cambridge, MA, 2004.
- [8] Atmel. Atmel Studio.
- [9] Texas Instruments Inc. Download CCS.
- [10] Arduino. Arduino - Libraries.
- [11] Paul Hill. Ten statistics that reveal the size of the maker movement, 2015.
- [12] Will Strauss. Early history of the DSP chip. *EE Times Asia*, (June), 2003.
- [13] Texas Instruments Inc. Fixed-Point Digital Signal Processor.
- [14] Eclipse Foundation. Eclipse Desktop & Web IDEs.

- [15] Texas Instruments Inc. Code Composer Studio Development Tools v3.3 Getting Started Guide. Technical report, 2006.
- [16] Texas Instruments Inc. C5535/C5545 eZdsp USB Stick Development Kit,.
- [17] Matthew Barber. On Teaching Digital Signal Processing to Composition Students. Master's thesis, University of Rochester, 2008.
- [18] Todd Atkins. Introduction to Simulink. Technical report, 2010.
- [19] C A Cansalar, E Mavi, and C Kasnakolu. Simulation time analysis of MATLAB/Simulink and LabVIEW for control applications. In *2015 IEEE International Conference on Industrial Technology (ICIT)*, pages 470–473, mar 2015.
- [20] ALFASOFT. SOLIDTHINKING EMBED 2016: VISUAL ENVIRONMENT FOR MODEL BASED DEVELOPMENT OF EMBEDDED SYSTEMS.
- [21] Signal Wizard Systems. Signal Wizard 3.
- [22] PJRC. Teensy Audio Library.
- [23] PJRC. Audio System Design Tool for Teensy Audio Library, 2016.
- [24] PJRC. Audio Adaptor Board for Teensy 3.0 - 3.6.
- [25] Raspberry Pi Foundation. About Us.
- [26] Pure Data. Pure Data.
- [27] Interaction Design Foundation. Make it Easy on the User: Designing for Discoverability within Mobile Apps, 2016.
- [28] Arduino. Arduino - ArduinoBoardYun, 2017.
- [29] EpicTinker. ArchPro.
- [30] Adafruit Industries. Adafruit Proto Shield for Arduino, 2015.

- [31] Burr-Brown / Texas Instruments. 16-Bit, 4-Channel Serial Output Sampling ANALOG-TO-DIGITAL CONVERTER. Technical report, 2003.
- [32] Stanford Research Systems. SR1+ Operation Manual. Technical report, 2014.
- [33] Martin Nawrath. Lab 3 - Laboratory for Experimental Computer Science. 2009.
- [34] Arduino. Arduino - Analog Write, 2017.
- [35] Digilent Inc. Analog Shield: High-performance Add-on Board for the Arduino Uno, 2017.
- [36] Arduino. Arduino Playground - Shield Pin Usage.
- [37] Adafruit Industries. RGB LCD Shield Kit.
- [38] Adafruit Industries. 2.8 TFT Touch Shield.
- [39] Texas Instruments Inc. TLV320AIC3204 Ultra Low Power Stereo Audio Codec. Technical report.
- [40] Texas Instruments Inc. TLV320AIC3204 Application Reference Guide. Technical report, 2012.
- [41] Texas Instruments Inc. TCA6416A Low-Voltage 16-Bit I²C and SMBus I/O Expander With Voltage Translation, Interrupt Output, Reset Input, and Configuration Registers. Technical report, 2015.
- [42] Amazon. No Title, 2017.
- [43] Arduino. Arduino - FAQ.
- [44] W. Gong W.-S. Gan, Y.-K. Chong and W.-T. Tan. Rapid prototyping system for teaching real-time digital signal processing. *Education, IEEE Transactions*, 43(1):19–24, 2000.
- [45] Robert Wessels. Energia.

- [46] Ben Fry Casey Reas. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press, 2014.
- [47] Texas Instruments Inc. Code Generation Tools for Texas Instruments Processors: Downloads.
- [48] Texas Instruments Inc. C5535/C5545 eZdsp USB Stick Development Kit.
- [49] Free Software Foundation, Inc. AVRDUDE - AVR Downloader/UploaDEr, 2004.
- [50] Texas Instruments Inc. TMS320C55x Chip Support Libraries (CSL).
- [51] Texas Instruments Inc. TMS320C55x Chip Support Library API Reference Guide, 2004.
- [52] R. Yates and R. Lyons. Dc blocker algorithms [dsp tips tricks]. *IEEE Signal Processing Magazine*, 25(2):132–134, March 2008.
- [53] Mark A. Wickert. REAL-TIME DSP BASICS USING ARDUINO AND THE ANALOG SHIELD SAR CODEC BOARD. In *IEEE Signal Processing and Signal Processing Education Workshop (SP/SPE)*, 2015.
- [54] Analog Devices. Fundamentals of Direct Digital Synthesis, 2008.
- [55] Open Music Labs. ArduinoFHT, 2016.
- [56] MathForum. Lissajous Curve, 2012.
- [57] Gregory Kovacs. Syllabus EE122A. Technical report, Stanford University, 2016.
- [58] Peter Heß. A “Ready Made” Modules - The Analog Shield, 2015.
- [59] Eric Bogatin. Monthly Archives: October 2015, 2015.
- [60] Lahav A. Zimmerman E, McMahon E, Doheny L, Levine P. Transmission of biological maternal sounds. *Journal of Pediatric and Neonatal Individualized Medicine*, 2012.

- [61] Fernando A. Mujica, William Esposito, Alexander Gonzalez, Charles R. Qi, Chris Vassos, Maisy Wieman, Reggie Wilcox, Greg. Kovacs and Ronald W. Schafer. Teaching digital signal processing with Stanford's lab in a box. In *IEEE Signal Processing Workshop*, 2015.
- [62] Woody Herman. DSPFuzz A Guitar Distortion Pedal Using The Stanford DSP Shield. Technical report, 2015.
- [63] Matthew Hu. Heart-Rate Detector Report. Technical report, 2015.
- [64] Terry Kong. Phase Vocoder Implementaiton with FLWT and TD-PSOLA. Technical report, 2015.
- [65] Sevy Cheng, Hanchel and Harris. Tempo Estimation and Manipulation. Technical report, 2015.
- [66] Eric D. Scheirer. Tempo and beat analysis of acoustic musical signals. *The Journal of the Acoustical Society of America*, 103(1):588–601, 1998.
- [67] Keysight Technologies. Concepts of Orthogonal Frequency Division Multiplexing (OFDM) and 802.11 WLAN.
- [68] Bitalino. BITalino - Biomedical Equipment.
- [69] MySignals. MySignals HW v2 - eHealth and Medical IoT Development Platform for Arduino.
- [70] Association for the Advancement of Medical Instrumentation. AAMI Stadar: ANSI/AAMI EC11:1991/(R)2001. Technical report, Association for the Advancement of Medical Instrumentation, Arlington, 2001.
- [71] et al. Buendía-Fuentes F, Arnau-Vives MA, Arnau-Vives A. High-Bandpass Filters in Electrocardiography: Source of Error in the Interpretation of the ST Segment. *ISRN Cardiology*, (706217), 2012.

- [72] Sangeeta Bagha. A Real Time Analysis of PPG Signal for Measurement of SpO₂ and Pulse Rate. *International Journal of Computer Applications*, 36(11):45–50, 2011.
- [73] Neil Townsend. Electrical Respiration Instrumentation. Technical report, 2001.
- [74] P. Castiglioni, A. Faini, G. Parati, and M. Di Rienzo. Wearable seismocardiography. In *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 3954–3957, Aug 2007.
- [75] Gert Mike Chi, Yu; Jung, Tzyy-Ping; Cauwenberghs. Dry-Contact and Noncontact Biopotential Electrodes: Methodological Review. *IEEE Reviews in Biomedical Engineering*, 14(February), 2010.
- [76] Anthony Atkielski. File:SinusRhythmLabels.svg, 2007.
- [77] Jaakko Malmivuo & Robert Plonsey. *Bioelectromagnetism - Principles and Applications of Bioelectric and Biomagnetic Fields chapter 6*. 2002.
- [78] Linear Technology. LT 1167 Datasheet: Single Resistor Gain Programmable, Precision Instrumentation Amplifier. Technical report, Linear Technology, Milpitas, 1998.
- [79] John G. Webster. Chapter 5, Biopotential Electrodes. In *Medical Instrumentation*, pages 189–215. John Wiley & Sons, Inc., Danvers, MA, 2010.
- [80] Malcolm S. Thaler. *The Only EKG Book You'll Ever Need*. Lippincott Williams & Wilkins, Philadelphia, PA, 2014.
- [81] Miguel. Spinelli, Enrique & Pallas-Areny, Ramon & Mayosky. AC-Coupled Front-End for Biopotential Measurements. *IEEE transactions on bio-medical engineering*, 50(5):391, 2003.
- [82] Silicon Labs. AN119: Calculating Settling Time for Switched Capacitor ADCs. Technical report, Austin.

- [83] Ippeita Dan Valer Jurcak, Daisuke Tsuzuki. 10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems. *NeuroImage*, 34(4):1600–1611, 2007.
- [84] Texas Instruments. Analog Front-End Design for ECG Systems Using Delta-Sigma ADCs. Technical report, Dallas, TX, 2010.
- [85] Digikey Inc. Search for Instrumentation Amplifiers.
- [86] Texas Instruments Inc. ADS129x Low-Power, 8-Channel, 24 Bit Analog Front-End for Biopotential Measurements. Technical report, 2015.
- [87] Analog Devices Inc. Data Sheet ADAS 1000/ADAS 1000-1/ADAS 1000-2. Technical report, 2014.
- [88] Texas Instruments Inc. ADS1258 16-Channel Analog-to-Digital Converter. Technical report, 2011.
- [89] Peter Selvey. ECG Filters - MEDTEQ. Technical report, 2017.
- [90] National Heart Lung and Blood Institute. Long QT Syndrome.
- [91] Page CB. Isbister GK. Drug induced QT prolongation: the measurement and assessment of the QT interval in clinical practice. *British Journal of Clinical Pharmacology*, f76(1):48–57, 2013.
- [92] P. Laguna, R. G. Mark, A. Goldberg, and G. B. Moody. A database for evaluation of algorithms for measurement of qt and other waveform intervals in the ecg. In *Computers in Cardiology 1997*, pages 673–676, Sept 1997.
- [93] The Math Works Inc. Filter Frames of a Noisy Sine Wave Signal in Simulink.
- [94] Rolin Moe. The brief & expansive history (and future) of the MOOC: Why two divergent models share the same name. *Current Issues in Emerging eLearning*, 2(1):24, 2015.
- [95] Raspberry Pi Foundation. What is a Raspberry Pi.

- [96] Texas Instruments Inc. C5517 Evaluation Module (EVM).