Oscilloscope BoosterPack (Low Cost)	Rose-Hulman

TI Innovation Challenge 2015 Project Report

Team Leader:	Eric Taylor, taylorem@rose-hulman.edu
Team Members:	Raymond Montgomery, montgora@rose-hulman.edu
Advising Professor:	Mark Yoder, yoder@rose-hulman.edu
Video	https://www.youtube.com/watch?v=cYiwDeTmkGY
Texas Instruments Mentor (if applicable):	Trey German,treygerman@ti.com
Date:	5/11/2015

Qty.	**List all TI analog IC and TI processor part number and URL**	**1) Explain where it was used in the project? 2) What specific features or performance made this component well-suited to the design?**
2	THS4121	This part was used for the Single-To-Differential conversion on the front end. Two were needed to get two channels. This was well suited because it ran of 3.3V and had the proper input range for the inputted signal and an adjustable common mode voltage to match with the LMH6882's common mode voltage.
1	LMH6882	This was used to have programmable gain on both of the channels. This was used because it has two channels on a single chip and had a wide range of gain settings which is useful for our application. The parallel bus also made it easy to control.
2	OPA140	These were used to buffer the common mode voltage coming out of the ADC into the THS4121 and LMH6880. These were used because of the proper input range and we just were looking for the cheapest general purpose op amp possible since no frequency specs were needed.
1	ADS5237	This was the ADC of the circuit used to sample both of the channels. It was used because of the fast sample rate of 65 MSPS and having a high 10 bit resolution. The 65 MSPS was around the limits of the EPI, and 10 bits was about our target when designing the front end. (Any more precision would be lost in the quality of the front end)
1	TM4C1294NCPDT	This was the microcontroller used to control our project. Since we are making a BoosterPack designed to attach to a Launchpad we picked the Connected LaunchPad because of the large amount of Pins and EPI bus capability,
3	OPA890	These were used as an amp to offset the triggering signal and as a comparator to compare the signal to a specific trigger level. These chips were selected because of the low amount of skew and fast switching capabilities.
1	TL7660	This was used as a negative source for the triggering circuit. This part was used because it was the cheapest part that did the intended job.

- https://www.youtube.com/watch?v=waJ3gXQC8M0
- https://www.youtube.com/watch?v=UX_YCJf1Lo4
  https://www.youtube.com/watch?v=qeyE9i_el2E

GitHub: https://github.com/steelerfan107/BoosterPackOscilloscope

Project abstract (a short high level written description of the design and motivation behind project), 1,000 words max:

This project's objective was to design an oscilloscope BoosterPack that is compatible with TI LaunchPad series that is a low cost alternative to the typical desktop oscilloscope. This project was open source so anyone can learn from the hardware and software design and be able to expand upon them. The final product must be able to function as a reliable oscilloscope at an affordable cost to both hobbyists and students. As a low end design, the oscilloscope BoosterPack needs to push its cost down as much as possible while having the features of a typical oscilloscope. The main goal was

to create a BoosterPack that has two measurement channels that can take a +/- 15V signal and has a bandwidth of 7MHz for under $50.

The design of this project had many aspects involving hardware and software. The hardware design consisted of three main circuits that we would need to design: sampling, front end, and triggering. The front end consisted of single to differential conversion along with adjustable gain to go into the sampling circuit. The sampling circuit consisted of an ADC to sample the signal from the front end and send it to the microcontroller. Finally the triggering circuit which takes a signal from the front end and compared it to a set level adjusted by the user and tells the microcontroller when to start sampling. All of this hardware will be on one board that is BoosterPack compatible and can be attached to the Connected LaunchPad. This allows for a display to be attached on top of this BoosterPack to display the signals. For this specific project we used a premade LCD BoosterPack and SPI controllers.

The other aspect was the software to control the device. For this we used Energia for the development and contained four parts. The first part was voltage and time deviation control. This was just done using the on board LaunchPad user buttons. The next was the amplifier control which was also just GPIO control. The next was the display drivers which were adapted from C drivers that came with the premade BoosterPack. Finally the sampling control with consisted of EPI, uDMA, and some other general code. The EPI was used to read in the 20 bit (both channels) quickly and the uDMA was used to store them in memory. Then two buffers were made to store the split samples and to determine if a certain samples changed positions on screen to keep screen refreshing to a minimum. However for our project we could not get the uDMA to work so we used timer interrupts to control how fast we took in samples. This greatly reduced the bandwidth of what we could measure, but since this project is open source and on GitHub there is the opportunity to add this feature in later.

Table of Contents

Contents

List of Tables and Figures

Project Description

The project was to create an oscilloscope BoosterPack for TI's Launchpad series which had the following features: two measurement channels, hardware triggering, user control, 6.5 MHz bandwidth, and +/ 15V input swing for all under $50. The reasoning behind the selection of these features can be seen below. In our final product we were able to accomplish most of what was needed but not all. The BoosterPack did have two working channels and a hardware trigger for under $50, however the software could not sample fast enough to get the 6.5 MHz bandwidth and adjustments are needed on the front end to get a full +/- 15 V input swing. The final board we made had a 200 kHz bandwidth and +/- 1.5V input swing.

2.1 Description of the Primary Stakeholders and Features

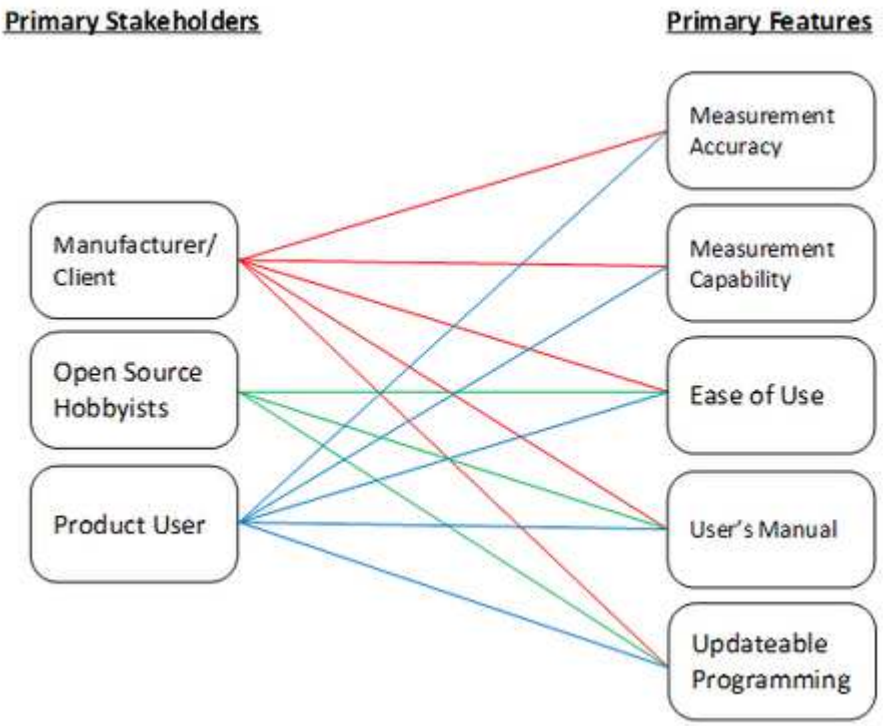


Figure 1. Primary Stakeholders and Features

The final product which consists of our physical circuit, the code that runs it, and the open source material all of which are determined by the primary stakeholders and features of our system as seen in Figure 1. The client, TI, provided this project submission and therefore cares about all the deliverables they requested. Anyone who uses the oscilloscope BoosterPack wants a functional final product and would want to reference the open source material for debugging errors and exploring the design of the product. Those who do own the product (like hobbyists and other recreational researchers) can still view the open source materials for studying circuit and code designs. The remaining features describe the actual function of the oscilloscope BoosterPack. 'Measurement Accuracy' and 'Measurement Capability' are distinct and necessary features for the final product. The accuracy describes the how well the oscilloscope BoosterPack's measurements are to the actual signal being measured, while capability is a necessary step in determining signal accuracy because we need confirm that the data displayed on the oscilloscope is measurement data. Since the program on the circuit's microcontroller can be changed, the user can see how making changes to the program can affect the operation of the oscilloscope and other programs can be loaded onto the LaunchPad so it can be used for different projects. As the project becomes easier to use, the user enjoys using the oscilloscope BoosterPack more which is why ease of use is a primary feature.

2.2 Description of the Solution

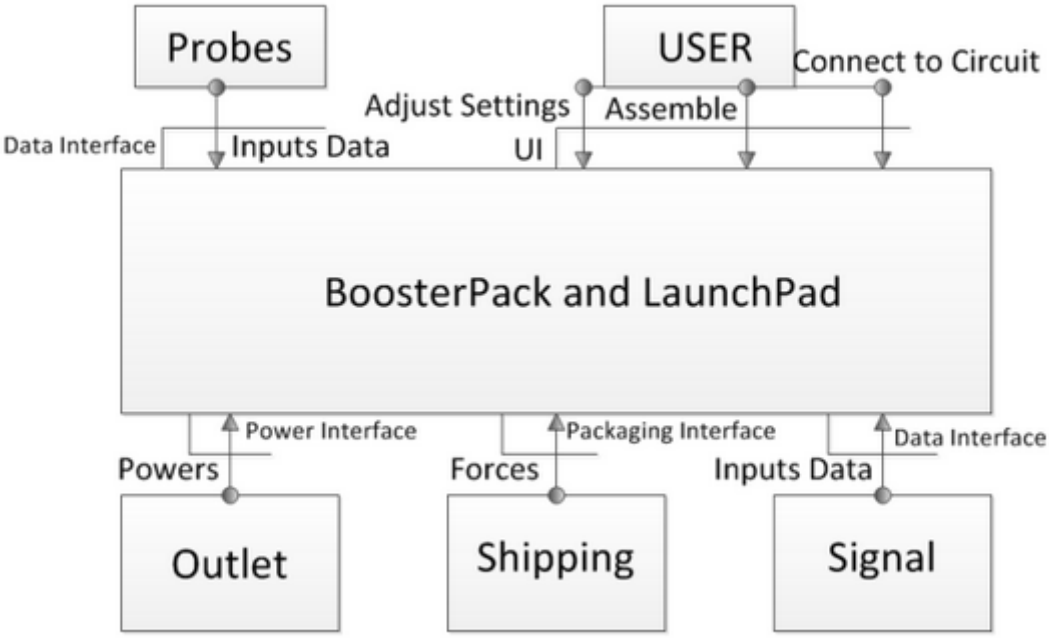


Figure 2. Domain Diagram

In Figure 2 the domain model of the project is presented. In this model the outside actors of the system are identified along with their interactions with the system. The first actor are the probes that are used in conjunction with the system. This directly related to measurement accuracy, ease of use, price limit, and measurement capabilities. The probes are what connects the signal to the circuit and have characteristics like input impedance and signal attenuation. These can effect what type of signal that could be measured and how accurate the system can be. The probes are also a separate component to reduce cost of the final package, and the user can use ones to their specification. For the final board design any BNC probe could be attached to measure signals. However when the 10x setting is used the signal does not go through. Further research is needed to figure out why this happens.

The next outside actor is the user which adjusts the settings, assembles/updates, and in general uses the scope. This directly relates to the ease of use, instruction manual, measurement capabilities, and price limit. The user needs to be able to operate the scope with ease, and should have a similar interface to other scopes on the market. This coincides with the user manual which explains the use of the scope to the user. Finally the user wants to be able to buy the product at a reasonable price given the performance the scope has. This was satisfied using buttons that came attached to the LaunchPad and a potentiometer attached to the BoosterPack. These controls allowed the user to adjust time and voltage per deviation, and also adjust the trigger level. Using controls that came with the LaunchPad also reduced the total cost of the board.

The next outside actors are outlet which directly effects power consumption, shipping which involves how the product is packaged, and the signal which related to accuracy and capability. The outlet determines how much power the scope can use. For power the final board can be powered by any micro usb source including portable battery packs. The shipping of the product relates the manufacturability and how the product is packaged. This was accounted for in the hardware design of the board. Finally the signal shape, size, and characteristics need to be known so the scope can be made to measure the variety of signals that it needs.

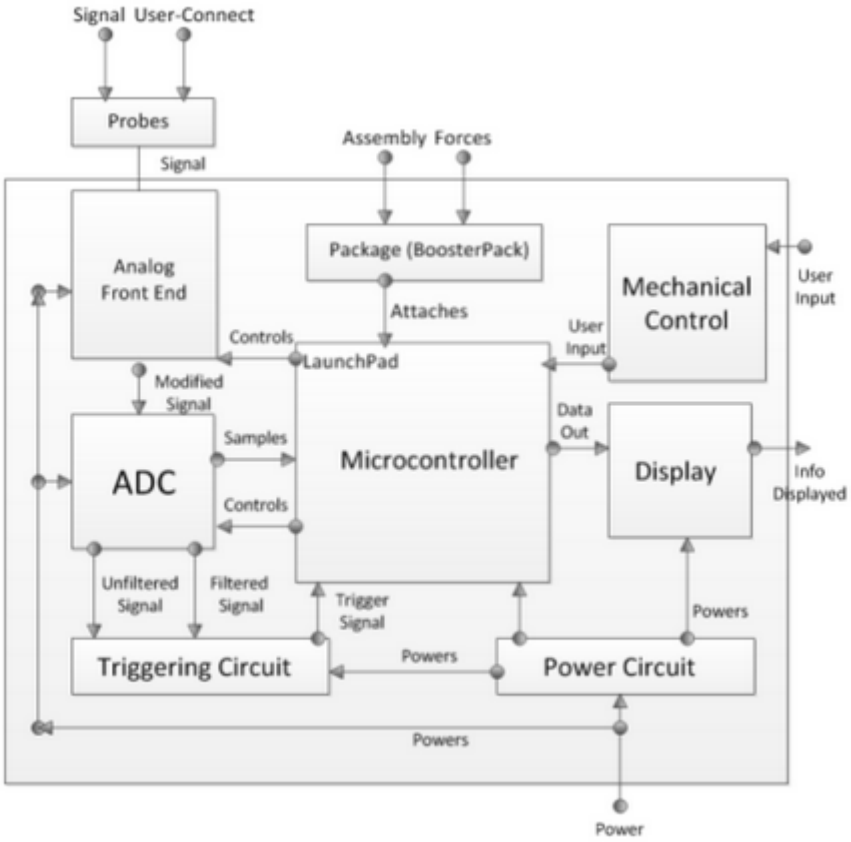


Figure 3. Logical Architecture

In Figure 3 we see the logical architecture of the system which describes how the scope will respond to the outside actors and shows how the system will function in general. Following the flow of the signal first comes the analog front end which accepts the signal from the scopes and proceeds to attenuate/amplify the signal, apply the proper common mode voltage, and make it differential. The signal is also put through a filter to reduce high frequency noise. The signal then proceeds to the ADC where it is sampled and then read into the memory of the microcontroller. The microcontroller the uses these samples and based on the user input displays the signal on a LCD screen.

The front end involves many parts, first of which is the THS4121 which does the single to differential conversion. This part was a cheap general part that is able to handle +/- 1.5V swings which was appropriate because we planned 10x probes working with the board which they did not. With 10x probes the +/- 1.5V swing would be enough for a +/- 15V input. This part also had the necessary bandwidth to handle 10 MHz signals.

The next part used was the programmable gain amplifier, LMH 6882. This part was first suggested by the client and worked well for our purpose. This amplifier has up to 25dB gain allowing for the measurement of small signals. This part also had two channels which reduced the part count on the board reducing cost.

Next came the sampling system or ADC. The ADC chosen was the ADS5237. This part had a sampling rate of 65 MSPS which allowed the measurement of signals up to 7MHz assuming the industry guideline of having 10 samples per period to have a good measurement on a signal. It also was selected because it had two channels which reduced part count and was relatively cheap compared to other ADCs that were up for consideration. Finally it had 10-bits of resolution which the client stated he wanted in the planning process.

The microcontroller which takes in the samples was satisfied using the Tiva C connected LaunchPad from TI. This board was needed because of the extra pins that were included which were needed to satisfy all of the requirements. This board also had EPI (External Peripheral

Interface) which was a parallel bus that was needed to receive the 20-bit data signal for both channels at appropriate speeds.

Other blocks include the triggering circuit which takes in the filtered and unfiltered signal and tells the microcontroller when to start sampling based on a desired trigger level. This was done using cheap general purpose op-amps because of their low cost, and had satisfactory bandwidth for our features. The power circuit takes in the power from the outlet or USB and provides the microcontroller and the circuit with power. The power circuit was on the LaunchPad itself and no further circuitry was needed. The package block includes how it has to be manufactured to be able to interact with the Launchpad. This will be done by following BoosterPack templates and specifications. Finally the mechanical control which includes buttons which are on the Launchpad and a potentiometer which is on the BoosterPack.

Finally the display requirement was satisfied by an external BoosterPack made by a third party. This allowed us to focus on other features while satisfying a requirement at low cost. A simple 2.2" LCD controlled by SPI was used because of its low cost and simplicity to control.

This solution is able to deal with all of the major features which includes measurement accuracy, ease of use, measurement capabilities, updateable programming, and user manual. The measurement accuracy and capabilities are determined by the front end and ADC. Ease of use is taken care of by the mechanical control, display, and probes. However it couldn't fully handle the features because the software wasn't able to take in samples fast enough. The hardware could handle this requirement but software could not. Other than this detail the hardware selected was able to satisfy all of the requirements

Hardware Design

Front End

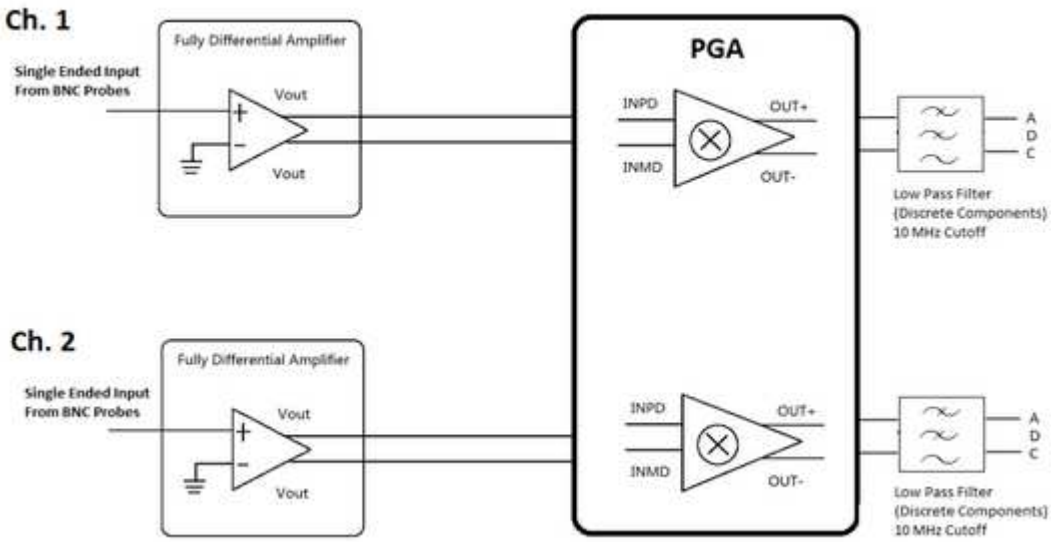


Figure 4. Front End Subsystem

For the Analog Front End, there is a BNC probe inputting the single ended signal. The reason we want the probes is to attenuate the signal by 10x and give the input impedance of 1MOhm. This is done because looking into the buffer there is a known high impedance of 1 Mohm and known attenuation.

Next, we have a fully differential amplifier (THS4121). Because the programmable differential amplifier requires two differential inputs, so we use the THS4121 to transfer the single signal to two differential input signals.

Last we have the dual channel differential PGA (LMH6882). The PGA Amplifies small signals so the ADC can measure the signals with proper resolution. This PGA is controlled by the microcontroller through parallel logic telling how much gain the signal need.

Finally there is a low pass filter to get rid of any noise to the ADC with a cutoff frequency of about 10 MHz.

Triggering Circuit

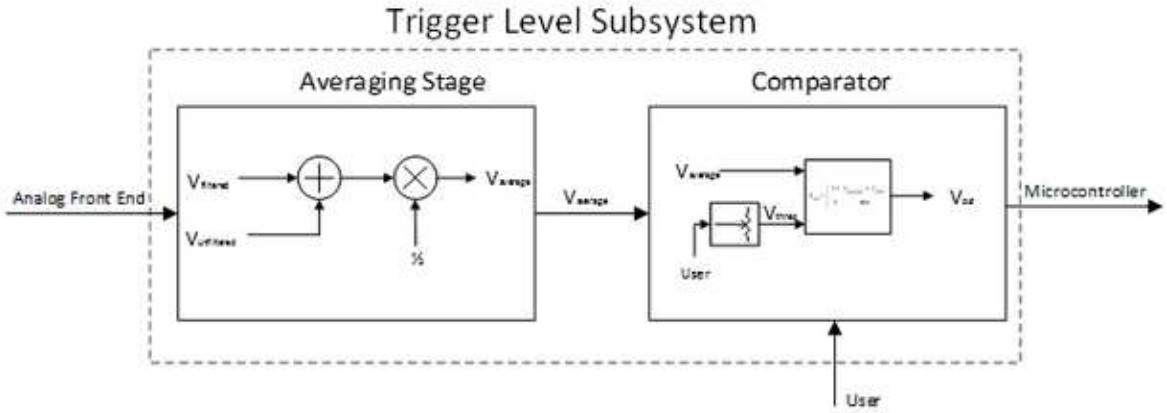


Figure 5. Trigger Level Subsystem

After a channel receives a signal. The Analog Front End scales the signal down so the op-amps in the Triggering Circuit can manipulate the input signal within the supply voltages. The user controls the potentiometer with a knob, or a similar control structure. The output of the Triggering Circuit is an array of logical values based on the sampling rate of the microcontroller and if there is one logical high in the array then the waveform is displayed. Logical high is an output of 5V and a logical low is an output of 0V

Sampling Subsystem

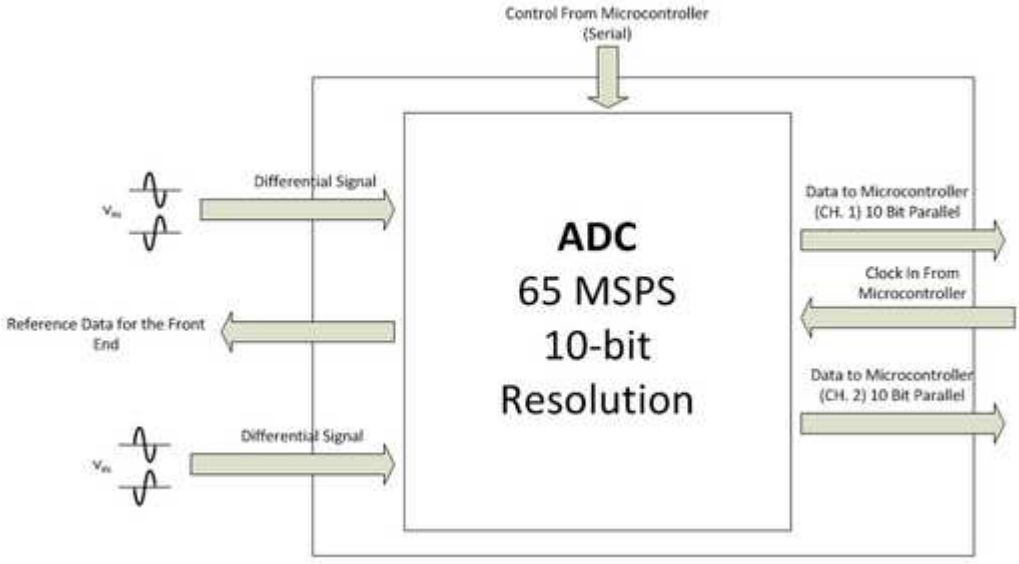


Figure 6. Sampling Subsystem

For our circuit we are going to use the ADS5237 ADC from TI. This chip is a dual channel, 10-bit resolution ADC that can sample at 65 MSPS. This will enable us to measure signals up to 6.5 MHz. This part has a listed use case for test equipment so that also gives reassurance it would be good to use in this case. The chip requires a clock signal from the microcontroller to tell it when to

sample, and the data is sent back to the microcontroller over two 10-bit parallel lines. The analog signal coming in comes from the ADC and needs to be a differential signal within 0 to 2 V peak to peak.
Our ADC will not need serial control for now so it is disabled in the schematic for now.

Microcontroller and Software

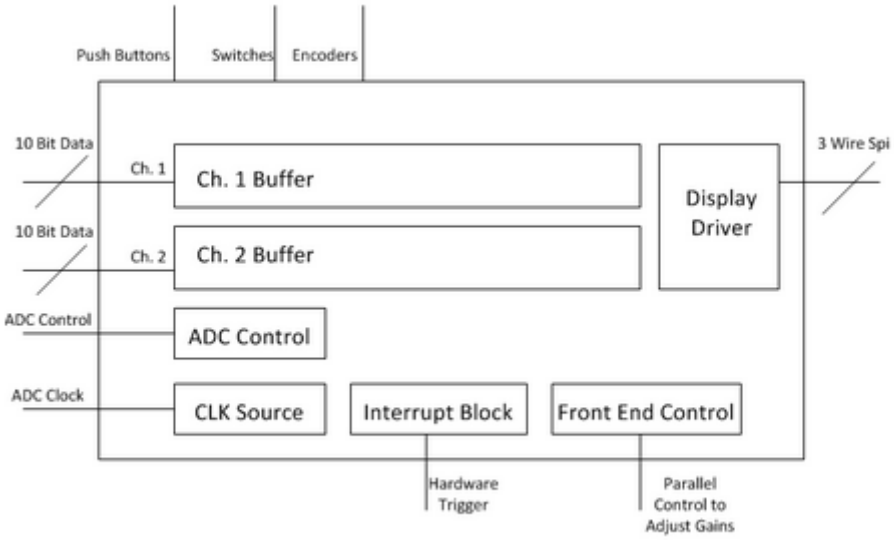


Figure 7. Software Subsystem

The microcontroller is the center of the project and will hold all of the software. We are using the Tiva C microcontroller that comes on the Connected series of Launchpad. The microcontroller is responsible for taking in the sampled data through an EPI (External Peripheral Interface) bus and storing it into a buffer after the trigger interrupts the microcontroller.

Knowing things such as voltage per division and time per division shown on the screen it will also control the sampling rate through increasing/decreasing the clock frequency to the ADCs. It will also control the amplification through the programmable gain amplifier which is a simple parallel interface.

The microcontroller also takes the user input from mechanical controls such as the push button, rotary encoders, and slide switches, and displays data on a LCD display which it will connect to through the built in SPI drivers.

Testing Methids

Much of the formal testing of the project involved testing spate subsystems of the project. The first subsystem tested was the analog front end which was our black box testing.

For the black box testing we wanted to confirm the input requirements to the scope. This includes features like how high frequency of a signal can be inputted to the scope, and how large of a signal. This testing revolved around the front end because that is what takes the input signal and delivers it to the ADC for sampling.

For the setup of this testing the final board was made so a scope probe can take an input signal and there were testing wires soldered to the output of the analog front end right before the ADC. This allows us to see when a certain signal is inputted that the signal gets to the ADC correctly to be sampled. Here we are mostly looking for shape because depending on the settings of the programmable gain amplifier the true amplitude is unknown and is only known to the microcontroller which provide the correct gain factor to the samples. For the sake of this testing the lowest settings of gain were used for the programmable gain amplifier.

To start testing the functionality was confirmed with a 50mV input which is reality simulates a .5 V input because of the 10x probes (probes used were set at 1X). At 50 mV a sine wave was put through

at 1kHz, 100kHz, 1MHz, and 10 MHz in addition to a square wave at 1MHz and a ramp wave at 200kHz. Again here we want to confirm the correct shape going into the ADC.

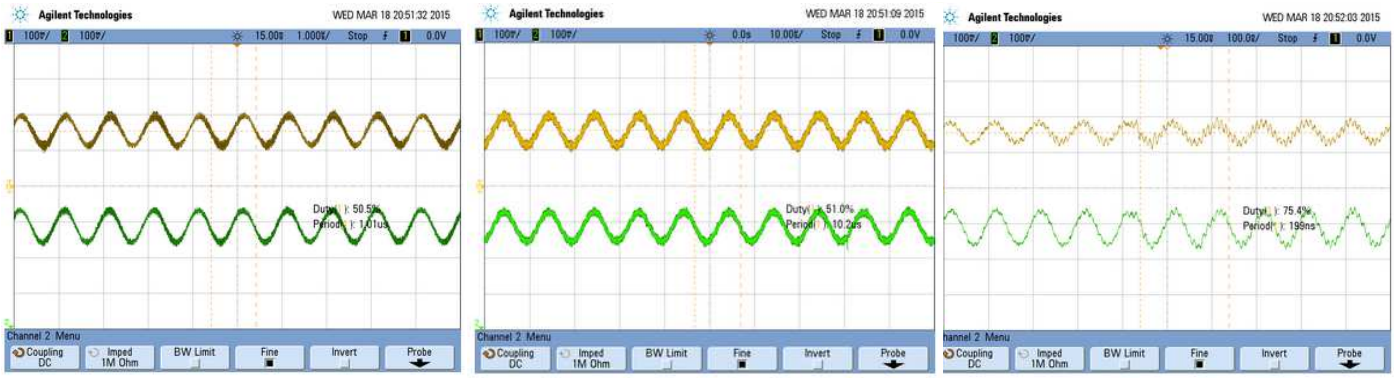


Figure 8. 50mV sine at (100kHz, 1MHz, 10MHz) (Top – Input to PGA, Bottom – Output of PGA)

Here is an example progression of the testing for the analog front end. In the above figure we specifically looked at small signals going into the front end and looking at what the ADC would see. Many configurations of this test were done at different frequencies and shapes but showing all would take too much space. Alongside this test our group also tested the triggering circuit individually.

For full system testing we assembled the oscilloscope and used it like anybody else would. We testing different shape, sizes, and frequencies of signals going into the scope. When we inputted a signal we tested what the screen was displaying. Many of these tests can be seen in the video and typically involved visual testing. Since our group ran out of time to finish the project in its entire little formal testing was done on the final system, it was mostly visual and user tested. The testing did go over well and showed what we got done worked, however it also did show what could be added to this project to make it better and where the shortcoming were. In the end we were able to input a +/- 1.5V signal up to 200 kHz, both channels were working, had adjustable voltage/time scaling, and had an adjustable hardware trigger. Fixes and further steps could be seen in the recommendations section.

Conclusions

In the end our group was able to get the core functionality working along with many features such as the controllable trigger level and adjustable time/voltage scaling. It did fall short of the expected goals however. The Brightside is that small additions to software such as uDMA will improve the sampling speed, and addition of a selectable attenuator could improve input amplitude range.

Future Work

Our project has fallen short of its expectations and its goals. The hardware is capable of handling the range of input signals (frequencies up to 7MHz) but the software can't handle samples fast enough. A micro DMA method in software would help to handle samples faster and enables the oscilloscope BoosterPack to appropriately handle and display the higher frequency signals by writing the samples directly to memory when they are received. The software also doesn't dynamically change the gain of the PGA to amplify smaller signals. However the control signals to do so are in hardware. More software would have to be added to allow the oscilloscope BoosterPack to measure the range of signals that we designed for. Finally adding a selectable attenuator to the front end would allow the hardware to accept larger signals. These recommendations would help the project to better meet the stakeholder features for Measurement Accuracy. Overall the core functionality works, meeting the Measurement Capability, Open Source Material, Ease of Use, and Updateable Programming features, but adding more software would allow the board achieve all its goals, and adding some hardware and software could help the project to meet these goals.

Resources and Bibliography

Annotated List of Research Memos

[1] Yaohui Wang, *Research of Analog Front End Design*. October 7, 2014.
Description of the purpose and operation of an analog front end and looked into characteristics our group should look at when making our analog front end.

[2] Eric Taylor, *Hold-Off Triggering,* September 28, 2014.
A description of what hold off triggering is and how it works. This provided understanding of an oscilloscope feature that could be put into the scope at a later date.

[3] Raymond Montgomery, *Oscilloscope Triggering.* October 15, 2014.
This provided research into how our project will handle the triggering in hardware and gave understanding into which parts we should use and the general circuit subsystem that had to be made. This deals more with triggering hardware whereas the previous memo is software triggering features.

Annotated Biblliography

[1] I. King , *High Performance Analog Front Ends,* Analog Edge, Vol. IV Issue 1.
This source was used when the group didn't have a good understanding of the purpose of an analog front end, and what characteristics of one made it a high performing one. This provided characteristics for the analog front end that our group could shoot for and design around.

[2] Microchip Technology Inc. *Non-Inverting Summing Amps* August 25, 2014. [Online]
This webpage explains the operation of amplifier circuits that sum signals together. This is useful in the triggering circuit to sum together the filtered and unfiltered signals.

[3] Circuits Today. *Voltage Comparator.* November 25, 2011. [Online]
This article was used to explain the user of voltage comparator circuits used in the triggering subsystem of our design.

[4] EEVBlog. *Oscilliscope Trigger Holdoff Tutorial.* March 29, 2011. #159 [Online]
This video provided understanding of holdoff triggering and how we could add it as a feature in our final scope.

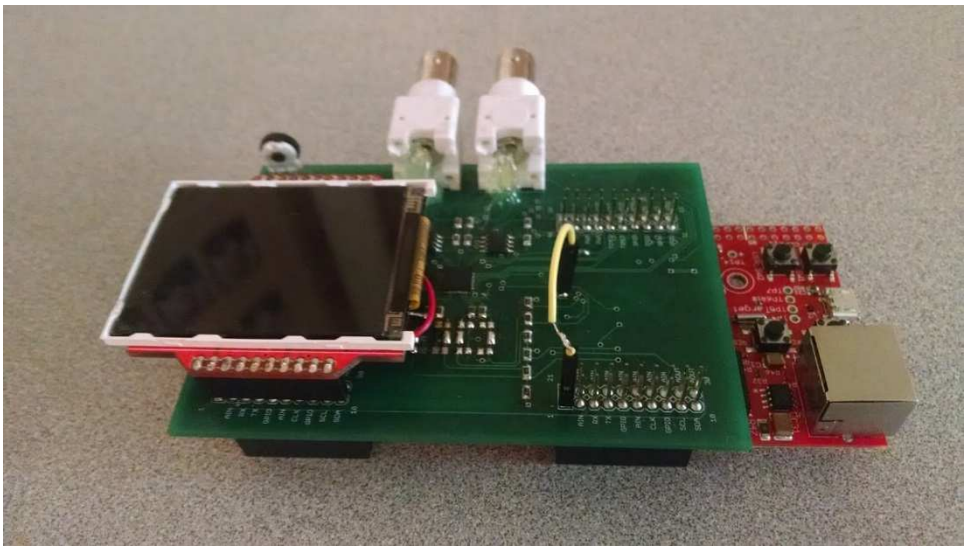
[5] Texas Instruments. *Oscilloscope Solution.* [Online]
This is a compilation of resources by Texas Instruments that describes solutions and systems needed to make an oscilloscope. This was useful to learn all the systems we needed for our final project.

Appendix

Note: All files and code on GitHub at steelerfan107/BoosterPackOscilloscope under Design Files

Critical IC BOM

Part	Value	Device	Package
-5V_SUPPLY	TL7660_D_8	TL7660_D_8	D8
AVERAGER	OPA890_DBV_6	OPA890_DBV_6	DBV6
COMPARATOR	OPA890_DBV_6	OPA890_DBV_6	DBV6
OFFSET	OPA890_DBV_6	OPA890_DBV_6	DBV6
U$3	LMH6882	LMH6882	WQFN-36-NJK
U1	ADS5237_PAG_64	ADS5237_PAG_64	PAG64
U2	THS4121_D_8	THS4121_D_8	D8
U3	THS4121_D_8	THS4121_D_8	D8
U4	OPA140AID	OPA140AID	SOIC127P600X175-8N 11MHz JFET Op Amp
U5	OPA140AID	OPA140AID	SOIC127P600X175-8N 11MHz JFET Op Amp



TI BOOSTERPACK OSCILLOSCOPE

User Manual 1.0

Eric Taylor and Raymond Montgomery | Rose-Hulman | Texas Instruments | Spring 2015

Purpose

The purpose of this user manual is to help you get started setting up and getting familiar with the project. This assumes that you have the board already constructed from the board files on GitHub using the BOM also on GitHub. This will also include how to improve upon this project and traverse the files on GitHub .

Table of Contents

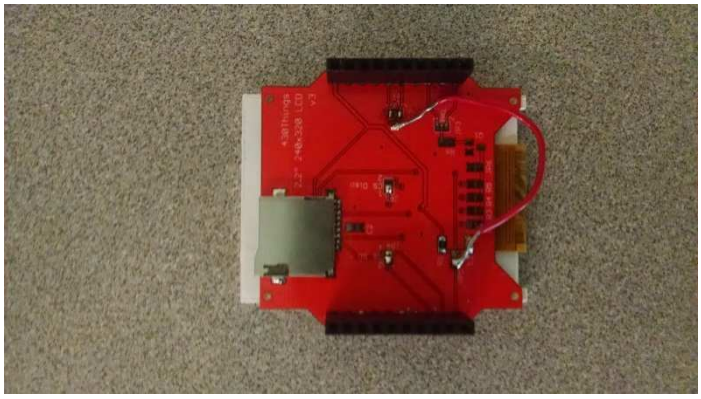
1.0 Hardware Set-Up

To set up the hardware you need three things:

- 2.2" Color LCD BoosterPack Kit by RobG
- A Fabricated and Populated Oscilloscope PCB
- Tiva C Connected LaunchPad

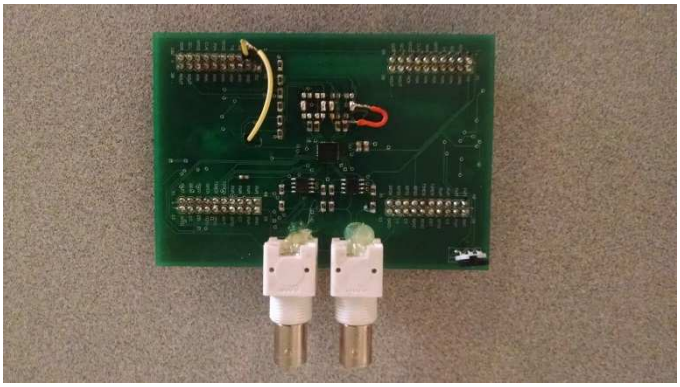
1.1 Color LCD Modifications

To make use of the EPI interface a small modification had to be made to the LCD BoosterPack hardware. You have to make sure neither jumper is soldered on the D/C pin, and then attach a wire to the middle pad and connect it to pad JP5. A picture showing this is below.



1.2 Oscilloscope Board Jumper and Passive Filter

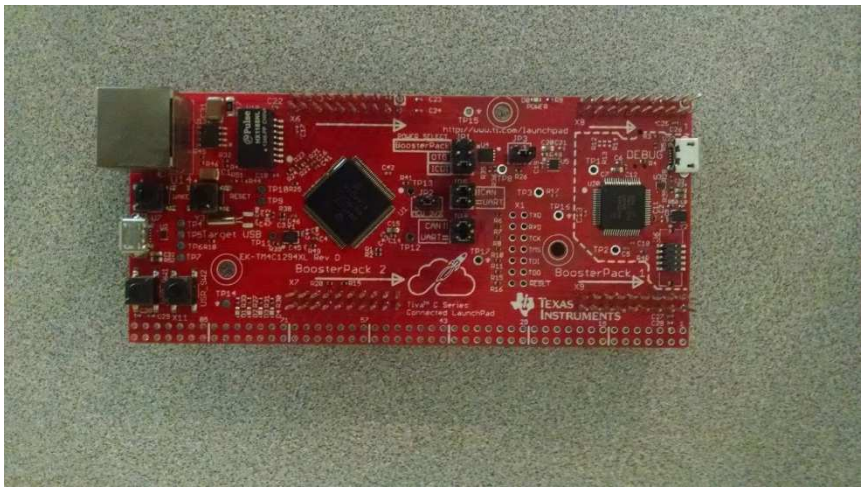
For the purposes of debugging a pin was left exposed on the ADC for this first revision. Next revisions should connect this pin to the standard LDD for the ADC. (Connect VDRV to VDD in the EAGLE files). This jumper is necessary to power the ADC. A typical female-to-female jumper was used. A picture showing it is seen below. (The Yellow Wire). Also leave passive filter disconnected and solder the cap connections so the signal can continue through. More explanation in TODO.



*Note - Orange Wire is there because of mistake while soldering

1.3 Tiva C Connected LaunchPad

No modifications should be needed. Make sure jumpers are connected like below.



Once all hardware is setup, you can attach all of the pieces as seen on the first page.

2.0 Software Setup

Four things are needed for the software setup.

- Energia w/ Tiva C Drivers
- Senior_Project_Main.ino (on GitHub under Software Code)
- Screen_ILI9340_Library.cpp/h

Minimal Setup is needed. First download Energia and put the screen driver and header files in the following path. (Note: The drivers are edited from the original online to account for the one pin being moved across the board.)

..\energia-0101E0013\hardware\lm4f\libraries\

Then open Senior_Project_Main.ino and select Tiva C tm4c129 as the board, Finally compile and run.

3.0 Usage of the Scope

1. To use the scope you can use any BNC connector or 1x probe. However 10x probes did not work when tested with the system. Connect probes up like a typical scope. For this revision you can only trigger of one channel. (BNC away from Screen).
2. Only signals from +/- 1.5 can be inputted at the end of the probe.
3. Power can be supplied through any micro USB source. Portable battery or laptop will work.
4. The trigger level can be adjusted using the potentiometer. Then the voltage per deviation and time per deviation can be adjusted using the two user buttons on the LaunchPad.

   Note: The GUI is still in its early stages and can be improved upon.

4.0 TODOs

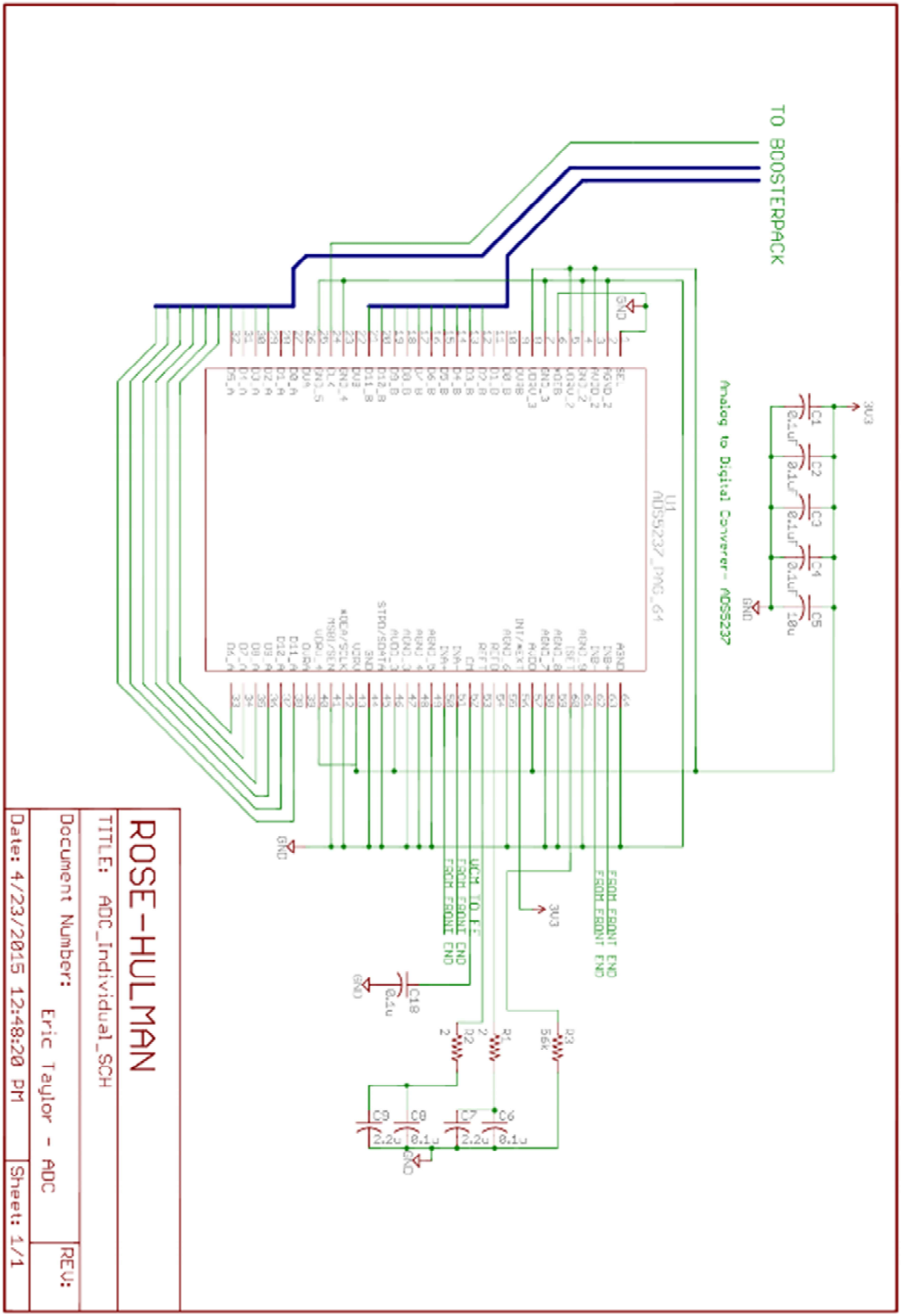
This section contains improvements that can be made on this project and steps to continue its development.

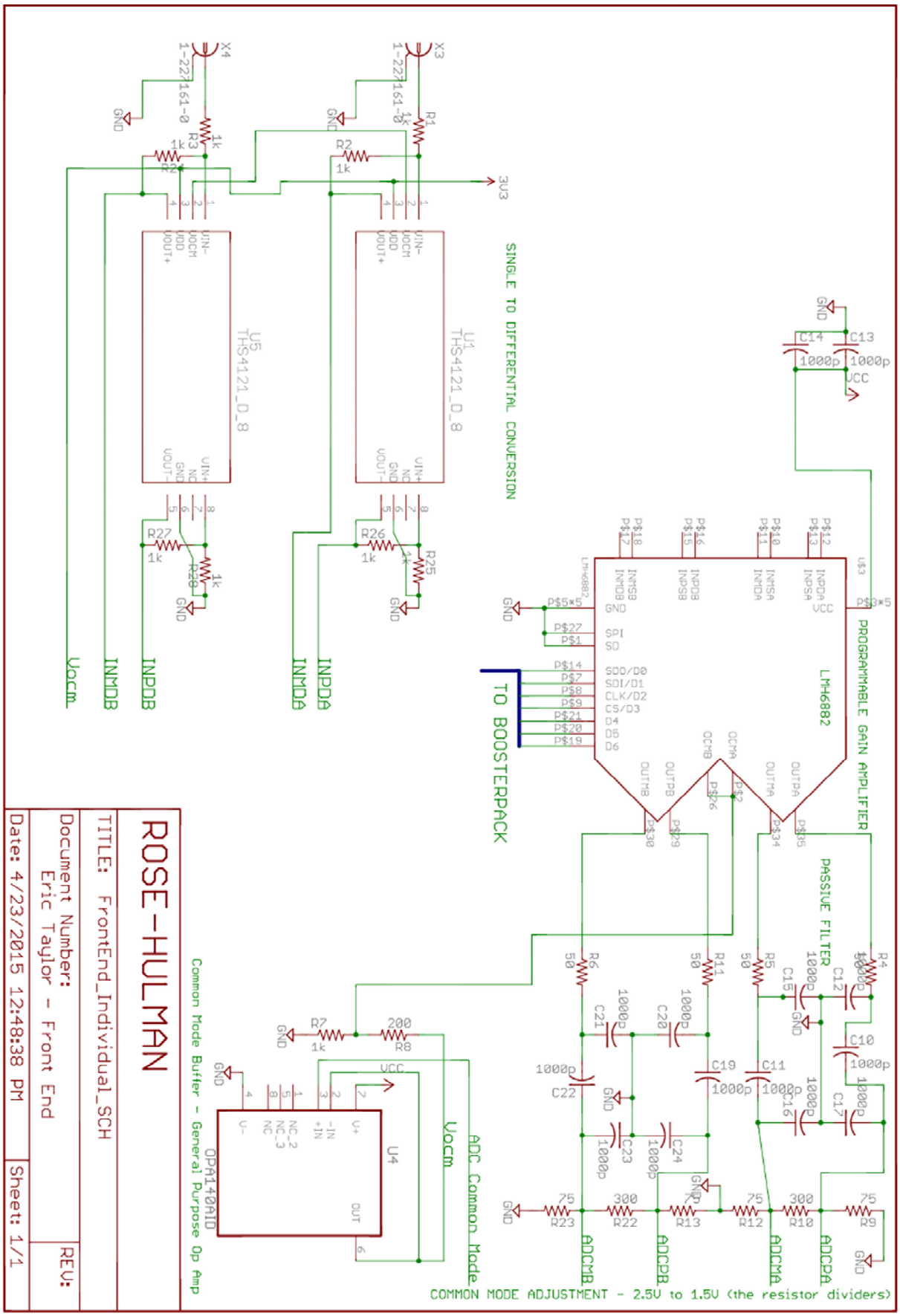
1. Switch BNC connections so that they are vertical. They are currently unstable.
2. Add buffer to front end so that 10x probes cannot load circuit.
3. Add switchable attenuator to front end.
4. Add uDMA to software to collect samples faster and be able to achieve full 7 MHz bandwidth . Currently a timer interrupt is used and the bandwidth is about 200kHz.
5. Put 10MHz flat pass band active filter instead of current passive filter.

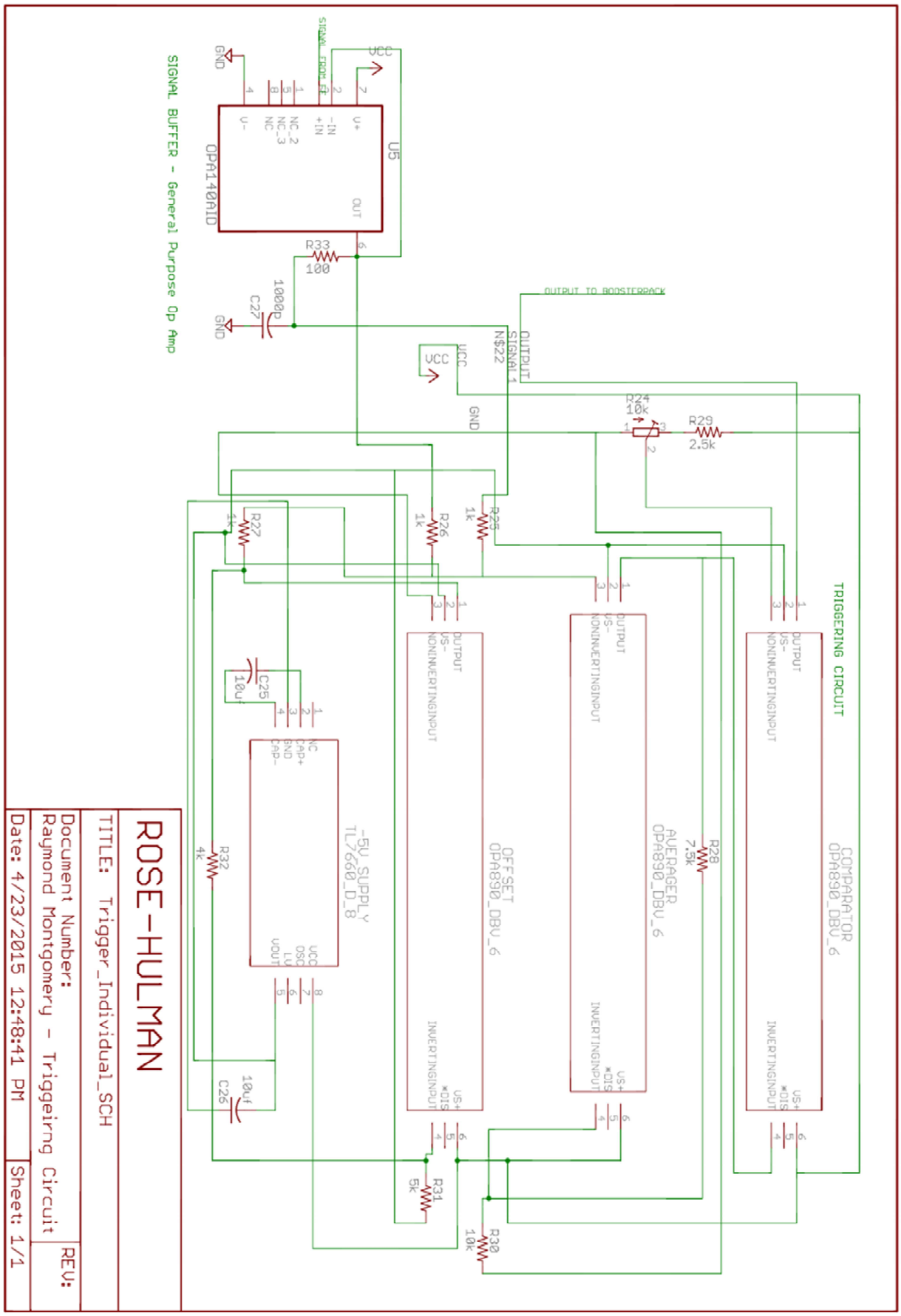
5.0 GitHub

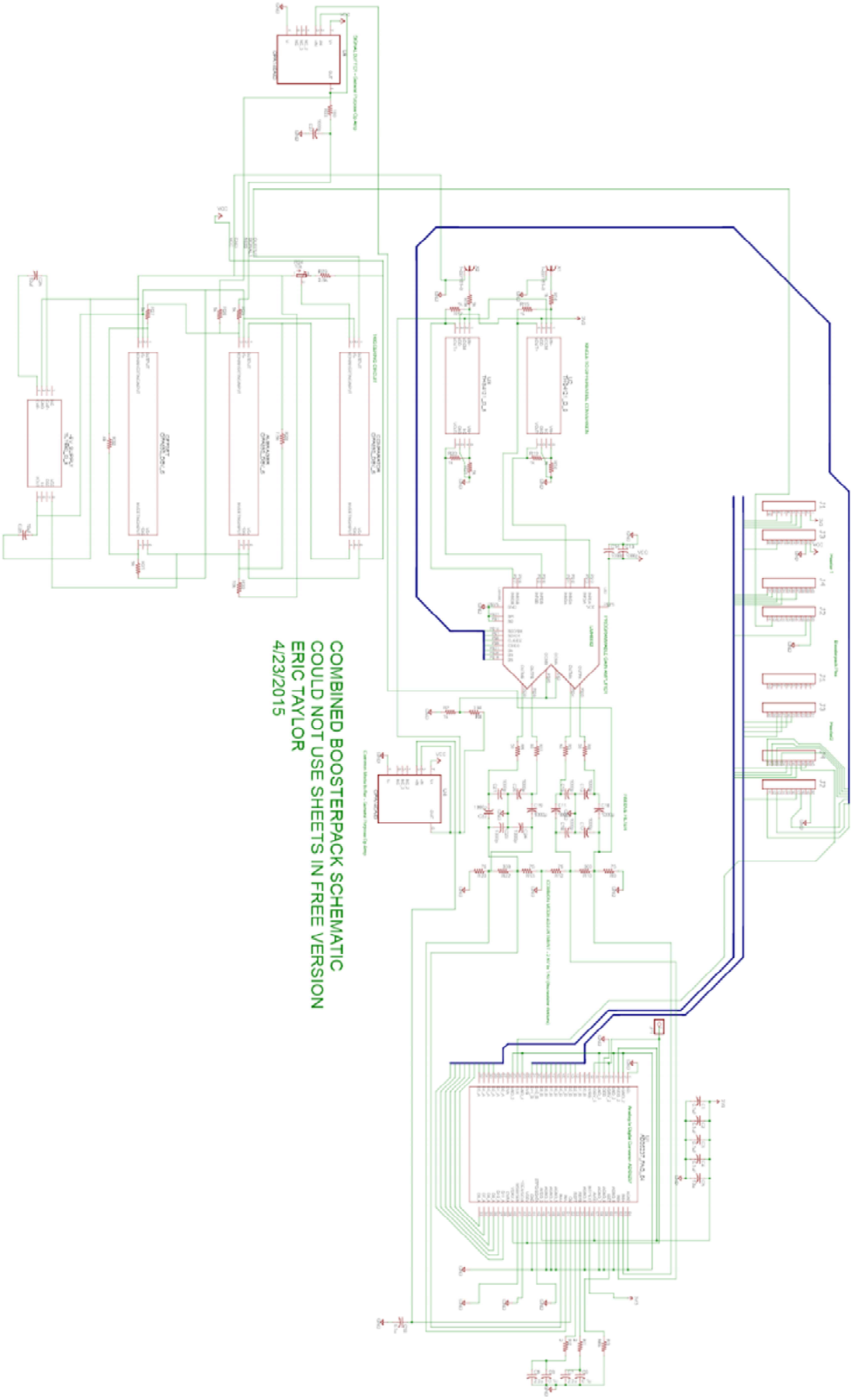
On GitHub you can find a directory listing on the front page giving explanation to the file directory and source files. GitHub at steelerfan107/BoosterPackOscilloscope

Schematics

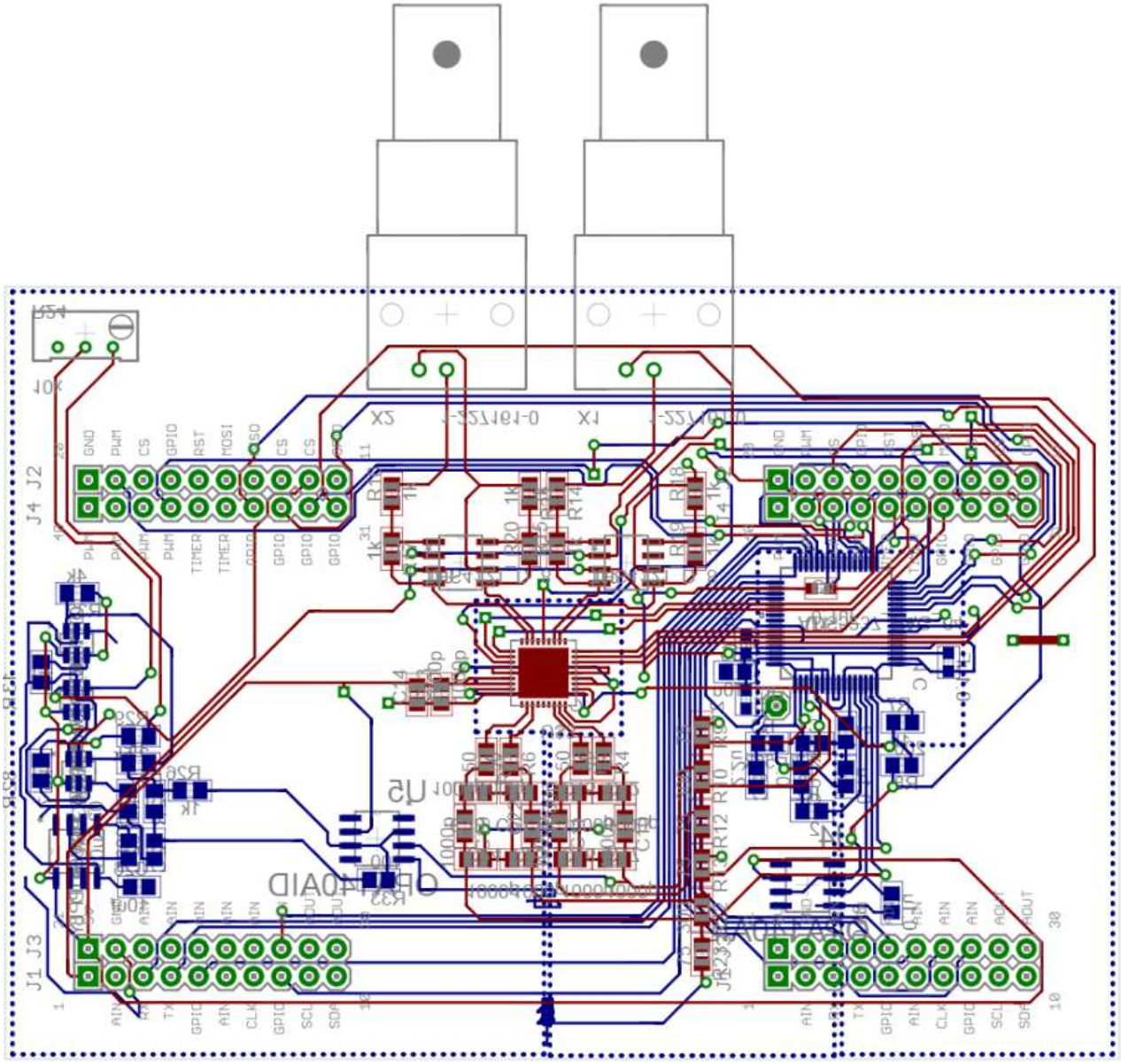

TO BOOSTERPACK
Analog to Digital Converter - ADS5237
U1
ADS5237_PAG_64
3V3
GND
C1
2.2uF
C2
0.1uF
C3
0.1uF
C4
0.1uF
C5
10u
C18
0.4u
R1
2
R2
2
R3
56k
C6
0.1
C7
2.2u
C8
0.1
C9
2.2u
FROM FRONT END
ROSE-HULMAN
TITLE: ADC_Individual_SCH
Document Number:
Eric Taylor - ADC
REV:
Date: 4/23/2015 12:48:20 PM
Sheet: 1/1

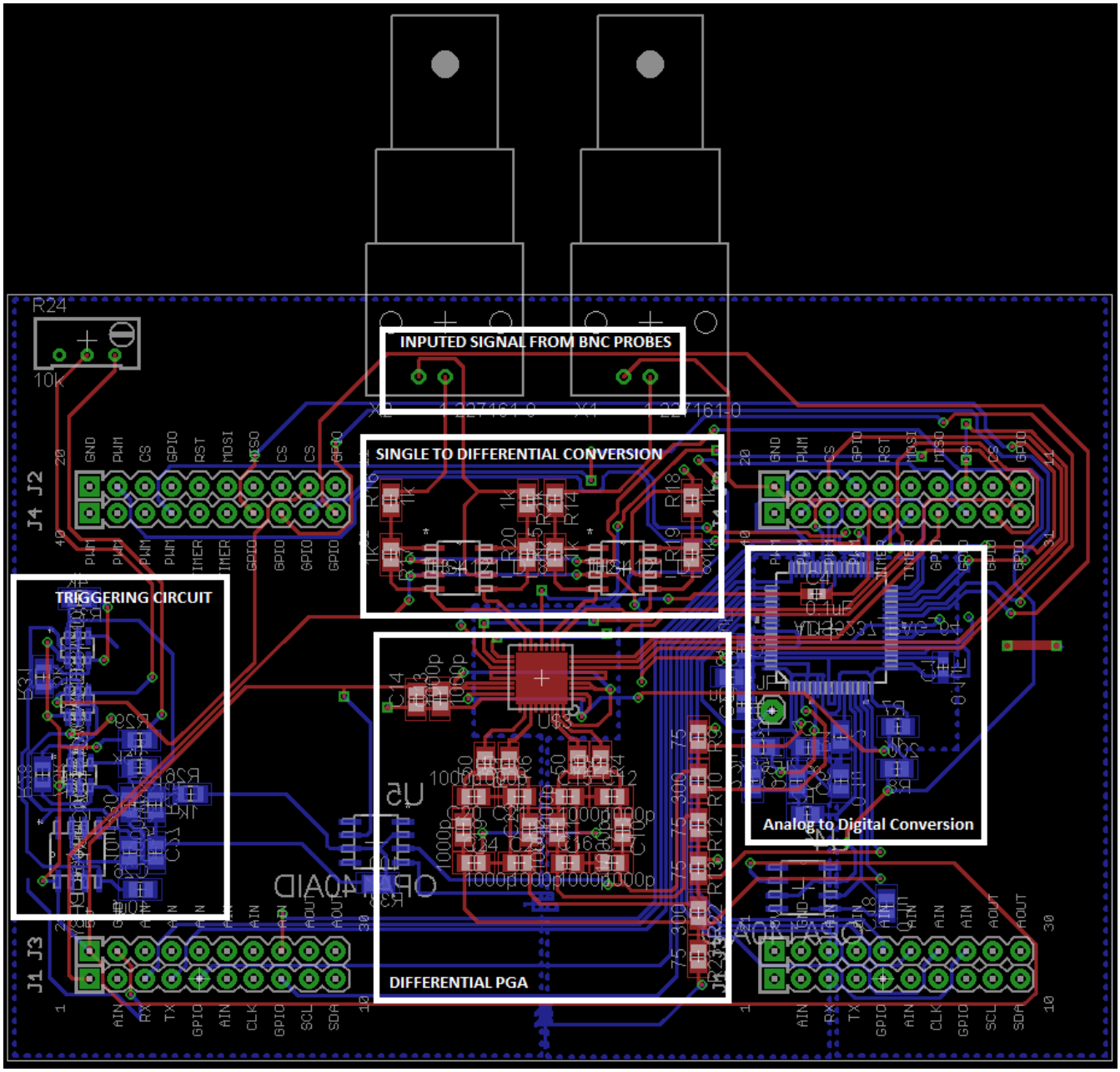

SINGLE TO DIFFERENTIAL CONVERSION
PROGRAMMABLE GAIN AMPLIFIER
PASSIVE FILTER
TO BOOSTERPACK
LMH6882
THS4121_D_8
OPA140AID
Common Mode Buffer - General Purpose Op Amp
ADC Common Mode
COMMON MODE ADJUSTMENT - 2.5V to 1.5V (the resistor dividers)
ROSE-HULMAN
TITLE: FrontEnd_Individual_SCH
Document Number:
Eric Taylor - Front End
REV:
Date: 4/23/2015 12:48:38 PM
Sheet: 1/1


SIGNAL BUFFER - General Purpose Op Amp
SIGNAL FROM FG
GND
VCC
OPA140AID
U5
R33
100
C27
1000p
OUTPUT TO BOOSTERPACK
OUTPUT SIGNAL N$22
R24
10k
R29
2.5k
R27
1k
R26
1k
R25
1k
TRIGGERING CIRCUIT
COMPARATOR OPA890_DBU_6
AVERAGER OPA890_DBU_6
OFFSET OPA890_DBU_6
R28
7.5k
C25
10u
-5V_SUPPLY TL7660_D_8
R32
4k
C26
10uf
R31
5k
R30
10k
ROSE-HULMAN
TITLE: Trigger_Individual_SCH
Document Number:
Raymond Montgomery - Triggering Circuit
REV:
Date: 4/23/2015 12:48:41 PM
Sheet: 1/1


COMBINED BOOSTERPACK SCHEMATIC
COULD NOT USE SHEETS IN FREE VERSION
ERIC TAYLOR
4/23/2015

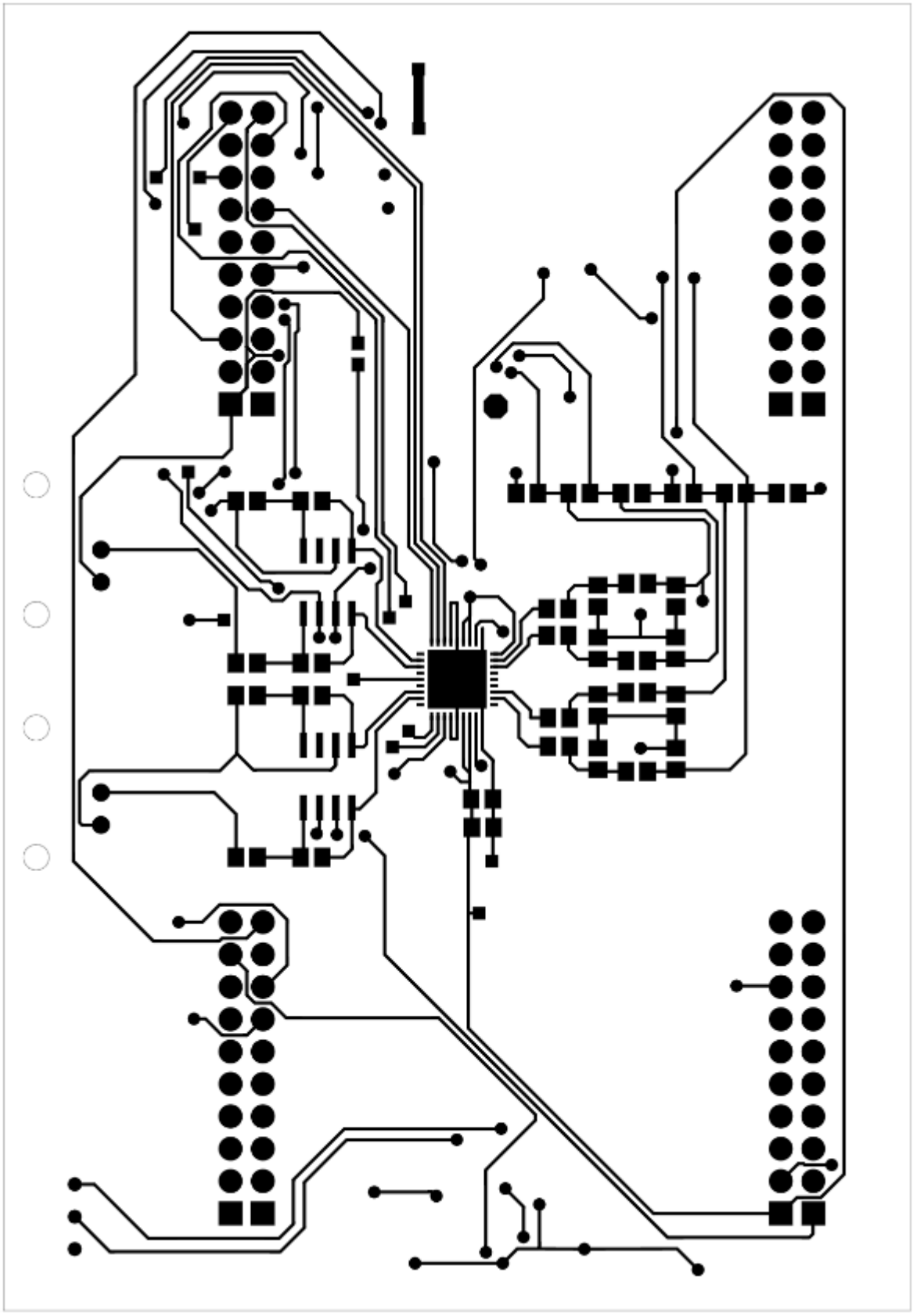
Board File



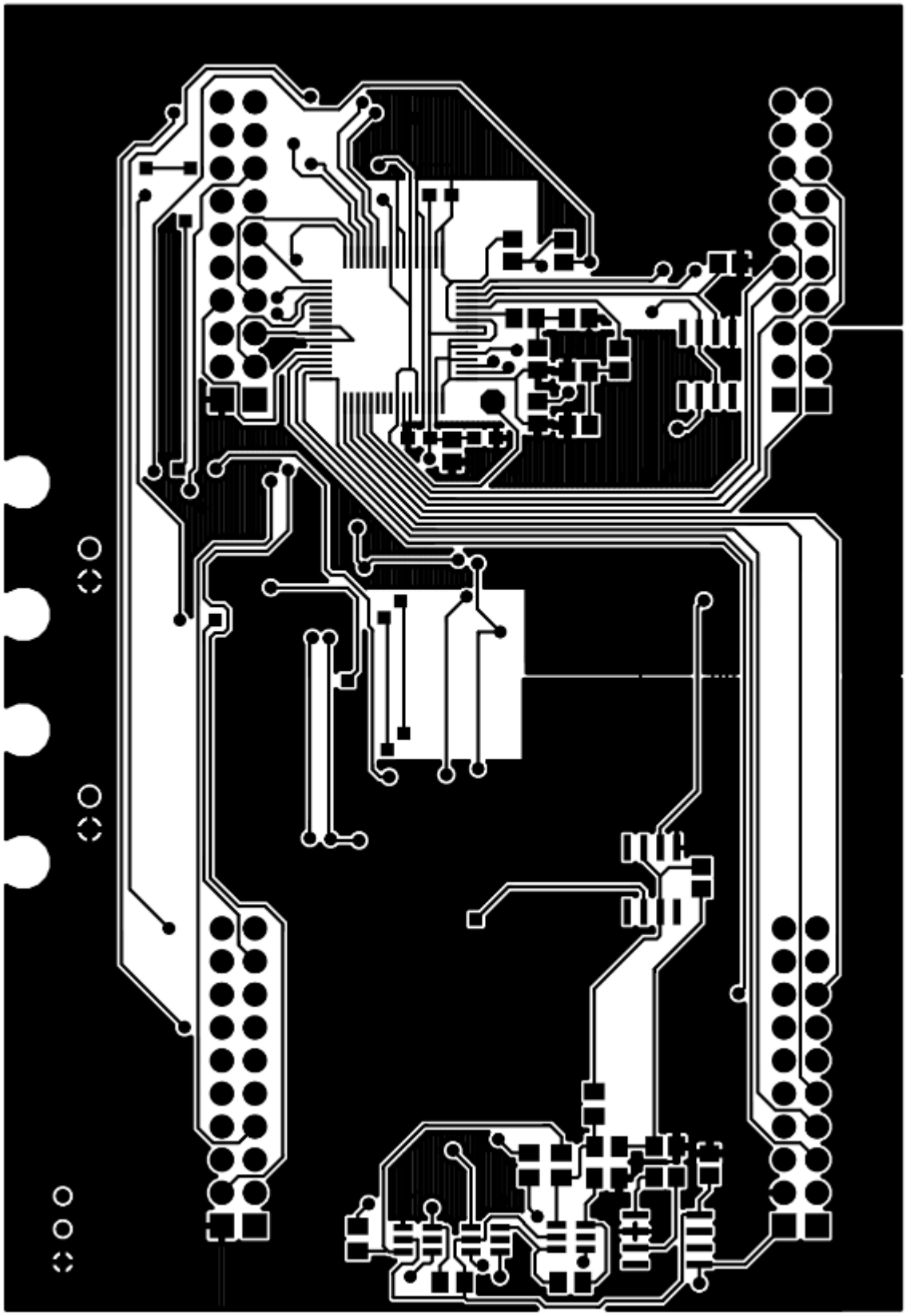

INPUTED SIGNAL FROM BNC PROBES
SINGLE TO DIFFERENTIAL CONVERSION
TRIGGERING CIRCUIT
Analog to Digital Conversion
DIFFERENTIAL PGA

Gerber Files

Top



Bottom



Code

```
//  Eric Taylor
//  Rose-Hulman - ECE 462 Senior Design
//  TI BoosterPack Oscilliscope Software
//
//  This program uses an EPI interface to read in 20 bit samples (First 10 bits -
Channel 1, Second 10 bits - Channel 2)
//  and uses a timer interupt to have a constant time between samples. These samples are
then shown on a LCD screen using
//  a SPI controller.
//
//  The following code was mide by me with exception of the SPI driver for the LCD
included in "Screen_ILI9340_Library.h"
//
//  TODO: Setup uDMA so sample can be read in faster allowing faster signals to be
measured.
////////////////////////////////////////
//  Header Files
////////////////////////////////////////

#include "Energia.h"
// Include application, user and local libraries
#include "SPI.h"
#include "driverlib/epi.h"
#include "driverlib/udma.h"
#include "driverlib/gpio.h"
// Screen selection
#define ILI9340 // HX8353E K35 HI32 W32 ILI9225B HY28A ST7735 PicasoSPE PicasoSGC


#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"


#if defined(ILI9340)
#include "Screen_ILI9340_Library.h"
Screen_ILI9340 myScreen;
#endif


////////////////////////////////////////////////
//  Global Variable Declarations
////////////////////////////////////////////////


uint8_t largeOrientation = 0;
uint32_t data_location = 0;
```

```
int flag = 0, test = 0;
unsigned long * data_addr = (unsigned long *)0x60000000;
volatile int temp = 1;
volatile int tempB = 1;


signed long buffer_a[320],old_buffer_a[320];
signed long buffer_b[320],old_buffer_b[320];
unsigned long sample;
unsigned long int_sample;


/*
void * PP_BUFFER_A = 0;
void * PP_BUFFER_B = 0;


////////////////////////////////////////////////
// DMA Setup (Not Used at the Moment)
////////////////////////////////////////////////


tDMAControlTable g_dmaCtrl[64] __attribute__ ((aligned(1024)));
void setupDMA(){
    PP_BUFFER_A = (void *)malloc(32*1024);
    PP_BUFFER_B = (void *)malloc(32*1024);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
    uDMAEnable();
    uDMAControlBaseSet(g_dmaCtrl);

    uDMAChannelAssign(UDMA_CH20_EPI0RX);

    uDMAChannelSelectSecondary(UDMA_DEF_TMR1A_SEC_EPI0RX);

    uDMAChannelAttributeEnable(UDMA_SEC_CHANNEL_EPI0RX, UDMA_ATTR_HIGH_PRIORITY);
    uDMAChannelAttributeDisable(UDMA_SEC_CHANNEL_EPI0RX, UDMA_ATTR_USEBURST |
UDMA_ATTR_REQMASK | UDMA_ATTR_ALTSELECT);

    uDMAChannelControlSet(UDMA_CH20_EPI0RX | UDMA_PRI_SELECT,UDMA_SIZE_32 |
UDMA_SRC_INC_NONE | UDMA_DST_INC_32 |UDMA_ARB_8);
    uDMAChannelTransferSet(UDMA_CH20_EPI0RX |
UDMA_PRI_SELECT,UDMA_MODE_PINGPONG,data_addr,PP_BUFFER_A,1023);

    uDMAChannelControlSet(UDMA_CH20_EPI0RX | UDMA_ALT_SELECT,UDMA_SIZE_32 |
UDMA_SRC_INC_NONE | UDMA_DST_INC_32 |UDMA_ARB_8);
    uDMAChannelTransferSet(UDMA_CH20_EPI0RX |
UDMA_ALT_SELECT,UDMA_MODE_PINGPONG,data_addr,PP_BUFFER_B,1024);

    //uDMAIntRegister(UDMA_INT_SW, DMA_HandleInterrupt);

    uDMAChannelEnable(UDMA_CH20_EPI0RX);
    //uDMAChannelRequest(UDMA_SEC_CHANNEL_EPI0TX);
}
*/


////////////////////////////////////////////////
// Setup Gain
```

```
// The pins in order control - D0, D1, D2, D3, D4, D5, D6 of the PGA. This sets up these
pins to be output enabled.
////////////////////////////////////////////////
void setupGain(){
    pinMode(51, OUTPUT);
    pinMode(53, OUTPUT);
    pinMode(58, OUTPUT);
    pinMode(74, OUTPUT);
    pinMode(73, OUTPUT);
    pinMode(72, OUTPUT);
    pinMode(71, OUTPUT);
}


////////////////////////////////////////////////
// Setup the External Periphrial Intrerface (EPI Interface Reads in Data From the ADC)
////////////////////////////////////////////////
void setupEPI()
{
  // Define what pins in each of the ports are used for the EPI
  #define EPI_PORTA_PINS (GPIO_PIN_6 | GPIO_PIN_7)
  #define EPI_PORTB_PINS (GPIO_PIN_3 | GPIO_PIN_2)
  #define EPI_PORTC_PINS (GPIO_PIN_7 | GPIO_PIN_6 | GPIO_PIN_5 | GPIO_PIN_4 )
  #define EPI_PORTG_PINS (GPIO_PIN_1 | GPIO_PIN_0 )
  #define EPI_PORTK_PINS (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
| GPIO_PIN_5)
  #define EPI_PORTL_PINS (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
| GPIO_PIN_5)
  #define EPI_PORTM_PINS (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3)
  #define EPI_PORTN_PINS (GPIO_PIN_2 | GPIO_PIN_4 | GPIO_PIN_5)
  #define EPI_PORTQ_PINS (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3)

  // Configuration options for the EPI
   unsigned long ulEPIConfig = EPI_GPMODE_CLKPIN
       | EPI_GPMODE_ASIZE_4   // address bus size of 4
       | EPI_GPMODE_DSIZE_24   //
       | EPI_GPMODE_READ2CYCLE  // The RD2CYC bit in the EPIGPCFG register must be set
at all times in General-Purpose
       | EPI_GPMODE_FRAMEPIN;  // frame signal


   // Enable EPI0
   SysCtlPeripheralEnable(SYSCTL_PERIPH_EPI0);

   // SET GPIO to connect to Hardware
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);

   //Configure Pins to drive 8mA
   GPIOPinConfigure(GPIO_PK0_EPI0S0);
```

```
 GPIOPinConfigure(GPIO_PK1_EPI0S1);
 GPIOPinConfigure(GPIO_PK2_EPI0S2);
 GPIOPinConfigure(GPIO_PK3_EPI0S3);
 GPIOPinConfigure(GPIO_PC7_EPI0S4);
 GPIOPinConfigure(GPIO_PC6_EPI0S5);
 GPIOPinConfigure(GPIO_PC5_EPI0S6);
 GPIOPinConfigure(GPIO_PC4_EPI0S7);
 GPIOPinConfigure(GPIO_PA6_EPI0S8);
 GPIOPinConfigure(GPIO_PA7_EPI0S9);
 GPIOPinConfigure(GPIO_PG1_EPI0S10);
 GPIOPinConfigure(GPIO_PG0_EPI0S11);
 GPIOPinConfigure(GPIO_PM3_EPI0S12);
 GPIOPinConfigure(GPIO_PM2_EPI0S13);
 GPIOPinConfigure(GPIO_PM1_EPI0S14);
 GPIOPinConfigure(GPIO_PM0_EPI0S15);
 GPIOPinConfigure(GPIO_PL0_EPI0S16);
 GPIOPinConfigure(GPIO_PL1_EPI0S17);
 GPIOPinConfigure(GPIO_PL2_EPI0S18);
 GPIOPinConfigure(GPIO_PL3_EPI0S19);
 GPIOPinConfigure(GPIO_PQ0_EPI0S20);
 GPIOPinConfigure(GPIO_PQ1_EPI0S21);
 GPIOPinConfigure(GPIO_PQ2_EPI0S22);
 GPIOPinConfigure(GPIO_PQ3_EPI0S23);
 // Commented out lines represent GPIO used elsewhere
 //GPIOPinConfigure(GPIO_PK7_EPI0S24);
 //GPIOPinConfigure(GPIO_PK6_EPI0S25);
 GPIOPinConfigure(GPIO_PL4_EPI0S26);
 GPIOPinConfigure(GPIO_PB2_EPI0S27);
 GPIOPinConfigure(GPIO_PB3_EPI0S28);
 GPIOPinConfigure(GPIO_PN2_EPI0S29);
 //GPIOPinConfigure(GPIO_PN3_EPI0S30);
 GPIOPinConfigure(GPIO_PK5_EPI0S31);
 GPIOPinConfigure(GPIO_PK4_EPI0S32);
 GPIOPinConfigure(GPIO_PL5_EPI0S33);
 GPIOPinConfigure(GPIO_PN4_EPI0S34);
 GPIOPinConfigure(GPIO_PN5_EPI0S35);

 // Set GPIO to EPI
 GPIOPinTypeEPI(GPIO_PORTA_BASE, EPI_PORTA_PINS);
 GPIOPinTypeEPI(GPIO_PORTB_BASE, EPI_PORTB_PINS);
 GPIOPinTypeEPI(GPIO_PORTC_BASE, EPI_PORTC_PINS);
 GPIOPinTypeEPI(GPIO_PORTG_BASE, EPI_PORTG_PINS);
 GPIOPinTypeEPI(GPIO_PORTK_BASE, EPI_PORTK_PINS);
 GPIOPinTypeEPI(GPIO_PORTL_BASE, EPI_PORTL_PINS);
 GPIOPinTypeEPI(GPIO_PORTM_BASE, EPI_PORTM_PINS);
 GPIOPinTypeEPI(GPIO_PORTN_BASE, EPI_PORTN_PINS);
 GPIOPinTypeEPI(GPIO_PORTQ_BASE, EPI_PORTQ_PINS);

//
// Set the EPI divider.
//
EPIDividerSet(EPI0_BASE, 1);

//
// Select GPIO
//
EPIModeSet(EPI0_BASE, EPI_MODE_GENERAL);

//
```

```
  // Activate External CLK on EPI0s31, Set data to 24 bits EPI0s0 to EPI0S23
  //
  EPIConfigGPModeSet(EPI0_BASE,ulEPIConfig,0,0);


  //
  // Set the address map.
  //
  EPIAddressMapSet(EPI0_BASE, EPI_ADDR_RAM_SIZE_256MB | EPI_ADDR_RAM_BASE_6);


  //
  // Start the EPI Interface
  //
  EPINonBlockingReadConfigure(EPI0_BASE, 0, EPI_NBCONFIG_SIZE_32, 0);
  EPINonBlockingReadStart(EPI0_BASE,0,1);
}







////////////////////////////////////////////////
//  Draw the GUI on the Screen
////////////////////////////////////////////


void GUI()
{
    myScreen.setOrientation(1);
    myScreen.setFontSize(0);
    myScreen.gText(4, 4, "V/D " + String(1) + ", Time/D: " + String(1)+ ",R/F: " +
String(1) , colour_white);


    for(int x = 1; x < 320; x++){
        if(x%20 == 0){
            if(x == 20*8){
                //Long Vertical Axis
                for(int i = -120; i < 120; i++){
                myScreen.point(x, 120 + i, colour_white);
                //myScreen.point(x, 240 + i, colour_white);

                if(i%20 == 0 && i != -120){
                // Lines on Vertical Axis
                  for(int y = -6; y < 6; y++){
                    myScreen.point(x+y, 120 + i, colour_white);
                    //myScreen.point(x+y, 240 + i, colour_white);
                  }
                }

              }
            }else{
               // Ticks on Horizontal Axis
              for(int i = -5; i < 5; i++){
                myScreen.point(x, 120 + i, colour_white);
                //myScreen.point(x, 240 + i, colour_white);
              }
```

```
            }
        }
        myScreen.point(x, 120, colour_white);
        //myScreen.point(x, 240, colour_white);

    }
}


////////////////////////////////////////
//  GUI UPdate
//   - Updates the GUI based on the input
////////////////////////////////////////
void GUI_update(uint8_t v_div,uint8_t t_div, uint8_t risefall )
{
    myScreen.setFontSize(0);
    myScreen.gText(4, 4, "V/D " + String(v_div) + ", Time/D: " + String(t_div)+ ",R/F: "
+ String(risefall) , colour_white);
    delay(100);
}


////////////////////////////////////////////////
// sSets Gain
// The pins in order control - D0, D1, D2, D3, D4, D5, D6 of the PGA. This sets up these
pins to the inputed state.
////////////////////////////////////////////////
void setGainData(int d6, int d5, int d4, int d3, int d2, int d1, int d0){
      if(d0 == 1){
          digitalWrite(51, HIGH);
      }else{
          digitalWrite(51, LOW);
      }

      if(d1 == 1){
          digitalWrite(53, HIGH);
      }else{
          digitalWrite(53, LOW);
      }

      if(d2 == 1){
          digitalWrite(58, HIGH);
      }else{
          digitalWrite(58, LOW);
      }

      if(d3 == 1){
          digitalWrite(74, HIGH);
      }else{
          digitalWrite(74, LOW);
      }

      if(d4 == 1){
          digitalWrite(73, HIGH);
      }else{
          digitalWrite(73, LOW);
      }

      if(d5 == 1){
```

```
          digitalWrite(72, HIGH);
      }else{
          digitalWrite(72, LOW);
      }

      if(d6 == 1){
          digitalWrite(71, HIGH);
      }else{
          digitalWrite(71, LOW);
      }
}



uint16_t size = myScreen.screenSizeX();
int i = 0,x = 0;
uint32_t data = 0;
uint32_t y = 0,z = 0,fill = 0;


unsigned long int timer_value;
volatile int buffer_count = 0;
int triggered = 0;
int temp_test = 0;
double gain = 4;
double old_gain = 4;
int    wait = 100;
signed int ya,yb,ya_o,yb_o;


const int TimePin = PUSH1;
const int VoltagePin = PUSH2;
const int TrigPin = 12;


int volState = 1;
int timeState = 1;




////////////////////////////////////////////
// Initialization Code
////////////////////////////////////////////


void setup()
{
    // Setup PGA Control
    setupGain();
    setGainData(1,1,1,1,1,1,1);

    pinMode(VoltagePin, INPUT_PULLUP);
    pinMode(TimePin, INPUT_PULLUP);

    attachInterrupt(VoltagePin, voldiv, RISING);
    attachInterrupt(TimePin, timediv, RISING);
```

```
    pinMode(12, INPUT);

    setupEPI();
    // Start LCD Screen
    myScreen.begin();

    myScreen.setFontSize(myScreen.fontMax());
    //while(1){
    // Draw GUI
    GUI();
    //}

    // Fill the old and new buffers
    for(x = 0; x <= 319; x+=1){
        buffer_a[x] = 0;
        old_buffer_a[x] = 0;
        buffer_b[x] = 0;
        old_buffer_b[x] = 0;
    }


    i = 0;
    buffer_count = 0;
    TimerSetup();
    attachInterrupt(TrigPin, trigger, RISING);
}




// Loop Code
void loop()
{
    //myScreen.gText(40, 40, String(int_sample&0x0000001F));

    //
    // Software Trigger
    //   - Waits until a value is read on EPI, once it finds the correct value it starts
the timer interrupt to sample the EPI

    //if(data_addr[0]  != 0 && triggered == 0){
    //  triggered = 1;
    //  TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    //}

    //
    // If the Scope was triggered and the buffer of samples was filled
    //

    if(triggered == 1){
      interrupts();



      if( buffer_count == 320 ){

        // Disable the Time Interrupt
```

```
        TimerIntDisable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
        noInterrupts();

        // DRAW SIGNAL
        for(x = 0; x <= 320; x+=1){
          ya = ((buffer_a[x]-512+115)/gain+120);
          yb = ((buffer_b[x]-512+115)/gain+120);

          if(buffer_a[x] != 10 && x % 20 != 0 && ya % 20 != 0){
            myScreen.point(x, ya, colour_red);
          }
          if(buffer_b[x] != 10 && x % 20 != 0 && yb % 20 != 0){
            myScreen.point(x, yb, colour_blue);
          }
        }
         // ERASE OLD SIGNAL
        for( x = 0; x < 320; x +=1){
          ya_o = ((old_buffer_a[x]-512+115)/old_gain+120);
          yb_o = ((old_buffer_b[x]-512+115)/old_gain+120);

          if(x % 20 != 0 && ya_o % 20 != 0 && old_buffer_a[x] != 10 && buffer_a[x] !=
old_buffer_a[x]){
            myScreen.point(x, ya_o, colour_black);
          }

          if(x % 20 != 0 && yb_o % 20 != 0 && old_buffer_b[x] != 10 && buffer_b[x] !=
old_buffer_b[x]){
            myScreen.point(x, yb_o, colour_black);
          }

          old_buffer_a[x] = buffer_a[x];
          old_buffer_b[x] = buffer_b[x];

        }

        //GUI();
        myScreen.gText(20, 20, "A: " + String(buffer_a[x]-100));
        myScreen.gText(70, 20, "B: " + String(buffer_b[x]-100));
        delay(10);
        buffer_count = 0;
        triggered = 0;
        old_gain = gain;
        // Start Intrrupts again
        interrupts();
        TimerLoadSet(TIMER1_BASE,TIMER_A, wait);
      }

    }
}


// When the voltage per deviation button is pressed this function changes the gain to
make displayed signal larger or smaller
void voldiv()
{
    volState++;
    if(volState == 3){
      volState = 0;
    }
```

```
    if(volState == 0){
      gain = 8;
    }
    if(volState == 1){
      gain = 4;
    }
    if(volState == 2){
      gain = 2;
    }

    GUI_update(volState,timeState, 1);

}


// When the time per deviation button is pressed this function changes the time beween
samples to change how much of a signal is measured
void timediv()
{
    timeState++;
    if(timeState == 3){
      timeState = 0;
    }

    if(timeState == 0){
      wait = 150;
    }
    if(timeState == 1){
      wait = 300;
    }
    if(timeState == 2){
      wait = 600;
    }
    GUI_update(volState,timeState, 1);
}


// When trigger is seen this function is called and starts collecting samples
void trigger()
{
      triggered = 1;
      interrupts();
      TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
}


//
// Timer Interrupt Initalization
//
void TimerSetup(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
  TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
  TimerIntRegister(TIMER1_BASE, TIMER_A, Timer1IntHandler);
  TimerEnable(TIMER1_BASE, TIMER_A);
  TimerLoadSet(TIMER1_BASE,TIMER_A,150);
  IntEnable(INT_TIMER1A);
}
```

```
//
// Timer Interrupt Handler
//
void Timer1IntHandler(void) {

       // Clear the timer interrupt
       TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);


        // If the buffer is not filled and the scope is triggered
        if( buffer_count < 320 && triggered == 1){
          // Get sample from EPI
          sample = data_addr[0];
          // Store First Sample
          buffer_a[buffer_count] = sample&0x00000003FF;
          // Store Second Sample
          buffer_b[buffer_count] = (sample>>10)&0x000003FF;
          // Count Number of Samples
          buffer_count++;
        }
}
```