An embeddable data acquisition system – interim report.

Mechanical Engineering Technical Report 2014/04
Peter Jacobs and Igor Dimitrijevic*
School of Mechanical and Mining Engineering
The University of Queensland.

February 2, 2015

Abstract

This report describes the hardware and firmware prototypes for a low-power data acquisition system for small aerodynamic models. The system is not fixed but is actually a set of recording and signal-conditioning boards that can be assembled into a custom system that can then be embedded into a specific model.

*CB Aerospace, P.O. Box 7156, Hemmant QLD, Australia. http://cbaerospace.com.au

Contents

1 Overview

eDAQS is a kit of components, both hardware and firmware, for building low-power embedded data acquisition systems. The central component of each system is one or more recording boards (See Fig.1) that can sample signals from both analog and digital sensors. The recording board is supervised via a serial communications port, using a simple command-response interface.

The intended applications include a completely self-contained system, powered by a battery and communicating via a Bluetooth serial connection, that can be embedded in a free-flying model. Also, using the I2C communication bus, several recording boards may be used simultaneously to record the transient data from many sensors. Being able to embed the recording boards within the aerodynamic model will allow shorter and simpler wiring to the individual sensors, and should result in cleaner signals.

The kit of components includes:

- A recording board, based on a microcontroller, together with an external static RAM chip for the main data store. The microcontroller has a limited on-chip RAM but good peripheral modules, including a 12-bit analog-to-digital converter, and relatively low current consumption. The "standard format" of the recording board is a 50 mm × 50 mm square board that has mounting holes in the corners. This layout allows convenient stacking of the boards, as shown in the middle of Fig. 1. Around the periphery of the board, there is accommodation for the input of 6 analog signals, in the range of 0-3.3 V or 0-5 V (selectable). There is also a digital port for two SPI devices and an I2C port for an arbitrary number of devices. Recording of digital signals intermixed with analog signals is allowed; it's a matter of customizing the firmware. Two of these boards have been manufactured for trials.

- A "slender-format" recording board that is electrically identical to the standard recording board but with layout has been compressed to make the minimum dimension 20 mm. Two of these boards have been manufactured for trials.

- A standard-format recording board with programmable-gain amplifiers. This board has the same capability as the standard recording board, with the enhancement that each analog channel has a programmable-gain amplifier to condition the incoming signal. None have been manufactured presently because planned experiments, so far, have intended to use either digital output sensors or analog-front-end boards.

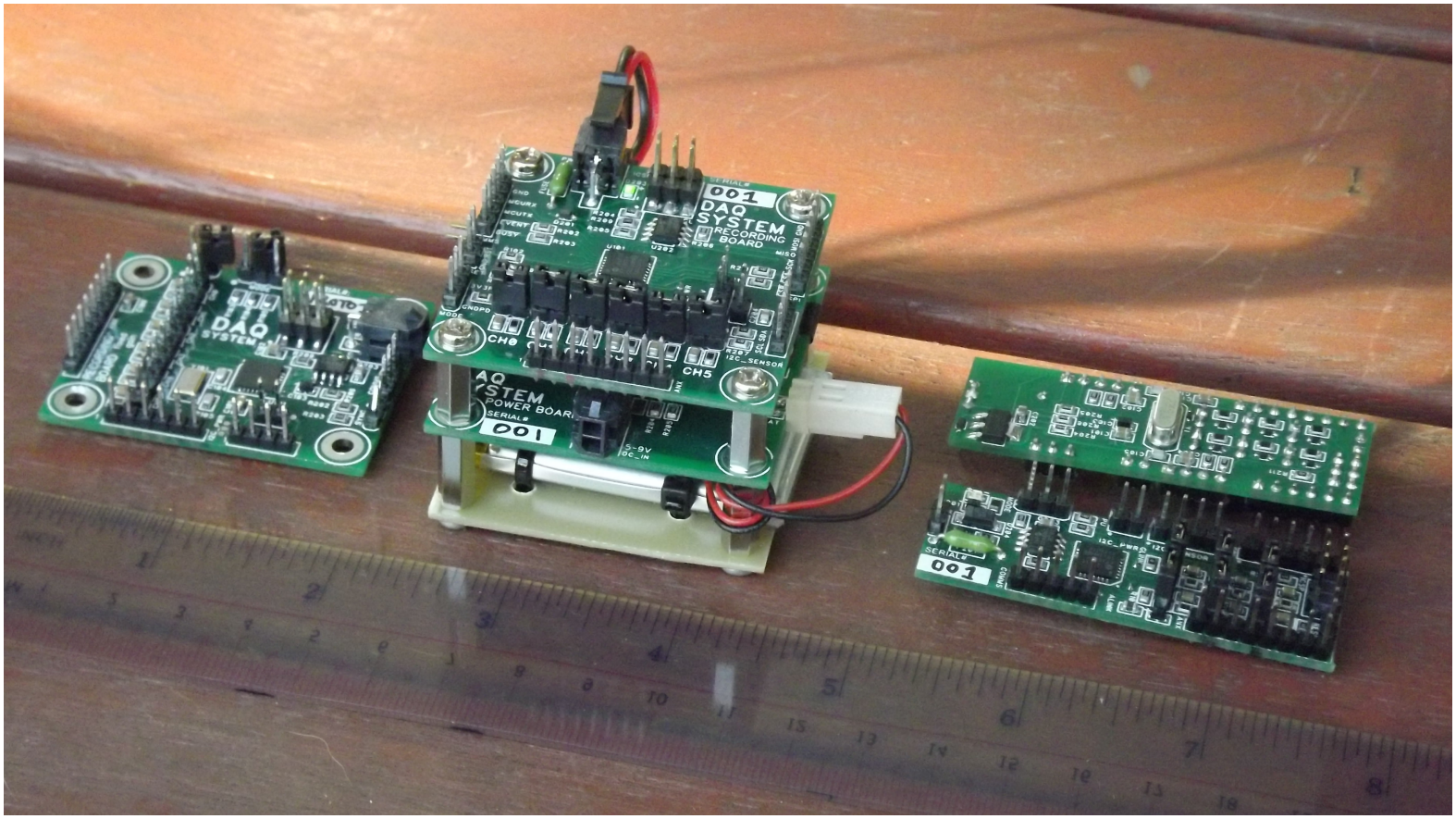


Figure 1: The left-most board is the prototype of the standard recording board from CB-Aerospace. The stack in the middle has the standard recording board on top, a power-conditioning board in the middle and a lithium battery on the lowest board. The boards on the right are the narrow-profile version of the standard recording board. One is upside-down to show the components on both sides.

- A power board manages a rechargable battery and can provide nominal-5 V power to several other boards. These might include several recording boards and analog-front-end boards. A single recording board can run for several hours from a fully-charged battery. Charging is performed by plugging in a suitable DC voltage (of 4V-9V).
- An analog-front-end (AFE) board for conditioning the signals from three thin-film gauges (TFGs) and three bridge-type sensors. The thin-film amplifiers are to Martin Oldfield's design and the amplifiers for the bridge-type sensors are INA122 instrumentation amplifiers. Several of these boards will be ready in Jan 2015.
- A small-format thin-film-gauge board. It is anticipated that the TFG amplifiers will be more generally useful so an independent small-format board has been designed. Several of these boards, with surface-mount components, will be ready in Jan 2015.
- A digital-pressure sensor adapter board for connecting several Honeywell ASDX pressure sensors via their I2C port.
- A small board for mounting an Invensense MPU-6050 inertial sensor.

So far, the recording board has progressed through two rounds of prototyping and four production boards have been produced, two in standard format and two in slender format. Version 1 of the firmware, as shown in Section 3, has been built for recording up to 6 analog signals. When run on the prototype board, the following sampling speeds were achieved.

Signals	Memory	Number of Samples	Sample Frequency	Recording Time
6	external	14563	15 kHz	0.976 s
4	external	21845	20 kHz	1.09 s
2	internal	2048	80 kHz	26.6 ms

Although the hardware design has settled, the firmware will be revised. The command-response format, currently designed for manual interaction, will be made more friendly for use with the I2C communications port.

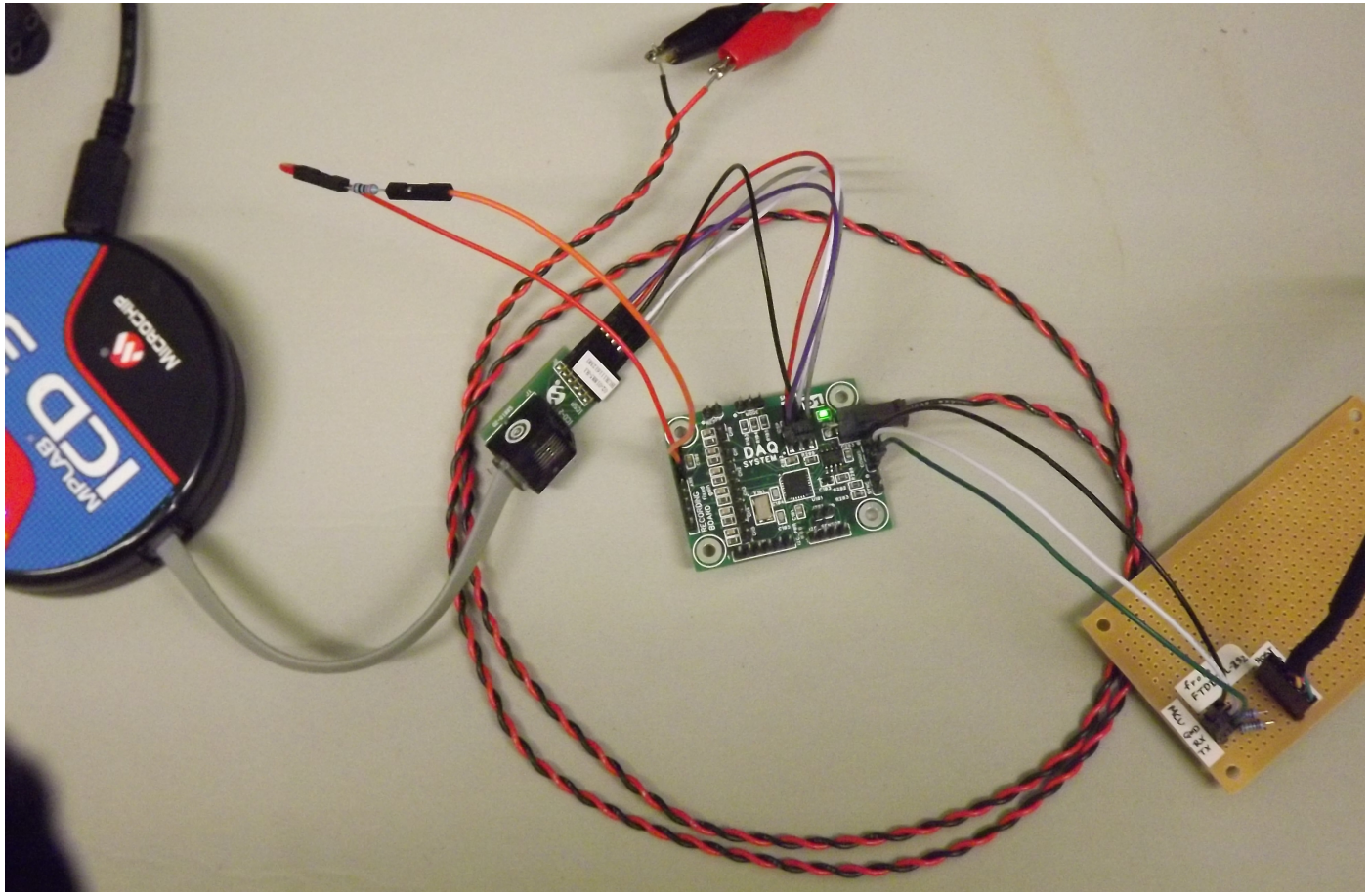


Figure 2: The prototype of the standard recording board powered with 5V and connected to the PC via the UART serial port. There is a small adapter board to transition from the 6-pin FTDI-232 cable to the three pins on the recording board. The pair of 1K resistors on this board limit the current passing through the microcontroller's pin-protection diodes when its power has been removed but the FTDI-232 cable is powered by the USB port of the monitoring PC. The ICD3 hardware programmer is also connected via it's 6-pin header.

2 Hardware

After an initial prototype built on strip board (described in Appendix A), CBAerospace has designed a collection of boards from which a customized data acquisition system can be assembled. The details of these boards are described, in schematic form, in the following pages.

The standard format of the boards is a 50 mm × 50 mm square board that has mounting holes in the corners. This layout allows convenient stacking of the boards, as shown in the middle of Fig. 1, where a standard recording board sits above a power-management board and a third board to which a Li-Poly battery is attached.

2.1 Recording board, standard format

One or more recording boards would form the core of any particular system. Each recording board is based on a Microchip dcPIC33EP256GP502 microcontroller[1], together with an external static 128k-byte RAM chip [2] (connected via the SPI port) as the main data store. The dsPIC is a 16-bit class microcontroller with limited on-chip RAM (of approximately 30k bytes) but nice 12-bit analog-to-digital converter capable of 500kSps conversions. There is also a good selection of digital communication interfaces available for connecting digital-output sensors.

Power to the board is intended to be a nominal 5 volts and, indeed, the power management board (Section 2.4) actually provides 5.3 V because there is a small fuse and a reverse-polarity protection diode in line before the VCC rail. If the input power wires are applied reversed, this diode will block any significant current and thus protect the board's other components. There is also a 6.5 V Zenner diode to protect the board from excessive voltage. Be careful to use a supply voltage less than or equal to 6 V. If large currents are drawn by some device attached to the board or too high a voltage is applied (or the power pins shorted), the on-board fuse should blow at 750 mA. This fuse is a through-the-hole component, and if blown, will involve some soldering to replace. In normal operation the Schottky protection diode will drop the voltage by about 0.3 V to the nominal 5 V value. The on-board green LED should light steadily if the power supply is good.

The TC1262 linear regulator [3] provides a 3.3 V reference rail that powers the microcontroller and is also used as the analog

conversion reference. This reference voltage should typically be within 0.5% of the nominal value and the resistors in the input chain are 1% tolerance, however, it would be a good idea to calibrate the full analog signal chain when the sensors are attached.

Around the periphery of the board, there is accommodation for the input of 6 analog signals, in the range of 0-3.3 V or 0-5 V as selected via jumper pins on the resistor voltage dividers. There is also a digital port for two SPI devices and an I2C port for an arbitrary number of devices. Figure 3 provides a key to the pin connections. Recording of digital signals may be intermixed with analog signals but the firmware described in Section 3 will need to be extended for each specific set of digital sensors.

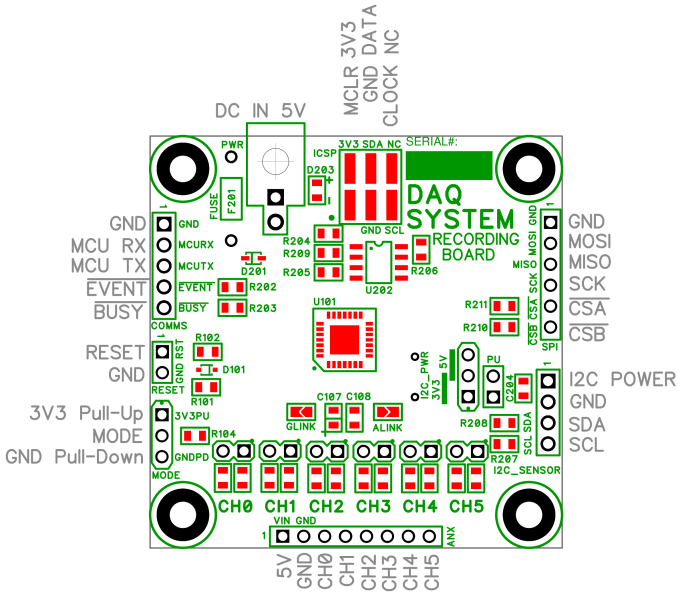


Figure 3: Pin layouts for the standard recording board.

The dsPIC is essentially a 3.3 V microcontroller, however, a number of its pins are 5 V compatible. The I2C port, for example, may be used with 3.3 V or 5 V devices and a selectable header is made available. The microcontroller has relatively low current consumption and, with an oscillator frequency of 29.5 MHz, approximately 20 mA is consumed.

PAGES

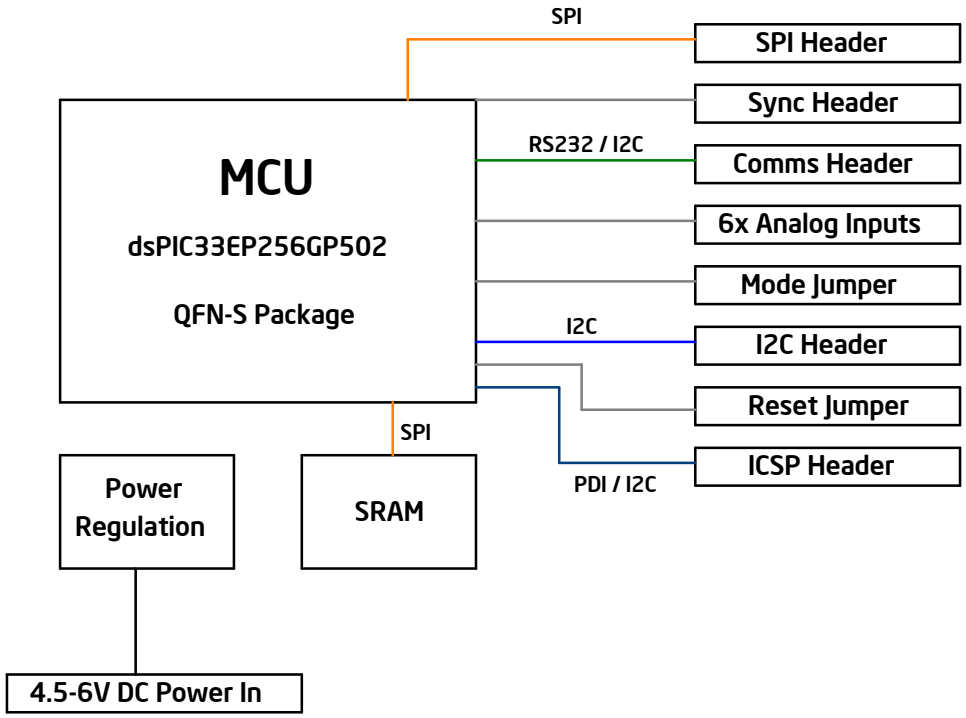
Revision History

Rev 1.0	Initial Release
Rev 1.1	See Box

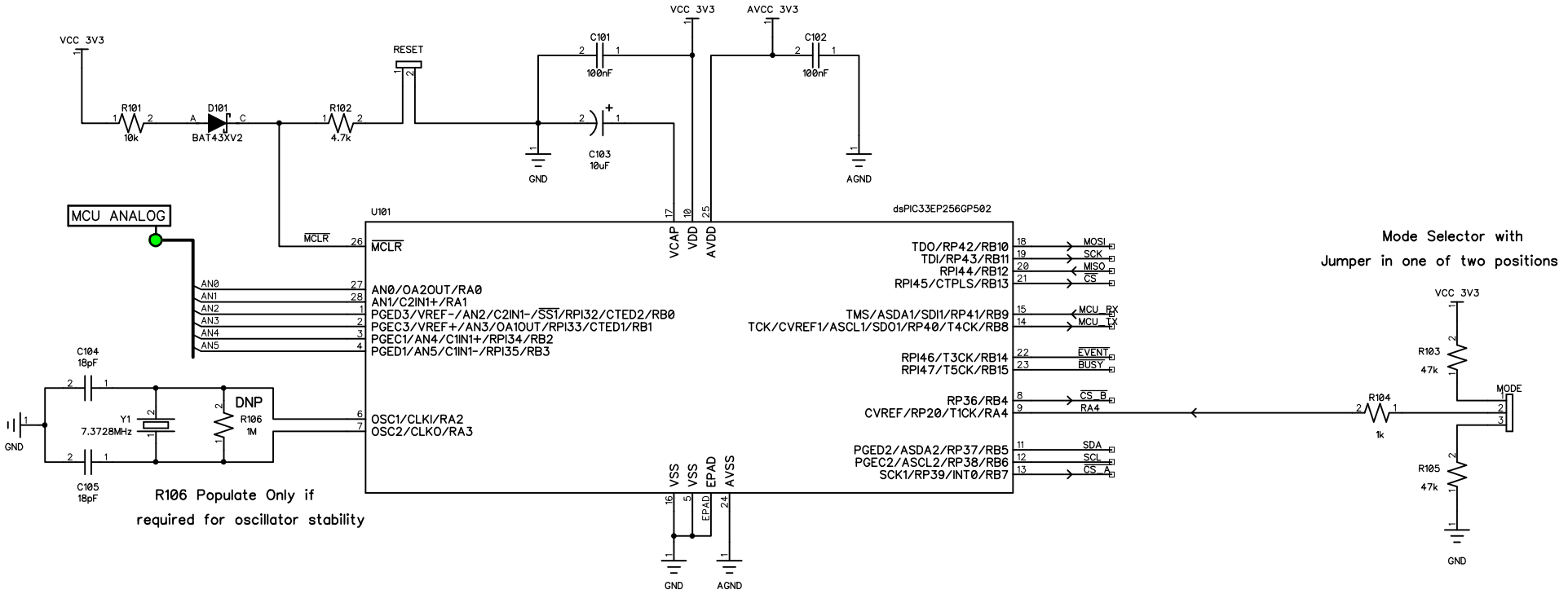
Rev 1.1 Changes

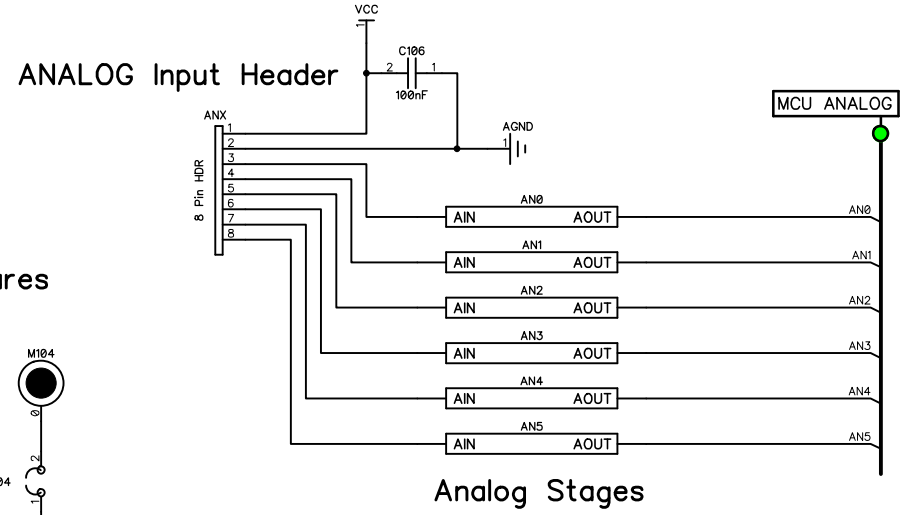
Changed SMD fuse to THR part
Merged COMMS and SYNC into single HDR
Changed MCU to dsPIC33EP256GP502
Added 10nF CAP on ALINK

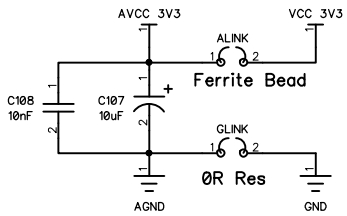
High Level Diagram

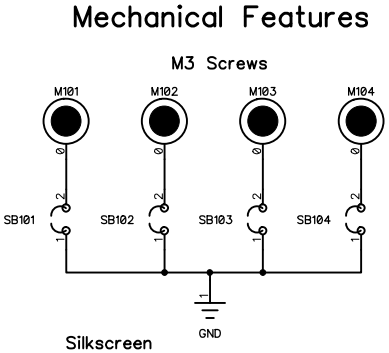


CB Aerospace	Engineer: Igor Dimitrijevic
Recording Board Index	Project: DAQ - Recording Board (Standard)
	Date: 8 May 2014
Page: 1 of 6	Revision Number: 1.1







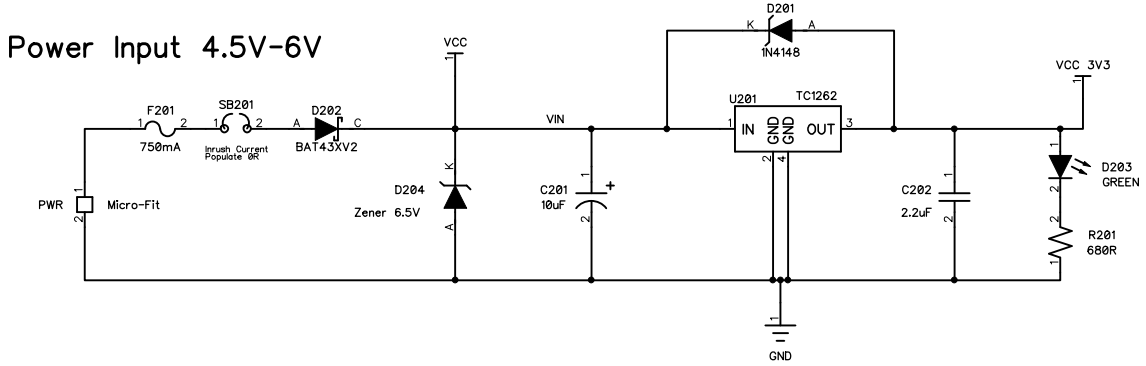


Silkscreen

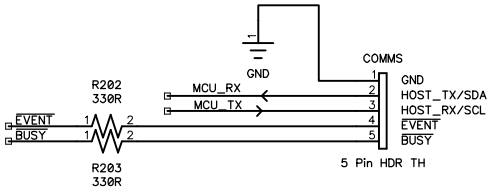
SERIAL NUMBER

CB Aerospace	Engineer: Igor Dimitrijevic
MCU	Project: DAQ - Recording Board (Standard)
	Date: 8 May 2014
Page: 2 of 6	Revision Number: 1.1

Power Input 4.5V-6V

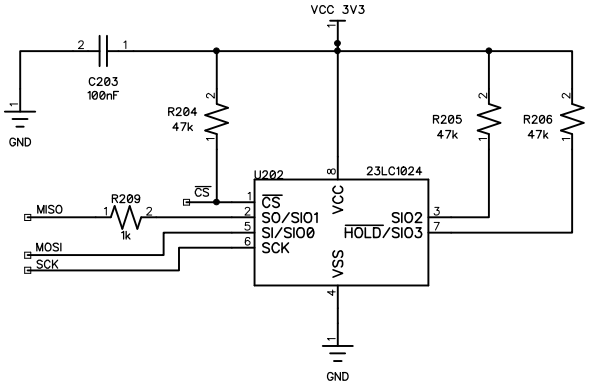


TTL_232 or I2C as selected
by MODE jumper - pin R4 on MCU
If used in I2C mode, pull-up resistors
assumed to be on host board.



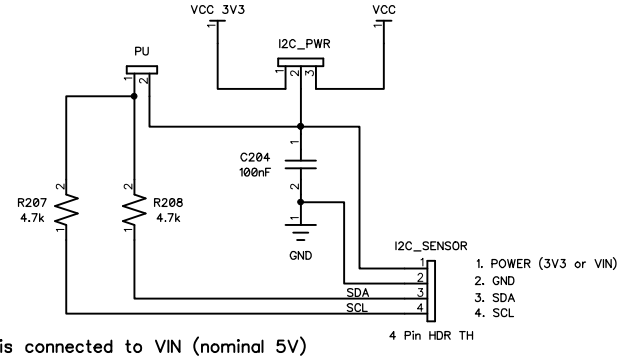
May be Input or Output
If used as output set to open-drain
so any recording board
can pull signal active low.

SRAM



I2C Header

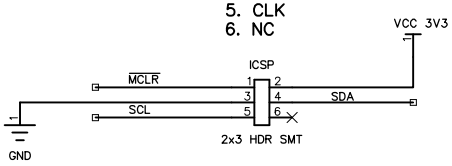
I2C_PWR Header Selects Sensor Power (3V3 or VIN)
PU Header Installs Pull-Ups on I2C Lines (Remove for Programming)



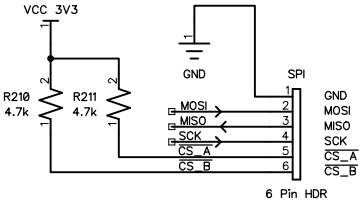
If I2C Power is connected to VIN (nominal 5V)
SDA and SCL Pins of the MCU are upto 5.5V tolerant

Programming Header

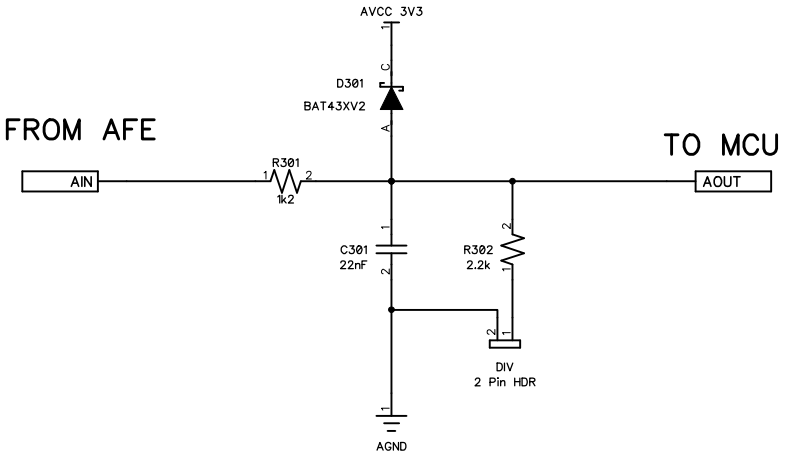
1. $\overline{\text{MCLR}}$
2. 3V3
3. GND
4. DATA
5. CLK
6. NC



SPI Header with two CS lines



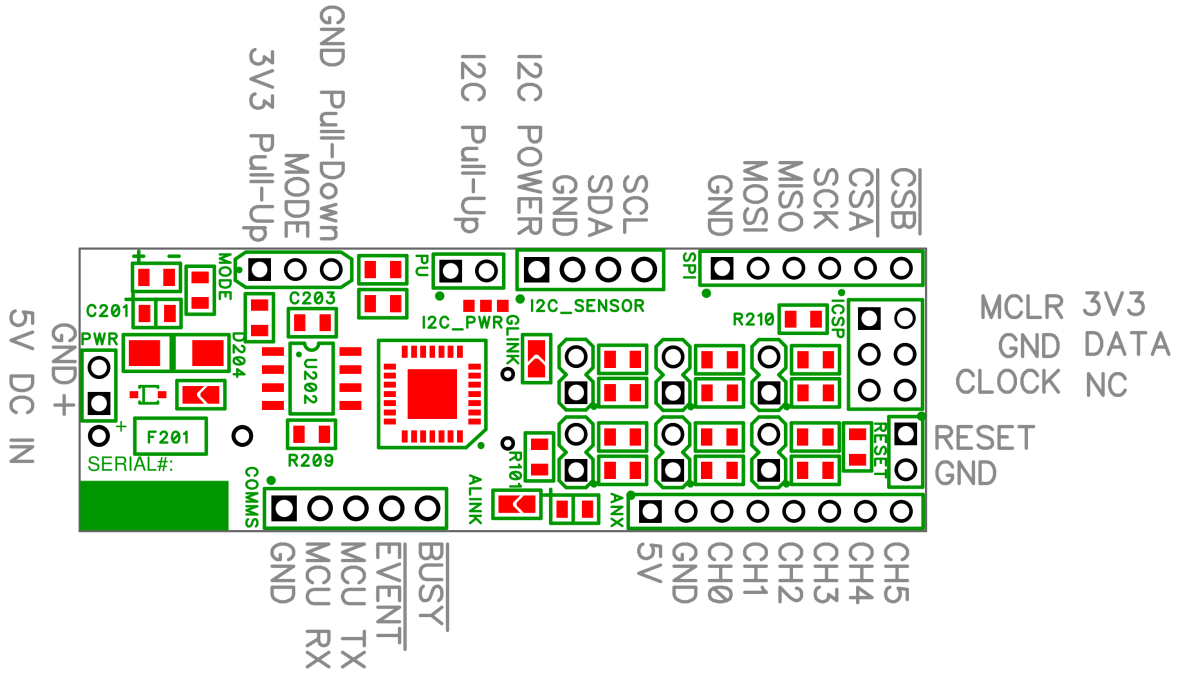
CB Aerospace	Engineer: Igor Dimitrijevic
Power and Comms	Project: DAQ - Recording Board (Standard)
	Date: 8 May 2014
Page: 3 of 6	Revision Number: 1.1



CB Aerospace	Engineer: Igor Dimitrijevic
Analog Stage	Project: DAQ - Recording Board (Standard)
	Date: 8 May 2014
Page: 4 of 6	Revision Number: 1.1

2.2 Recording board, slender format

The slender-format recording board is electrically identical to the standard recording board but its layout has been compressed to make the minimum dimension 20 mm. In doing so, the labels of the pins around the periphery of the board have been omitted. The figure below provides a key to the pin connections.



PAGES

Revision History

Rev 1.0	Initial Release
Rev 1.1	See Box
Rev 1.2	See Box

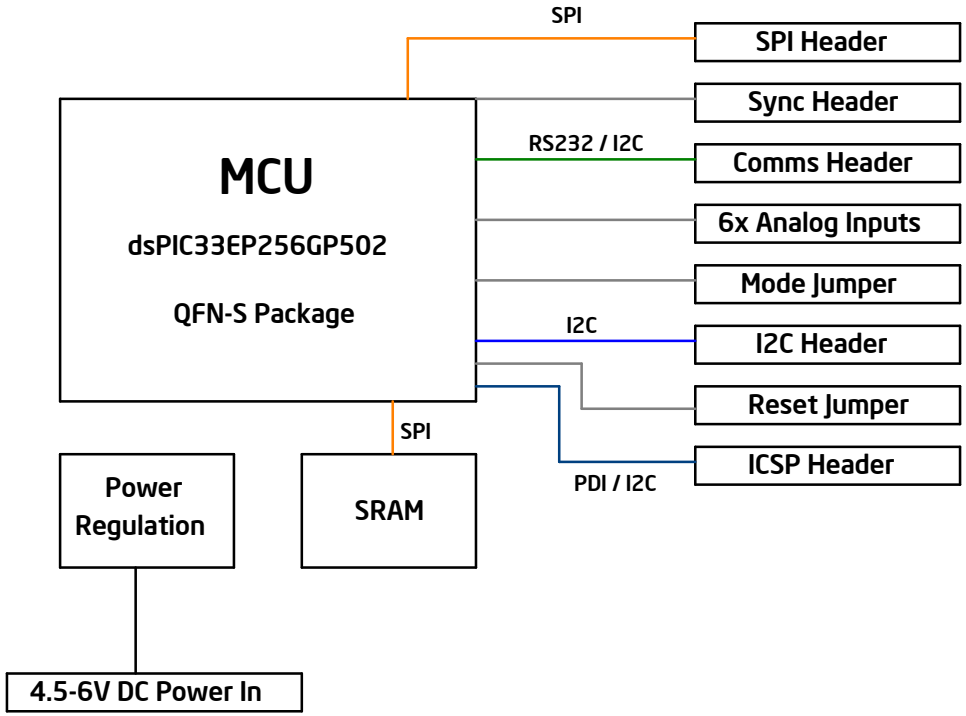
Rev 1.1 Changes

Changed SMD fuse to THR part
Merged COMMS and SYNC into single HDR
Changed MCU to dsPIC33EP256GP502
Added 10nF CAP on ALINK

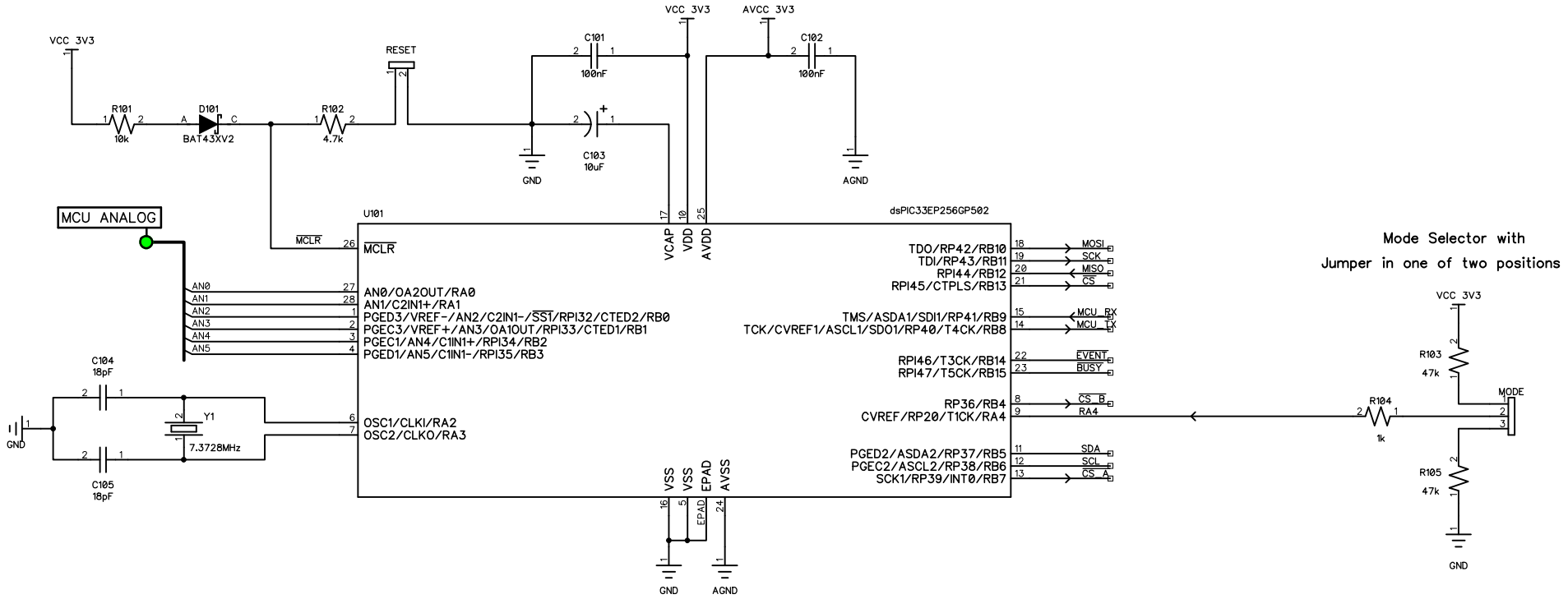
Rev 1.2 Changes

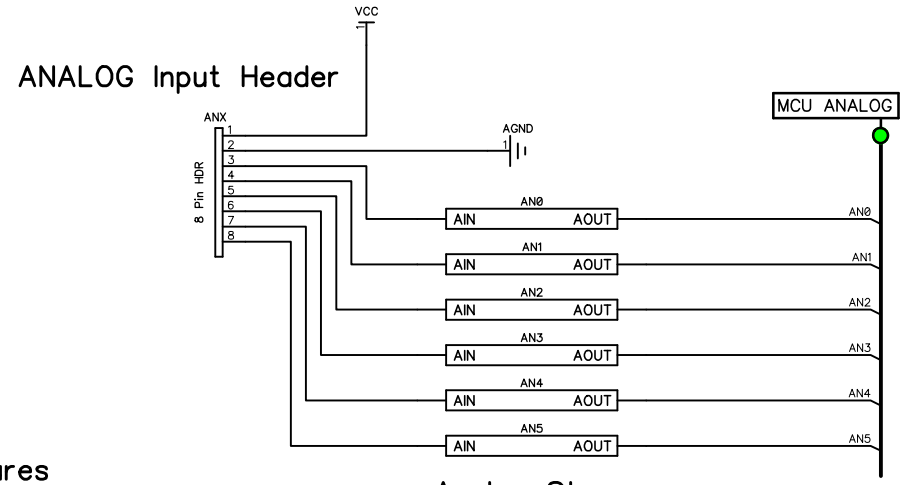
Change PWR connector to 2 PIN HDR
Removed Mounting Holes
Changed ICSP to Through Hole HDR

High Level Diagram

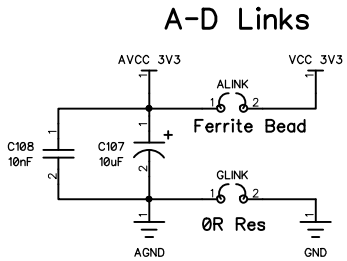


CB Aerospace	Engineer: Igor Dimitrijevic
Recording Board Index	Project: DAQ - Recording Board (Slender)
	Date: 8 June 2014
Page: 1 of 6	Revision Number: 1.2





Analog Stages



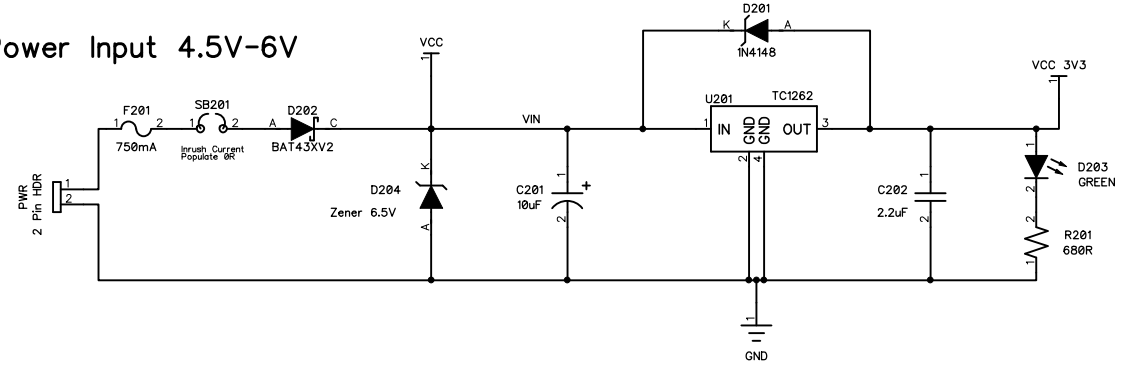
Mechanical Features

Silkscreen

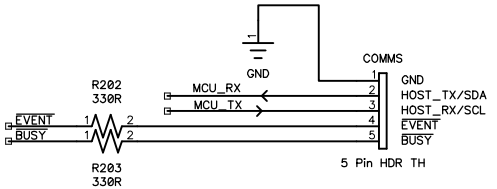
SERIAL
NUMBER

CB Aerospace	Engineer: Igor Dimitrijevic
MCU	Project: DAQ - Recording Board (Slender)
	Date: 8 June 2014
Page: 2 of 6	Revision Number: 1.2

Power Input 4.5V-6V

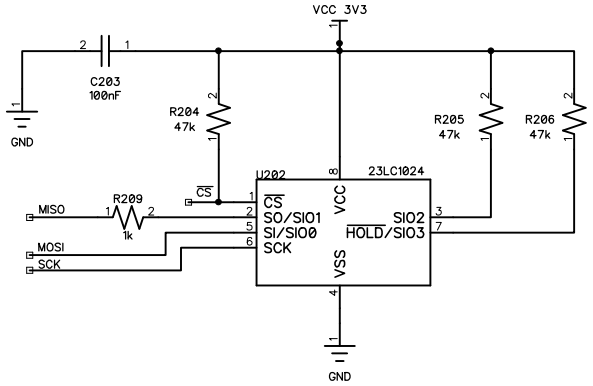


TTL_232 or I2C as selected
by MODE jumper - pin R4 on MCU
If used in I2C mode, pull-up resistors
assumed to be on host board.



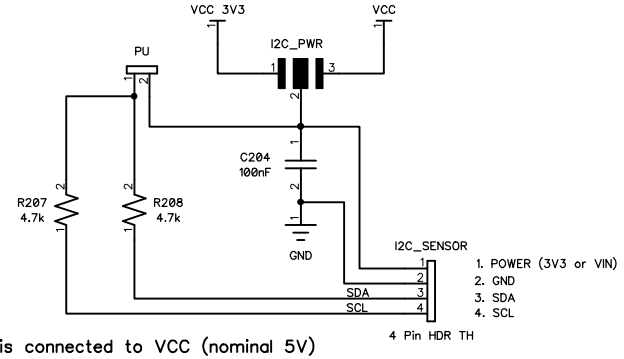
May be Input or Output
If used as output set to open-drain
so any recording board
can pull signal active low.

SRAM



I2C Header

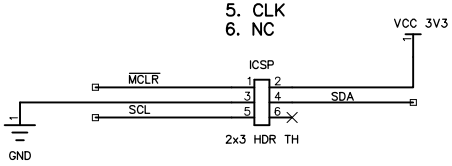
I2C_PWR Solder Bridge Selects Sensor Power (3V3 or VIN)
J201 Header Installs Pull-Ups on I2C Lines (Remove for Programming)



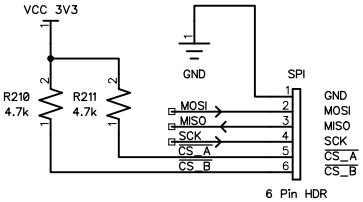
If I2C Power is connected to VCC (nominal 5V)
SDA and SCL Pins of the MCU are upto 5.5V tolerant

Programming Header

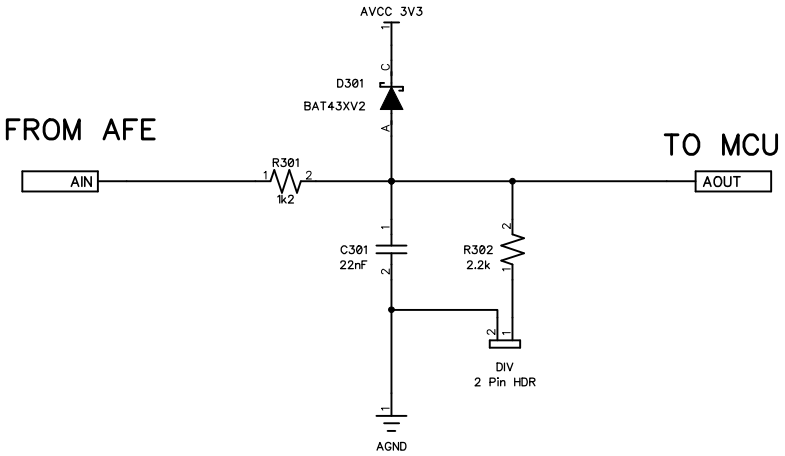
1. $\overline{\text{MCLR}}$
2. 3V3
3. GND
4. DATA
5. CLK
6. NC



SPI Header with two CS lines



CB Aerospace	Engineer: Igor Dimitrijevic
Power and Comms	Project: DAQ - Recording Board (Slender)
	Date: 8 June 2014
Page: 3 of 6	Revision Number: 1.2



CB Aerospace	Engineer: Igor Dimitrijevic
Analog Stage	Project: DAQ - Recording Board (Slender)
	Date: 8 June 2014
Page: 4 of 6	Revision Number: 1.2

2.3 Recording board, with programmable-gain amplifiers

The recording board with programmable-gain amplifiers has the same digital capability as the standard recording board, but with the enhancement that each analog channel has a programmable-gain amplifier.

PAGES

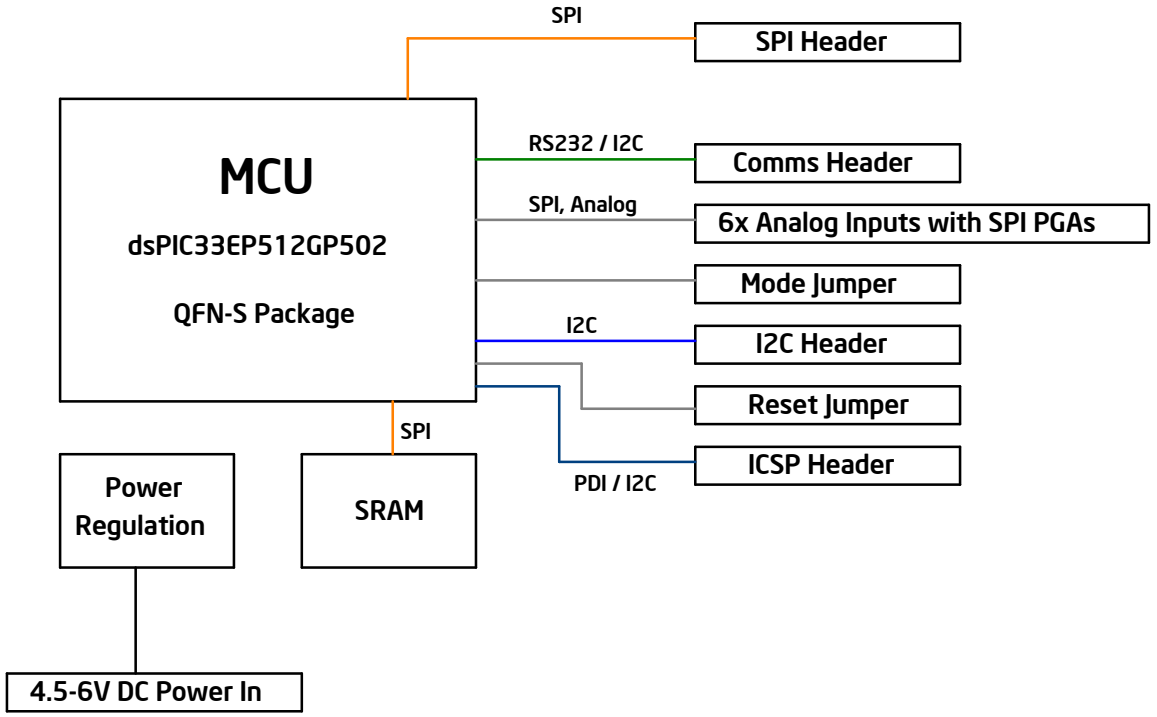
Revision History

Rev 1.0	Initial Release
Rev 1.1	See Rev Box

Rev 1.1 Changes

1. Merged $\overline{\text{EVENT}}$ Signal to Comms Header
2. Changed fuse to THR part
3. Changed MCU to dsPIC33EP256GP502
4. Added a 10nF cap to ALINK link
5. Recommed ferrite and 0R for ALINK and GLINK

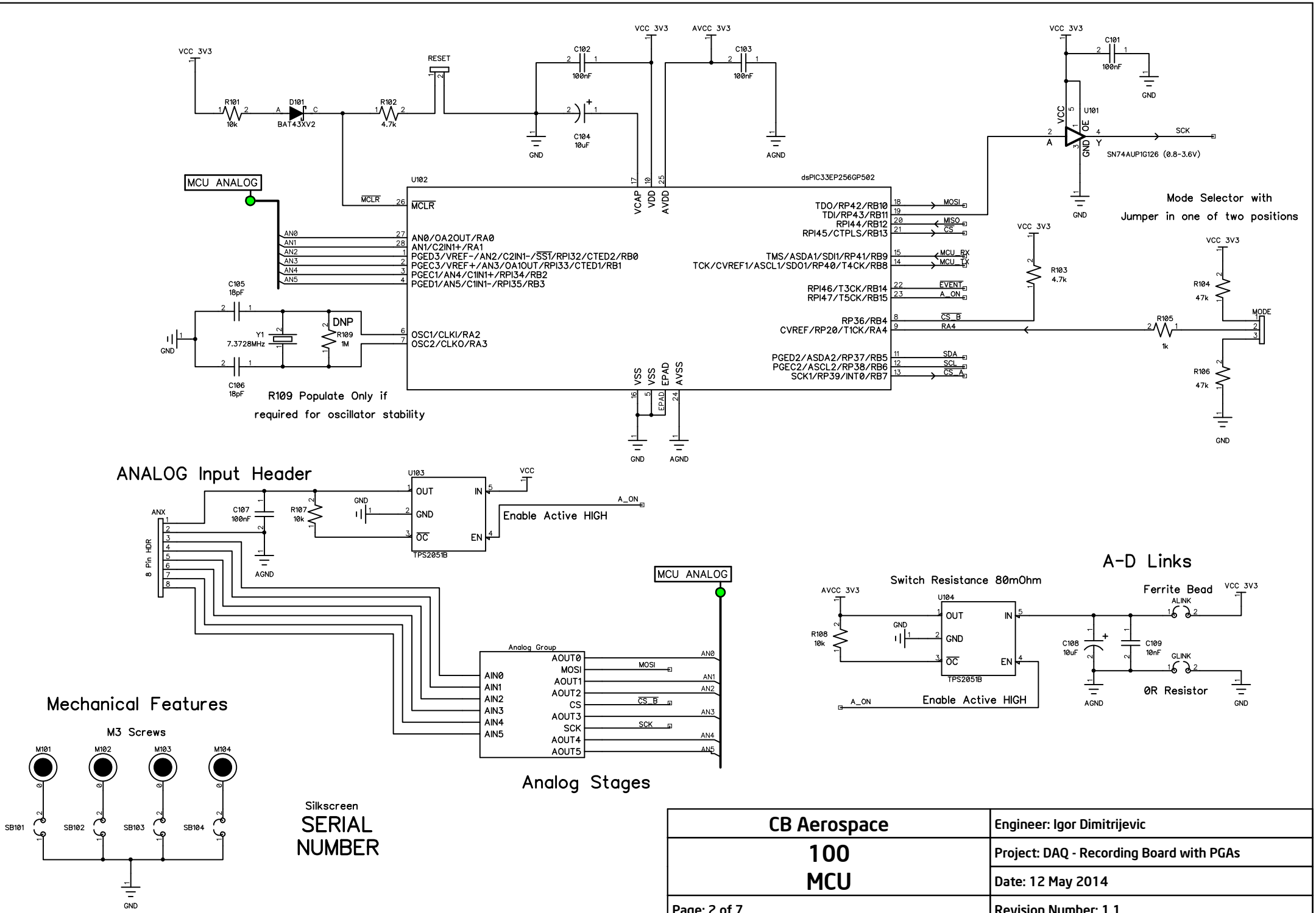
High Level Diagram



Differences to Recording Board Fixed Gain

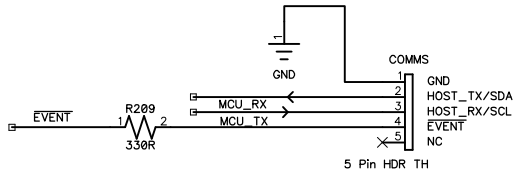
1. Added PGAs to analog stages
2. MCU RB15 pin turns power on to Analog circuitry and analog header (Active HIGH)
3. Removed $\overline{\text{Busy}}$ signal
4. Signal $\overline{\text{CS\_B}}$ now used for digipots on analog stages
5. $\overline{\text{CS\_B}}$ not availabe for external use and removed from SPI header
6. SPI Header contains $\overline{\text{Event}}$ signal instead of $\overline{\text{CS\_B}}$
7. Removed Sync header
8. MCU Pin RB11 now buffered

CB Aerospace	Engineer: Igor Dimitrijevic
INDEX	Project: DAQ - Recording Board with PGAs
	Date: 12 May 2014
Page: 1 of 7	Revision Number: 1.1

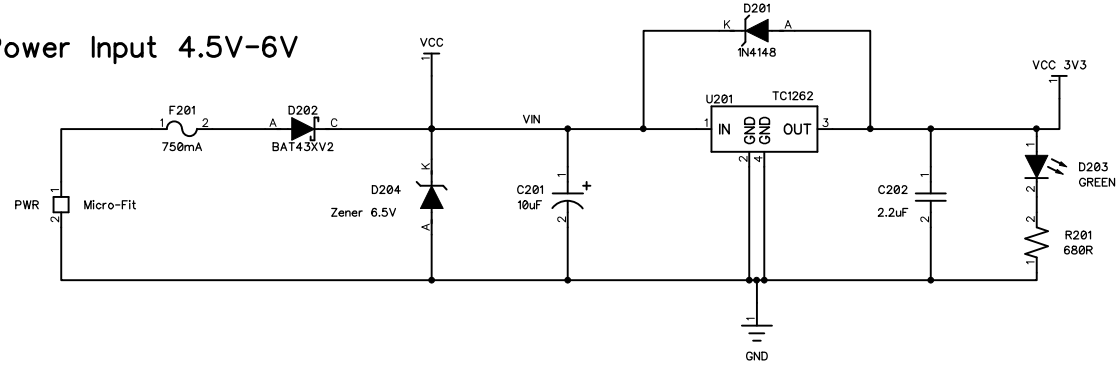


CB Aerospace	Engineer: Igor Dimitrijevic
100 MCU	Project: DAQ - Recording Board with PGAs
	Date: 12 May 2014
Page: 2 of 7	Revision Number: 1.1

TTL_232 or I2C as selected
by MODE jumper - pin R4 on MCU
If used in I2C mode, pull-up resistors
assumed to be on host board.

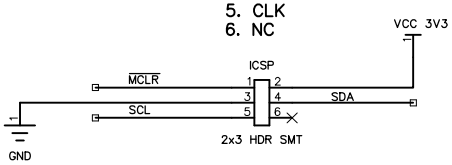


Power Input 4.5V-6V

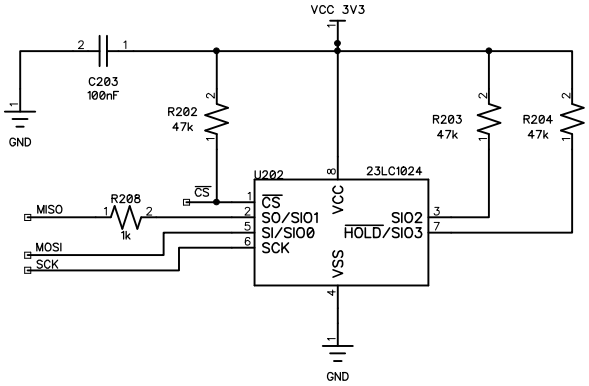


Programming Header

1. $\overline{\text{MCLR}}$
2. 3V3
3. GND
4. DATA
5. CLK
6. NC

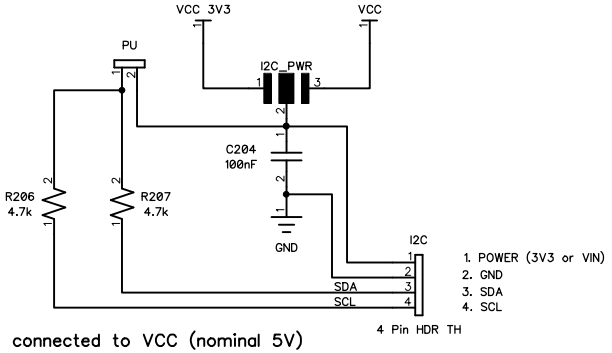


SRAM



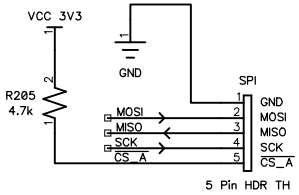
I2C Header

I2C_PWR Solder Bridge Selects Sensor Power (3V3 or VIN)
J201 Header Installs Pull-Ups on I2C Lines (Remove for Programming)

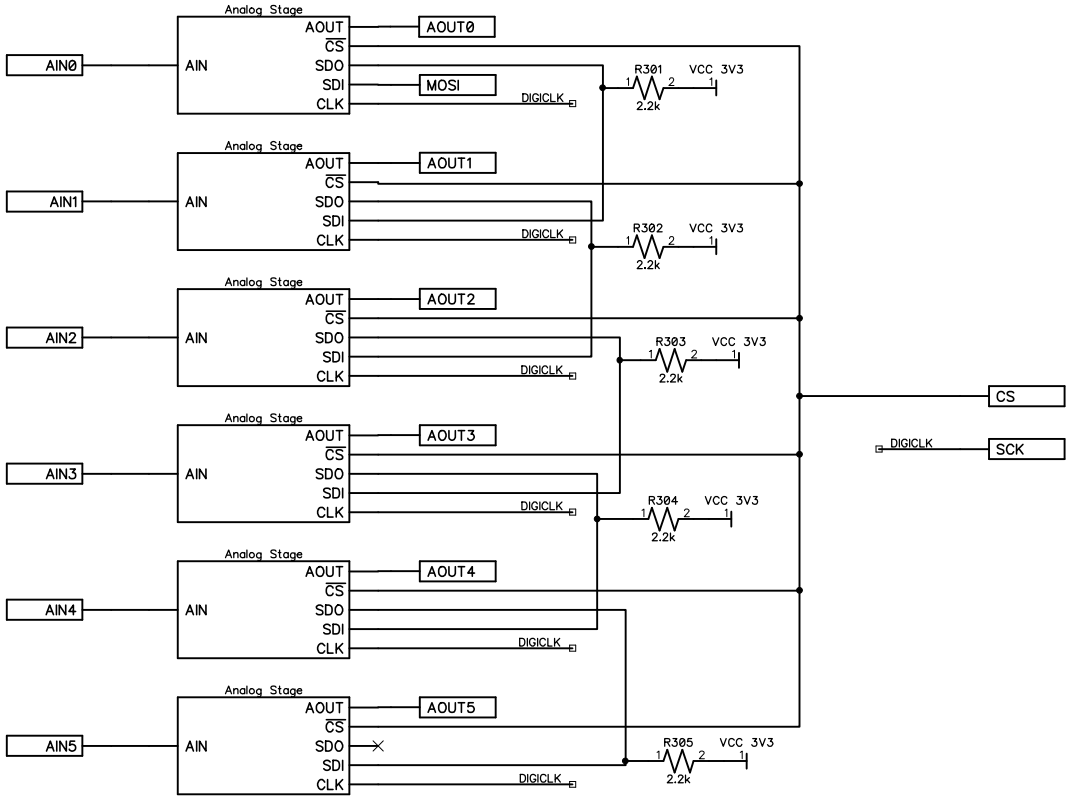


If I2C Power is connected to VCC (nominal 5V)
SDA and SCL Pins of the MCU are upto 5.5V tolerant

SPI Header with CS line, $\overline{\text{EVENT}}$ signal



CB Aerospace	**Engineer: Igor Dimitrijevic**
200 **Power and Comms**	**Project: DAQ - Recording Board with PGAs**
	Date: 12 May 2014
Page: 3 of 7	**Revision Number: 1.1**



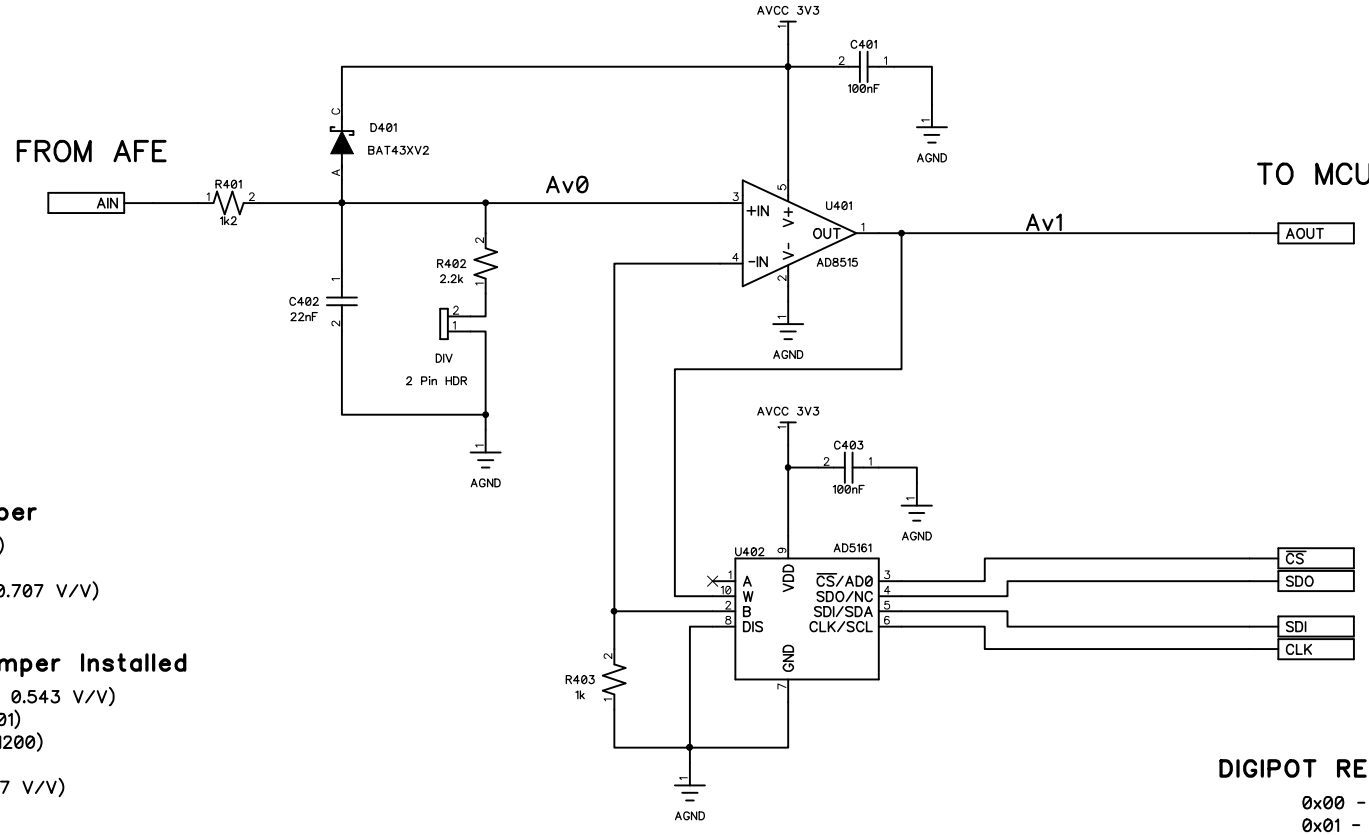
CB Aerospace	**Engineer: Igor Dimitrijevic**
300 **Analog Group**	**Project: DAQ - Recording Board with PGAs**
	Date: 12 May 2014
Page: 4 of 7	**Revision Number: 1.1**

Non-Inverting Amplifier Configuration

Av0 / Av1 = Aout = 1 + (Rwb / R303)
Aout = 1 + (Rwb / R303)

At D = 0 (0x00), DIV jumper not installed
Aout = 1 + (60 / 1k) = 1.06 V/V = 0.506 dB
At D = 128 (0x80), DIV jumper not installed
Aout = 1 + (50060 / 1k) = 51.06 V/V = 34.16 dB
At D = 255 (0xFF), DIV jumper not installed
Aout = 1 + (99610 / 1k) = 100.61 V/V = 40.05 dB

At D = 0 (0x00), DIV jumper installed
Aout = [1 + (60 / 1k)] * 0.647 = 0.685 V/V = -3.276 dB
At D = 128 (0x80), DIV jumper installed
Aout = [1 + (50060 / 1k)] * 0.647 = 33.03 V/V = 30.37 dB
At D = 255 (0xFF), DIV jumpe installed
Aout = [1 + (99610 / 1k)] * 0.647 = 65.09 V/V = 36.27 dB



RC Filter - No Jumper

Fc = 1 / (2*pi*R301*C301)
Fc = 1 / (2*pi*1200*22n)
Fc = 6028 Hz (-3dB or 0.707 V/V)
Av0 at DC: 0dB (1 V/V)

RC Filter - With Jumper Installed

Fc = 6028 Hz (-5.3dB or 0.543 V/V)
Av0 = R302 / (R302+R301)
Av0 = 2200 / (2200 + 1200)
Av0 = 0.647
Av0 at DC: -3.78dB (0.647 V/V)

DIGIPOT RESISTANCE and HEX CODES

0x00 - 60 Ohm	Rwb(D) = (D / 256) * Rab + Rw
0x01 - 450 Ohm	Rab = 100k
0x02 - 841 Ohm	Rw = 60 Ohm
0x80 - 50060 Ohm	Midscale (D=128) - Powerup Condition
0xFF - 99610 Ohm	Full Scale: Rab - 1LSB + Rw

NOTE: For Connecting Wheatstone Bridge Sensors

1. Use InAmp such as INA125 on sensor AFE in differential to single ended configuration, or
2. Use two ADC channels for V+ and V- and process the difference in software

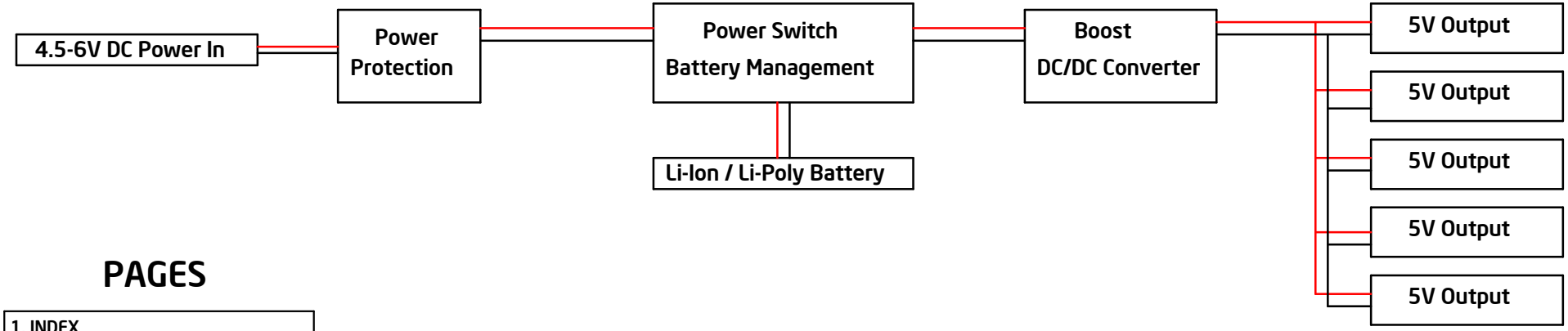
CB Aerospace	Engineer: Igor Dimitrijevic
400 Analog Stage	Project: DAQ - Recording Board with PGAs
	Date: 12 May 2014
Page: 5 of 7	Revision Number: 1.1

2.4 Power board, standard format

The power management board may be used to distribute nominal 5 V power to a number of recording, AFE boards and/or sensors. The bq24074 battery management chip [4] on the board can simultaneously provide power to downstream devices and charge a 4.2 V Li-Ion or Li-Poly battery. It can also operate in the absence of a battery.

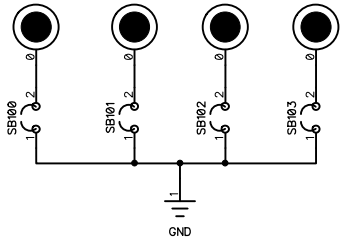
Also, on the power board is a TPS55340 switching regulator [5], which boosts the 4.4 V output from the battery management chip to a 5.3 V level for distribution to the other boards in a system. Five Micro-Fit 2-pin headers are mounted on the periphery of the board to provide convenient and reliable connections.

High Level Diagram



PAGES

Mechanical Features

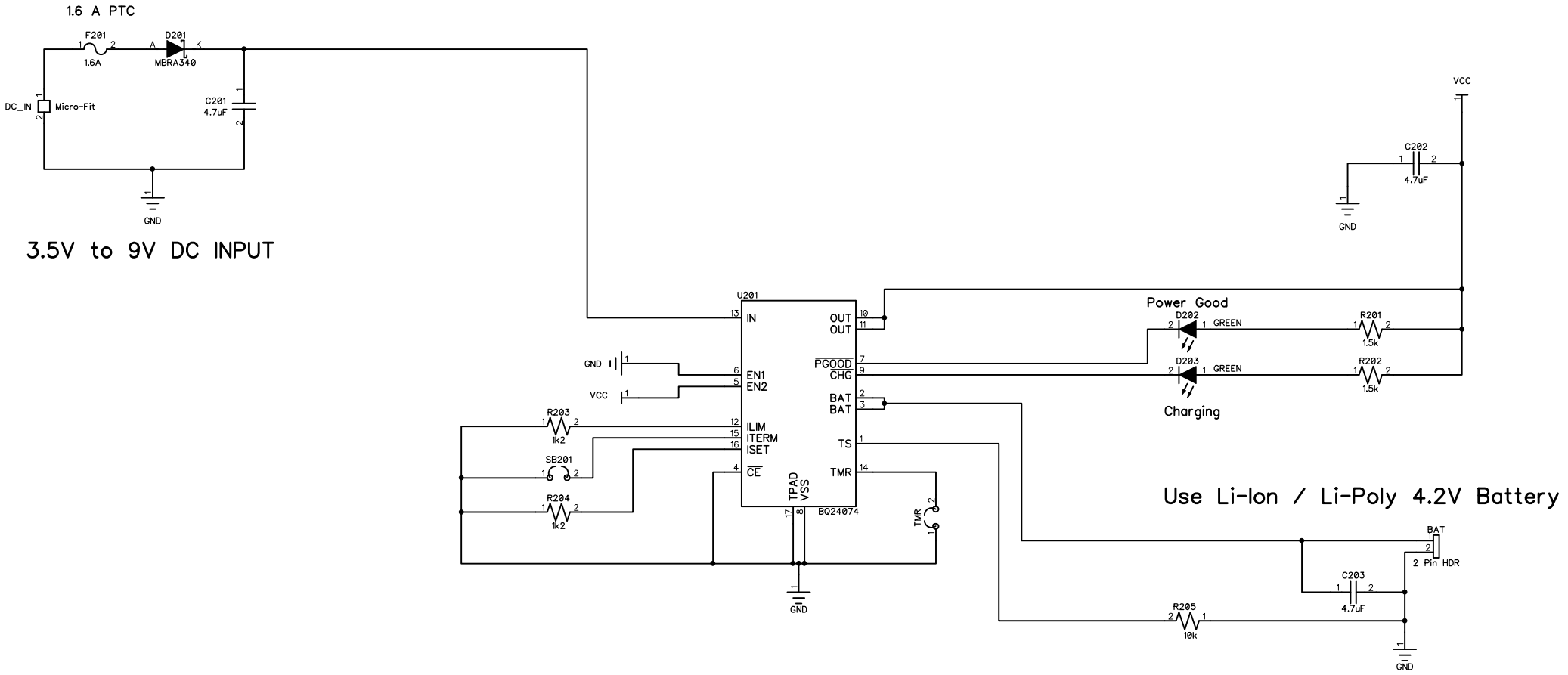


SERIAL
NUMBER

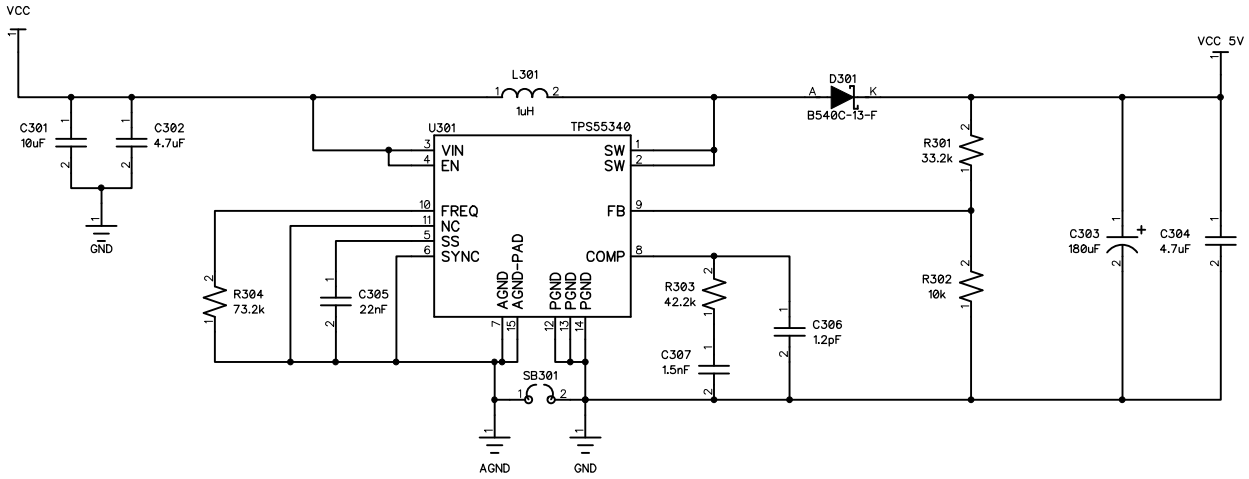
Revision History

Rev 1.0	Initial Release

CB Aerospace	Engineer: Igor Dimitrijevic
100 Index	Project: DAQ - Power Board
	Date: 11 June 2014
Page: 1 of 5	Revision Number: 1.0

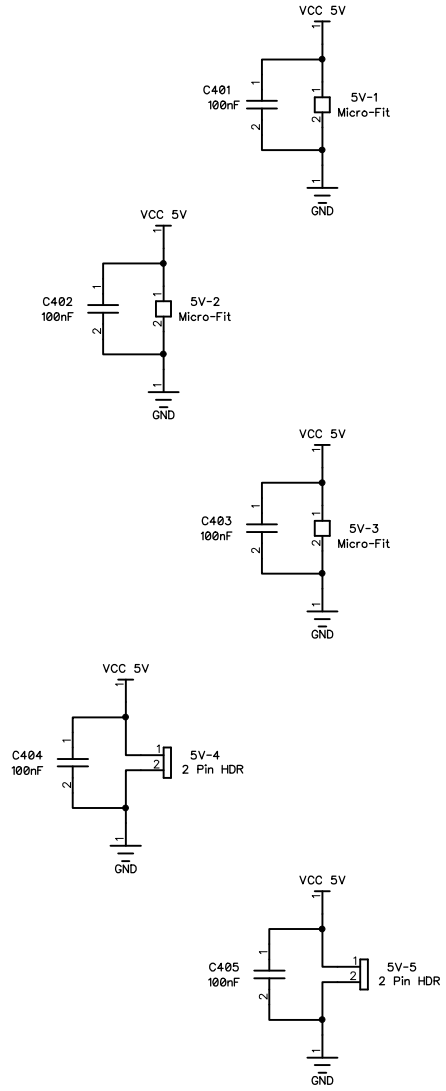


CB Aerospace	Engineer: Igor Dimitrijevic
200 POWER PATH	Project: DAQ - Power Board
	Date: 11 June 2014
Page: 2 of 5	Revision Number: 1.0



SB301: Populate with 0R close to IC

CB Aerospace	Engineer: Igor Dimitrijevic
300 BOOST CONVERTER	Project: DAQ - Power Board
	Date: 11 June 2014
Page: 3 of 5	Revision Number: 1.0



CB Aerospace	**Engineer: Igor Dimitrijevic**
400 OUTPUTS	**Project: DAQ - Power Board**
	Date: 11 June 2014
Page: 4 of 5	**Revision Number: 1.0**

2.5 Analog-front-end board, standard format

The Analog Front End (AFE) board contains precision analog circuitry to perform signal conditioning of the transducer outputs and thus make them suitable for analog to digital conversion by the microcontroller on the Recording Board (or different hardware of choice). The AFE board contains three pressure channel circuits and three Thin Film Gauge (TFG) circuits but only the pressure-channel circuits are discussed here.

The pressure channels are meant to amplify a differential analog signal as output by unamplified Wheatstone bridge type pressure transducers. Examples of transducers of this type used in the plasma tunnels and experiments are the Honeywell SDX series [6] and the Kulite XCEL series.

As an example, we consider the use of an SDX 100 psi absolute sensor (such as SDX100A2). At a nominal 12V supply voltage, the sensor has a Common Mode (CM) of 1.5-5V, and a Full Scale Span (FSS) of 90mV. The AFE board supplies a precision reference of 4.5V for the transducer bridge, and hence the CM and FSS need to be adjusted for a supply of 4.5V as opposed to nominal 12V. The new CM range is now 0.5625-1.875V and the new FSS is 33.75mV. With these values, the full scale span between zero pressure (or pressure difference) and the top measurable pressure difference of 33.75mV is too small to utilize the full dynamic range of the analog to digital converter (0-3.3V) on the Recording Board. Hence, the sensor output is fed into an instrumentation amplifier, INA122, set to a gain of 100.24 V/V by the resistor connected to pins 1 and 6 of the INA122 IC. This implies the difference seen between the signals Vin+ and Vin- on the AFE is amplified 100.24 times before being output to the ADC. Now the full scale span of 33.75mV becomes 3.3831 Volts, which fully utilizes the range of the ADC.

The recording board then contains an anti-aliasing filter to ensure the higher frequency elements are heavily attenuated and that the signal can be reconstructed after analog to digital conversion unambiguously.

When using the absolute pressure sensors such as mentioned above, the Vin- signal is fixed referenced to vacuum and hence only the Vin+ signal swings, effectively making the sensor appear as a single ended sensor. In the case when gauge sensors are used, the Vin- signal can swing higher than the Vin+ signal in a truly differential manner. For this purpose, the reference of the INA122 instrumentation amplifier can be changed by a reference set by a OPA(2)347 amplifier. In default configuration, the reference is set to about 450mV, hence the output signal can be shifted by this amount. When using the absolute transducers, set

the reference to 0V (ground) but when using differential or gauge pressure transducers, offset the reference to 450mV if required. The reference can be set by moving the jumper labeled REF on the AFE board itself. Offsetting the reference is required to avoid clipping the output signal when the Vin- input signal is higher than the Vin+ signal. In the common applications, it is expected that transducers of absolute type will be used and thus not require this reference shift.

In case the 450mV reference shift is not sufficient, the reference can be adjusted to any value by changing the ratio of through hole resistors connected to the OPA(2)347 amplifier non-inverting input. For instance, if a differential sensor with two ports is used, and if the pressure on both ports is expected to be able to swing "full scale negative" as well as "full scale positive", the resistor ratio should be set to 1 (both resistors equal value), to offset the reference to mid-rail, and then the output signal swing can vary from mid-rail to positive rail and mid-rail to ground, again utilizing the full dynamic range of the ADC.

Finally, as the instrumentation amplifier rejects the common mode and only amplifies the difference of the input signals, the common mode offset is irrelevant. This is worthy of note, because the common mode of the SDX sensor can vary quite a bit from sensor to sensor, and due to this instrumentation amplifier, no adjustments for the common mode are required.

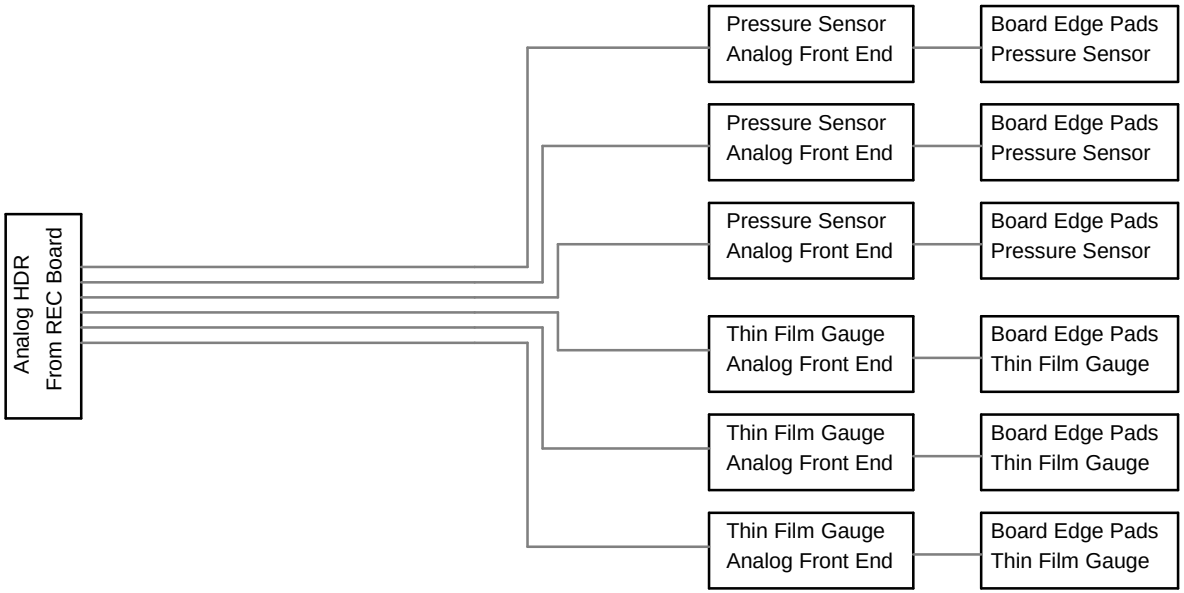
The in-situ calibration of the sensors is highly recommended. This is usually performed while preparing the tunnel for the experiment. Before the pressure in the tunnel is reduced, an atmospheric measurements should be made for each sensor, hence giving one calibration point. As the tunnel is evacuated and pressure brought to a vacuum or other required level, the second calibration point should be taken, hence producing a two point calibration of the sensors. If better calibration is required, it can be obtained by use of a special calibration rig, and then the same setup (in particular the same pressure sensors connected to the same ports on the AFE board and recording board) as used in calibration should be used for experimentation. This is because the calibration applies to each transducer (as each transducer is slightly different), to the analog circuitry of each channel, and to the different ADC offset and bias of each ADC channel.

PAGES

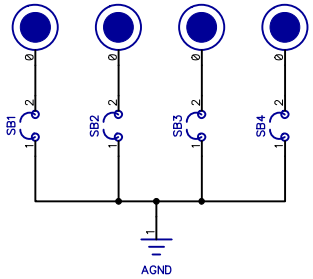
Revision History

Rev 1.0	Initial Release
Rev 1.1	Removed Output Buffers

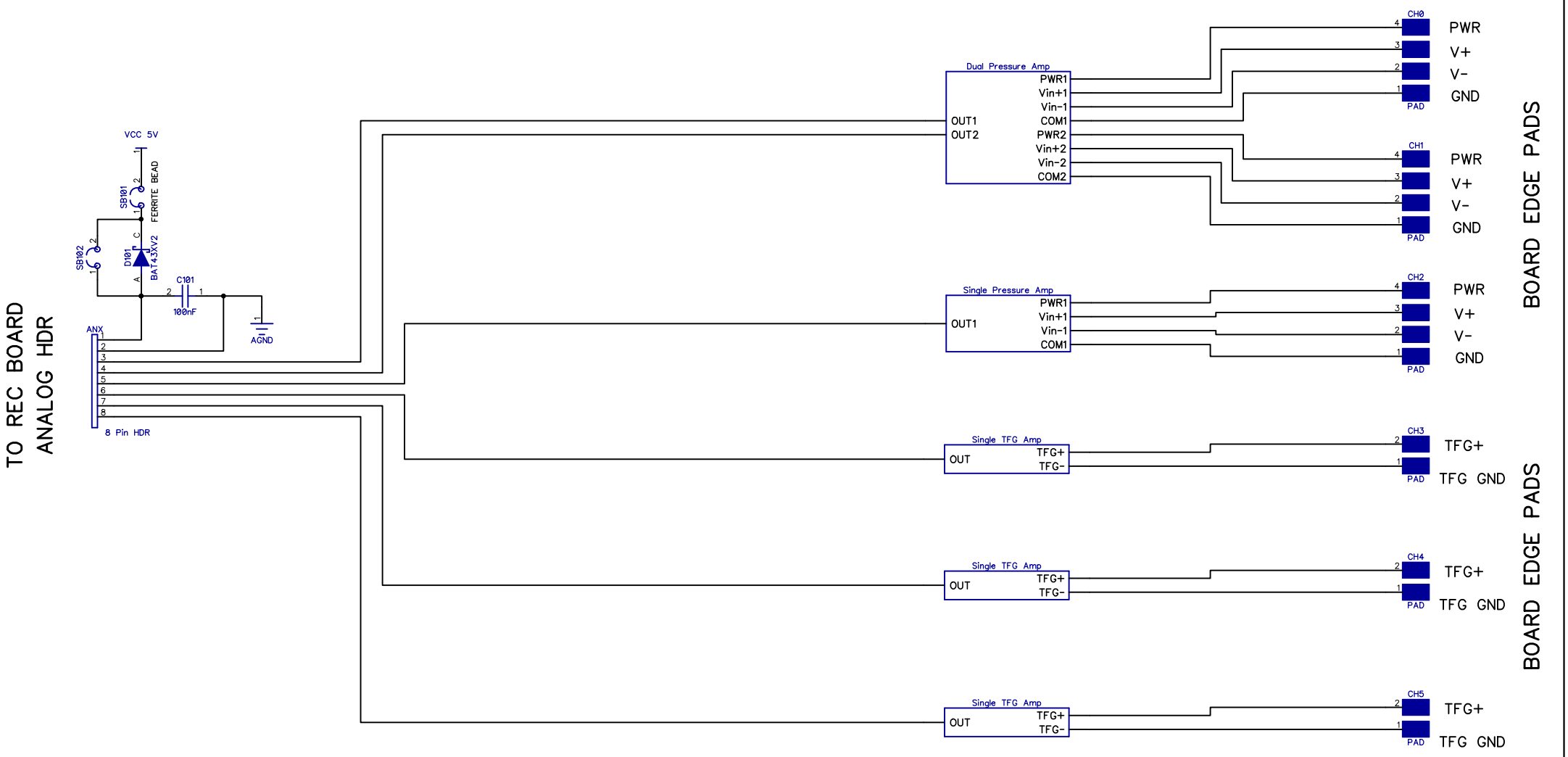
High Level Diagram



Mechanical Features



CB Aerospace	Engineer: Igor Dimitrijevic
INDEX	Project: DAQ - AFE Board
	Date: 24 October 2014
Page: 1 of 6	Revision Number: 1.1



CB Aerospace	Engineer: Igor Dimitrijevic
100 Analog Pressure Board	Project: DAQ - AFE Board
	Date: 24 October 2014
Page: 2 of 6	Revision Number: 1.1

Gain of INA122

G = 5 + 200k / Rg
With Rg = 2.1k; G = 100.24 V/V

SDX Absolute Pressure Sensor
Common Mode at Vs = 4.5V
Vcm(4.5) = 1.125 V
Full Scale Span at Vs = 4.5V
FSS(4.5) = 33.75 mV
Output: Vo = (Vin+ - Vin-)*G+Vref
At FSS:
Vo = [(1.125+(0.03375/2)) - (1.125+(0.03375/2))] * 100.24 + Vref
Vo = 3.3831 V + Vref

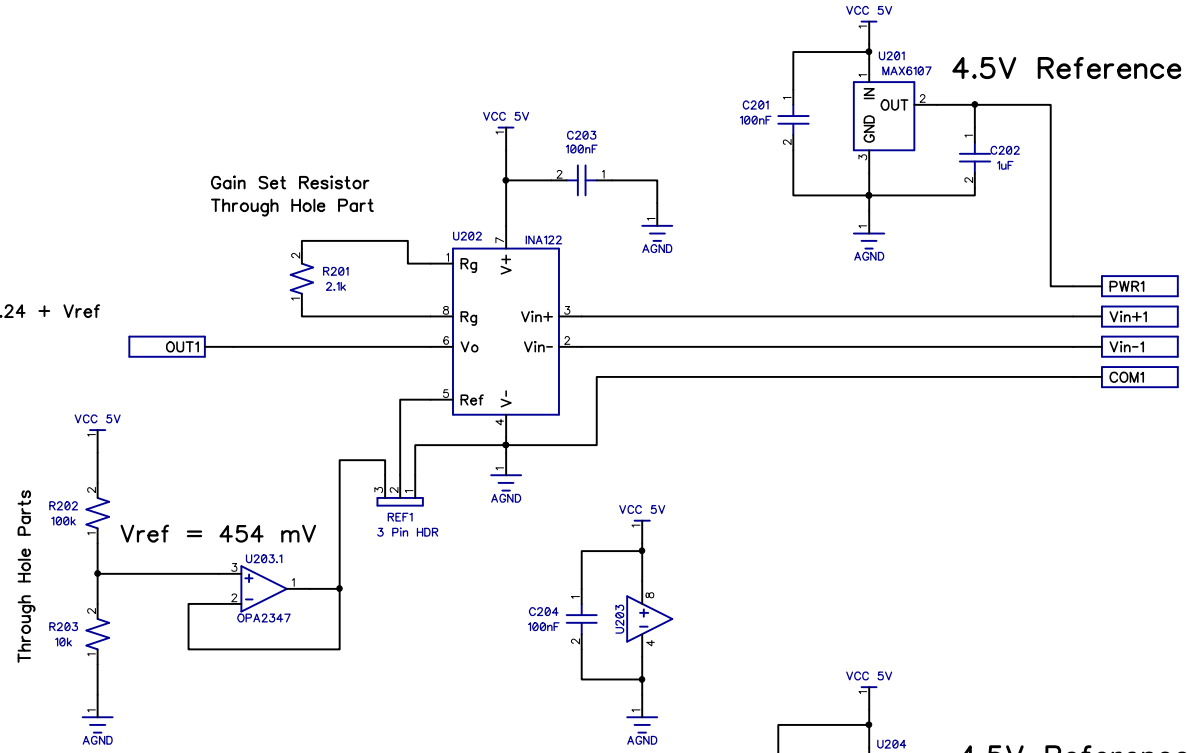
Bridge Sensors Note

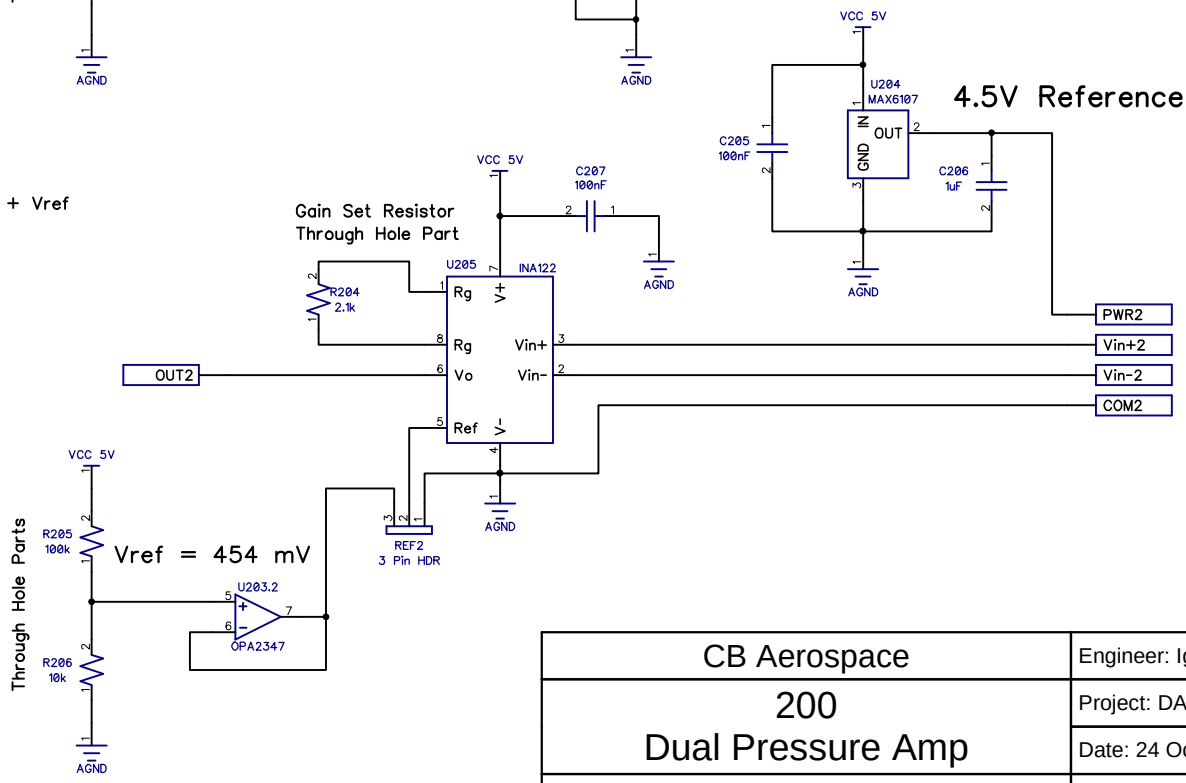
Common Mode Voltage Vcm and
Output Voltage Vin+ and Vin-
Are Ratiometric to Supply Voltage Vs

SDX Nominal Vs = 12V
Nominal Vcm = 3V at Vs = 12V
Nominal FSS = 90mV at Vs = 12V

Kulite Nominal Vs = 10V
Nominal Vcm = 5V at Vs = 10V
Nominal FSS = 100mV at Vs = 10V

Kulite XCEL Absolute Pressure Sensor
Common Mode at Vs = 4.5V
Vcm(4.5) = 2.25V
Full Scale Span at Vs = 4.5V
FSS(4.5) = 0.0375V
Output: Vo = (Vin+ - Vin-)*G+Vref
Vo = [(2.25+(0.0375/2)) - (2.25+(0.0375/2))] * 100.24 + Vref
Vo = 3.759 V + Vref





Voltage Reference Max Load: 5mA

SDX Input Impedance: 4k
Iref = 4.5/4k = 1.125mA

Kulite Input Impedance 1k
Iref = 4.5/1k = 4.5mA

This circuitry allows
setting the output reference
it shifts the output to avoid
saturating the amplifier and
clipping the signal waveform

CB Aerospace	Engineer: Igor Dimitrijevic
200 Dual Pressure Amp	Project: DAQ - AFE Board
	Date: 24 October 2014
Page: 3 of 6	Revision Number: 1.1

Gain of INA122

G = 5 + 200k / Rg
With Rg = 2.1k; G = 100.24 V/V

SDX Absolute Pressure Sensor
Common Mode at Vs = 4.5V
Vcm(4.5) = 1.125 V
Full Scale Span at Vs = 4.5V
FSS(4.5) = 33.75 mV
Output: Vo = (Vin+ - Vin-)*G+Vref
At FSS:
Vo = [(1.125+(0.03375/2)) - (1.125+(0.03375/2))] * 100.24 + Vref
Vo = 3.3831 V + Vref

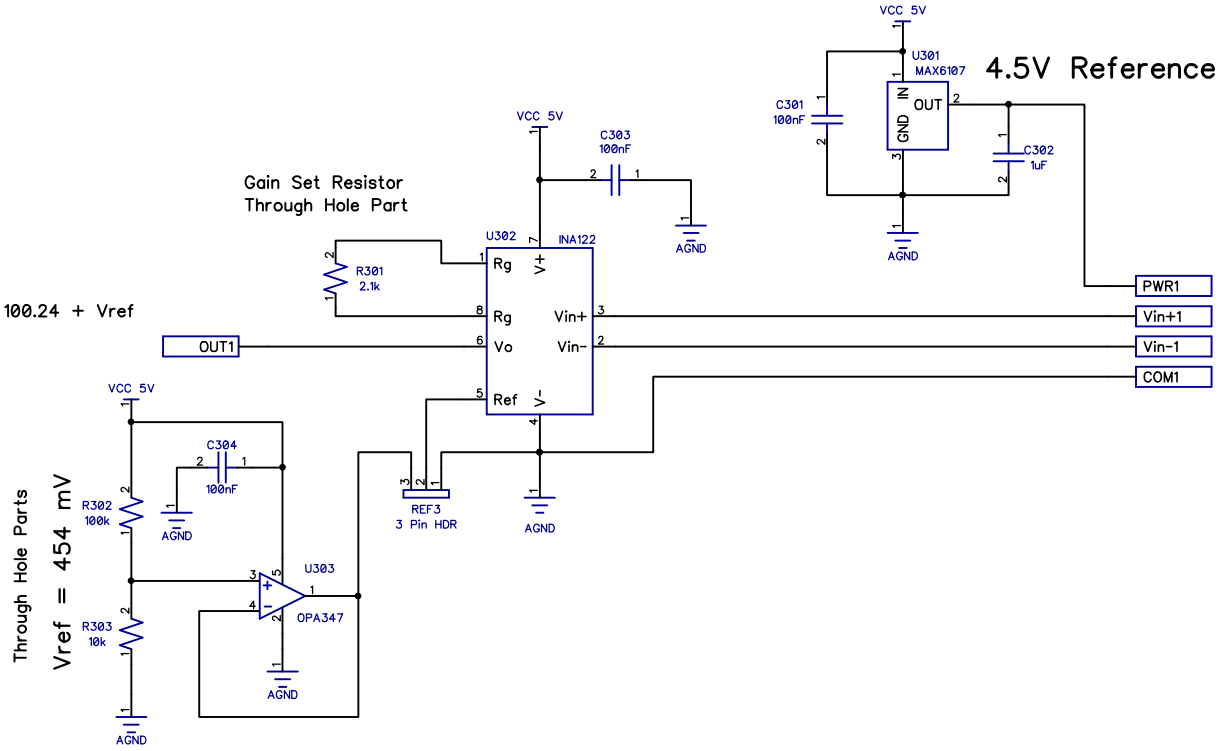
Bridge Sensors Note

Common Mode Voltage Vcm and
Output Voltage Vin+ and Vin-
Are Ratiometric to Supply Voltage Vs

SDX Nominal Vs = 12V
Nominal Vcm = 3V at Vs = 12V
Nominal FSS = 90mV at Vs = 12V

Kulite Nominal Vs = 10V
Nominal Vcm = 5V at Vs = 10V
Nominal FSS = 100mV at Vs = 10V

Kulite XCEL Absolute Pressure Sensor
Common Mode at Vs = 4.5V
Vcm(4.5) = 2.25V
Full Scale Span at Vs = 4.5V
FSS(4.5) = 0.0375V
Output: Vo = (Vin+ - Vin-)*G+Vref
Vo = [(2.25+(0.0375/2)) - (2.25+(0.0375/2))] * 100.24 + Vref
Vo = 3.759 V + Vref

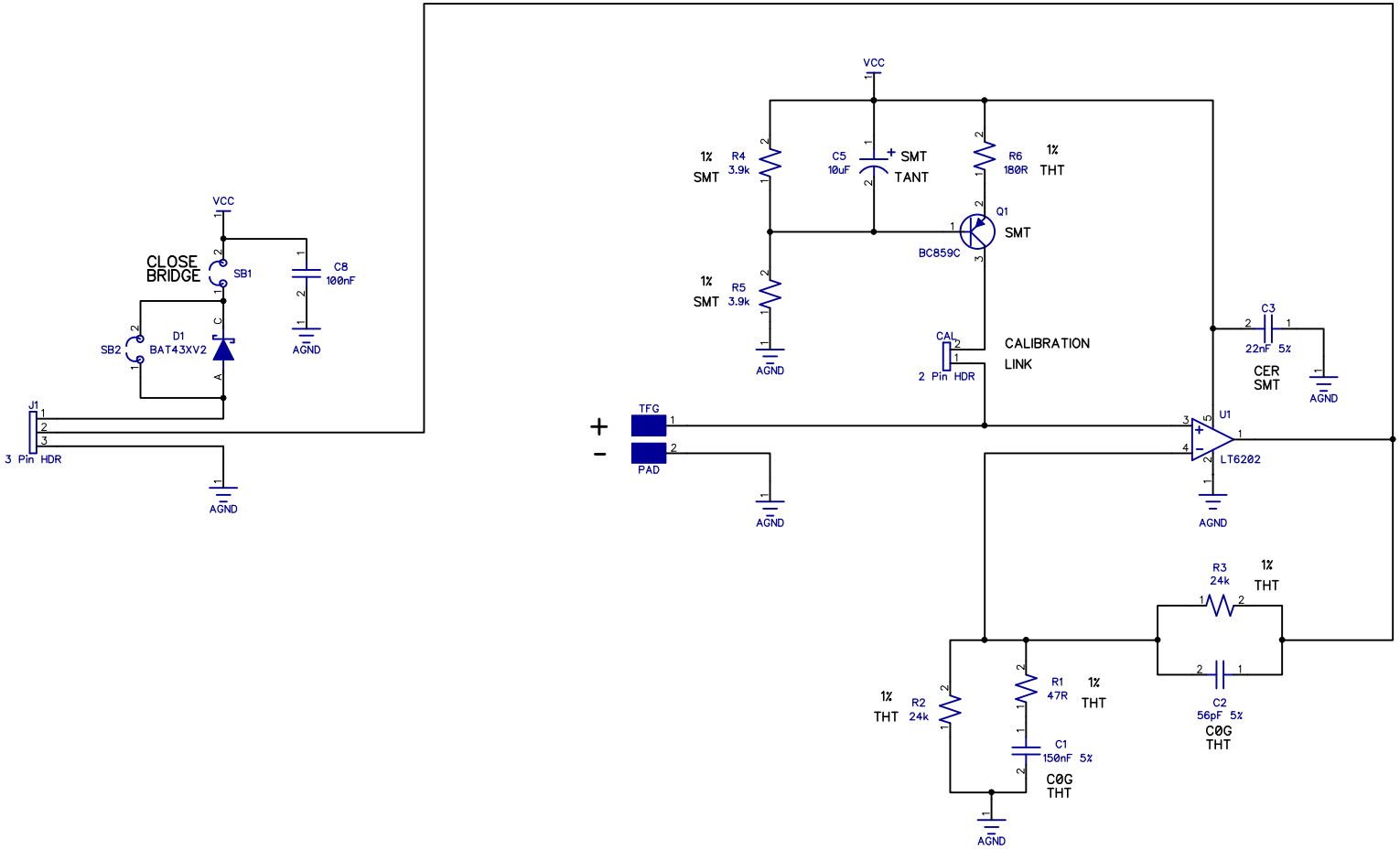


This circuitry allows setting the output reference it shifts the output to avoid saturating the amplifier and clipping the signal waveform

CB Aerospace	Engineer: Igor Dimitrijevic
300 Single Pressure Amp	Project: DAQ - AFE Board
	Date: 24 October 2014
Page: 4 of 6	Revision Number: 1.1

2.6 Thin-film-gauge board, small format

Although the Thin Film Gauge circuits have been designed and boards manufactured, discussion of the details is left to another report.



CB Aerospace	Engineer: Igor Dimitrijevic
Thin Film Gauge AFE HTA V1.5	Project: DAQ - TFG Board Small
	Date: 24 October 2014
Page: 1 of 2	Revision Number: 1.1

2.7 Digital pressure board, standard format

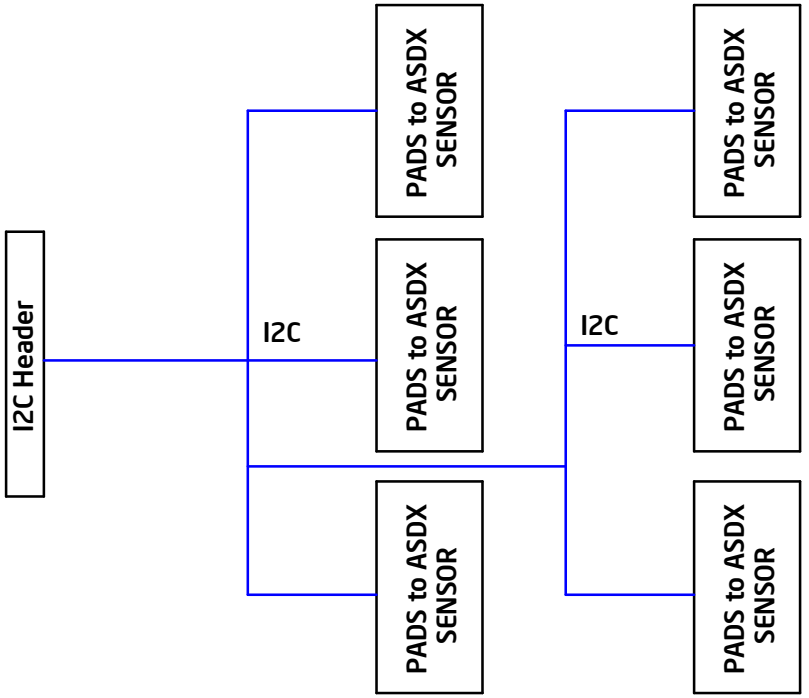
The principal functions of the digital pressure sensor board are to provide power distribution to the sensors [7, 8] and route the digital output signals back to the I2C port of the recording board.

PAGES

Revision History

Rev 1.0	Initial Release

High Level Diagram

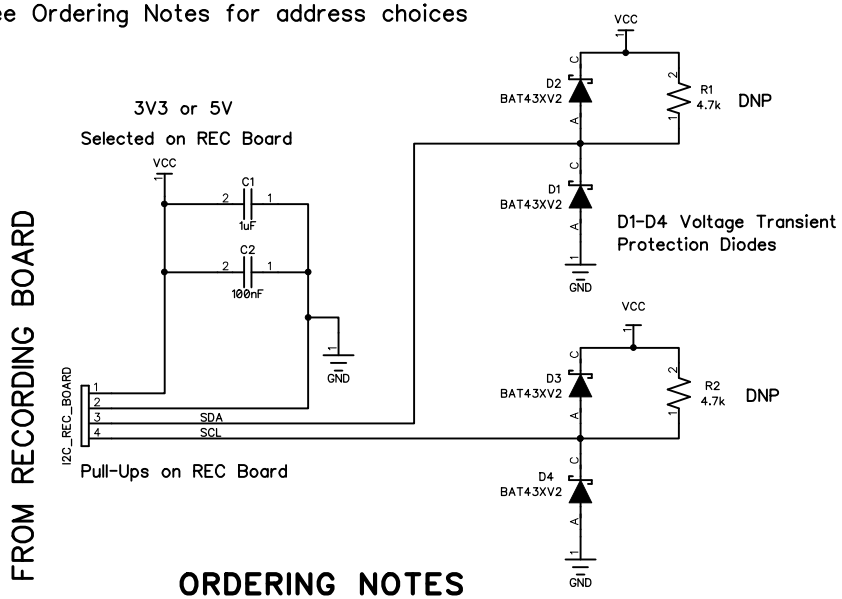


CB Aerospace	Engineer: Igor Dimitrijevic
DIGITAL PRESSURE INDEX	Project: DAQ - Digital Pressure Board
	Date: 22 May 2014
Page: 1 of 4	Revision Number: 1.0

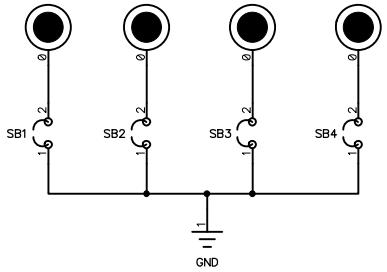
BOARD DESIGNED FOR HONEYWELL ASDX DIGITAL PRESSURE SENSORS

ASDX Digital Sensors come in I2C or SPI options
SPI is not daisy-chainable, hence seperate CS lines for each sensor are required
This is not supported by the Recording Board, hence I2C protocol has been chosen

Honeywell Produces I2C sensors with 6 different addresses
Each sensor on this board must have a unique address
See Ordering Notes for address choices



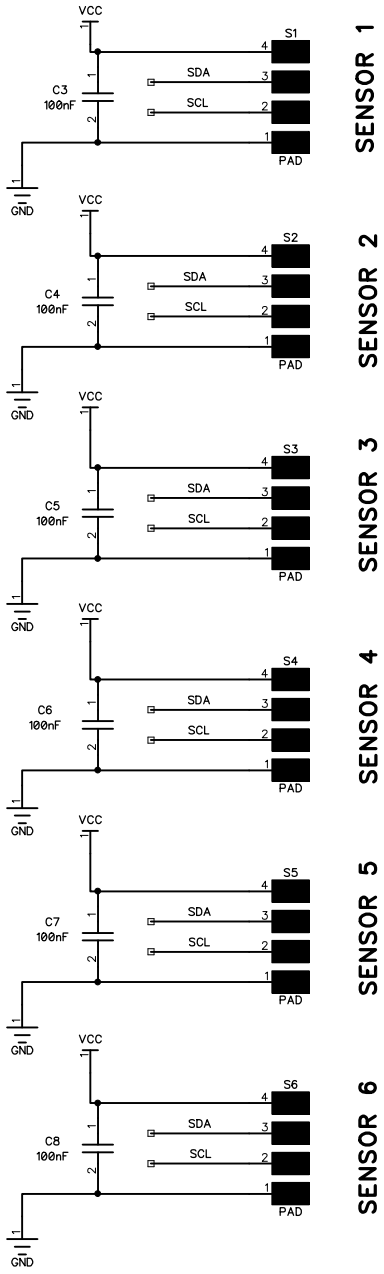
MECHANICAL FEATURES



R1 and R2:

Populate only if stronger pull-up for noise immunity and rise time on I2C bus is required

If populated and pull-ups on Rec Board are also active, pull up resistance on bus becomes 4.7k | 4.7k = 2.35k



ORDERING NOTES

Absolute Sensors supported:

1	2	3	4	5	6	7
ASDX	AC	X	015PA	2	A	3
			030PA	3	B	5
			100PA	4		
			001BA	5		
			002BA	6		
			007BA	7		
			100KA			
			200KA			
			700KA			

1 SERIES ASDX
2 PACKAGE TYPE [AV,RR,AC,RV]
3 FUTURE OPTION
4 PRESSURE RANGE [PSIA,BAR,KPA]
5 I2C ADDRESS
6 CALIBRATION LIMIT [10-90% or 5-95%]
7 SUPPLY VOLTAGE [3V3 or 5V]

NOTE: ALL SENSORS ON THIS BOARD MUST HAVE DIFFERENT ADDRESSES (OPTION 5)
NOTE: SUPPLY VOLTAGE FROM RECORDING BOARD MUST MATCH SENSOR SUPPLY VOLTAGE (OPTION 7)
NOTE: SEE NEXT PAGE FOR FULL SELECTION CHART

CB Aerospace	Engineer: Igor Dimitrijevic
DIGITAL PRESSURE Sensors	Project: DAQ - Digital Pressure Board
	Date: 22 May 2014
Page: 2 of 4	Revision Number: 1.0

ASDX ORDER GUIDE

Package Selection[13] *Calibration Selection*

ASDX **X**

Series

Power Supply Voltage
3 = 3.3 Vdc
5 = 5.0 Vdc

Transfer Function Limits[14]
A = 10% to 90% calibration
B = 5% to 95% calibration

Output Type[15]
S = SPI
2 = I^2C, Address 0x28
3 = I^2C, Address 0x38
4 = I^2C, Address 0x48
5 = I^2C, Address 0x58
6 = I^2C, Address 0x68
7 = I^2C, Address 0x78

Pressure Port

AV = Axial port on top, vented cover on bottom

RR = Radial port on top, radial port on bottom

AC = Axial port, sealed cover (commonly used for absolute)

RV = Radial port, single

Future Option

Pressure Range[16, 17, 18]

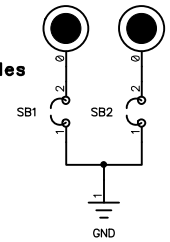
Gage	Differential	Absolute
-	**005ND** = ±5 in H_2O	-
010NG = 10 in H_2O	**010ND** = ±10 in H_2O	-
-	**015CD** = ±15 cm H_2O	-
025CG = 25 cm H_2O	**025CD** = ±25 cm H_2O	-
001PG = 1 psi	**001PD** = ±1 psi	-
005PG = 5 psi	**005PD** = ±5 psi	-
015PG = 15 psi	**015PD** = ±15 psi	**015PA** = 15 psi
030PG = 30 psi	**030PD** = ±30 psi	**030PA** = 30 psi
100PG = 100 psi	-	**100PA** = 100 psi
-	**015MD** = ±15 mbar	-
025MG = 25 mbar	**025MD** = ±25 mbar	-
050MG = 50 mbar	**050MD** = ±50 mbar	-
100MG = 100 mbar	**100MD** = ±100 mbar	-
200MG = 200 mbar	**200MD** = ±200 mbar	-
500MG = 500 mbar	**500MD** = ±500 mbar	-
001BG = 1 bar	**001BD** = ±1 bar	**001BA** = 1 bar
002BG = 2 bar	**002BD** = ±2 bar	**002BA** = 2 bar
007BG = 7 bar	-	**007BA** = 7 bar
003KG = 3 kPa	**003KD** = ±3 kPa	-
004KG = 4 kPa	**004KD** = ±4 kPa	-
005KG = 5 kPa	**005KD** = ±5 kPa	-
010KG = 10 kPa	**010KD** = ±10 kPa	-
020KG = 20 kPa	**020KD** = ±20 kPa	-
050KG = 50 kPa	**050KD** = ±50 kPa	-
100KG = 100 kPa	**100KD** = ±100 kPa	**100KA** = 100 kPa
200KG = 200 kPa	**200KD** = ±200 kPa	**200KA** = 200 kPa
700KG = 700 kPa	-	**700KA** = 700 kPa

2.8 Accelerometer board

The Invensense MPU-6050 sensor [9] includes a 3-axis accelerometer and a three-axis gyroscope. The sensor requires a 3.3 V power supply and output is digital, via the I2C port.

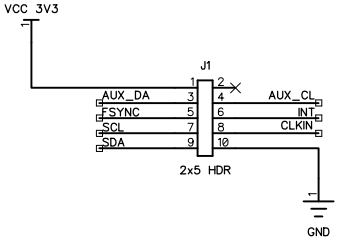
Mounting Holes (2xM3)

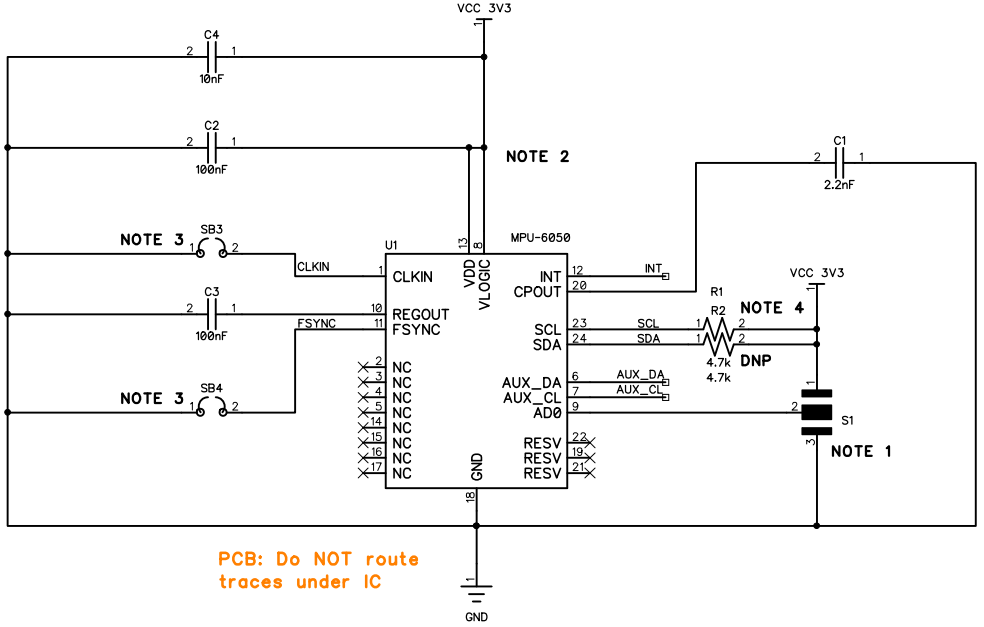
Short SB1, SB2 to ground mounting holes



SERIAL NUMBER

Right Angle Header 2x5 TH





PCB: Do NOT route traces under IC

NOTE 1:

I2C Address Select Via S1 Jumper
AD0 High - Address b1101001
AD0 Low - Address b1101000
ADDRESS MUST BE SET BEFORE USE

NOTE 2:

I/O Voltage set in HW to 3V3

NOTE 3:

If NOT using external CLKIN short SB3 to GND

If NOT using external FSYNC short SB4 to GND

NOTE 4:

Do Not Populate R1, R2
Pull-Ups already present on Recording Board

CB Aerospace	Engineer: Igor Dimitrijevic
Accelerometer Board	Project: DAQ - Acc Board
	Date: 9 FEB 2014
	Revision Number: 1.0

3 Firmware for the prototype recording board

The prototype firmware running within the microcontroller is written in C and communication with the monitoring computer (probably a PC) is via the UART port. The application within the microcontroller is structured around a command-line processor that interprets text-based commands from the PC, takes action and reports text back to the PC. A monitor program, running on the PC, can communicate with the command interpreter and coordinate activities on the microcontroller.

Following hardware initialization, the principal activity of the microcontroller is to listen to the serial port, accumulating incoming characters and saving them into a buffer. On the arrival of a carriage-return character, the string of characters is tested against a limited number of key-words to decide what command to execute. When executing a command, the full attention of the microcontroller is dedicated to that activity and, on completion, the microcontroller resumes the task of listening to the serial port.

There are commands for setting configuration and recording parameters, sampling the analog signals, and reporting the recorded data via the serial port. In this first version of the firmware, the sampling task must see a trigger event within the analog signals in order to complete the recording. The next version will make listening to the serial port a higher priority by running that task via hardware interrupts. It will then be possible to terminate sampling (and recording) via a command. As well, I2C slave functionality will be added so that communication with the monitor PC can be done via the I2C port that shares the same pins as the current UART port. The protocol for the I2C communication will be based on small packets of bytes. At power-up, the type of serial connection will be chosen by looking at the mode selector on the recording board.

The real action is mostly contained in the function `doSampling`, starting at line 186 in the `edaqs_master.c` file. This function will sample and record 2, 4 or 6 analog signals and save the data in static RAM, either internal to the microcontroller or in the SRAM chip connected to the microcontroller via the SPI port. There are also SPI and I2C ports available for communicating with sensors such as accelerometers, and digital-output pressure sensors. Since the number of possible variations of incoming data formats is large, the sampling function will need to be customized for each particular experiment. The next version of the firmware will include an example that uses the MPU-6050 inertial sensor.

3.1 edaqs_master.c

```
/*
 * File:   edaqs_master.c
 * Author: peterj
 *
 * Created on 29 January 2014.
 * Start of the command-line interpreter (mostly ported from daqs-drb).
 */

#ifdef __dsPIC33EP256GP502__
# include <p33Exxxx.h>
#else
# ifdef __dsPIC33FJ128GP802__
#  include <p33Fxxxx.h>
# else
#  error "not a valid MCU selection"
# endif
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Set the relevant bits in the configuration registers.
// Start up using FRC oscillator
_FOSCSEL(FNOSC_FRC)
// Enable clock switching; we have a 7.3728MHz crystal, medium speed.
_FOSC(FCKSM_CSECMD & OSCIOFNC_OFF & POSCMD_XT)
// No watchdog timer for the moment.
_FWDT(FWDTEN_OFF)
// Disable JTAG and have debug pins at PGED2,PGEC2
_FICD(JTAGEN_OFF & ICS_PGD2)

#include "my_delay.h"
#include "osc.h"
#include "my_uart.h"
#include "analog.h"
#include "spi_chips.h"
#include "my_timer.h"

// ------------------------------------------------------------------

#define VERSION_STRING "0.03 08-Feb-2014"
#define PROMPT_STRING "\r\nok> "
// A buffer to contain incoming text commands.
```

```
45 #define TEXT_BUFFER_SIZE 32
46 static char text_buffer[TEXT_BUFFER_SIZE];
47 // Flag to indicate that the getCommandString function should echo characters
48 static char echo_characters = 0;
49 // Sample period in microseconds
50 static unsigned int sample_period_us = 100;
51
52 // The data_blob is an array of bytes on the MCU for accumulating
53 // our recorded data.  We can store data quickly to this blob.
54 // There is also an external 23LC1024 SRAM chip sitting off the SPI port.
55 // It is available is for the slower but longer records -- the default.
56 #ifdef __dsPIC33EP256GP502__
57 // Keep array less than 32kB
58 # define MAX_SAMPLES_ON_MCU 2048
59 #endif
60 #ifdef __dsPIC33FJ128GP802__
61 // Keep array less than 8kB, but don't really know if this is the limit.
62 # define MAX_SAMPLES_ON_MCU 512
63 #endif
64 #define MAX_ANALOG_SIGNALS 6
65 static unsigned int data_blob[MAX_ANALOG_SIGNALS][MAX_SAMPLES_ON_MCU];
66 #define BYTES_IN_SRAM_CHIP 131072L
67 // we pack signal data into sample sets of 16-bytes
68 // This allows room for digital data at a later date.
69 enum memory_options {RAM_INT, RAM_EXT};
70 static enum memory_options storage = RAM_EXT;
71
72 // Mapping from analog signal number to ANx index
73 // Will be used to select the signal to be routed to the sample-hold amp.
74 // The reason for this mapping is that, in order to get 10 channels
75 // on a dsPIC33, we need to go to higher analog inputs, beyong An7 etc.
76 unsigned char anx[] = {0, 1, 2, 3, 4, 5};
77 // We may want to (later) reduce the number of scanned signals.
78 static unsigned int number_of_signals = MAX_ANALOG_SIGNALS;
79
80 // We're going to make some assumptions here, that we'll be using
81 // the full array of signals and packing two 12-bit values into 3 bytes.
82 #define NOMINAL_BYTES_PER_SAMPLE_SET ((MAX_ANALOG_SIGNALS * 3) / 2)
83 static unsigned char bytes_per_sample_set = NOMINAL_BYTES_PER_SAMPLE_SET;
84 #define MAX_SAMPLE_SETS (BYTES_IN_SRAM_CHIP / NOMINAL_BYTES_PER_SAMPLE_SET)
85 static unsigned long total_samples = MAX_SAMPLE_SETS;
86 static unsigned long sample_id = 0; // identifies the oldest sample
87     // which is also the next sample to be written over when recording
88 static unsigned long post_trigger_samples = MAX_SAMPLE_SETS / 2;
89
90 // Trigger event settings
91 enum trigger_options {TRIG_INT, TRIG_EXT};
```

```
92 static enum trigger_options trigger_source = TRIG_INT;
93 static unsigned char trigger_signal = 0; // trigger source
94 enum slope_options {POS, NEG};
95 static enum slope_options trigger_slope = POS;
96 #define MAX_TRIGGER_LEVEL 4095
97 static unsigned int trigger_level = MAX_TRIGGER_LEVEL / 2; // half-way
98
99 // Buffers for use with the external SPI SRAM chips.
100 static unsigned char pageBuf0[32];
101 static unsigned char pageBuf1[32];
102
103 // We'll use the pin to flag certain events such as the start
104 // and stop of analog conversion, writing to SPI-RAM etc.
105 #define LED_PIN LATAbits.LATA4
106
107 #define EVENT_PIN PORTBbits.RB14
108
109 // ------------------------------------------------------------------
110
111 void initHardware(void)
112 // Initialize the MCU peripheral hardware.
113 {
114     RCONbits.SWDTEN = 0;  // Disable the watchdog timer.
115     TRISAbits.TRISA4 = 0; // LED is on RA4. Set to output
116     LATAbits.LATA4 = 0;   // Initially LED is off.
117     TRISBbits.TRISB14 = 1; // !event pin defaults to input
118     CNPUBbits.CNPUB14 = 1; // enable weak pull-up here
119 #   ifdef __dsPIC33EP256GP502__
120       // Analog input
121       ANSELAbits.ANSA0 = 1;
122       ANSELAbits.ANSA1 = 1;
123       ANSELBbits.ANSB0 = 1;
124       ANSELBbits.ANSB1 = 1;
125       ANSELBbits.ANSB2 = 1;
126       ANSELBbits.ANSB3 = 1;
127       // Not analog.
128       ANSELAbits.ANSA4 = 0;
129       ANSELBbits.ANSB8 = 0;
130 #   endif
131 #   ifdef __dsPIC33FJ128GP802__
132       // Analog input
133       AD1PCFGLbits.PCFG0 = 0;
134       AD1PCFGLbits.PCFG1 = 0;
135       AD1PCFGLbits.PCFG2 = 0;
136       AD1PCFGLbits.PCFG3 = 0;
137       AD1PCFGLbits.PCFG4 = 0;
138       AD1PCFGLbits.PCFG5 = 0;
```

```
139       // Not analog input; we use these pins for SPI
140       AD1PCFGLbits.PCFG9 = 1;
141       AD1PCFGLbits.PCFG10 = 1;
142       AD1PCFGLbits.PCFG11 = 1;
143       AD1PCFGLbits.PCFG12 = 1;
144 #   endif
145     TRISBbits.TRISB10 = 0; // SDOx output
146     TRISBbits.TRISB11 = 0; // SCKx output
147     TRISBbits.TRISB12 = 1; // SDIx input
148     TRISBbits.TRISB13 = 0; // chip-select-SRAM output
149     LATBbits.LATB13 = 1;
150     TRISBbits.TRISB7 = 0;  // chip-select-AFE output
151     LATBbits.LATB7 = 1;
152     //
153     // Determine the function of some remappable pins
154     // First, unlock Peripheral Pin Select
155     OSCCON = 0x0046;
156     OSCCON = 0x0057;
157     OSCCONbits.IOLOCK = 0;
158     // Map pins depending on which MCU
159 #   ifdef __dsPIC33EP256GP502__
160       RPINR18bits.U1RXR = 41;      // U1RX <-- RP41(RB9)
161       RPOR3bits.RP40R = 0b000001;  // RP40(RB8) <-- U1TX
162       RPINR22bits.SDI2R = 44;      // SDI2 <-- RPI44(RB12)
163       RPOR4bits.RP42R = 0b001000;  // SDO2 --> RPI42(RB10)
164       RPOR4bits.RP43R = 0b001001;  // SCK2 --> RPI43(RB11)
165 #   else
166 #     ifdef __dsPIC33FJ128GP802__
167         RPINR18bits.U1RXR = 9;      // U1RX <-- RP9(RB9)
168         RPOR4bits.RP8R = 0b00011;   // RP8(RB8) <-- U1TX
169         RPINR20bits.SDI1R = 12;     // SDI1 <-- RP12(RB12)
170         RPOR5bits.RP10R = 0b00111;  // SDO1 --> RP10(RB10)
171         RPOR5bits.RP11R = 0b01000;  // SCK1 --> RP11(RB11)
172 #     else
173 #       error "not a valid MCU selection for this project"
174 #     endif
175 #   endif
176     // Finally, lock PPS
177     OSCCON = 0x0046;
178     OSCCON = 0x0057;
179     OSCCONbits.IOLOCK = 1;
180
181     return;
182 }
183
184 //------------------------------------------------------------
185
```

```
186 void doSampling()
187 // Start up the timer and start sampling,
188 // writing sample sets to the SRAM chips.
189 {
190     unsigned char done = 0;
191     unsigned char triggered = 0;
192     unsigned long post_trigger_count = 0;
193     unsigned int avalue[MAX_ANALOG_SIGNALS];
194     unsigned char i;
195     unsigned long byte_address = 0; // in SRAM chip
196     unsigned char *byte_ptr; // pointer to the first of three bytes of packed data
197     unsigned int v0, v1;
198
199     initTimer2(sample_period_us);
200     sample_id = 0; // start with a clean slate
201     while ( !done ) {
202         LED_PIN = 1; // Start oscilloscope timing signal.
203         // 1. Take a set of samples.
204 #       if 0
205         // Tidy but slow code...
206         for ( i = 0; i < number_of_signals; ++i ) {
207             avalue[i] = readADC(anx[i]);
208         }
209 #       else
210         // Unrolled loop is faster; gets us full conversion speed.
211         // Clear DONE bit, just in case it is left on
212         // from a previous conversion.
213         // If we don't do this, we may drop straight through the
214         // waiting (while) loop and pick up an old result.
215         AD1CON1bits.DONE = 0;
216         // Always do at least An0 and An1.
217         AD1CHS0bits.CH0SA = 0;
218         AD1CON1bits.SAMP = 1;
219         while ( !AD1CON1bits.DONE ) ; // wait
220         avalue[0] = ADC1BUF0;
221         AD1CHS0bits.CH0SA = 1;
222         AD1CON1bits.SAMP = 1;
223         while ( !AD1CON1bits.DONE ) ; // wait
224         avalue[1] = ADC1BUF0;
225         if ( number_of_signals > 2 ) {
226             AD1CHS0bits.CH0SA = 2;
227             AD1CON1bits.SAMP = 1;
228             while ( !AD1CON1bits.DONE ) ; // wait
229             avalue[2] = ADC1BUF0;
230             AD1CHS0bits.CH0SA = 3;
231             AD1CON1bits.SAMP = 1;
232             while ( !AD1CON1bits.DONE ) ; // wait
```

```
233             avalue[3] = ADC1BUF0;
234         }
235         if ( number_of_signals > 4 ) {
236             AD1CHS0bits.CH0SA = 4;
237             AD1CON1bits.SAMP = 1;
238             while ( !AD1CON1bits.DONE ) ; // wait
239             avalue[4] = ADC1BUF0;
240             AD1CHS0bits.CH0SA = 5;
241             AD1CON1bits.SAMP = 1;
242             while ( !AD1CON1bits.DONE ) ; // wait
243             avalue[5] = ADC1BUF0;
244         }
245 #       endif
246         // 2. Test for trigger event.
247         if ( trigger_source == TRIG_INT ) {
248             unsigned int trig_value = avalue[trigger_signal];
249             if ( (trigger_slope == POS && trig_value > trigger_level) ||
250                  (trigger_slope == NEG && trig_value < trigger_level) ) {
251                 triggered = 1;
252             }
253         } else {
254             if ( !(EVENT_PIN) ) triggered = 1;
255         }
256         if ( triggered ) ++post_trigger_count;
257         if ( storage == RAM_EXT ) {
258             // 3. Pack the 12-bit analogue data into 16-bytes.
259             byte_ptr = pageBuf0;
260             for ( i = 0; i < number_of_signals; i += 2 ) {
261                 v0 = avalue[i];
262                 v1 = avalue[i+1];
263                 *byte_ptr++ = (char) v0; // bottom 8 bits
264                 // Second byte gets 4 bits from each value.
265                 *byte_ptr++ = (((char) (v0 >> 8)) & 0x0f) | ((char) (v1 << 4));
266                 *byte_ptr++ = (char) (v1 >> 4); // top 8 bits from original 12
267             }
268             // 4. Send the sample-set to the SRAM chip.
269             // 2014-02-02 Note that we send only just enough data bytes
270             // presuming an even number of channels.
271             byte_address = sample_id * bytes_per_sample_set;
272             writeSRAMdataFast(byte_address, pageBuf0, bytes_per_sample_set);
273         } else {
274             for ( i = 0; i < number_of_signals; i++ ) {
275                 data_blob[i][sample_id] = avalue[i];
276             }
277         }
278         // Increment for the next sample set.
279         // Eventually, we will be overwriting the oldest data.
```

```
280         ++sample_id;
281         if ( sample_id >= total_samples ) sample_id = 0;
282         // 5. We finish the loop if we have collected the required
283         //    number of post-trigger samples.
284         if ( post_trigger_count >= post_trigger_samples ) done = 1;
285         // 6. We sit and do nothing until the next period starts.
286         LED_PIN = 0; // Terminate oscilloscope timing signal.
287         waitTimer2();
288     } // end while
289     stopTimer2();
290     return;
291 } // end doSampling()
292 
293 
294 void reportSamples(void)
295 {
296     unsigned long byte_address = 0; // in SRAM chip
297     unsigned long next_sample;
298     unsigned char *byte_ptr;
299     unsigned char i, b0, b1, b2;
300     unsigned int avalue[MAX_ANALOG_SIGNALS];
301     unsigned int j;
302 
303     // 0. Header line to label columns.
304     printf("\r\n\"Count\",");
305     for ( i = 0; i < number_of_signals; ++i ) {
306         printf("\"A%d\"", i);
307         if ( i < number_of_signals-1 ) printf(",");
308     }
309 
310     next_sample = sample_id; // identify the oldest sample set
311     for ( j = 0; j < total_samples; ++j ) {
312         if ( storage == RAM_EXT ) {
313             // 1. Fetch the next sample set into the local buffer and unpack it.
314             byte_address = next_sample * bytes_per_sample_set;
315             readSRAMdata(SPI_RAM, byte_address, pageBuf0, bytes_per_sample_set);
316             byte_ptr = pageBuf0;
317             for ( i = 0; i < number_of_signals; i += 2 ) {
318                 b0 = *byte_ptr++;
319                 b1 = *byte_ptr++;
320                 b2 = *byte_ptr++;
321                 avalue[i] = ((unsigned int)(b1 & 0x0f) << 8) | b0;
322                 avalue[i+1] = ((unsigned int)(b2) << 4) | ((b1 & 0xf0) >> 4);
323             }
324         } else {
325             for ( i = 0; i < number_of_signals; i++ ) {
326                 avalue[i] = data_blob[i][next_sample];
```

```
            }
        }
        // 3. Report the values as ASCII text in comma-separated-value format.
        printf("\r\n%d,", j);
        for ( i = 0; i < number_of_signals; ++i ) {
            printf("%d", avalue[i]);
            if ( i < number_of_signals-1 ) printf(",");
        }
        // 4. Increment to the next sample set, wrapping around, if necessary.
        ++next_sample;
        if ( next_sample >= total_samples ) next_sample = 0;
    } // end for j

    return;
} // end reportSamples()

//-------------------------------------------------------------

unsigned int max_number_of_sample_sets(void)
{
    if ( storage == RAM_INT ) {
        return MAX_SAMPLES_ON_MCU;
    } else {
        return BYTES_IN_SRAM_CHIP / bytes_per_sample_set;
    }
}

void set_derived_parameters(void)
{
    bytes_per_sample_set = (number_of_signals * 3)/2;
    total_samples = max_number_of_sample_sets();
    post_trigger_samples = total_samples / 2;
    return;
}

double record_period_in_seconds(void)
{
    return sample_period_us * total_samples * 1.0e-6;
}

void interpretCommandString(char *buf)
{
    // Keep ivalue off the stack so we don't need extended pointers
    // for dsPIC33EP which may put the stack above 32kB.
    static unsigned char which_ANx, i;
    static unsigned int avalue, ivalue;

```

```
374     if ( !strncmp(buf,"help",4) || !strncmp(buf,"?",1) ) {
375         putsU1("\r\nAvailable commands are:");
376         putsU1("\r\n    help|?       prints this message");
377         putsU1("\r\n    status       prints version string");
378         putsU1("\r\n    echo on|off  echo characters as they are received");
379         putsU1("\r\n    a0..a5       reports analogue value");
380         putsU1("\r\n    sample       begin sampling (waits for trigger event)");
381         putsU1("\r\n    report       reports the complete set of sampled data");
382         putsU1("\r\n    test spi ram");
383         putsU1("\r\n");
384         putsU1("\r\n    set pll <normal|slow>");
385         putsU1("\r\n    set storage <int|ext>");
386         putsU1("\r\n    set number_of_signals <value>");
387         putsU1("\r\n    set total_samples <value>");
388         putsU1("\r\n    set sample_period_us <value>");
389         putsU1("\r\n    set post_trigger_samples <value>");
390         putsU1("\r\n    set trigger_source <int|ext>");
391         putsU1("\r\n    set trigger_signal <value>");
392         putsU1("\r\n    set trigger_level <value>");
393         putsU1("\r\n    set trigger_slope +|-");
394         putsU1("\r\n");
395         putsU1("\r\nReady.");
396         putsU1("\r\n");
397         return;
398     }
399     if ( !strncmp(buf,"status",6) ) {
400         putsU1("\r\nFirmware version: ");
401         putsU1(VERSION_STRING);
402         putsU1("\r\nSettings:");
403         printf("\r\n    TEXT_BUFFER_SIZE= %u", TEXT_BUFFER_SIZE);
404         printf("\r\n    echo_characters= %u", echo_characters);
405         printf("\r\n    current FCY= %lu", getFCY());
406         printf("\r\n    sample_period_us= %u", sample_period_us);
407         printf("\r\n    period_register_count= %u",
408                compute_period_register_count(sample_period_us));
409         printf("\r\n    number_of_signals= %u", number_of_signals);
410         printf("\r\n    bytes_per_sample_set= %u", bytes_per_sample_set);
411         printf("\r\n    storage= ");
412         if ( storage == RAM_INT ) printf("INT"); else printf("EXT");
413         printf("\r\n    total_samples= %lu", total_samples);
414         printf("\r\n    record_period_in_seconds= %f", record_period_in_seconds());
415         printf("\r\n    sample_id= %lu", sample_id);
416         printf("\r\n    trigger_source= ");
417         if ( trigger_source == TRIG_INT ) printf("INT"); else printf("EXT");
418         printf("\r\n    post_trigger_samples= %lu", post_trigger_samples);
419         printf("\r\n    trigger_signal= %u", trigger_signal);
420         printf("\r\n    trigger_slope= ");
```

```
421         if ( trigger_slope == POS ) printf("POS"); else printf("NEG");
422         printf("\r\n    trigger_level= %u", trigger_level);
423         printf("\r\n");
424         putsU1("\r\nReady.");
425         printf("\r\n");
426         return;
427     }
428     if ( !strncmp(buf,"echo on",7) ) {
429         echo_characters = 1;
430         putsU1("\r\nEcho characters is on.");
431         return;
432     }
433     if ( !strncmp(buf,"set pll slow",12) ) {
434         setPLLslow();
435         initUART1(DEFAULT_BAUD_RATE);
436         printf("\r\ncurrent FCY= %lu", getFCY());
437         return;
438     }
439     if ( !strncmp(buf,"set pll normal",14) ) {
440         setPLLnormal();
441         initUART1(DEFAULT_BAUD_RATE);
442         printf("\r\ncurrent FCY= %lu", getFCY());
443         return;
444     }
445     if ( !strncmp(buf,"set storage int",15) ) {
446         storage = RAM_INT;
447         putsU1("\r\nStore data in MCU internal RAM.");
448         set_derived_parameters();
449         printf("\r\ntotal_samples= %lu", total_samples);
450         printf("\r\npost_trigger_samples= %lu", post_trigger_samples);
451         return;
452     }
453     if ( !strncmp(buf,"set storage ext",15) ) {
454         storage = RAM_EXT;
455         putsU1("\r\nStore data in SRAM chip external to MCU.");
456         set_derived_parameters();
457         printf("\r\ntotal_samples= %lu", total_samples);
458         printf("\r\npost_trigger_samples= %lu", post_trigger_samples);
459         return;
460     }
461     if ( !strncmp(buf,"sample",6) ) {
462         putsU1("\r\nBegin sampling...");
463         doSampling();
464         putsU1("\r\nDone.");
465         return;
466     }
467     if ( !strncmp(buf,"report",6) ) {
```

```
468         reportSamples();
469         return;
470     }
471     if ( buf[0] == 'a' || buf[0] == 'A' ) {
472         which_ANx = (unsigned char) (buf[1] - '0');
473         if ( which_ANx > 5 ) which_ANx = 5;
474         avalue = readADC(anx[which_ANx]);
475         printf("\r\nA%d=%d", which_ANx, avalue);
476         return;
477     }
478     if ( !strncmp(buf,"test spi ram",12) ) {
479         strcpy((char *)pageBuf0, (const char *)"aaaaaaaaaaaaaaaaaaaa");
480         writeSRAMdata(SPI_RAM, 0x0000, pageBuf0, 32);
481         readSRAMdata(SPI_RAM, 0x0000, pageBuf1, 32);
482         for ( i = 0; i < 32; ++i ) {
483             printf("\r\n%d 0x%x 0x%x", i, pageBuf0[i], pageBuf1[i]);
484         }
485         return;
486     }
487     if ( !strncmp(buf,"set number_of_signals",21) ) {
488         sscanf(&(buf[21]), "%u", &number_of_signals);
489         // Want an even number because of the way we pack data.
490         if ( number_of_signals % 2 ) number_of_signals += 1;
491         number_of_signals = min(max(2,number_of_signals),MAX_ANALOG_SIGNALS);
492         set_derived_parameters();
493         printf("\r\nnumber_of_signals= %u", number_of_signals);
494         printf("\r\ntotal_samples= %lu", total_samples);
495         printf("\r\npost_trigger_samples= %lu", post_trigger_samples);
496         return;
497     }
498     if ( !strncmp(buf,"set total_samples",17) ) {
499         sscanf(&(buf[17]), "%lu", &total_samples);
500         total_samples = max(128,total_samples);
501         total_samples = min(total_samples, max_number_of_sample_sets());
502         post_trigger_samples = min(post_trigger_samples,total_samples/2);
503         printf("\r\ntotal_samples= %lu", total_samples);
504         printf("\r\npost_trigger_samples= %lu", post_trigger_samples);
505         printf("\r\nrecord_period= %f seconds", record_period_in_seconds());
506         return;
507     }
508     if ( !strncmp(buf,"set sample_period_us",20) ) {
509         sscanf(&(buf[20]), "%u", &sample_period_us);
510         printf("\r\nsample_period_us= %u", sample_period_us);
511         printf("\r\nrecord_period= %f seconds", record_period_in_seconds());
512         return;
513     }
514     if ( !strncmp(buf,"set post_trigger_samples",24) ) {
```

```
515         sscanf(&(buf[24]), "%lu", &post_trigger_samples);
516         if ( post_trigger_samples > total_samples )
517             post_trigger_samples = total_samples;
518         printf("\r\npost_trigger_samples= %lu", post_trigger_samples);
519         return;
520     }
521     if ( !strncmp(buf,"set trigger_source int",22) ) {
522         storage = TRIG_INT;
523         putsU1("\r\nTrigger source is one of the analog channels.");
524         return;
525     }
526     if ( !strncmp(buf,"set trigger_source ext",22) ) {
527         storage = TRIG_INT;
528         putsU1("\r\nTrigger source is the !event pin on the MCU.");
529         return;
530     }
531     if ( !strncmp(buf,"set trigger_signal",18) ) {
532         sscanf(&(buf[18]), "%u", &ivalue);
533         if ( ivalue < 0 ) ivalue = 0;
534         if ( ivalue >= number_of_signals ) ivalue = number_of_signals - 1;
535         trigger_signal = (char)ivalue;
536         printf("\r\ntrigger_signal= %u", trigger_signal);
537         return;
538     }
539     if ( !strncmp(buf,"set trigger_level",17) ) {
540         sscanf(&(buf[17]), "%u", &trigger_level);
541         if ( trigger_level > 4095 ) trigger_level = 4095;
542         printf("\r\ntrigger_level= %u", trigger_level);
543         return;
544     }
545     if ( !strncmp(buf,"set trigger_slope",17) ) {
546         if ( buf[18] == '-' ) {
547             trigger_slope = NEG;
548             printf("\r\ntrigger_slope= NEG");
549         } else {
550             // default is positive slope
551             trigger_slope = POS;
552             printf("\r\ntrigger_slope= POS");
553         }
554         return;
555     }
556     putsU1("\r\nUnknown command: ");
557     putsU1(buf);
558     return;
559 } // end interpretCommandString()
560
561
```

```
562 int main(int argc, char** argv)
563 {
564     initHardware();
565     initPLL();
566     initUART1(DEFAULT_BAUD_RATE);
567     initADC();
568     initSPIx();
569     my_delay(255);
570     printf("\r\nedaqs-master: %s", VERSION_STRING);
571     printf("\r\nPeter Jacobs");
572     printf("\r\nSchool of Engineering, Uni of Qld");
573     printf("\r\n"); printResetCause();
574     printf("\r\nPut SPI SRAM chip into sequential mode.");
575     unsigned char register_value = setSRAMmode(SPI_RAM, SRAM_SEQUENTIAL_MODE);
576     printf("\r\nmode register=0x%x", register_value);
577     if (register_value == SRAM_SEQUENTIAL_MODE) printf(" OK"); else printf(" Bad");
578     putsU1("\r\n");
579     if ( echo_characters ) putsU1(PROMPT_STRING);
580     while ( 1 ) {
581         LED_PIN ^= 1; // toggle LED
582         getsnU1(text_buffer, TEXT_BUFFER_SIZE, echo_characters);
583         interpretCommandString(text_buffer);
584         if ( echo_characters ) putsU1(PROMPT_STRING);
585     }
586     return(EXIT_SUCCESS);
587 }
```

3.2 osc.h, osc.c

```
1  // osc.h
2  // Oscillator frequency, assuming a 4xPLL.
3
4  #ifndef OSC_H
5  #define OSC_H
6
7  #define FCRYSTAL 7372800L
8  // PLLmultiple    FCY(Hz)   TCY(ns)
9  // 1(i.e. no PLL) 3686400    271.3
10 // 2              7372800    135.6
11 // 4             14745600     67.82
```

```
12
13 void initPLL();
14 void setPLLnormal();
15 void setPLLslow();
16 unsigned long getFCY();
17
18 #endif
```

```
1  // osc.h
2  // Clock setting functions.
3  // PJ, 08-Feb-2014
4
5  #ifdef __dsPIC33EP256GP502__
6  # include <p33Exxxx.h>
7  #else
8  # ifdef __dsPIC33FJ128GP802__
9  #  include <p33Fxxxx.h>
10 # else
11 #  error "not a valid MCU selection"
12 # endif
13 #endif
14
15 #include "osc.h"
16 static char PLL_multiplier = 2;
17
18 void initPLL()
19 {
20     // Configure PLL to multiply FIN by 4
21     // This gives FOSC=29.49MHz from FIN=7.3728MHz
22     PLLFBD = 30;
23     CLKDIVbits.PLLPOST = 1;
24     CLKDIVbits.PLLPRE = 0;
25     PLL_multiplier = 4;
26     // New clock source will be primary oscillator with PLL.
27     __builtin_write_OSCCONH(0b011);
28     // Now, we want to make the switch without upsetting other settings.
29     __builtin_write_OSCCONL(OSCCON | 0x01);
30     // Wait for Clock switch to occur
31     while (OSCCONbits.COSC != 0b011);
32     // Wait for PLL to lock
33     while(OSCCONbits.LOCK!=1);
34
35     return;
36 }
37
```

```
38 void setPLLnormal()
39 {
40     CLKDIVbits.PLLPOST = 1;
41     PLL_multiplier = 4;
42 }
43
44 void setPLLslow()
45 {
46     CLKDIVbits.PLLPOST = 3;
47     PLL_multiplier = 2;
48 }
49
50 unsigned long getFCY()
51 {
52     return FCRYSTAL / 2 * PLL_multiplier;
53 }
```

3.3 my_uart.h, my_uart.c

```
1  // my_uart.h
2
3  #ifndef MY_UART_H
4  #define MY_UART_H
5
6  // A value of 115200L works everywhere, but is oh-so-slow.
7  // We have successfully run at 921600L with minicom, however,
8  // Tcl/Tk did not successfully work at that speed.
9  // Looking into the source code of tclUnixChan, only baud rates
10 // up to 460800 are listed; let's go with that.
11 #define DEFAULT_BAUD_RATE 460800L
12
13 // Various special characters that we might encounter.
14 #define ACK 0x01
15 #define BS 0x08
16 #define CR 0x0D
17 #define LF 0x0A
18 #define EOT 0x04
19 #define ETX 0x03
20
21 void initUART1(unsigned long baud_rate);
```

```
22 char putcU1(char c);
23 char getcU1(void);
24 unsigned int putsU1(char *s);
25 char * getsnU1(char *buf, unsigned int n, unsigned char with_echo);
26 int write(int handle, void *buffer, unsigned int len);
27 void printResetCause(void);
28
29 #endif
```

```
1  // my_uart.c
2  // Functions adapted from the Microchip datasheet information.
3  // PJ
4  // 20-Feb-2009
5
6  #ifdef __dsPIC33EP256GP502__
7  # include <p33Exxxx.h>
8  #else
9  # ifdef __dsPIC33FJ128GP802__
10 #  include <p33Fxxxx.h>
11 # else
12 #  error "not a valid MCU selection"
13 # endif
14 #endif
15
16 #include <stdlib.h>
17 #include "osc.h"
18 #include "my_uart.h"
19
20
21 void initUART1(unsigned long baud_rate)
22 // Initialize the serial port hardware, assuming
23 // low-speed range, no parity, 8 data bits, 1 stop bit.
24 // It is also assumed that appropriate connections to
25 // external pins are made elsewhere.
26 {
27     const unsigned int S = 16;
28     U1BRG = getFCY()/S/baud_rate - 1;
29     U1MODEbits.UARTEN = 1;
30     U1STAbits.UTXEN = 1; // U1TX pin will be controlled by UART1
31     U1STAbits.OERR = 0; // clear error flags
32     U1STAbits.FERR = 0;
33     U1STAbits.PERR = 0;
34     return;
35 }
36
```

```
37 char putcU1(char c)
38 // Put a character out through the serial port.
39 {
40     while ( U1STAbits.UTXBF ) /* wait while buffer is full */ ;
41     U1TXREG = c;
42     return c;
43 }
44
45 char getcU1(void)
46 // Get the next character from the serial port.
47 // Note that this call is blocking.  It will wait for the character.
48 {
49     if ( U1STAbits.FERR ) /* continue */ ;
50     if ( U1STAbits.OERR ) /* continue */ ;
51     while ( !U1STAbits.URXDA ) /* wait for a new character */ ;
52     return U1RXREG;
53 }
54
55 unsigned int putsU1(char *s)
56 // Put the string out through the serial port.
57 {
58     unsigned int count = 0;
59     while ( *s != '\0' ) {
60         putcU1( *s );
61         ++s;
62         ++count;
63     }
64     return count;
65 }
66
67 char * getsnU1(char *buf, unsigned int n, unsigned char with_echo)
68 // Get a text string from the serial port up to a maximum of n characters
69 // (including the terminating null character)
70 // or up until a carriage-return '\r' is encountered.
71 // A null-terminated string is returned.
72 {
73     char *p = buf; // We are going to add characters at p.
74     int length_remaining = n-1;
75     while ( length_remaining > 0 ) {
76         char c = getcU1();
77         if ( c == '\r' ) break; // Don't include the return character.
78         if ( c == '\n' ) continue; // Discard new line characters.
79         if ( (c == BS) && (p > buf) ) {
80             // Remove the previous character.
81             p--; length_remaining++;
82             if ( with_echo ) { putcU1(BS); putcU1(' '); putcU1(BS); }
83             continue;
```

```
84          }
85          if ( c == BS ) continue; // Discard excess backspace characters.
86          // Add the character to the buffer.
87          *p = c;
88          p++; length_remaining--;
89          if ( with_echo ) { putcU1(c); }
90      }
91      *p = '\0';
92      return buf;
93  }
94
95  int write(int handle, void *buffer, unsigned int len)
96  // Service function for stdio library.
97  // For the moment, put everything through UART1.
98  {
99      int i;
100     char *cp;
101
102     switch ( handle ) {
103         default:
104         case 0:  // stdin
105         case 1:  // stdout
106         case 2:  // stderr
107             for ( i = len; i; --i ) {
108                 cp = (char *)buffer++;
109                 putcU1( *cp );
110             }
111             break;
112     } // end switch
113     return len;
114 }
115
116 void printResetCause(void)
117 // Derived from Figure 8.18 of Reese et al. (page 279)
118 {
119     if ( RCONbits.SLEEP ){
120         putsU1("Device has been in sleep mode.");
121         RCONbits.SLEEP = 0;
122     }
123     if ( RCONbits.POR ) {
124         putsU1("Power-on reset.");
125         RCONbits.POR = 0;
126         RCONbits.BOR = 0;
127     } else {
128         // non-POR causes
129         if ( RCONbits.SWR ) {
130             putsU1("Software reset.");
```

```
131             RCONbits.SWR = 0;
132         }
133         if ( RCONbits.WDTO ) {
134             putsU1("Watchdog timeout reset.");
135             RCONbits.WDTO = 0;
136         }
137         if ( RCONbits.EXTR ) {
138             putsU1("Master-clear-pin reset.");
139             RCONbits.EXTR = 0;
140         }
141         if ( RCONbits.BOR ) {
142             putsU1("Brown-out reset.");
143             RCONbits.BOR = 0;
144         }
145         if ( RCONbits.TRAPR ) {
146             putsU1("Trap conflict reset.");
147             RCONbits.TRAPR = 0;
148         }
149         if ( RCONbits.IOPUWR ) {
150             putsU1("Illegal condition device reset.");
151             RCONbits.IOPUWR = 0;
152         }
153         if ( RCONbits.CM ) {
154             putsU1("Configuration mismatch reset.");
155             RCONbits.CM = 0;
156         }
157         return;
158     }
159     return;
160 }
```

3.4 my_delay.h, my_delay.c

```
1 // my_delay.h
2
3 #ifndef MY_DELAY_H
4 #define MY_DELAY_H
5
6 int my_delay(int count);
7
8 #endif
```

```
// my_delay.c
// Simple software delay loop.

int my_delay(int count)
{
    unsigned int i, j, k;
    k = 0;
    for ( i = 0; i < count; ++i ) {
        k = 0;
        for ( j = 0; j < 0xff; ++j ) {
            k += 1;
        }
    }
    return k;
} // end my_delay()
```

3.5 my_timer.h, my_timer.c

```
// my_timer.h
// PJ, 23-Feb-2009 from daqs-drb project

#ifndef MY_TIMER_H
#define MY_TIMER_H

unsigned int compute_period_register_count(unsigned int period_us);
void initTimer2(unsigned int period_us);
void startTimer2();
void stopTimer2();
void waitTimer2();

#endif
```

```
// my_timer.c
// We'll use timer 2 to provide a regular tick.
```

```
3
4  #ifdef __dsPIC33EP256GP502__
5  # include <p33Exxxx.h>
6  #else
7  # ifdef __dsPIC33FJ128GP802__
8  #  include <p33Fxxxx.h>
9  # else
10 #  error "not a valid MCU selection"
11 # endif
12 #endif
13
14 #include "osc.h"
15
16 unsigned int compute_period_register_count(unsigned int period_us)
17 {
18     // The following arrangement is an attempt to avoid overflow
19     // of the intermediate result and premature truncation.
20     // We also assume. PRE=8.
21     return (int)(((long)period_us * (getFCY()/800L))/10000L) - 1;
22 }
23
24 void initTimer2(unsigned int period_us)
25 // Get Timer2 ticking over as a 16-bit timer.
26 // Period is in microseconds.
27 {
28     T2CONbits.T32 = 0;  // 16-bit
29     T2CONbits.TCS = 0;  // MCU clock as source
30     T2CONbits.TGATE = 0;
31     T2CONbits.TCKPS = 0b01; // prescale 8
32     TMR2 = 0;
33     // Clear the interrupt flag.
34     // Even though we are not using interrupts, this flag
35     // is set by the hardware when TMR2 reaches PR2.
36     IFS0bits.T2IF = 0;
37     PR2  = compute_period_register_count(period_us);
38     T2CONbits.TON = 1; // enable timer
39     return;
40 } // end initTimer2()
41
42 void startTimer2()
43 {
44     TMR2 = 0;
45     IFS0bits.T2IF = 0;
46     T2CONbits.TON = 1;
47     return;
48 }
49
```

```
50 void stopTimer2()
51 {
52     T2CONbits.TON = 0;
53     TMR2 = 0;
54     IFS0bits.T2IF = 0;
55     return;
56 }
57
58 void waitTimer2()
59 {
60     while ( !IFS0bits.T2IF ) /* wait */ ;
61     IFS0bits.T2IF = 0; // reset the interrupt flag
62     return;
63 }
```

3.6 spi_chips.h, spi_chips.c

```
1  // spi_chips.h
2
3  #ifndef SPI_CHIPS_H
4  #define SPI_CHIPS_H
5
6  // Mode register values for 23LC1024
7  #define SRAM_BYTE_MODE 0b00000000
8  #define SRAM_PAGE_MODE 0b10000000
9  #define SRAM_SEQUENTIAL_MODE 0b01000000
10
11 enum spi_chips_t {SPI_RAM, SPI_AFE};
12
13 void selectSPIchip(enum spi_chips_t whichChip);
14 void releaseSPIchip(enum spi_chips_t whichChip);
15 void initSPIx(void);
16 unsigned char putcSPIx(unsigned char cout);
17 unsigned char setSRAMmode(enum spi_chips_t which_chip, unsigned char mode);
18 unsigned char writeSRAMdata(enum spi_chips_t which_chip, unsigned long addr,
19                             unsigned char *bufout, unsigned int n);
20 unsigned char writeSRAMdataFast(unsigned long addr,
21                                 unsigned char *bufout, unsigned int n);
22 unsigned char readSRAMdata(enum spi_chips_t which_chip, unsigned long addr,
23                            unsigned char *bufin, unsigned int n);
```

```
24
25 #endif
```

```
1  // spi_chips.c
2  // Service functions for the 23LC1024 SRAM chip in SPI mode.
3  // PJ, 01-Feb-2014 adapted from the daqs-drb project
4
5  #ifdef __dsPIC33EP256GP502__
6  # include <p33Exxxx.h>
7  # define SPIxCON1bits SPI2CON1bits
8  # define SPIxSTATbits SPI2STATbits
9  # define SPIxIF SPI2IF
10 # define SPIxBUF SPI2BUF
11 # define IFSxbits IFS2bits
12 #else
13 # ifdef __dsPIC33FJ128GP802__
14 #  include <p33Fxxxx.h>
15 #  define SPIxCON1bits SPI1CON1bits
16 #  define SPIxSTATbits SPI1STATbits
17 #  define SPIxIF SPI1IF
18 #  define SPIxBUF SPI1BUF
19 #  define IFSxbits IFS0bits
20 # else
21 #  error "not a valid MCU selection"
22 # endif
23 #endif
24
25 #include "osc.h"
26 #include "spi_chips.h"
27
28 #define CS_SRAM (LATBbits.LATB13)
29 #define CS_AFE  (LATBbits.LATB7)
30
31 void selectSPIchip(enum spi_chips_t whichChip)
32 {
33     switch ( whichChip ) {
34         case SPI_RAM:
35             CS_SRAM = 0; // Active-low selects chip.
36             CS_AFE = 1;  // Must not have more than one selected.
37             break;
38         case SPI_AFE:
39             CS_AFE = 0;
40             CS_SRAM = 1;
41             break;
42         default:
```

```
43              CS_SRAM = 1;
44              CS_AFE = 1;
45          }
46      return;
47  } // end selectSPIChip()
48
49  void releaseSPIchip(enum spi_chips_t whichChip)
50  {
51      switch ( whichChip ) {
52          case SPI_RAM:
53              CS_SRAM = 1;
54              break;
55          case SPI_AFE:
56              CS_AFE = 1;
57              break;
58          default:
59              CS_SRAM = 1;
60              CS_AFE = 1;
61          }
62      return;
63  } // end releaseSPIChip()
64
65  void initSPIx(void)
66  {
67      // Assume that the IO-pins have been already assigned
68      // back in initHardware().
69      SPIxCON1bits.MSTEN = 1; // Master mode
70      SPIxSTATbits.SPIROV = 0; // clear receive overflow flag
71      //
72      // Primary * Secondary prescale to give clock freq of about 7.4MHz
73      SPIxCON1bits.PPRE = 0b11; // 1:1 primary prescale
74      SPIxCON1bits.SPRE = 0b110; // 2:1 secondary prescale
75      //
76      // The 23K256 data sheet shows the SPI clock with positive polarity
77      // having the clock pulses starting with a rising edge.
78      // It is also stated that the SRAM chip puts its data out on the
79      // falling clock edge (which is half-way through a cycle).
80      // Hence we should wait half an SPI clock cycle before sampling
81      // the incoming data.
82      SPIxCON1bits.SMP = 1; // input data sampled at end of data output time
83      SPIxCON1bits.CKE = 1; // serial output data changes on transition
84                            // from active to idle clock state
85      SPIxCON1bits.CKP = 0; // Clock polarity, idle low, active high
86      // Accept power-on defaults for other bits.
87      IFSxbits.SPIxIF = 0; // clear interrupt flag
88      SPIxSTATbits.SPIEN = 1; // turn it on
89      return;
```

```
90 } // end initSPIx()
91
92 // 8-bit SPI SRAM commands
93 #define SRAM_READ 0b00000011
94 #define SRAM_WRITE 0b00000010
95 // RDSR: Read Status Register
96 #define SRAM_RDSR 0b00000101
97 // WRSR: Write Status Register
98 #define SRAM_WRSR 0b00000001
99
100 unsigned char putcSPIx(unsigned char cout)
101 {
102     while ( SPIxSTATbits.SPITBF ) /* wait for transmit buffer to empty */ ;
103     SPIxBUF = cout;
104     while ( !SPIxSTATbits.SPIRBF ) /* wait for transmission to finish */ ;
105     unsigned char cin = (unsigned char)(SPIxBUF & 0xFF);
106     IFSxbits.SPIxIF = 0; // clear interrupt bit; we don't use it
107     return cin;
108 } // end putcSPIx()
109
110 unsigned char setSRAMmode(enum spi_chips_t which_chip, unsigned char mode)
111 {
112     unsigned char register_value, dummy;
113     selectSPIchip(which_chip);
114     dummy = putcSPIx(SRAM_WRSR);
115     dummy = putcSPIx(mode);
116     releaseSPIchip(which_chip);
117     Nop();
118     selectSPIchip(which_chip);
119     dummy = putcSPIx(SRAM_RDSR);
120     register_value = putcSPIx(0xff); // 0x00 or 0xff, don't care
121     releaseSPIchip(which_chip);
122     return register_value;
123 } // end setSRAMMode()
124
125 unsigned char writeSRAMdata(enum spi_chips_t which_chip, unsigned long addr,
126                             unsigned char *bufout, unsigned int n)
127 {
128     unsigned char dummy;
129     unsigned int i;
130     selectSPIchip(which_chip);
131     dummy = putcSPIx(SRAM_WRITE);
132     // 23LC1024 expects a 24-bit address (the first 7 bits are don't care)
133     dummy = putcSPIx((unsigned char)(addr >> 16));
134     dummy = putcSPIx((unsigned char)(addr >> 8));
135     dummy = putcSPIx((unsigned char)addr);
136     for ( i = 0; i < n; ++i ) {
```

```
137         dummy = putcSPIx(*bufout++);
138     }
139     releaseSPIchip(which_chip);
140     return 0;
141 } // end writeSRAMdata()
142 
143 unsigned char writeSRAMdataFast(unsigned long addr,
144                                 unsigned char *bufout, unsigned int n)
145 {
146     unsigned char dummy;
147     unsigned int i;
148     CS_SRAM = 0;
149     while ( SPIxSTATbits.SPITBF );
150     SPIxBUF = SRAM_WRITE;
151     while ( !SPIxSTATbits.SPIRBF );
152     dummy = SPIxBUF;
153     // 23LC1024 expects a 24-bit address (the first 7 bits are don't care)
154     while ( SPIxSTATbits.SPITBF );
155     SPIxBUF = (unsigned char)(addr >> 16);
156     while ( !SPIxSTATbits.SPIRBF );
157     dummy = SPIxBUF;
158     while ( SPIxSTATbits.SPITBF );
159     SPIxBUF = (unsigned char)(addr >> 8);
160     while ( !SPIxSTATbits.SPIRBF );
161     dummy = SPIxBUF;
162     while ( SPIxSTATbits.SPITBF );
163     SPIxBUF = (unsigned char)addr;
164     while ( !SPIxSTATbits.SPIRBF );
165     dummy = SPIxBUF;
166     for ( i = 0; i < n; ++i ) {
167         while ( SPIxSTATbits.SPITBF );
168         SPIxBUF = *bufout++;
169         while ( !SPIxSTATbits.SPIRBF );
170         dummy = SPIxBUF;
171     }
172     IFSxbits.SPIxIF = 0;
173     CS_SRAM = 0;
174     return 0;
175 } // end writeSRAMdataFast()
176 
177 unsigned char readSRAMdata(enum spi_chips_t which_chip, unsigned long addr,
178                            unsigned char *bufin, unsigned int n)
179 {
180     unsigned char dummy;
181     unsigned int i;
182     selectSPIchip(which_chip);
183     dummy = putcSPIx(SRAM_READ);
```

```
184     dummy = putcSPIx((unsigned char)(addr >> 16));
185     dummy = putcSPIx((unsigned char)(addr >> 8));
186     dummy = putcSPIx((unsigned char)addr);
187     for ( i = 0; i < n; ++i ) {
188         *bufin++ = putcSPIx(0xff);
189     }
190     releaseSPIchip(which_chip);
191     return 0;
192 } // end readPage()
```

3.7 analog.h, analog.c

```
1 // analog.h
2
3 void initADC(void);
4 unsigned int readADC(unsigned char which_ANx);
```

```
1  // analog.c
2  // Set up and read the ADC channels on DRB's DAQS board.
3  // PJ, 22-Feb-2009
4  // Adapted from PIC24 experiments.
5
6  #ifdef __dsPIC33EP256GP502__
7  # include <p33Exxxx.h>
8  #else
9  # ifdef __dsPIC33FJ128GP802__
10 #  include <p33Fxxxx.h>
11 # else
12 #  error "not a valid MCU selection"
13 # endif
14 #endif
15
16 #include "osc.h"
17
18 void initADC(void)
19 {
20     AD1CON1bits.ADON = 0; // turn off for config
```

```
21      // 0. 12-bit converter
22      AD1CON1bits.AD12B = 1;
23      // 1. Specify port pins as analog input
24      //    (was done in initHardware())
25      // 2. Select voltage references
26      //    AVss and AVdd are used as Vref- and Vref+ respectively
27      AD1CON2bits.VCFG = 0b000;
28      // 3. Conversion clock
29      AD1CON3bits.ADRC = 0; // clock derived from system clock
30      // AD1CON3bits.ADRC = 1; // use internal RC clock
31      AD1CON3bits.ADCS = 1; // Tad = 2 * Tcy = 135ns (> 118ns minumum)
32      AD1CON3bits.SAMC = 3; // Tsample, >=3 Tad (minimum)
33      // 4. Sample and hold channel
34      // We will be later changing CH0SA when selecting signals.
35      AD1CON2bits.CHPS = 0b00; // converts CH0 only
36      AD1CHS0bits.CH0NA = 0; // Channel 0 negative input, sample A
37      AD1CHS0bits.CH0SA = 0; // Channel 0 positive input
38      AD1CHS0bits.CH0NB = 0; // Channel 0 negative input, sample B
39      AD1CHS0bits.CH0SB = 0; // Channel 0 positive input
40      AD1CON2bits.ALTS = 0; // always use inputs for sample A
41      // 5. Appropriate sample/conversion sequence.
42      AD1CON1bits.SSRC = 0b111; // automatic conversion to start after sampling
43      AD1CON1bits.ASAM = 0; // sampling starts when SAMP bit set
44      AD1CSSL = 0; // no scanning required
45      // 6. Presentation of conversion results
46      AD1CON1bits.FORM = 0b00; // present as unsigned int
47      // 7. Turn on the ADC module.
48      AD1CON1bits.ADON = 1;
49      return;
50  } // end initADC()
51
52  unsigned int readADC( unsigned char which_ANx )
53  // Select and sample a particular analog pin.
54  {
55      AD1CHS0bits.CH0SA = which_ANx; // Channel 0 positive input
56      AD1CHS0bits.CH0SB = which_ANx; // Channel 0 positive input
57      // Clear DONE bit, just in case it is left on
58      // from a previous conversion.
59      // If we don't do this, we may drop straight through the
60      // waiting (while) loop and pick up an old result.
61      AD1CON1bits.DONE = 0;
62      AD1CON1bits.SAMP = 1; // start sampling
63      while ( !AD1CON1bits.DONE ) ; // wait for conversion
64      return ADC1BUF0;
65  } // end readADC()
```

4 Monitor Program

```
#! /usr/bin/wish
# monitor-edaqs.tcl
#
# Need to run with sufficient privilege to access the serial port.
# On Ubuntu, this can be done by starting the program so:
# $ sudo wish ./monitor-edaqs.tcl
#
# P.Jacobs
# School of Engineering, Uni of Qld.
#
# 08-Feb-2014 Adapted to eDAQS project
# 13-April-2009 Finished basic functionality for daqs-drb to collect data.
# 11-March-2009 Adapted from monitor-fbox.tcl to just monitor channels every second.
# 10-August-2008 monitor-fbox.tcl
#
# ----------------------------------------------------------------
package require cursor

# set ::baudRate 921600
# set ::baudRate 115200
set ::baudRate 460800
set ::numberOfSignals 6

# Start laying out the GUI

# Main menu allow us a convenient way to exit.
menu .mb
. configure -menu .mb
menu .mb.file -tearoff 0
.mb.file add command -label "Exit" -command { tidyUp; exit }
.mb add cascade -label File -menu .mb.file
menu .mb.action -tearoff 0
.mb.action add command -label "Start Monitor" \
    -command { cancelPendingEvents; updateAnalogValues }
.mb.action add command -label "Stop Monitor" \
    -command { cancelPendingEvents }
.mb.action add command -label "Begin Sampling" \
    -command { beginSampling }
.mb.action add command -label "Fetch Data" \
    -command { fetchData }
.mb add cascade -label Action -menu .mb.action

# A grid of labelled entries to show the analog values.
```

```
# Grid them in two rows, with the aentry array being global.
set aFrame [frame .af]
for {set i 0} {$i < 5} {incr i} {
    set i2 [expr $i * 2]
    set i5 [expr $i + 5]
    set lab [label $aFrame.label$i -text "$i"]
    set ent [entry $aFrame.entry$i -width 10 -textvariable ::avalue($i)]
    grid $lab -column $i2 -row 0
    grid $ent -column [expr $i2+1] -row 0
    set lab [label $aFrame.label$i5 -text "$i5"]
    set ent [entry $aFrame.entry$i5 -width 10 -textvariable ::avalue($i5)]
    grid $lab -column $i2 -row 1
    grid $ent -column [expr $i2+1] -row 1
}
pack $aFrame -fill x

# A place to enter commands directly.
set cmdFrame [frame .cf]
set ::cmdString {}
label .cf.lab -text "Command:"
entry .cf.entry -width 30 -textvariable ::cmdString
button .cf.but -text "Send Command" -command { submitCommandString }
pack .cf.lab -side left
pack .cf.entry -side left -fill x -expand 1
bind .cf.entry <KeyPress-Return> { submitCommandString }
pack .cf.but -side left
pack $cmdFrame -fill x -expand 1

proc submitCommandString {} {
    set response [sendCommand $::cmdString]
    addToLogText [cleanUpText $response]
    addToLogText "\n"
    return
}

# A scrolling text window to log messages
set textFrame [frame .tf]
set ::logText [text .tf.t -height 20 -width 70 \
    -font Courier -wrap none \
    -yscrollcommand [list $textFrame.vsb set] ]
set textScrollBar [scrollbar .tf.vsb -orient vertical \
    -command {$::logText yview} ]
pack $::logText -side left -expand 1 -fill both
pack $textScrollBar -side left -fill y
pack $textFrame -fill both -expand 1

proc addToLogText { txt } {
```

```
    $::logText insert end "$txt"
    $::logText yview moveto 1.0
    return
}

wm title . "monitor-edaqs"

proc myBusy { action widget } {
    # Just change the cursor to a watch/sand-timer
    # because using a transparent toplevel doesn't
    # work so well on MS-Windows.
    if { [string equal $action hold] } {
        ::cursor::propagate $widget watch
    } else {
        ::cursor::restore $widget
    }
    return
    # Used to cover over the toplevel manually.
    #     action is expected to be either "hold" or "forget"
    #     widget is usually the toplevel "."
    if { [string equal $action hold] } {
        # Put up a transparent toplevel and give it focus.
        if { [winfo exists .busyDialog] } {
            raise .busyDialog
        } else {
            toplevel .busyDialog -background {} -cursor watch
        }
        wm geometry .busyDialog [winfo geometry $widget]
        focus .busyDialog
    } else {
        # Assume that we want to release the busyDialog
        wm withdraw .busyDialog
        focus $widget
    }
    return
}; # end proc myBusy

# -------------------------------------------------------------
# Make the serial-port connection to the eDAQS

if { [string equal $tcl_platform(platform) windows] } {
    console show
    set ::serialPort "com2"
} else {
    set ::serialPort "/dev/ttyUSB0"
}; # end if

```

```
138 set txt "Attempting to open $::serialPort\n"
139 addToLogText $txt
140
141 set ::tty [open $::serialPort r+]
142 # We allow a short timeout period to prevent the read function
143 # from stalling for too long.
144 fconfigure $::tty -mode $::baudRate,n,8,1 -timeout 100 \
145     -translation binary -handshake none
146
147 proc sendCommand { command } {
148     append command \r
149     # puts "about to send command: $command"
150     puts -nonewline $::tty "$command"
151     flush $::tty
152     set response [read $::tty]
153     return $response
154 }
155
156 proc cleanUpText { txt } {
157     regsub -all {\x00} $txt "" txt
158     regsub -all {\r} $txt "" txt
159     # Since DRB-DAQS uses \r\n at start of lines, we need to remove
160     # leading \n character, if it is there.
161     regsub {^\n} $txt "" txt
162     return $txt
163 }
164
165 proc firstLine { txt } {
166     return [lindex [split $txt \n] 0]
167 }
168
169 #-------------------------------------------------------------
170
171 proc updateAnalogValues {} {
172     for {set i 0} {$i < $::numberOfSignals} {incr i} {
173         set cmd "a[expr $i]"
174         # addToLogText [sendCommand "$cmd"]
175         set aresponse [firstLine [cleanUpText [sendCommand "$cmd"]]]
176         addToLogText $aresponse; addToLogText "\n"
177         set ::avalue($i) [lindex [split $aresponse "="] 1]
178         update idletasks
179     }
180     after 1000 { updateAnalogValues }
181     return
182 }
183
184 proc cancelPendingEvents {} {
```

```
185     set listOfEventIds [after info]
186     while { [llength $listOfEventIds] > 0 } {
187         after cancel [lindex $listOfEventIds 0]
188         set listOfEventIds [lreplace $listOfEventIds 0 0]
189     }
190     return
191 }
192
193 proc beginSampling {} {
194     puts "beginRecordMode"
195     cancelPendingEvents
196     addToLogText "Send sample command...\n"
197     puts -nonewline $::tty "sample\r"
198     flush $::tty
199     # We are expecting that this command will take a long time
200     # so look for characters arriving in the incoming queue.
201     # We are expecting the immediate response "\r\nBegin sampling."
202     # followed, some time later, by "\r\nDone."
203     myBusy hold .
204     set countList [fconfigure $::tty -queue]
205     # puts "countList= $countList"
206     while { [lindex $countList 0] < 20 } {
207         set countList [fconfigure $::tty -queue]
208         # puts "countList= $countList"
209         update; # so we don't lock the GUI
210     }
211     set aresponse [cleanUpText [read $::tty]]
212     myBusy forget .
213     # addToLogText $aresponse; addToLogText "\n"
214     addToLogText "Finished sampling; you may want to fetch the data now.\n"
215     return
216 }
217
218 proc fetchData {} {
219     puts "fetchData"
220     cancelPendingEvents
221     addToLogText "Send report command; this may take some time...\n"
222     update
223     puts -nonewline $::tty "report\r"
224     flush $::tty
225     myBusy hold .
226     set ::CSVdata [cleanUpText [read $::tty]]
227     myBusy forget .
228     addToLogText "Finished receiving CSV data."
229     set fileName [tk_getSaveFile -title "Save to CSV file" \
230                       -initialfile "edaqs-data.csv"]
231     if { [string length $fileName] > 0 } {
```

```
232         set fp [open $fileName "w"]
233         puts $fp $::CSVdata
234         puts $fp "\n"
235         close $fp
236     } else {
237         puts "Zero-length fileName; nothing written."
238     }
239     return
240 }
241 
242 #-------------------------------------------------------------
243 
244 proc tidyUp {} {
245     if { [catch {close $::tty} err] } {
246         puts "close serial port failed: $err"
247     }
248     return
249 }
250 
251 #-------------------------------------------------------------
252 # At this point, the event loop will take over.
253 set ::edaqsVersion [firstLine [cleanUpText [sendCommand "status"]]]
254 addToLogText "eDAQS has version string: $::edaqsVersion\n"
255 update idletasks
256 # Start the monitoring of analog values from the pull-down menu.
257 # after 1000 { updateAnalogValues }
```

A Initial prototype boards

The initial protoype recording boards were built in “master” and “node” versions, with through-the-hole components on stripboard. The external SRAM chip was put on a separate board and was connected via the SPI port. The master started with both 5V and 3.3V linear regulators while the node prototype had only the 3.3V regulator, with 5V being supplied by the master board.

The current recording boards in Section 1 make no distinction. They’re all recording nodes. If you use more than one in a model, communicate with them via the I2C port. When using just one, the choice between UART and I2C communication is arbitrary.

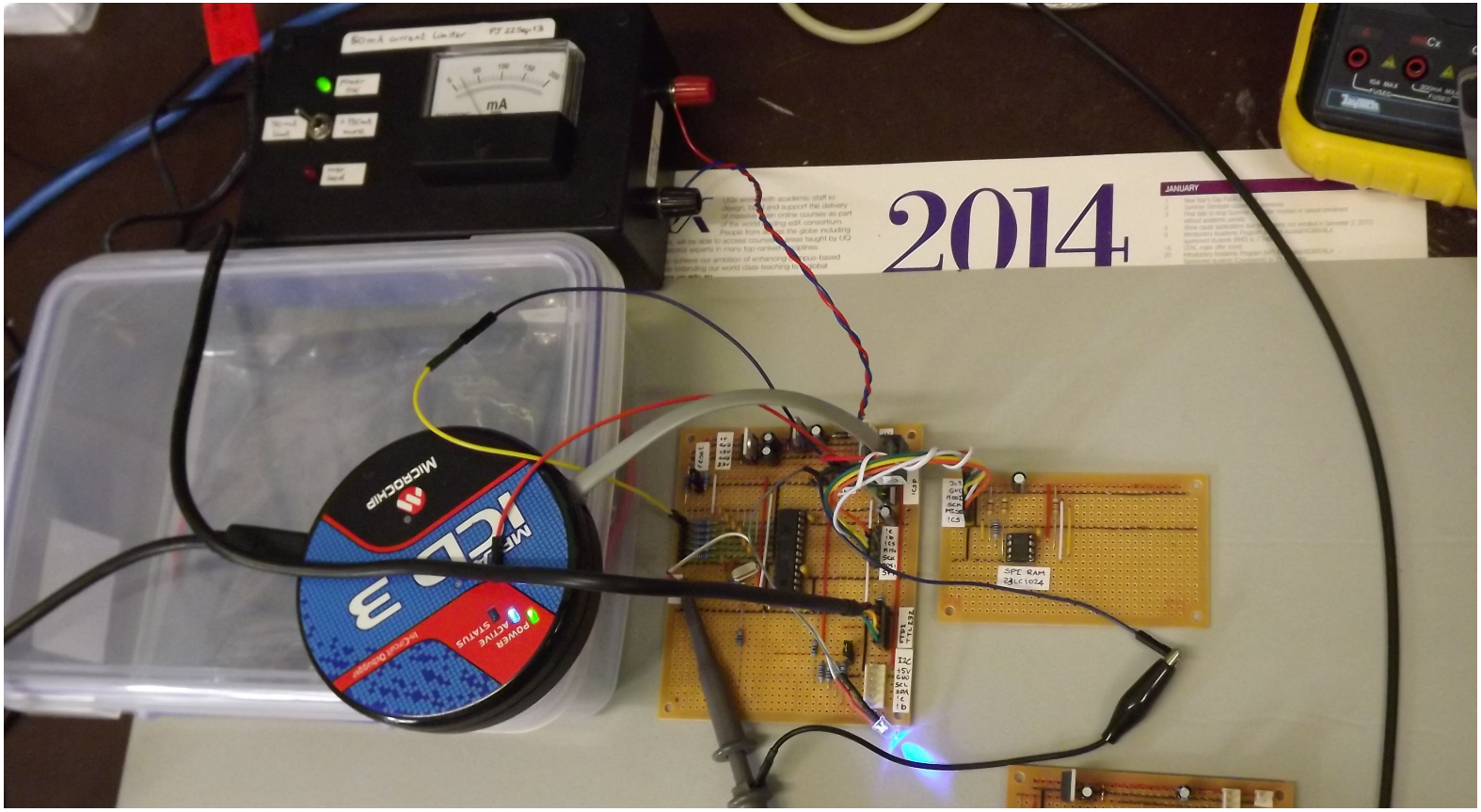
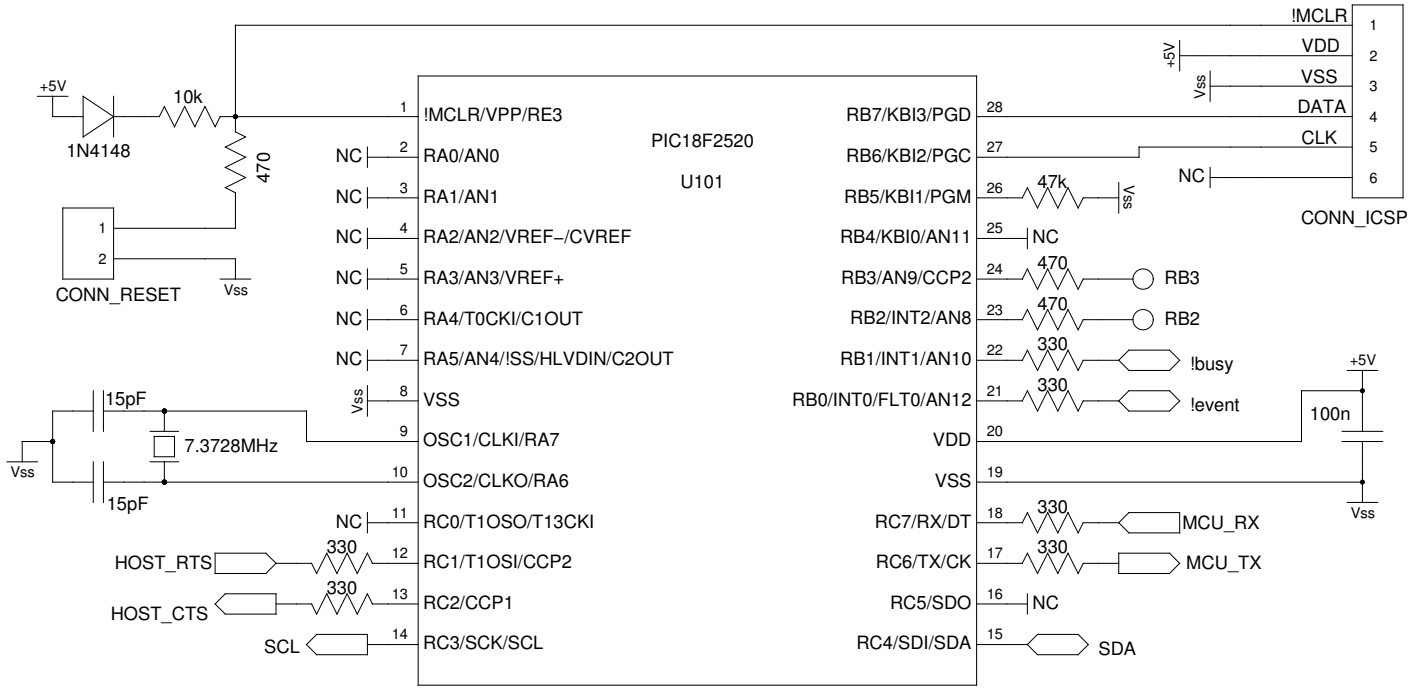
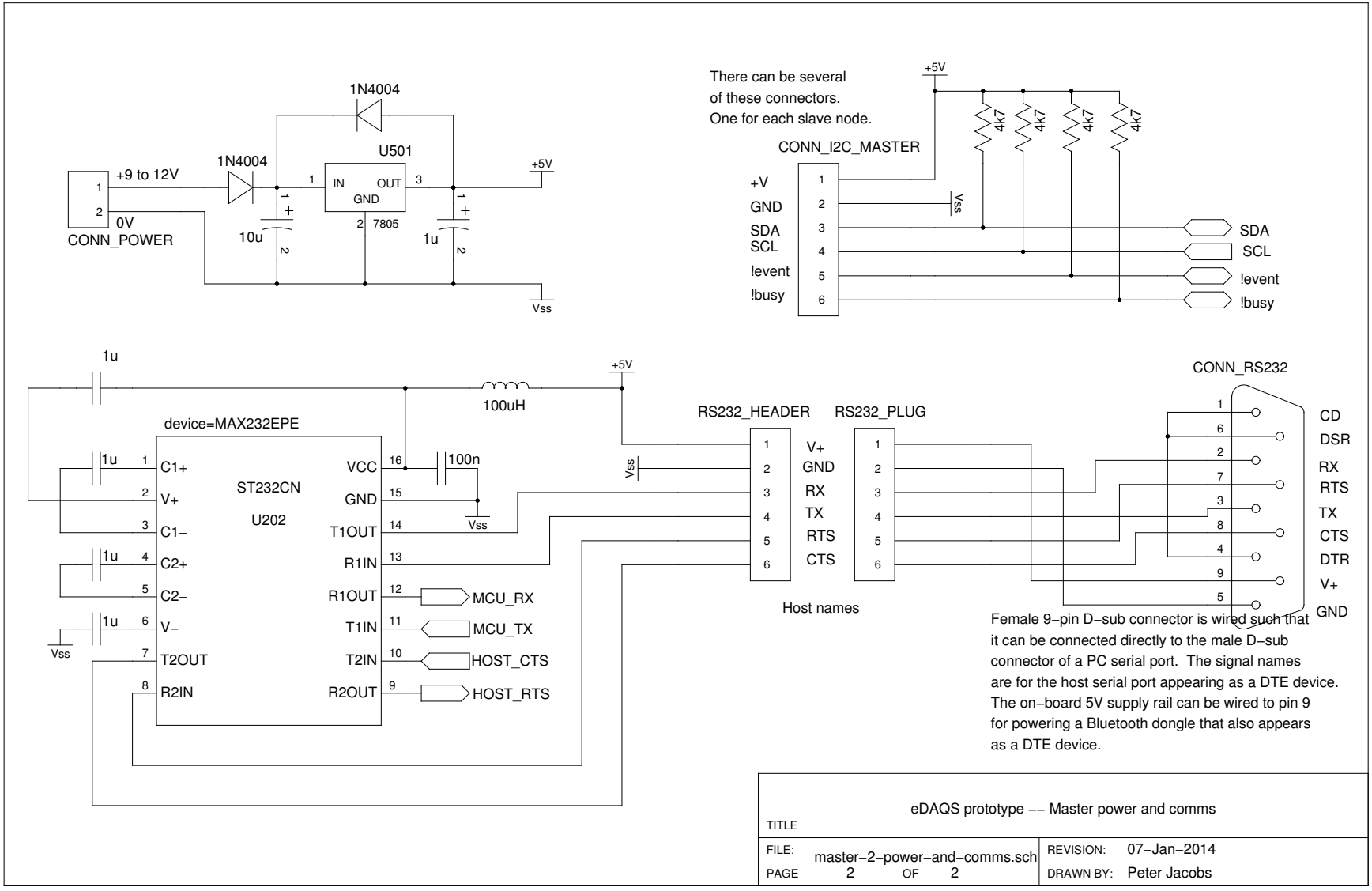
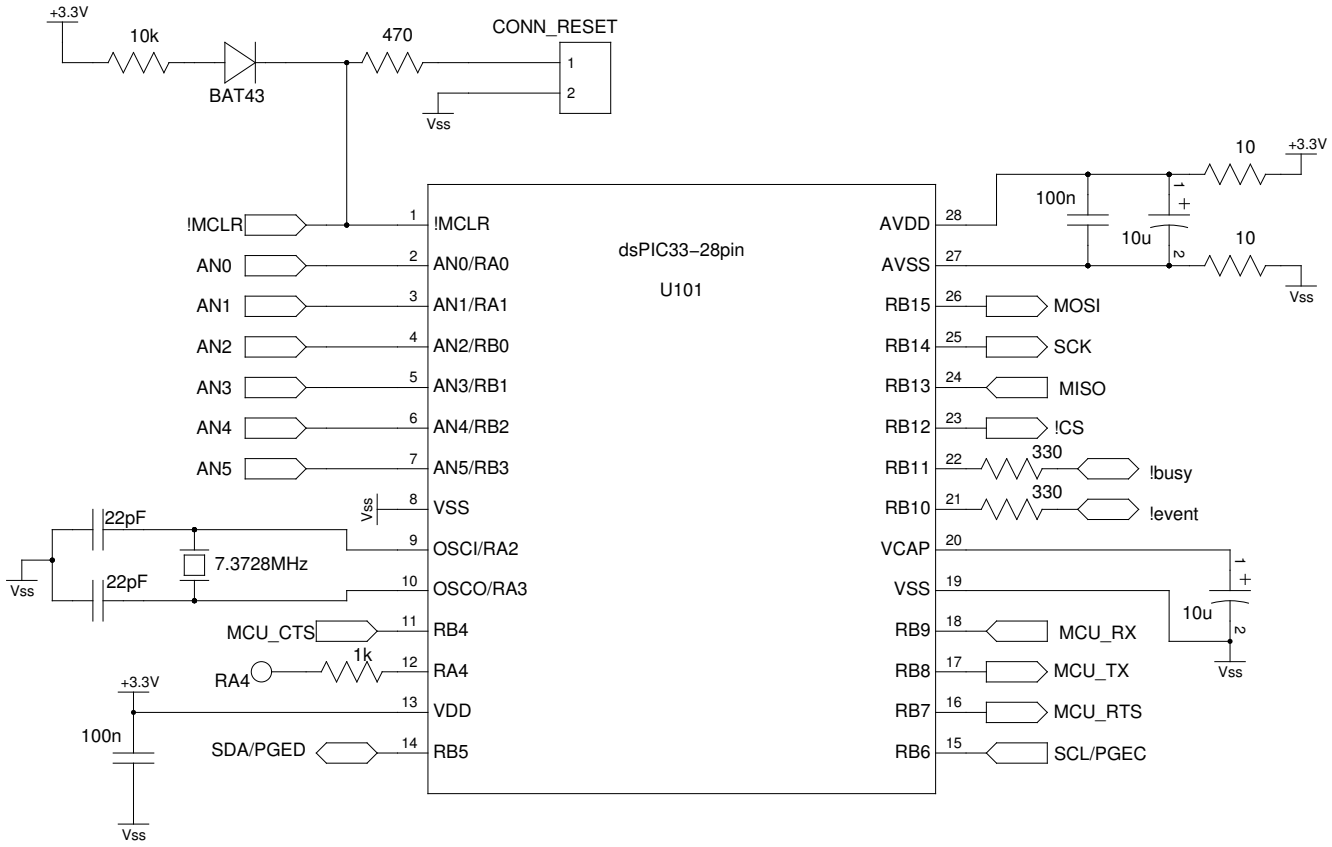


Figure 4: The initial protoype recording board with external SRAM chip on a separate strip-board. The current-limiting power box (black box, top-left) is indicating a current draw of about 20 mA.

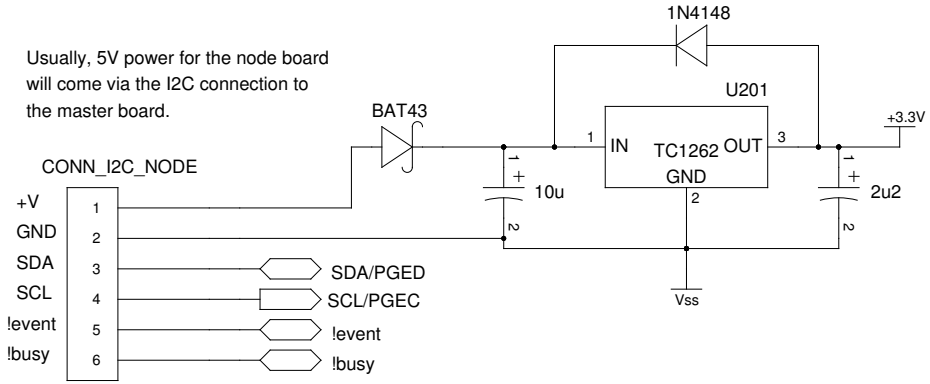


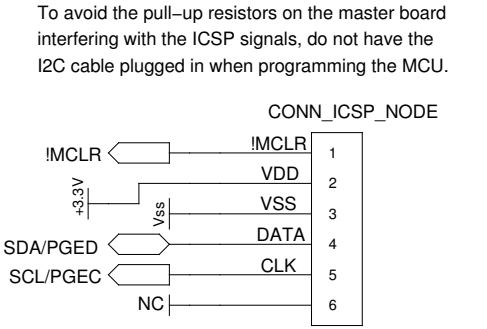
TITLE	eDAQS prototype −− master MCU	
FILE: master−1−mcu.sch		REVISION: 07−Jan−2014
PAGE 1 OF 2		DRAWN BY: Peter Jacobs

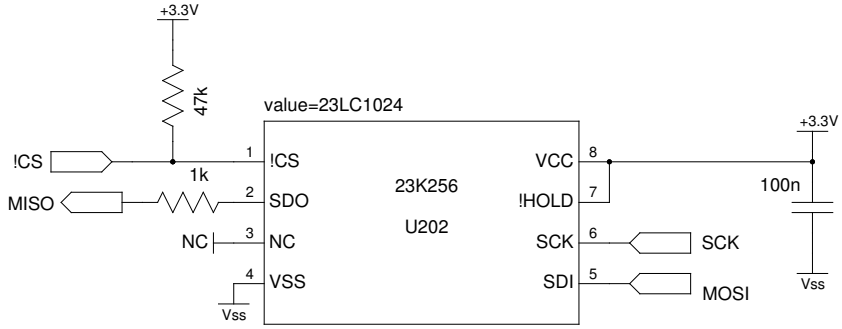

1N4004
1N4004
U501
+9 to 12V
0V
CONN_POWER
IN
OUT
GND
7805
10u
1u
+5V
Vss
There can be several
of these connectors.
One for each slave node.
CONN_I2C_MASTER
+V
GND
SDA
SCL
!event
!busy
4k7
4k7
4k7
4k7
SDA
SCL
!event
!busy
1u
100uH
device=MAX232EPE
ST232CN
U202
C1+
V+
C1−
C2+
C2−
V−
T2OUT
R2IN
VCC
GND
T1OUT
R1IN
R1OUT
T1IN
T2IN
R2OUT
100n
MCU_RX
MCU_TX
HOST_CTS
HOST_RTS
RS232_HEADER
RS232_PLUG
V+
GND
RX
TX
RTS
CTS
Host names
CONN_RS232
CD
DSR
RX
RTS
TX
CTS
DTR
V+
GND
Female 9−pin D−sub connector is wired such that
it can be connected directly to the male D−sub
connector of a PC serial port. The signal names
are for the host serial port appearing as a DTE device.
The on−board 5V supply rail can be wired to pin 9
for powering a Bluetooth dongle that also appears
as a DTE device.
TITLE
eDAQS prototype −− Master power and comms
FILE:
master−2−power−and−comms.sch
REVISION:
07−Jan−2014
PAGE
2
OF
2
DRAWN BY:
Peter Jacobs

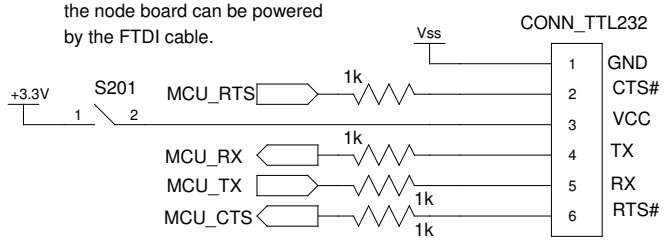


TITLE	eDAQS prototype −− node MCU	
FILE:	node−1−mcu.sch	REVISION: 07−Jan−2014
PAGE	1 OF 3	DRAWN BY: Peter Jacobs

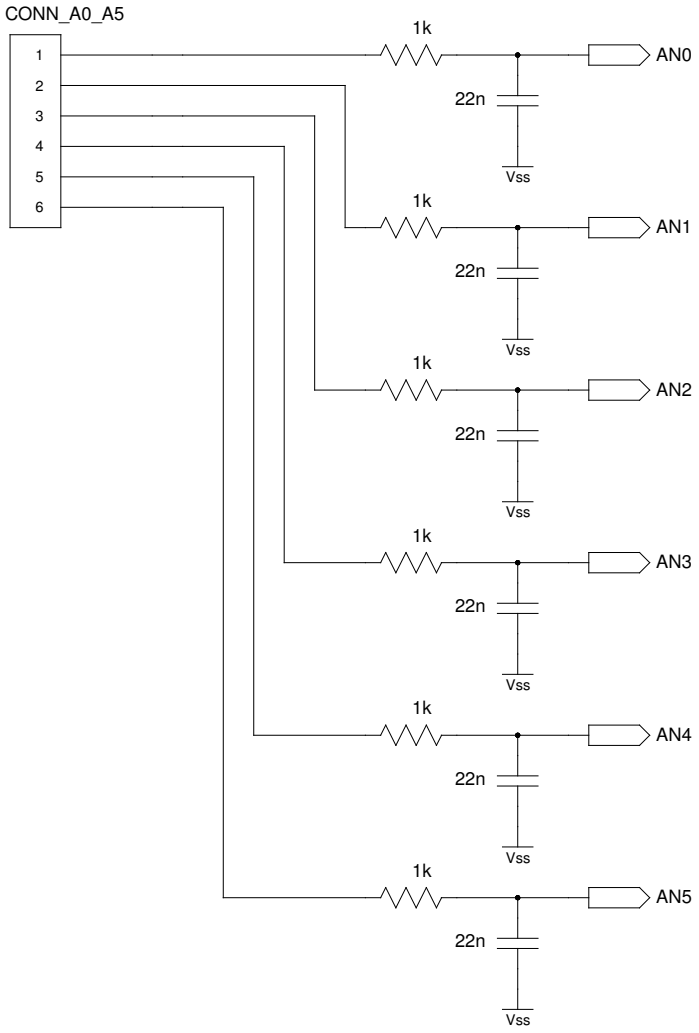




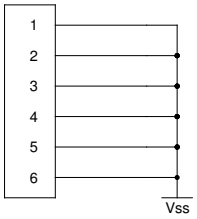




TITLE	eDAQS prototype: node power, communications, RAM		
FILE:	node−2−power−comms−ram.sch	REVISION:	07−Jan−2014
PAGE	2 OF 3	DRAWN BY:	Peter Jacobs



CONN_GND



In a more refined prototype, the analogue connection could be via a 6x2 connector. Power, at 5V, could also be supplied by another header.

TITLE	eDAQS prototype –– node analogue filters	
FILE:	node–3–analog–filters.sch	REVISION: 07–Jan–2014
PAGE	3 OF 3	DRAWN BY: Peter Jacobs

References

[1] Microchip Technology Inc. dsPIC33EPXXXGP502, dsPIC33EPXXXMC20X/50X, and PIC24EPXXXGP/MC20X 16-bit microcontrollers and digital signal controllers with high-speed PWM, op amps and advanced analog. Datasheet DS70000657H, Microchip Technology Inc., www.microchip.com, 2013.

[2] Microchip Technology Inc. 23A1024/23LC1024: 1Mbit SPI serial SRAM with SDI and SQI interface. Datasheet DS20005142B, Microchip Technology Inc., www.microchip.com, 2013.

[3] Microchip Technology Inc. TC1262: 500mA fixed output CMOS LDO. Datasheet DS21373C, Microchip Technology Inc., www.microchip.com, 2012.

[4] Texas Instruments Inc. bq24072, bq24073, bq24074, bq24075, bq24079 1.5A USB-friendly Li-Ion battery charger and power-path management IC. Datasheet SLUS810I, Texas Instruments Inc., www.ti.com, Jan 2014.

[5] Texas Instruments Inc. TPS55340 integrated 5-A 40-V wide input range boost/SEPIC/flyback DC-DC regulator. Datasheet SLVSBD4B, Texas Instruments Inc., www.ti.com, Oct 2012.

[6] Honeywell. SDX series microstructure pressure sensors. low cost, compensated, DIP package 0 psi to 1 psi up to 0 psi to 100 psi pressure sensors. Datasheet 008103-2-EN, Honeywell International Inc., www.honeywell.com/sensing, 2003.

[7] Honeywell. ASDX series digital pressure sensors. low pressure and ultra-low pressure digital output, ±2% total error band, 10 inches H_2O to 100 psi. Datasheet 008095-11-EN, Honeywell International Inc., www.honeywell.com/sensing, July 2010.

[8] Honeywell. I^2C communications with honeywell digital output pressure sensors. Application Note 008201-3-EN, Honeywell International Inc., www.honeywell.com/sensing, May 2012.

[9] Invensense Inc. MPU-6000 and MPU-6050 product specification, Revision 3.4. Datasheet PS-MPU-6000A-00, Invensense Inc., www.invensense.com, Aug 2013.