

USBxCH User Manual

Manual Revision (2015/12/01)

Board Revision E

Symmetric Research

www.symres.com

Combined USB4CH and USB8CH User Manual

(with circuit diagrams)

Contents

| | | |
|----------|--------------------------------------|-----------|
| 1 | Introduction | 8 |
| 2 | Getting started | 10 |
| 2.1 | Unpack the software | 10 |
| 2.2 | Install the driver | 11 |
| 2.3 | Connect the cables | 12 |
| 2.4 | Run Diag, DVM, Scope | 14 |
| 3 | Software Overview | 16 |
| 3.1 | Utilities and pipelines | 16 |
| 4 | Acquisition Programs | 18 |
| 4.1 | Ascii | 19 |
| 4.2 | DVM | 20 |
| | 4.2.1 starting the program | 22 |
| | 4.2.2 ini syntax | 22 |
| | 4.2.3 output files | 23 |
| 4.3 | Scope | 24 |
| | 4.3.1 starting the program | 26 |
| | 4.3.2 ini syntax | 26 |
| | 4.3.3 output files | 27 |
| 5 | Utilities - Pipeline | 28 |
| 5.1 | command line usage | 29 |
| 5.2 | ini files and syntax | 30 |
| 5.3 | error reporting | 31 |
| 5.4 | running by hand | 31 |
| 5.5 | running from batch files | 32 |
| 5.6 | where are the data files ? | 33 |
| 5.7 | Short reference | 33 |
| | 5.7.1 Blast | 34 |
| | 5.7.2 Pak2Bin | 37 |
| | 5.7.3 Bin2Asc | 38 |

| | | |
|-----------|--|-----------|
| 6 | Utilities - Standalone | 39 |
| 6.1 | Diag | 40 |
| 6.2 | SetDid | 42 |
| 6.3 | EquipStatus | 44 |
| 6.4 | GpsProg | 45 |
| 6.5 | NmeaTime | 46 |
| 6.6 | DigitalIo | 48 |
| 6.7 | Calibrate | 49 |
| 6.8 | Presence | 50 |
| 6.9 | YmdHms | 50 |
| 7 | BIN data structure layout | 51 |
| 8 | Fast acquisition | 54 |
| 9 | Sampling rates | 56 |
| 9.1 | Permitted rates | 57 |
| 9.2 | Interpolation to 1Hz, 100Hz, and other rates | 58 |
| 9.3 | Master clock stability | 58 |
| 10 | FIFO Depth and Overflow | 60 |
| 10.1 | FIFO Depth | 60 |
| 10.2 | FIFO Overflow | 62 |
| 10.3 | FIFO Creep | 62 |
| 11 | A/D reference voltage | 63 |
| 11.1 | Standard reference | 64 |
| 11.2 | Alternate references | 64 |
| 12 | Analog inputs | 65 |
| 12.1 | Differential signals | 65 |
| 12.2 | DB front panel analog pin assignments | 69 |
| 12.3 | Twisted pair cabling | 71 |
| 12.4 | Static shielding | 72 |
| 12.5 | Input impedance | 73 |
| 12.6 | Input voltage range | 74 |
| 12.7 | Op amp gain | 74 |
| 12.8 | RC antialias filtering | 75 |
| 13 | Analog DC calibration | 76 |
| 13.1 | Full Scale Voltage Span and Counts | 77 |
| 13.2 | Theoretical counts per volt | 78 |
| 13.3 | Calibration slope and offset | 79 |

| | |
|--|------------|
| 14 Analog AC calibration | 80 |
| 14.1 Theoretical AC transfer function | 81 |
| 14.2 Measured transfer function | 82 |
| 15 Digital IO | 84 |
| 15.1 Digital input | 84 |
| 15.2 Digital output | 85 |
| 15.3 Additional digital timing and GPS signals | 85 |
| 15.4 DB25 pin assignments | 86 |
| 15.5 Static shielding | 88 |
| 15.6 User configuration byte | 88 |
| 15.7 Programming the front panel red and yellow LEDs | 89 |
| 15.8 Seeing the digital inputs in Scope | 89 |
| 16 GPS Time Stamping | 90 |
| 16.1 What is GPS ? | 90 |
| 16.2 Required GPS signals | 91 |
| 16.3 DB25 pin assignments | 91 |
| 16.4 Using a Garmin 16x HVS with the USBxCH | 93 |
| 16.5 Determining RS232 polarity | 94 |
| 16.6 Determining PPS polarity | 95 |
| 16.7 Front panel red LED | 96 |
| 16.8 Expected NMEA strings | 97 |
| 16.9 Programming the GPS antenna | 98 |
| 16.10 Seeing the GPS time stamps in Scope and Blast | 99 |
| 16.11 What does time stamping mean ? | 100 |
| 16.12 Driving multiple USBxCH systems from one GPS | 101 |
| 17 NTP Time Stamping | 102 |
| 17.1 Setting the PC clock from GPS | 102 |
| 18 Power Supply | 103 |
| 18.1 Connectors | 103 |
| 18.2 Voltage and current requirements | 104 |
| 18.3 LED power status indicators | 105 |
| 18.4 Power History signal | 106 |
| 18.5 Reset and power cycling | 106 |
| 18.6 Current limiting | 106 |
| 19 Temp Sensor | 107 |
| 19.1 Temp records | 107 |
| 20 Specifications | 108 |
| 20.1 Specifications table | 109 |
| 20.2 Noise floor | 110 |

| | |
|---|------------|
| 21 Circuit Diagrams | 113 |
| 21.1 USB4CH schematics | 114 |
| 21.12 USB8CH schematics | 125 |
| 22 Examples and Experiments | 136 |
| 22.1 AA battery | 137 |
| 22.2 Absolute calibration | 142 |
| 22.3 10 turn potentiometer | 143 |
| 22.4 Ratiometric technique | 148 |
| 22.5 Solar cell | 149 |
| 22.6 GnuPlot | 150 |
| 22.7 Twisted pair | 153 |
| 22.8 Acquiring 1Hz data | 154 |
| 22.9 Multiple USBxCH | 155 |
| 22.10 Powering with batteries | 156 |
| 23 Frequently Asked Questions | 161 |
| 23.1 General | 161 |
| 23.2 Hardware | 163 |
| 24 Extra supplies | 166 |
| 24.1 Small Parts for cables etc | 167 |
| 25 Using Adobe Reader effectively | 170 |
| 26 Getting Technical Help | 171 |

List of Figures

| | | |
|------|--|----|
| 1.1 | mini Netbook with USB4CH and geophone | 9 |
| 4.1 | Ascii sample display | 19 |
| 4.2 | DVM sample display | 20 |
| 4.3 | Scope sample display | 24 |
| 9.1 | SPS rate table | 57 |
| 10.1 | FIFO hold out time table | 61 |
| 12.1 | Differential vs single ended signals | 66 |
| 12.2 | Balanced differential inputs | 67 |
| 12.3 | Two terminal floating sensor connection | 68 |
| 12.4 | Analog DB15 pin assignment table | 69 |
| 12.5 | Analog DB15 pin numbers viewed from front panel | 70 |
| 12.6 | Analog DB15 pin signals viewed from front panel | 70 |
| 12.7 | Magnetic coupling of 50/60Hz noise | 71 |
| 13.1 | A/D counts with differential input | 77 |
| 13.2 | A/D counts with single ended input | 78 |
| 14.1 | AC response: theoretical sinc | 81 |
| 14.2 | AC response: physical measurement setup | 82 |
| 14.3 | AC response: physical measurement setup | 82 |
| 14.4 | AC response: physical measurement setup | 83 |
| 15.1 | Digital DB25 pin assignment table | 86 |
| 15.2 | Digital DB25 pin numbers viewed from front panel | 87 |
| 15.3 | Digital DB25 pin signals viewed from front panel | 87 |
| 15.4 | Digital DB25 UserCfgByte | 88 |
| 16.1 | GPS DB25 pin assignment table | 92 |
| 16.2 | GPS DB25 pin signals viewed from front panel | 92 |
| 16.3 | Garmin GPS 16x HVS wires | 93 |
| 16.4 | Garmin GPS 16x HVS wiring | 94 |
| 16.5 | Garmin GPS 16x HVS finished cabling | 95 |

| | | |
|-------|--|-----|
| 16.6 | PPS signal polarities | 96 |
| 20.1 | Specifications table | 109 |
| 20.2 | Noise floor time domain plots | 110 |
| 20.3 | Noise floor histogram at 130Hz | 111 |
| 20.4 | Noise floor histogram at 1302Hz | 111 |
| 22.1 | AA battery test lead setup | 137 |
| 22.2 | AA battery DVM display screen | 138 |
| 22.3 | AA battery circuit floating | 139 |
| 22.4 | AA battery with AGND connection | 139 |
| 22.5 | AA battery circuit grounded | 140 |
| 22.6 | AA battery 6 volt circuit | 141 |
| 22.7 | 10 turn circuit | 143 |
| 22.8 | 10 turn photo top | 144 |
| 22.9 | 10 turn photo bottom | 145 |
| 22.10 | 10 turn Calibrate screen | 146 |
| 22.11 | 10 turn DVM display screen | 146 |
| 22.12 | Solar cell light level measurements | 149 |
| 22.13 | GnuPlot graph | 151 |
| 22.14 | NiMH AA battery pack | 156 |
| 22.15 | Battery voltage divider | 157 |
| 22.16 | NiMH AA battery pack discharge plot | 158 |
| 22.17 | Lead acid 3Ah battery | 159 |
| 22.18 | Lead acid 3Ah battery discharge plot | 160 |

Chapter 1

Introduction

The SR USB4CH and USB8CH are precision 24 bit analog data acquisition systems for use with host computers having USB ports. They feature 4 or 8 analog channels, and except for the number of analog channels and physical size, are drop in replacements for each other. Collectively they are referred to as the USBxCH.

The systems both feature a 24 bit A/D converter per channel, eliminating the crosstalk and skew common with single multiplexed A/D systems. Additional features are :

- Simple USB PC / host computer interface
- High precision 24 bit A/D converter per analog channel
- Simultaneous synchronous analog and digital recording, no skew
- Analog input range +/- 4 volts, balanced differential, +/- 10 optional
- Response to true DC, sampling rates to 9.7 kHz
- GPS time stamping with 800 nanosecond accuracy
- 2MB/4MB FIFO for no data loss due to PC latencies
- On board continuously recorded temp sensor
- Finished ready to go applications, and pipeline utility library
- Windows 7/8/10 and Linux support (32 and 64 bit versions for all)
- Free web software downloads

This manual covers the software and hardware aspects of the USBxCH systems. Also see the ReadMe.txt files in many of the software directories for further information.

We hope the USBxCH is a useful tool for your applications

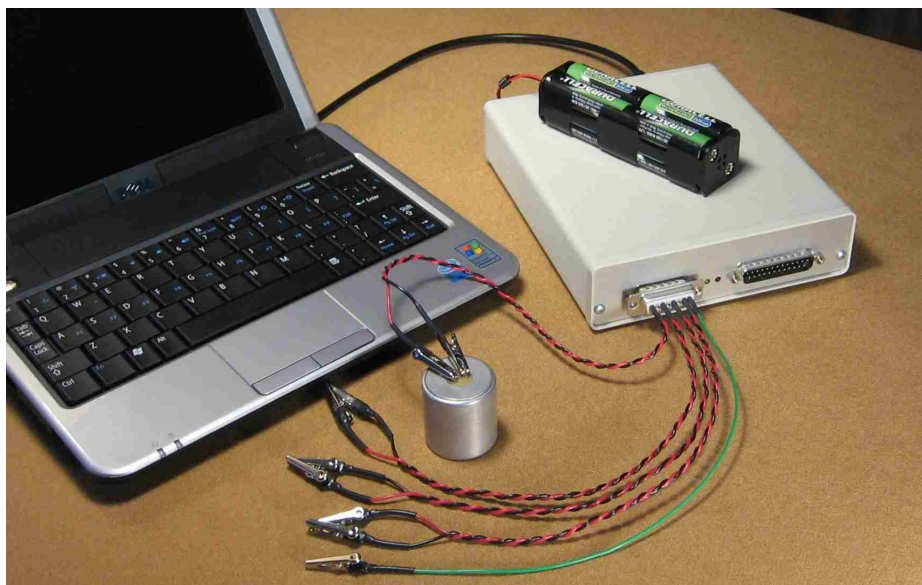


Figure 1.1: mini Netbook with USB4CH and geophone

One popular USBxCH configuration is with a mini Netbook for portable data acquisition. In the photo above, the small cylinder is a geophone, a passive sensor used for seismic surveys. A wide variety of other sensors can be used for applications from DC to 10kHz. For software support, the [Scope](#) program can be used to easily acquire, display, and save data in real time.

In the setup above, the geophone sensor is connected to one analog channel, while the other three USB4CH channels are still available. The red/black alligator pairs are differential inputs for each of the analog channels. The green alligator wire is AGND, which is not required for a floating passive sensor. A [GPS](#) interface is provided on the DB25 for precision data time stamping.

The USBxCH is self powered. Power supplies varying from 8 to 24 volts are acceptable. A 110/220 vac wall transformer is supplied with the system. Various types of batteries can also be used for power. As shown above, eight AA NiMH batteries will power the USB4CH for over 20 hours. See [powering with batteries](#) for a discussion. Many other configurations are possible. The USBxCH can be used with any Windows or Linux computer having a USB port.

Chapter 2

Getting started

Installation of the USBxCH is fairly easy. The quick steps are:

- ✓ **Unpack the software** find the CDROM or download from the web
- ✓ **Install the driver** follow the directions in Driver.txt
- ✓ **Connect the cables** attach the power supply and USB cable
- ✓ **Run Diag, DVM, Scope** take it on a test drive

2.1 Unpack the software

The USBxCH software can be found on a CD included in the shipping box, or it can be downloaded from the web.

The CD and web distributions are the same, except for updates available with the web versions. However, unpacking from the CD and web are slightly different:

■ unpacking from the CD

The CD contains both the Windows and Linux distributions on the same disk. Place the disk in your CD drive. There is no autorun. Instead you must start a command line prompt and go to the d: drive (or e: etc if the CD is installed there) and then the Windows or Linux directory as suitable. Once there run:

```
cmd:prompt> install.bat
```

at the command prompt. This will create the directory C:/SR/USBXCH on your hard drive

and unzip the CD files there. *This step only copies files to your hard disk.* No registry entries or driver installation are made with this step. Once the files are unpacked, examine the subdirectories. In particular, `/SR/USBXCH/Driver` will be needed later.

If you wish to remove the software from your system at this point, simply delete the `/SR/USBXCH` directory. Nothing else is required. Later, after the device driver has been installed, you must follow the driver instructions to fully remove it also.

You must accept installation into the standard /SR/USBXCH directory. Installation into other directories is not supported. Most of the executables can be run from any directory, but shortcuts and batch files may have the standard path hardwired into them. If you need to reinstall the software, delete or rename the current `/SR/USBXCH` directory and then run `install.bat` again.

■ unpacking from the web: www.symres.com

If you want to upgrade to the latest software version, or simply to review the product, the full software package is always available for free download from :

www.symres.com/download

There you will find zip/tgz files with the releases for Windows/Linux. Download the distribution of your choice to a temp directory on your hard disk.

Once in the temp directory, extract the zip/tgz files by hand. Among the files in those extraced from the web zip/tgz will be an `install.bat`. Execute this `install.bat` while in the temp directory to further unpack the software and create `/SR/USBXCH`. If desired, delete the temp directory when done by hand.

■ confirming you have the right distribution: check `OS.txt`

After unpacking the software confirm the distribution by going to `/SR/USBXCH` and checking the contents of the `OS.txt` file. If you have the wrong installation, delete and try again. *Windows executables will not run on Linux, and visa versa !!*

2.2 Install the driver

After the software has been unpacked and is in the `/SR/USBXCH` directory you must install the USBxCH driver.

... once upon a time in a land far away ...
you could connect the USB cable and the wheels turned

Unfortunately ... in the age of security concerns life is more complicated. Although the USBxCH driver has been designed to be easy to use, dealing with the security certificates requires user involvement.

For a detailed list of driver installation steps, refer to the readme files:

```
        /SR/USBXCH/Driver/"Driver ReadMe.txt"      ( << Windows )  
/usr/local/SR/USBXCH/Driver/"Driver ReadMe.txt"      ( << Linux )
```

for Windows or Linux. *Read these files carefully*, the steps given there will carry you through driver installation, including setting up Windows security certificates and Linux device and port permissions.

The USBxCH driver is built on top of the WinUsb (Windows) or LibUsb (Linux) drivers. WinUsb/LibUsb are usually installed as part of the operating system, and are available by default on almost all Windows and Linux computers.

Building the USBxCH software on top of WinUsb/LibUsb greatly increases portability. Most operating system specific issues are taken care of in WinUsb/LibUsb. The USBxCH software stays the same regardless of the Windows service pack or Linux kernel.

If your computer does not have WinUsb/LibUsb, you may have to install it. This mainly happens with single board computers with small Linux installations. *Beagle Bone Black Rev C users can relax, it has a full Debian with LibUsb preinstalled !*

2.3 Connect the cables

After unpacking the software and installing the driver, you need to connect the USBxCH power and USB cable. Do the following:

■ Hook up power

✓ Find the wall transformer

Look for the wall transformer in the shipping box. Typically it will be a small universal switcher compatible with both US 110 and Intl 220 vac wall power. The output should be 9vdc 300ma. See the USBxCH spec sheet for the full range allowed.

The USBxCH *is not powered* from the USB cable. You must use the supplied wall transformer or other equivalent power source. In USB terminology, the USBxCH is a self powered device.

✓ **Plug the wall transformer into the wall**

International customers may need a plug adapter to match your wall socket.

The few customers who receive linear wall transformers without universal inputs *should not* connect to a wall socket for which it is not rated. Plugging a linear 110 transformer into 220 wall power is dangerous.

✓ **Plug the 2.1mm barrel power connector into the USBxCH**

Plug the 2.1mm barrel connector at the other end of the wall transformer cable into the USBxCH back panel power jack. There are two jacks in parallel for daisy chaining power to other devices if needed. It does not matter which one you use. *Make sure the 2.1mm plug is fully seated into the jack.*

✓ **Is the green LED on ?**

If the wall transformer is energized, the green LED on the back panel near the 2.1mm power jacks will light up. If the green LED is off, then there is a basic problem with the wall transformer. Check the wall power and connections. You must fix it to continue.

✓ **Is the red LED off ?**

The red LED near the 2.1mm power jacks may momentarily light up when power is applied, but will go off in a second or two. *If the red LED stays on, it indicates the power is not within specifications and something is wrong.* Perhaps the wall transformer voltage is low, or the USBxCH is suffering a short. If the red LED is on, the problem must be fixed before the system will function correctly.

At this point, if the *green LED is on and the red LED is off*, then the USBxCH is properly powered and you are ready to connect the USB cable.

■ **Hook up the USB cable**

If the device driver has been installed, and the USBxCH is powered and connected, then the PC will automatically detect the new USB hardware:

✓ **Find the USB cable**

Find the USB cable included in the shipping box. One end has a flat type A connector for the PC, the other a square type B connector for the USBxCH.

✓ **Plug the flat type A end into the PC**

Plug the flat type A end into your computer. Note that the connector is polarized. Do not use excessive force and plug it in upside down.

✓ **Plug the square type B end into the USBxCH**

Connecting to the USBxCH will start a sequence of events behind the scenes. You should see a small green notification come up in the system tray.

No hardware Wizard or otherwise will come up. All the work formerly done by the Wizard has already been done when installing the driver earlier and running "pnputil". See the /SR/USBXCH/Driver/"Driver ReadMe.txt" file.

✓ **Check the Device Manager**

Besides seeing the green notification in the system tray, you should go to the computer Control Panel and bring up the Windows or Linux Device Manager. Among the devices listed should be the "SR Instrumentation" group. That confirms the system recognizes the USBxCH ... success!

The USBxCH *must be connected and powered up* to appear in Device Manager. A powered off USBXCH will not appear in Device Manager.

Suppose you unplug the USBxCH and plug it onto a different host USB port, what will happen? In a few moments the PC will find the new hardware and automatically install an instance of the USBxCH driver to service the new port. If plugged and replugged on the same port the USBxCH will be recognized almost instantly.

If you are running multiple USBxCH systems on one PC you will need to know their names. This will be one of: `SrUsbXch(0,1,2,3,...)` depending on how their EEPROMs are programmed. See the **SetDid** utility for information. New systems from the factory have the name:

`SrUsbXch0 << device name`

Finally, to remove the driver, follow the instructions in the "Driver ReadMe.txt" file.

2.4 Run Diag, DVM, Scope

■ **More confirmation of installation: run Diag.exe**

Run the SR **Diag** utility to check the hardware and software. Execute it by double clicking on the "Run Diag Install" shortcut icon, or from the command line:

```
cmd:prompt> Diag install
```

Run with `/?` for help. If trouble, run `Diag debug` and **email** us the report files.

■ **Have some fun: run DVM and Scope !**

After running Diag, users should test acquiring data with **DVM** or **Scope**. For quick starts with default parameters, double click on the shortcut icons in the respective directories.

When running, it is tempting to touch the analog input pins to apply small voltages. *Avoid the temptation to do this.* You will inadvertently discharge static electricity into the inputs. Even small static shocks such as those from walking on a carpet will do damage. Generally the damage is cumulative, with calibration and analog performance steadily degraded with each ESD event. If you must touch the input pins, *touch the metal enclosure immediately before doing so.* This will help discharge any static. Better yet, wear an antistatic wrist band clipped onto the front panel.

See the **Analog inputs** chapter for details on the analog input voltage ranges and differential signals. For introductory hands on usage see the **Examples and Experiments** chapter.

Chapter 3

Software Overview

The SR USBxCH software has changed in major ways with the 2015 release. This chapter gives a overview. The new design is pipeline oriented and can be easily configured for a variety of applications with a minimum of programming.

For those who want to acquire data quickly, the finished applications:

```
"App - Ascii"  
"App - DVM"  
"App - Scope"
```

provide easy real time displays, with DVM and Scope also saving their data to disk files.

Parameters for these finished applications such as sampling rate, display settings, etc are specified in INI files which can be edited with any text editor. Use the Windows shortcuts and Linux script files to run with various prefixed settings. See [Acquisition Programs](#) for details. For many users these apps are all that is needed, you need look no further.

3.1 Utilities and pipelines

Underlying pretty finished programs like DVM and Scope is the core of how the system works. Utilities and pipelines. In previous releases the system relied on temporary files. Data progressed from one temporary file to the next as it was processed ...

The new system passes data from one utility to the next with *named pipes*. This is a big advantage. A named pipe is a FIFO maintained by the operating system in the DRAM memory of the computer. The data *is not* written to disk while in the pipe. Temporary intermediate files do not have to proliferate. Data does not have to touch a mass storage device until the very end for a final save.

For systems using flash memory the advantage of pipelines is huge, where excessive use of the file system directory structure is very slow. For some applications pipeline data never even has to touch local mass storage at all, it can go directly to an internet connection!

Many users will already be familiar with the utility and pipeline concept. In Windows and Linux it is fairly common to use a command line like:

```
dir c: | more
```

to *pipe* the output of "dir c:" to "more", pausing as the dir listing flies by. The vertical character | is the pipe symbol. Output is piped from one utility to another, often even through a sequence of many utilities. You never know the name of the pipes involved, but clearly there is an intermediate FIFO (first in first out) data structure in place between each pair of utilities. Linux users often become skilled using pipelines to achieve complicated custom effects. All without having to write and compile any code at all.

The *named pipes* we use in this USBxCH release are similar to "|" but even more flexible. With the familiar | symbol data always progresses from stdout to stdin, and is targeted at ASCII text. Named pipes are designed for use with any type of data. The layout for using named pipes with the USBxCH looks like this:

```
Blast inifile    /to Pak2Bin
Pak2Bin inifile  /to Bin2Asc
Bin2Asc inifile  /to NULL
```

The pipeline is laid out in vertical fashion rather than horizontal. Here, three utilities are used, with the data from Blast being sent to Pak2Bin, and then from Pak2Bin onward to Bin2Asc. Bin2Asc is the end of the pipeline, the data goes no further.

Blast moves the data from the USBxCH board to its output pipe. At that point the data is still raw USB packets, PAK data. Pak2Bin then converts the PAK data to easier to use binary BIN records. Bin2Asc then takes the BIN data and converts to Ascii files you can read with a text editor.

Additional details are required to have it work in real life. For example, you may want to run each utility in the background rather than the foreground. Or, maybe to add a pipeline stage to present a GUI display. See the [Pipeline utilities](#) chapter for more.

Despite the flexibility and ease of using pipelines, there are cases where saving data to files is preferable. Acquisition at high rates on slow host computers may not have enough bandwidth to do much more than save data directly to PAK files. Details of running in these modes are covered in [Fast acquisition](#).

The next chapter describes the finished apps, Ascii, DVM, Scope. Following that are the chapters covering custom programming.

Chapter 4

Acquisition Programs

The USBxCH comes with three finished acquisition programs: Ascii, DVM, and Scope. With these applications you can acquire data, display it on the screen, and save it to disk. Even for those planning to write their own custom software, these applications are a useful introduction to how the system works.

Each program fills a different set of requirements:

- **Ascii** acquires data and displays it as text columns on the screen. It is a bare bones example of how to use the pipeline utilities.
- **DVM** has a familiar digital voltmeter style display and is suitable for low frequency sampling rates. Data is also saved to disk.
- **Scope** presents its data as horizontal traces like an oscilloscope and is suitable for low and medium sampling rates. Data is also saved to disk.

All three programs use initialization files to specify their parameters. Users should modify the supplied sample ini initialization files to suit their particular needs. You can find the software for each program in these /SR/USBXCH/ distribution directories:

```
"App - Ascii"  
"App - DVM"  
"App - Scope"
```

The Windows versions of DVM and Scope have GUI displays. Under Linux the apps have text only console displays. The following sections give more details.

4.1 Ascii

Ascii acquires data with the USBxCH, displaying the results on the screen as text columns. It demonstrates how to use the pipeline utilities for realistic work. It is also a good way to become familiar with the USBxCH data fields.

To run Ascii, go to the directory:

```
/SR/USBXCH/"App - Ascii"
```

and execute the batch file `Ascii.bat`. You can execute by either double clicking on `Ascii.bat` in File Explorer, or by starting up a command window and running from the command prompt. It does not matter.

Once started you should see a screen like:

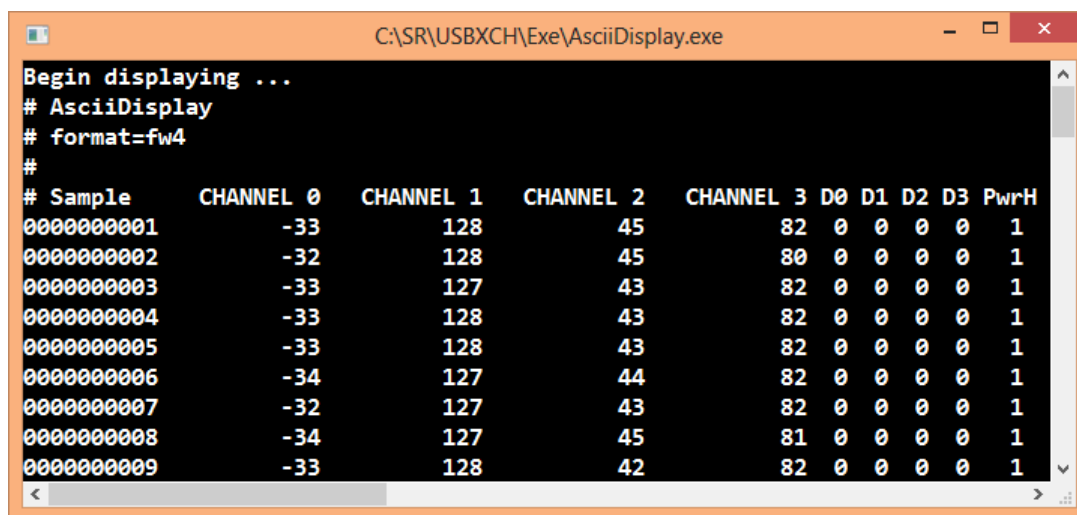


Figure 4.1: Typical Ascii display

If you scroll the window to the right there will be further columns with additional data fields for each sample point. Power history, temp, and time stamps are a few of the additional fields. Adjust your font size and window width to show the entire display if needed. Ctrl+s will pause/restart the display, Ctrl+c will close the window.

Ascii displays its analog values as counts. The number of counts is proportional to the input voltage. See [Analog DC calibration](#) for details. Use DVM if you want a display calibrated in volts.

Ascii does not save the data it acquires to disk files. See DVM or Scope for that additional functionality. Edit the batch file `Ascii.bat` to see how the pipeline utilities are used. To review `Ascii.bat` send it to your favorite text editor rather than double clicking on it.

4.2 DVM

DVM is an acquisition program for the USBxCH with a display and function much like a multichannel digital volt meter. If hand held instruments such as Fluke meters are familiar, then you will find DVM easy to use.

One feature of DVM is besides displaying values as A/D counts, it can display the analog inputs in other units such as volts, degrees, etc. Use the [Calibrate](#) program to compute the slope and offset coefficients for such displays.

DVM saves its acquired values to ASCII disk files. This makes it easy to analyze experiment results, import data into spreadsheets, etc. Parameters in the DVM ini file control the output file format.

Under Windows, DVM uses a graphic display. For Linux, the display is console text only. Both use the same ini keywords. Set the ini parameters to display 4 or 8 channels as appropriate for the USBxCH model you are using.

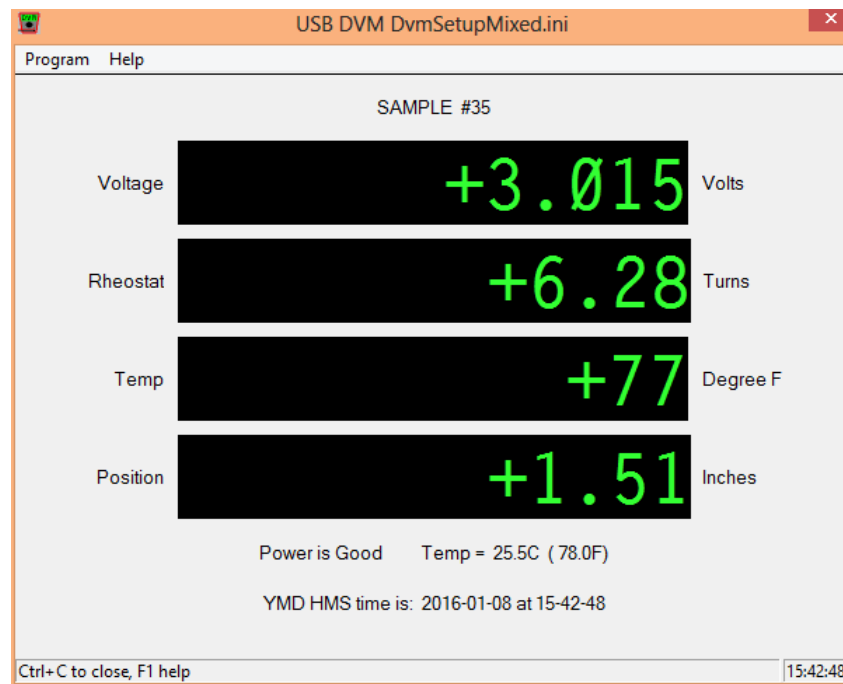


Figure 4.2: Typical Windows DVM display

The above screen shot is for a USB4CH four channel system. Each of the four readouts shows the value for a particular channel measured in the units on the right. Here each channel has been calibrated into physical units suitable for the sensor connected to that

channel. Channel 0 is showing volts, Chan 1 is calibrated into angular units as might be measured by 10 turn potentiometer, Chan 2 degrees from a temp sensor, Chan 3 linear units from a string gauge. You can calibrate as appropriate for a particular application and sensor.

The title bar of the screen shot shows the name of the DVM program, and the ini file currently in use. This is the ini file from which the parameters for the current run have been taken.

Below the title bar, the current sample number is shown. This number increments with time as successive samples are taken. The USBxCH samples all channels in parallel at exactly the same instant of time.

DVM defaults to sampling at a 1Hz rate, similar to a Fluke meter. You can set the ini parameters to sample at other rates if needed. Clearly a display like DVM is not suited to sampling at high rates. At 100Hz the numbers would pass by fast and not be of much use. At higher rates the **Scope** program provides a more useful display.

Across the bottom of the display the USBxCH power status and board temp are shown. The temp of the USBxCH itself is probably much different than that of a temperature sensor hooked up the system. Both board power and temp are important indications of the USBxCH system health and that things are running correctly.

Below the power and temp the current YMDHMS time is shown. Those letters stand for (year, month, day, hour, minutes, seconds) of when the sample was taken. Data from the current run will be saved in a YMDHMS directory created when the run was stated.

If you have a GPS antenna connected and ini parameters set correctly, the YMDHMS time will be the current GPS time. With GPS time, samples will be aligned with the PPS second tick and time stamped accordingly.

To kill the program and stop the run, type the speed key Ctrl+c. To begin a new run stop and restart the program. Ascii data from a particular run is saved in a YMDHMS directory created at the start of the run. The layout of the data file is much like the display of the "App - Ascii" program. Data fields for each sample point listed in columns. Several formats are possible and are selected by the ini paramters for the Bin2Asc stage of the DVM pipeline. The CSV (comma separated variables) format is particularly useful for importing into spreadsheets and data analysis tools.

DVM is a good tool for low frequency applications. It may meet all of your needs without any further programming. Feel free to modify Dvm.bat and the ini files as required. Many small tweaks are easily accomplished with only a few edits.

The following sections give more details about using DVM

4.2.1 DVM: starting the program

To run DVM go to the directory:

```
/SR/USBxCH/"App - DVM"
```

and execute the batch file `Dvm.bat`, either by double clicking on it in File Explorer or running from a command prompt. There are also shortcuts in the "App - DVM" directory to run with sample ini files. Double click on them for quick results.

An ini file may be specified on the `Dvm.bat` command line:

```
cmd:prompt> Dvm.bat [inifile]
```

If no ini file is given a default will be used. `Dvm.bat` does not recognize `/?` for help. Instead, refer to the underlying pipeline utilities and their `[/c,/i,/f]` options. See the [Pipeline Utilities](#) chapter for general info on how to use the utilities.

`Dvm.bat` is the executable for DVM, there is no `Dvm.exe`. The `Dvm.bat` file runs the underlying pipeline utilities comprising DVM. Symbolically the pipeline is:

```
Blast -> Pak2Bin -> Interp -> Bin2Asc -> DvmDisplay
```

Edit `Dvm.bat` with a text editor to see the details of how the pipeline is set up.

4.2.2 DVM: ini syntax

The pipeline utilities of DVM all look to the single ini file specified on the `Dvm.bat` command line for their initialization parameters. Within this shared ini file are `[sections]` with the parameters for each stage of the pipeline. Customize DVM by modifying these parameters with a text editor.

For info about the valid ini keywords and parameter values see the `[/c,/i,/f]` help screens for each pipeline utility. The [Pipeline Utilities](#) chapter has general information, or refer to the `Docs` directory in the software distribution.

4.2.3 DVM: output files

DVM creates a YMDHMS directory for each run and saves data there. The actual name of the directory will be something like:

2014-07-25-at-09-10-42

reflecting the (year, month, day, hour, minute, seconds) when the run was started.

The Bin2Asc stage of the DVM pipeline is responsible for creating the output files. Data files are Ascii text, with the format specified by the ini parameter:

```
[Bin2Asc]    OutputFormat = {fw4,fw8,cs4,cs8,none}
```

The "fw" fixed width formats are easy to read, while the "cs" comma separated values formats are useful for importing data into programs like spreadsheets. You can examine the output files in any text editor to become familiar with their formats.

File size is set by the Bin2Asc MaxLines ini parameter. It sets the number of sample points saved in the output file before starting another file. Filenames always have a sequential part, starting at:

00000000.ext

The eight decimal digits increment as you go from one file to the next. The extension to use is specified by the Bin2Asc OutputExt parameter, with .txt or .asc typical. You can also assign a filename prefix. There are no fixed rules for the extension and prefix.

Often users would like to save their data in a particular location. No doubt the first time you ran Dvm.bat in the "App - DVM" directory you created data and report files in the distribution directory.

To save data in a directory of your own called "c:/MyExperiment", simply change to MyExperiment and execute:

```
cmd:prompt> c:/sr/usbxch/"App - DVM"/Dvm.bat
```

and data will be saved there. You do not have to copy Dvm.bat to the experiment directory. Set up helper batch files to avoid retyping the path names.

4.3 Scope

In many ways Scope is identical to DVM, differing only in its output display. Where DVM displays each USBxCH channel as a readout, Scope displays its data as horizontal traces much like a multichannel oscilloscope. Scope is a good match for viewing the AC characteristics of your inputs.

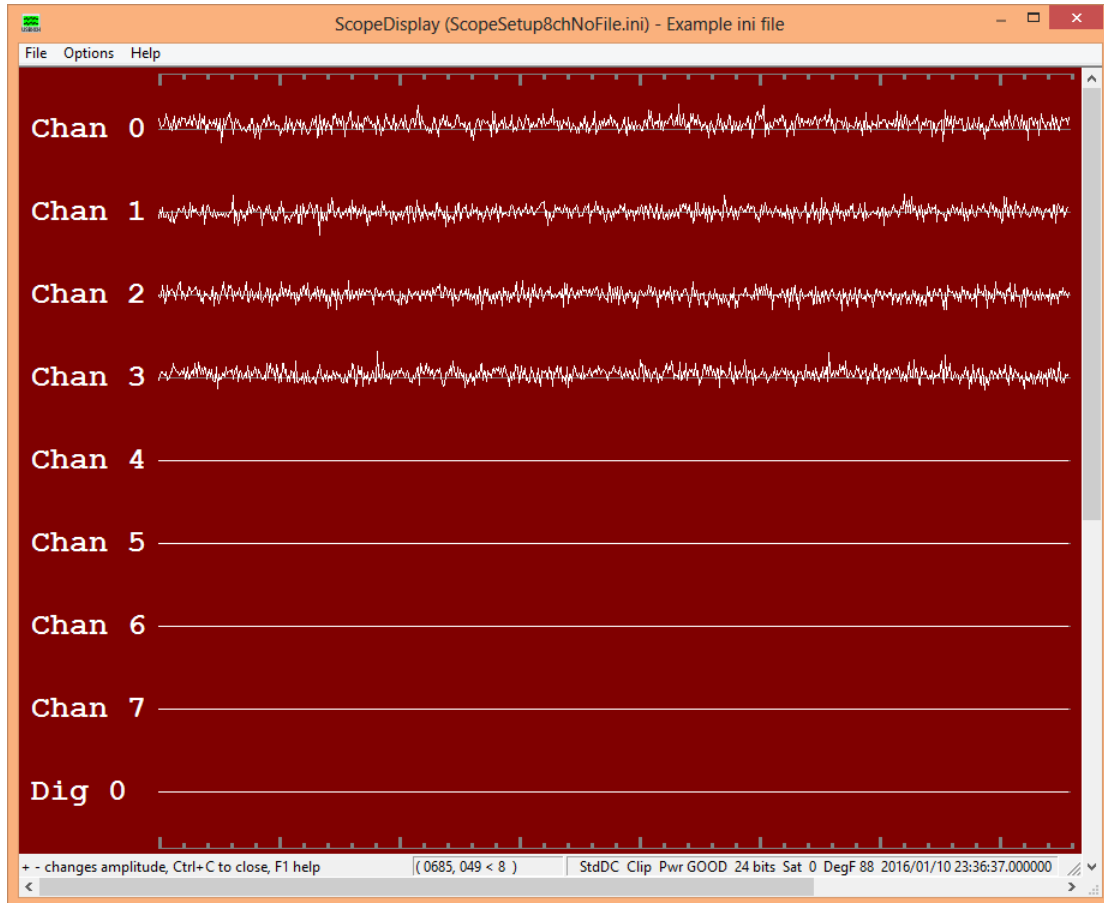


Figure 4.3: Typical Windows Scope display (showing noise floor)

The above screen shot is for the USB8CH eight channel system. The eight analog channels are displayed as a horizontal traces, with the display continually moving to the left one buffer at a time as new data is acquired.

Each horizontal pixel represents one sample point, while the vertical axes are measured in scaled A/D counts. The vertical scaling is controlled with the (+,-) or (up,down) arrow keys. When set to 24 bits, one vertical pixel represents one 24 bit A/D count.

This particular display was taken with all inputs shorted, and the vertical display set to 24 bits. It shows the typical noise floor. By decreasing the number of A/D bits shown with **down** arrow, you can come to a flat line and determine the number of single sample repeatable A/D bits the system has at a particular sampling rate. At lower sampling rates the sigma delta converters have more time to signal average and a lower noise floor. See the **noise floor** graphs in the Specifications chapter for more details.

Besides the analog input channels, Scope can also display other USBxCH inputs. Use the vertical scroll bar to bring them into view. Digital inputs, power history, GPS PPS time toggle, and GPS satellite count are among those available. Plotting these values along with the analog inputs can be useful checks on the validity of the data acquired. For example, if system power has browned out the analog A/D values are no longer valid. Customize the trace display for your application by setting your ini file parameters.

Information like the cursor position, current GPS time, and others are displayed on the status bar across the bottom of the screen. With the cursor position you can measure the amplitude of a particular analog signal. The items on the status bar give a quick view of the system status.

Scope saves its data to Ascii files. Much like DVM, a new YMDHMS data directory is created for each run. The data files are generated by the Bin2Asc stage of the Scope pipeline. For more information about the ini parameters controlling file output, see the Bin2Asc [/c,/i,/f] help screens. With the various formats provided by Bin2Asc you can easily import your data to post processing programs like spreadsheets.

To quit the program, use the speed key Ctrl+c.

The following sections give more details about using Scope

4.3.1 Scope: starting the program

To run Scope go to the directory:

```
/SR/USBxCH/"App - Scope"
```

and execute the batch file `Scope.bat`, either by double clicking on it in File Explorer or running from a command prompt. There are also shortcuts in the "App - Scope" directory to run with sample ini files. Double click on them for quick results.

An ini file may be specified on the Scope.bat command line:

```
cmd:prompt> Scope.bat [inifile]
```

If no ini file is given a default will be used. Scope.bat does not recognize `/?` for help. Instead, refer to the underlying pipeline utilities and their `[/c,/i,/f]` options. See the [Pipeline Utilities](#) chapter for general info on how to use the utilities.

Scope.bat is the executable for Scope, there is no Scope.exe. The Scope.bat file runs the underlying pipeline utilities comprising Scope. Symbolically the pipeline is:

```
Blast -> Pak2Bin -> Bin2Asc -> ScopeDisplay
```

Edit Scope.bat with a text editor to see the details of how the pipeline is set up.

4.3.2 Scope: ini syntax

The pipeline utilities of Scope all look to the single ini file specified on the Scope.bat command line for their initialization parameters. Within this shared ini file are `[sections]` with the parameters for each stage of the pipeline. Customize Scope by modifying these parameters with a text editor.

For info about the valid ini keywords and parameter values see the `[/c,/i,/f]` help screens for each pipeline utility. The [Pipeline Utilities](#) chapter has general information, or refer to the `Docs` directory in the software distribution.

4.3.3 Scope: output files

Scope creates a YMDHMS directory for each run and saves data there, similar to DVM. The actual name of the directory will be something like:

2015-03-08-at-08-05-46

reflecting the (year, month, day, hour, minute, seconds) when the run was started.

The Bin2Asc stage of the Scope pipeline is responsible for creating the output files. Data files are Ascii text, with the format specified by the ini parameter:

```
[Bin2Asc]    OutputFormat = {fw4,fw8,cs4,cs8,none}
```

The "fw" fixed width formats are easy to read, while the "cs" comma separated values formats are useful for importing data into programs like spreadsheets. You can examine the output files in any text editor to become familiar with their formats.

File size is set by the Bin2Asc MaxLines ini parameter. It sets the number of sample points saved in the output file before starting another file. Filenames always have a sequential part, starting at:

00000000.ext

The eight decimal digits increment as you go from one file to the next. The extension to use is specified by the Bin2Asc OutputExt parameter, with .txt or .asc typical. You can also assign a filename prefix. There are no fixed rules for the extension and prefix.

Often users would like to save their data in a particular location. No doubt the first time you ran Scope.bat in the "App - Scope" directory you created data and report files in the distribution directory.

To save data in a directory of your own called "c:/MyExperiment", simply change to MyExperiment and execute:

```
cmd:prompt> c:/sr/usbxch/"App - Scope"/Scope.bat
```

and data will be saved there. You do not have to copy Scope.bat to the experiment directory. Set up helper batch files to avoid retyping the path names.

Chapter 5

Pipeline Utilities

The pipeline utilities are the core tools for designing and modifying your own applications. Programs like DVM and Scope are built from them. If you wish to write custom applications for the USBxCH use the pipeline utilities.

Executables for all of the pipeline utilities are in the directory:

`/SR/USBXCH/Exe`

Add this directory to your execution path for easy access to them all.

Besides the pipeline utilities, other executables like the **Standalone Utilities** are also located in **Exe**. There are two ways to tell if an **Exe** program is a *pipeline utility* or not:

First, run any of the **Exe** executables with the `/c` command line usage option. If it is a pipeline utility, the first line of the usage help screen will contain one of the following:

```
[ PAK PIPELINE UTILITY ]  
[ BIN PIPELINE UTILITY ]
```

Second, refer to the documentation directory:

`/SR/USBXCH/Docs`

It is organized by utility type, with a complete listing of the available pipeline utilities.

The following pages review the basics of using the pipeline utilities in general, and give a short listing of three of the most commonly used ones.

5.1 Pipeline utilities: command line usage

The pipeline utilities are command line programs. With only a few exceptions, they all take the same command line arguments. There are three formats:

```
FORMAT 1: utility.exe [/c,/i,/f,/?,/h]
FORMAT 2: utility.exe inifile /syntax
FORMAT 3: utility.exe inifile /to nextprog
```

Command lines not following one of these formats will return an error. In more detail the allowed command line formats are:

Format 1: various help screens

```
/c = command line usage
/i = ini syntax and keyword listing
/f = functional reference

[/?,/h] = command line usage alternatives
```

These options give quick references on how and when to use a particular utility. Only one / at a time. The same documentation is in the `Docs` directory.

Format 2: inifile /syntax

Utility parameters beyond the input and output pipes are specified in an inifile. Before actually running the pipeline use the `/syntax` option to check for INI errors. Errors must be fixed before running the pipeline. The order of `/syntax` and inifile does not matter, `"/syntax inifile"` is fine.

Format 3: inifile /to nextprog

Use this format when ready to run. The name of the input pipe is fixed to be that of the utility being executed. The name of the output pipe is specified by the `"/to nextprog"` argument. The keyword `"NULL"` can be used for `nextprog` when you wish to terminate a pipeline.

Both the input and output pipe names are automatically decorated with a prefix of `SrPipe` before being opened at the operating system level. You do not have to specify the decoration.

Exceptions to these formats are usually obvious. `PakTee` and `BinTee` are examples, each requiring two output pipes. When in doubt, run with `/c`, `/i`, or `/f`.

5.2 Pipeline utilities: INI files and syntax

Utility parameters beyond the input and output pipe names are all specified in an INI file. This reduces the amount of clutter on the command line, and within batch and script files keeps the focus on how the pipelines are strung together.

INI files are ASCII text. It is customary to use a `.ini` filename extension, but not required. An inifile must be specified for every utility. With the section name construct discussed below, a single inifile can be shared between several utilities.

Comments within an inifile are denoted by a semicolon `;`. A comment starts at the semicolon and extends to the end of the line. The comment can start anywhere on a line, it does not have to begin in the first column.

All inifiles *must have a header id line*:

```
SR_USBXCH_Ini ; <<< Required INI header id line
```

The header must come before any section names. If a utility does not see the header line it will reject the file as an invalid inifile and give an error. Any number of comments and white space can come before the header.

Following the header line are *section names*. Every utility looks for a section with its name. Section names are denoted with square brackets `[.]`. An inifile can have multiple sections, and the same inifile can be shared with multiple utilities. Using a single inifile file makes it easier to manage all the utility parameters across an application.

A fragment of an inifile might look like:

```
SR_USBXCH_Ini ; ini header id line

[Blast]
    DriverName = "SrUsbXch0" ; identifies A/D board
    SampleRate = "s1"        ; select the USBxCH sampling rate

[Pak2Bin]
    Verbose = 0              ; limit the amount of reporting

[AsciiDisplay]
    OutputChannels = 8       ; display 8 analog channels
    RepeatTitle = 25         ; repeat column titles after 25 lines
```

The parameters coming after the section name `[Blast]` are picked up by `Blast.exe` when it runs, and likewise for the other `[.]` sections. For a listing of the ini keywords recognized by any particular utility, run `"utility.exe /i"` or see the Docs directory.

5.3 Pipeline utilities: error reporting (.rpt files)

Let's admit it ... we all write bad code sometimes. The computer can have its problems too. Perhaps running out of disk space or a power failure. The pipeline utilities report their errors to "utility.rpt" files. A report file is generated by each utility.

Errors *are not combined* into a single report file. For a typical application with several utilities running at once, combining errors creates a tangled mess impossible to peel apart. Single files associate an error with a particular utility with certainty.

Besides the .rpt files, error reporting may also come to the command line screen with printf. An example is running a /syntax check on your INI files. The on screen printf reporting makes it easy to find and fix the errors.

When running an app, always review the .rpt files to make sure everything is running correctly. *It may be the only signal something has gone wrong.* If you have trouble, zip up the .rpt files from a bad run into a single file and email them to us.

5.4 Pipeline utilities: running by hand, infinite loops

Every pipeline utility has an infinite loop!

Each waits for data to appear on its input pipe, processes it, and then passes the result on to its output pipe. They wait over and over for work to do, always faithful engines.

This means a command as simple as:

```
cmd:prompt> Interp interp.ini /to NULL
```

run by hand from the command line in the foreground will never return for the next command. New users may have a *moment of panic* ... don't worry. To break the infinite loop under Windows do one of the following:

- press: Ctrl+C (mild)
- press: Ctrl+Break (stronger)
- click the big x in the upper right hand corner of the Window
- start another command window, run tasklist then taskkill /f /im
- start the GUI TaskManager and kill it there

with similar steps under Linux. If you indeed want to run several pipeline utilities by hand in the foreground, one way is to start each one up in its own command window. The result will be several command windows on your screen, each running its own infinite loop. Data will flow through each, and it will work!

However, a better way is to start each utility in the background while working in a single command line window. Do this by executing:

- under Windows: `start /b Interp interp.ini /to NULL`
- under Linux: `Interp interp.ini /to NULL &`

A leading "`start /b`" is the Windows way of running in the background. Under Linux a trailing "&" marks the task for background execution. Both return to the command line once the task is running. Use `tasklist` and `taskkill`, or `ps` and `kill` under Linux, to further manage processes.

Even better yet, set up a batch or script file to automate starting all your processes in the background as in the next section ...

5.5 Pipeline utilities: batch/script files, background tasks

The best way to set up a pipeline is from a batch or script file. Background tasks started from the batch file are all children. Killing an entire app can be done by terminating the single parent batch file. Under Windows a typical batch file fragment might look like:

```
@echo off

rem Set one common INI file, and start up background processes:

set INIFILE=Myfile.ini

start /b          Blast.exe %INIFILE% /to Pak2Bin
start /b          Pak2Bin.exe %INIFILE% /to AsciiDisplay
start            AsciiDisplay.exe %INIFILE% /to NULL

echo Pipeline started ... see *.rpt files for info ...
```

For an expanded version of this see `"/SR/USBXCH/App - Ascii/Ascii.bat"`.

With this kind of approach most applications can be programmed from a batch file, all without any compiling. Pipeline stages can be added to process the data as you would like. See the `Docs` directory for a complete listing of the available utilities.

Almost all pipelines will start with `Blast.exe`. Its job is to move acquired data off the USBxCH board onto the host PC as quickly as possible. Succeeding pipeline stages then take care of further processing the data.

5.6 Pipeline utilities: controlling data file locations

The pipeline utilities do not take long path names. Generally they only work with local filenames. Output files are always emitted to the local current working directory, which is the directory from which the utility is executed.

To set the working directory, `cd` to it at the top of your batch file. Your batch file *does not have to be in the working directory*. Just `cd` there and then execute.

Finished apps like DVM and Scope are examples of this. They create a new "YMDHMS" directory for each run to hold the data from that run. After creating the data directory, they immediately `cd` there before executing their pipeline. This pattern not only keeps the data from a particular run together, it also keeps the report files together in one place.

5.7 Pipeline utilities: short reference

Symbolically, one of the most basic pipeline chains is:

```
Blast -> Pak2Bin -> Bin2Asc
```

Blast sets the basic acquisition parameters like sampling rate etc, and then steadily moves USBxCH PAK records to the host computer at a constant rate. **Pak2Bin** then converts the Blast USB packet data into easier to use binary BIN records. To finish, **Bin2Asc** formats and saves the BIN records into ASCII files.

The next pages describe these three basic utilities in more detail:

| | |
|----------------|--|
| Blast | move USBxCH packets to the host computer |
| Pak2Bin | convert PAK data to BIN records |
| Bin2Asc | create files from BIN records |

For additional utilities to use in the later stages of the pipeline see the Docs directory in the software distribution.

For examples of how to add pipeline stages for more complete programs, see apps like DVM and Scope. Additional examples can also be found in `/SR/USBXCH/Examples`.

5.7.1 Pipeline utility: Blast

Blast is "the core" pipeline utility and the first stage in almost all USBxCH applications. Its job is to set basic parameters like the sampling rate, and then move raw USB packets from the USBxCH to the host PC as quickly as possible with a minimum of features.

There is no numerical processing or GUI display in Blast. Downstream processing on Blast PAK data is the job of other pipeline utilities. For example ... add pipeline stages like `Interp` or `ScopeDisplay` for more features.

To run Blast and send PAK data to its output pipeline, use the command line:

```
cmd:prompt> Blast.exe inifile /to nextprog
```

Blast runtime parameters are specified with inifile keywords as covered next.

If host cpu time is limited and you cannot run a downstream pipeline, or if you wish to save the original PAK data as part of the permanent record, Blast can optionally save its data to files. See the keyword `PakFileSize`.

Blast: INI keywords

The leading `[Blast]` ini section keywords are:

`SampleRate = "sn" ; set the sampling rate, where "sn"`

| | |
|-----|----------------|
| s1 | 19.531250 Hz |
| s2 | 32.552083 Hz |
| s3 | 39.062500 Hz |
| s4 | 65.104167 Hz |
| s5 | 78.125000 Hz |
| s6 | 130.208333 Hz |
| s7 | 651.041667 Hz |
| s8 | 1302.083333 Hz |
| s9 | 2604.166667 Hz |
| s10 | 4882.812500 Hz |
| s11 | 9765.625000 Hz |

These are the only native rates supported by the USBxCH. If other frequencies like 100Hz are required use utilities like `Interp.exe` to downsample from higher rates. See [Sampling rates](#) for more details.

[Blast] ini section keywords continued ...

GpsName = "gn" ; set the GPS timesource, where gn is

| | | |
|----|----|---------------------------------------|
| g0 | -> | no timing |
| g1 | -> | timing taken from PC time << default |
| g2 | -> | timing taken from Garmin GPS antenna |
| g3 | -> | timing taken from Trimble GPS antenna |
| g4 | -> | timing taken from Meinberg GPS clock |

The "gn" specification must agree with the type of GPS antenna connected to the system. If PC time "g1" is specified, it will override GPS, *even if a GPS antenna is connected to the system.* See [GPS Time Stamping](#).

PakFileSize = "size" ; set PAK file size, where size is

| | | |
|--------|----|--|
| None | -> | no file saving << default |
| Little | -> | PAK file size = approx 1-5 seconds of data |
| Big | -> | PAK file size = approx 10 minutes of data |

Sequential files are created. Small files put less data at risk if the host computer crashes. Larger files result in fewer filenames and less burden on the disk directory system. See [Fast acquisition](#).

DriverName = "SrUsbXch0..9" ; set the driver name

If you are using multiple USBxCH boards with a single PC, each will have its own driver named (SrUsbXch0..9). Specify the board you wish Blast to acquire data from by giving its driver name. See [Multiple USBxCH](#).

UserConfigByte = "0xNN" ; set the digital IO config

UserConfigByte should be a two digit C style hex number, like 0x14. It selects various Digital IO and GPS features. See the [User configuration byte](#) section in the Digital IO chapter.

When starting up, Blast displays a summary report of the parameters it is running with. This gives you an opportunity to verify the system is actually using the values specified in the INI file. A copy of the summary report is also placed in the "Blast.rpt" file.

For a complete listing of the Blast ini keywords run "Blast /i", or see the Docs directory. For an example of using Blast see programs like "App - Ascii".

Blast: PAK data format

Data is transmitted from the USBxCH to the PC in 64 byte packets over the USB port. Whether on the USB cable, in a pipeline, or in files, PAK data is comprised of these raw USB packets. The format is dense, highly multiplexed, and requires significant decoding. Although users may save PAK data directly to disk, its format is proprietary and is not published.

Users should convert Blast PAK data to BIN format with the `Pak2Bin.exe` conversion utility. Although binary, **BIN data** structures are fully resolved and time stamped with one record per sample point. Each BIN structure has all the analog inputs, digital inputs, system temp, and time stamping for that sample point. All synchronously sampled.

When BIN records are stacked vertically one record at a time, the BIN data fields form columns. Viewed this way, BIN data is sometimes referred to as "**Binary column**" format. For a visual presentation run the "**App - Ascii**" application.

If you have saved your data to PAK files, you can reconstitute a Blast PAK pipeline with the `PakFread.exe` utility. It takes PAK files as input and sends them to an output pipe. At that point it is indistinguishable from Blast. There are many low level utilities to manipulate both PAK and BIN pipeline data, sending data to and from files, creating pipe tees, etc. See the `Docs` directory in the software distribution.

5.7.2 Pipeline utility: Pak2Bin

Pak2Bin.exe converts Blast PAK data to easier to use **BIN data** structures. A BIN structure has all the USBxCH data fields for a sample point time stamped and ready to go for further downstream processing.

Pak2Bin usually appears early in a pipeline sequence:

```
Blast -> Pak2Bin -> further BIN processing
```

Once in BIN format, there are many numerical, real time display, and file saving utilities for further processing. See the sections earlier in this chapter for general **Pak2Bin** command line usage and typical batch file fragments.

Pak2Bin: INI keywords

The leading **[Pak2Bin]** ini section keywords are:

```
OutputCfgFile = {0,1}    ; << default = 1
OutputStatusFile = {0,1} ; << default = 1
OutputNmeaFile = {0,1}   ; << default = 1
OutputEquipFile = {0,1}  ; << default = 1
```

These parameters select the intermediate report files **Pak2Bin** generates, where 0 or 1 turns a particular file off or on.

For example, if a GPS antenna is connected to the system, it will not only time stamp data, but also generate geographic location info once per second. Typically this info is too long and redundant to include with each analog sample that occurs between the one second intervals. Particularly at analog sampling rates like 10kHz!

To reduce such redundancy, **Pak2Bin** saves location data to a separate intermediate file in the raw GPS NMEA format at the 1 second (PPS) rate. You can control saving this file by setting the **OutputNmeaFile** keyword on or off.

Intermediate files are saved in the current execution directory. The files are not very large, growing slowly with time. We recommend leaving them on. They are useful debugging tools when things go wrong. The intermediate file formats are simple Ascii text which can be viewed with any text editor.

For a complete listing of the **Pak2Bin** ini keywords run "**Pak2Bin /i**" or refer to the **Docs** directory. For an example of using **Pak2Bin** see programs like "**App - Ascii**".

5.7.3 Pipeline utility: Bin2Asc

`Bin2Asc.exe` converts **BIN data** into Ascii text files. The files have all the data fields for each sample point organized as rows, an Ascii version of the binary BIN data.

The data fields include the analog and digital inputs, GPS time stamping, system temp, power condition, etc. If acquiring at 100Hz, there will be 100 rows per second with the data fields listed across each row.

There are two basic file formats to choose from, "`fw`" and "`cs`". Fixed width is easy to read in a text editor, while comma separated values are useful for importing into spreadsheets and similar. Format selections are made in the `[Bin2Asc]` ini section.

Data can also be scaled into user units with a (`slope,offset`) straight line conversion from A/D counts into user units. Scaling parameters are specified in a `[Calibrate]` ini section which can be shared across many of the pipeline utilities.

Bin2Asc: INI keywords

The leading `[Bin2Asc]` and `[Calibrate]` ini section keywords are:

```
[Bin2Asc]

OutputFormat = {"fw4","fw8","cs4","cs8","none"}
OutputPrefix = String
OutputExt = String
Title# = String
Places# = N
Digits# = N

[Calibrate] ; shared among several utilities ...

Units# = String
Slope# = F
Offset# = F
```

Parameters with a `#` symbol are per channel. Specify (`Title0`, `Title1`, ...) for the number of USBxCH channels you have. `N` is an integer, `F` is a floating point number.

For a complete listing of the `Bin2Asc` keywords run "`Bin2Asc /i`" or refer to the `Docs` directory. For an example of using `Bin2Asc.exe` see programs like "`App - DVM`".

Chapter 6

Standalone Utilities

This chapter describes the USBxCH standalone utilities. They perform a number of system level housekeeping tasks. Checking the hardware, setting PC time from GPS, and twiddling digital io bits are a few of these tasks. When you need something from the system other than acquiring data, these are the tools to use.

The leading standalone utilities are:

| | |
|--------------------|--|
| Diag | general system hardware diagnostic |
| SetDid | set the USBxCH EEPROM DID for multiboard installations |
| EquipStatus | report the system power and temp status |
| GpsProg | program a GPS antenna through the USB port |
| NmeaTime | report the NMEA strings / set PC time for systems with GPS |
| DigitalIo | twiddle the digital IO bits |
| Calibrate | compute (slope,offset) conversion into user units |
| Presence | test for the presence of a USBxCH system |
| YmdHms | create a YMDHMS data directory |

Executables are in the `/SR/USBXCH/Exe` directory. They are all command line oriented programs with options.

Run with `/c` or `/f` for help screens, or refer to the `Docs` directory for the utilities included with your software distribution. See the `Tools` directory for shortcuts and config files.

The following sections cover the leading standalones in more detail.

6.1 Diag

Diag is the swiss army knife of USBxCH hardware maintenance. Use it to verify correct operation after installation or whenever else the system needs to be checked. Diag is a command line program, with two command line formats:

```
FORMAT 1:  Diag [/c,/f]
FORMAT 2:  Diag test [gn] [driver]
```

"Format 1" displays help information. The /c option gives command line usage, while /f displays functional reference info. Use "Format 2" to actually run the program, where the arguments are `test`, `[gn]`, and `[driver]`:

`test`: specifies a particular diagnostic to run

The following are a few of the available tests. Run with /c to see the full list:

```
install    -> Installation test suite (dev,power,led,analog,nmea)
show       -> Show all active USB devices
dev        -> Show USB device descriptor
reset      -> Hardware reset USBxCH board with full power cycle
rev        -> Get firmware revision number
power      -> Check if power is good
led        -> Toggle the yellow LED
analog     -> Start A/D's and display analog data
nmea       -> Start A/D's and display NMEA strings
equip      -> Start A/D's and display equipment info like temp
digin      -> Read digital input
digout     -> Send digital output
dram       -> Run several DRAM FIFO tests
```

`gn`: optional time source specification

If not specified, the PC clock `[g1]` will be used by `Diag` as the time source. See [GPS Time Stamping](#) for more information. Among the allowed time sources are:

```
g1  -> PCTIME    ( NMEA = ZDA )      << default
g2  -> GARMIN    ( NMEA = RMC, GGA )
g3  -> TRIMBLE   ( NMEA = ZDA, GGA )
g4  -> MEINBERG  ( NMEA = RMC )
```


Diag command line options continued ...

driver: optional USBxCH board and driver specification

Specify this parameter if you have multiple USBxCH boards connected to a single PC and wish to run **Diag** on a particular one. For a list of all the systems attached to the PC run "**Diag show**".

All systems must have one of the names (**SrUsbXch0..9**). System names must be unique when attached to the same PC. See **SetDid** for programming the USBxCH EEPROM with a particular board name.

If not specified, the **Diag driver** parameter defaults to **SrUsbXch0**. The USBxCH EEPROM is programmed at the factory with **SrUsbXch0**.

As part of any new installation, run:

```
cmd:prompt> Diag install
```

to confirm the hardware and driver is correctly installed. If the system develops a hardware problem after running successfully for awhile, then run:

```
cmd:prompt> Diag debug
```

and **email** us the file "**SrRpt-diag.rpt**" for technical help.

6.2 SetDid

SetDid programs the USBxCH EEPROM USB device identifier (DID). This is required only for multiboard installations to uniquely identify each system. Users running single board installations do not need to deal with setting the DID. Use this utility only to prepare several USBxCH systems to run with a single PC.

The SetDid command line syntax is:

```
cmd:prompt> SetDid [driver] [0-9] [show] [/c] [/f]
```

where [] indicates the parameter is optional. Running with /c will give a command line usage message. The other command line parameters are:

driver: specifies an existing USBxCH board to program with a new DID. It must be one of (**SrUsbXch0..9**). The board driver name will change accordingly when programmed with a new DID.

0-9: new DID value to program the board EEPROM with. The new DID must be a single digit (0 ... 9). After programming, the board will have the new driver name **SrUsbXchN** where N is the new DID number. If you want to program the board back to its original DID, you must specify the new **driver** name when calling SetDid.

show: scans the PC for all **SrUsbXchN** that are installed. A list of who responds and who doesn't is displayed. Use the **showall** option before and after programming to check the results are what you expected.

USB devices are identified by three numbers, their (VID/PID/DID) numbers, standing for the *vendor*, *product*, and *device* id respectively. The VID and PID are fixed for a particular vendor and product, while DID numbers are free to change as needed. Sometimes the DID is used to indicate a firmware revision, however it may also be used to distinguish between multiple devices of the same type connected to a single PC. The second meaning is used with the USBxCH, where up to 10 boards may be connected to a single PC at a time. The multiple devices are referred to as **SrUsbXch0**, **SrUsbXch1**, etc where the trailing digit is the same as the DID number.

The USBxCH product is shipped from the factory with its on board EEPROM DID set to 0. This is suitable for computers with a single USBxCH attached. All of the ini files supplied with software like Scope and Blast are set up to use the device driver name **SrUsbXch0**.

If you wish to run multiple USBxCH on a single PC, *attach one at a time to the machine* and run SetDid to specify the new DID you wish each to have. Then create the corresponding ini initialization files or command line arguments to refer to the particular board you desire.

This way you can run multiple instances of DVM, Scope, etc with each referring to its respective USBxCH board.

You cannot have two boards with the same DID value connected to the same PC at the same time. The second board will fail to install. In this situation, connect the boards one at time, programming each with a unique DID number, and try again. If you forget the DID number of a particular board, run `SetDid showall` to scan for the board and relearn its identifier.

For step by step instructions on setting up a multiboard installation, see the Examples and Experiments section [Multiple USBxCH](#).

6.3 EquipStatus

EquipStatus reports the USBxCH power status and temperature. It is a "health status" utility for use from the command line, batch, and script files. You do not need to use EquipStatus to get a continuous record of power and temperature. DVM, Scope, etc automatically record the power and temp history in their data files.

The EquipStatus command line syntax is:

```
cmd:prompt> EquipStatus [forever] [driver] [/c]
```

where [] indicates the parameter is optional. Running with /c gives a command line usage message. The other command line parameters are:

forever: indicates displaying power status and temperature over and over once a second, or until a 'q' keypress is made. If forever is not specified, power and temp will be returned once.

once: display the power status and temperature once and then quit.

driver: if you are running a multiboard system, specify the name of the USBxCH board EquipStatus should use. It must be one of (SrUsbXch0..9). If not specified the utility will default to SrUsbXch0.

For use with batch files, EquipStatus returns an errorcode of 0 if power is good and 1 if not. Use this errorcode to set environment variables in batch and script files. The power status and temp are also displayed on the screen with printf statements.

The power reported by EquipStatus reflects the condition of the USBxCH power supply. The most likely cause of bad power is the off board power supply, perhaps a battery, is less than the required 7.5 volts. Below 7.5 volts the system will still communicate with the USB bus, but the analog front end will start to fail. See the [Power Supply](#) chapter for more details.

EquipStatus reports the system temperature as both °C and °F. The temp is determined by a dedicated chip located near the USB controller on the board. After executing for awhile, the board temperature is uniform and the reported temp is the same as that of the entire USBxCH. See the [Temp Sensor](#) chapter for more details.

For an example of using EquipStatus see [Powering with batteries](#) in the Examples and Experiments chapter.

6.4 GpsProg

GpsProg is a command line utility for programing GPS antennas. The Garmin antennas supplied with SR USBxCH orders are already programmed correctly at the factory and are ready to go. You do not need to run GpsProg on them. Customer supplied antennas may require programming.

With GpsProg you can:

- select the NMEA strings emitted by the antenna
- enable the PPS (pulse per second) signal
- get the antenna firmware revision

With GpsProg *DO NOT*:

- set the antenna NMEA RS232 baud rate
- download new antenna firware

GpsProg works by sending NMEA strings to the antenna to program its firmware. Only certain features defined by the NMEA committee are programmable this way. Baud rate is not one of them.

To set the baud rate or download antenna firmware you must use a utility program from the antenna manufacturer (which is usually proprietary). The USBxCH software distribution includes the Garmin utility for the antennas we supply. If using a different antenna, consult its documentation for setting the NMEA baud rates, etc.

The USBxCH expects the RS232 baud rate for its NMEA strings to be set to 4800 baud. Other baud rates are not supported. The antenna *must be set to 4800 baud*.

Garmin antennas supplied by SR as part of USBxCH orders are already programmed with the correct baud rate, NMEA strings, and PPS. Changing these parameters may break programs like DVM and Scope, returning incorrect time stamping.

Once your antenna is programmed, use the **NmeaTime** utility to check its NMEA string lineup when connected to the USBxCH. Confirm it agrees with that described in the **GPS Time stamping** chapter. Also make sure the PPS is active by watching the USBxCH front panel red led toggle at the 1 second rate.

See the software distribution directory **Tools/GpsProg** for config files to use with GpsProg. Run **GpsProg /c** for command line usage.

6.5 NmeaTime

NmeaTime reads and displays NMEA strings from the GPS antenna. *It is not required for time stamping analog data.* Programs like DVM, Scope, etc automatically time stamp data with higher accuracy because they also process PPS. Use NmeaTime only for checking the GPS antenna and optionally setting the PC clock.

The NmeaTime command line syntax is:

```
cmd:prompt> NmeaTime [gn] [forever] [set] [0xUC] [driver] [/c] [/f]
```

where [] indicates the parameter is optional. Run with /c to display command line usage. The other command line parameters are:

"gn": specifies the time source to display:

```
g1  -> PCTIME    ( NMEA = ZDA )      << default
g2  -> GARMIN    ( NMEA = RMC, GGA )
g3  -> TRIMBLE   ( NMEA = ZDA, GGA )
g4  -> MEINBERG  ( NMEA = RMC )
```

Note that even with a GPS antenna attached to the system NmeaTime defaults to the PCTIME source. The correct time source *must be specified to display the NMEA strings from that source.*

forever: displays NMEA strings continuously until a 'q' keypress is made. If not specified, display stops after the first string with a valid satellite count is received.

set: sets the PC time from the first valid NMEA string. The time will only be accurate to the 1 second NMEA string rate. NmeaTime can be scheduled as a task with the operating system to reset the PC clock periodically. For precision PPS time stamping use DVM, Scope, etc.

0xUC: sets the User Configuration Byte. You may require the UC byte if you are running from an antenna with non standard signals. An example would be for an antenna with inverted RS232 polarity. See [GPS Time Stamping](#).

driver: for multiboard systems, specify the name of the USBxCH board NmeaTime should use. It must be one of (SrUsbXch0..9). The default is SrUsbXch0.

Note that if the GPS antenna has been programmed to emit more or different strings than are expected by one of the NmeaTime "gn" options, then it will not work correctly. You may get truncated versions of the strings if there are too many, or the PC time may not

be set because there is not enough time information in the strings you are emitting. *Check the results of NmeaTime before assuming they are correct.*

NmeaTime *cannot be run concurrently* while DVM or Scope is running. Only one program at a time can have the USBxCH device driver open, and other programs are excluded.

The NmeaTime **set** option is *not intended* to time stamp analog data from DVM or Scope. It is intended for those who wish to set their PC clock to one second accuracy while booting, or at other periodic times, as an independent stand alone task. Scope and Blast will automatically time stamp data as part of their normal operation.

6.6 DigitalIo

DigitalIo is a small command line utility for twiddling the USBxCH front panel DB25 digital IO pins. After starting up, the user is prompted for output values and they are written to the port. Besides programming the output pins, the four DB25 input pins are read and displayed on the screen as part of programming each value. For the pin number locations, see the [DB25 pin assignments](#).

The DigitalIo command line syntax is:

```
cmd:prompt> DigitalIo.exe [driver] [/c]
```

where **driver** is optional and indicates the particular USBxCH to program. It must be one of (SrUsbXch0..9) for multiboard USBxCH installations.

Output values are specified as single hex digits. A value of **3** would result in the bit pattern (0,0,1,1) with the least significant bit on the leftmost DB25 pin, while a value of **F** would result in (1,1,1,1). The values on the pins are latched, with the same value remaining until the next value is specified.

Note that programs like Scope and Blast *synchronously save the digital inputs along with the analog inputs* automatically. The effect is the same as a mixed signal digital oscilloscope. For most applications you do not need the DigitalIo utility to save the digital inputs.

6.7 Calibrate

Sometimes users are surprised to learn A/D converters do not output their results as volts. The output from an A/D converter is a digital integer that is only *proportional* to the input voltage. The conversion from this digital integer to units such as volts is referred to as the calibration.

The concept of calibration for an A/D converter can be carried even further into physical sensor units. Suppose you have a sensor like a potentiometer or temperature gauge. At the minimum setting the sensor may result in a particular A/D count, and likewise at its maximum setting another count value. These two points define a line, and conversion of the count values directly into sensor units, such as potentiometer turns or °C can be done with a linear transformation having a *slope* and *offset*.

The Calibrate program makes it easy to obtain the slope and offset coefficients for such transformations. It allows you to record the transducer min and max settings, and then outputs an ini file fragment for use with DVM, Scope, etc.

Calibrate is a command line program, run "`Calibrate.exe /c`" for command line usage. Programs like DVM and Scope are provided with default calibration slope and offset in their distribution ini files. However, because all A/D converters, references, and resistors have tolerances, calibration must be performed on each individual system and channel for precision results.

Absolute precision calibration requires a lab grade reference with repeatable fixed voltages. For such applications, the SR VREF-399 heater stabilized precision reference may be useful. See www.symres.com. Applications not requiring absolute calibration may find ratiometric techniques useful and not need calibration at all.

For demos of absolute calibration, calibration into physical sensor units, and ratiometric technique see the [Examples and Experiments](#) chapter.

For a discussion of the number of A/D counts per volt refer to [Analog DC calibration](#).

6.8 Presence

Presence determines whether or not a particular USBxCH board is connected to the system. It is a "helper utility" for batch and script files. The command line syntax is:

```
cmd:prompt> Presence.exe [driver] [/c] [/f]
```

Run with /c for command line usage. If testing for a particular board specify the **driver** to be one of (SrUsbXch0..9). If not specified, **Presence** will default to searching for SrUsbXch0.

If the USBxCH board is found, **Presence.exe** will return an errorcode of 0. If the board is not found, errorcode will be 1. Check the errorcode in batch files to take appropriate action. Usually that will be to exit if no board is found.

See DVM.bat or Scope.bat for examples of using **Presence.exe**.

6.9 YmdHms

YmdHms is a "helper utility" for batch and script files. Programs like DVM.bat and Scope.bat use it to create their data directories.

The acronym YMDHMS stands for (year, month, day, hour, minutes, seconds). Where a particular instance might be: 2015-01-15-at-07-46-23. In normal conversation you might refer to this date as: January 15, 2015 at 7:46AM. Directories with names in the YMDHMS format sort nicely based on the name alone, a big advantage ...

When **YmdHms.exe** runs it makes a directory and creates a short batch file. Inside the batch file is a "set" command to create an environment variable with the current YMDHMS name. Larger batch files can call this "helper batch" to get the name of the current working data directory. See DVM.bat and Scope.bat for examples.

If you have a GPS antenna, use the "NmeaTime set" utility to automatically set the PC time accurately when starting up, and guarantee the YMDHMS data directories have the correct date.

Chapter 7

BIN data structure layout

The binary BIN data structure is the format of most USBxCH pipeline data. There is one structure for each sample point and a time series is a sequence of BIN structures.

BIN structures have the analog inputs for each channel, the digital inputs, time stamps, and many other USBxCH data fields for that sample point. The data fields for each structure are all sampled at the same instant of time.

Opening a BIN output pipe from a SR pipeline utility and bringing data into your own code is relatively easy. See the [Examples and Experiments](#) chapter for demos. Generally, you will open and read the output pipe from an upstream SR utility and process BIN structures one at a time in your code.

The C code layout of a BIN structure is given on the next page. The following comments describe the structure in more detail.

First, the width of each data type is:

| | |
|----------------------|-----------------------------------|
| <code>char</code> | 1 byte |
| <code>int</code> | 4 bytes (32 bit signed integer) |
| <code>SRINT64</code> | 8 bytes (64 bit signed integer) |

Second, fields are laid out to maintain alignment to 64 bit integer boundaries. When you see a char field, there are four of them grouped together to fill out a 64 bit integer, etc. Keeping aligned in this way is important for portability across operating systems and current 64 bit processors.

(the text continues after the C code structure on the next page ...)

```

// BIN structure layout:

#define BIN_ID      (0x31425253L)    // = 'SRB1'
#define BIN_REV     (0x00000001L)    // Bin layout Rev 1

#define MAXCHANNELS      8           // max num channels for any USBxCH

struct BIN {

    int      Id;                     // BIN ID field
    int      Rev;                    // BIN Rev number

    SRINT64 Sample;                  // USBxCH sample count

    int      Analog[MAXCHANNELS];    // supports both USB4CH and USB8CH

    int      reserved_1;
    int      reserved_2;
    int      reserved_3;
    int      reserved_4;

    int      PpsToggleCount;         // PPS count
    int      reserved_5;

    char      reserved_6;
    char      DigitalAll;             // four digital inputs
    char      PpsToggle;             // PPS square wave
    char      reserved_7;

    char      VoltageGood;            // power history
    char      NumSat;
    char      reserved_8;
    char      reserved_9;

    int      Lat10K;                  // Latitude times 10000 +N,-S
    int      Lon10K;                  // Longitude times 10000 -W,+E
    int      Alt10K;                  // Altitude times 10000 in meters
    int      DegC2;                   // Degrees C times 2

    int      reserved_10;
    int      reserved_11;

    SRINT64 reserved_12;
    SRINT64 reserved_13;
    SRINT64 reserved_14;
    SRINT64 reserved_15;
    SRINT64 reserved_16;
    SRINT64 reserved_17;

    SRINT64 TimeStamp;                // time in nanoseconds since 1970

    ... };

```

Third, fields marked as reserved are not published. *Do not modify them when passing data from one SR pipeline utility to another.* User code should only rely on the named fields for their applications. In more detail each field is:

Sample: This is the sample count. It is 0 at the beginning of the run and increments by 1 for each sample taken. A 64 bit integer is used so you can acquire data for as long as you like and still not wrap, even at 10kHz.

Analog[]: These are the analog inputs. Each analog value is a 32 bit signed integer. The A/D converters only return 24 bits, which is packed into the lower three bytes of the 32 bit word with sign extension. Each BIN record has the space for eight channels. On a USB4CH four channel system, the values `Analog[4,5,6,7]` are always 0.

DigitalAll: These are the digital inputs. The four digital input bits are packed in the low nibble of this byte.

PpsToggle: On the PPS tick this value toggles. If it is 0, it becomes 1 at the PPS tick. And likewise from 1 to 0 on the next PPS tick. Plotting this value makes a nice square wave toggling every second, making it easy to tell when the PPS tick occurred.

VoltageGood: The condition of the USBxCH power supply. A value of 1 indicates all is well, 0 indicates some kind of brown out occurred during the run.

Lat10K, Lon10K, Alt10K: These are the GPS location when the sample was taken. Values are scaled up by 10,000 to preserve the fractional part.

DegC2: Divided by 2, this is the temperature of the system board in °C when the sample was taken. Use this value to confirm system health and correct TC in some situations.

TimeStamp: Time when the sample was taken. Measured in nanoseconds since 1970. A nanosecond equals 10^{-9} seconds. This number is saved in a 64 bit integer and will not wrap. Longer than the age of the universe measured in nanoseconds.

See the `Docs` directory in the software distribution for the BIN data structure in the form of a C include file. Use this file in your source code when writing custom C pipeline routines.

Even better, the [Examples and Experiments](#) chapter also has the source code for complete typical utilities in both C and VB.

Chapter 8

Fast acquisition

Data piles up faster than most users realize at first. Consider acquisition at 10kHz. Across eight channels, the analog values alone require:

$$8 \text{ chan} * 4 \text{ Bytes/sample} * 10,000 \text{ samples/sec} = 320,000 \text{ Bytes/sec}$$

and in one hour you would accumulate approximately:

$$0.32 \text{ MBytes/sec} * 60 \text{ sec/min} * 60 \text{ min/hour} = 1,152 \text{ MBytes/hour}$$

That is 1.152 GigaBytes per hour! A TeraByte is soon required. Even with a fast host computer you may run out of bandwidth and storage quickly. Particularly if the host computer has many other tasks to perform.

One way to cut down on the required bandwidth is to save USBxCH data with the least processing possible, leaving any postprocessing until later. This is usually done by saving PAK files directly to disk.

Saving PAK files directly to disk is an excellent scheme for stand alone field box applications. Data can pile up on devices like removable flash disks and later be brought back home for offline post processing.

Among the pipeline utilities there is a complete set of helpers to take PAK files as input and regenerate the pipeline for such offline post processing. See the **Pak*.exe** utilities. **PakFread.exe** in particular can take the PAK files generated by Blast as input and send them to an output pipe.

There are similar pipeline utilities for BIN data. Some applications may have enough horsepower to get the data into BIN format, but wish to leave other processing to later. You can split the processing at any point you wish.

Saving data directly to PAK files has another advantage in some situations. It preserves the original data in its most raw format. Any questions about postprocessing algorithms can be reviewed after the fact and not lost. For many apps this makes saving the raw PAK data files desirable.

The space taken up by PAK data can be greatly reduced with standard compression utilities. Reductions by factors of 8 are common. See the [Examples and Experiments](#) chapter for a demo of these techniques and saving data directly to PAK files.

Chapter 9

Sampling rates

The USBxCH returns converted analog data at a variety of programmable rates. Rates are measured in *SPS* (samples per second) or a frequency in Hz. To set the sampling rate in programs such as **Scope** specify the appropriate ini keywords.

Available rates are listed in the **Rate table** in Figure 9.1. Only rates listed on the table can be selected. The same rate applies to all channels, *you cannot program different rates on different channels*. Both analog and digital inputs are sampled at the same instant with no channel skew.

The USBxCH A/D chips are sigma delta converters. A simple model for their architecture is they oversample their inputs at a high rate and then signal average to produce the final result. As the amount of signal averaging increases, the final resolution improves at the cost of a lower output sampling rate. The input oversampling for the USBxCH TI ADS1255 chips is determined by the on board master clock frequency. For a master clock frequency of Fclk, the oversampling rate is:

$$\text{Input oversampling rate} = \text{Fclk} / 256$$

If Fclk is maximized, then oversampling is also maximized, and the noise floor at any particular final output rate is minimized because there is more signal averaging. The fastest Fclk the ADS1255 can work with is 10.0 MHz, and this is the master clock used on the USBxCH. Using the above formula, the oversampling rate is:

$$\text{Oversampling rate with 10 MHz master clock} = 39,062.50 \text{ Hz}$$

All final USBxCH output rates are averages from this 39 kHz oversampling rate.

9.1 Permitted rates

At the chip level, the final output sampling rate is selected by programming the ADS1255 Data Rate (DR) register. This sets the number of averages to perform on the sigma delta oversampling. From the TI ADS1255 data sheet:

$$\text{SPS rate} = (\text{Fclk}_{\text{in}} / 256) * (1 / \text{NumAvg})$$

$$\text{SPS rate} = 39,062.50 / \text{NumAvg}$$

where the second equation is with a 10MHz Fclk_{in} master clock as on the USBxCH. The following table gives the allowed USBxCH rates rounded to four decimal places:

SPS RATES FOR USBxCH 10.000 MHz (Fclk_{in}) master clock ...

| DR[7:0] | | NumAvg | | SPS rate |
|----------|---|--------|----|--------------|
| 11010000 | = | 4 | -> | 9765.6250 Hz |
| 11000000 | = | 8 | -> | 4882.8125 Hz |
| 10110000 | = | 15 | -> | 2604.1667 Hz |
| 10100001 | = | 30 | -> | 1302.0833 Hz |
| 10010010 | = | 60 | -> | 651.0417 Hz |
| 10000010 | = | 300 | -> | 130.2083 Hz |
| 01110010 | = | 500 | -> | 78.1250 Hz |
| 01100011 | = | 600 | -> | 65.1042 Hz |
| 01010011 | = | 1000 | -> | 39.0625 Hz |
| 01000011 | = | 1200 | -> | 32.5521 Hz |
| 00110011 | = | 2000 | -> | 19.5313 Hz |

Figure 9.1: USBxCH sampling rates with 10 MHz master clock

Only the above rates are allowed. Do not poke the DR register with other bit patterns hoping intermediate rates are possible. The results will be unpredictable. Rates are also limited by USBxCH hardware besides the ADS1255:

At NumAvg = 1 or 2 (39kHz or 19kHz), the USB 1.0 interface and DRAM FIFO cannot keep up, and those rates are not allowed.

At NumAvg > 2000 (19Hz), the GPS time stamp counter overflows, and those rates are also not allowed.

Nothing can be done to produce other native chip rates. However, rates like 1Hz or 100Hz

are readily possible with the `Interp.exe` pipeline utility as discussed next.

9.2 Interpolation to 1Hz, 100Hz, and other rates

Some applications may require specific sampling rates such as 1Hz or 100Hz, or perhaps to be phase locked with timing marks such as GPS PPS ticks. One way to meet these requirements is to interpolate. Because sigma delta A/D converters oversample and signal average there is already a great deal of smoothing inherent in the data they acquire, making interpolation even more attractive.

The `Interp.exe` pipeline utility is a simple way to post process data to various exact sampling rates and align to GPS time. The interpolator of course requires data that is sampled at a rate greater than or equal to the final requested output rate. Run "`Interp.exe /f`" for more details.

See the `DVM` program and `Dvm.bat` for an example of acquiring data at exactly 1Hz and how to use `Interp.exe`.

9.3 Master clock stability

All parts have tolerances and the USBxCH master clock is no exception. The 10MHz clock used on the USBxCH is a TCXO (temperature compensated xtal oscillator) with an initial accuracy of +/- 1.5ppm, and frequency stability of +/- 2.5ppm over the entire temperature range (-30 to +85°C). Thus, at constant room temperature `Fclkin` is actually something between:

$$(10\text{MHz} - 15\text{Hz}) \leq \text{Fclkin} \leq (10\text{MHz} + 15\text{Hz})$$

while over the temp range `Fclkin` could vary by 25Hz. This is much better than the average PC computer clock, which typically has 100ppm variation. And, for many applications, the TCXO is good enough to provide an absolute time base.

But what does all of this really mean? As an example, suppose a particular master clock initial frequency is high by 1.5ppm. The clock period would then be:

$$\text{Tclkin} = 1 / (10\text{MHz} * (1 + 1.5\text{ppm})) = (1/10\text{MHz}) * (1 - 1.5\text{ppm})$$

where the second equal sign uses the first order approximation $1/(1+x) \approx 1-x$. If this clock runs for a day the amount of time it will lose is very close to:

$$(1 \text{ day}) * 1.5\text{ppm} = 0.129,600 \text{ seconds lost}$$

This amount of error could represent many samples at high sampling rates. At 1kHz and with a 1.5ppm error, the analog sample thought to be at the next day boundary would actually have come 129 samples before the true day boundary.

If you know the clock is high by 1.5ppm, you could account for that fact. A good way to calibrate the master clock is to use the USBxCH GPS subsystem, see the [GPS Time Stamping](#) chapter. With GPS time stamping you can count exactly how many samples occur in a day and get an exceptionally accurate initial TCXO calibration.

Of course, calibrating for the TCXO initial accuracy does not make up for its 2.5ppm variation with temperature. If your application is exposed to significant temp variations you should track the time base with the GPS time stamps for an accurate record. If you don't have GPS, and are using the TCXO as an absolute time base, the USBxCH has an on board temp sensor. System temperatures are recorded once per second as part of the data stream. Do a thermal calibration of the time base at various temps, and then use the calibration along with the recorded temp record to make corrections.

Chapter 10

FIFO Depth and Overflow

10.1 FIFO Depth

The USBxCH is equipped with a hardware memory buffer so data can accumulate on board even if the PC is unable to read it immediately. This feature is essential if no data is to be lost during continuous acquisition runs. Even if the PC is busy with Ethernet or other activity when new data becomes ready, the USBxCH on board memory stores the data until the host PC can service the task.

The buffer is organized as a FIFO (First In First Out) memory. As soon as data is acquired it is placed in the buffer for the host PC to read. Once read, the buffer memory is returned to the end of the FIFO. This way there are no latency or fall through times and the FIFO stands only as a guard in case the PC is busy.

The FIFO is organized as $4M \times N$ bit words, where $N = (4 \text{ or } 8)$ depending on the number of USBxCH analog channels. In the case of the USB4CH this is a total of 2 Mbytes, for the USB8CH 4 Mbytes. An important question is: how long can data accumulate before FIFO overflow occurs? Knowing this length of time affects both continuous and burst acquisition. In continuous mode it defines the amount of time the PC can be away on other tasks. At high sampling rates, where the PC bandwidth cannot stay up continuously, it defines how long a burst mode run can last.

The FIFO hold out time of course depends on the sampling rate, filling more quickly the faster the rate. Calculating the hold out time requires knowing how many words per sample are saved. The USBxCH saves 32 words per sample. The 32 words include: the analog data from all channels, the synchronously sampled digital data, GPS time stamps, and error correction codes.

32 FIFO words per sample (includes all four or eight channels)

Since the FIFO has a total size of 4 Meg words = 4,194,304 (decimal), a total of 4,194,304 / 32 words = 131,072 samples can be stored.

$$\text{FIFO size in samples} = 4\text{M} / 32 = 131,072 \text{ total samples}$$

The hold out time is the total number of FIFO samples divided by the sampling rate:

$$\text{FIFO BUFFERING TIME in sec} = 131,072 / \text{SPS in Hz}$$

So for example, if the sampling rate is 100Hz, then the FIFO can hold out for a total of 131,072 / 100 = 1,310 seconds = 21.83 minutes. Something is wrong if the PC can't service the USBxCH in 20 minutes.

The following table summarizes the FIFO depth at the allowed sampling rates. These times are the same for any of the USBxCH products regardless of the number of channels. Note that at the slowest rates data can accumulate for over an hour ... !

FIFO HOLD OUT TIMES FOR USBxCH with 10.000 MHz master clock:

| SAMPLING RATE | FIFO BUFFER HOLD OUT TIME | | | |
|---------------|---------------------------|---|------------|------------|
| 9765.63 Hz | 13.42 sec | = | 0.22 min | = 0.00 hrs |
| 4882.81 Hz | 26.84 sec | = | 0.45 min | = 0.01 hrs |
| 2604.17 Hz | 50.33 sec | = | 0.84 min | = 0.01 hrs |
| 1302.08 Hz | 100.66 sec | = | 1.68 min | = 0.03 hrs |
| 651.04 Hz | 201.33 sec | = | 3.36 min | = 0.06 hrs |
| 130.21 Hz | 1006.63 sec | = | 16.78 min | = 0.28 hrs |
| 78.13 Hz | 1677.72 sec | = | 27.96 min | = 0.47 hrs |
| 65.10 Hz | 2013.27 sec | = | 33.55 min | = 0.56 hrs |
| 39.06 Hz | 3355.44 sec | = | 55.92 min | = 0.93 hrs |
| 32.55 Hz | 4026.53 sec | = | 67.11 min | = 1.12 hrs |
| 19.53 Hz | 6710.89 sec | = | 111.85 min | = 1.86 hrs |

Figure 10.1: USBxCH FIFO hold out times

10.2 FIFO Overflow

The USBxCH buffer maintains an Overflow Flag. Overflow occurs if the USBxCH has been left unserved by the PC for so long that the FIFO is completely filled. If an overflow occurs then any further A/D data is discarded at the USBxCH hardware level. Data saved in the FIFO before overflow is preserved and can be read back by the PC. This way, data from a burst run is not lost.

The FIFO Overflow Flag (OV) is sticky. Once overflow has occurred, the flag stays set until a hardware FIFO reset is done. A reset can be forced by stopping and restarting acquisition in programs like **Scope**, **DVM**, or stopping and starting **Blast.exe**.

10.3 FIFO Creep

For continuous data acquisition, the bandwidth of the PC, hard disk, and USB interface must all be able on average to keep up with the data acquisition rate of the USBxCH. The FIFO allows for the PC to be sporadic in its allocation of bandwidth to service the USBxCH, *but on average the PC must still be able to keep up*.

In cases where the PC can *almost but not quite* keep up with offloading data, the USBxCH FIFO *may creep very slowly toward full*, requiring hours or even days before overflow occurs. How can you know before then if the PC is not keeping up with the average bandwidth requirements?

A Partially Full (PF) flag is maintained for the FIFO. The PF flag should be clearing regularly if the PC is keeping up with the average bandwidth requirements at the particular sampling rate. Check on this flag to make sure the system is keeping up.

Chapter 11

A/D reference voltage

All A/D converters require a reference voltage to compare their analog input against. When the analog input is equal to the reference voltage the converter outputs full counts. For other input voltages the reference sets the linear scale for the converter, see the [Analog DC calibration](#) chapter for details on the DC transfer function.

For conversions to be reliable the reference must be the same from one power on cycle to the next and stable across temperature changes. The USBxCH uses a single precision REF02 part for its reference with the same reference voltage applied to all four or eight channels. A DIP 8 plastic through hole package is used for maximum mechanical stability. The pinout of the REF02 is standard across several manufacturers with many grades of performance available.

A goal of the USBxCH is to achieve *absolute voltage accuracy* rather than simply ratiometric measurement, so the selection of the reference is an important consideration. However, users should be aware other factors may be equally important. Issues such as the precision and drift of the master clock as well as analog front end resistor matching are also important. For many applications, it may be more effective to use the USBxCH on board temperature sensor to perform system level corrections accounting for TC temperature changes rather than pursuing a particular precision reference.

11.1 Standard reference

The reference on the standard USBxCH is populated with a generic REF02 from Texas Instruments. The following are a few of its specifications:

```
TI part number = REF02AP
  Technology = Buried Zener
Reference voltage = 5.0 volts
Initial accuracy = 15 millivolts
  Max TC = 15 ppm/C
  Typical TC = 4 ppm/C
  Temp range = -40 to +85C
  Package = plastic DIP 8 through hole
```

The output of this reference is buffered and scaled to be suitable for use with the USBxCH ADS1255 A/D converters. A single REF02 is used to supply the converters for all channels with the same reference voltage, see the [Circuit Diagrams](#) chapter. The package is DIP 8 to reduce mechanical board stress related drift. Be aware, even though many small monolithic surface mount references advertise excellent TC ppm, they have changes with mechanical stress far exceeding their spec sheet claims.

11.2 Alternate references

High performance versions of the REF02 are available from Cirrus (formerly Thaler):

```
Cirrus part number = VRE3050
  Technology = Buried Zener
Reference voltage = 5.0 volts
Initial accuracy = 0.5 millivolts
  Max TC = 0.6 ppm/C
  Typical TC = 0.3 ppm/C
  Temp range = -40 to +85C
  Package = hybrid DIP/SMT 8
```

This is representative of the very best in the REF02 class. Besides having excellent temperature compensation, the hybrid technology has been specifically designed to relieve variations due to mechanical stress. There are also acceptable devices comparable with the standard TI part listed above from companies such as Analog Devices, Linear Technology, etc. Check their spec sheets for suitability.

Chapter 12

Analog inputs

This chapter reviews the USBxCH analog inputs. General information is given covering the nature of differential inputs, the front panel connector pin assignments, appropriate cabling techniques, and how to adjust the front end circuitry for custom input ranges and other options. See the [Analog DC calibration](#) chapter for the exact relationship of input volts to counts. For an easy hands on example to get familiar with the analog input characteristics, see the [AA battery](#) experiment.

12.1 Differential signals

The USBxCH has *differential* analog inputs. Although many users are probably more familiar with *single ended inputs*, differential inputs have many benefits. Even though the system can be used in single ended fashion, differential signaling is recommended for achieving the highest precision.

The concept of a differential signal is as follows. On the USBxCH, each analog channel is equipped with three inputs: (+,-,AGND). The A/D count for the channel is given by the voltage *difference* between the (+,-) pins,

$$\text{A/D counts} = \text{slope} * (V+ - V-) + \text{offset}$$

where the slope and offset are the [Analog DC calibration](#), and $(V+, V-)$ are measured with respect to AGND. The advantage of this is any noise which is common on both the + and - pins is subtracted away and not part of the final returned value. Common mode noise rejection with differential signals is a powerful way to reduce ground noise which is present on almost every system.

Note that the differential - pin is not ground. It is an input just like the + pin, with the

only difference being its effect on the A/D counts is inverted. In fact, you can reverse the (+, -) connections on any test setup and all that happens is the count values are inverted. Note that if you are making a single ended connection, you could just as well tie the + pin to AGND and connect the signal to the - pin. Everything would work the same as the more commonly implemented - pin grounded and signal connected to the + pin configuration, except for the inversion.

To see how differential inputs can reduce ground noise, consider the following figure:

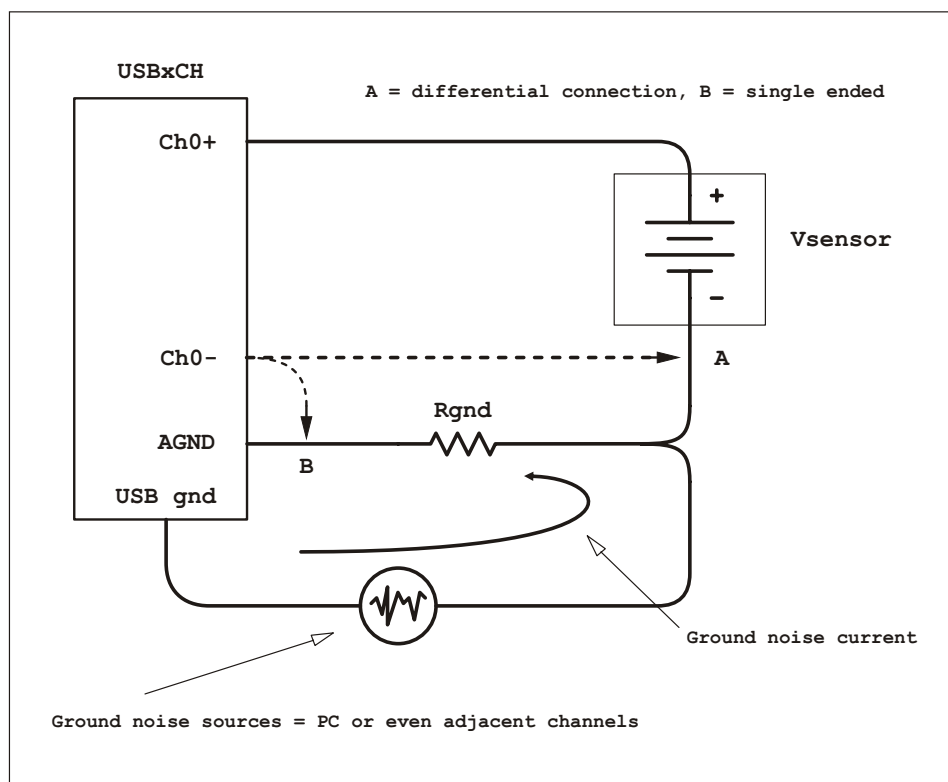


Figure 12.1: Differential vs single ended signals

The sensor or voltage which is to be connected to the USBxCH is indicated by the battery symbol inside the smaller rectangle. The sensor itself is probably not a battery, but it has an output that can be idealized as some voltage that varies with time. It will also be connected to the USBxCH with wires of unknown length depending on the installation.

It is reasonable one terminal of the sensor should be connected to the + terminal of the USBxCH as shown in the figure. But the real question is: what to do with the other terminal? It could be connected to AGND, and the system would work. In fact for a *single ended* system with multiple sensors, all of the second terminals would be connected to AGND with a single common wire, where sometimes even a chassis might be used. The resistance

of that common connection is labeled **Rgnd** in the figure and is where the trouble starts.

Let's estimate the magnitude of **Rgnd**. 24 gauge stranded wire has a typical resistance of 0.020 ohms per foot, and a 5 foot length would have a total resistance of 0.1 ohm. Next, suppose a device sharing the common ground conducts 10ma. By Ohm's law, the voltage across **Rgnd** would be 0.001 volt, and *this would appear in series with Vsensor* from the viewpoint of the USBxCH. The **Vsensor** measurement is at the mercy of other devices and will suffer crosstalk simply because of the single ended connections.

Just how big a voltage is 0.001 volts or 1 millivolt? It is huge given the 1 *microvolt* precision of the USBxCH. *The error caused by a mere 1 millivolt ground noise would be 1000 times greater than the USBxCH sensitivity!* To cancel the contribution of ground noise the - input should be used. Rather than leaving it unconnected, or connected to point B in the figure, connect it to point A. Since only the difference between the (+,-) inputs is converted into counts, the contribution from **RGND** is not included in the measurement.

The precision offered by differential measurements are significant. Many active sensors already have differential outputs and connections are easy:

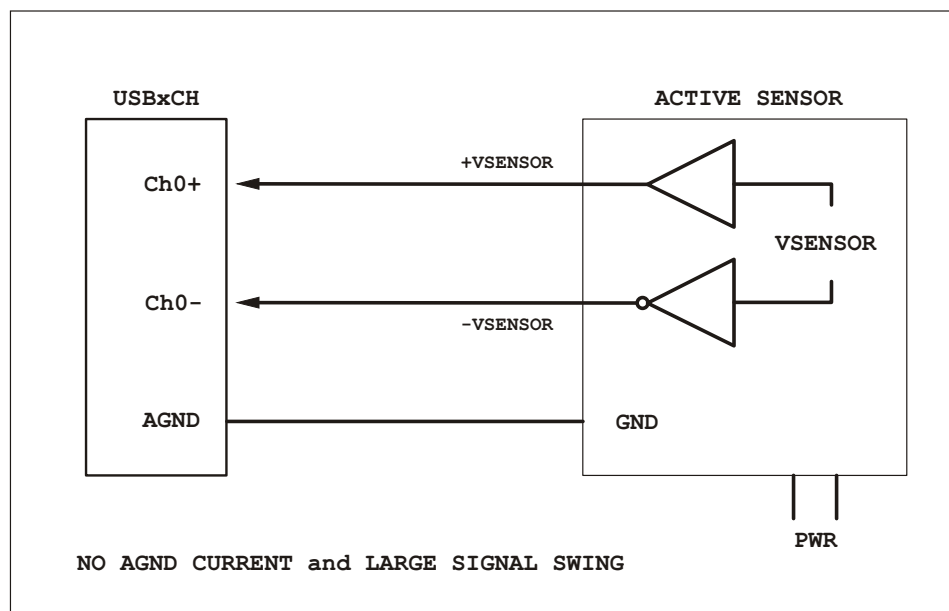


Figure 12.2: Balanced differential inputs with an active sensor

Often, active sensors go even further and have *balanced differential outputs*. That means the - output is exactly as negative as the + output is positive. The result is all the current going out the + output is matched by that coming in the - output and there is *no current at all on AGND*. The **AGND** connection is still required to reference the differential signals, but with no current flowing on it common mode noise is not an issue.

To connect a floating sensor to the differential inputs simply hook the two sensor terminals to the (+, -) pins as in the next figure. Here the voltages are automatically self balancing and referenced to **AGND** because of the USBxCH input circuit. A few examples of floating sensors might be geophones, microphones, magnetic pickups, piezo cells, etc. It would still work if you incorrectly connect the - pin to **AGND**, but the inputs would no longer be balanced around **AGND** and you would be more susceptible to ground noise.

See the [Examples and Experiments](#) chapter for a demo with a floating AA battery.

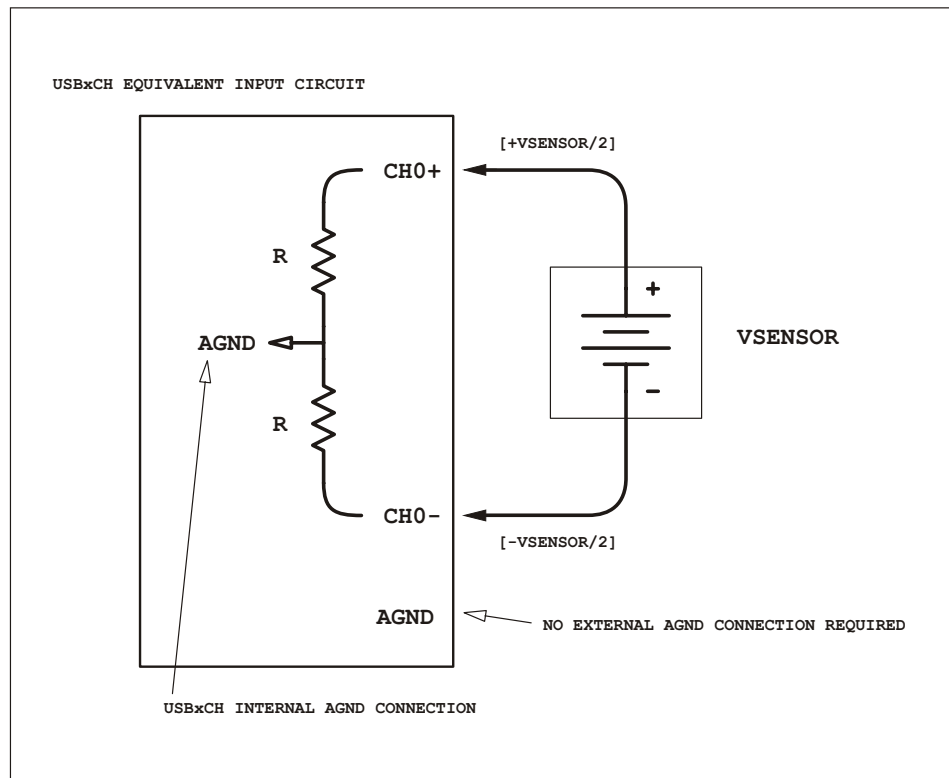


Figure 12.3: Two terminal floating sensor connection

12.2 DB front panel analog pin assignments

The USBxCH analog inputs are on the left front panel D subminiature (DB) connector. This is where analog input connections should be made, with the (+,-,AGND) inputs for each differential channel.

The four channel USB4CH uses a DB15 pin connector, while the eight channel USB8CH uses a DB37. The pin pattern is similar on both, with the USB8CH only being wider for the additional channels. The following table gives the USB4CH pin assignment:

| DB15 pin | Analog signal | Description |
|----------------|---------------|----------------------|
| Diagonal pairs | | |
| 1 | Channel 0 + | + differential input |
| 9 | Channel 0 - | - differential input |
| 2 | AGND | |
| 10 | AGND | |
| 3 | Channel 1 + | + differential input |
| 11 | Channel 1 - | - differential input |
| 4 | AGND | |
| 12 | AGND | |
| 5 | Channel 2 + | + differential input |
| 13 | Channel 2 - | - differential input |
| 6 | AGND | |
| 14 | AGND | |
| 7 | Channel 3 + | + differential input |
| 15 | Channel 3 - | - differential input |
| 8 | AGND | |

Figure 12.4: USB4CH analog DB15 pin assignments

Note the physical connector pin numbering starts at 1, while the USB4CH channel numbering starts at 0. On the DB15, there are 8 pins across the top row, and 7 pins across the bottom. These are grouped diagonally for the various channels. For example, channel 0 (+,-) are the diagonal DB15 pair pins (1,9). This diagonal grouping keeps differential pairs next to each other when using ribbon cable connectors. All of the AGND pins are connected together and can be assigned to channels as convenient.

See Figures 12.5 and 12.6 on the next page for pictorial views of the USB4CH front panel

DB15 and the diagonal pin assignments.

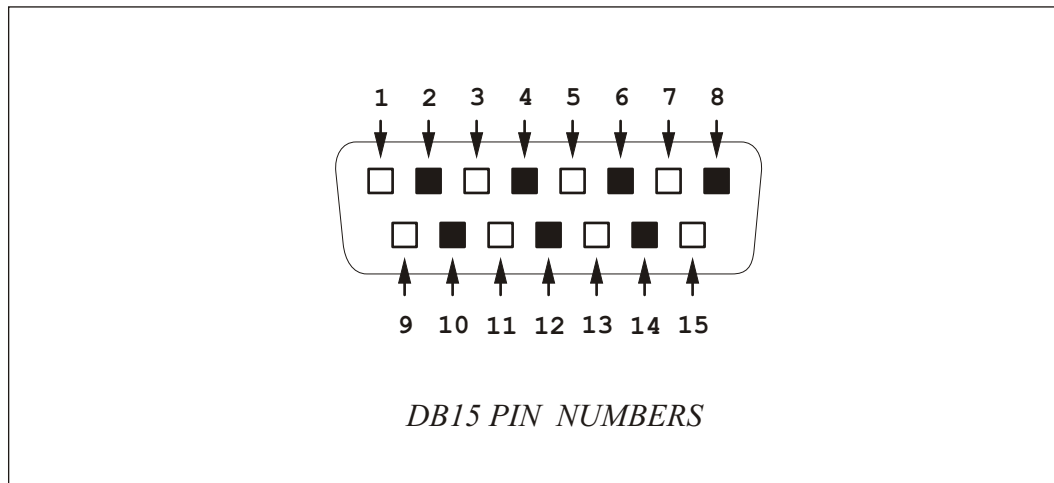


Figure 12.5: Analog DB15 pin numbers viewed from front panel

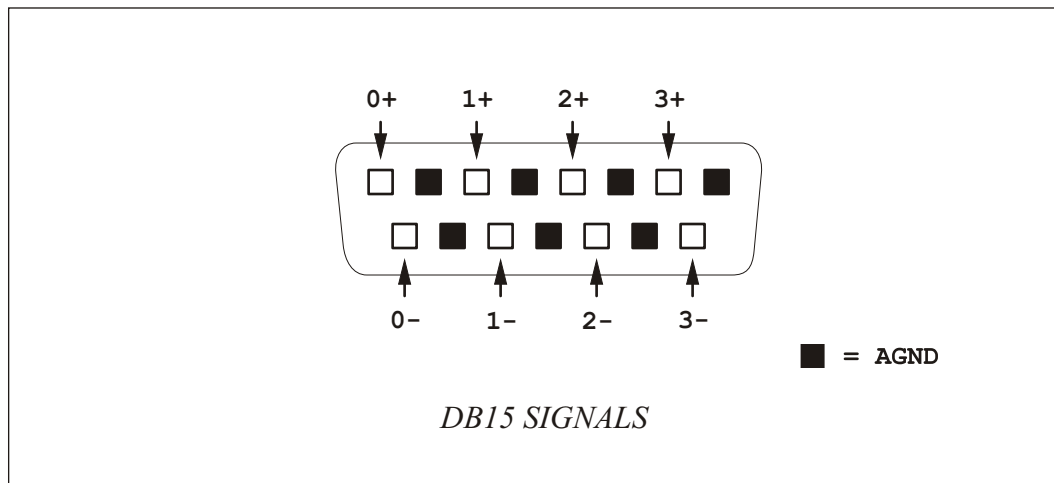


Figure 12.6: Analog DB15 signals viewed from front panel

The pin layout for the USB8CH is similar to that in Figure 12.6, only wider. See the [Circuit Diagrams](#) for additional details.

Additional details regarding analog cabling are in the following sections. Please read them before making connections. In particular avoiding ESD damage requires care.

12.3 Twisted pair cabling

Despite the benefits of differential signals, additional measures may be required to guard against noise. Two examples are magnetically coupled noise, and thermocouple effects. Both of these can degrade microvolt measurements.

By Faraday's Law, whenever time varying magnetic flux lines, $d\Phi/dt$, cross the wire loop formed by the (+,-) input cabling a noise current will be generated. While such noise can occur at any frequency, by far the most common is 50/60Hz powerline noise. At many installations, the wall power wiring is close enough and surrounds the work area sufficiently to magnetically couple to the USBxCH input cabling. The action is like a transformer with the wall wiring as primary and the USBxCH input cabling as secondary:

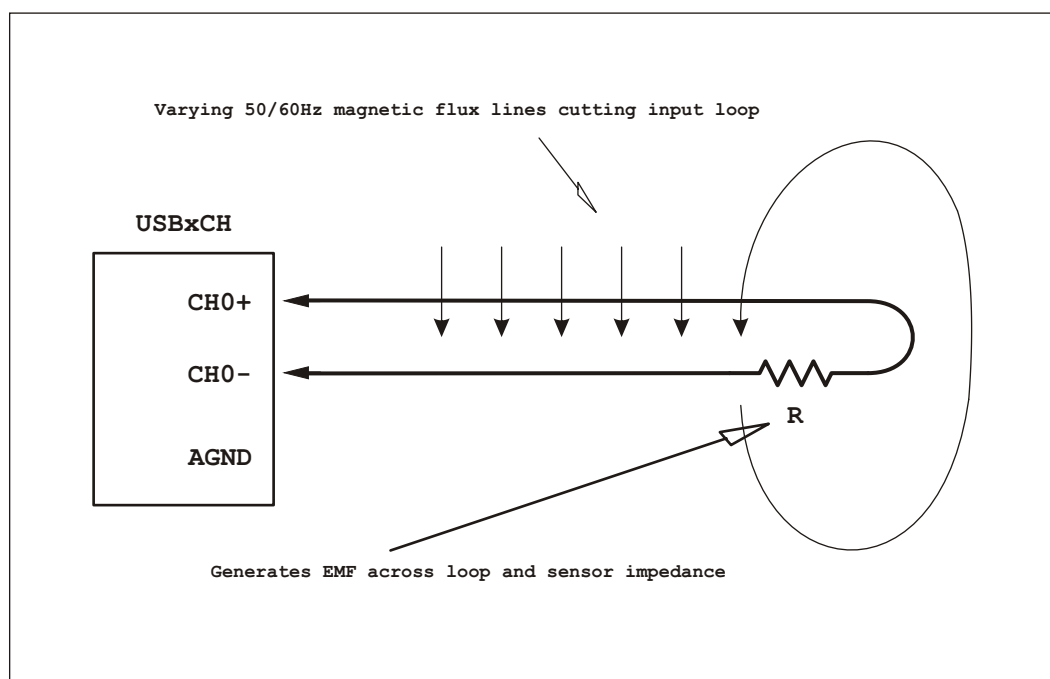


Figure 12.7: Magnetic coupling of 50/60Hz noise

Because the voltage generated by the varying flux is in series with the differential loop, it is not removed as common mode noise. The best defense against this type of noise is to prevent it from entering the system in the first place. According to Faraday's Law, reducing the area of the cable loop will reduce the EMF voltage generated. Placing or binding the leads close to each other as might be done with flat ribbon cable will help. However, even more effective is using twisted pair cable. The EMF generated by magnetic flux lines cutting each small twist has opposite polarity from one twist to the next, and cancels out. The technique is surprisingly effective, see the [Examples and Experiments](#)

chapter. Note that even the flat areas on twisted pair ribbon cable can cause trouble. Discrete wire twisted pairs give the best performance.

Shielded cable is not particularly effective against magnetic coupling. You will have better results with simple twisted pair in reducing 50/60Hz noise. Shielded cable is excellent for ESD protection and is discussed in the next section.

Thermocouple effects are another common problem. Whenever two dissimilar metals are connected together they generate a temperature dependent junction voltage. For a copper to tin/lead junction as might occur in a solder joint, the voltage is 3 microvolts per °C. This can be a substantial noise contribution at 24 bit sensitivity.

The way to battle against thermocouple noise is to keep both the + and - differential inputs exactly balanced with metal to metal junctions. If a junction is introduced on the + side, add one to the - side even if it is not needed. By keeping the thermocouples exactly the same on the both sides, the differential signaling will cancel the two equal sides. If the (+, -) sides are kept in the same thermal bath, the cancellation will be exact across all operating temperatures.

12.4 Static shielding

Electrostatic discharge (ESD) events into the front panel DB analog input connector are guaranteed to damage the USBxCH. *This is not maybe, it is guaranteed.* Even though the system may appear to continue working correctly, at a minimum calibration will be changed with each ESD event.

Of course, drastic ESD events such as lightning strikes deliver irreversible damage that is visible. For such installations, ground rods, gas arrestor tubes, TVS zener diodes, and cable shielding should all be used as protection. However at the other end of the spectrum, even a mild spark generated by a finger touch from walking over a carpet will also cause damage and should be avoided.

Surprisingly the most likely components to be damaged from ESD finger touch events are the series input resistors (R402/x, R403/x), and not the integrated circuits. See the circuit diagram [A/D signal conditioning](#) page. Each mild ESD event vaporizes a little bit of the resistor metalization, changing the input characteristics. The damage is interior to the resistors and will not be visible. It is cumulative, with each event damaging a little more metalization. Eventually, the resistor is blown open much like a fuse, and the channel will appear to have failed.

If you have a channel go dead, it is easy to see if the series input resistors are compromised. Measure their resistance with a multimeter such as a Fluke. If they measure higher than the value listed on the circuit diagram, then they have been damaged and should be replaced. One function of the input series resistors is specifically to help guard against ESD events

and their replacement in such cases may be required.

If you must touch the inputs with your fingers, *prevent static damage by touching the case immediately before*. This will bring you and the system to the same potential. This simple step will go a long way to preventing trouble. A further step is to *use shielded cable for the analog inputs*. The shield should be connected to the case of the USBxCH, which in turn is connected to the USB cable shield, which is connected onward to the PC case, and ultimately to third prong of the wall plug. The entire string of shielding forms a Faraday cage and mild ESD events are dissipated away from the sensitive electronics. *This approach is an excellent defense for installations with mild ESD risk.*

Use round shielded twisted pair multiconductor cable such as that from Belden to make such connections. The cable foil shield provides good static protection and the twisted pair is an added benefit for 50/60Hz noise rejection. See [Extra supplies](#) for part numbers.

12.5 Input impedance

Analog input signals on the USBxCH are connected to non-inverting op amp pins through mild passive signal conditioning. Because of this the system has an inherently high input impedance of 10M ohms. See the TI OPA2277 data sheet in the software `Docs` directory for op amp specifications.

For most applications an input impedance of 10M ohms is too high for good performance. With the tiny signal currents implied by an input impedance this high, even the smallest noise sources compete and signal to noise ratio is lost. To control input impedance, the USBxCH is equipped with *resistors in parallel with the inputs*. See the circuit diagram [A/D signal conditioning](#) and resistors (R400/x, R401/x).

The parallel impedance setting resistors are normally populated with 51K ohm values. For applications with active sensor outputs this is a good value. It draws a reasonable amount of signal current, but does not overload the active sensor output either. At this happy compromise, noise sources like the triboelectric effect of input cabling will not compete with small input signals.

For applications such as passive sensors requiring other input impedances, the resistors (R400/x, R401/x) should be adjusted appropriately. There are SMT 0805 solder pads on the circuit board for making changes. Note that if the system is set to 10M ohm input impedance and the A/D inputs are allowed to float, conversions will be noisy and not pulled to ground reliably. To measure the USBxCH noise floor with high input impedance, short the inputs to AGND or other low impedance sources.

12.6 Input voltage range

The normal full scale input range of the USBxCH is (+/-) 4 volts as described in the [Voltage Span and Counts](#) section of the Analog DC Calibration chapter. This is the input range on the (+,-) pins *as measured with respect to AGND*. The inputs are differential, but only as long the voltage on each pin is within this range.

Other input ranges can be accommodated by changing resistor values on the board. For *larger* input ranges, there are solder pads on the input circuit for adding resistor dividers. For *smaller* input ranges, the gain of the front end op amp amplifier must be increased.

The location of the resistor dividers for setting larger ranges is on the circuit diagram page [A/D signal conditioning](#), see resistor pairs (R402/x, R404/x) and (R403/x, R405/x). Use matched components to keep the differential inputs balanced.

One popular larger input range is (+/-) 10 volts. This is a fairly common requirement for active sensors with op amp outputs. For this range replace (R402/x, R403/x) with 15K ohm values, and (R404/x, R405/x) with 10K ohms. By Ohm's law:

$$(10K / (10K + 15K)) * 10 \text{ volts} = 4 \text{ volts}$$

bringing the larger input within range. Use at least 1% metal film resistors, and preferably even better 0.1% thin film resistors to preserve differential balance. Note the resistor dividers also set the input impedance. This is usually not a problem for active sensors with op amp outputs.

12.7 Op amp gain

Besides resistive dividers, there are also solder pads on the board to increase the front end amplifier gain so the input voltage range is *smaller* than the factory (+/-) 4 volt default. See the circuit diagram [A/D input amplifiers](#) page, resistors R410/x, R412/x, R413/x.

The resistor R410/x is commonly referred to as the gain setting resistor R_G , while R412/x and R413/x are the feedback resistors R_F . To keep the differential inputs balanced use matched R_F pairs. The transfer function for the differential gain is then:

$$(V_{out+} - V_{out-}) = (1 + 2*(R_F/R_G)) * (V_{in+} - V_{in-})$$

so for example, with $R_G = 1K$ and $R_F = 10K$ the differential gain would be 21. For unity gain, leave R_G empty (infinite ohms) and set $R_F = 0$ with a zero ohm resistor. This avoids offsets due to op amp input bias currents flowing through R_F .

12.8 RC antialias filtering

There are number of places on the input circuit where capacitors can be added for simple RC filtering. The capacitors are populated at the factory as follows:

(C400/x, C401/x):

These are for snubbing ESD events on the inputs. The 47pF value snubs ESD but still does not put much capacitive loading on active sensor outputs which might oscillate if overloaded.

(C406/x, C407/x, C408/x):

If your site has radio frequency contamination aliasing into the measured signal, these locations can be populated to provide some RC filtering. They are left empty at the factory.

(C412/x, C413/x):

Populate these if you wish to roll off the amplifier response. Leave these unpopulated if running the OPA2277 at unity gain.

(C414/x, C415/x):

It is critical these locations are populated with 220pF values to avoid OPA2277 oscillations. The capacitive load presented by the input circuit of the ADS1255 must be compensated for with capacitors in these locations.

The amplifier front end *is not* designed to provide antialias filtering with a sharp notch at 50/60Hz. Applications requiring powerline noise rejection should use either a custom op amp filter in front of the USBxCH, or sample at higher rates and apply numerical filtering. Twisted pair input cabling also provides an effective way to reduce 50/60Hz noise.

Chapter 13

Analog DC calibration

The physical quantity measured by the USBxCH is voltage, and its A/D converters return a count proportional to their input voltage. When the input voltage is low, a low count value is returned, and as the input increases the count value increases.

The exact relationship between a particular input voltage and the A/D counts is referred to as the *DC calibration*. It is well approximated by a straight line with a *slope and offset*. And, performing a calibration means determining the slope and offset of this straight line so applications can have an equation to convert A/D counts into volts.

As shipped, the USBxCH software is calibrated with slopes and offsets for a theoretically perfect board, and programs like **DVM** have that slope and offset in their default INI files. Those values are reasonably good, and are certainly good enough for starting work with 0.1% accuracy. But, for the highest absolute accuracy each USBxCH channel must be further calibrated on site by the user.

The following sections cover the calculations involved with DC calibration. For an easy to use calibration program see the standalone utility **Calibrate**.

13.1 Full Scale Voltage Span and Counts

The USBxCH has 24 bit A/D converters with 2^{24} counts spread out over the entire input voltage span of the system:

$$\text{TOTAL 24 BIT A/D COUNTS} = 2^{24} = 16,777,216 \text{ (decimal)}$$

Counts are returned as 32 bit *signed integers*. Theoretically, for zero input volts, zero counts should be returned, and as the input goes positive or negative the count value goes positive or negative, with 2^{23} counts above zero and 2^{23} below zero.

The USBxCH has +/- 4 volt *differential inputs*. Users are often surprised to learn this actually implies an input voltage span of 16 volts where:

Positive full scale counts are returned when the + analog input is + 4 volts, AND the - analog input is - 4 volts, a difference of + 8 volts.

Negative full scale counts are returned when these voltages are reversed, with a difference of - 8 volts.

This results in the 16 volt span for the full count range, running all the way from a + 8 volt differential input at positive full scale to a - 8 volts at negative full scale. Some of the count values from positive full scale down to negative full scale are:

| + Input (volts) | - Input (volts) | Counts (hex) | Counts (decimal) |
|-----------------|-----------------|---------------|------------------|
| +4 | -4 | 0x00 7FFFFFFF | 8,388,607 |
| ... | | | |
| +e | -e | 0x00 000002 | 0,000,002 |
| 0 | 0 | 0x00 000000 | 0,000,000 |
| -e | +e | 0xFF FFFFFE | -0,000,002 |
| ... | | | |
| -4 | +4 | 0xFF 800000 | -8,388,608 |

Figure 13.1: A/D counts with differential input

Differential inputs have many benefits as described in the [Analog inputs](#) chapter. In addition, the 16 volt input span provides an excellent signal to noise ratio with low voltages. The +e above represents a small voltage, approximately 1 microvolt.

The USBxCH can be used as a *single ended* system by connecting the - input to ground and applying the signal to the + input only. However, noise immunity with such a connection

is greatly reduced, and the count range is cut in half effectively losing one bit of resolution. Nevertheless, for applications where single ended is appropriate the count values are listed in the table in Figure 13.2.

| + Input (volts) | - Input (volts) | Counts (hex) | Counts (decimal) |
|-----------------|-----------------|--------------|------------------|
| +4 | 0 | 0x00 3FFFFFF | 4,194,303 |
| ... | | | |
| +e | 0 | 0x00 000001 | 0,000,001 |
| 0 | 0 | 0x00 000000 | 0,000,000 |
| -e | 0 | 0xFF FFFFFFF | -0,000,001 |
| ... | | | |
| -4 | 0 | 0xFF C00000 | -4,194,304 |

Figure 13.2: A/D counts with single ended input

13.2 Theoretical counts per volt

With the total counts and 16 volt input span from the previous section, the theoretical USBxCH counts per volt is:

$$\text{Counts per Volt} = (2^{24})/16 = 1,048,576 \text{ counts / volt}$$

or equivalently,

$$\text{Counts per Millivolt} = 1,049 \text{ counts / millivolt}$$

$$\text{Counts per Microvolt} = 1.0 \text{ count / microvolt}$$

where the last two values are rounded. Of course the volts per count is just the inverse:

$$\text{Volts per Count} = 16/(2^{24}) = 0.953 \text{ microvolts / count}$$

Approximately 1 microvolt/count. Because the A/D converters have noise floors greater than 24 bits, the resolution implied by these numbers may not be possible. For example, at 130 Hz the ADS1255 has a repeatable noise free count of 21 bits. The three noise bits represent $2^3 = 8$ microvolts of noise. You may prefer to work in 21 bit counts and 8 microvolts per count at 130 Hz. See the [Noise floor specifications](#).

Note the input voltage span depends on the gain setting of the front end op amps. The

above calculation assumes a gain of 1 on those amplifiers. By changing the on board resistors it is possible to run the amplifiers with gains of 1 to 100 with no increase in the noise floor. Changing the gain is discussed in the **Op amp gain** section. The full scale input range will be smaller with added gain, but the counts per volt will be more sensitive.

We *do not recommend* using the ADS1255 PGA feature to increase the counts per volt. As with most sigma delta A/D converters claiming to have a PGA, the ADS1255 implements this function by changing the effective oversampling. When doing this the noise floor increases in direct proportion. This is in contrast to changing the gain of the USBxCH op amps, where the gain can be increased without corresponding noise floor increases.

13.3 Calibration slope and offset

Programs like **DVM** express the DC transfer function as:

$$(V+ - V-) = \text{slope} * (\text{A/D counts}) + \text{offset}$$

where for a theoretically perfect A/D converter the **offset** is zero, and the **slope** is the volts per count (rather than counts per volt). Using the volts per count from the previous section gives:

$$(V+ - V-) = 9.53674\text{e-}007 * \text{A/D counts}$$

$$9.53674\text{e-}007 = 1/1,048,576$$

With this, the default **DVM** INI parameters for voltage display are:

```
SlopeN = 9.53674e-007    ; = ( 1/1,048,576 )
OffsetN = 0
```

where N is the channel number. Of course, the USBxCH components are never theoretically perfect and the true slope and offset will be slightly different from this ideal channel by channel.

For an exact calibration of any particular USBxCH channel, use the standalone utility **Calibrate**. Finally, besides calibration into volts, it is easy to calibrate **A/D counts** into other physical quantities such as temperature, light level, etc as suitable for the sensor you are using.

Chapter 14

Analog AC calibration

Suppose you have sinusoidal input on an analog channel. How does its amplitude and phase get modified by the USBxCH? The AC calibration is the relationship between sinusoidal input and the digitized output as a function of input frequency. This is also known as the analog transfer function or response.

Because sigma delta A/D converters signal average, it is easy to imagine the USBxCH will have an inherent low pass effect on sinusoidal input. At frequencies agreeing exactly with averaging length there is zero output, because the average of a sinusoid over a full period is zero regardless of its phase. Meanwhile at DC there is no effect, the average of a constant is a constant. Indeed this is the case and the next sections show both the theoretical and measured response from the A/D converters agree with this expectation.

As an experiment, you can use a sine wave signal generator and measure the response frequency by frequency as you turn the dial. This does not require an expensive generator, even the most basic will do. It is instructive to experiment in this way. Start up the **Scope** program, hook up a signal generator, and see what happens as you twist the dial from DC on up. *Doing this is highly recommended, you will quickly get an intuitive feel for the response.*

The AC response to all frequencies can also be determined in a single test by measuring the response to a square wave step input. Since a square edge contains sinusoidal components of all frequencies, the entire spectral response can be computed this way in a single run. The experiment is easy to set up, using a USBxCH digital output to generate the square wave. The measured results presented here have been determined by this method.

14.1 Theoretical AC transfer function

The dominate part of the USBxCH frequency response comes from its A/D converters. Because sigma delta converters oversample their inputs and then signal average, they impose a low pass filter characteristic on the response. Nothing else in the USBxCH signal chain has a comparable effect.

From the Texas Instruments ADS1255 spec sheet the theoretical transfer function for the A/D converter is:

$$A(f) = |sinc(f)|^5 * |sinc(N)| / Normalization$$

A copy of the TI spec sheet is in software Docs directory, see page 19 "Frequency Response". When plotted this function looks like:

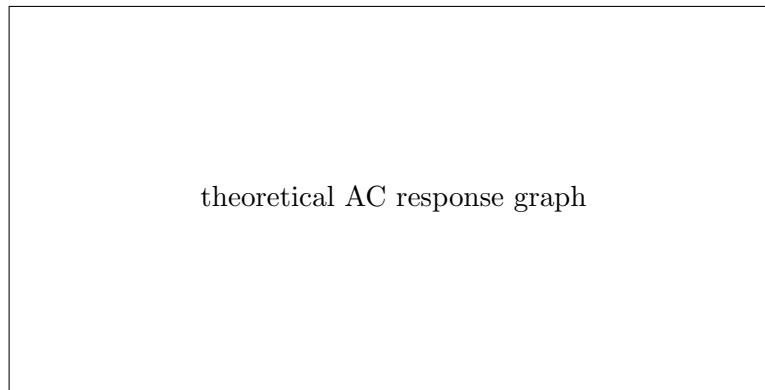


Figure 14.1: AC response: theoretical sinc

which has the familiar low pass bouncy ball response expected from a finite length average. As the input frequency increases, when the wavelength or several wavelengths agree with the averaging length the output falls to zero.

Note the first zero occurs exactly at the ADS1255 sampling rate. Halfway from DC to this first zero, the amplitude decreases only mildly. When running experiments stay under the left half of this first lobe. *Use sampling rates at least twice that of the signals you plan to capture to do so.* Applications following this recommendation often do not require deconvolving the converter response.

The phase response is also that of signal averaging. A linear phase delay. As the input frequency increases, the phase delay increase in direct proportion. This results in a constant time delay for any input signal. A phase response that does no damage to your data.

14.2 Measured AC transfer function

For the experimental setup to measure the AC frequency response see the [Examples](#) chapter. A square wave from a USbxCH digital output is used to drive an analog input. The results are:

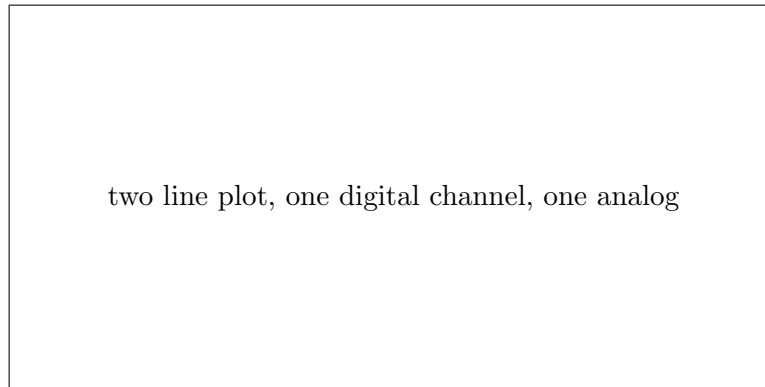


Figure 14.2: AC response: physical measurement setup

and if the amplitude of the Fourier transform of the output is plotted we get:

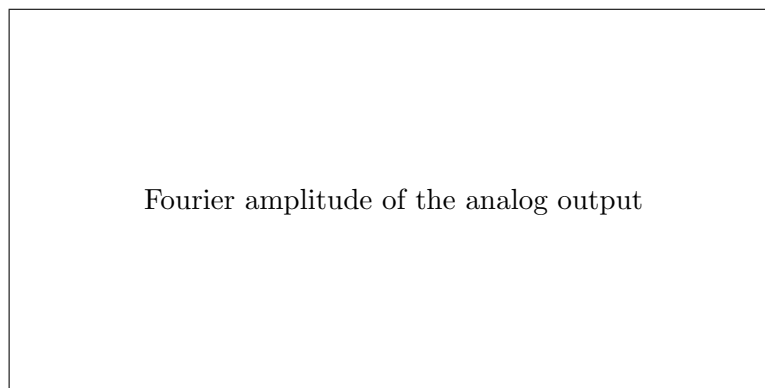


Figure 14.3: AC response: physical measurement setup

and the phase:

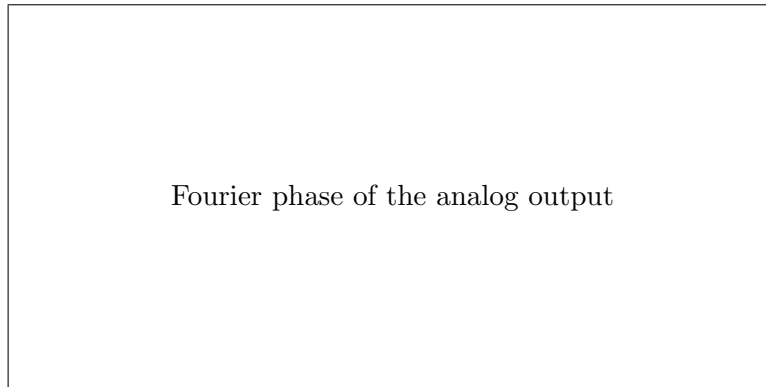


Figure 14.4: AC response: physical measurement setup

Note even in the first plot in the time domain you can easily see the mild low pass averaging of the input square wave, along with constant time delay of the output to the input. The results presented in the Fourier transforms are no surprise.

If you have custom electronics ahead of the USBxCH you can even use the DB25 digital output to measure the response of your front end and USBxCH combined. Simply move the USBxCH digital square wave the USBxCH analog input to your custom front end inputs and use the same techniques as reviewed here.

Often customers are concerned about the contributions of the USBxCH passive signal conditioning and op amp front end. Those have been designed on purpose to be extremely mild. Their contribution to the AC response is negligible.

Chapter 15

Digital IO

The USBxCH has digital io in addition to its analog A/D inputs which can be used for simple communication with external equipment and sensors. The digital io is accessed on the DB25 front panel connector, where besides general purpose bits, there are also timing signals and a GPS interface.

15.1 Digital input

The USBxCH has four general purpose digital inputs. The digital inputs are *synchronously* sampled and automatically saved along with the 24 bit A/D analog inputs. No special commands are required, the save happens automatically, with the effect being similar to a mixed signal oscilloscope.

The digital inputs use standard 3.3 volt TTL/CMOS logic levels. Inputs near 0 volts are saved as a 0, while those near 3.3 volts are saved as a 1. All four inputs are buffered through HC14 Schmitt inverters for noise immunity and slow edge tolerance. An inversion is automatically added to the HC14 buffers so the saved digital values agree with the DB25 pins.

Besides HC14 buffering, each digital input also has RC filtering for further noise and glitch immunity. Depending on the application, you may wish to change the RC filtering to be different than the factory values. There is also a pull down resistor on each input so they do not float when left unconnected. See the table below for the DB25 pin assignments and the [Circuit Diagrams](#) chapter for details.

15.2 Digital output

The USBxCH has four general purpose digital outputs.

These four outputs may be set with the standalone **DigitalIo** utility. Once written, they stay latched and do not change until the next write. The power on/reset values are 0.

The digital outputs use standard 3.3 volt TTL/CMOS logic levels. If you program an output to 0, the corresponding DB25 pin will be close to 0 volts, and likewise programming a 1 will result in the output pin being near 3.3 volts.

All the digital outputs are driven by HC14 inverters, and like the digital inputs another inversion is added so the DB25 pin agrees with the programmed value. Each output has a 300 ohm *series resistor* to limit the current that can be drawn from the DB25 pins. Direct shorts across the output bits can be tolerated. The 300 ohm resistors also make it easy to connect LEDs as indicators with no additional resistors required. See the table below for the DB25 pin assignments and the **Circuit Diagrams** chapter for details.

15.3 Additional digital timing and GPS signals

Along with general purpose digital io, there are also USBxCH timing signals and a GPS interface on the DB25. The signals are:

Adrdy: This output is a buffered copy of the ADS1255 A/D converter data ready signal, indicating when an A/D conversion has been completed. Adrdy strobes at the analog sampling rate with the rising edge indicating conversion completion. Applications can synchronize to the A/D sampling rate with this signal. This signal is disabled at power on and must be enabled in the **UserCfgByte**.

However ... be aware! Adrdy *does not* indicate the instant at which analog samples are taken. Oversampling sigma delta converters like the ADS1255 only compute an average of the analog input over the sampling period. Adrdy *only indicates* when the converters have a new sample ready to be moved onward to the PC.

GPS: By applying GPS signals from a GPS receiver to these pins you can automatically record GPS time stamp and location information along with the analog and digital inputs. See **GPS Time Stamping** for details.

Trigger: This signal is not yet implemented. Future firmware revisions may integrate it for use with programs like Scope. Software triggers are possible with the pipeline utilities with the **Trigger.exe** utility.

15.4 DB25 pin assignments

The USBxCH digital io signals are available on the male Dsub 25 pin connector (DB25) located on the right hand side of the front panel. Both the four channel USB4CH and the eight channel USB8CH have the same DB25 connector.

The pin assignments are:

| DB25 pin | Signal | USBxCH | Power on polarity |
|-------------------|-------------------|--------|-----------------------|
| Top row | | | |
| 1 | Din bit 0 | input | pull down |
| 2 | Din bit 1 | input | pull down |
| 3 | Din bit 2 | input | pull down |
| 4 | Din bit 3 | input | pull down |
| 5 | Dout bit 0 | output | low |
| 6 | Dout bit 1 | output | low |
| 7 | Dout bit 2 | output | low |
| 8 | Dout bit 3 | output | low |
| 9 | Trigger | input | idle low, rising edge |
| 10 | Adrdy sync pulse | output | disabled |
| 11 | GPS PPS | input | idle low, rising edge |
| 12 | GPS RS232 NMEA RX | input | idle high |
| 13 | GPS RS232 NMEA TX | output | idle high |
| Bottom row | | | |
| 14 - 25 | Ground | - | - |

Figure 15.1: USBxCH digital DB25 pin assignments

It is easy to count pins on the DB25. Looking at the connector from the front, all the signal pins are on the top row. All the ground pins are on the bottom row. Counting from the left on the top row, the first four signal pins are the digital inputs, the next four signal pins are the digital outputs, etc. Note that the digital bits are counted starting from 0, while the DB25 pins are counted starting from 1.

All signals must be referenced to the ground on the DB25, where each signal should be paired with a matching ground return if possible. The following two diagrams show the pins viewed from the front panel:

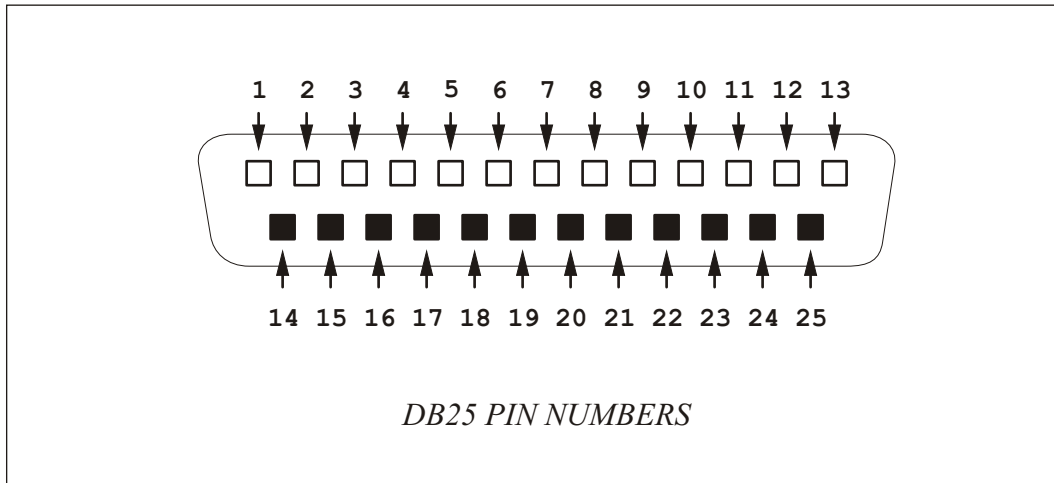


Figure 15.2: Digital DB25 pin numbers viewed from front panel

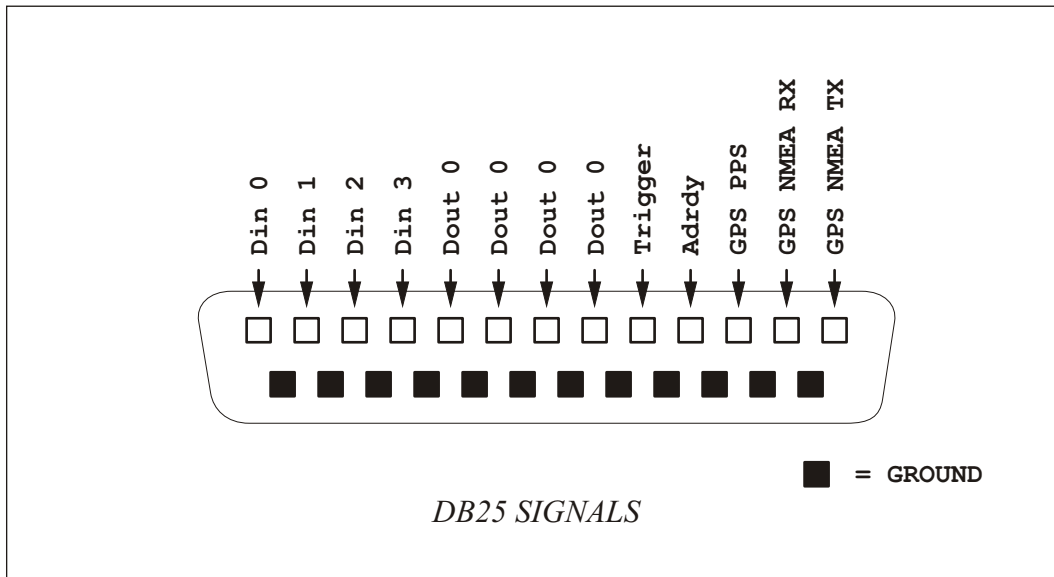


Figure 15.3: Digital DB25 signals viewed from front panel

You can test programming the digital outputs with the utility program [DigitalIo](#). Measure their values with a multimeter like a Fluke. The digital inputs can be viewed in real time with the [Scope](#) program.

15.5 Static shielding

Static electricity discharge (ESD) is damaging to all solid state equipment. The DB25 connector is designed to tolerate small static shocks, *but users must take reasonable steps to avoid ESD problems*. If the ESD event is large enough, such as a lightning strike, there is no hope, but *reasonable steps against smaller events are also necessary* and can avoid failure. Even the static discharge from walking across a carpet in winter will cause severe damage.

Using *shielded* multiconductor cable for the digital connections to the DB25 is one of the best preventions possible. The foil shield should be connected to the metal DB25 housing. When a static discharge hits the cable, the jolt will be conducted down the shield, onto the USBxCH enclosure, down the USB cable shield, onto the PC enclosure, and hopefully onward to the third wire on the wall power plug. If the case of the PC is not grounded in such a fashion, it should be. Field installations should be equipped with a ground stakes to dissipate the static burst.

Foil shields connected to properly grounded conductive enclosures form a Faraday cage around the electronic equipment keeping the static discharge on the outside. If your equipment is exposed to ESD, building a Faraday cage in this fashion is recommended.

15.6 User configuration byte

The polarity of the timing and GPS signals on the DB25 can be programmed by the user. For example, you may want to enable or disable the Adrdy signal, or change the idle state of PPS. These selections can be made with the UserCfgByte passed to the Open function. The bit layout of UserCfgByte is shown in the following table:

| Bit | Function | Values | Power on default |
|-----|-----------------|--------------------------|------------------|
| 0 | TriggerPolarity | idle low or high | idle low |
| 1 | reserved | - | - |
| 2 | Adrdy | disabled or enabled | disabled |
| 3 | reserved | - | - |
| 4 | PpsPolarity | idle low or high | idle low |
| 5 | PpsLedToggle | pps toggle or programmed | pps toggle |
| 6 | NmeaRxPolarity | idle high or low | idle high |
| 7 | NmeaTxPolarity | idle high or low | idle high |

Figure 15.4: USBxCH UserCfgByte bit assignments

Most users will specify the values of these configuration bits by setting the values of ini keywords in programs like Scope.

15.7 Programming the front panel red and yellow LEDs

The front panel red and yellow LEDs are always under hardware and program control and serve various functions depending on the application.

The red LED typically indicates the GPS PPS signal. It will toggle with every PPS tick on the DB25 connector.

The yellow LED typically indicates when a buffer of analog data has been acquired and transferred to the host PC. If the yellow LED is toggling while Scope or DVM is running then host PC communication is alive and well.

15.8 Seeing the digital inputs in Scope

Use the **Scope** program to see the digital inputs in real time. The digital inputs are saved synchronously along with the 24 bit analog channels.

Chapter 16

GPS Time Stamping

If you have a GPS receiver you can use it to accurately *time stamp* data acquired with the USBxCH. Using a particular receiver is usually only a matter of making the correct cable connections. If you don't have a GPS receiver, Symmetric Research carries ready to go antennas with the required cabling.

Besides GPS time stamping, the USBxCH also saves the GPS location. This is useful for applications with multiple systems that must know locations to triangulate. Both GPS time and location are saved continuously as part of the USBxCH data stream.

For sites where GPS reception is not possible, the USBxCH can time stamp with NTP (network time protocol). Although less accurate, NTP is useful when open sky visibility is not available for GPS. NTP time stamping is covered in the next chapter.

16.1 What is GPS ?

As we all know, GPS is a system of earth orbit satellites transmitting signals designed to determine location and time. A GPS receiver uses multiple satellites to triangulate its location and then computes precise time by making propagation delay corrections. The final time accuracy far exceeds traditional methods like WWV.

By far the most popular use of GPS is to determine location for travel and mapping. However, for many applications *accurate time* is equally essential, allowing for the precise correlation of analog data between widely separated installations.

GPS receivers are surprisingly small, often no larger than a hockey puck. They are complete stand alone units receiving satellite signals directly. One downside to GPS is buildings, cement, and other obstacles may all interfere with reception. In those cases a long cable may be required to place the GPS receiver where reception is possible.

Some GPS receivers combine their antenna and receiver into the same enclosure, while others separate the two. If the antenna and receiver are separated, they will be connected by a RF coax cable. With either type of packaging, there is a final cable from the receiver with digital signals indicating the location and time. It is these digital signals the USBxCH requires to time stamp and locate its data.

Two major GPS manufacturers are Garmin and Trimble. Garmin usually combines the antenna and receiver into the same enclosure, while Trimble often has a separate antenna. The USBxCH is compatible with both of these brands and many others. The general nature of the final GPS digital output is reasonably standard, and the USBxCH has programmable features to accommodate several of the common variations. As a specific example, this chapter will show how to connect a Garmin GPS 16x HVS.

16.2 Required GPS signals

The USBxCH requires *two* specific digital outputs from the GPS receiver:

The first GPS output is a RS232 serial signal. This signal has location and *coarse* time information. It is comprised of ASCII strings which are emitted once per second in predefined formats, with the most popular being NMEA (national marine electronics association) format. Because NMEA strings are only emitted once per second, it is difficult to determine time to better than one second with them alone, and so ...

The second GPS output is a PPS (pulse per second) TTL/CMOS signal. The leading edge of this signal is precisely aligned with the global GPS second tick. The accuracy of the PPS edge is usually within a few microseconds, where expensive receivers may even achieve nanosecond accuracy. The USBxCH is designed to use PPS in conjunction with the RS232 NMEA strings to time stamp acquired data entirely on board. The time stamping is done automatically with a dedicated FPGA. Accuracy is maintained regardless of cpu loading, interrupts, USB activity, or other system considerations.

16.3 DB25 pin assignments

The GPS receiver NMEA and PPS signals should be connected to the USBxCH DB25 front panel connector. Use a *female* DB25 to make the connections to the front panel *male* connector. The following table gives the DB25 GPS pin assignments and a diagram of the connector. These pins are also covered in the [Digital IO](#) chapter.

All signals must be reference to the USBxCH DB25 ground pins. *This is crucial.* Any noise, glitch, or grounding problems with respect to the USBxCH will result in unreliable time stamping. It is the user's responsibility to provide *clean* GPS signals *at the front panel DB25* in order to have reliable time stamping.

| DB25 pin | GPS signal | USBxCH | Power on polarity |
|-------------------|---------------|--------|-----------------------|
| Top row | | | |
| 11 | PPS | input | idle low, active high |
| 12 | RS232 NMEA RX | input | idle high |
| 13 | RS232 NMEA TX | output | idle high |
| Bottom row | | | |
| 14 - 25 | Ground | - | - |

Figure 16.1: DB25 GPS pin assignment table

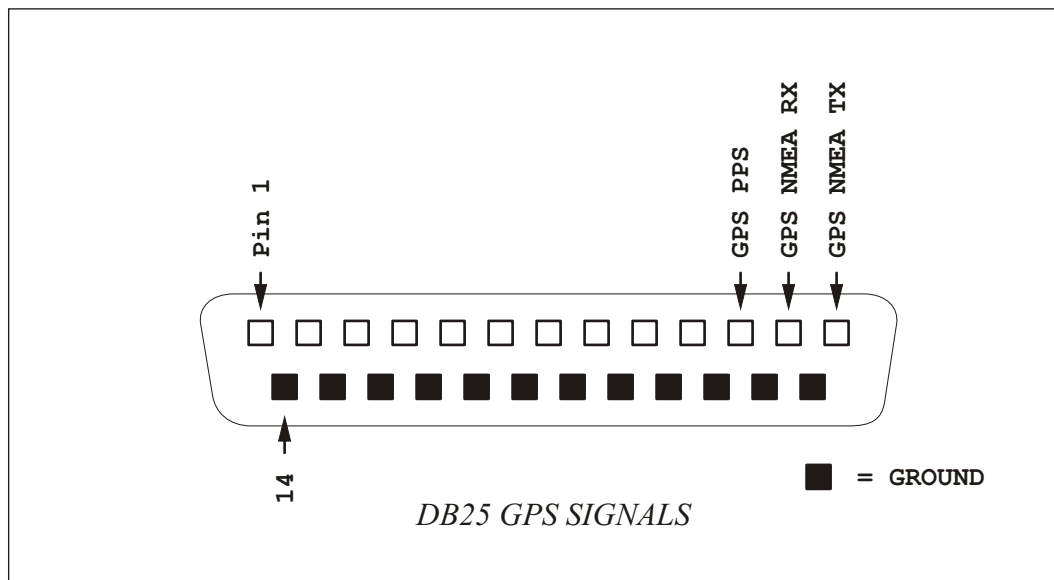


Figure 16.2: DB25 GPS signals viewed from front panel

Note besides having a RX pin to listen to the GPS NMEA output strings, there is also a TX pin. Most GPS receivers are configurable, and this allows the USBxCH to program receiver features. The TX programming strings are defined as part of the NMEA standard and are an output from the USBxCH point of view.

Also note the NMEA RX pin can accept a variety of input voltage levels. Some receivers may output full +/- 12 volt RS232, while others may emit lower level TTL/CMOS signals. The RX pin accepts either. All other DB25 pins expect + 3.3 volt digital levels.

The power on defaults for the signal polarities are given in the last column of table 16.1. For an explanation of selecting different polarities with the UseCfgByte, see the sections RS232 polarity and PPS polarity.

16.4 Using a Garmin 16x HVS with the USBxCH

As a specific example, this section shows how to connect a Garmin GPS 16x HVS receiver to the USBxCH. The Garmin 16x is similar to many GPS receivers, and the steps given here should be similar to those for others. If you have difficulty making cable connections, SR offers ready to go antennas.

The Garmin 16x is a rugged unit designed for outdoor use. It can be mounted on flat wall surfaces and the ends of pipes with flanges and adapters. The unit has its antenna and receiver packaged together in a single hockey puck enclosure. For connections, there is a round multiconductor cable with outputs, inputs, and power for the puck. The following lists the Garmin multiconductor cable wire assignments:

| Garmin wire | Color | Signal | Desc |
|-------------|--------|-------------------|----------------------------|
| 1 | red | power | +12 VDC power |
| 2 | black | ground | power and signal ground |
| 3 | yellow | enable on/off | power on = low |
| 4 | blue | RS232 NMEA in | NMEA config programming |
| 5 | white | RS232 NMEA out | NMEA string output +/- 12 |
| 6 | gray | PPS | Pulse per second |
| 7 | green | port 2 / data in | Garmin reserved / not used |
| 8 | violet | port 2 / data out | Garmin reserved / not used |

Figure 16.3: Garmin GPS 16x HVS wires

To hook up the Garmin 16x, its cable must be terminated with suitable connectors. In this example, we will use a female DB25 connector for signals, and a 2.1mm barrel connector for power. Note the Garmin cable has only one common ground wire for both power and signals. This single ground wire should be connected to a USBxCH DB25 ground pin to avoid ground bounce at the DB25. Figure 16.4 shows the wiring diagram. Note also the Garmin cable has a foil shield. Garmin recommends *not connecting the foil shield to anything* because it is part of the antenna system. We have followed that recommendation. Finally, because the 16x HVS can be powered from a wide range of supply voltages, we have hooked it up to the USBxCH wall transformer using the daisy chain 2.1mm connector on the USBxCH back panel. If you are using an antenna that must be powered from a regulated voltage such as +5 volts, *do not connect it to the wall transformer*. The voltage will be too high. In those cases you must provide the antenna with its own +5 volts. The photo in Figure 16.5 shows the finished cabling with the 2.1mm power and DB25 connectors.

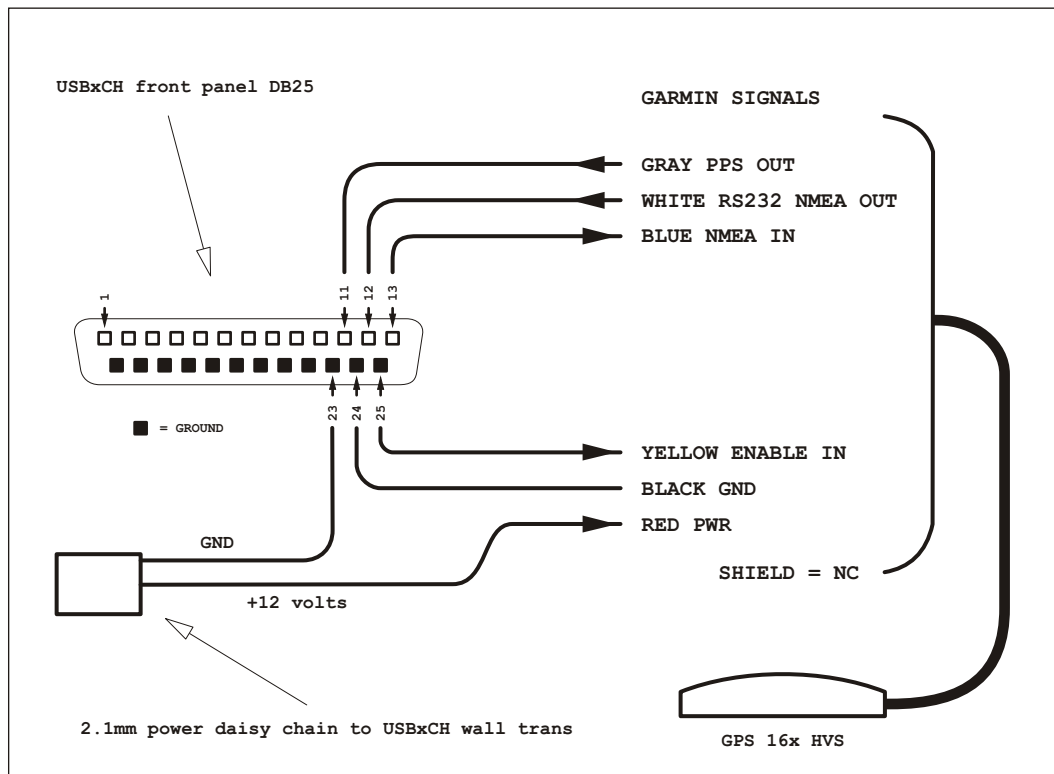


Figure 16.4: Garmin GPS 16x HVS wiring

16.5 Determining RS232 polarity

Suppose you are using a receiver other than the Garmin 16x. It may have a RS232 polarity different than the USBxCH power on default. Even within a particular GPS vendor catalog there are likely to be differences between models that must be accommodated.

To make a successful RS232 connection, there are two major considerations. The RS232 *baud rate* and *polarity*. The USBxCH expects a 4800 baud rate signal, and the GPS receiver *must be configured for that rate*. Most GPS receivers come with a utility program to set their baud rate, and you should use it to set 4800 baud. The RS232 polarity is generally less configurable, so the USBxCH has provisions to accept inverted or non inverted RS232.

RS232 polarity comes in *two* styles: Full +/- 12 volt RS232, or TTL/CMOS. Typically, 12 volt RS232 has inverted polarity, while TTL/CMOS RS232 is non inverted. Even if the signal levels are not +/- 12 volts, you may still have inverted outputs. You can determine the polarity by studying the output on a scope. If the output idles high between characters, then the signaling is 12 volt style and the USBxCH should be programmed to idle high. You can also probably do this measurement with a multimeter, watching activity between



Figure 16.5: Garmin GPS 16x HVS finished cabling

bursts of signal. Even if not a full + 12 volts, if the output reads more than a volt or two between bursts of activity you probably have 12 volt inverted idle high signaling.

If on the other hand the RS232 NMEA string output idles near 0 volts between bursts of activity, then you probably have TTL/CMOS style signaling and the USBxCH should be programmed to expect non inverted idle low.

Selecting RS232 idle high or low is programmed with the USBxCH **UserCfgByte**. Most users should specify an antenna selection with **DVM** or **Scope** ini keywords.

16.6 Determining PPS polarity

As with the RS232 polarity, GPS receivers also vary in their PPS polarity. It is most common for PPS to idle low and the *rising edge* to indicate the PPS tick.

However, there are receivers that idle high and use the *falling edge* of PPS to indicate the tick. You can accommodate the variation with the USBxCH **UserCfgByte**. The PPS convention used by a particular receiver can be determined with an oscilloscope or multimeter and observing the behavior between bursts of activity. Note if you have the USBxCH pro-

grammed for idle low but the GPS receiver actually idles high, your system will appear to work ... but the time stamps will be off by the width of the PPS pulse!

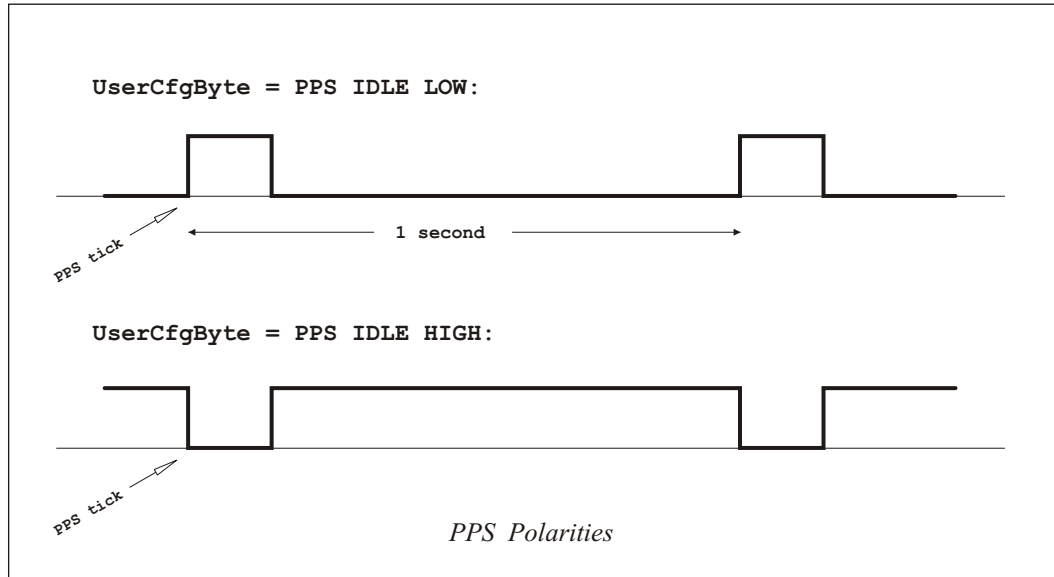


Figure 16.6: PPS signal polarities

The PPS pulse width does not matter to the USBxCH as long as it is greater than 100 *nanoseconds*. Within that constraint, it is only the leading PPS edge that signals a tick. In practice, we have never seen a PPS tick narrower than 10 microseconds, or wider than 0.5 sec. The 100 ns minimum width is easily met by most receivers. Note that if the PPS width is as wide as 0.5 sec you will have trouble determining the polarity with a multimeter. Consult the receiver's User Manual in that case.

As with RS232 polarity, the easiest way to make a PPS polarity selection is with ini keywords.

16.7 Front panel red LED

If you have a GPS antenna hooked up and successfully receiving, the front panel red led should toggle on and off once a second in response to the PPS signal. If it is not toggling, there is a problem with PPS. It may be as simple as the GPS receiver taking awhile to get full satellite lock at power on. A half hour to get lock from a cold start is not uncommon with many receivers. If the receiver has been on for awhile and successfully sending PPS, then perhaps the number of satellites has fallen too low, or the receiver needs better visibility. If you see the red led toggling, then you know PPS is being signaled by the antenna.

16.8 Expected NMEA strings

Most GPS antennas can be programmed to output a wide variety of RS232 NMEA strings. In fact, it is even possible to program many antennas with obviously incorrect settings such as emitting more NMEA characters than it is possible for the RS232 baud rate to transmit in 1 second. So ... care is required.

The USBxCH hardware and software expect particular NMEA strings, and the GPS antenna should be programmed accordingly. If the antenna is programmed differently the USBxCH may give incorrect results. Antennas supplied by SR will already be programmed with the correct NMEA string line up.

The first consideration for the NMEA strings is the total number of characters emitted. The USBxCH hardware buffer can only hold 255 characters per second. A NMEA string lineup less than this must be selected. The second consideration is the strings themselves. Many NMEA strings do not have sufficient coarse time and location information to provide the quantities needed to compute the time stamp. In particular we recommend the antenna be programmed to output the following two strings in this order:

| | |
|---------|-------------------------------|
| \$GPRMC | for coarse time and date info |
| \$GPGGA | for location information |

All other strings should be turned off and not emitted. For a description of the Garmin NMEA strings, please refer to section 4.2.2 in:

</SR/USBXCH/Docs/GpsSpecs/GPS16x.TechnicalSpecifications.pdf>

A copy of this Garmin document is included with the SR software distribution for convenience. In particular, the RMC and GGA string pairs expected by the USB4CH will typically read as:

```
$GPRMC,185442,A,3608.9181,N,11517.9605,W,000.0,000.0,150310,013.4,E*6F
$GPGGA,185442,3608.9181,N,11517.9605,W,2,10,1.2,880.3,M,-26.0,M,,*70
```

where the comma separated fields contain the time and location information. For example in the RMC sting, the A indicates the antenna has lock and the fields are valid, while the N indicates the latitude is in the northern hemisphere, and so forth. Note each string is terminated with a *xx checksum. The field values will of course vary depending on when and where you run the system. See the Garmin document for details about all the fields. When using Scope and Blast, you *do not have to decode* the time and date fields. Those programs and their conversion utilities automatically take care of the decoding. To simply verify the NMEA strings are being received, use the **NmeaTime** or **Diag** utilities.

16.9 Programming the GPS antenna

If your Garmin or Trimble GPS antenna came from Symmetric Research, it will already be programmed correctly for the USBxCH. Users with other antennas will need to program the baud rate, NMEA string lineup, and PPS enable to be compatible. Note that *even the Garmin and Trimble factory defaults* are usually not compatible with the USBxCH. They tend to output too many strings and have PPS disabled.

Included with the USBxCH software distribution is a Garmin utility for setting the basic antenna features. Besides programming the NMEA strings, be sure to also enable the PPS signal. You can find a copy of the Garmin utility in:

```
/SR/USBXCH/Docs/..  Garmin ../snsrxcfg_200.exe
```

You will need a computer with an RS232 port and a custom cable. See the ReadMe.txt file in this directory for more info. Garmin does not supply a programming utility under Linux.

You can also use the SR utility **GpsProg** to program a Garmin antenna *while connected to the USBxCH*, rather than a separate RS232 port. Both Windows and Linux versions of this SR command line utility are supplied.

16.10 Seeing the GPS time stamps in Scope and Blast

To see the time stamps from **DVM** or **Scope** examine their output files. Time stamps will appear as a columns associated with each row of samples.

The following would be a typical fragment:

| # | Sample | Time in Seconds | Time as YMD HMS |
|------------|--------|-------------------|----------------------------|
| 0000000000 | | 1268679282.069689 | 2010/03/15 18:54:42.069689 |
| 0000000001 | | 1268679282.070457 | 2010/03/15 18:54:42.070457 |
| 0000000002 | | 1268679282.071225 | 2010/03/15 18:54:42.071225 |
| 0000000003 | | 1268679282.071993 | 2010/03/15 18:54:42.071993 |
| 0000000004 | | 1268679282.072761 | 2010/03/15 18:54:42.072761 |
| 0000000005 | | 1268679282.073529 | 2010/03/15 18:54:42.073529 |
| 0000000006 | | 1268679282.074297 | 2010/03/15 18:54:42.074297 |
| 0000000007 | | 1268679282.075065 | 2010/03/15 18:54:42.075065 |
| 0000000008 | | 1268679282.075833 | 2010/03/15 18:54:42.075833 |
| 0000000009 | | 1268679282.076601 | 2010/03/15 18:54:42.076601 |
| 0000000010 | | 1268679282.077369 | 2010/03/15 18:54:42.077369 |
| 0000000011 | | 1268679282.078137 | 2010/03/15 18:54:42.078137 |
| 0000000012 | | 1268679282.078905 | 2010/03/15 18:54:42.078905 |

"Time in Seconds" is measured in seconds since 1970.

"Time as YMD HMS" is the same as (Year, Month, Day, Hour, Minute, Seconds).

This particular data was acquired at 1302Hz. At that rate the sample period is $1/1302 = 0.0007680$ seconds, and you can confirm each time stamp is indeed incrementing by that amount. In fact the GPS controlled time stamping is so accurate you can even measure the frequency of the on board 10MHz master A/D clock controlling sampling rate with extreme precision. If you wish to also see the latitude and longitude displayed as a columns, run the conversion utilities with the **showall** command line option.

16.11 What does time stamping mean ?

Many users ask if the USBxCH is synchronized to the GPS PPS tick. The answer is:

The USBxCH *does not* phase lock its master clock (or Adrdy) to PPS

A true statement is:

The USBxCH uses GPS to *time stamp* the data it acquires. When an A/D sample *is available*, the GPS time at which it became available is recorded as part of the data stream.

This statement should be taken literally. For example, the time stamp *does not even indicate the exact instant when the sample was acquired*. Analog samples acquired with sigma delta A/D converters are averaged results over the sampling period. *The time stamp only indicates when the end of the A/D oversampling period occurred.*

The next question often is, how accurate is the time stamp recorded at the end of the oversampling period? It depends on two factors. The first is the accuracy of the PPS signal itself. Most receivers have accuracies in the low microseconds, while a few may be as good as nanoseconds. The length of the cable connecting the GPS receiver to the USBxCH can also be a similar issue. Two installations with different cable lengths will not deliver their PPS ticks at the same instant at the USBxCH DB25. If you need better accuracy, you have no choice but to improve the receiver or cabling.

The second factor is the accuracy of the USBxCH itself. This is guaranteed to be 800 nanoseconds worst case. The USBxCH time stamp is computed by a dedicated onboard FPGA controller, and is *always within 800 ns of the PPS tick*. No cpu polling, interrupt, or other activity is involved.

We are also often asked if the time stamps are the same across all the USBxCH channels. Since each channel is clocked from the same master clock and controlled in parallel by the same FPGA, there is *no skew between channels*. Each channel has identically the same timing environment. Each channel asserts Adrdy at exactly the same nanosecond, and the associated time stamp is the same across all.

Finally, users sometimes ask about using timing signals other than GPS. For example, there are systems generating PPM (pulse per minute) signals. With appropriate interpretation of the resulting data files, these systems can work with the USBxCH. With a PPM signal, timing marks will simply occur in the output files every minute rather than every second.

16.12 Driving multiple USBxCH systems from one GPS

It is possible to drive *several* USBxCH systems from *one* GPS receiver. Doing so only requires daisy chaining the GPS NMEA RS232, PPS, and ground from one USBxCH system to the next, connecting them all in parallel.

If you choose to daisy chain GPS in this fashion, *DO NOT use a full width DB25 ribbon cable* when connecting one USBxCH DB25 to the next. This will unintentionally connect the *digital outputs* on one system to the next, creating contention. The DB25 connectors have many signals and you must treat them all appropriately. Only daisy chain the NMEA, PPS, and ground with discrete wires.

Chapter 17

NTP Time Stamping

NTP (network time protocol) can be used to time stamp data acquired with the USBxCH. Although less accurate than GPS, NTP is useful for applications where GPS is not possible because of physical location, expense, or power consumption.

First set the host PC to listen and sync its clock to NTP. This can usually be done with utilities provided by the operating system. Consult the host PC system documentation.

Next, set the USBxCH ini files (probably in **Blast**) to use the PC clock as its time source. Using the PC clock this way has its advantages. All hardware running in parallel on the PC has access to the same time base.

NTP has its inherent disadvantages. Because USB, Web, or Ethernet delays all depend on the activity of the attached peripherals, NTP does not have the absolute accuracy of GPS. By comparison GPS computes its time signals directly on the USBxCH board with a custom controller giving consistent microsecond accuracy.

17.1 Setting the PC clock from GPS

Rather than setting the PC clock from NTP time, users may want to set the coarse PC time and date from their GPS antenna.

Use the **Nmeatime** utility for this, calling it in a batch file setup.

Chapter 18

Power Supply

Reliable power is critical to the USBxCH. Many seemingly mysterious problems can be the result of unstable power. The power supply must have sufficient voltage and current for the system to work correctly. There must be a solid ground connection between the power source, USBxCH, PC, and other peripherals without a voltage difference between one ground point and another. Failure to give adequate consideration to these aspects of the power supply will result in unreliable operation. This chapter covers some of the basics.

18.1 Power connectors

Power connectors for the USBxCH are on the back panel. Most users will already be familiar with the wall transformer 2.1mm barrel connector. Actually, there are two 2.1mm jacks on the back panel connected in parallel. This makes it easy to daisy chain several devices from the same wall transformer. It does not matter which 2.1mm jack the wall transformer is plugged into and which is used as the daisy chain.

Besides the 2.1mm power jacks, there is also a 4 pin molex power header which is available when using the USBxCH as a bare board. It is intended for field box applications where the USBxCH is part of a hardware stack. In such applications, power is often distributed on discrete wire cables with molex terminations.

It is important the power cables for the USBxCH be of sufficient gauge. 24 gauge stranded wire or larger is recommended. The inductance of skinny wires may choke the power supply off when the USBxCH hits an execution peak and demands more power. It is particularly important *the ground wire be of sufficient gauge*. If the ground wire establishes system ground and all signals are referenced to it, then any ground bounce will cause false signaling and digital errors.

18.2 Voltage and current requirements

The USBxCH requires only a single positive power supply. *The supply must be DC.* AC supplies directly from an AC transformer *will not work*. So for example, the power from a 9vdc 500ma unregulated wall transformer is acceptable, while that from a 9vac 500ma wall transformer is not.

The off board power is conditioned and regulated by active circuitry on the USBxCH. In fact, there are multiple linear regulators to generate the precision voltages required for 24 bit A/D converters. See the [Power supply](#) circuit diagram pages.

Off board power may vary over a wide range. The minimum is 8 volts, and the maximum 24 volts. If the power goes below 8 volts, the board will detect the error and indicate it by turning on the back panel red LED, as well as setting the power status flag to an error. If you use an unregulated DC wall transformer, it is ok to have 50/60Hz power ripples, *as long as none of the ripples go below 8 volts*. AC wall transformers do not meet this requirement and the red LED will light up.

If possible, power supplies in the range of 9 to 12 volts are recommended. Because the USBxCH uses linear regulators and consumes approximately 100ma regardless of supply voltage, the excess power at higher supply voltages is dissipated as heat and the board will run hotter. Supplies as high as 24 volts are acceptable, but cooler electronics is always better, and those in the 9 to 12 volt range are ideal. Voltages beyond 32 volts simply exceed component specifications and will cause permanent damage even before the board overheats.

The current consumption of the USBxCH depends somewhat on the sampling rate and other activities of the board. Typically it is 100ma for a USB4CH with +/- 5ma variation. At power up, while on board capacitors are charging, the board may draw as much as 350ma. The off board power supply must be capable of supplying this amount of inrush current to successfully power up. The inrush current surge may last for 2 seconds, but no longer. Active on board circuitry *always limits* the total current consumption to 350ma +/- 10% even under most short circuit conditions.

Besides wall transformers, many other supplies meeting the voltage and current requirements are acceptable. This includes lab bench supplies as well as batteries. For example, you may power the USBxCH directly from 12 volt lead acid batteries. We have even seen 9 volt transistor radio batteries used. Of course, the length of time the system will run is directly proportional to the Ah (amp-hour) rating of the battery. A 9 volt transistor radio battery will not last very long.

When running from batteries, we highly recommend including a manual switch so the battery can be connected with one clean quick action. Sparks often occur when attaching an active load to a battery with alligator clips. Not only will the sparks bounce the power and probably crash the USBxCH, it is also dangerous if there are any flammable hazards

in the area. Use a sealed snap action toggle switch to make the final battery connection.

Wall transformers, batteries, and some lab power supplies are floating and not necessarily connected to any ground system. In such cases, be aware if other devices are using the same floating supply, because the USBxCH will be sharing the ground connection with those devices. In such installations make sure a heavy ground wire is used so all the devices share the same common ground. *CPU boards are often serious offenders in this regard.* They may draw many amps when hitting execution peaks, and create inductive ground bounces for all the other devices. *If your system is behaving unreliably, review the ground system carefully.*

18.3 LED power status indicators

On the USBxCH back panel there are two LEDs, one green, one red. Both are near the 2.1mm power connectors and give a visual indication of the power status.

The green LED will light up if there is any power supply voltage at all. Even power as low as 3 volts will light it up to indicate the wall transformer or other supply is energized at some level. If the green LED is off, then check for very basic problems. Things as simple as a wall plug not being turned on are common, or if you are running from batteries they may be dead. *If the green LED is not on, check the off board power and correct the problem.*

The red LED further indicates whether the USBxCH regulators are within specifications. *If the red LED is on, the on board regulators are not functioning correctly.* The problem is most likely the off board power supply voltage is too low. For example, if the power supply is 5 volts, the green LED will be on *and the red LED will also be on.* You must raise the power to at least 8 volts for the red LED to turn off.

The red LED may indicate other conditions besides low power supply voltage. For example, during power up the on board capacitors are charging and the regulators are momentarily out of specifications. In this case, the red LED should go off within 2 seconds and then the USBxCH will be ready to run. If the red LED stays on permanently, there is a problem that must be fixed. After verifying the off board voltage is sufficient, the red LED may be indicating there is a short on the USBxCH board itself and the current limit is activated.

Do not run for extended periods with the red LED on. Diagnose the problem.

18.4 Power History signal

An ongoing record of the power status is automatically saved once a second in programs like **DVM** and **Scope**. This information is continually reported in the pipeline data and final output files. Besides the power, system temperature is also recorded as described in the next chapter.

In Bin2Asc files PwrH (power history) is 1 if power is good, 0 if not. Power history is a latched value. If power goes bad at any time during the run, power history is permanently set to 0. It will remain 0 even if power returns to acceptable levels. Quitting and restarting is the only way to clear the PwrH bit. *Any data acquired while PwrH is 0 is invalid.*

18.5 Reset and power cycling

A hardware jumper is provided on the USBxCH board to force a full power reset cycle. This is the most drastic reset possible. The power supply performs a momentary power down of the entire board. The PC plug and play system will act as if the USBxCH has been unplugged from the system, and do a complete restart.

The power cycle feature is only available if the system is being used as a bare board. The power reset header is J940 on the back edge of the board near the molex power connector. Placing a short between its two header pins will trigger the reset. The short can be made with a header jumper, a piece of wire, a screwdriver, or an open collector output from a logic device.

Power cycling is particularly useful for field box applications. If the field box can only be remotely accessed you may want to periodically take the system down for a complete system reset.

18.6 Current limiting

Power consumption on the USBxCH is current limited. The current drawn from the power supply will never be more than 350ma. This includes power up inrush current. There are active power clamps enforcing a hard limit on the power. Normal power consumption after inrush current has subsided should be approximately 100ma. If more current is being consumed then something is wrong *and should be fixed.*

If the USBxCH is in current limit mode, the regulator heat dissipation will be excessive. Extended periods at the peak power supply voltage of 24 volts with the back panel red LED continuously on will push the thermal limits of what the board can withstand. The system will survive, but turn the system off and diagnose the problem.

Chapter 19

Temp Sensor

The USBxCH has an on board temperature sensor that is recorded once per second. With this information you can monitor system health and even perform corrections for voltage reference, op amp, and resistor temperature coefficient drifts.

The physical sensor is a Microchip TCN75. A manufacturer spec sheet is included in the software `Docs` directory. The component has a max error of +/- 2°C from - 40 to + 120°C. Of course the temp sensor does not record the temperature at the exact location of the A/D voltage reference or any other particular component. However, being a multilayer board with copper ground and power planes, the temperature is reasonably uniform across the board. This is particularly true if the system is inside a field box along with other equipment.

19.1 Where is the temp data stored ?

Temp is recorded automatically and does not require any special actions by the user. The temp reading is saved once per second as part of a status packet included with the GPS NMEA strings. If you don't have a GPS antenna you can still get the system status reports by setting the `DVM` or `Scope` ini file parameter "`gn`" to `PC.TIME`. The temp is displayed on the DVM and Scope GUI screens and saved to their data files.

Plotting the system temp along with the other data health status indicators like power condition is an important validity check on the analog data acquired.

Chapter 20

Specifications

The table on the next page lists the USBxCH operating specifications. Many performance parameters are also discussed in more detail in their respective chapters.

Manufacturer spec sheets for several of the leading analog components are also included in the software Docs directory:

| | | | |
|-----------------------|---|-----------|---------------------|
| A/D converter | = | ADS1255 | (Texas Instruments) |
| A/D voltage reference | = | REF02AP | (Texas Instruments) |
| Front end op amp | = | OPA2277UA | (Texas Instruments) |
| Temp sensor | = | TCN75 | (Microchip) |

20.1 Specifications table

| USB4CH Specs | | | | | |
|---------------------------|------------------|--------|----------|-----------|------------------------------|
| Parameter | Comment | Min | Typ | Max | Units |
| Analog input: | | | | | |
| Input voltage range (1) | | - 4.0 | | + 4.0 | volts |
| Sampling rate (2) | | 3.26 | | 9,765.625 | Hz |
| Resolution | | | 24 | | bits |
| Noise Floor (3) | at 130 Hz | | | 3 | bits |
| | at 1302 Hz | | | 5 | bits |
| Input Sensitivity (4) | | | 0.953 | | $\mu\text{V} / \text{count}$ |
| Input Impedance (5) | | | 51K | 10M | Ω |
| Overvoltage tolerance (6) | 10K series R | - 20.0 | | + 20.0 | volts |
| Digital IO: | | | | | |
| Logic 1 high voltage | | | 3.3 | | V |
| Logic 0 low voltage | | | 0.0 | | V |
| Output current | 300 ohm series R | | | 11 | ma |
| GPS interface: | | | | | |
| NMEA RX baud rate | fixed | | 4800 | | baud |
| NMEA TX baud rate | fixed | | 4800 | | baud |
| PPS pulse width | | 100 | - | unlimited | ns |
| Time stamp error | worst case | - | - | 800 | ns |
| USB cable: | | | | | |
| USB type / bit rate | USB 1.1 | | | 12 | Mbit/sec |
| USB cable length | | | | 6 | feet |
| Power supply: | | | | | |
| Supply voltage | | 8 | 12 | 24 | Vdc |
| Supply inrush current | 2 s at power up | | | 350 | ma |
| Supply current (7) | converting | | 100 | | ma |
| Supply current (8) | sleeping | | 2 | | ma |
| General: | | | | | |
| Temp range | | 0 | | 70 | $^{\circ}\text{C}$ |
| Board dimensions | | | 5.28 x 6 | | inches |

Figure 20.1: USB4CH specifications table

- (1) Min and max input on any + or - analog input pin for valid conversions
- (2) The USB 1.1 interface and DRAM FIFO bandwidth limit the sampling rate to this maximum
- (3) Absolute maximum noise floor measured in peak to peak bits, see 20.2 for plots
- (4) Input sensitivity at +/- 4 volt input range, see 13.2 for details
- (5) Input impedance is set by onboard resistors, 51K factory default, see 12.5 for details
- (6) Absolute min/max, beyond this damage occurs, 10K series R is internal to board
- (7) Power consumption varies little with conversion rate
- (8) Sleep mode occurs when Close is executed

20.2 Noise floor

The USBxCH noise floor varies with sampling rate. This is characteristic of all sigma delta A/D converters, where at lower final output rates there is more time for averaging the oversampled input. The typical performance at various rates is:

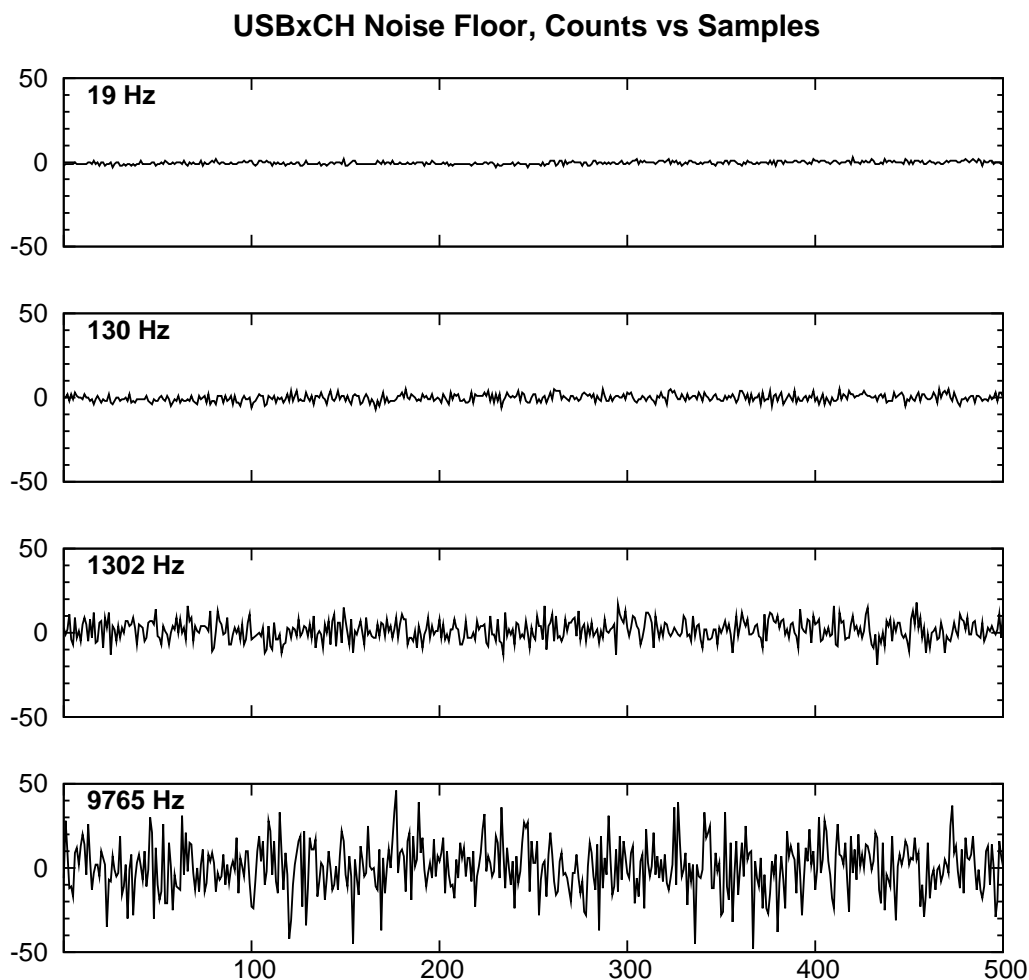


Figure 20.2: Noise floor variation with sampling rate

These graphs show the peak to peak noise versus time. The vertical axes are in counts, where 1 count is the 24th bit of the converted result. Similar results can be seen running the [Scope](#) program in real time. If the results from Scope are worse than the plots above, you have another noise source in the overall system besides the USBxCH.

Besides plotting the noise floor as a function of time, it is also useful to display the results as histograms:

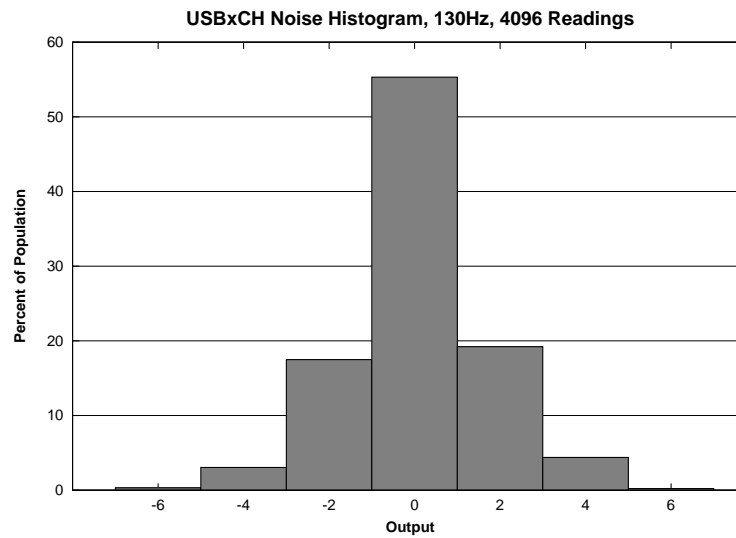


Figure 20.3: Noise floor histogram at 130Hz

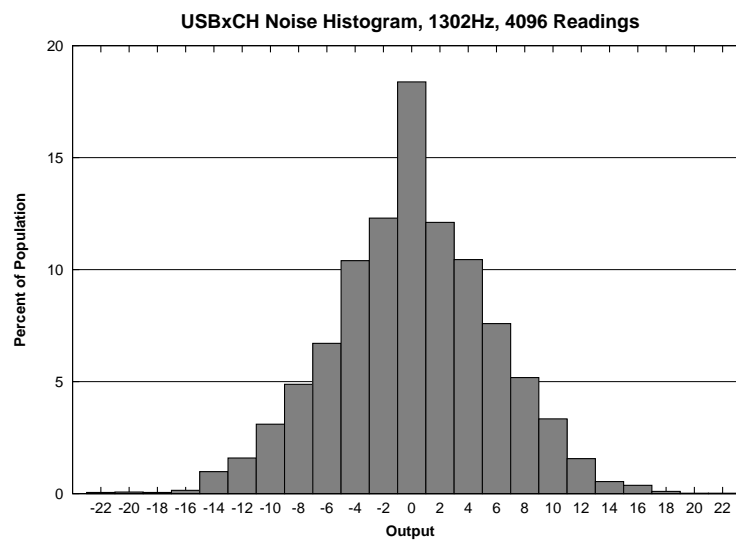


Figure 20.4: Noise floor histogram at 1302Hz

At the higher sampling rate, the width of the histogram becomes wider showing the increase

in the noise floor with sampling rate. At 130Hz, the width above the 1% level spans from -4 to +4 counts, a peak to peak total of 8 counts or 3 bits. At 1302Hz, the width spans from -16 to +16 counts, a peak to peak total of 32 counts or 5 bits. These results are in agreement with the specifications published by TI for the ADS1255 A/D converters, and include the noise performance of the USBxCH amplifier front end. To compare with the TI quoted specs, see the TI ADS1255 spec sheet, page 13 Table 6.

Chapter 21

Circuit Diagrams

The following schematics have the USBxCH front and back panel connectors, analog signal conditioning, A/D converters, and other details required to interface external equipment and sensors to the system.

Separate schematics are given for the USB4CH and USB8CH. They are similar in most respects. For each model the analog front end is given only once. Additional channels are simply the same circuit repeated for either four or eight times.

The *physical board* itself has only a few component id part designators printed in white ink for correlation with these diagrams. The board is too dense for part labels on each and every component to be legible.

Instead, view the board with the circuit diagram in hand. The physical layout of the parts is nearly identical between the board and diagrams. In the analog front end all signal traces are on the top layer of the board. Use a magnifying glass to follow the signal paths.

Some users will be interested in changing the analog input voltage ranges from the native +/- 4 volts to +/- 10 volts. See the [Input voltage range](#) of the [Analog inputs](#) chapter for more information. Items like RC filtering can also be modified in obvious ways for particular applications ... only make changes if you have experience with surface mount soldering and are knowledgeable about analog circuit design. *Customer modified boards are not covered by warranty.*

Portions of the system such as the USB microcontroller, DRAM FIFO, and power supply are proprietary and are not published.

21.1 USB4CH schematics

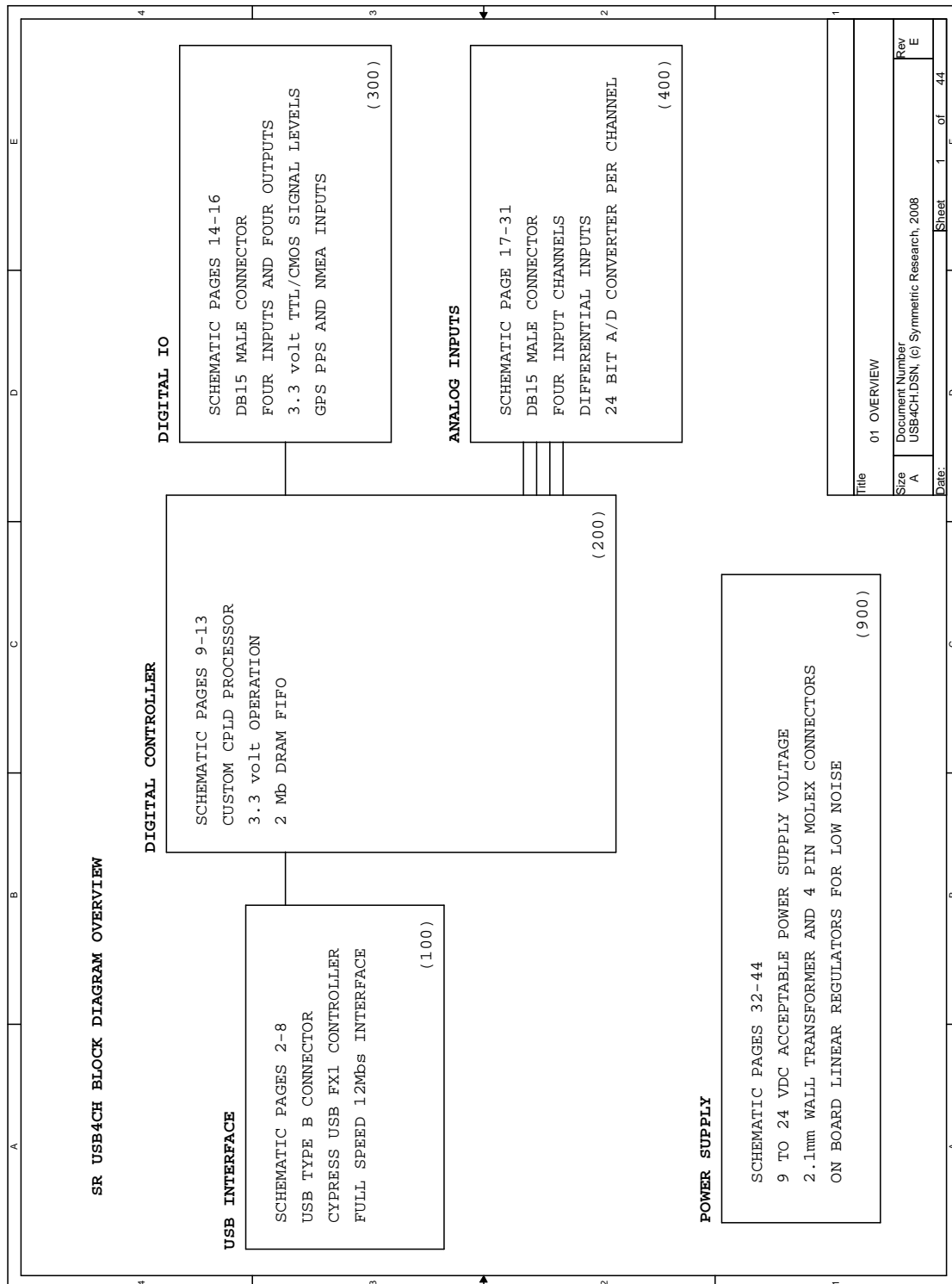
This section has the four channel USB4CH circuit diagrams. Those portions required for interfacing to external equipment and sensors are provided.

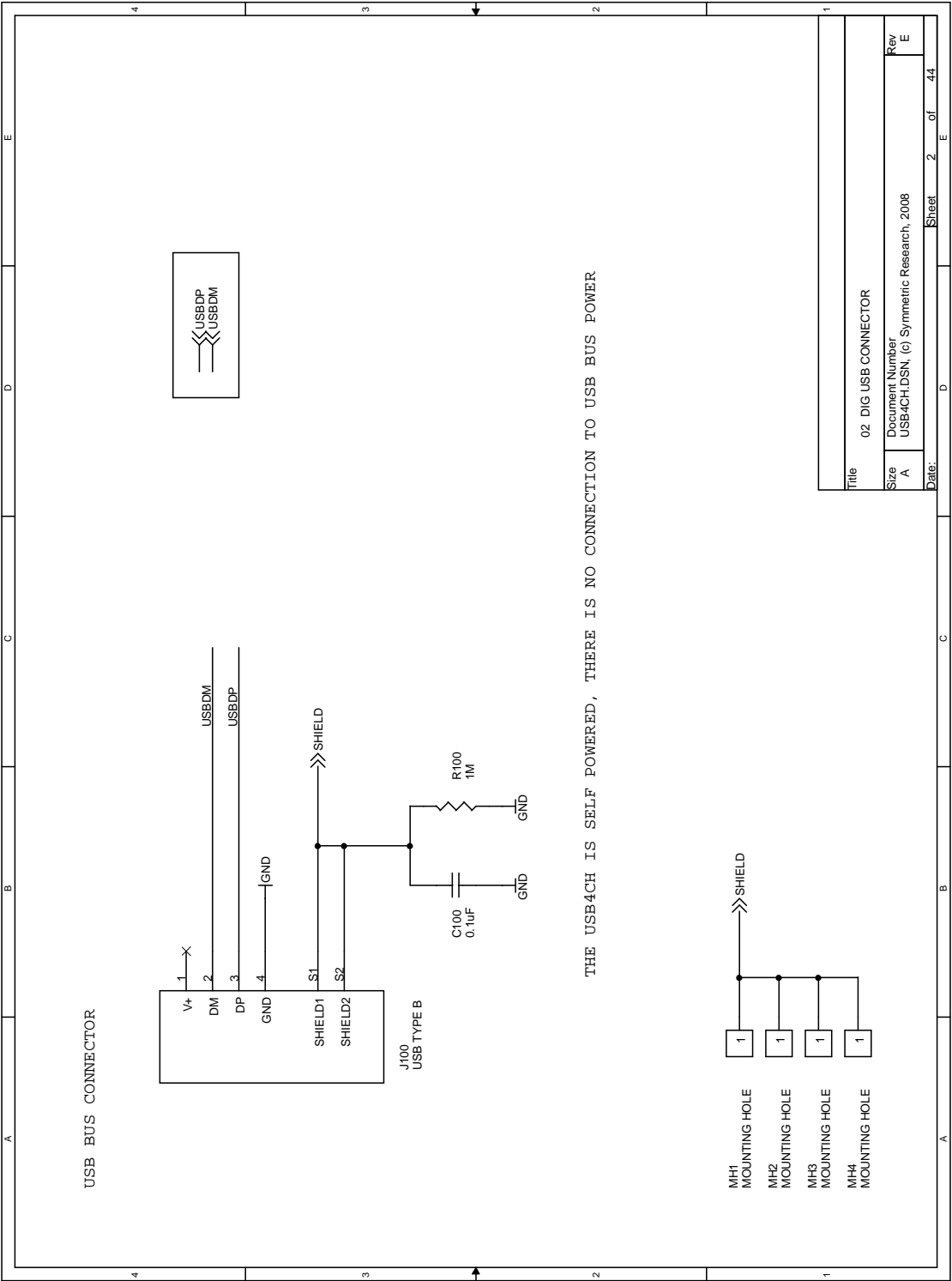
Repetitive parts of the circuit appear only once. For example, the analog front end is the same from one channel to the next and is given only once even though the physical circuit appears four times on the board.

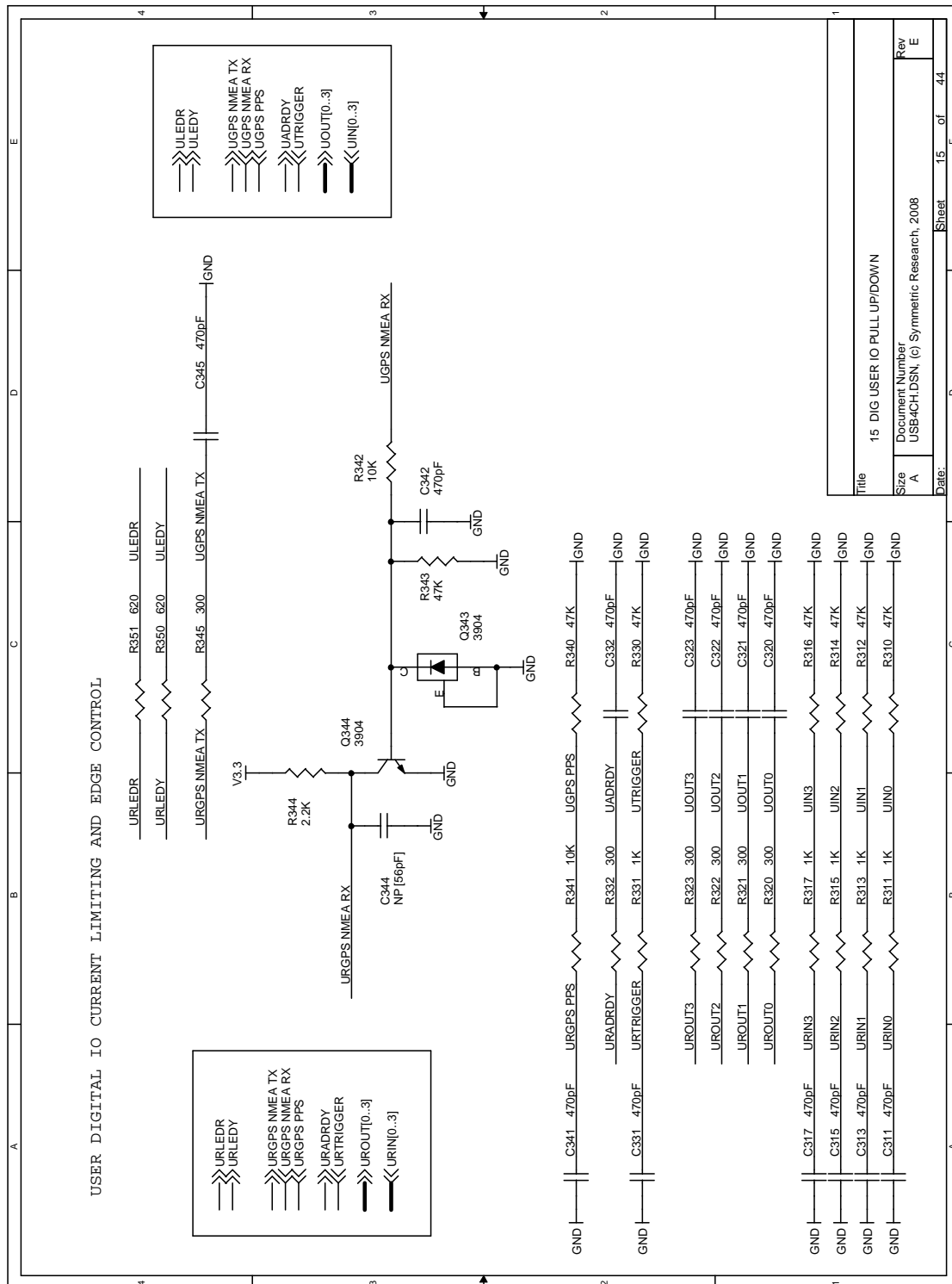
Use the Adobe Reader View/RotateView/Clockwise command for more comfortable horizontal viewing. See [Using Adobe Reader](#) for more viewing tips.

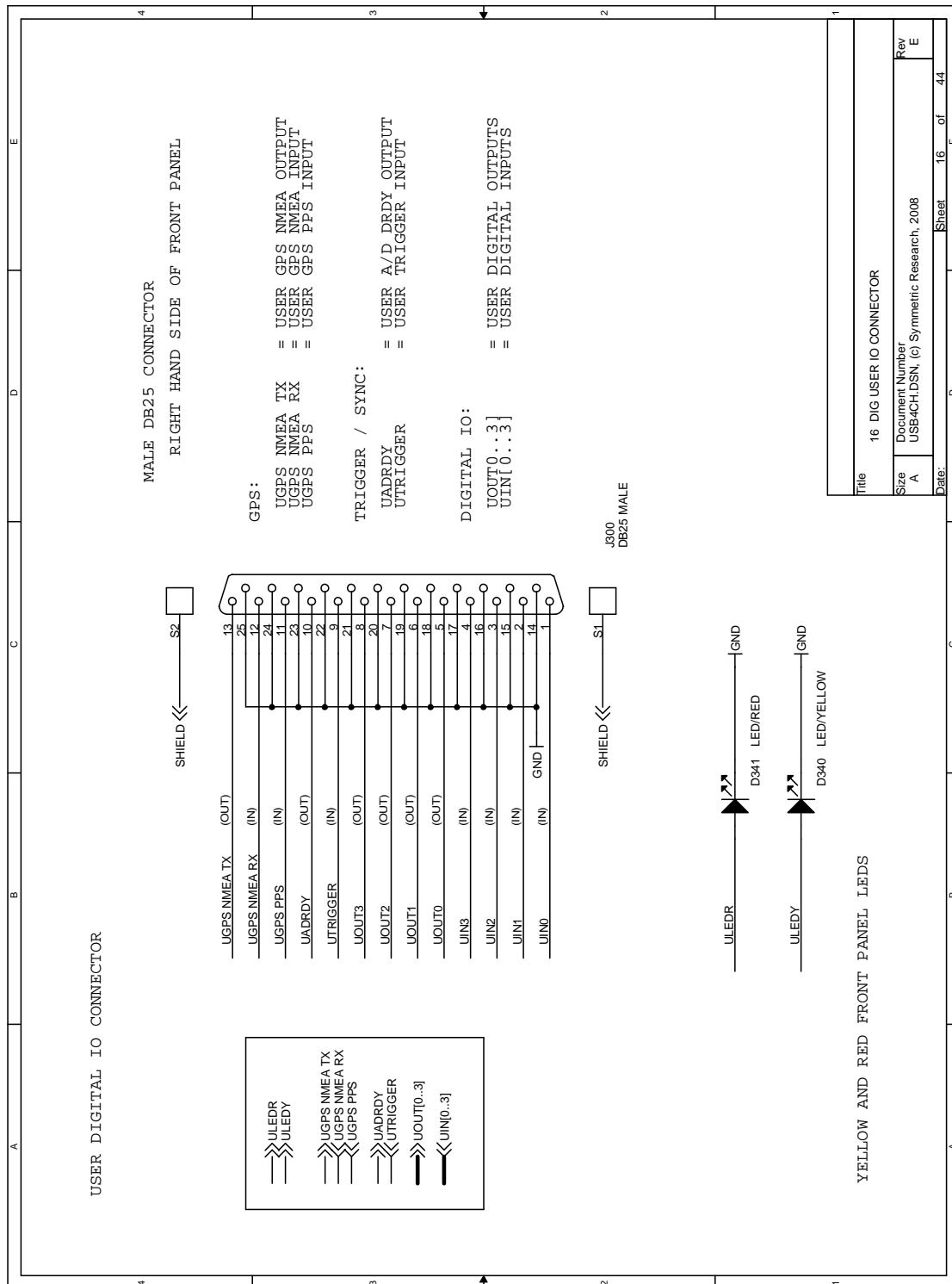
USB4CH circuit index:

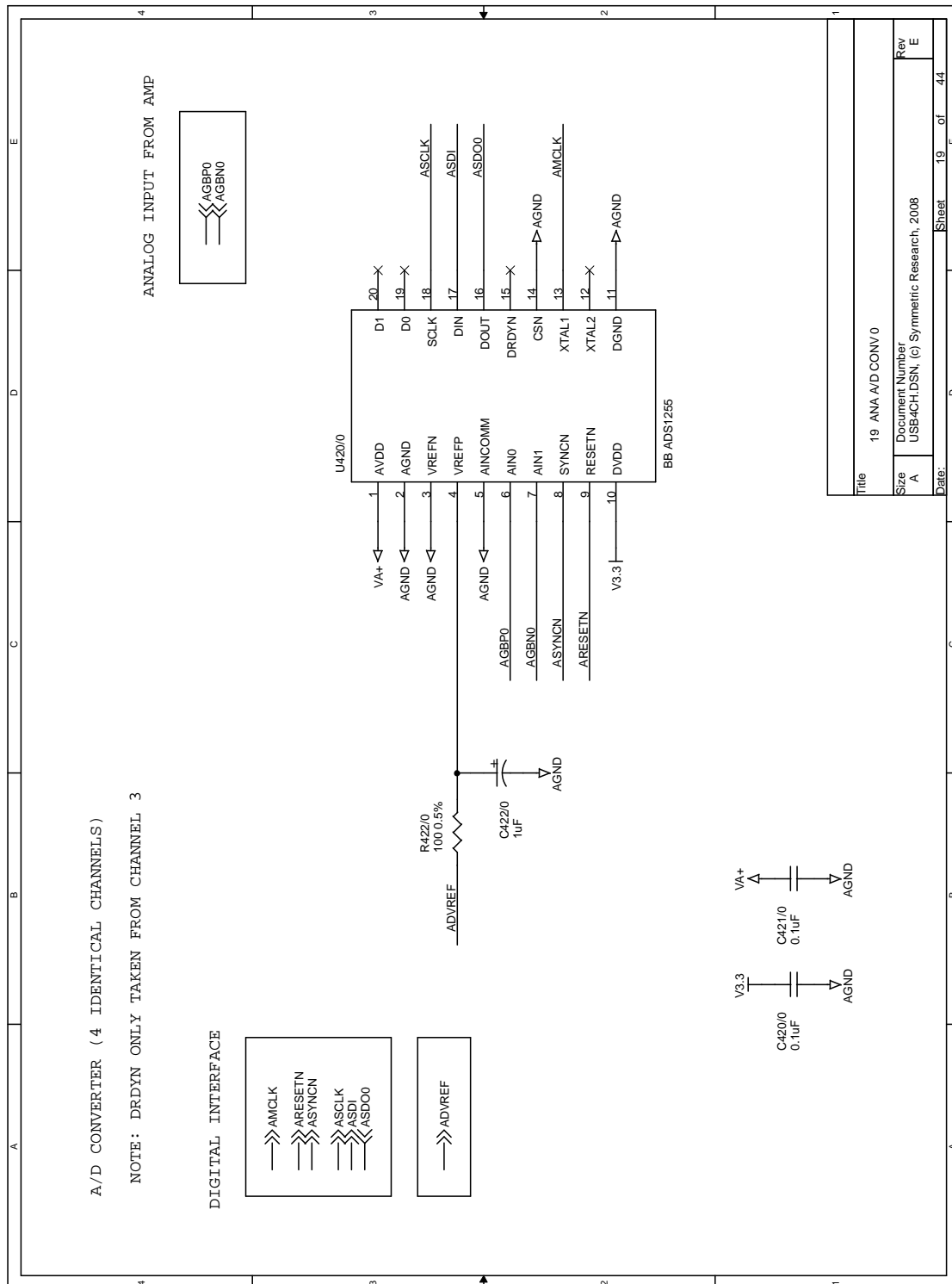
| | |
|---|-----|
| Circuit Overview | 115 |
| USB connector | 116 |
| User digital IO buffering | 117 |
| User digital IO connector | 119 |
| A/D converters | 120 |
| A/D input amplifiers | 121 |
| A/D signal conditioning | 122 |
| Analog input connector | 123 |
| Power supply connector | 124 |

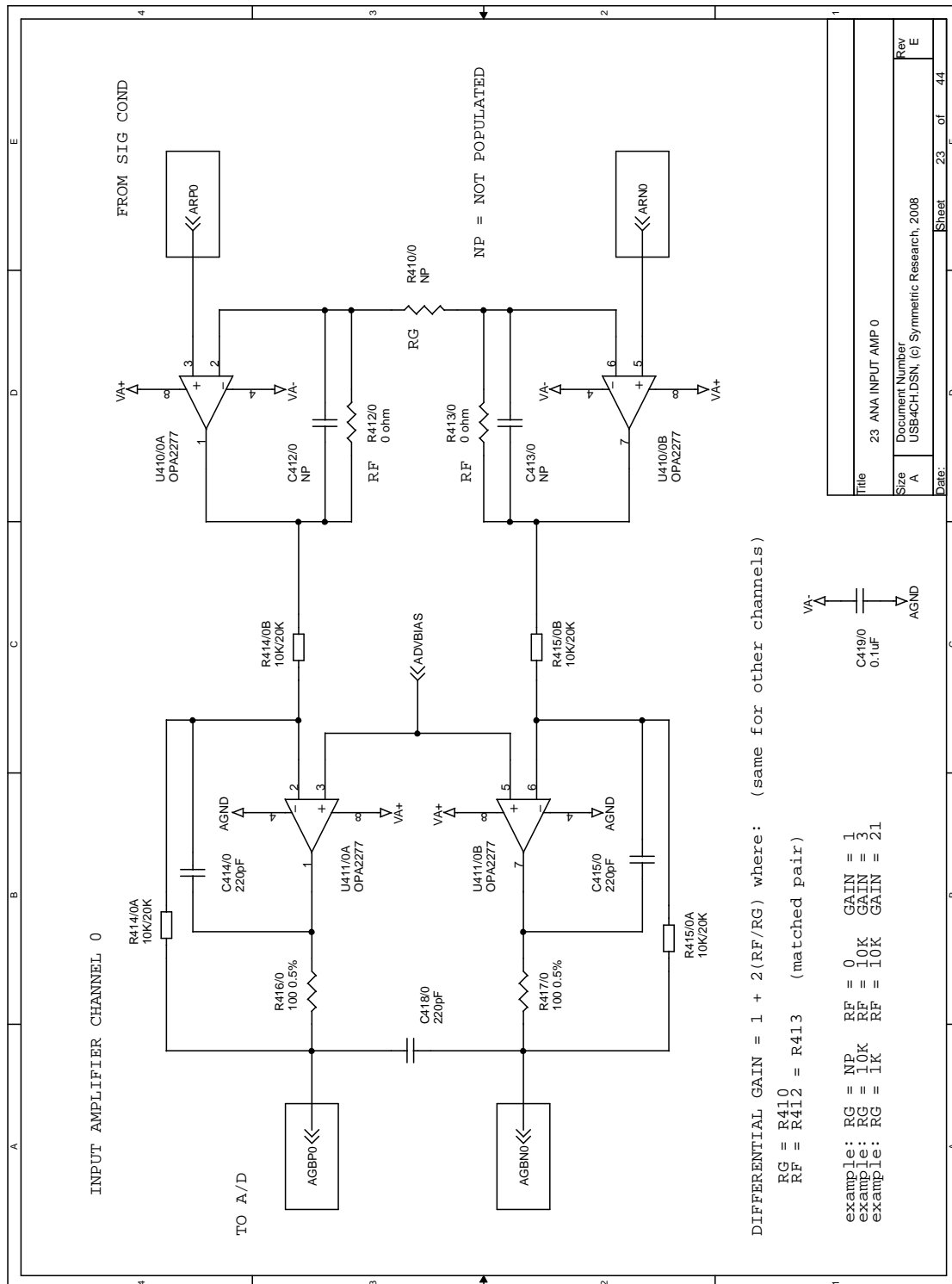


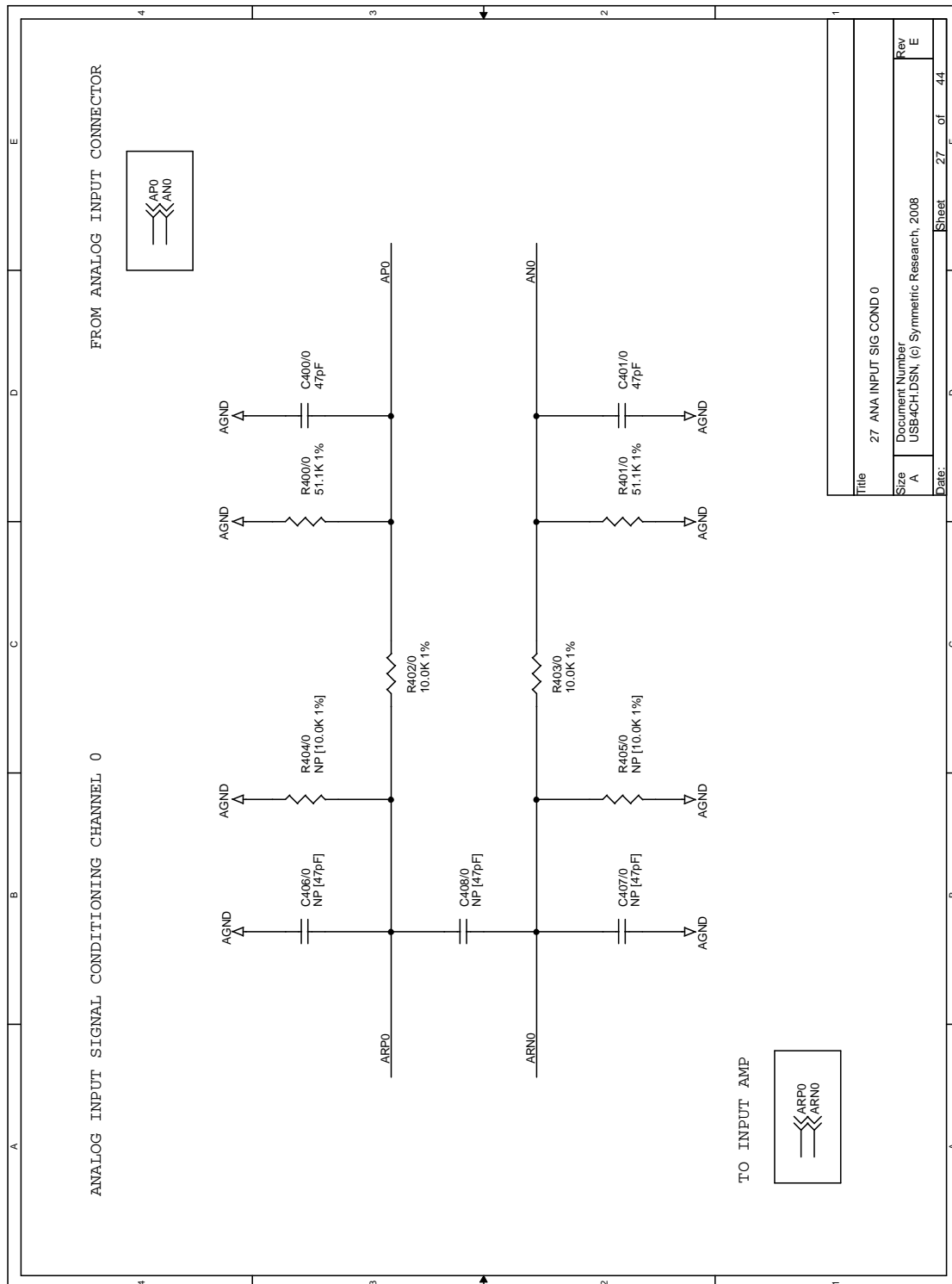


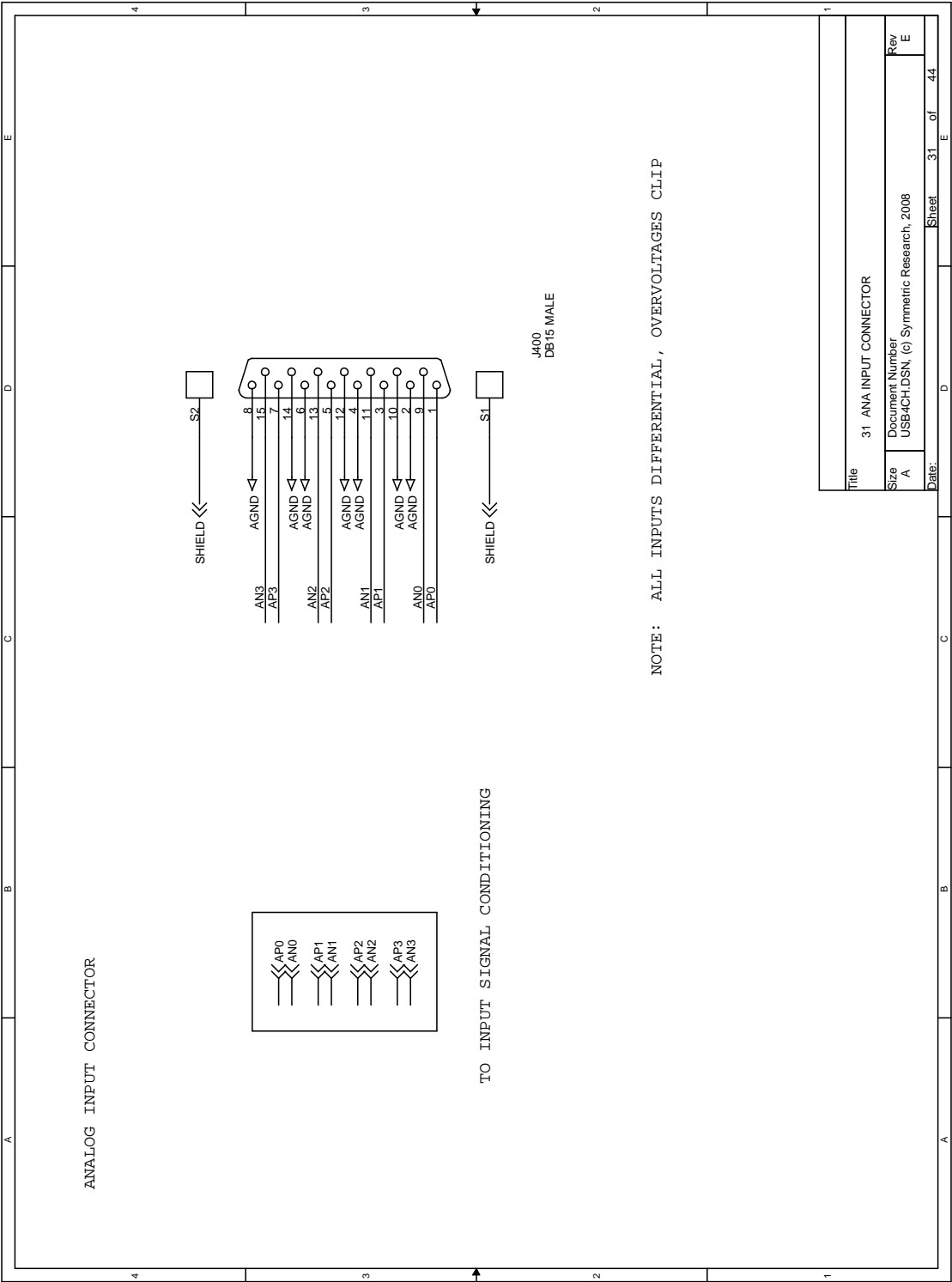


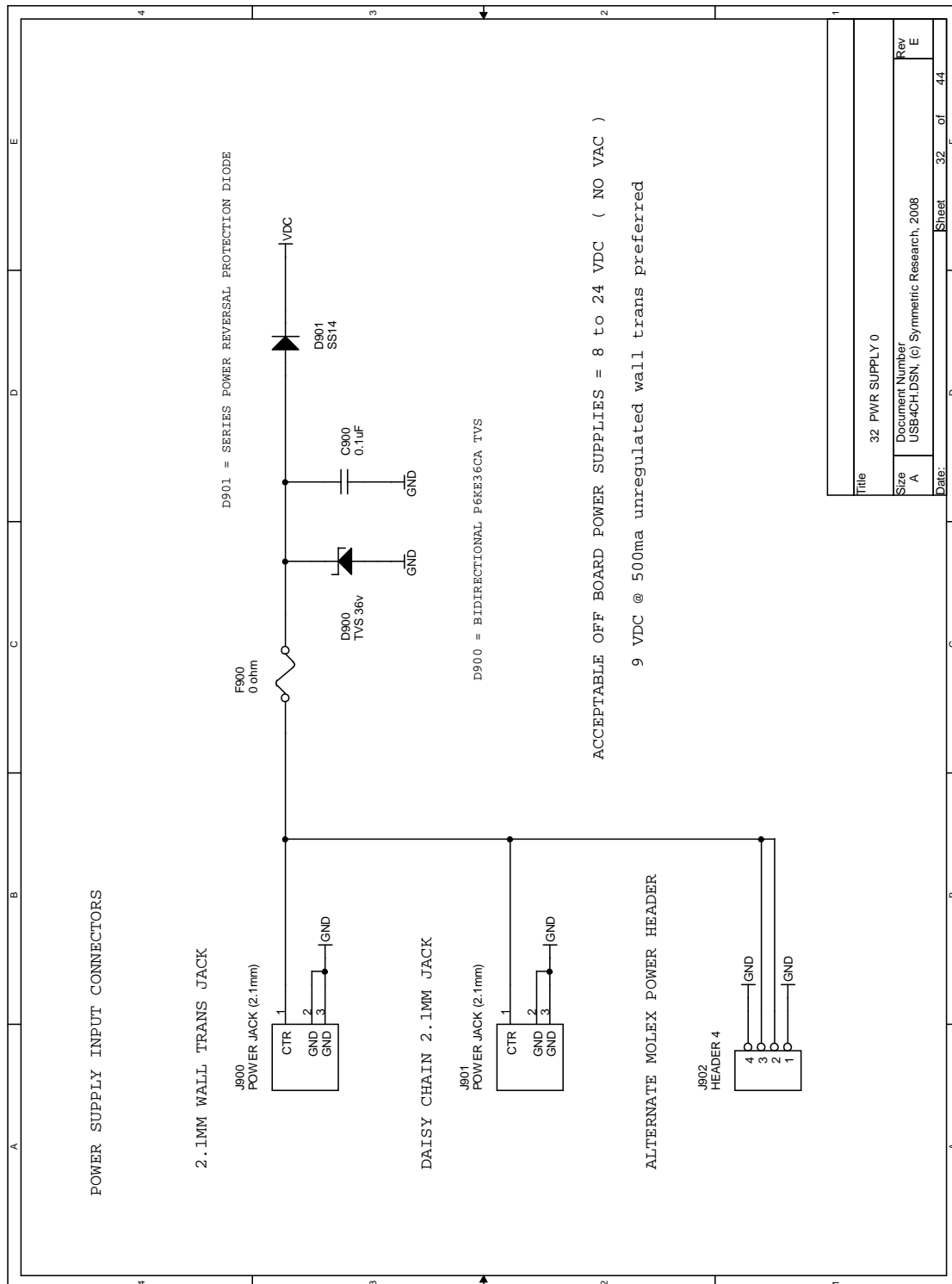












21.12 USB8CH schematics

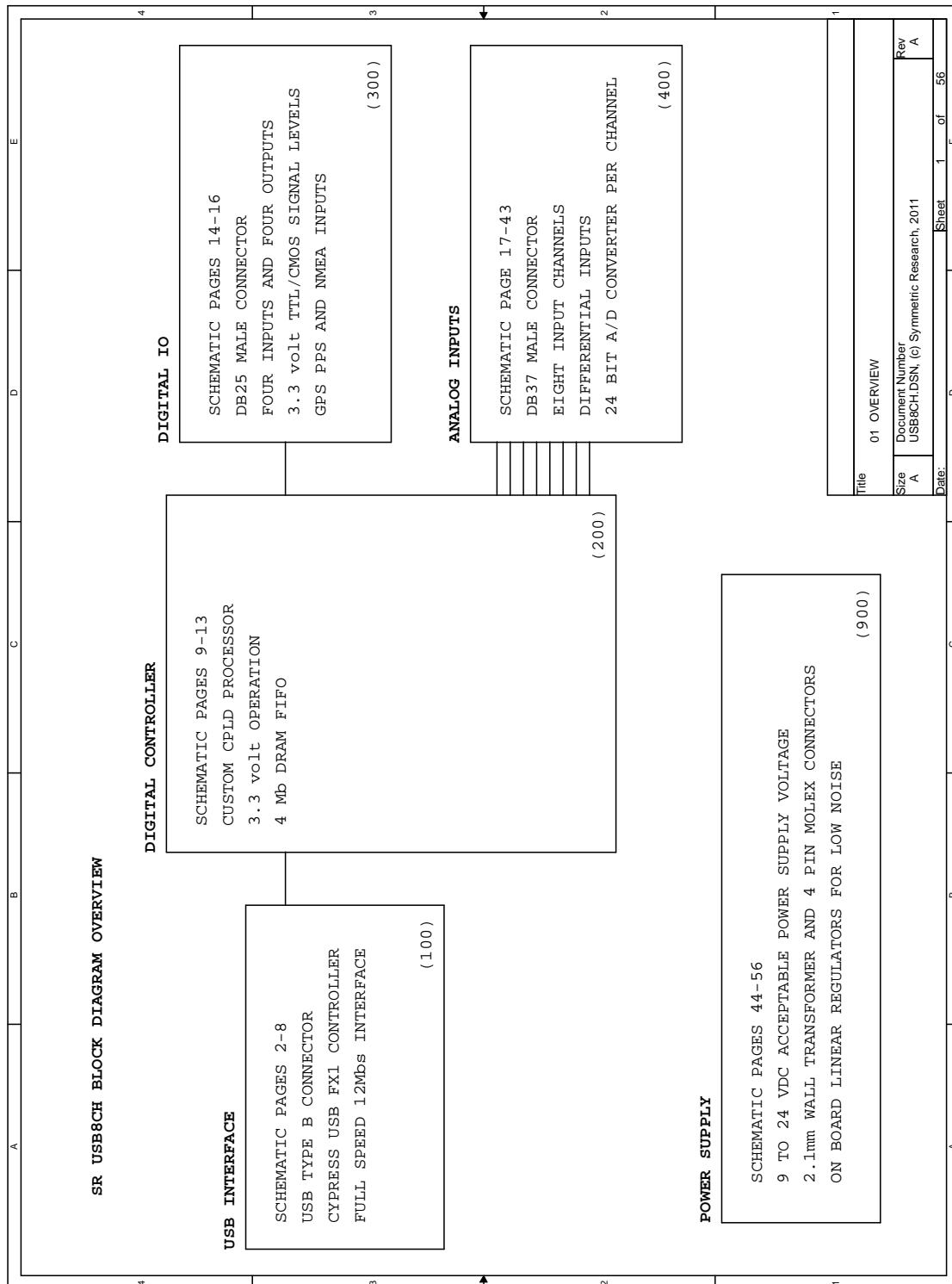
This section has the eight channel USB8CH circuit diagrams. Overall the USB8CH is similar to the USB4CH. The major difference being four additional channels to make eight total. Although they have different physical size and analog connectors, from the viewpoint of software and analog response they are interchangeable.

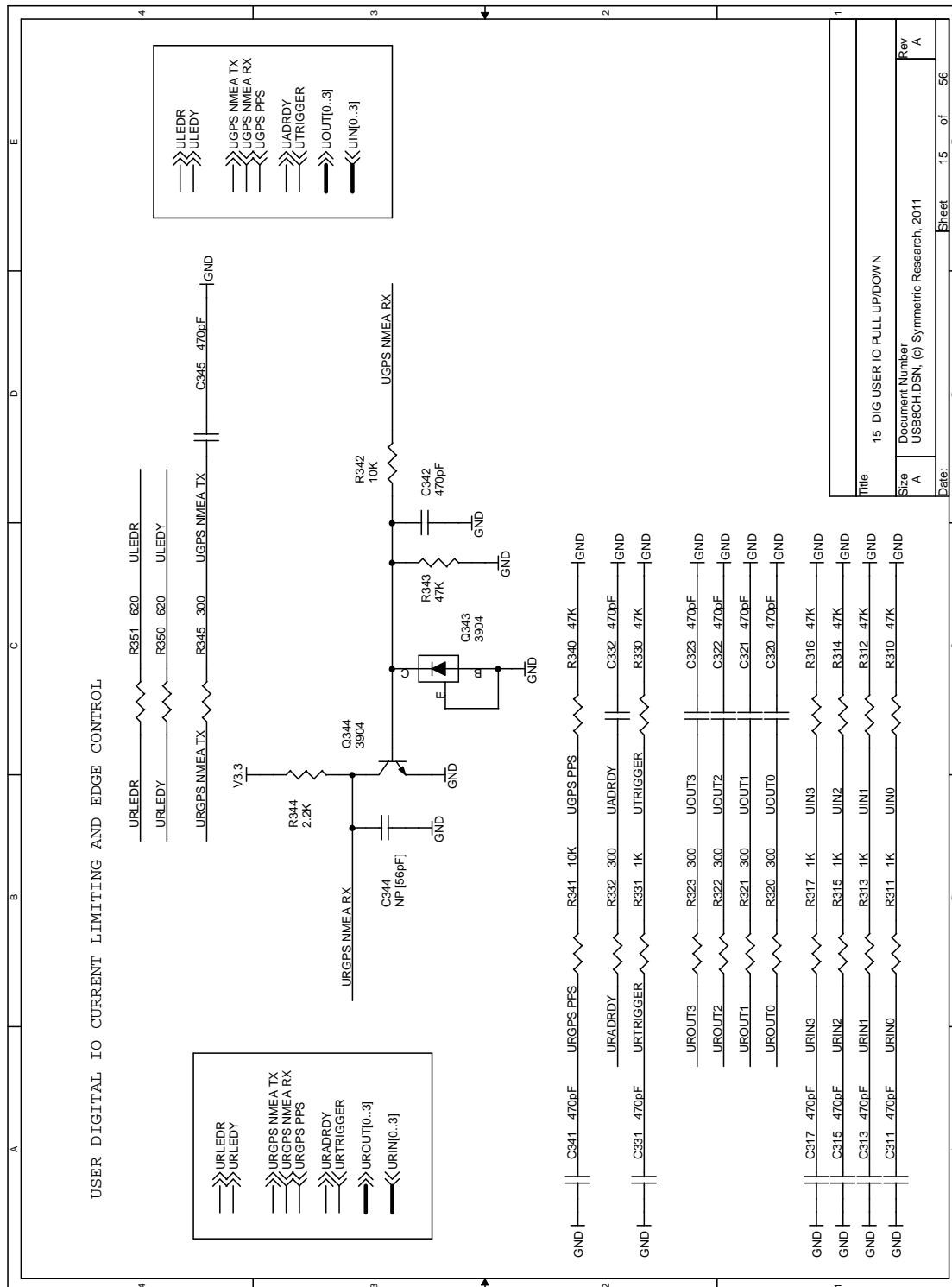
Repetitive parts of the circuit appear only once. For example, the analog front end is the same from one channel to the next and is given only once even though the physical circuit is repeated eight times on the board.

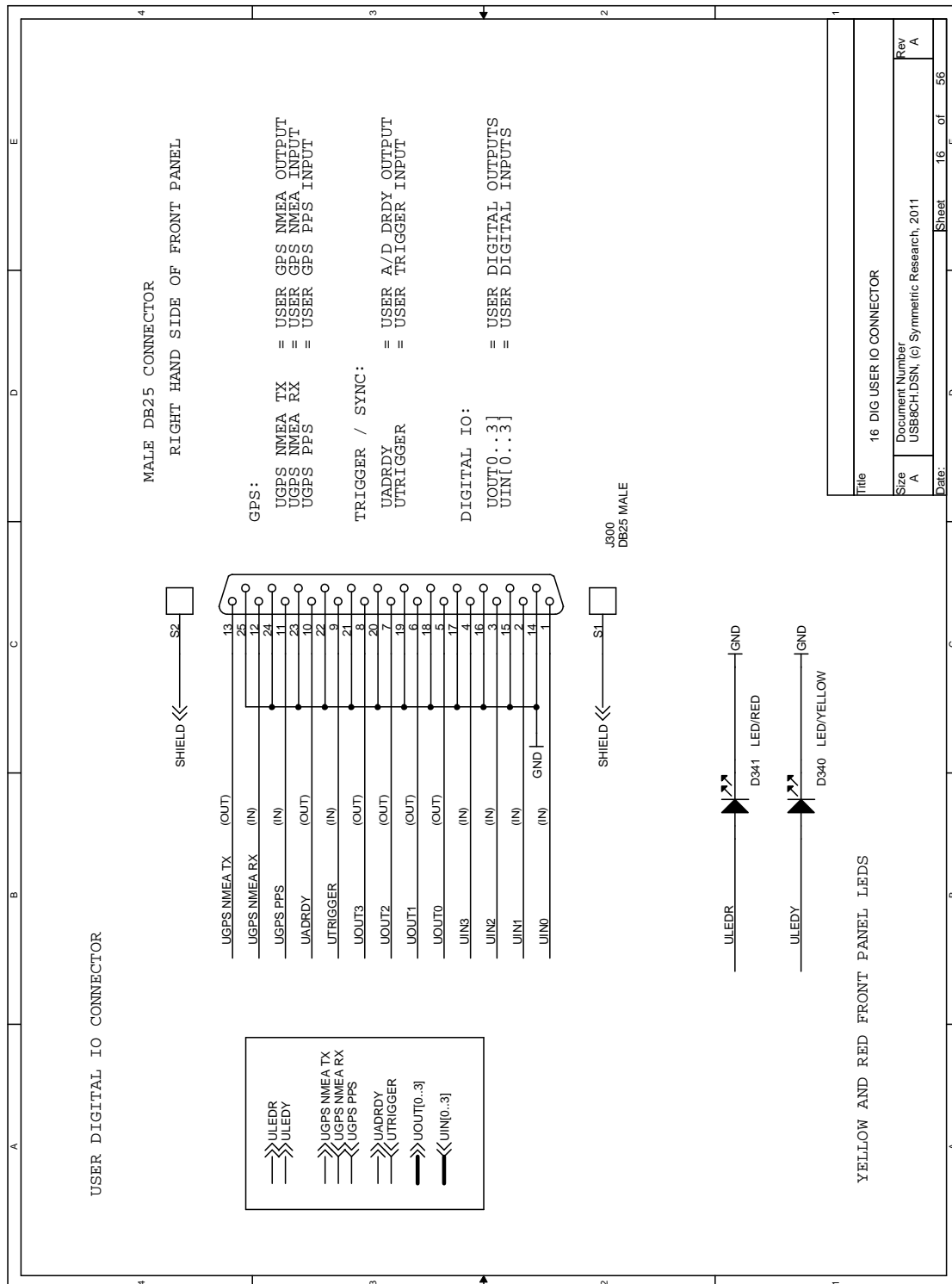
Use the Adobe Reader View/RotateView/Clockwise command for more comfortable horizontal viewing. See [Using Adobe Reader](#) for viewing tips.

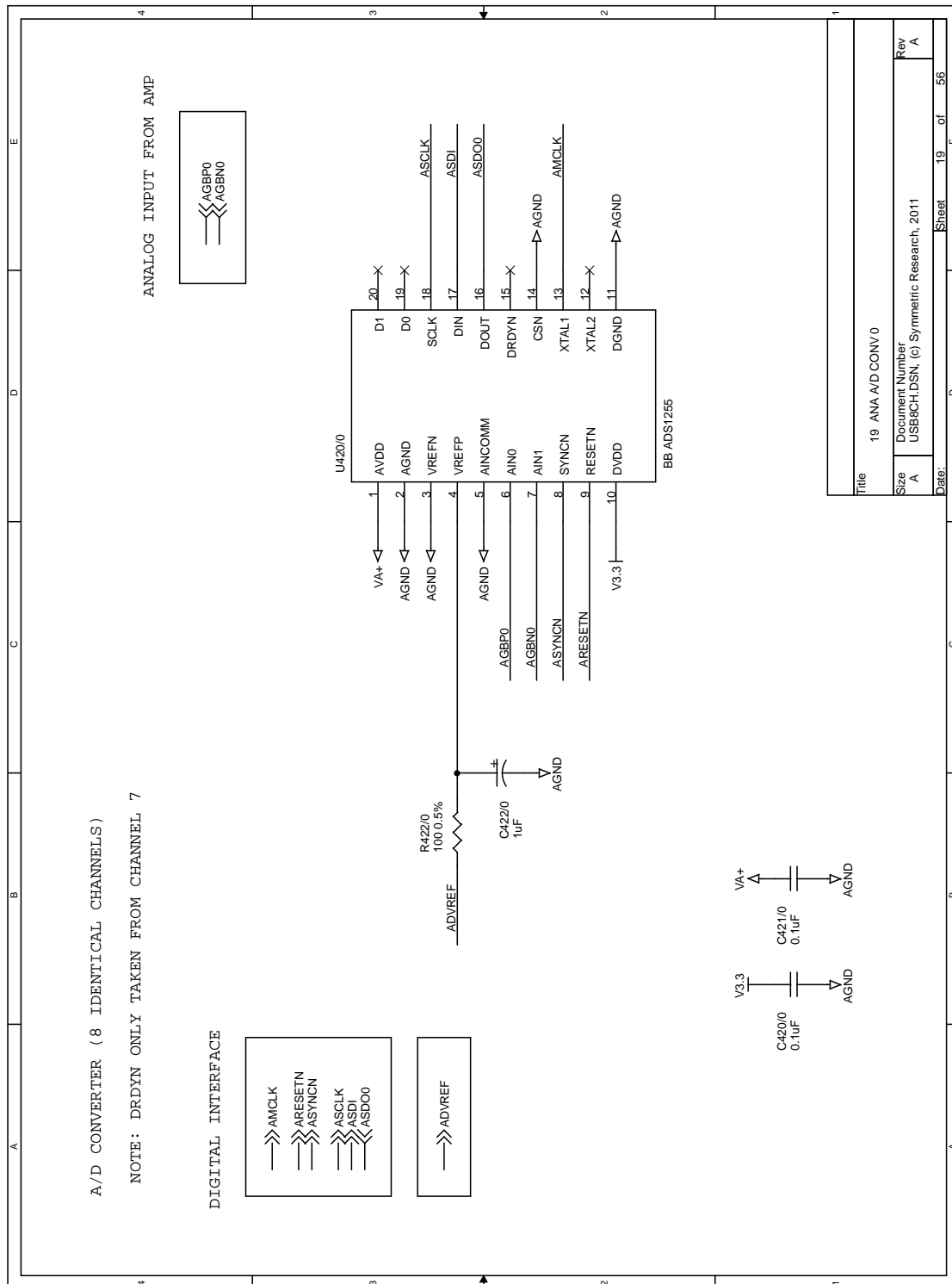
USB8CH circuit index:

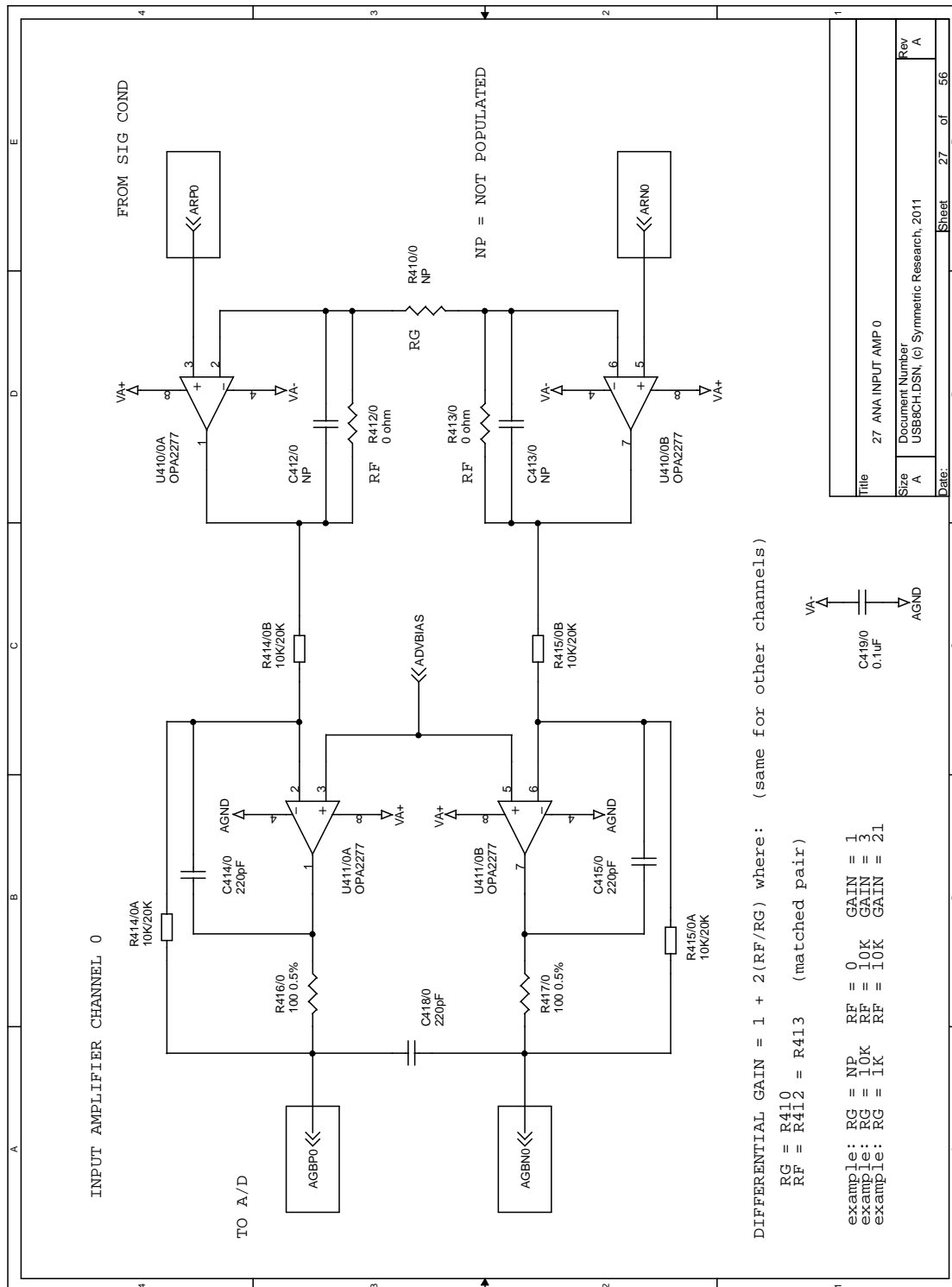
| | |
|---|-----|
| Circuit Overview | 126 |
| USB connector | 127 |
| User digital IO buffering | 128 |
| User digital IO connector | 130 |
| A/D converters | 131 |
| A/D input amplifiers | 132 |
| A/D signal conditioning | 133 |
| Analog input connector | 134 |
| Power supply connector | 135 |

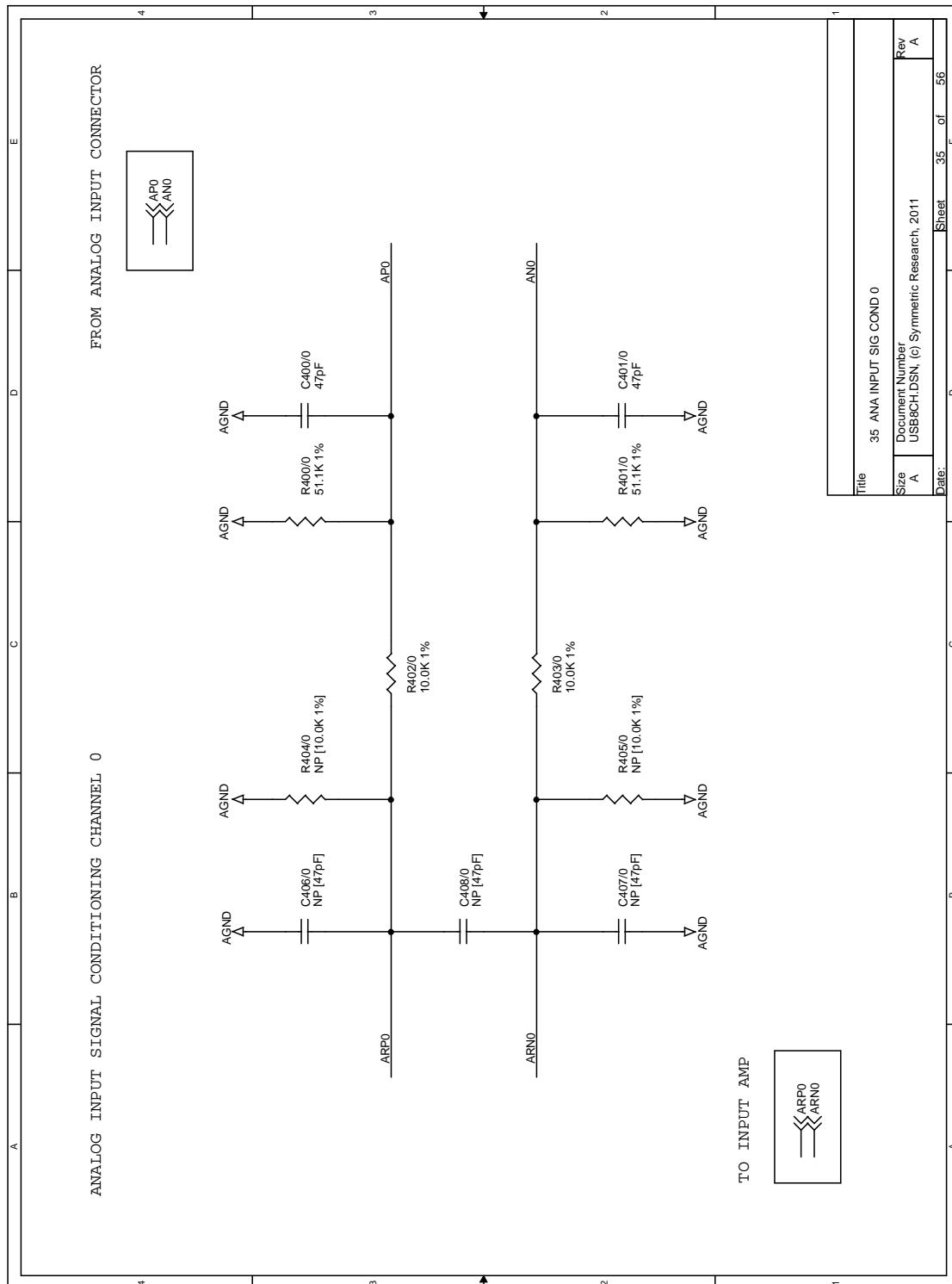


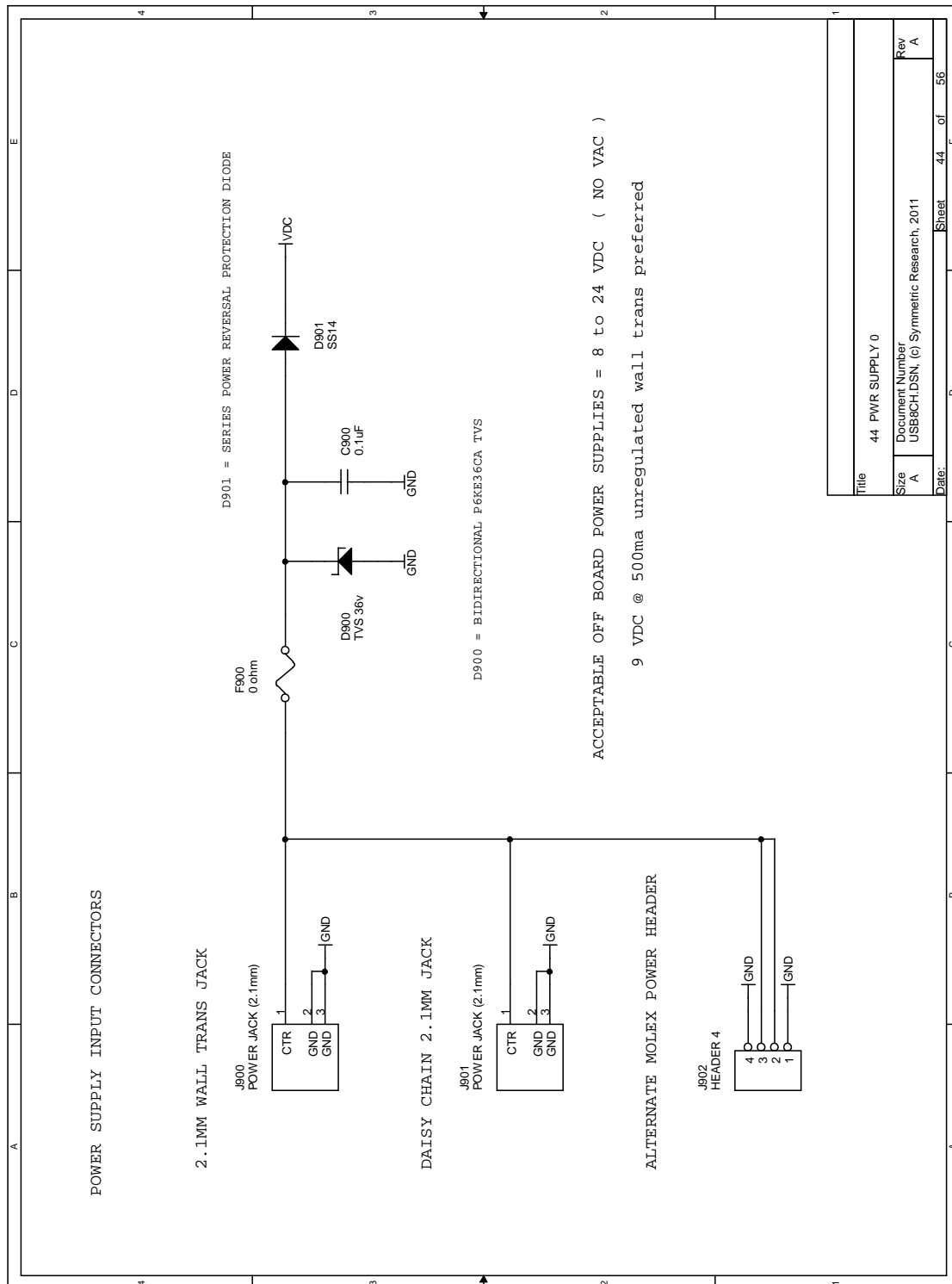












Chapter 22

Examples and Experiments

This chapter reviews a few simple hands on experiments to help you get started with the USBxCH. Related files and data are in the `/SR/USBXCH/Examples` directory with the software distribution.

All users, new and old, are encouraged in particular to study the [AA battery example](#) to become familiar with the USBxCH differential inputs.

- | | |
|---|---|
| ▷ AA battery | using a battery as an analog DC test signal |
| ▷ Absolute calibration | absolute voltage calibration with DVM |
| ▷ 10 turn potentiometer | calibrating DVM in potentiometer turns |
| ▷ Ratiometric technique | ratiometric methods versus absolute |
| ▷ Solar cell | measuring light levels with a solar cell |
| ▷ GnuPlot | plotting USBxCH data with GnuPlot |
| ▷ Twisted pair | twisted pair for 50/60Hz noise rejection |
| ▷ Acquiring 1Hz data | interpolate and GPS align Blast data |
| ▷ Multiple USBxCH | running several USBxCH from one PC |
| ▷ Powering with batteries | battery power for portable and field apps |

22.1 Measuring a AA battery

With a AA battery and the **DVM** program you can experiment with several of the USBxCH DC analog input characteristics. Besides showing basic usage, we will also review a few details regarding the three wire differential (+,-,AGND) inputs. For additional theory behind differential signals, see the **Analog inputs** chapter.

The most basic test with a AA battery is to simply connect it between the (+,-) of an analog channel, like this:

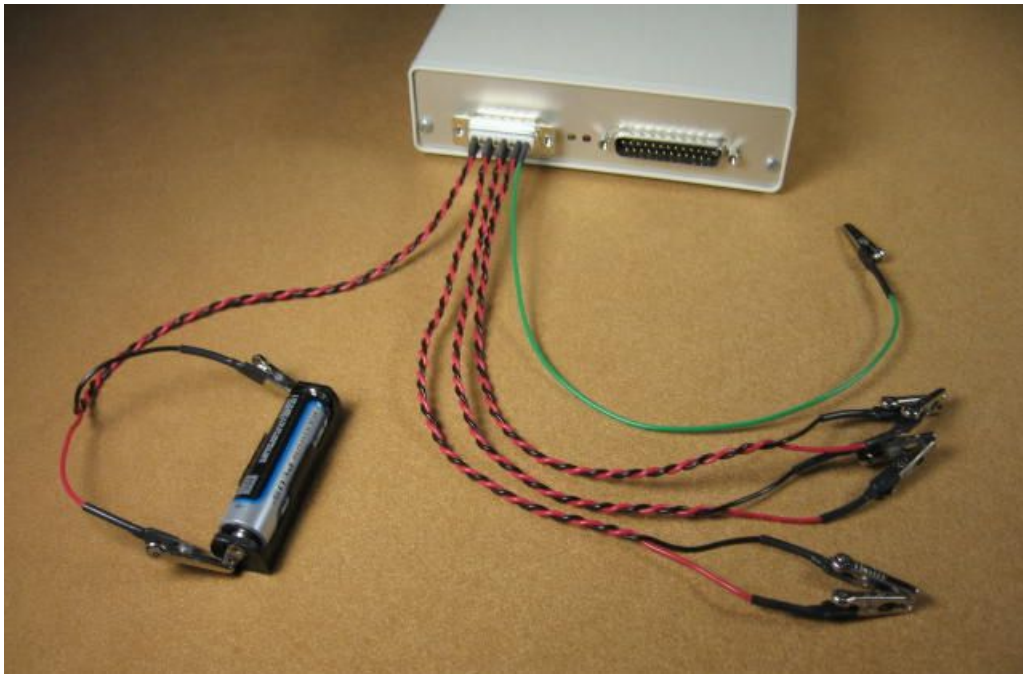


Figure 22.1: Two lead AA battery voltage measurement

In this setup, we have used a battery holder with tab lugs so the alligator clips can be easily connected, with the red and black wires of analog channel 0 connected to the (+,-) battery terminals. The USBxCH green AGND alligator clip is not connected at all, leaving the battery floating. Note the black - input *is not ground*. It is the inverted differential input.

Since a AA cell has approximately 1.5 volts, if you run the DVM program by double clicking on the "Run DvmGui Volts" shortcut you will see the following on your computer:

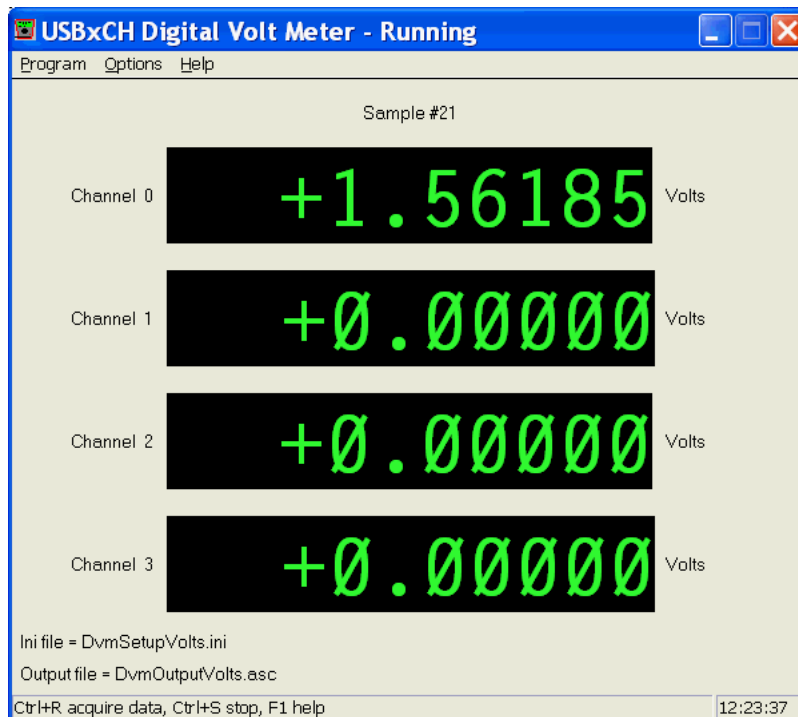


Figure 22.2: DVM with a AA battery hooked up

This is the result we expect, a fresh AA alkaline battery has about 1.5 volts. When making this measurement, if the differential leads are hooked up in reverse, DVM will simply read a negative voltage. Also experiment with hooking the battery up to the other channels and see the voltage appear on their respective DVM readouts. Unused (+,-) inputs should normally be left shorted together.

Performing this test, it is important to be aware that while the red and black alligators are much like the test leads of a handheld Fluke multimeter, *they also may not go more than plus or minus 4 volts with respect to AGND, or else clipping will occur.*

Given this constraint, how can we be sure the floating AA battery keeps the USBxCH (+,-) input pins within the 4 volt range of AGND? After all, there is no obvious AGND connection in Figure 22.1. The answer is a weak ground connection is maintained by the USBxCH internally as shown in the circuit fragment on the following page. In this fragment, half of the floating battery voltage is *above* AGND, while the other half is *below* AGND. If you physically measure with a handheld voltmeter you will find this is indeed true. Because of this weak ground connection, you often don't have to make an explicit AGND connection for floating voltages and sensors.

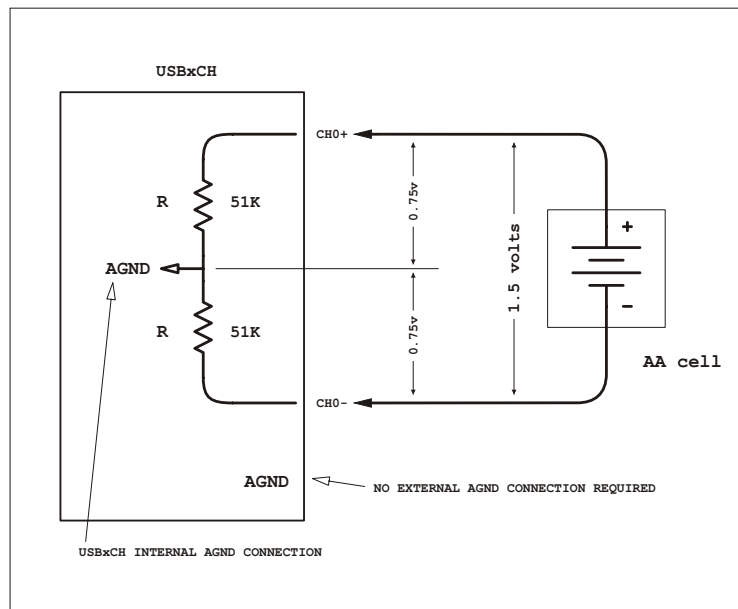


Figure 22.3: Floating AA battery circuit

What happens if instead of floating, we crowbar one terminal of the AA battery to AGND as in the following setup?

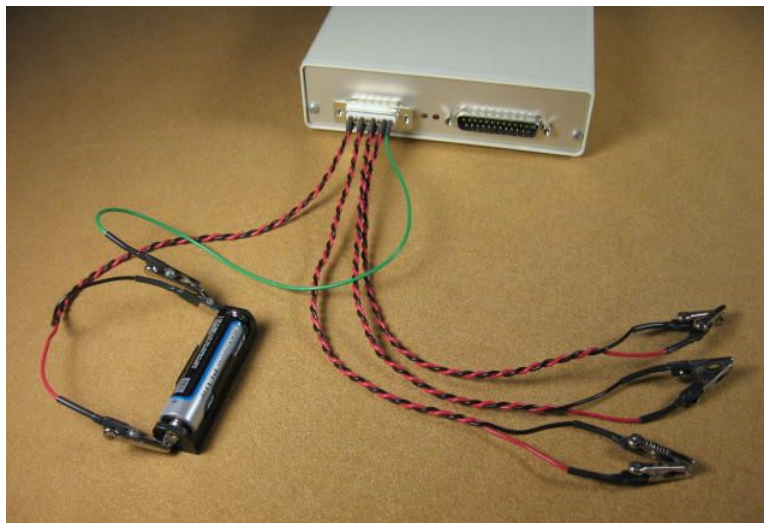


Figure 22.4: AA battery measurement with AGND connection

Here we have simply connected the green AGND alligator to the metal thumb tab of the black alligator. You will still measure 1.5 volts with DVM as before, but now the voltages

between the *inputs and AGND are different than the floating case*. Here the + input is at 1.5 volts with respect to AGND, while the - input is at AGND. The difference still remains at 1.5 volts, but the + input is pushed 0.75 volts closer to the 4 volt clipping limit. We have effectively put a short across the lower internal 51K ohm resistor as in the following circuit fragment:

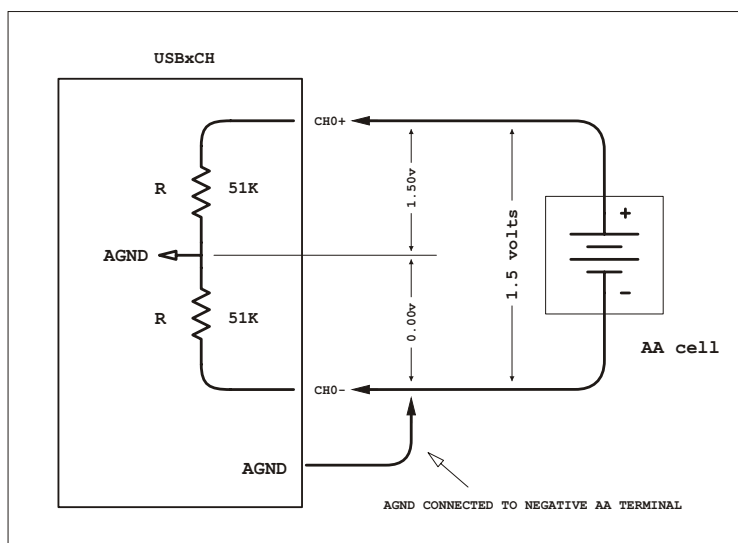


Figure 22.5: Grounded AA battery circuit

To modify the experiment further, imagine if instead of a 1.5 volt AA cell we had used a 6 volt battery. When floating, DVM would report the correct 6 volts, with 3 volts above AGND and 3 volts below, avoiding clipping. But if we connect one of the (+, -) terminals to AGND, the other terminal will be forced to 6 volts with respect to AGND and clipping will occur. You can construct a 6 volt battery from 4 AA cells in series. The figure on the next page shows some of the results.

It is also acceptable to use adjustable floating lab supplies less than 20 volts for such experiments. Even transistor radio 9 volt batteries can be used, although their voltage is high enough so clipping will occur even in the floating case. The USBxCH inputs will not be damaged or harmed with the 9 volts of overvoltage.

!!! ■ Never attempt to measure high voltages such as 110/220 vac from the wall socket with the USBxCH. Not only are such voltages life threatening with the types of connectors on the system, you will also instantly destroy the front end circuitry and void the warranty. If you must measure high voltages, use resistive dividers, TVS/Zener overvoltage protection, and suitably insulated connectors in front of the USBxCH.

Note the USBxCH AGND is connected to its power supply ground as well as the PC ground when making measurements. Because of this, a ground connection is often present in ways you might not expect. For example ... if you are using a desktop machine, AGND is connected through the USB cable ground wire to the PC ground, which in turn is probably connected to the PC chassis and the third prong on the wall power. At the same time, if you use a signal generator also powered from wall power, it is probably connected to wall ground. This completes a ground loop you probably didn't expect, and provides opportunities for noise contributions. Review the entire system carefully to avoid hidden ground connections you didn't count on.

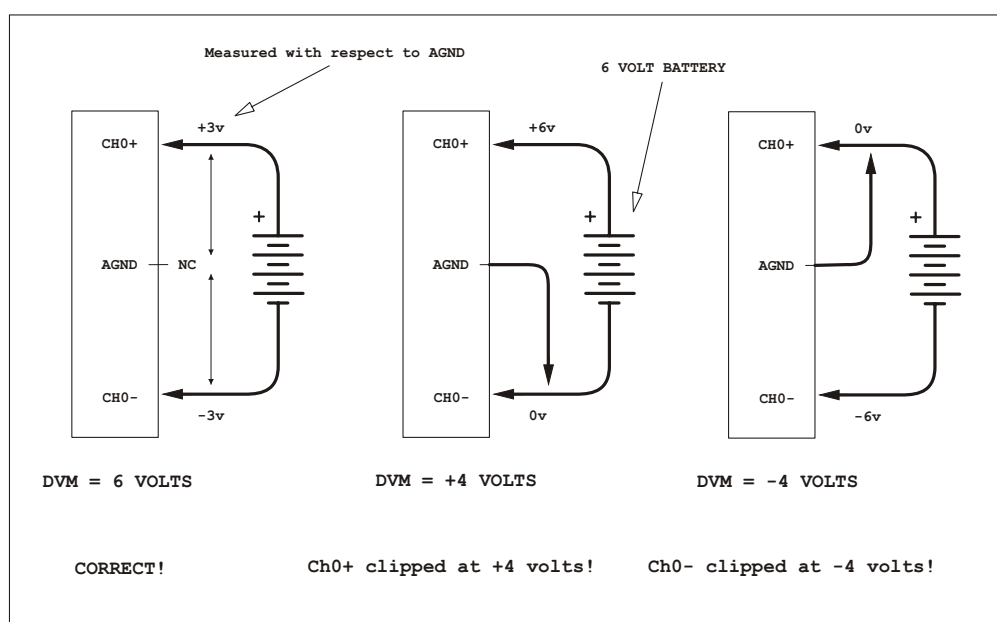


Figure 22.6: 6 volt battery circuit showing clipping depends on ground

Finally, in this example all the DVM channels have been displayed as volts. You can also calibrate DVM to display other physical units as appropriate for the application. See the [10 turn potentiometer](#) example. For easy ways to plot results, see the [GnuPlot](#) example.

22.2 Absolute calibration

Work underway . . .

Show how to perform absolute calibration with DVM and the VREF-399. All calibration coefficients should be specified in the ini file.

22.3 Using DVM with a 10 turn potentiometer

Besides volts, **DVM** can also display physical sensor units. This example shows how to set up a 10 turn potentiometer with DVM displaying "**turns**". Using the same techniques as this example you can easily calibrate to display other desired physical units. The circuit for the potentiometer experiment is:

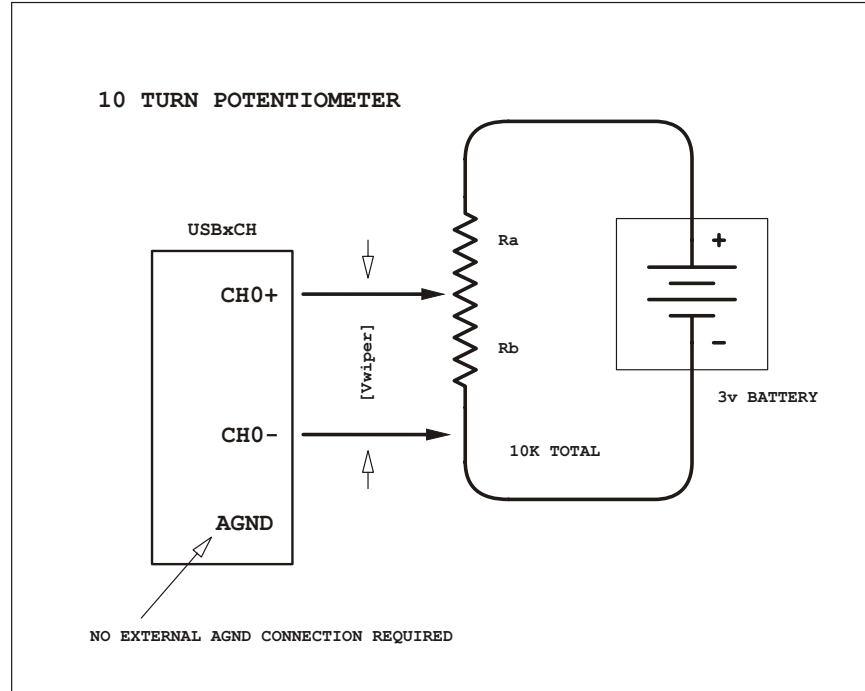


Figure 22.7: 10 turn potentiometer circuit fragment

We will assume the battery is a pair of AA cells in series providing 3 volts. It is a good idea to keep the end to end potentiometer resistance reasonably high so not too much current is drawn from the battery. With a 10K pot there would be 0.3 ma of current flowing, which is reasonable.

If R is the total end to end potentiometer resistance, R_a the resistance above the wiper, and R_b the resistance below the wiper, then we have:

$$\begin{aligned} R_a &= R \cdot (1 - T/10) \\ R_b &= R \cdot (T/10) \end{aligned}$$

where T is the physical number turns at the current setting of the 10 turn potentiometer. Of course, this assumes the wiper resistance varies linearly with turns. The linearity should

be available on the potentiometer spec sheet, or perhaps even stamped on the side of the unit. You could also check the linearity with an experiment like this. Using Ohm's law, $V = IR$, and the equation for R_b , the *voltage* across the wiper would be:

$$\begin{aligned} V_{wiper} &= (V_{bat}/R) \cdot R_b \\ &= (V_{bat}/R) \cdot (R \cdot (T/10)) \\ &= (V_{bat}/10) \cdot T \end{aligned}$$

showing V_{wiper} is directly proportional to the the number of turns T . To calibrate DVM for turns, we will use the companion program **Calibrate** to determine the slope and offset parameters. The physical setup looks like this:

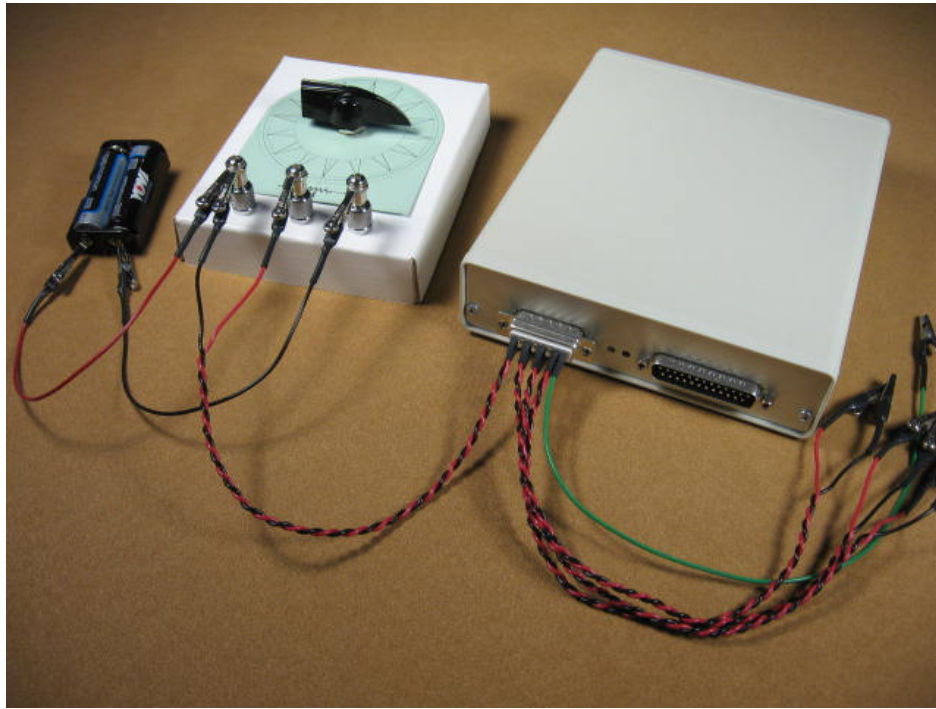


Figure 22.8: 10 turn potentiometer measurement

The potentiometer has been mounted on a simple cardboard enclosure with a green dial made from index paper. Connections to the stimulus battery, potentiometer, and USBxCH have been made with alligator clips. The USB cable and computer are not shown. Being a 10 turn potentiometer, the black knob can spin around and around and it is easy to lose track of which turn it is on, let alone measure the exact angular position.

The bottom side of the potentiometer is in the next photo, with simple wires to the three potentiometer terminals. You could easily do this experiment with the potentiometer just

lying on the table, but the cardboard enclosure and dial are simple to make for more accurate measurements.

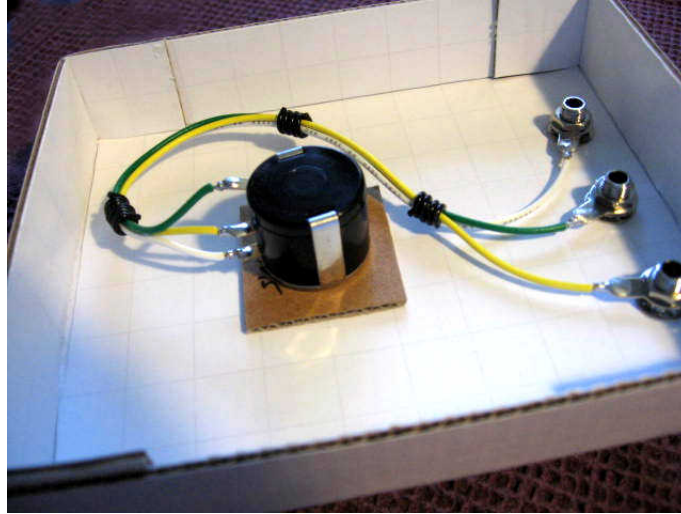


Figure 22.9: 10 turn potentiometer photo bottom

Next, start up the DVM **Calibrate** program. To calibrate, first turn the potentiometer knob to its minimum setting, aligning the knob with the zero mark on the dial, then enter 0 for Value A and click on the Measure A button. After that, move the potentiometer thru 10 turns to its maximum setting, aligning with the zero mark again, then enter 10 for Value B and then click on the Measure B button. These two measurements define a straight line on a plot of **turns** versus A/D counts. Clicking on the Save button will automatically save the ini file "**Cal.ini**" with the corresponding slope and offset.

Loading the new "**Cal.ini**" into DVM will display the potentiometer channel in "**turns**". Note when running Calibrate you can enter the Channel Title and Units as **Potentiometer** and **Turns**, so they will also display. The Calibrate screen should look like Figure 22.10, and the resulting DVM screen like Figure 22.11.

With practice, calibration for other types of sensors should be quick. The steps are always the same. Put the sensor at its minimum setting, and take a measurement. Then put the sensor at its maximum setting, and take another measurement. Use the resulting ini file with DVM. There is nothing special about a 10 turn potentiometer, the same process can be used with any sensor that has a linear response.

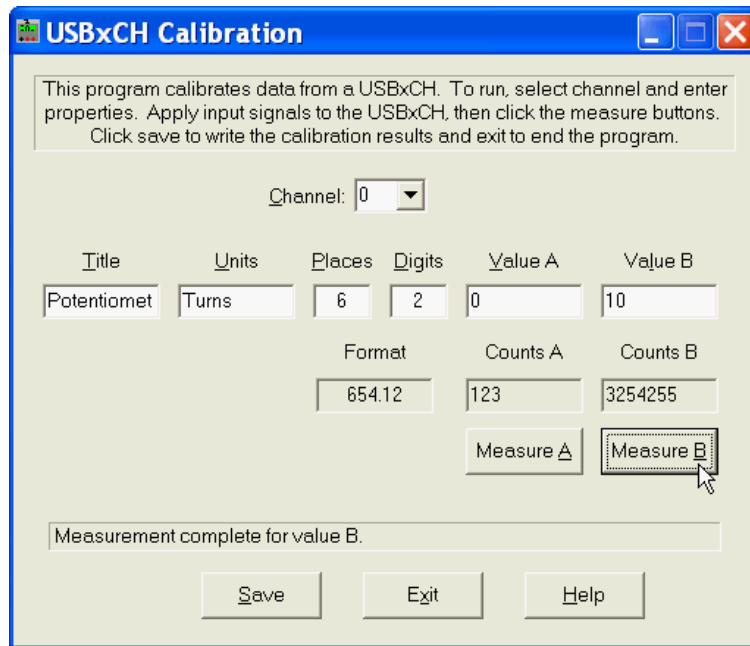


Figure 22.10: 10 turn potentiometer Calibrate screen

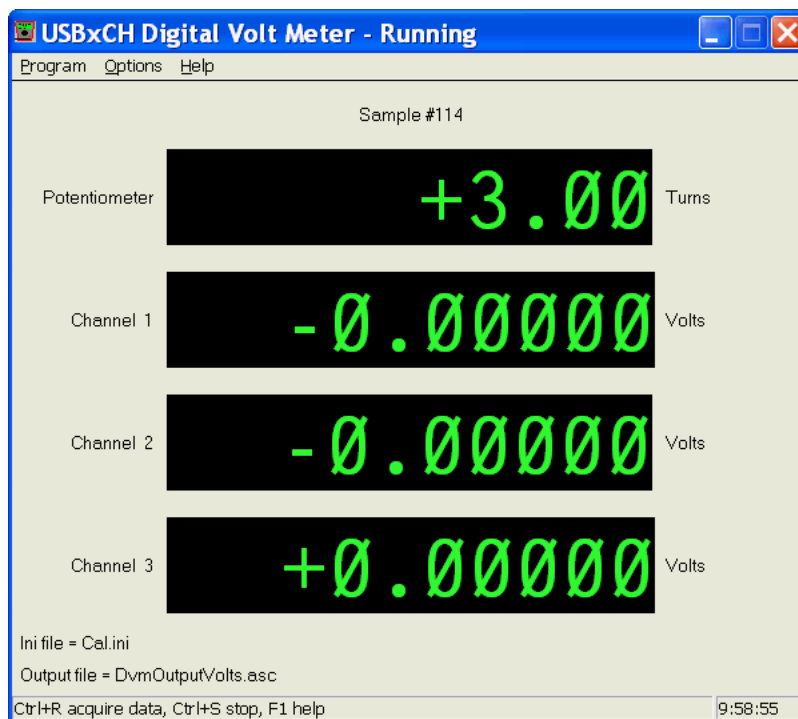


Figure 22.11: 10 turn potentiometer DVM screen

No doubt 24 bits is more resolution than is typically needed for the average potentiometer measurement. However, the idea is to apply these methods to applications that do require high precision. Even with this experiment, an 8 bit A/D would only have an angular resolution of 14 degrees, while with 24 bits you can easily achieve 0.1 degree resolution. Other types of potentiometers offer linear styles for length measurements, while many simple devices can serve as remarkably good sensors for other parameters. Silicon diodes as temperature sensors, and LEDs as light sensors are examples. There is also a huge variety of prepackaged sensors for measuring parameters of every type. See your favorite electronics catalog as in the [Extra supplies](#) chapter for possibilities. Regardless, you can use the Calibrate program to create a DVM readout in physical sensor units.

Some will observe there is a flaw with the core equation we have used in this example. It comes from the equation:

$$V_{wiper} = (V_{bat}/10) \cdot T$$

The wiper voltage is not only a function of the physical number of turns T , *but also depends on V_{bat}* . Any change in the battery voltage will result in a change in the number of turns reported on the DVM screen, regardless of whether the physical turns on the potentiometer changed or not. You won't have to work with the system very long before you notice changes in the battery voltage because of discharging or changes in temperature have an effect. At 24 bits even a small drift becomes apparent.

What can be done? One solution would be to use a battery or reference that has an extremely constant output across time, load, and temperature. The SR VREF-399 is an example of a stable potentiometer excitation source that would be suitable. However for sensors like potentiometers with outputs proportional to their stimulating voltage, there is an even better way: [ratiometric techniques](#). Ratiometric methods are simple and give excellent results.

To change this example into a ratiometric measurement the battery excitation voltage V_{bat} is simply measured on another channel at the same time as the wiper voltage. From the V_{wiper} equation above you can see dividing by the battery voltage will give the actual number of physical turns regardless of the absolute values of the voltages. Ratiometric measurements are easy to set up with a multichannel DVM. The next section shows the results that are possible.

22.4 Ratiometric technique

Work underway . . .

Set up an adjacent channel to scale with the excitation in a ratiometric measurement.
Review the equations to show how battery excitation droop, and temp drifts are canceled.

22.5 Measuring light levels with a solar cell

Sometimes sensors can be made from unexpected devices. Any component with a voltage that varies in response to an external stimulus is a candidate.

Solar cells find their most common use as a power source. However, the voltage from a cell varies with the light level, serving as a light intensity sensor. With DVM, measurements from a cell can even be calibrated directly into light intensity units. As an example, the photo below shows a small flexible cell mounted in a cardboard holder for testing. This cell has a voltage ranging from 0 volts in total darkness to approximately 3 volts in full brightness. The cell is connected directly to USBxCH analog channel 0 with no signal conditioning required.

Using the **Calibrate** program you can set up a DVM readout to be 0 at total darkness and 10 at total brightness. You can also have DVM save its values to an output file along with time stamps. Acquiring data for days or months creates a record of light intensity.

The next section shows how to use GnuPlot with values saved in `DvmOutputSolar.asc` to plot results for reports and presentations.

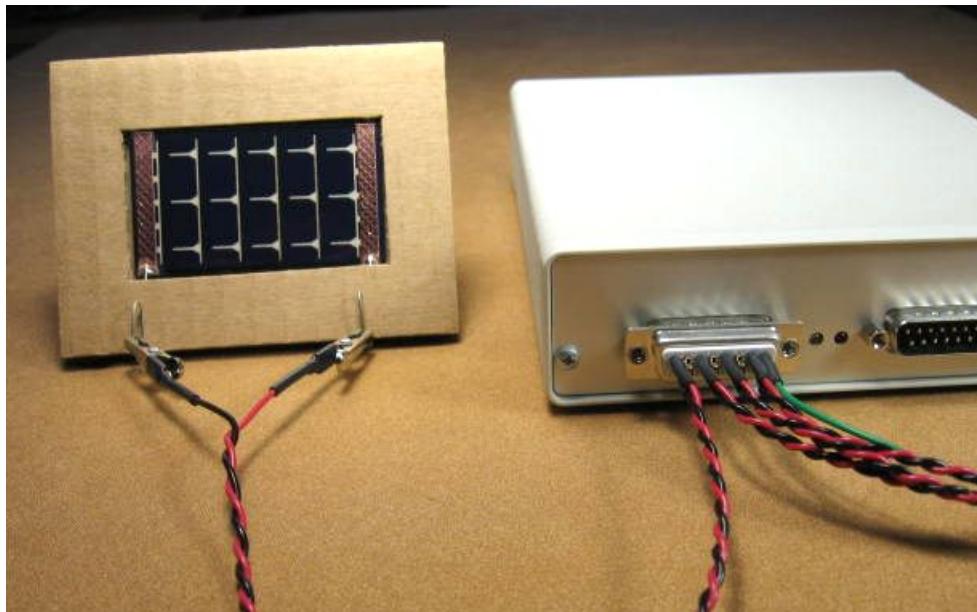


Figure 22.12: Measuring light levels with a solar cell

22.6 Plotting results with GnuPlot

After acquiring data with the USBxCH you may want to plot the values. This example shows how to use GnuPlot to generate display and report presentations. GnuPlot is a public domain program and can be downloaded for free from:

www.sourceforge.net

or found by searching the web. Versions are available for both Windows and Linux, with documentation included when unpacked. Excel and MatLab users may use techniques similar to the ones shown here to import data.

To use GnuPlot you first need a data file. For this example we assume you have the solar cell data collected with DVM as in the previous example. The following file fragment from `DvmOutputSolar.asc` would be typical:

| Sample | Volts | OneToTen | Time (Sec) | Time (YMD HMS) |
|--------|-------|----------|----------------|-------------------------|
| 1 | +0.7 | +1.42 | 1202171952.468 | 2008/02/05 00:39:12.468 |
| 2 | +0.7 | +1.43 | 1202171953.629 | 2008/02/05 00:39:13.629 |
| 3 | +0.7 | +1.43 | 1202171954.791 | 2008/02/05 00:39:14.791 |
| 4 | +0.7 | +1.45 | 1202171955.953 | 2008/02/05 00:39:15.953 |
| 5 | +0.7 | +1.49 | 1202171957.114 | 2008/02/05 00:39:17.114 |
| 6 | +1.0 | +2.01 | 1202171958.276 | 2008/02/05 00:39:18.276 |

The DVM ini parameters to create this file format were:

```
; Solar cell output format and channel 1 calibration:

OutputFileName      = "DvmOutputSolar.asc"
OutputFileComment   = "Measuring solar panel response"
OutputFileShowHeader = ON
OutputFileShowIndex = ON
OutputFileShowTimeSec = ON
OutputFileShowTimeYmd = ON
RunMode             = CONTINUOUS

DisplayTitle 1 = "Light Level"
DisplayUnits 1 = "OneToTen"
DisplayPlaces 1 = 6
DisplayDigits 1 = 2
DisplaySlope 1 = 6.02121e-007
DisplayOffset 1 = -0.00147941
```

Besides specifying the output file format, these ini parameters also calibrate the second readout to light level units of 1 to 10. Calibrate into other light intensity units as appropriate.

Once the data file has been acquired, start GnuPlot and issue the commands:

```
# SET FILE FORMAT and PLOTTING AXIS PARAMETERS:

DataFile = "DvmOutputSolar.asc"
ColSample = 1
ColVolts = 2
ColLight = 3
ColSeconds = 4
ColTime = 5

set xtics nomirror font "Helvetica Bold,14"
set ytics nomirror font "Helvetica Bold,14"
set xlabel "Sample" font "Helvetica Bold,18"
set ylabel "Volts" font "Helvetica Bold,18"
set xrange [0:120] # About 2 minutes of data
set yrange [0:5] # Solar panel outputs ~3.3v in strong sunlight

# PLOT THE SOLAR DATA:

set title "SER1CH-UA Solar Panel Data" font "Helvetica Bold,18"
plot DataFile index 1 using ColSample:ColVolts linestyle 1 title "Solar panel voltage"
```

You can also save these commands to a file and run them with the GnuPlot load command, see `Examples/Solar.gp`. The plot command does the serious work, everything else is setup. Refer to the GnuPlot PDF documentation for details. After running, the following should appear on the screen:

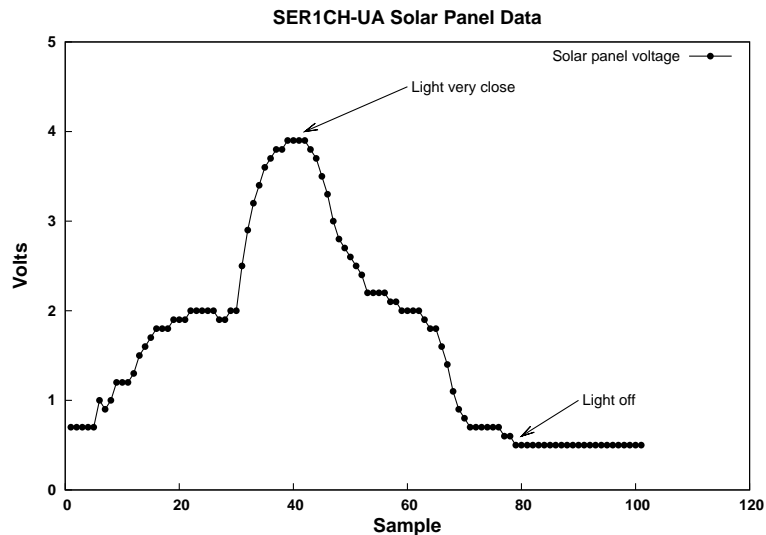


Figure 22.13: GnuPlot graph

For including the plot in a report, you'll probably want to save it as a PDF file. Other formats are possible, read about them in the GnuPlot documentation. For PDF, first run GnuPlot to output a .ps PostScript file, and then convert this file to PDF. The conversion to PDF can be done with either Adobe Acrobat Distiller or the public domain GhostScript. In the examples directory we show how to do it with GhostScript commands.

22.7 Twisted pair for 50/60Hz rejection

Work underway ...

Demo making a set of twisted pair. Measure the improvement in 50/60Hz rejection.

22.8 Acquiring 1Hz data with Blast

Work underway . . .

Demo showing how to interpolate 19Hz Blast data to 1Hz. Discuss batch files and also give GnuPlots.

22.9 Multiple USBxCH with one PC

Work underway ...

Show how to run with multiple USBxCH. Use the work done for Matt Watson to demo how to set up the batch processing file and utilities.

22.10 Powering with batteries

For portable applications the USBxCH can be powered from batteries. The system requires 100 ma of current at 8 volts or greater, and it is entirely possible to run from AA and larger batteries for periods of days. Rechargeable batteries are particularly attractive. Two popular technologies are: NiMH (nickel metal hydride), and lead acid. This example shows how to use eight rechargeable NiMH AA cells to power the system, and also a small 3Ah (amp hour) rechargeable lead acid battery.

NiMH rechargeable batteries are readily available and used widely in digital cameras and other electronic devices. The AA size has excellent power density with 2Ah or more typical. Here we will be using Duracell AA batteries, which claim 2.6Ah. At a drain rate of 100 ma, in principle they should be able to power the USBxCH for 26 hours. In actual tests they reliably power the system for 22 hours, enough for a day of field work. The following photo shows the setup:

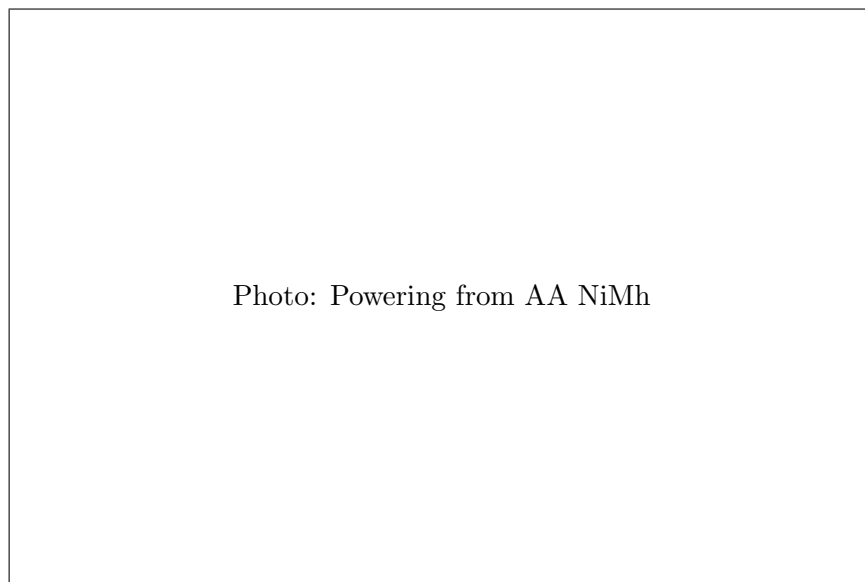


Figure 22.14: Powering from a rechargeable NiMH AA battery pack

To hold the 8 AA cells, an inexpensive battery holder organized as 4x2 cells in series has been used. Since NiMH cells have a fully charged cell voltage of at least 1.2 volts, this gives approximately 9.6 volts total. This is enough to power the USBxCH. In addition to the battery holder, there is also an on-off toggle switch. We recommend using a switch so the power can be applied at one clean instant to the USBxCH. Using alligator clips as a power switch is unreliable, sometimes drawing sparks and causing the system to crash on start up. Besides the battery holder and switch, there is also a 2.1mm power connector so

the battery pack can be quickly connected and disconnected for recharging.

Determining the discharge characteristics for any particular battery setup is easy: have the USBxCH measure and record its own battery voltage, giving a detailed plot of the discharge curve! One problem is the battery voltage will be greater than the +/- 4 volt USBxCH input range. To solve this, use a simple resistive divider to scale the battery voltage into range. Keep the divider resistances reasonably high so the divider doesn't drain much additional current from the battery. The following circuit is one example:

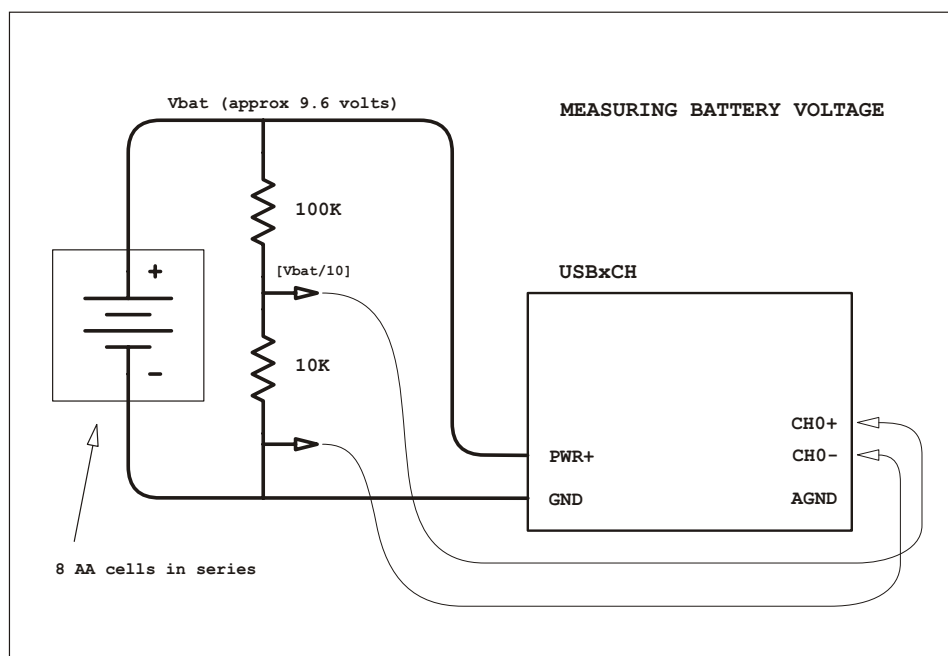


Figure 22.15: Dividing battery voltages for USBxCH measurement

To acquire the data, use the **Blast** acquisition program at a low sampling rate. There really isn't much point to sampling at high rates. The following would be typical:

```
cmd:prompt> Blast s1 g1 nokeypress
```

The **s1** option specifies sampling at 19.5Hz, while the **g1** option uses the PC clock to time stamp the data. After the run use the **Pak2Asc** utility to convert the binary PAK files to ASCII columns. As an alternative to Blast, you could also use the **DVM** program to gather data and even have a real time display of the battery voltage during the run.

While measuring the battery voltage with Blast, check the red LED on the USBxCH back panel near the 2.1mm power connectors. When the battery voltage drops low enough, the red LED will light up indicating it is too low for the on board regulators to function

correctly. At this point the run is over and it is time to stop Blast. The condition of the red power good LED is also given as a column in the Pak2Asc output file. The following **GnuPlot** graph made from the Pak2Asc data shows a typical run:

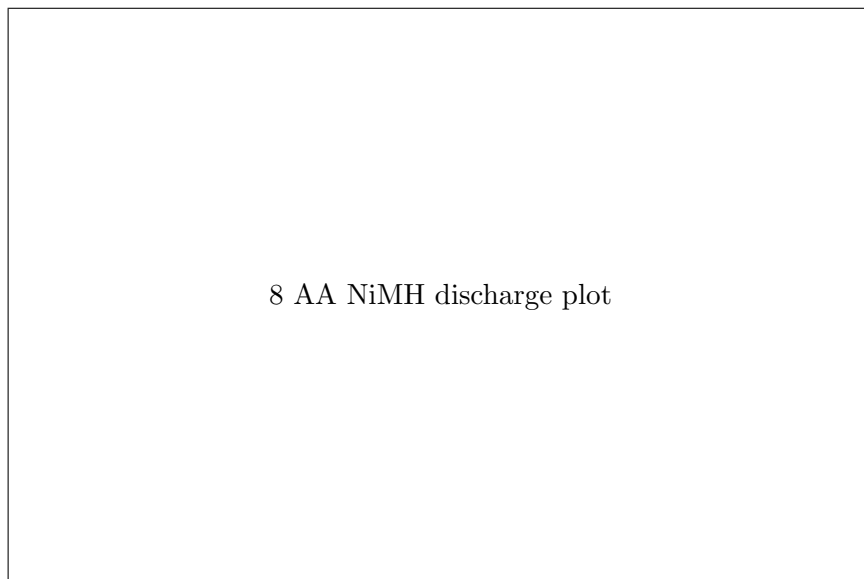


Figure 22.16: 8 AA NiMH discharge plot

Both the analog battery voltage and the power good status bit (red LED) are plotted. The power good signal (when the red LED comes on) failed at approximately 7.5 volts as it should. Note that although the USB digital portion of the system can function at battery voltages even as low as 3.5 volts, the analog subsystems such as the A/D reference will begin to fail at 7.5 volts. The USBxCH *will not meet analog specifications at voltages less than 7.5 volts*. The elapsed time for this particular run was approximately 22 hours. Given their relatively small size and good availability, NiMH AA cells are an excellent match for many applications.

Recharging NiMH batteries is fairly easy. While you could remove the 8 AA cells from the battery holder and place them in a commercial recharger, you can also recharge them all while in series still in the holder. Charging rates are quoted in fractions of a full charge C. So, for example, if the capacity is 2Ah, then a C/10 rate would use a constant current of 200ma for charging. If you have a lab bench supply, the eight AA cells in series can be recharged by applying 12 volts at a current limit of 200ma for 10 hours to the battery holder. Disconnect the battery holder from the USBxCH while recharging. Actually, you should apply the recharging voltage for 12 to 14 hours because battery charging cycles are not 100% efficient. *Do not charge at more than a C/10 rate with this method. Do not continue charging for more than 12 to 14 hours with this method either. Do not leave*

the batteries for extended periods with low rate constant trickle charging applied. Failing to observe any of these rules may cause the batteries to become hot and even explode. The problem is oxygen builds up internal to the battery once charging is complete. *We do not recommend fast rate charging at currents greater than $C/10$ for reliable battery life.* With any charging method, stop charging immediately if the batteries become hot.

For longer runs, and applications where trickle charging is required, lead acid batteries are useful. They come in a variety of sizes and Ah ratings. At one extreme are car batteries. Actually, rather than car batteries, deep discharge marine batteries are often a better match. Marine lead acids are designed for complete discharge cycles, while car batteries are designed for short bursts of extremely high current. Both types are readily available at automotive supply stores. Such batteries have an impressive amount of power, with 50Ah or more common. That represents 500 hours = 20 days of continuous operation. Of course the battery is probably powering other equipment too, so the overall run time will be correspondingly less. For more modest applications, there are smaller lead acids. The following photo shows a 12 volt 3Ah lead acid weighing in at 2.9 pounds:

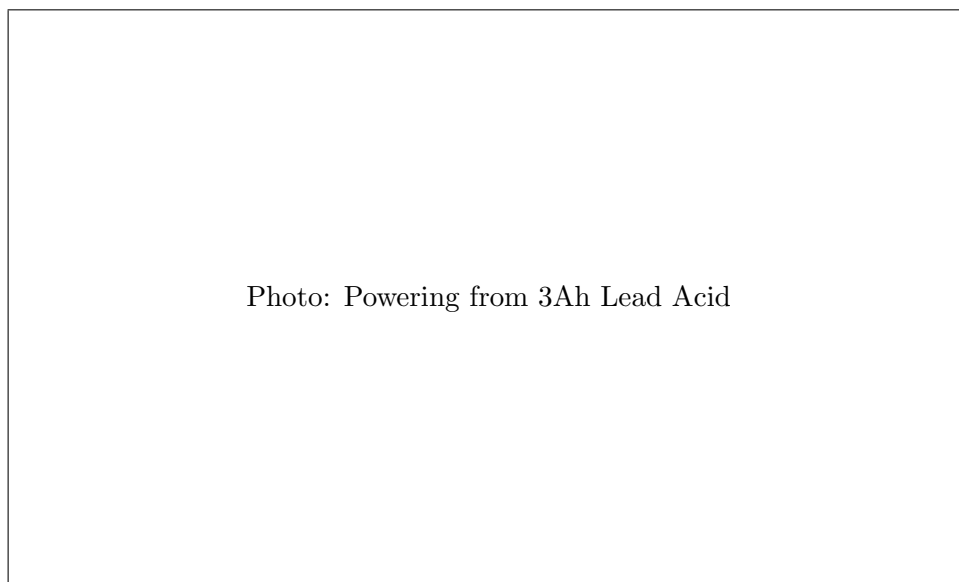


Figure 22.17: Powering from a rechargeable 3Ah lead acid battery

Lead acids are typically 12 volts, so only one battery is required. Although the photo shows alligator clips connecting to the battery itself, there is still a toggle switch so power can be cleanly applied to the USBxCH without excessive sparking. This photo also shows a voltage divider for monitoring the battery voltage with the USBxCH.

It is interesting to compare the discharge characteristics of lead acid batteries with NiMH.

Manufacturer Ah ratings are often poorly specified with regard to discharge rate, so real world tests are important. Figure 22.18 shows the results of a typical run with the 3Ah lead acid. The system actually ran for ?? hours, while one might have expected only 30 hours from the Ah rating. One explanation is the 3Ah lead acid starts at a higher voltage than the NiMH. Another is the low voltage discharge characteristics of lead acid are also better.

Recharging lead acids is not difficult. Any automotive battery charger can be used. Most have an initial fast charge cycle, followed by a trickle charge for maintenance until use.

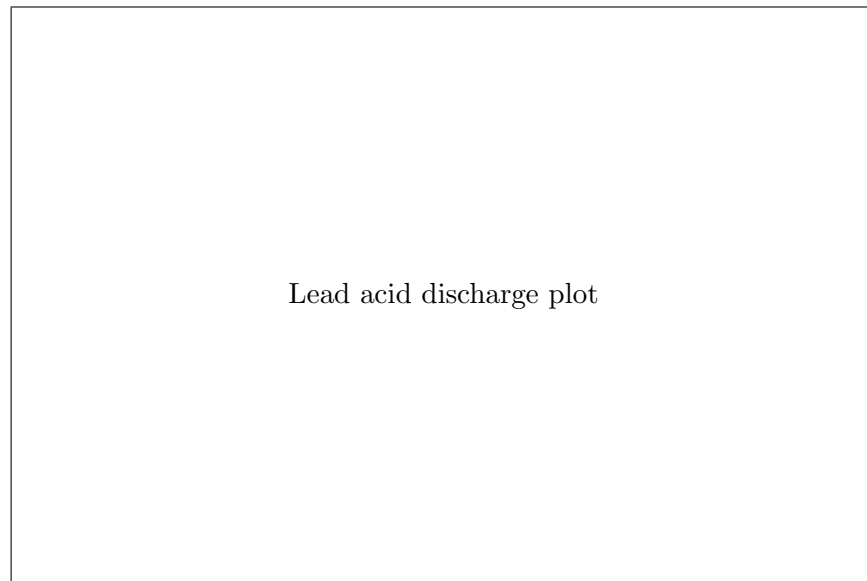


Figure 22.18: Lead acid battery discharge plot

Work underway ...

Fix up the where to purchase section, add reference to NiMH plus lead acid. Also make a new reference for the 4x2 battery holder from DK or MO.

Chapter 23

Frequently Asked Questions

The following FAQ answer a few basic USBxCH questions.

23.1 General

What are the complete system specifications ?

The [Specifications](#) chapter is a good place to start. Other chapters in this manual have substantial additional information. Also refer to the software [Docs](#) directory. Besides the SR documentation, you will also find the Texas Instruments ADS1255 and other key component spec sheets.

Do you include circuit diagrams ?

The [Circuit Diagrams](#) cover the power supply connections, front panel pinouts, analog signal conditioning, and A/D converters. These diagrams have the details required to interface external sensors to the USBxCH. Specifics of the USBxCH internal power supply design and the digital controllers are not published.

Do you include source code ?

The software is pipeline based. Custom programming should start with the pipeline utilities and batch/script files, and not compile C or VB source code. See the [Software overview](#), [DVM](#), and [Scope](#) for details.

C code for a sample a pipeline utility is included in the software [Examples](#) directory. Low level code for the driver etc is not published.

Does the USBxCH save its data to disk files ?

Programs like **Blast**, **DVM**, and **Scope** save their data to disk files. They use various ASCII and binary formats. Refer to each program for details.

What are the Ascii file formats like ?

User data is organized in rows and columns, with one row for each sample point. The columns have the analog and digital inputs, time stamping, and other data associated with that point. See **App - Ascii** for a visual display.

There are options for the files to be "fixed width" or "comma separated values" format. "csv" is convenient for importing into spreadsheets and other third party analysis programs.

What are the binary file formats like ?

Like the Ascii formats, the binary formats are also organized into rows and columns. Each row is a data structure with its fields as the columns. The **Bin data structure** chapter describes the layout in the C language.

The BIN structure can also be easily described in other languages. For a Visual Basic demo see the software **Examples** directory. Most languages can open and read BIN pipelines for custom processing.

Can I use the USBxCH with Excel or MatLab ?

Importing data from saved Ascii files is the easy way to go. MatLab users may also find importing data from binary BIN data files useful.

Is there a user C function library I can link to ?

No. Run the SR utilities to create a BIN output pipe. Then open that pipe in the destination language and process BIN structures as they arrive. This is the easiest way to bring real time data into various programs and languages. And it avoids the problems of linking of object files from different languages.

Can I use the system with LabView ?

LabView and other third party instrumentation apps are not supported. The best approach with such programs is to import one of the file formats, or open an output pipe from the SR utilities and process BIN structures as they arrive. Write LabView VI functions along the lines of the Visual Basic demo.

Can I run under Linux ?

Yes, drivers and applications are provided for Linux. Running under Linux requires LibUsb to be installed on your computer. Most Linux distributions already have this driver installed as part of their installation. See [Getting started](#) for more information.

23.2 Hardware

Is the USBxCH powered by the USB cable ?

No. The USBxCH must be powered by a separate power source like a wall transformer. The USBxCH is referred to as a *self powered USB peripheral*. Power sources such as lab bench supplies and batteries are also acceptable.

Are wall transformers with AC outputs ok ?

No. Wall transformers with AC outputs *will not work*. The USBxCH requires a DC power supply.

The standard wall transformer supplied with the system is a 9vdc unit with 2.1mm center positive connector. Unregulated DC wall transformers are also acceptable. The 50/60Hz ripples are ok if the lows are greater than + 8 volts, and will not degrade analog performance. See the [Specifications](#) table for the allowed power supply ranges.

What do the back panel green and red LEDs mean ?

The back panel green LED will light up if the USBxCH has any power at all. It indicates the wall transformer or other power source is at least energized.

The back panel red LED further indicates whether the power is sufficient for the on board regulators to run correctly. If the back panel red LED is off, then power is ok. Conversely, *if the red led is continuously on something is wrong with the power and must be fixed before the system will run*.

Momentary flashes of the red LED during power up lasting for two seconds or less are normal. See the [Power Supply](#) chapter for more information. After a few seconds if the red LED is off and the green LED is on, the system is good.

Does the USBxCH have any digital buffering ?

Yes. It has a deep DRAM FIFO buffer for storing data, so results can be read

back over the USB cable when the PC has available time. The FIFO prevents data loss even if the PC is heavily interrupted with other activities such as graphics or networking. See the [FIFO Depth and Overflow](#) chapter for how long the system can hold out before requiring service from the PC.

What is the analog input voltage range ?

The analog inputs may vary from -4 to +4 volts for valid count values. This is the voltage on any input pin as measured with respect to AGND. Note the analog inputs are differential, with (+,-,AGND) pins. See the [Analog Inputs](#) chapter for more details.

For *larger* full scale input ranges, there are solder pads on the board for resistor dividers. For *smaller* full scale ranges, the front end amplifier gain may be increased by adjusting the RG and RF resistors. For divider and gain resistor locations, see the [A/D signal conditioning](#) and [A/D input amplifiers](#) circuit diagrams. Note that SR *does not warrant customer modified boards*.

Is it ok to overdrive an analog input ?

Moderate overvoltages less than +/- 20 volts are ok. Each input has a 10K series resistor limiting the current that can flow. Of course, overdriving an input is hard on the system. It is better to add suitable front end circuitry to deal with the problems at a particular site rather than having the USBxCH take the brunt of the overload. Zener/TVS diodes, gas arrestor tubes, and spark gaps are all useful devices depending on the size of the problem.

Do static shocks hurt the analog inputs ?

Yes! Although the USBxCH analog inputs have resistors and capacitors to help snub static shocks, any static discharge is damaging to the system. *This includes small sparks generated when walking over a carpet and touching the analog inputs.* Even barely perceptible ESD (electro static discharge) events can degrade system calibration even if total failure does not occur.

Before touching the USBxCH, discharge yourself by first touching a nearby metal object such as a desk or a computer case. Then touch the USBxCH metal enclosure, and finally the input connectors for any connections. Taking a moment to perform this simple routine will significantly reduce static damage. See the [Analog inputs](#) chapter for more information.

Is the USBxCH connected to system ground ?

Yes. The AGND and GND pins on the front panel DB15 and DB25 connectors

are connected to power ground and consequently also to PC ground via the USB cable. The USBxCH 2.1mm power supply connector is also connected to the power supply ground. If you use a grounded power supply, the system will also be connected to that ground. The SR provided wall transformer is a floating power supply. However even with a floating supply, you are still connected to the PC ground via the USB cable.

Is the system opto-isolated ?

No. The USB signals are direct wire connections between the PC and USBxCH. Users requiring a floating system should use a USB opto-isolator from companies such as Sea Level.

Is there an antialias filter ?

The USBxCH input circuitry has solder pads for simple RC filtering at a number of signal nodes. This type of antialias filtering is intended for protection against RF contamination. The factory default values for these positions are mild so no roll off occurs even for the highest A/D sampling rates. See the [A/D signal conditioning](#) circuit diagrams for details.

What is the 50/60Hz rejection ?

If you need powerline rejection, there are three options: numerical filters, op amp notch filters, and twisted pair. For numerical filtering use high sampling rates and then apply the numerical filter. Any op amp notch filters must of course be placed externally in front of the USBxCH. The most effective solution for 50/60Hz rejection is often to use twisted pair for the inputs.

Do you recommend twisted pair for 50/60Hz rejection ?

Yes! Twisted pair along with proper use of the differential inputs is often a very effective way to achieve 50/60Hz rejection. *Often more effective than any amount of numerical or op amp filtering.* Ideally the twisted pair would also have a foil shield that is connected to the USBxCH case to provide ESD shielding. See the [Analog inputs](#) chapter for more information.

How long can the USB cable be ?

The formal USB specification quotes 6 feet as the maximum. You can probably use a longer cable if needed, 12 feet being reasonable. However, you will not be successful using a 100 foot cable. That is simply beyond USB capabilities. For longer cable runs use USB hubs and repeaters for reliable operation.

Chapter 24

Extra supplies

Extra supplies such as cable and connectors may be required depending on the installation at a particular site. SR can often provide such supplies, however many customers may prefer to purchase such parts themselves. This chapter lists a few typical suppliers and parts for reference. Many others are equally acceptable.

DigiKey
www.digikey.com
1(800)344-4539

Mouser Electronics
www.mouser.com
1(800)346-6873

JDR Microdevices
www.jdr.com
1(800)538-5000

Abbreviations used below:

MFG = Manufacturer
DK = DigiKey
MO = Mouser
JDR = JDR Microdevices

24.1 Small Parts for cables etc

Some of the more common small parts used in USBxCH applications are as follows. See the respective vendor web sites for spec sheets on many of these parts.

2.1mm power plug

MFG part number = CUI Inc PP3-002A
DK part number = CP3-1000A-ND

MFG part number = Kobiconn
MO part number = 1710-2131

These are discrete wire 2.1mm power plugs suitable for soldering wires to. They are useful for connecting batteries and other custom power sources to the USBxCH. Use these connectors rather than chopping the plug off of your wall transformer!

Molex 4 pin power connector

MFG part number = Molex 50-57-9404 (housing)
DK part number = WM2902-ND
MO part number = 538-50-57-9404

MFG part number = Molex 16-02-0103 (female crimp pin)
DK part number = WM2512-ND
MO part number = 538-16-02-0103

For custom field box installations using the bare USBxCH board, you may prefer to provide power to the 4 pin right angle Molex header on the back edge of the board rather than the 2.1mm connectors. Use these Molex parts to build discrete 4 wire power cables. There are a variety of crimp pins with different platings and insertion forces to choose from. The one listed is thin gold plate, normal insertion force.

Alligator clips

MFG part number = Mueller BU-30
DK part number = 314-1010-ND

MFG part number = Silvertronic 501793
MO part number = 835-501793

The Mueller BU-30 has a good spring and strong teeth. These have crimp connections, which should also be soldered for low resistance. Making cables with these clips requires some skill, but results in easy to use connections of good quality. The Silvertronic part is equivalent to the Mueller. This is the clip on the standard USBxCH analog input cable.

DB15 / DB25 solder cup Dshell connectors

MFG part number = generic

JDR part number = DB15S (female) / DB25S (female)

For making custom USBxCH analog and digital input cables, solder cup DB connectors are convenient. They are available almost everywhere at varying prices. The JDR parts above are generic and of good quality at a reasonable price.

Wall transformers

MFG part number = Triad WDU9-500

M0 part number = 553-WDU9-500 (9vdc 500ma 2.1mm plug, 110vac input ** US)

MFG part number = CUI Inc EMS090066-P5P-SZ

DK part number = T948-P5P-ND (9vdc 660ma 2.1mm plug, 110/220vac ** INTL)

The Triad wall transformer is suitable for the US and other countries with 110 vac two prong wall power. A simple unregulated DC wall supply, it will provide good performance at low cost. Many other types of unregulated wall transformers are equally acceptable. The 2.1mm plug must be center positive.

The Cui unit is a switching supply suitable for 110 or 220 vac, with multiplug options for nearly all countries. Although switching supplies will work fine with the USBxCH and not degrade its performance, they will introduce ground noise into a system. Typically the noise will be small millivolt spikes in the 100kHz range. If you are having trouble with small ground noise spikes, consider changing over to a traditional linear supply.

Twisted pair shielded cable

MFG part number = Belden 9501

M0 part number = 566-9501-100

Twisted pair provides reasonable powerline noise immunity for analog inputs. This Belden cable also has a foil shield for ESD protection and is jacketed. There are a large number of twisted pair cable types available. This is only one example.

Double twisted pair shielded cable

MFG part number = Belden 9502

M0 part number = 566-9502-100

If necessary, cable with multiple twisted pairs inside one shield is available. There is no particular advantage other than having more connections inside one jacket.

Potentiometers, 10 turn, 10K ohm

MFG part number = Vishay/Spectrol 534-11103

MO part number = 594-53411103

MFG part number = Bourns 3540 series

DK part number = 3540S-1-103L-ND

*A 10 turn pot is a simple and reasonably accurate device to use as a voltage divider when testing the USBxCH. It is also useful as a sensor for angular measurements. Note **DVM** can be easily calibrated into volts, turns, or other units as necessary.*

Potentiometers, linear slider, 10K ohm

MFG part number = Alps RS6011YA6009

MO part number = 688-RS6011YA6009

Slider pots like the one above are useful for making linear position measurements over small distances of a few inches. They are an example of the variety of potentiometers available. For linear measurements over longer lengths string potentiometers are a popular choice.

Sensors, general

DK part number = Many, see their sensors listings

MO part number = Many, see their sensors listings

Both DK and MO have extensive selections of sensors suitable for use with the USBxCH. Temperature and pressure sensors are just a few in their listings. They can often provide an easy solution to sensor selection. The USBxCH has its own sensor for tracking board temperature. For external temperature measurements a simple silicon diode with suitable excitation is often an inexpensive solution. Geophones for various seismic measurements are obtained from specialty suppliers.

Batteries, 12 volt lead acid

MFG part number = BB Battery BP1.2-12-T1 (12v 1.2Ah)

DK part number = 522-1007-ND

MFG part number = Power Sonic (12v 1.4Ah)

MO part number = 547-PS-1212

These are small lead acid batteries. A good point about lead acid batteries is they are rechargeable. They can even be continuously trickle charged. A 1.2Ah battery has enough power to run the USBxCH continuously for approximately 12 hours. NiMH batteries provide excellent power density, but are not designed for continuous trickle charging. However, eight NiMH cells are an ideal match for 24 hour day runs.

Chapter 25

Using Adobe Reader effectively

Most users will be viewing this PDF document in Adobe Reader. The following tips may help you navigate more quickly. These comments are for Adobe Reader 11, with the same or similar commands and keystrokes in other versions.

Red text items like [Analog inputs](#) are hyperlinks. Clicking one goes to that item. Alt+BackArrow returns from a hyperlink. Alt+ForwardArrow/BackArrow traverse the recent viewing chain, including zoom levels.

You may wish to rotate the [Circuit Diagrams](#) by 90 degrees. Use the key combinations Ctrl+Shift+Plus/Minus to rotate the page. To find instances of particular signal names, do a text search with Ctrl+F.

You may wish to change the page display and zoom level depending on where you are looking in the manual:

| | |
|------------|---------------------------------|
| Alt+V+P, S | single page display |
| Alt+V+P, C | single page display, continuous |
| Alt+V+P, P | two up page display |
| Ctrl+0 | set zoom to fit page |
| Ctrl+1 | set zoom to actual size |
| Ctrl+2 | set zoom to fit width |

If the "single key accelerators" option is turned on in the Adobe General Preferences, then type a Z and drag a rectangle around a specific area you would like to view in more detail. H returns to the normal hand tool.

This manual does not have an index. Use the table of contents, navigation bookmarks, and Adobe text search to find topics of interest.

Chapter 26

Getting Technical Help

Answers to most general questions, resources such as software downloads, and product pricing, can all be found on the SR web site.

If you have *specific technical questions* email us.

General product information: www.symres.com

Software downloads: www.symres.com/downloads

SR technical help email: info@symres.com

If the USBxCH develops a hardware problem, run **Diag debug** and email us a general description along with the diag report file for help.

Emails regarding price discounts will not be acknowledged

That information is available on the web site!



Avoid the ditch
Cathedral Gorge, Panaca, Nevada, 2006

USBxCH User Manual
Copyright ©, Symmetric Research, 2004, 2015

*This material can only be reproduced in whole or part with the written permission of
Symmetric Research*

No guarantee of suitability for any application is made with this document
All liabilities are the responsibility of the user

Email: info@symres.com Web: www.symres.com