SU QuarkNet Workshop 2012 — Lab Activity 5

ELECTRONICS II: ADCs & DAQ

Laboratory Goals

1. Learn about data conversion (analog to digital, ADC).
2. Understand how an ADC works, measure the calibration curve, and determine the frequency limits of an ADC.
3. Learn about computer interfacing, data acquisition (DAQ) systems.
4. Write interactive code to perform digital logic, and read out an ADC.

1. INTRODUCTION

For a general introduction, refer to the first electronics lab activity (OpAmps + SiDets). The present lab will concentrate on the data conversion process and data acquisition systems. Both the hardware and software are available to take with you.

2. PRIMER ON DATA CONVERSION

In the context of the experimental readout chain for high energy physics, we will concern ourselves with the conversion of analog data to digital data. This is the more common direction.

What is analog data? Analog data is often called "real-world data". Mathematically, analog data uses real numbers, and is continuous. This applies to things like time, streaming light, tides, and anything that happens continuously.

What is digital data? Digital data is discrete data. This means it can be recorded in finite, small intervals. To compare with analog data, analog data can be illustrated by a smooth curve on a graph, whereas digital data will always have steps (or simply individual points). One can make the steps so small that the curve appears flat; however in so far as there are separate steps at all, it is discrete data.

Why do we need to convert between them? How do we do it? Refer back to the question on analog data and remember the example of time. Though we have divided time into hours and seconds, and milliseconds—it is in fact continuously flowing, and passing in infinitesimal intervals. So when we need computers to keep time for us, the information needs to be digital. Computers (and digital processors) process only digital data, in the form of bits.

Side note: the word *digital* comes from the word "digit," meaning a single whole number. The word *bit* which comes from the words "binary digit," as in ones and zeros of binary code. A bit is always a combination of ones and zeros.

So when we want a computer to understand, measure, record, or read-out time, it needs to be in intervals, legible as bits. It follows that the most accurate time-keeping devices use the smallest intervals (down to the millisecond, nanosecond, etc.), and seem continuous.

3. AN ASIDE ON BINARY NUMBERS

What are binary numbers and why do we need them? The binary number system is a number line that works like the one we are used to; ours is a number system with base ten, in which there are ten numbers, and all other values are made as a combination of the initial ten. Our system is called the decimal system. "Deci" means ten, and we have base of ten. So one can guess that Binary numbers has base two—one and zero (from "bi" meaning two).

Think of a binary number line, and think of all the sequential numbers using only the digits one and zero. It is: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, etc. In decimal, that list reads: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Or, perhaps it should be written: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, etc., as four-bit "words."

In the decimal system, when we have a three-digit number, we know that the first digit (counting from the rightmost place) is to be multiplied by 10^2, the second digit is multiplied by 10^1, and the third digit by 10^0. It is because we use the decimal system that we take 10 to the power of the place corresponding to the digit. In binary, because it is of base 2, you take 2 to that corresponding power, and that is the multiplier for that digit.

For example, let's look at the number 13. Using the decimal system, we know:

$$13 = (1*10^1) + (3*10^0)$$
$$= 1*10 + 3*1$$
$$= 13$$

Using binary, count up to 13 using the number line above described, and you will see 13 is 1101. How would you identify this number without having to count to it?

$$1101 = (1*2^3) + (1*2^2) + (0*2^1) + (1*2^0)$$
$$= (1*8) + (1*4) + (0*2) + (1*1)$$
$$= 8 + 4 + 0 + 1$$
$$= 13$$

There you have it.

The reason this code was invented and chosen for use in computers, is because it is, in fact, simpler. Once a computer knows how it works, it only has to keep track of two digits, which is the simplest system. The larger a system's base, the harder it is to process. So computers use it. There are different sizes, resolutions, so to speak, of binary digits (bits). Very commonly, a computer will take in 8-bits of information at a time. What does this mean? This means all numbers are in the format: *bbbbbbbb*, the lowest number being; 00000000 and the highest: 11111111. We know 00000000 is just zero. For good practice, let's compute what 1111111 is, and also find out the highest value an 8-bit number can have.

$$11111111 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$
$$= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$$
$$= 255$$

4. ANALOG TO DIGITAL CONVERTERS (ADC)

The process of digitizing real-world data is often done with a device called an ADC (a.k.a. A/D Converter). This ADC (Analog to Digital Converter) can read in volts, but will output an *N*-bit number to the output device, e.g. a computer. The data conversion may be done in a number of ways. For simplicity, you can consider it to be a series of comparisons done in the ADC integrated circuit itself, at a certain characteristic speed for a given ADC, called the sampling frequency.

For more technical details, you can consult the online resources on the activities webpage.

4.1. Transfer Function

The function that relates the input and output is called, quite generally, the **transfer function**. For an ADC, this relates the input voltage to the output digitized number. The *ideal* transfer function of an ADC depends on: (1) the input gain of the ADC, which we will take as unity; (2) the number of bits in the digitized output, called the resolution of the ADC; and (3) the voltage rails for the device, given by the minimum and maximum voltages the ADC input can handle.

Thus the *ideal* transfer function for an *N*-bit ADC is linear, given by

$$D = a + bV_{in}$$

where $a = 0$,

$$b = \frac{2^N}{(V_{max} - V_{min})}$$

and *D* is the digital output number in units of "ADC counts" or "LSBs" (least significant bits). It is a pure number. However the slope contains the units to reconstruct the original electronic quantity—here, Volts.

4.2. Nyquist Frequency

The **Nyquist frequency** is the lowest rate at which one can digitally sample a wave and get a faithful representation of the original wave.

The Nyquist frequency is defined as twice the maximum frequency component of the wave to be sampled. It is a limit. When sampling below this frequency, you will experience a phenomena called "aliasing". This shifts the measured frequency increasingly far away from the actual frequency. Whatever you are using to digitize, it isn't sampling often enough to be able to adequately reconstruct the input shape, and the result becomes distorted and incorrect. So:

$$f_{\text{sampling}} \geq f_{\text{Nyquist}} = 2 \times f_{\text{input}}^{\text{max}}.$$

An ADC can only sample its input so fast. When you input a wave from a function generator, or even just from a potentiometer, the incoming signal is being sampled. Its sampling frequency is a characteristic of a given ADC. Hence in all cases there is a Nyquist frequency with which to contend.

5. CHARACTERIZE AN ADC IN AMPLITUDE AND FREQUENCY

The TNG-3B is an ADC which has eight channels of analog input and eight channels of digital input. For details of its design and operation, see the TNG-3B FAQ on the activities webpage. Each participant should work with their own TNG-3B which they may take with them at the end of the workshop.

Note that the TNG-3B analog inputs are limited to input voltages between 0 and 5 V. ***Avoid putting in negative voltages or positive voltages greater than 5 V.***

1. Create a Transfer Function for the TNG-3B ADC

The first thing to understand about an ADC is how it converts the input analog voltage to an output digitized number. This is the core of the data conversion process.

Set up the equipment to measure the transfer function. You will need to use the LabVIEW program (see instructor) to read out and record the TNG-3B values. Connect the TNG-3B to the PC via the serial port. Connect one of the potentiometers to the TNG-3B analog input channels. Use either the rotary pot or the slider pot. (The rotary pot may be easier.)

Notice that both the rotary and the slider have exposed wires (before they disappear into the cable). When you want to make a measurement, you will measure across the exposed parts, because that's where the current flows. Remember that you are measuring voltage, and you must set the correct magnitude on the voltmeter.

Make a series of measurements of the input voltage (across the pot) using a voltmeter and the output digitized number using LabVIEW. Record these pairs. Measure the full range of input, from 0 V to 5 V, in increments of 0.25 V.

The data you collect will write from the LabVIEW file to an Excel file that you will name and store whereever you choose. Just input the file location in the dialog box that says <File Name>. Remember: whatever you name it must end in .xls for it to be an Excel file. Keep the file closed once you begin collecting data.

Plot the TNG-3B input and output values. Analyze the graph. Make a fit and determine if it is linear. Record the slope and intercept.

Now calculate the ideal transfer function to compare with the slope obtained. Estimate the errors involved. Is this ADC ideal?

Set the input to be some value. Use your calibration constants and the digital output to predict the input voltage. Then measure the voltage with a voltmeter and see how close your prediction was to the actual value. What kind and magnitude of error do you expect to encounter using an ADC? This is one typical characteristic of what you would encounter with any modern device, since almost all use some kind of data conversion process. Examples would be a digital thermometer, digital clocks, cell phone transmission, and digital video (streaming on the net or HDTV).

2. Determine the Nyquist Frequency for the TNG-3B ADC

Connect the ADC to a function generator. Set the generator output to a sine wave, with amplitude between 0 and 5 V (never going negative).

Adjust the frequency of the wave you are sending to the ADC. Start with a low frequency (under 1 Hz). Slowly increase the frequency of the sine wave. Observe the output. As you increase the frequency on the wave you are generating, record the frequency at which it stops resembling a clean sine wave: the output becomes very jagged, and the apparent frequency falls. Eventually you will observe beats. The lowest frequency at which this occurs is an approximate measure of the Nyquist frequency. The input sine wave is still a sine wave, but the wave as digitized (at a fixed sampling rate) becomes distorted.

Look up the sampling rate of this ADC. Calculate the expected Nyquist frequency. Compare your estimate of the Nyquist frequency to the calculated number.

The ADC is not sampling quickly enough to be able to read enough points for every period of the sine wave for it to actually look like a sine wave. The Nyquist frequency is half of the the sampling rate of your ADC. If your ADC samples f_S times/second, and the wave you put in has a frequency of ½ f_S or higher, then the data points are no longer trustworthy for creating a true representation of the input wave. The data points are real (true), but you cannot accurately construct a curve or any other fit to them. Imagine an audio signal distorted in this manner.

When you sample below the Nyquist frequency, you get "aliasing", or shifting of the measured frequency. The name is indicative of the process, as it is when the input signal becomes a signal with another frequency.

3. Use Transducers as Inputs to ADC

Try the different sensors and transducers with the ADC. The sensor set includes:

- Rotary potentiometer
- Slider potentiometer
- Tact switch (digital)
- Bend (flex) sensor
- CdS photocell
- Pressure sensor
- OP999 silicon photodiode
- Magnetic reed relay switch (digital)
- Hall effect switch

See the webpage for the lab activities for details on these (or, here is the direct link: http://www.sensyr.com/manuals/TNG3BSensors.pdf)

Pick a few that you would like to be part of your take-away package. Inform the instructor of your choices.

Don't neglect to try the switches in order to familiarize yourself with the digital inputs.

6. NT SOFTWARE PACKAGE

The major driver of this activity is to allow you some time to familiarize yourself with this TNG3B ADC hardware, senors, and NT software on a PC, as an example of a DAQ system. Beyond that, you can spend some time familiarizing yourself with these items and evaluate it to determine if it would be useful to you in your classroom or lab. Therefore, this lab activity has been structured to be open-ended.

NT software comes with the TNG3B Interface Unit. With NT, you can communicate with the device from the PC and read it out. This can be useful for displaying the output of sensors that are connected to TNG3B. It also can do a number of other things, such as make demonstrations of digital logic gates. The software may be useful for you in classroom demonstrations, as having a simple ADC to read out is often quite handy. It is available for you to download at NeatTools.org.

The TNG3B hardware is also available for you to take. Each participant can get an ADC unit, a serial connector, and two sensors of their choice. Pick the ones you would like from the eight or so sensors available and we will provide them to you in a day or two.

What is NT Software?

To be able to read out your TNG3B on the computer, you need to operate NeatTools. (This is the other software, other than LabView, that you can use and take with you.) NeatTools is software that allows you to design and build circuits and systems of various kinds.

Try some demonstrational/educational examples we have for you before using it to read out the TNG-3B.

Watch the video on Boolean Algebra operations, found on the activities webpage (or directly at http://www.youtube.com/watch?feature=player_embedded&v=rDbeKW1kG-Q#!). Open a new file in NT Software and try to follow along.

Building a Logic Function in NT Software

Next, generate a logic function, a decision-maker of sorts. Do the following:

Open up a new file, and put in two <*SwitchObj*> from the Display Toolbox (labeled DS in the sidebar). As you've probably already figured out from the instructional video and the actual program, there are two strips of buttons to the left of the program window. In those strips are many different functions and toolboxes. Mostly, we'll be using the buttons that <save file> or open a "toolbox". To understand which button is which, hover over the button and see the label that corresponds to it appear in the top line of the window. Generally, the name of anything you hover over will appear in that top line.

Connect them to an <ANDObj> from the Digital Logic Toolbox (labeled DL). Remember: just like in the example videos, to "connect" two things, you hover over the input or output, then click and drag the "string" to the input or output of the second module. . The output of a function is generally on the right side of the image. The input is typically on the right or the top of the image.

Connect the output of the AND function to an LED (also from the Display Toolbox). Why do this? Imagine a scenario in which two things would need to be true for a third to occur.

Push the sA (<Save As>) button to save this program (e.g. to your USB disk).

Now, test the operation of the program. Pressing the top button and the bottom button (i.e. both TRUE) at the same time will cause the output button to be TRUE.

For instance, you'd have to be both hungry and have access to food in order to have <lunch>. Your top button can be the first condition, your bottom-the second. The LED lighting up is the symbol for <lunch>, then.

Maybe you can guess the scenario, but one implementation looks like Figure 1.

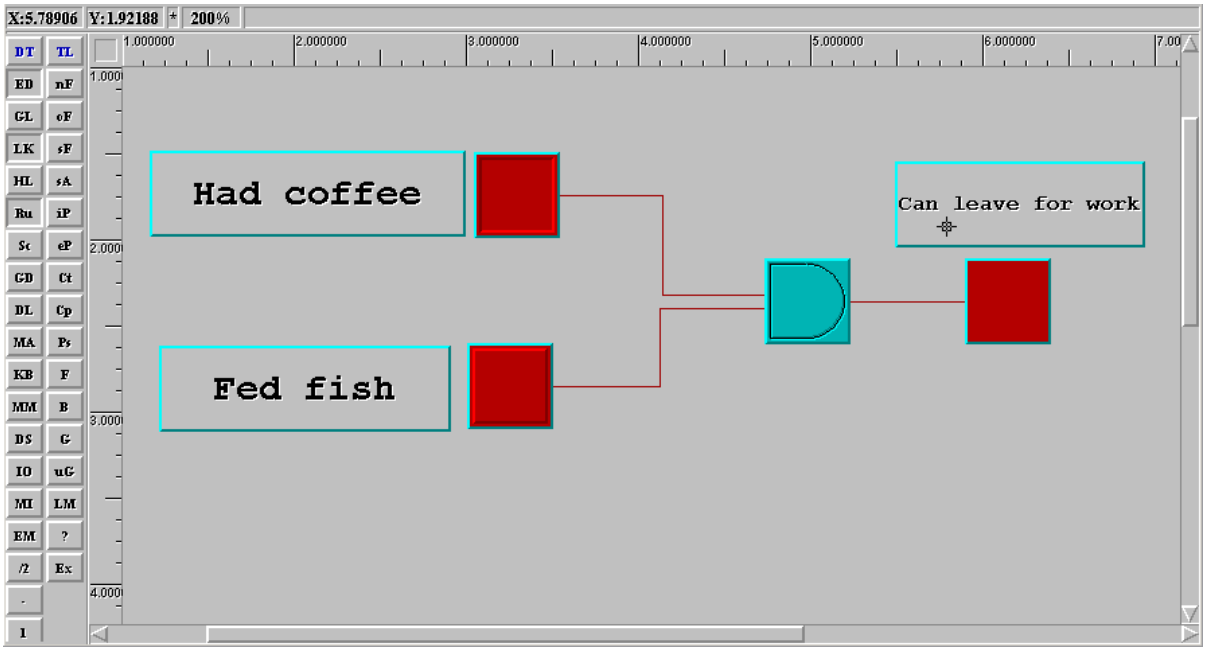

Figure 1. Front panel in NT software. An example of a decision-maker.

Using NT Software to read out the TNG-3B

The developers of TNG3B, SenSyr, have created a program in NT Software to help you operate the TNG-3B.

Once you open it, you will see many buttons, indicators and graphs. If you hover over any button (or function, this includes the ones in the left sidebar), the name of that tool/button/gadget appears in line at the sop of the window.

The eight graphs you see correspond to the eight channels on the TNG-3B itself. The top is the first, second from the top is the second, etc.

Let's go back to the graphs: they represent the signal you read in. To the left of each graph is a function called the CalibrateObj, which takes the "raw" signal the function it gets from the TNG-3B and calibrates it to fit the scale of the graph. As you change the amplitude of the signal

you read it, your graph will change; press the calibrate button names COH1: top row, second from the left (this calibrates all the graphs at once).

To the left of the graphs and the calibration function, is a large function that says TNG-3B; this is the NT Software custom function to read in the TNG-3B.

Modifying the TNG 3B

There are some things you might want to add to this file that aren't there. Things we found helpful: an indicator next to the graph that reads out the actual value of the graph (and indicated the actual voltage you read in).

Here's how to do the first: have a function simply to tell you the value on the graph. Hover over the buttons in the left sidebar until you find a button called: DS, which stands for Display Toolbox Toggle. Click it. Now you have a window of functions you can insert. Choose <IntegerObj Module>. Put it in a convenient place on the diagram (maybe move some things out of the way), and connect it with a wire to the same wire that leads from the Calibration Module to the Graph. Thus you have taken the value that reads into the graph, and also made it read out into a window that indicates the value.

The original is shown in Figure 2, and the modified in Figure 3.

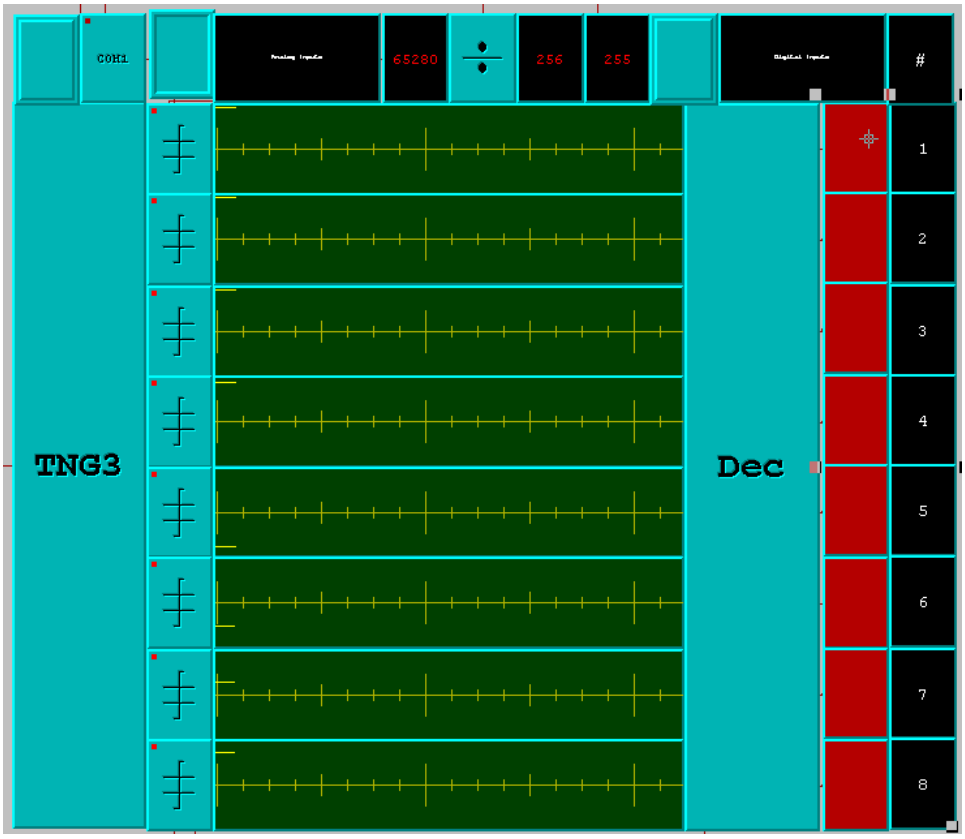

Figure 2. Un-modified TNG3B Program in NT.

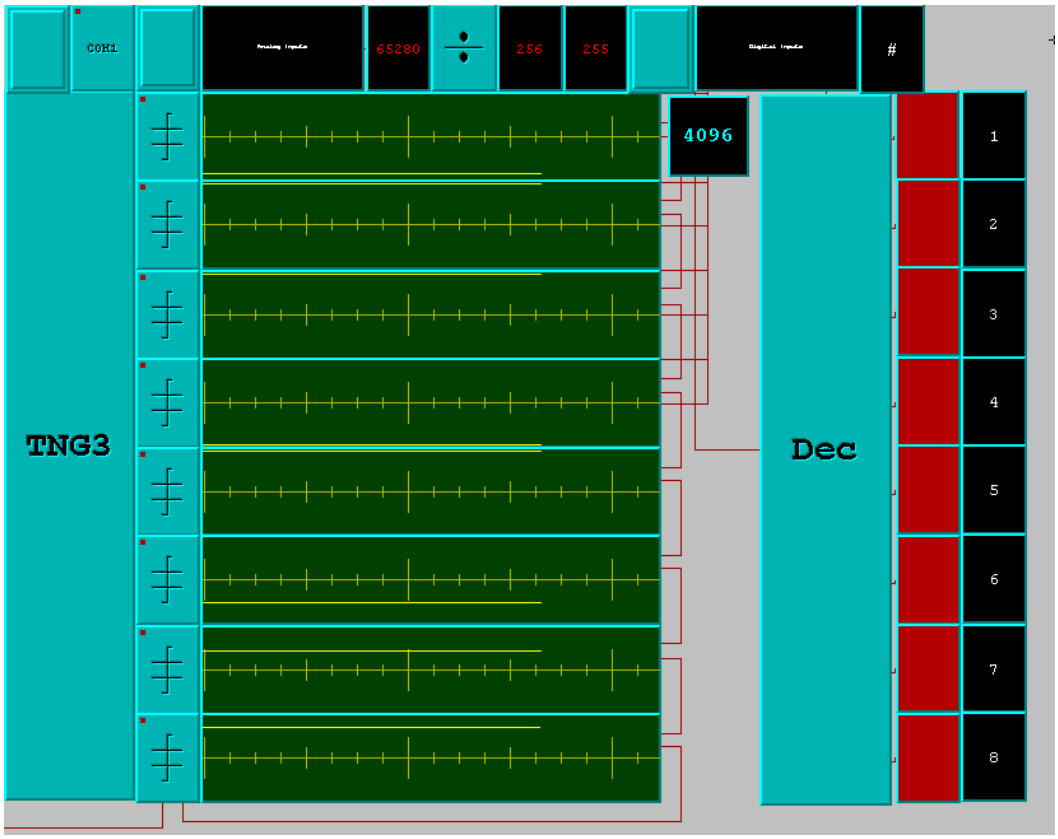

Figure 3. Modified TNG3B Program in NT.

The <IntegerObj Module> was added from a toolbox accessible through the button in the sidebar labeled: DS. It is then connected to the Calibration Module, so that what reads into the graph is what reads into the "counter." The number actually displayed is the number the TNG-3B was reading in at the time. Examine how this first one was done and build/connect the remaining seven yourself. When "examining", just notice that when you hover over modules and the strings connecting them, they illuminate so as you can track them.

More Programming

The programming component of this lab activity is completely open-ended.

If you wish to get more experience with NT software, there are additional demos online. See the webpage http://www.neattools.org/downloads.html, and for video demos on basic functions, refer to http://www.neattools.org/wiki/index.php?title=Category:Tutorials.

7. REFERENCES

[1] TNG3B product page [http://www.sensyr.com/store/index.php?act=viewProd&productId=2]
[2] TNG3B FAQ page [http://www.sensyr.com/TNG3B/TNG3B.FAQ.pdf].
[3] NT NeatTools page [http://www.neattools.org/]
[4] R. Salgado, *Neat tools* (2008). Retrieved from http://www.neattools.org/docs/ntmanual-2008.pdf.

This document was written by: R. Mountain and A. Fadeeva, for the SU QuarkNet Workshop (2012).