Using NCSL DAQ Software to Readout a Struck 3820 Scaler

Timothy Hoagland

December 2, 2004

Abstract

The paper's purpose is to guide the reader in setting up and reading out data from a Struck 3820 scaler. It covers what electronics will be needed and how they should be setup. It will show what modifications will need to be done to the software and gives code to be used. This paper assumes that you are at least somewhat familiar with Linux since the DAQ software runs on a Linux box. It also assumes that you little familiar with C++. Finally it will be helpful if you know how to use an oscilloscope, as that will be needed to setup your electronics.

Contents

List of Figures

1 A Brief Description of the Struck 3820

The Struck 3820 is a scaler that has multiple operating modes, currently only the latched mode is supported. Running in latched mode, the scaler records counts until a latch command is sent at that point the scaler records the current count to the latch register. The count is then reset. The data read out in this example is the from the latch register.

2 A Minimal Electronics Setup

The following items will be needed to build our simple setup:

- Struck 3820
- VME Crate
- VME controller
- NIM Crate
- NIM Discriminator
- NIM Gate and Delay generator
- NIM–ECL converter
- VME CAEN V262 - I/O controller
- Pulser
- 50 ohm LEMO terminator
- Various length LEMO cables
- Oscilloscope

Before starting be sure that you can secure all of these items, many are available through the NSCL electronics pool. Figure 1 shows the complete setup of the system. A description of some the modules used can be found at http://docs.nscl.msu.edu/daq/samples/

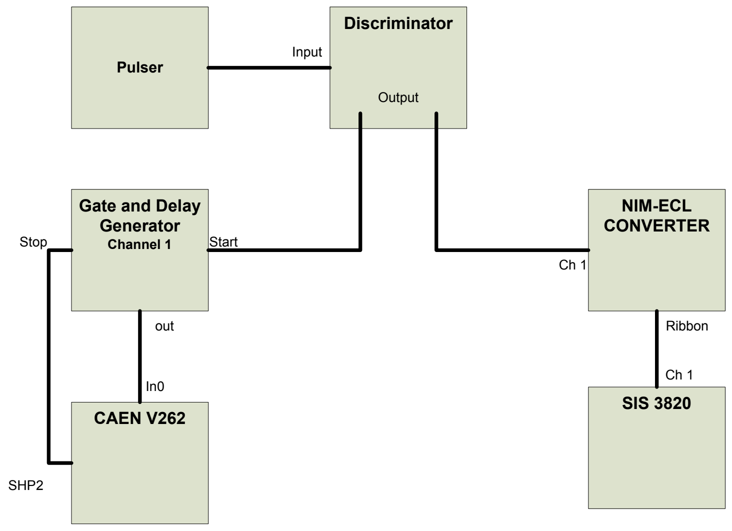


Figure 1: A simple electronics setup to readout a SIS3820

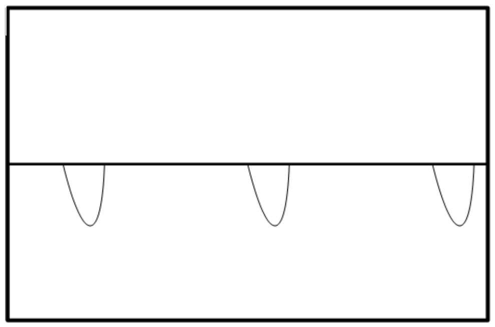


Figure 2: Pulser output signal

We will first look at the signal from the pulser. A pocket will work well for this. A pocket pulsers will give a small negative pulse, but we will invert it later. The signal directly from the pulser will look like figure 2 when viewed on a scope. The pulser only works when terminated with 50 Ohms.

The pulser output will be connected to the input of a discriminator. This could be either a leading edge or a constant fraction discriminator. A discriminator decides when an event has occurred. It does this by putting out a logic pulse every time the input signal reaches a user defined threshold value. Set the threshold so that background noise does not trip the discriminator. It will be useful to look at the both the logic pulse and the signal from the pulser at the same time when setting the threshold value.

NOTE: Ribbon cable can difficult to work with because it is easy to get it twisted and lose track, of which end is which. To avoid this look carefully at the coloring on the cable you are using and be sure it is plugged in the correct channel of the ADC

When the threshold is set connect the discriminator to one of the 3820's input channels via a NIM-ECL converter.

A second output from the discriminator will go to a gate and delay generator; this combined with the CAEN V262 will trigger the computer when an event occurs. This channel will be run in latched mode meaning that it will be given both a start and a stop signal for the gate. The output signal will start being "true" when the start signal is received and stops being "true" when the stop signal is received. The start signal is the signal from the discriminator. The discriminator output will go to the IN0 of the CAEN V262 I/O module. The stop signal is generated by the computer to indicate it has responded to the trigger. That signal is present at the SHP2 output of the V262 and should be connected to the stop on the gate and delay generator.

At this point your setup should be complete. Now is good time to make sure that your setup is the same as Figure 1.

3 Sofware Modifications

The software modification that need done fit into two categories. The first is setting up the scaler, the second is setting up the ScalerDisplay script.

3.1 Setting up the Scaler

The first task we've got to do is define a class for our scaler. The class we create will inherit properties from the CScaler class.

3.1.1 Defining MyScaler

We will start by creating a file called MyScaler.h. This will be the header file for our class. It should look like:

```
#ifndef __MYSCALER_H
#define __MYSCALER_H
#include <CScaler.h>
#include <CSIS3820.h>

class MyScaler : public CScaler
{
  private:
    CSIS3820* module;
  public :
    MyScaler(long BASEA);
    virtual void Initialize();
    virtual void Read(vector<unsigned long>& Scalers);
    virtual void Clear();
    virtual unsigned int size();
};
#endif
```

This includes all of the libraries and classes that we will take advantage of, as well as declaring all of our functions. The next step is to implement the functions. A file called MyScaler.cpp should be created for that purpose. Begin by including a the header file we just created.

Now we can declare our constructor. The parameter BASEA is the base address of the 3820 and will be passed from the skeleton file.

```
MyScaler::MyScaler(long BASEA)
{
  module = new CSIS3820(BASEA);
}
```

We now have to implement the four functions we declared in the header file, let's start with Initialize.

```
void MyScaler::Initialize()
{
  module->Reset();
  module->setOperatingMode(CSIS3820::LatchingScaler);
  module->setInputMode(CSIS3820::InputLatchInhibitAll);
  module->DisableClearOnLatch();
  module->Arm();
  module->Enable();
}
```

Any one time one-time setup detail that you would want to do could be included in the initialize function. The next function we will do is Clear.

```
void MyScaler::Clear()
{
  module->ClearChannels();
}
```

The size function is no longer called but must still be defined we will define it as:

```
unsigned int MyScaler::size()
{
  return 64;
}
```

The Read function causes the current count to be sent to the latch register, reads the data from the latch register, and then clears it to prepare it for the next read.

```
void MyScaler::Read(vector<unsigned long>& Scalers)
{
  unsigned long data[32];
  module->LatchAndRead(data);
  Scalers.insert(Scalers.end(),data, data+32);
}
```

3.1.2 Modifying Skeleton.cpp

At this point we are now ready to modify the readout skeleton so it knows to access our new class. After opening Skeleton.cpp add:

```
#include "MyScaler.h"
```

Now locate the block of code that reads:

```
void
CMyExperiment::SetupScalers(CExperiment& rExperiment)
{
    // Insert your code below this comment.
    // For test,setup an LRS 1151 at 0x200c00

    //CScaler* pScaler = new CVMEScalerLRS1151(0xc00200);
    //   rExperiment.AddScalerModule(pScaler);
}
```

Replace that block of code with:

```
void
CMyExperiment::SetupScalers(CExperiment& rExperiment)
{
    rExperiment.AddScalerModule(new MyScaler(0x10000000));
}
```

The number used after MyScaler is the VME base address to be passed to the constructor we defined in MyScaler.cpp.

3.1.3 Making Readout

We are now ready to compile our code but first we will have to modify the Makefile. Open Makefile and on the objects line add MyEventSegment.o. The line should look like this when your done.

```
Objects=Skeleton.o MyScaler.o
```

Now simply type ”make” at the command prompt. This will compile and link all the needed files and create a file called Readout. Fix compilation errors before proceeding.

You have now created an executable file called Readout, run this program. At the prompt, type ”begin” to start the run. In a separate terminal window type ”bufdump”, to scroll the data you are reading. Type end to end the current run and look at the data that you collected, amongst other information you should have a section that looks like:

```
------------------ Non event ------------
Buffer size is: 90
90 2 680 0 30430 3 32 0
0 0 5 258 772 258 0 0
```

The second word in the readout is the data type; data type 2 is scaler data. If you have scaler data you are ready to proceed.

3.2 Writing the ScalerDisplay Script

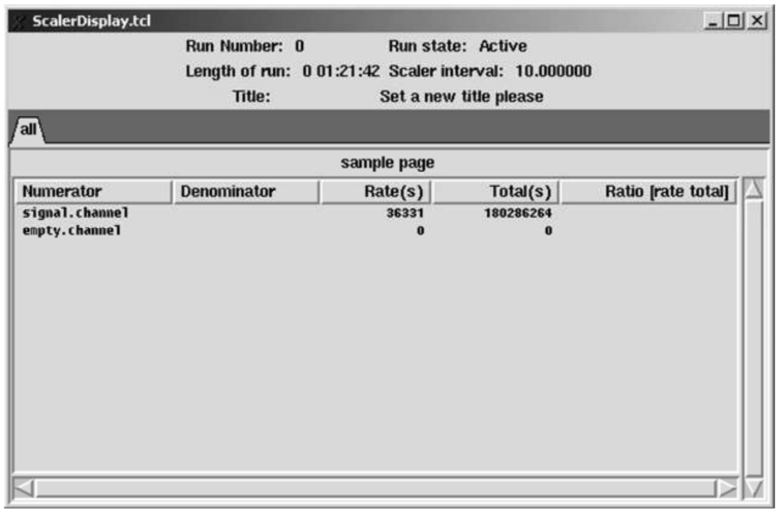
The last piece of code we have to write is the ScalerDisplay setup script. This will be a Tcl script that defines what scaler channels are interesting, names them, and defines diplay options. Call you new file scaler.tcl. It should look like:

```
DAQHOST=spdaq22 //daq computer name, depends on what computer you use

// define channels:  channel  name  ch.#
channel signal.channel  0
channel empty.channel 1

// define pages: page   name   "description"
page all "All channels"

// display single shows counts, on page, channel name
display_single all signal.channel
display_single all empty.channel
```

This piece of code will display channel 0 that has the signal going into it and channel 1 that will have no counts.

4 Running the ScalerDisplay

To display the data read from the scaler you will need to run ScalerDisplay using you scaler.tcl script as a command line argument like this:



Run Number: 0 Run state: Active
Length of run: 0 01:21:42 Scaler interval: 10.000000
Title: Set a new title please

Numerator	Denominator	Rate(s)	Total(s)	Ratio [rate total]
signal.channel		36331	180286264	
empty.channel		0	0	

Figure 3: ScalerDisplay window showing an active and inactive channel

```
ScalerDisplay scaler.tcl
```

After starting ScalerDisplay, start Readout and begin a run. The run status of the ScalerDisplay should change to ”Active”. After a couple of seconds the count data should show up in the counts column. You should now have a display that looks like figure 3)

5 More information

The above example is a very quick and easy way of counting events, however it barely scratched the surface of what can be done to count and display your data. Plese visit http://docs.nscl.msu.edu/daq/scalerdisplay/scalerdisplay.html to get more information on ScalerDisplay.

More information is available at: **http://docs.nscl.msu.edu/** and should be your first source for help. If that doesn't help contact **daqdocs@nscl.msu.edu**

Please help with the accuracy of this paper. If you find any kind of error please report it at **daqdocs@nscl.msu.edu** .

6 Complete Sample Code

6.1 MyScaler.h

```
#ifndef __MYSCALER_H
#define __MYSCALER_H
#include <CScaler.h>
#include <CSIS3820.h>

class MyScaler : public CScaler
{
  private:
    CSIS3820* module;
  public :
    MyScaler(long BASEA);
    virtual void Initialize();
    virtual void Read(vector<unsigned long>& Scalers);
    virtual void Clear();
    virtual unsigned int size();
};
#endif
```

6.2 MyScaler.cpp

```
#include "MyScaler.h"

MyScaler::MyScaler(long BASEA)
{
  module = new CSIS3820(BASEA);
}

void MyScaler::Initialize()
{
  module->Reset();
  module->setOperatingMode(CSIS3820::LatchingScaler);
  module->setInputMode(CSIS3820::InputLatchInhibitAll);
  module->DisableClearOnLatch();
  module->Arm();
  module->Enable();
}

void MyScaler::Read(vector<unsigned long>& Scalers)
{
  unsigned long data[32];
  module->LatchAndRead(data);
  Scalers.insert(Scalers.end(),data, data+32);
}

void MyScaler::Clear()
{
  module->ClearChannels();
}

unsigned int MyScaler::size()
{
  return 64;
}
```