VME Data Acquisition System, ADC Read

UTA-HEP/LC 0023

Shashwat Udit

University of Texas at Arlington

August 25, 2008

Abstract:

This document presents the design concept and the functionality of the newly developed VME DAQ software based on LabView8.0. While the original VME Data Acquisition System that was set up by Dr. Armen Vartapetian was used several years ago in PMT testing for ATLAS experiment, LabView software upgrades to version 8.0 made this change necessary. The system had been controlled by a desktop computer with Windows 95 operating system and LabVIEW 5.1, but the current computer, heppc24 is operating under Windows XP, and LabVIEW software also has a version updates to 8.0. The primary objective of this project was to be able to collect and analyze data from a Gaseous Electron Multiplier (GEM) prototypes and other detector systems through the use of the existing VME DAQ system. The new FAQ software is also allows for accessing all available 32 ADC data channels, along with speed, reliability, data storage in a way that is conducive to analysis, and ease of use.

1.0 The Setup:

1.1 Hardware:

The GEM detector itself works by taking an electron and multiplying its ionization electrons many thousands of times over through an avalanche of electrons in a strong electric field. However, this is still far too small a signal to be sent over long distances through electric resistance, so a preamp converts the small amount of current into a voltage pulse, and from there the amplified signal goes to the Analog to Digital Converters (ADCs).

In order to test the signal readout through the ADCs, a pulse generator is used to provide pulses with known heights. The ADCs currently in the VME Crate are 4 LeCroy 1182 ADCs and occupy slots 17 through 20 in the VME crate. They have 8 analog input channels, two gate channels to receive the trigger signals, two fast clear channels, and one conversion in progress output channel.

eGEM detector itself works by taking an electron and multiplying its strength many thousands of times over. However, this is still far too small a signal to be sent over long distances, so a preamp converts the small amount of current into a voltage pulse, and from there it will go on the Analog to Digital Converters (ADCs). However, for testing purposes, a pulse generator is used to provide the pulse. The ADCs currently in the VME Crate are 4 LeCroy 1182 ADCs, residing in Slots 17-20. They have 8 analog input channels, two gate channels to receive the trigger signals, two fast clear channels, and one conversion in progress output channel.

Two vitally important facts about the hardware setup as follows:

- This conversion in progress output channel must be connected to a logic gate in a way to ensure that no triggers are sent while the ADC is still converting. The ADC will probably freeze if this is not done.
- These are charge integrating ADCs, meaning that they record the total amount of charge between triggers, and convert it into ADC counts. The fact that the ADCs are charge integrating while the Signal is a voltage pulse might require either a charge preamp or a peak sensing ADC.

The VME crate is connected to the computer to through a National Instruments PCI to VME Bridge. This consists of a VME-MXI-2 card in Slot 0 of the VME bus, a MXI-2 bus cable connecting the VME-MXI-2 card to a PCI-MXI-2 card that connects to a PCI Bus on the Computer's Motherboard.

1-2 Drivers: The program to communicate with the VME Crate is written with LabVIEW. A driver is needed to connect the VME Crate and LabVIEW. National Instruments provides two drivers for this purpose. These are NI-VXI and NI-VISA. NI-VXI is the lower-level driver, the one that actually communicates with the instrument. It also comes with a program called Measurement and Automation Explorer (MAX), which enables to user to configure devices.

*It also comes with a program called VXI Resource Manager, RESMAN for short, which needs to be run every time either the computer or VME crate is turned on or off.

NI-VISA is a higher level driver. It calls upon NI-VXI to communicate with the instrument, but it is simpler to use, because several NI-VXI functions can be carried out with one NI-VISA function. Four VISA functions are key to the program. They are VISA Open, VISA Close, and VISA In 16, and VISA out 16. VISA Open and VISA Close, as their name implies open and close VISA sessions to a device with the specified resource name. VISA In 16 reads a 16 bit block of data, given a VISA Session and specified offset. VISA out 16 writes a 16 bit block of data, given a VISA Session and specified offset.

1-3. Process for adding a new ADC:

When adding a new ADC into a slot that is not already used previously, one needs to follow the steps below to allow the software to recognize the new ADC in a new slot.

1. Open up Measurement and Automation Explorer. There is an icon on the desktop for this. Alternatively, following Start→Programs→National Instruments→Measurement and Automation would also work.

2. On the left side in MAX window, click on the plus sign next to Devices and Interfaces to view the systems connected to the computer. (a screen capture would be helpful for this)

3. Select the system the newly added ADCs are part off. If they are put into the VME Crate as currently attached, it is VXI System 0. Right Click and select Create New VME Device. What other systems are there?

4. A new window should open up. It will ask for a series of information. Most of the information, such as manufacturer is ADC Device dependent. Pseudo-logical address can be anything between 256 and 512 that is not being used. (256-259 are in use currently). If MAX does not have the model code of the device in it lists of model codes, the user is going to have to add it themselves. The key step is to know the address range and memory space required.

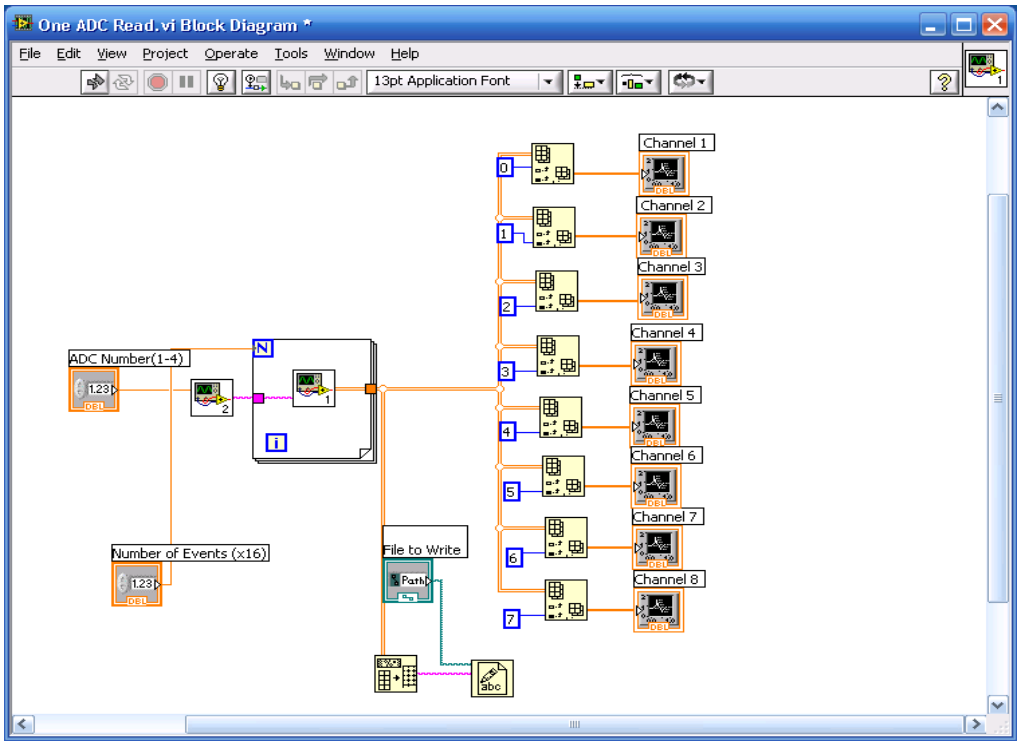
5. Once all the above steps are completed the new device should appear under VXI System 0 (or whichever other system as VXI0::(three digit pseudo logical address selected)::INSTR, as the form of resource name.

2. About New LabVIEW Based VME DAQ Software:

A LabVIEW Program is called a Virtual Instrument or VI for short. Each VI consists of two components. One is the front panel, which is the user interface. Whenever someone opens up a program it is the front panel that is shown. Right click the front panel, and the control palette will appear, enabling one to add controls or indicators to the Front Panel. The second component is the block diagram, which is accessed by going to Window, then Show Block Diagram. The Block Diagram shows the actual code. LabVIEW differs from most other program such as C++ in that while they are text-based programming languages that use written commands, LabVIEW is a graphical programming language. Graphical programming languages have icons that serve as functions, and wires connected between them represent the flow of data. Right click the block diagram, and the functions palette will appear, which enables one to add stuff to the block diagram. An invaluable resource when using LabVIEW is the online help. Just type in LabVIEW and For

Loop or Case Structure whatever else information on is needed, and it should come up in the first couple hits.

2.1



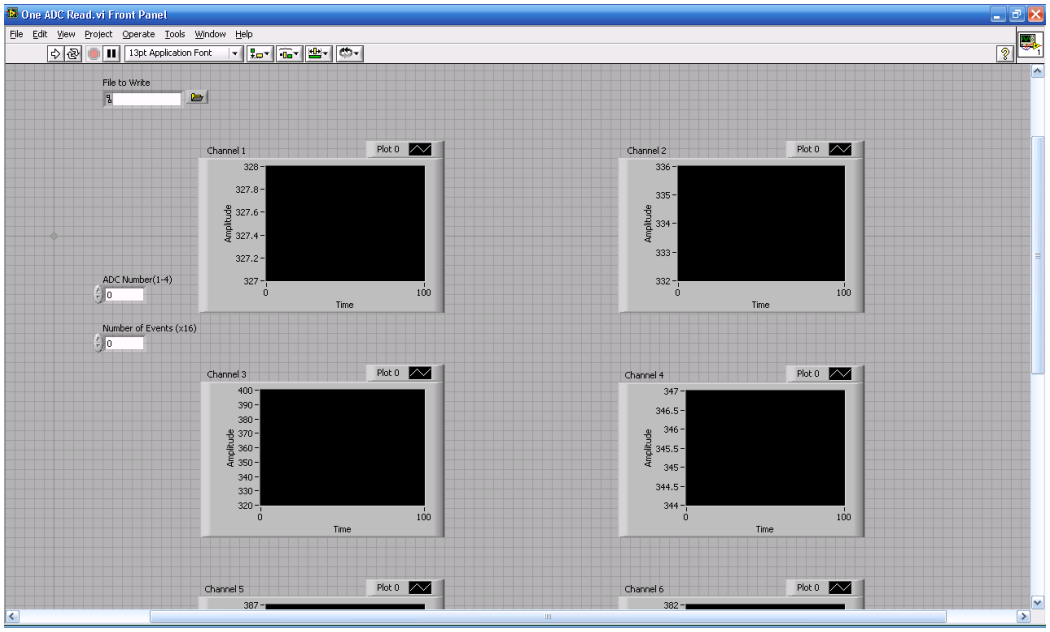


Figure 1: Screen Capture of Front Panel and Block Diagram of One ADC Read.

Overview of the Program:

Looking at the front panel and block diagram shown in Fig. 1, you should notice common features.. Features on the block diagram that correspond to objects on the front panel have labels above them that are the same as the front panel object. Most obvious are the 8 graphs on the front panel, (there are some on the bottom not visible in the picture), and the 8 graph icons on the block diagram. These are the outputs of the program. You should also notice the inputs ADC Number and Number of Events (x16) that appear on the left side of the block diagram, and "File to Write" menu item towards the bottom. These are the user inputs. The wires on the block diagram represent the flow of data. An orange wire means numeric data or array of numeric data. Dark blue also is numeric, but it is a numeric data that is constant and cannot change from run to run. The purple wire indicates string. The light blue indicates file path. The double orange lines mean a 2-D array of numeric data. The 3-D looking set of black lines is a "for loop". There a two blue boxes inside the "for loop". The blue box with the n inside of it is the number of the times that the loop will run. The blue boxes with the""" inside it is the iteration number, or the run the for loop is currently on. The other blue boxes on the block diagram, the ones with numbers inside of them are numeric constants. A purple box with characters inside it is a string constant. The icons are the stuff that actually do something.

An icon like the one to the right, with an oscilloscope, a multiplication sign on it, and a number in the lower right corner indicates a SubVI. A SubVI is a smaller program the larger program calls up. Double click on a SubVI and it opens up. There are two SubVIs in this program that do most of its work.

This icon is found toward the bottom of the block diagram. A 2-D Array goes in to it, and a string comes out. The icon is the Array to Spreadsheet string function. As it name and data flow indicate, it takes an Array and converts it into a group of strings. The string constant containing %f indicates to it that the array contains floating point values.

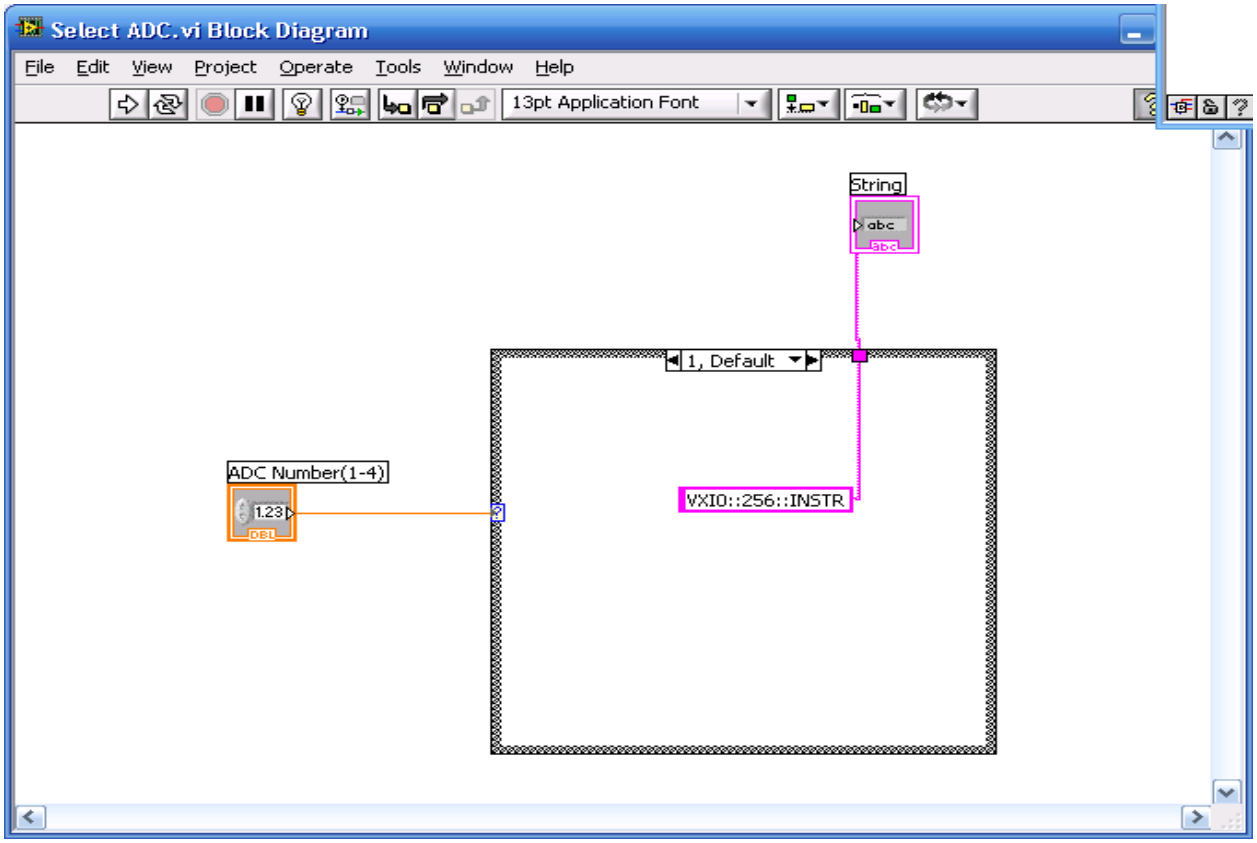
This icon receives the string that leaves Array to Spreadsheet string, as while as the file path that the user specifies. This is write to text file, and it takes the data and saves it to a txt file.

There are 8 of these icons to the right side of the block diagram. They receive the 2-D array and a value from one of the 8 numeric constants. They output a one-day array to the graphs. These are the index array functions. They take an array and the number of a column, and return the values of the column to the graph.

The way that the program works is readily apparent. The user specifies an ADC Number, which goes to a SubVI, which returns a string that goes on the other subVI. The second SubVI lies inside a for loop, which executes the same number of times as user specified for the number of

events, and the Second SubVI returns data in the form of a 2D array. One copy of the data is converted into strings and saved to a .txt file, while another is sorted and displayed in graphs corresponding to the appropriate channel number. However, the bulk of the work is done in the SubVIs, and so it is difficult to understand the program without understanding how they work.

The first SubVI, which somewhat paradoxically has a 2 in the lower right corner, has its block diagram



shown on the next page.

Figure 2: Select ADC.vi

The thick black square shown in Figure 2 is a case structure. A case structure has a number of bits of code called cases, only of which executes. Which one executes depends on what value is received by the input terminal. This particular case structure has 4 cases, numbered 1 through 4. Inside each case is a String Constant which contains a VISA Resource Name for an ADC. Case 1 has the Resource Name for the leftmost ADC, and Case 4 has the Resource Name for the rightmost ADC. In effect, the user types in a number 1-4, and this SubVI outputs the resource name corresponding to that ADC. If some other number is typed, or nothing at all, Case 1 executes.

* Keep in mind that this SubVI works only with the current setup. If ADC as added, removed, or replaced, cases might need to be added, removed, or the resource name might need to be changed. To find out how to know the resource name, refer to the process for adding a new ADC.

Looking back at the block diagram for the original program, it can be noted that resource name goes on to another subVI, which is inside the for loop and outputs a 2-D Array of numeric data. This SubVI is the part of the program that actually communicates with the ADC. The block diagram is shown below, and as you can see, this SubVI consists **mainly of two SubVIs of its own.**

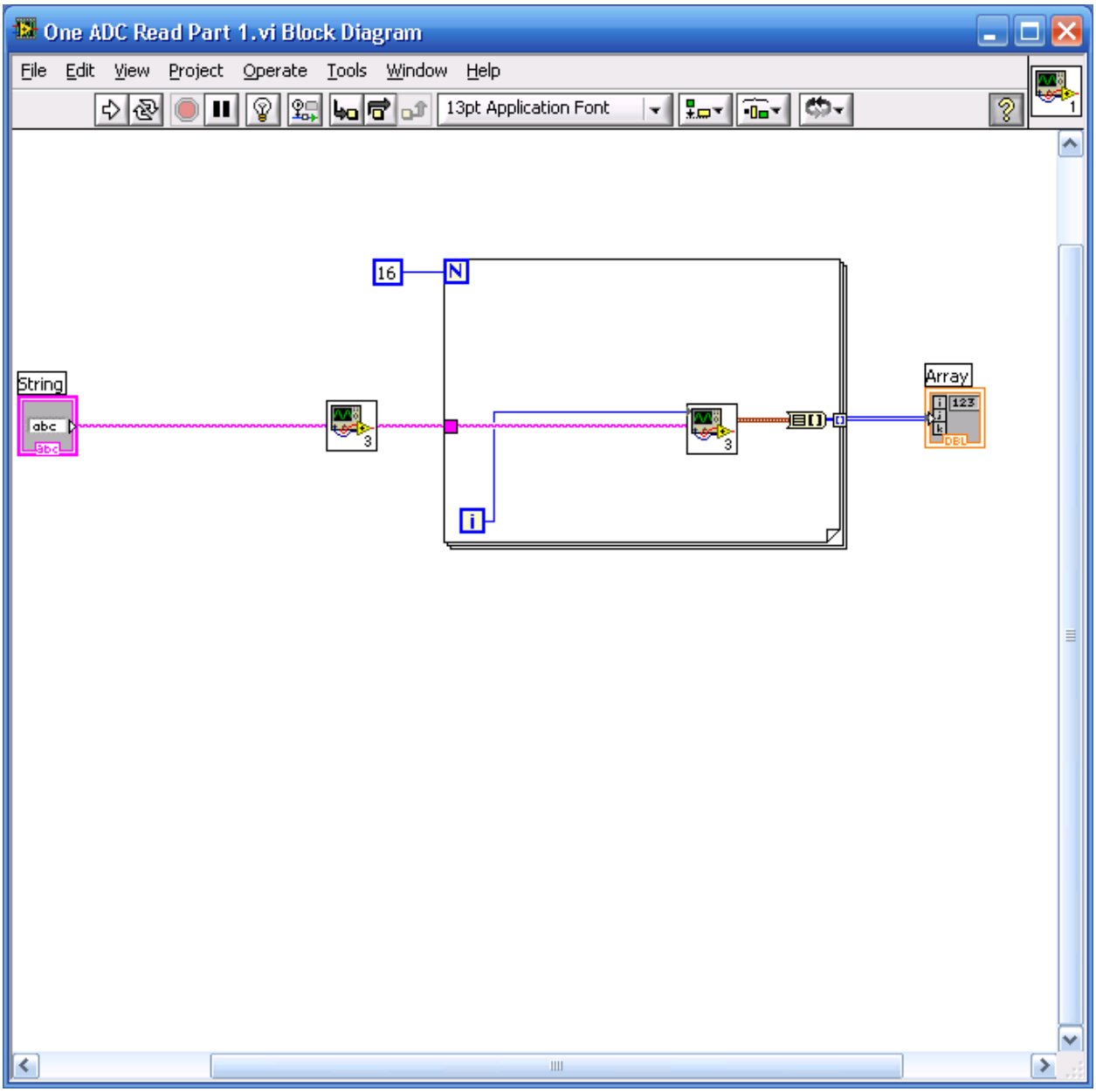


Figure 3: One ADC Read Part 1

These two SubVIs are Reset CSR0 and ADC Block 2. They use the NI-VISA Functions discussed in the drivers' section to communicate with the ADC whose resource name was specified by the previous SubVI. The block diagram for Reset CSR0 is shown on the next page.

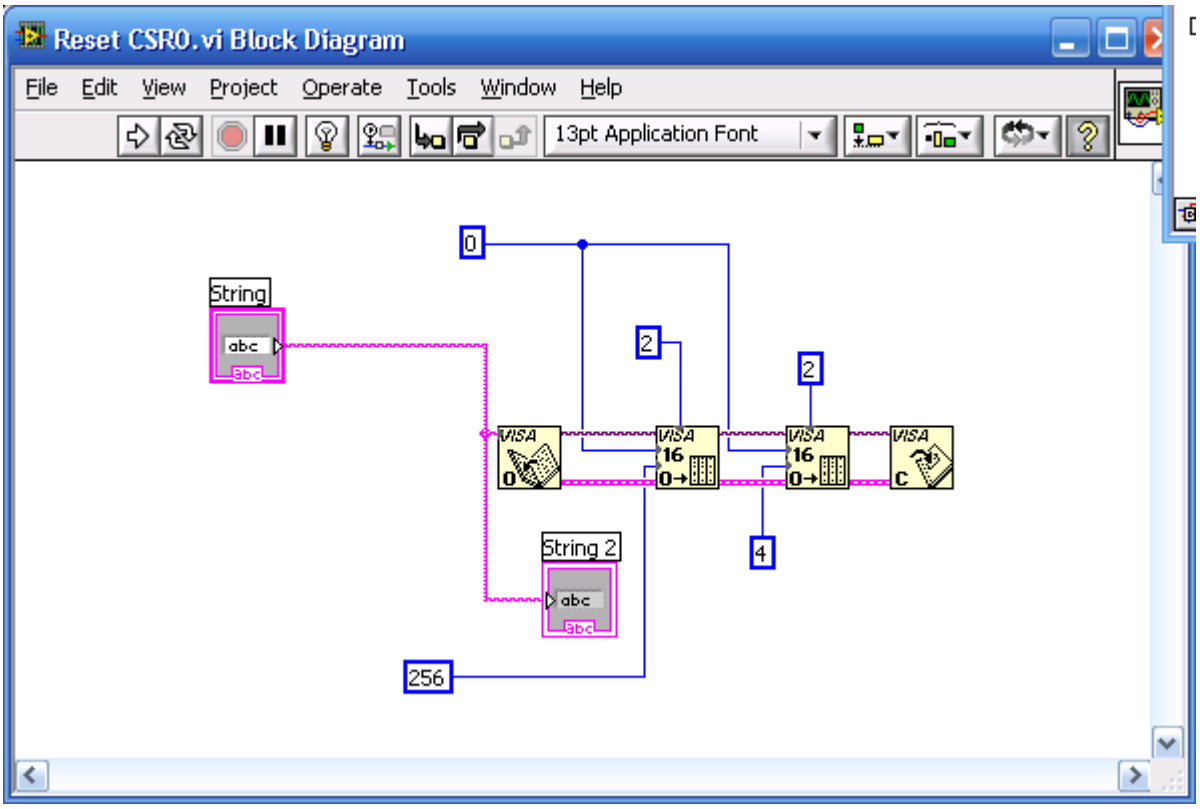


Figure 4: Reset CSR0

As shown in Figure 4, this SubVIs' only input is the resource name of the ADC. There a four NI-VIS A functions in it. These are respectively VISA Open, VISA Out 16, VISA Out 16 again, and VISA close. The VISA out functions requires 4 inputs. The first input is the VISA Session which is the dark purple line towards the top, which is an output of the VISA Open Session. There is the numeric input toward the top, which is two for both of them. This specifies that the device uses A24 memory space. If it was one, it would mean A16 Space, and if it was 3, it would mean it uses A32 Space. If new ADC are installed, this would be another input that might need to be changed. There are two more numeric inputs to the side of both VISA Out 16 functions. One is zero for both. This is the offset. The second numeric input is 256 for the first function and

4 for the second one. To understand what this does, one must look at the Address Space configuration for the ADC, as well as a description of the Control Status Register bits.

CSR0 Bit Definitions	
D0	CONV_COMP (Conversion complete) (read only).
D1	CIP (Conversion In Progress) (read only).
D2	Rear panel/front panel gate select; when a 1 is written, front panel is selected. Rear panel is the CERN Jaux Dataway, if in place.
D3	Event Buffer not full (EVENT_FULL*); when a 0 is read, the Event Count (D4 Đ D7) will be 0 indicating that 16 events have been stored in the memory.
D4 Đ D7	Event Count (0 Đ 15).
D8	Clear module; when a 1 is written, CONV_COMP and EVENT_FULL are cleared and reset. The event counter and channel counter are both reset to 0. Resets itself, always reads 0.
D9	Causes a test gate of 500 nsec to be generated when a 1 is written and then clears itself, always reads 0.
D10 Đ D15	Not Implemented.

Address Structure	
A16 Đ A23	Base address (switches)
A9 Đ A15	Invalid addresses

A1 Ð A8	CSR and RAM space
A0	Undefined in VME
0x0000, 0x0001	CSR 0
0x0002, 0x000F	Invalid addressing
0x0100, 0x0101	Channel 0 Event 0
0x0102, 0x0103	Channel 1 Event 0
0x0104, 0x0105	Channel 2 Event 0
0x0106, 0x0107	Channel 3 Event 0
0x0108, 0x0109	Channel 4 Event 0
0x010A, 0x010B	Channel 5 Event 0
0x010C, 0x010D	Channel 6 Event 0
0x010E, 0x010F	Channel 7 Event 0
0x011_	Channel 0 Event 1

The offset of zero means it is writing to the control status register (CSR 0), and the value of 256 is writing a 1 to bit 8, which clears it. The value of 4 is writing a one to bit 2, which tells the ADC to read from the front.

* If the ADC is changed to read from the back, which it don't think is possible without a special type of adapter, this will need to be changed.

Only after the four is sent does the ADC start reading. The intent of this SubVI is to clear the ADC, and then have it start reading again. Looking back at the block diagram, one will notice that the string input which contains the resource name also goes on the string output (String two). This goes on to the next SubVI within the VI, ADC Block 2. The reason it happens like this is to make sure Reset CSR0 executes before ADC Block 2. LabVIEW will run multiple things at the same time, so the data flow must be controlled to make sure things will run in order.

ADC Block 2 is in a for loop that executes 16 times. This is because the ADCs memory buffers hold sixteen events. Each time that ADC Block 2 executes, one event is read from each of the eight channels and outputted as a numeric cluster. This is converted to an array using the cluster to array function, and then when it reaches the for loop is indexed into a two-dimensional array which is outputted. The block diagram for ADC Block 2 is shown below.

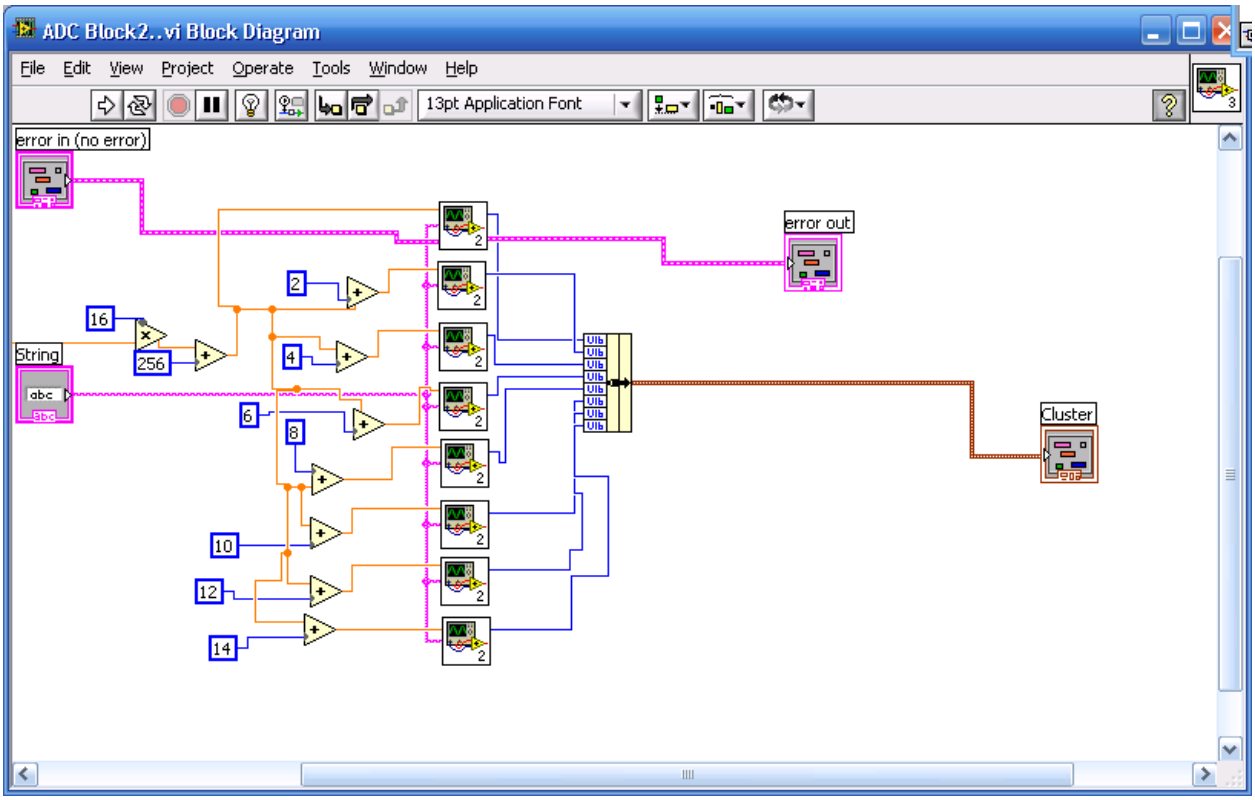


Figure 5: ADC Block 2

Figure 5 shows that ADC Block 2 has eight instances of a subVI inside of it. This is Block 16.vi. It is a simple VISA Open, In 16, and Close. However , the most complicated part of ADC Block 2 is the offset calculation. ADC Block 2 receives as a numeric input the number of iteration the for loop is on. It is multiplied by 16 to get the correct offset for the corresponding event. This is added to 256, the offset at which the memory for the events start. Then the value is sent to all 8 Block 16 SubVIs, but before it is sent in, the offset value corresponding to the channel is added. It is 0 for the first channel, 2 for the next one, 4 for the next on and so on.

* The offsets are another and probably most difficult thing to change if another ADC is installed

The Primary differences between this and All ADC read, is that there is no selector, and all the string constants are used and there are four output arrays, which are clustered and un-clustered as needed, so the SubVI structure looks a little different. However, when the SubVIs are opened up, besides the 4 fourfold multiplication and the unbundling and bundling, the mechanisms are the same.

3. Remaining Tasks and Conclusion

The main thing left undone is that there is no coordination between the program and the rate that triggers come in. The program as it currently stands assumes that triggers come in faster than the program can read, and that the memory buffers of the ADC are not overwritten. It therefore reads 16 events, clears the ADC and starts reading again. However, if the triggers come in at a slower rate, it might be necessary to add conditional statements, perhaps a while loop, that first reads the control status register to ensure that there are new events to be read, before having the program read for events. In contrast, if the triggers come in extremely quickly, and the event buffer is overwritten, it might make sense to get rid of the Reset ADC.vi altogether. It is not even entirely certain whether the writing a one to the CSR 0 bit 8 even clears the event buffers, or just the control status registers. There has been some stickiness noted in the values read from the ADC, though this might just because the noise level stays fairly constant. Finally, the manner in which the data is displayed might not be fully consistent with the needs of analysis. Currently it is written to a text file, which each channel a column, and displayed on a waveform chart, one chart for each channel. LabVIEW has a variety of ways to handle data. There are many forms of indicators and graphs that can be used to display the data, there are a variety of array functions that could be used to organize that data in different way , and data can be written to several kinds of files, including spreadsheet and binary. Feel free to make whatever changes are necessary.