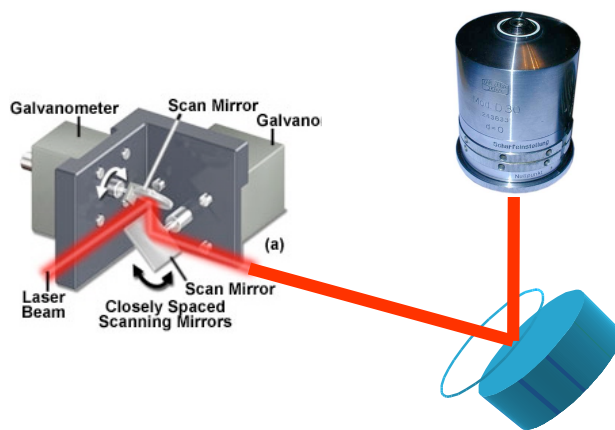
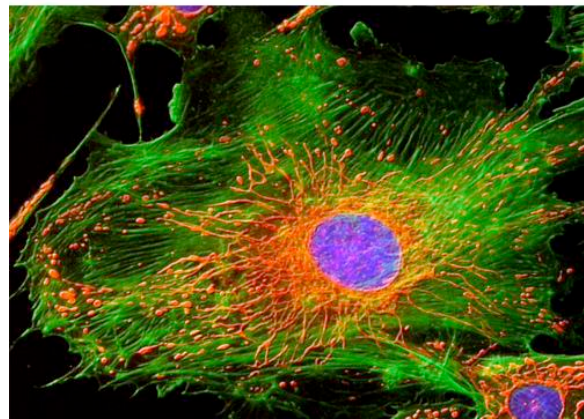


# Software Control of a Confocal Microscope

Giang Vu



Bovine Pulmonary Artery Endothelial Cells (BPAE Line)



Master Thesis in Electrical Engineering

30 hp, Oct 2007- Jun 2008

Department of Measurement Technology and Industrial Electrical Engineering.

Department of Chemical Physics

Supervisors: Tomas Jansson, Associate Professor, Lund Institute of Technology, Lund.

Ivan Scheblykin, Associate Professor, Lund University

# Contents

|   |    |
|---|----|
| 1. Abstract .....                           | 3  |
| 2. Acknowledgements .....                   | 4  |
| 3. Introduction.....                        | 5  |
| 3.1 Background and purpose.....             | 5  |
| 3.2 LabView .....                           | 5  |
| 3.3 Confocal microscope.....                | 6  |
| 3.3.1 Fluorescence .....                    | 6  |
| 3.3.2 The pinhole.....                      | 7  |
| 3.3.3 Dichromatic mirrors.....              | 7  |
| 3.4 The Microscope .....                    | 7  |
| 4. Materials .....                          | 9  |
| 4.1 Overview.....                           | 9  |
| 4.2 SC 2000 Digital Scan Controller .....   | 9  |
| 4.2.1 System Overview .....                 | 9  |
| 4.2.2 Command Line Interface .....          | 11 |
| 4.2.3 Assemble Language .....               | 12 |
| 4.3 Data Acquisition... ..                  | 12 |
| 4.3.1 Digital Signals .....                 | 13 |
| 4.3.2 DAQ Hardware .....                    | 13 |
| 4.3.3 Driver and Application Software ..... | 14 |
| 5. Implementation .....                     | 16 |
| 5.1 Scan program generation .....           | 17 |
| 5.2 Buffered Counter .....                  | 18 |
| 5.3 Buffered Counter at 1 spot .....        | 27 |
| 5.4 Read Binary File .....                  | 27 |
| 6. Result and Discussion .....              | 29 |
| 7. References .....                         | 30 |

# 1. Abstract

Confocal fluorescence microscopy is an optical imaging technique that has high resolution and can be used to detect individual fluorescing molecules. It is commonly used in biology, medicine and materials research. A confocal microscope system contains scanning mirrors, objective lens, laser and detector. The laser beam is directed by 2 scanning mirrors and it will be focused by an objective lens onto the sample and the emitted fluorescent light is collected by a photodetector. In order to obtain an image of the sample, the laser beam is scanned along the x-y axis and the intensity is function of the coordinate  $I(x,y)$ .

When a photon hits the photodetector, a short TTL pulse is generated. In order to obtain light intensity we need to know the number of photons (TTL pulses) over a fixed time interval (for example 10 ms). This has been realized with a National Instruments Data Acquisition (DAQ) card model NI 6602, which contains 8 high frequency counters.

This master thesis is about developing a LabView software to control the scanning and the DAQ card. It is divided into four different programs, the first for controlling the mirrors, the second is for plotting the intensity on a graph and save the data in a binary file, the third is for counting the number of pulses on one specific spot and the last is for analysing data.

## 2. Acknowledgements

This master thesis is the last part of my education to Master of Science in Electrical Engineering. This project has been performed from October 2007 to May 2008 at Chemical Physics, Lund University.

I would like to thank my supervisors Tomas Jansson at Electrical Measurement and Ivan Scheblykin at Chemical Physics for their help and guidance. The reason for me having this project was I found LabView programming was fun and when this project came up I thought this is the chance to enter deeply into LabView.

Lund, May 2008

Giang Vu

# 3. Introduction

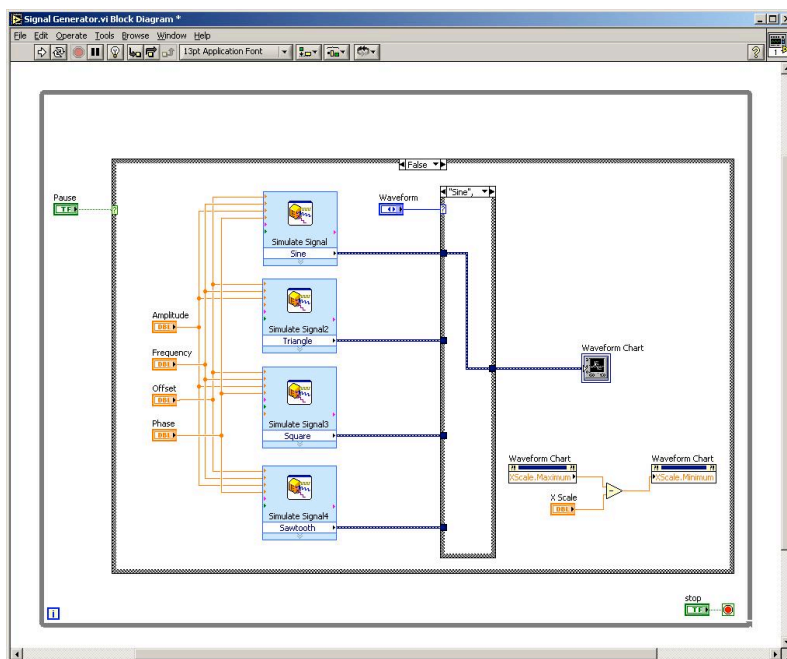
## 3.1 Background and purpose

In the Chemical Physics department the use of fluorescence technique is very common and it uses as a part of this confocal microscope. This kind of microscope gives several advantages over a traditional microscope, for example it is able to reproduce a 3-D image. This department has bought a confocal microscope with software enclosed and what they needed was a program for computer control of the microscope and the data acquisition, which will be discussed later. The requirement of this project was to develop software that can manage the whole operation from scanning to analysing data. The software for both the microscope and the DAQ card was implemented in Labview.

## 3.2 LabView

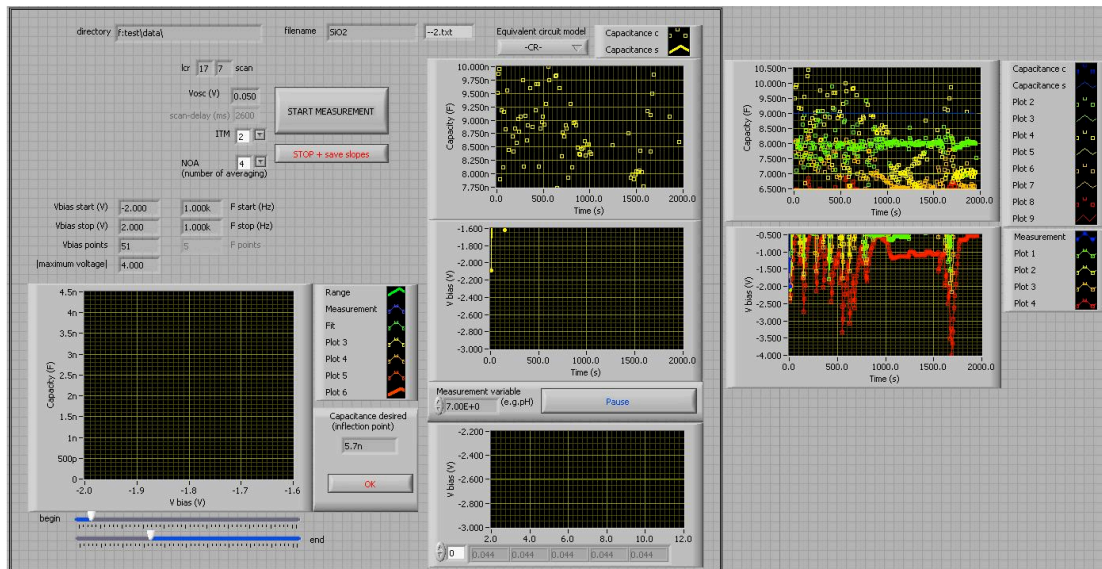
**LabView** (Laboratory Virtual Instrumentation Engineering Workbench) is a graphical programming language from National Instruments. When programming in Labview the user place symbols and wire them together instead of typing in text as traditional programming language like C, Java, Visual Basic etc. Therefore it is easier to program even for the user with less programming experience. Labview is commonly used for data acquisition, instrument control and industrial automation because of an extensive support for instrumentation hardware.

Labview is available in different operating system as Windows, Mac, Unix and Linux. Every Labview program has three components, a block diagram, a front panel and a connector pane. Block diagram is the main part of the program, it is here the user do the programming, the program is built up by placing and connecting VI:s (symbols) together. The picture below shows an example of a block diagram.



Block diagram.

The front panel is where the programmer creates the user interface, defines input and output data by using controls and/or indicators.



Front panel

Controls can be push buttons, knobs, switches or other input device. Indicators can be graphs, LEDs, and other output displays.

The connector pane is used when the programmer wants to use a VI as a subVI, it is here the programmer defines input and output of the subVI. It corresponds to parameters when calling a function in textbased programming. A subVI corresponds to a subroutine.

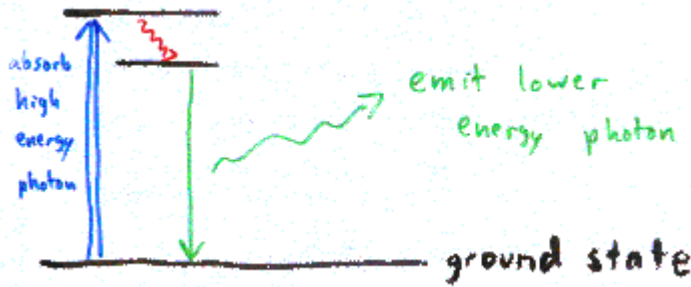
### 3.3 Confocal Microscope

Before explaining how a confocal microscope works I would like to go through some principles which together build up the microscope.

#### 3.3.1 Fluorescence

Fluorescence is a process where a substance sends out light with a different wavelength than the light it absorbed earlier.

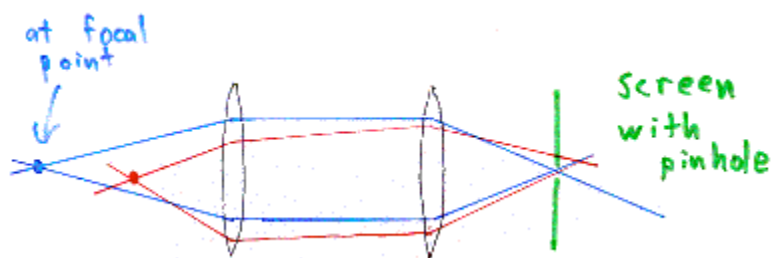
For example if you shine blue light on some molecules it will cause the electrons of the molecules to jump to a higher energy state (top black line in the picture). Some of the energy is lost internally, maybe in form of heat (representing squiggly red arrow). When the electrons jump back to the ground state the molecule emits green light that contains lower energy than blue light.



One of the advantages of fluorescence for microscopy is that you can attach the dye molecules to your sample and therefore only that part of the sample can be seen by the microscope.

### 3.3.2 The pinhole

Picture below shows how the lens focus light from a focal point to another point, the light is represented by a blue line. The red line is light from other point than the focal point but still get imaged by the lens of microscope.



The idea with the pinhole is to block light that is not from focal point. Normally, the excitation light illuminates whole sample, so the entire sample is fluorescing at the same time. If we build up an image with light from all points at the same time then the background would be hazy. What we want to do is look only at the light from the focal point and this can be done by adding a pinhole so all light passes from the focal point but light from other point is blocked.

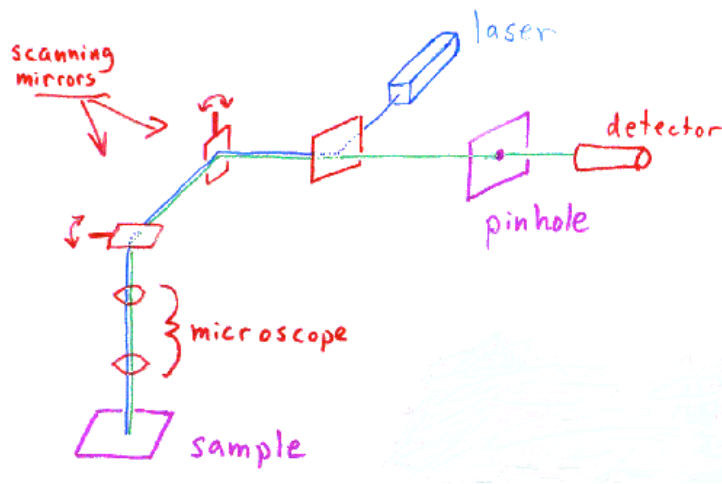
### 3.3.3 Dichromatic mirror

A Dichromatic mirror is a mirror that reflects light shorter than a certain wavelength but passes light longer than that wavelength.

### 3.3.4 The microscope

Now we have all concepts to build a confocal microscope.

A laser is used to provide the excitation light since it is necessary to obtain high intensity. The blue light reflects by a dichromatic mirror and hits two scanning mirrors that are mounted on motors. These mirrors are used to scan the laser beam across the sample. Dye in the sample fluoresces and emits green light, the light take the same way back but get through the dichromatic mirror and is focused onto the pinhole. A detector measures the light that passes through the pinhole.



The light from the sample does not give the whole picture because the laser does not cover up entire sample, only from one point to another. While the laser scans across the sample, the detector collects emitting light from these points and builds up an image of the sample. The limitation of the microscope is the scanning mirrors, the faster the better.

Confocal microscopy has several advantages over conventional optical microscopy, it allows one to visualize deep into cells and tissues. By having the pinhole, the microscope is able to eliminate out-of-focus light, which gives it much better resolution. The best thing is it can build up a very clean 3-D image by using the pinhole to eliminate out-of-focus light.



## **4. Materials**

### **4.1 Overview**

Confocal microscope's main parts are laser, photodetector, objective lens and mirrors. When scanning over an area only the mirrors move and the rest of the microscope are fixed. The mirrors are mounted on a board called SC2000 Digital Scan Controller from GSI Lumonics, this SC2000 can be programmed in order to control the mirrors movement.

When a photon hits the photodetector, a short TTL pulse is generated from the photodetector and we need to count the number of pulses coming from the detector. This can be done with a Data Acquisition card (DAQ card), which has a built in counter.

### **4.2 SC 2000 Digital Scan Controller**

#### **4.2.1 System Overview**

The Scan Controller is a small intelligent system that is used to control GSI Lumonics SAX family of scanner servo amplifiers and associated equipment.

It can work in either a stand-alone configuration or it can work together with a host computer. Below are the basic components of a complete system:

1. One or two axes of SAX/Galvo micropositioning
2. One SC2000 Scan Controller
3. Cabling
4. Power Supply
5. A host computer (for setup and optionally for operation), including software
6. Other system components interfaced to the Scan Controller

The Scan Controller can be programmed either with GSI Lumonics supplied LabView interface or customer-designed host-side software.

By interfacing directly to the position command, position feedback and binary communication of the axis servos, the Scan Controller has full position control of the system and position feedback. It allows also the host computer to read galvo temperature status and calibration information.

Besides, the system has also interlock which means a device used to help prevent a machine from damaging itself by stopping it when something unusual happens.

Figure 1 and Figure 2 on next page show overview of the system.

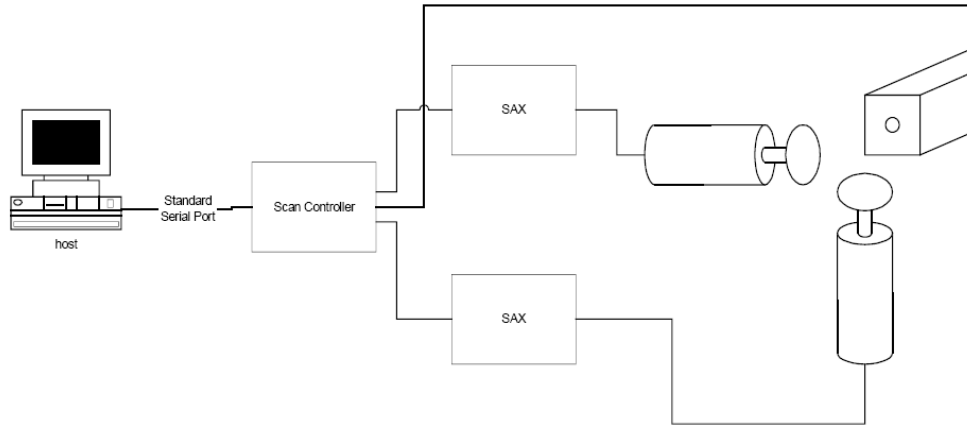


Figure 1 Scan Controller with host computer

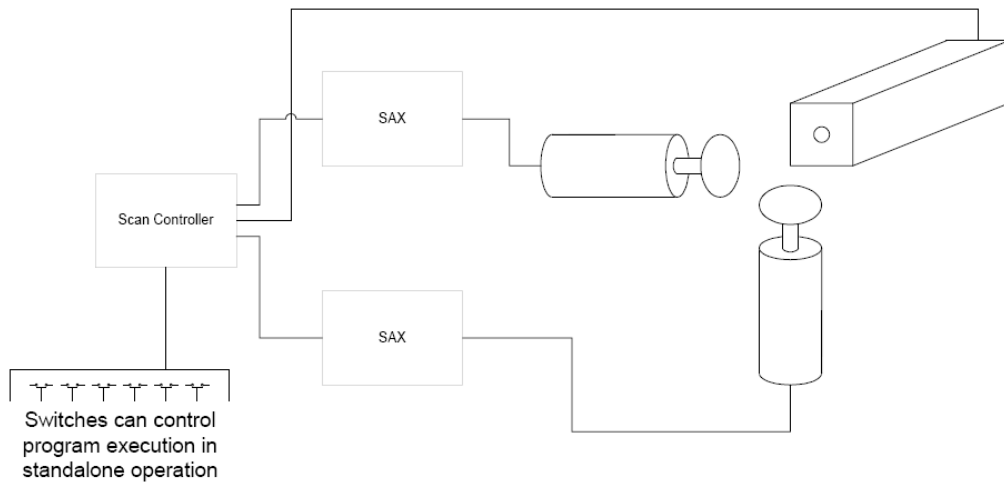


Figure 2 Typical standalone system

The Sync/Cal connector provides access to the synchronization/calibration I/O as well as the pixel clock output of the Scan Controller. Figure 3 on next page shows Scan Controller layout, pin out and connector part numbers. In this project only “sync 1 (out)” and “sync 2 (out)” will be needed to communicate with DAQ card.

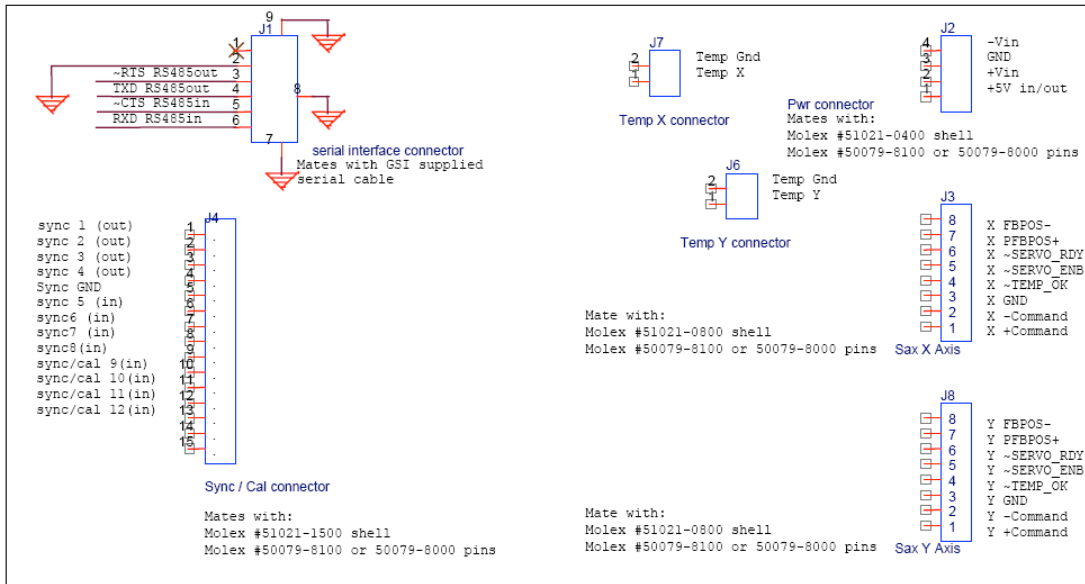


Figure 3

## 4.2.2 Command Line Interface

The Command Line Interface software is provided together with Scan Controller, the software was programmed in LabView environment. It allows users to communicate with the Scan Controller quickly and easily. It is used primarily as an environment for designing and programming as stand-alone application, and for system evaluation. Command Line Interface is easy to use for understanding of functionality and capabilities of Scan Controller.

Once the functionality and limitations of the Scan Controller are understood, it is much easier for the user to do the programming. Figure 4 shows Command Line Interface software.

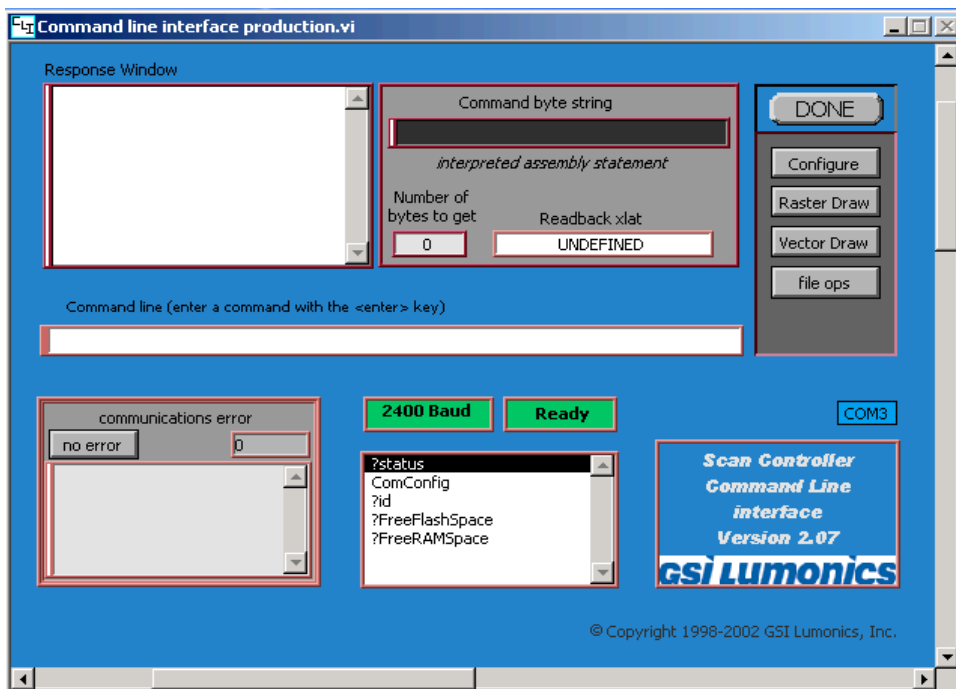


Figure 4

### 4.2.3 Assemble language

With Command Line Interface the user has two different ways to write a motion program for the SC 2000, this Scan Controller accepts either a motion program language that consists of a binary machine language or assembler language. It is very easy for the programmer to use assembler language because it consist of English language commands for controlling axis motion, it requires only that the programmer has basic English knowledge.

If the programmer chooses assembler language then the programs can be typed into a text file with a text editor such as WordPad. There are two basic operating modes in the Scan Controller. Vector mode is used to control X and Y axis galvos to move both X and Y direction. Raster mode is used to control a selected single axis. Vector commands can differ from raster commands by writing a 'xy' suffix. For example, 'SlewXY' is a vector command and 'Slew' is a raster command.

## 4.3 Data Acquisition

Data acquisition means collecting signals from measured source and process the signal and present on a PC. Before it was common that measurements was done on stand alone instruments like multi meters, oscilloscopes, counters etc but nowadays it have become more important to gather measured data and present it on a PC. A Data Acquisition Card is used to measure the signal and transfer data to the computer. A DAQ card normaly contains both ADC and DAC, which mean the DAQ card is able to have analogue/digital as input as well as output channels.

There are five components to be considered when building a basic DAQ system (Figure5):

- Transducers and sensors
- Signals
- Signal conditioning
- DAQ hardware
- Driver and application software

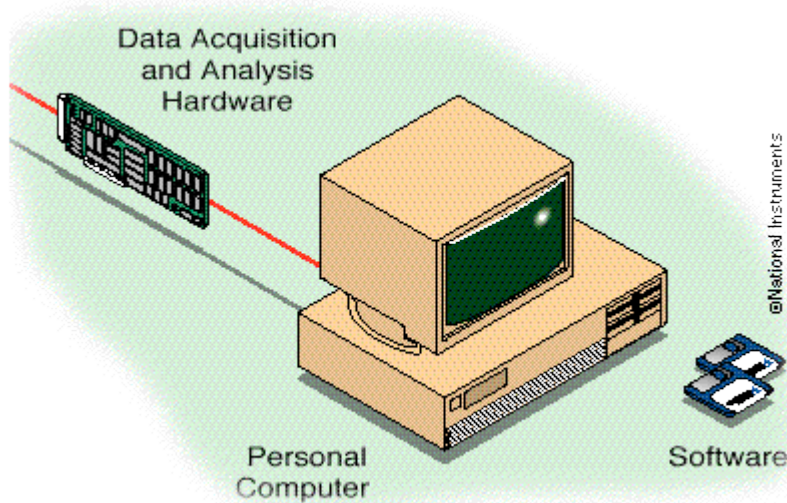


Figure 5. Data Acquisition System

In this project a photon detector is connected to a DAQ card, this photon detector generates digital pulses and sends it directly to the DAQ card and therefore no other external transducers and signal conditioning is needed.

### 4.3.1 Digital Signals

Unlike an analogue signal, which is continuous, and can take on any value with respect of time, a digital signal is a discrete signal and has two possible levels: high and low. A TTL (transistor to transistor) pulse defines the signal is low when it is between 0V to 0.8V and high when 2V to 5V. The useful information that you can measure from a digital signal is the state and the rate. (Figure 6).

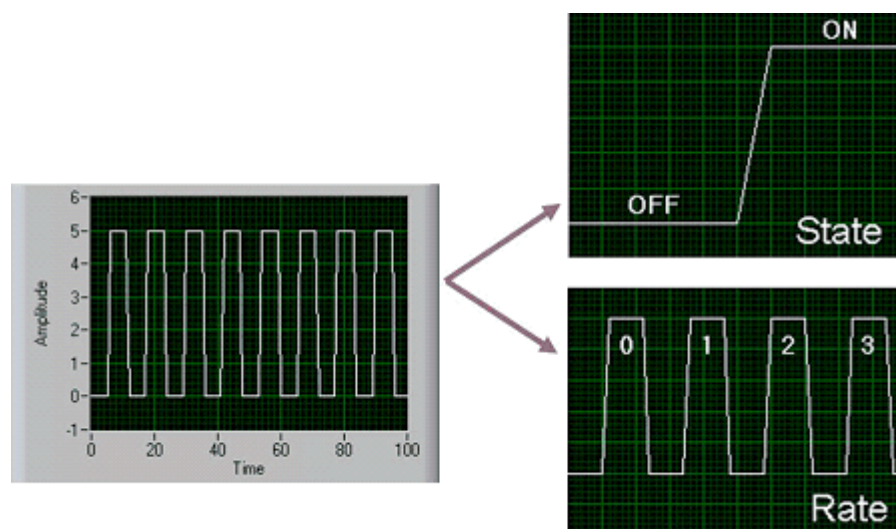


Figure 6. Primary Characteristics of a Digital Signal

#### *State*

The state of a digital signal is simply the level of the signal - on or off, high or low. Monitoring the state of a switch - open or closed - is the most common way to use digital signals.

#### *Rate*

The rate of digital signal measures how large portion of a signal occurs with respect to time, for example an MP3 file that is created has a bit rate setting of 128 kbit/s.

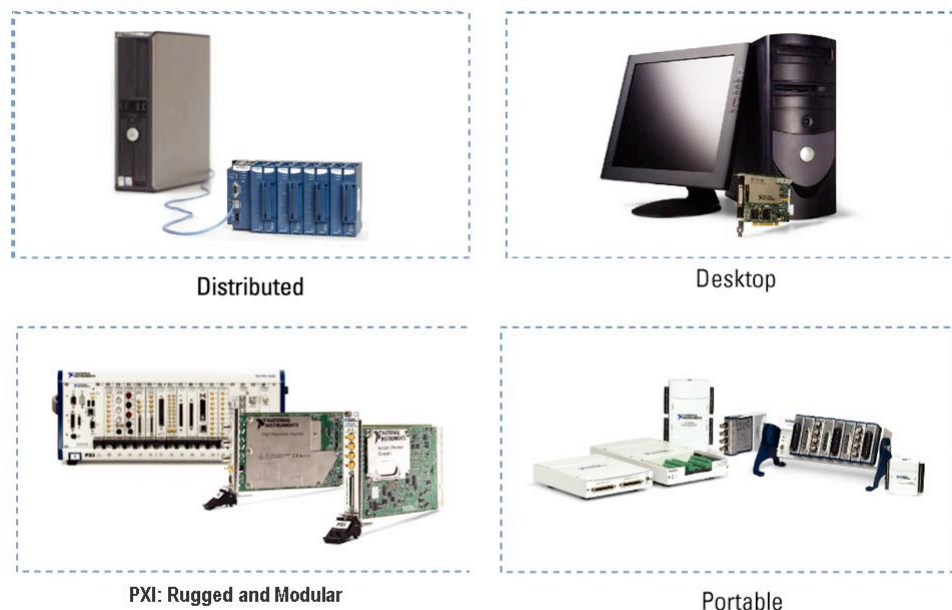
### 4.3.2 DAQ Hardware

DAQ hardware functions as the link between the computer and sensor, detector etc. It normally use as a device that converts incoming analogue signals to digital signal so that the computer can interpret them but there are other functions available:

- Analogue Input/Output
- Digital Input/Output
- Counter/Timers
- Multifunction- a combination of analogue, digital, and counter operations on a single device

National Instruments offers several hardware platforms for data acquisition. The most common available platform is the desktop computer. National Instruments offers several type of DAQ hardware for different purposes:

- \* PCI DAQ boards that plug into any desktop computer.
- \* PXI/CompactPCI is a more robust modular computer platform specifically for measurement and automation applications. PXI is ideal for mid- to large-size applications.
- \* Compact FieldPoint is designed for industrial control, the NI Compact FieldPoint programmable automation controller (PAC) offers the flexibility and ease of a PC and the reliability of a programmable logic controller (PLC).
- \* NI has also DAQ devices for portable or handheld measurements, the DAQ devices are connected via USB to a laptops or PocketPC PDAs (Figure 7).



**Figure 7. DAQ Hardware Options from NI**

In this project National Instruments PCI 6602 will be used.

### **4.3.3 Driver and Application Software**

#### **Driver Software**

Driver Software that usually comes with the DAQ hardware, it allows the operating system to recognize the DAQ hardware. Without software to control or drive the hardware, the DAQ device will not work properly. A good driver offers high and low level access.

## **Application Software**

The application layer can be an environment that you build an application that meet your own criteria or it can be a configuration-based program with preset functionality. Depend on your application's complexity you can choose an application software that is appropriate for the purpose.

NI has three development environment software that make it possible to design complete system of instrumentation, acquisition, and control applications, as mention before only LabView will be used in this project:

- LabVIEW with graphical programming methodology
- LabWindows™/CVI™ for traditional C programmers
- Measurement Studio for Visual Basic, C++, and .NET

With the introduction of NI SignalExpress, it is possible to use configuration-based software to build up a program and the user does not need to do the programming. The disadvantage is that the program become slower.

## 5. Implementation

There are three different types of command programming software available for SC2000:

- Command Line Interface: Learning Environment for understanding SC 2000 commands. Use for development of stand alone application.
- Library of LabView VIs: Contains library of command blocks which can be incorporated into the users LabView program.
- VisualBasic Assembler: Convert English Language commands to machine code binary on a command or file basis. Perform error checking and returns flags.

In this project LabView will not be used to control SC 2000 because the manufacture did not provide any example or documentation, instead Command Line Interface will be used as a stand alone application. LabView will only be used to generate the assembler code for controlling the mirrors. There are 4 inputs and 4 outputs on the SC 2000 board, which can be used to synchronize with the DAQ card.

In order to complete the whole operation from scanning to analysing the data it must be divided in four different programs and each program contains several parts that together meet the specification for that program. All of the programs are implemented in LabView even though the program, which controls SC 2000 is in assembler.

1. Scanprogram generation
2. BufferedCounter
3. BufferedCounter at 1 spot
4. Read binary file

The first program is for generating a scanning program (a text file) and this scanning program will be downloaded in SC 2000. The second program is the main program, which counts the number of pulses and plots it in an *Intensity Graph* (image). The third program has the same function as second program but the difference is this is not a page scan, the laser stay at one specific point and the program will count the number of pulses and plot it in a *Waveform Graph* (intensity over time  $I(t)$ ). The last program is for analysing data.



## 5.1 Scanprogram generation

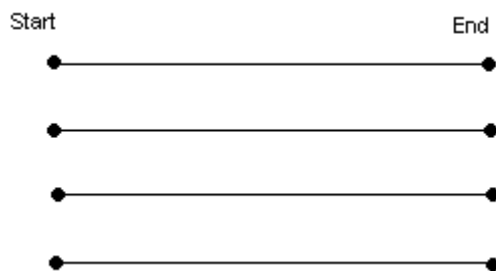
### Assemble programming

The scan program is divided into two assembler programs, the first for setting the SC 2000 into the default state (start position, sync channel, etc). The second program is for performing page scan which will be explained in next section. These programs do not run separately, the first program functions as a main program and calls the second program which functions as a sub program just like in Pascal or C.

Since these programs will be saved as a text file then they can be implemented in Labview, the idea is to create a whole scanning program in string constants and then use "*Search and Replace String*" VI to replace these constants with desired parameters.

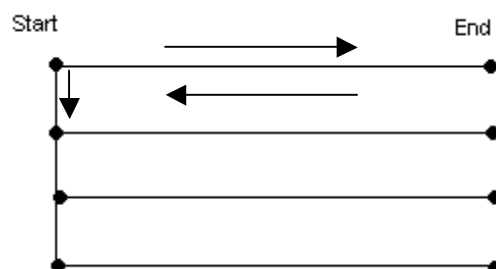
### Page scan

The objective of this project is to scan over an area and it is natural to scan this area as figure below show.



The laser beam moves from start point to end point and in this time the SC 2000 board will turn on sync1 and gives the DAQ card permission to count.

Since the laser shuttle is not available and an extern laser is used so it cannot be turned off, the laser will be switched on all the time. The scanning will look like this instead.



The laser beam moves from the start point to the end point and then backs to start point again and finally moves to the next line. During the time when the laser beam moves back from end point to start point and to the next line, the DAQ card shall not count. The whole operation continues until it reaches the number of desired lines.

An assembler program is a simple text file and can be downloaded to SC 2000.

In order to control the mirrors (laser) movement there are two parameters to be specified, the coordinates and the time. The coordinates have a range between  $-32000$  and  $+32000$ , the time have the ranges  $-32000$  ticks and  $+32000$  ticks and 1 tick corresponds approximately  $23 \mu\text{sec}$ . A program or a command will not work if these coordinates or time exceeds their maximal values. For simplicity  $30.000$  will be considered as the maximum.

In this project the main task is to count the number of pulses during a specific coordinate and time interval, for example the full range of coordinates is  $5000$  and the time interval is  $100$  ticks. Note here that the coordinates and the time are independent of each other and the coordinate is not a problem because the interval never exceeds the limit since only small samples will be scanned, only time is an issue here. When a scanning program is created the user must specify how many time interval (let us call it  $\tau$ ) will be on a line, for example  $\tau = 100$  ticks and it will be  $100$  of  $\tau$  per a line, then the total time is  $100 \times 100 = 10.000$  ticks and it is below  $30.000$  ticks. So far so good.

The problem is when the user want to have  $400 \tau$ ,  $500 \tau$  or  $600 \tau$ , the total time will be in example  $100 \times 400 = 40.000$  ticks and it exceed the limit of  $30.000$  ticks. The technique is to split the scanning into 2 parts, look at the picture below.



From "Start" to "Stop" is  $20.000$  ticks and it is less than  $30.000$  ticks so it's ok, after the laser reached "Stop" it will immediately go to "End" point and it will take  $20.000$  ticks. By doing this way it will not be any conflict with the limitation of SC 2000. This technique will work up to  $60.000$  ticks, if the user wants to scan with a total time of  $70.000$  ticks then it will not work and the user must change some parameter because  $70.000/2 = 35.000$  which exceed the limit,  $70.000/3 = 23333,3$  and SC 2000 cannot handle decimal numbers.

## 5.2 Buffered Counter

This program is the biggest and the most important part of this project, it contains several modules which together will perform the following steps:

- 1) Estimate how much time it will take to perform a page scan.
- 2) Calculate the time that has elapsed for a page scan. It will show the time when scanning is finished.
- 3) Count number of pulses from the detector during a specific interval and plot number of pulses on an "Intensity Graph" (image). Each dot on the "Intensity Graph" corresponds to a spot on the sample.
- 4) Generate a command which moves the laser to a particular point on the sample that the user wants to examine closer. This command is called "DeltaSlewXY" and is a string in Labview which the user can copy and paste in the Command Line Interface.
- 5) Save the result as a binary file.
- 6) Change the intensity with cursors and see how the intensity is changed along the x-axis and y-axis.

## Implementation of these tasks in Labview

1) To estimate how much time it will take for a page scan it is not enough to sum all the time in the assembler program, other time parameters must be considered, for example a delay caused by software commands or delay from the galvo response.

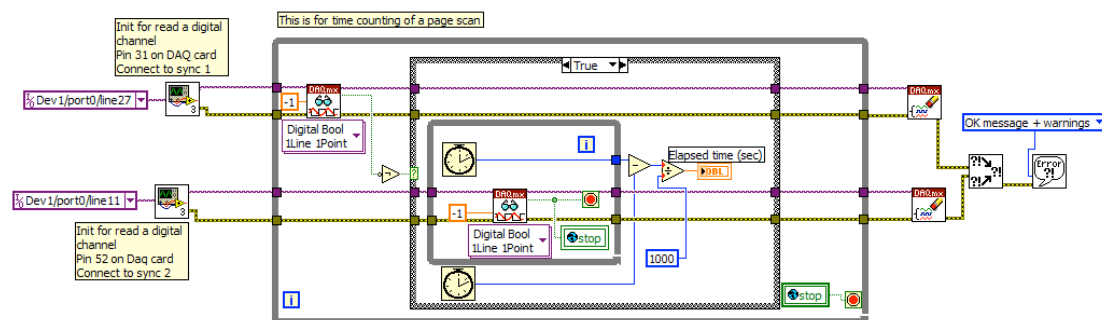
The delay from the software command is 315 $\mu$ sec and the delay from the galvo response is 250 $\mu$ sec.

The table below shows the estimated values and the simulating value for 100x100, 200x200 and 300x300 points, respectively.

| Number of points | Theoretical values | Simulated value |
|------------------|--------------------|-----------------|
| 100x100          | 30 sec             | 30 sec          |
| 200x200          | 107 sec            | 106 sec         |
| 300x300          | 230 sec            | 229 sec         |

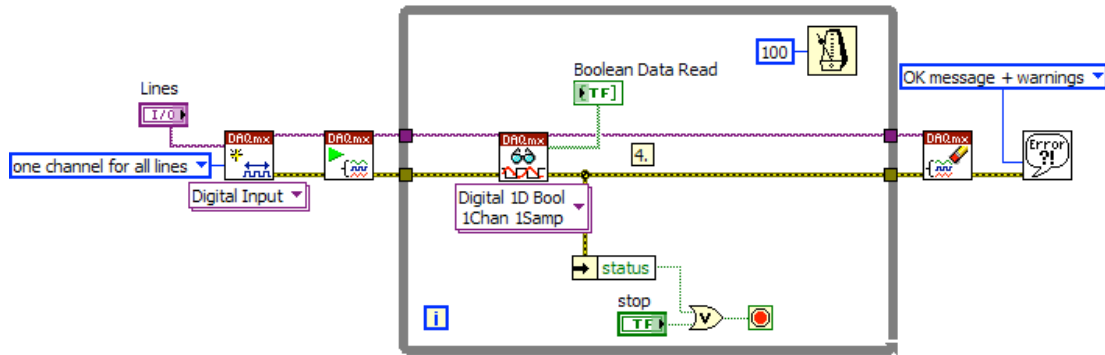
The theoretical values are values that were calculated with respect to the delays and laser movement before scanning. The simulated values are the time from the beginning to the end of scanning.

2) When the scanning is complete it is interesting to know how much time has elapsed. The implementation of this task is showed in the figure below.



The idea is use two "Read Digital Channel" modules to start and stop the timer. These modules read digital signals from sync 1 and sync 2. When the scanning begins, sync1 turns to high and the *case loop* executes, the clock outside the *while loop* is fixed and the clock inside the *while loop* counts until the scanning is finished which means sync 2 turns to high. After the *while loop* has executed, the program subtracts the inner clock with the outer clock to obtain the time difference. To get the time in seconds it is necessary to divide by 1000 because the values from the clocks are in milliseconds. A "global Stop" VI is used to stop the other *while loop* in counting pulses module so they can be synchronized with each other.

The principle of the program "Read Digital Channel" is shown in the figure on next page.



This program reads a digital signal from a line(s) and returns the value in boolean format. Note that all "Read Digital Channel" has been made as a subVI to save space.

Note that when a "Start Task" VI (after "Digital Input") is used, a "Clear Task" VI must also be used to stop the task if necessary, and to release any resources the task reserved. In this case "Clear Task" VI is placed after the while loop.

3) This part of *Buffered Counter* is constructing a program which counts the number of pulses in a specific time interval and plots it in an *Intensity Graph* so that every point (x,y) on the sample corresponds to a dot on the *Intensity Graph* (image). The image is the intensity over coordinates I(x,y).

First several important ideas will be explained so the reader can understand why the program was implemented in this way.

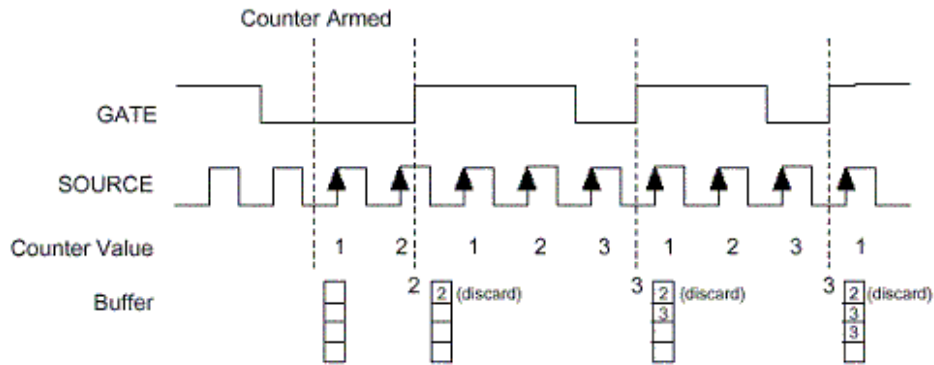
**Property Node** gets reads and/or sets write properties of a reference. Below are some examples of Property Nodes:

The *DAQmx System Property Node* can return the available devices, tasks, or channels on the current computer.

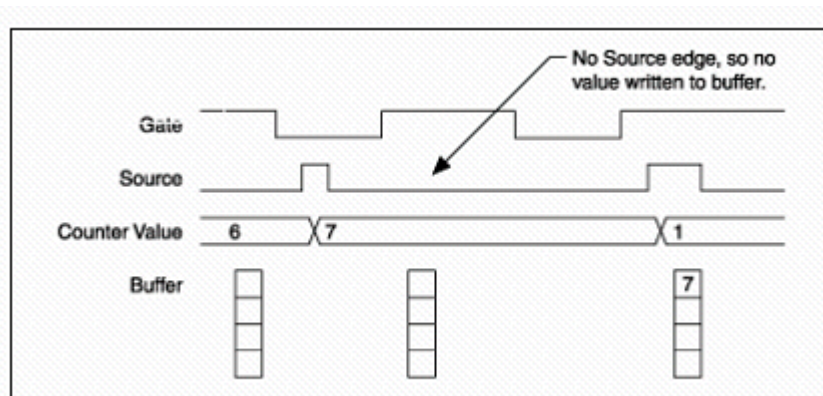
The *DAQmx Device Property Node* can return device settings like bus type or serial number.

The *DAQmx Channel Property Node* can be used to read or writes properties of a channel.

**Buffered Period Measurement** is a technique that measures the number of periods in a specific time interval between rising or falling edges of a pulse. Each time the GATE input becomes high, the counter begins to counts number of periods on the SOURCE input and the values are inserted in the buffer. The first result of the measurement is not an accurate representation because it is a measure of the number of edges of the SOURCE signal between the beginning of the measurement and the first rising edge of the GATE. Therefore, the first value is unreliable and should be discarded. The figure below shows a three period measurement but only the last two are correct, the first value can be removed easily in the LabView program.



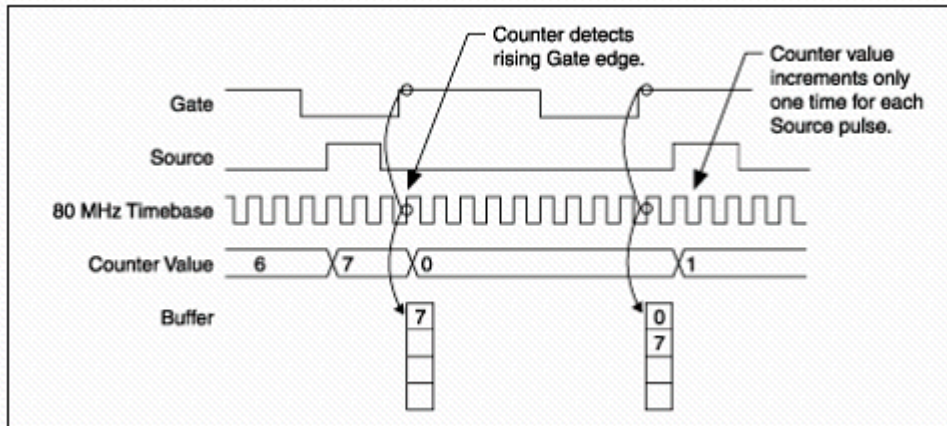
**Duplicate Count.** When a Source signal is slow or non periodic and there is not a single pulse during a period of the Gate, then no value will be inserted in the buffer. This situation is duplicate counting. This kind of behaviour is not desirable (in this project) because the counter is expected to count the number of pulses coming from the Source in each period of Gate no matter how many pulses there are, even there are no pulses. In other word if there are no pulses, the buffer should register number zero.



*Duplicate Count*

### **Duplicate Count Prevention**

Duplicate Count Prevention is used in buffered counter applications like measuring frequency or period. It is necessary to use when the application has slow or non-periodic signals, as it ensures the right values are inserted in the buffer. Normally the counter starts to count when the Gate is high and if a pulse coming from the Source, if there is no pulse on the Source then it will not count. With duplicate count prevention enabled, the counter synchronizes both Source and Gate signals to the maximum (80MHz) internal (DAQ card) timebase. By doing this, the counter detects edges on the Gate even if there is no pulse on the Source and the count value changes synchronously with the timebase.

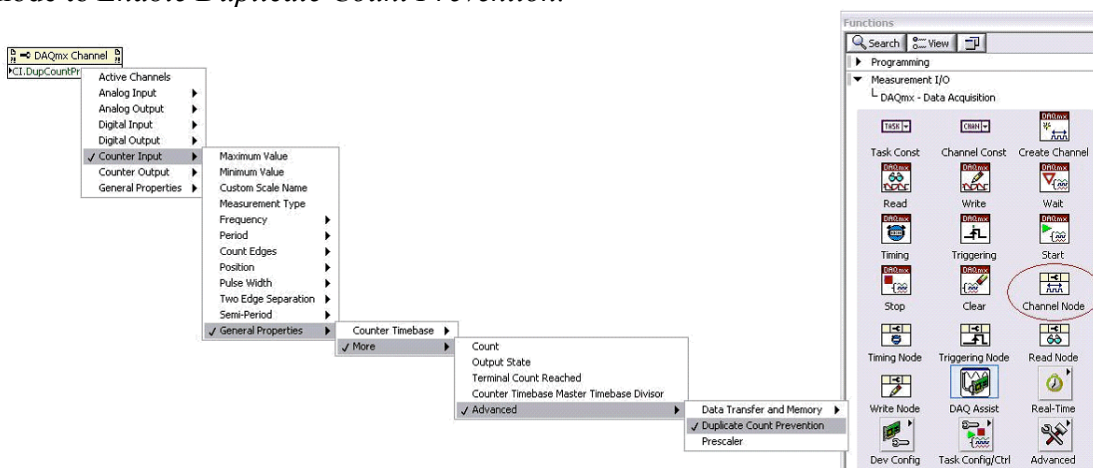


The first edge of timebase is used to increase the counter if there are pulses on the Source and all other edges of timebase are ignored. The picture above shows the synchronization between Gate, Source and timebase. The counter arms when the Gate becoming high and start counting when an edge of timebase is raised. If no pulses are detected on the Source during a period the counter will register the value 0. For NI-STC II and NI-TIO boards, using duplicate count prevention reduces the maximum frequency of the Source to quarter of the maximum timebase frequency (80MHz). Duplicate count prevention should only be used if the frequency of the Source signal is 20 MHz or less.

Duplicate count prevention should only be used in the following situations:

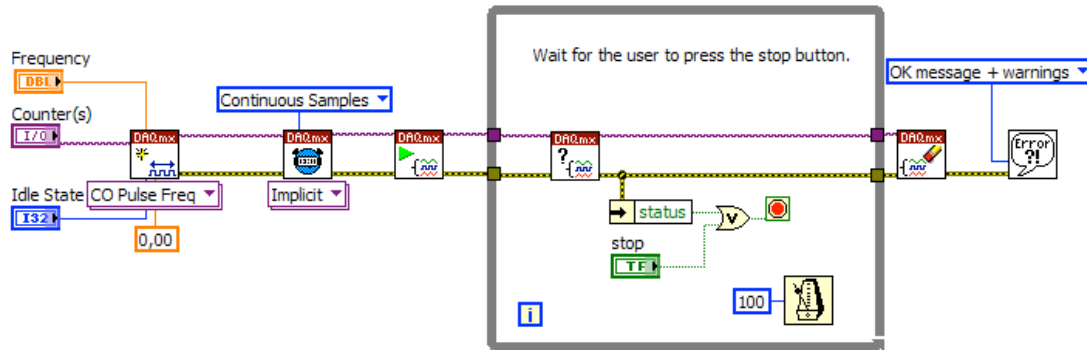
- Counter measurements
- The counter Source is using an external signal (in this project the TTL pulses from photodetector).
- The frequency of the external source is 20 MHz or less

To enable duplicate count prevention in NI-DAQmx, you can set the channel property node to *Enable Duplicate Count Prevention*.

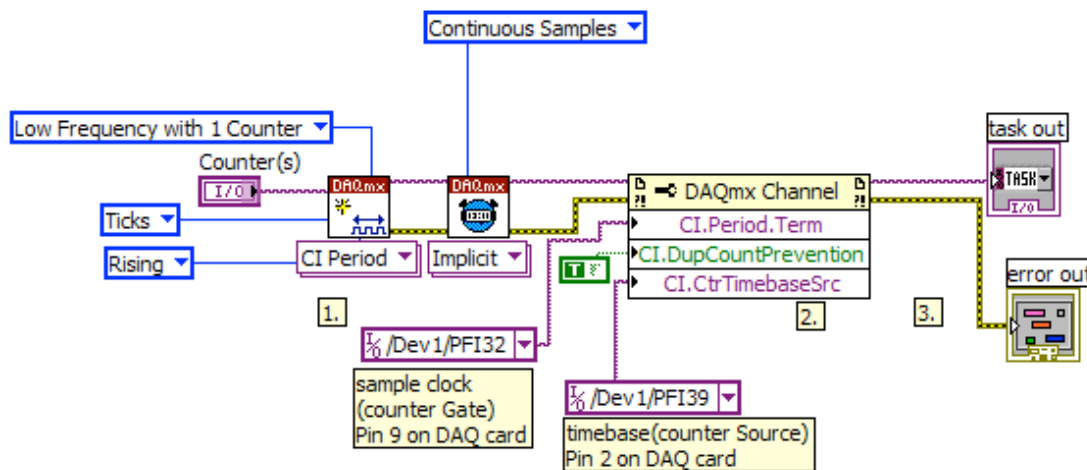


Place a *DAQmx Channel Property Node* and select *Counter Input->General Properties->More->Advanced->Duplicate Count Prevention* as shown in Figure above.

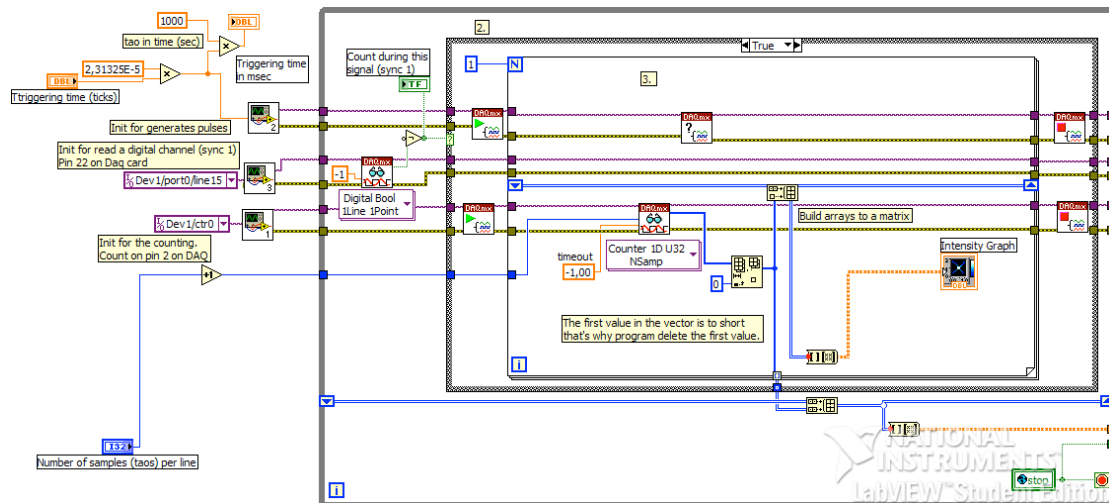
When implementing this counter task, the GATE will be software timed which means a pulse train will be sent to the GATE and the counter will count the number of rising edges that occur on SOURCE between each pair of active edges of GATE. Figure below shows the principle of how to send a pulse train to a specific address.



The figure below shows how the initial state of the counter has implemented as a subVI. It contains the necessary parts like Duplicate Count Prevention, and addresses of Source and Gate.



Now we have all the parts to make this counter program to work. The figure below shows how the program was made.

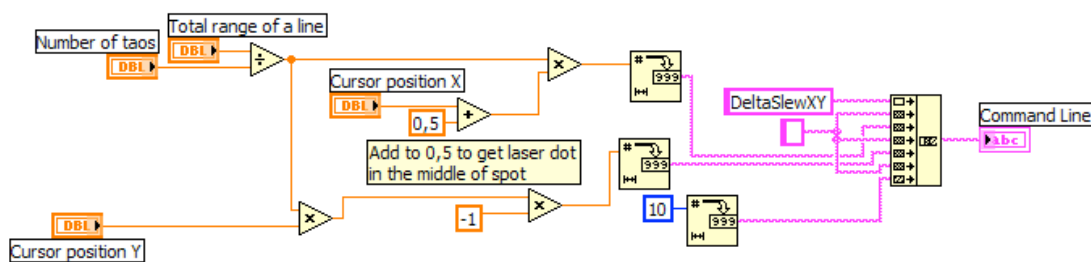


This is the main part of the Buffered Counter program.

When sync 1 turns to high, the case loop and the *for loop* will execute and the *counter VI* counts the number of pulses coming from the Source. It returns all the values as an array, but since the first value is not correct it must be deleted from the array.

In order to plot on an *Intensity Graph* all the arrays must be converted to a matrix. It can be done by using a *Build Array VI* and a *Shift Register*. Together they build arrays to a 2-D array. When 2-D array is received it is connected to an *Array To Matrix* to convert it into a matrix and then it is ready to be plot on the *Intensity Graph*.

4) After a page scan sometimes the user wants to examine a specific spot on the sample. In order to do that the laser must move to this point and stay there. The task is to generate a string (text) so the user can copy and paste it in the Command Line Interface. The figure below shows how it has been implemented as a subVI.



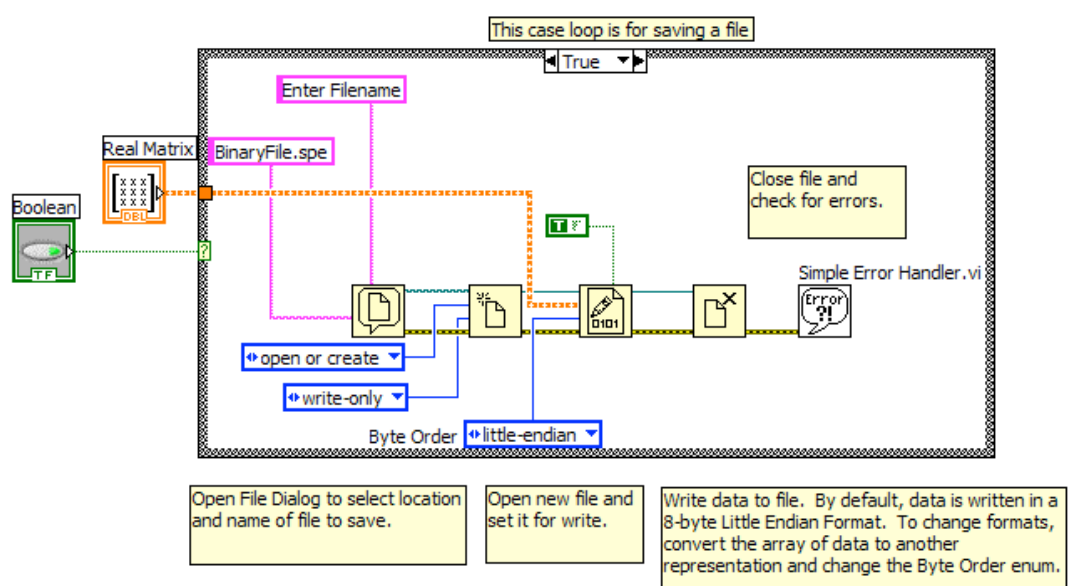
On the *Intensity Graph* there are two cursors and one of them is used for this purpose. When a cursor moves on the *Intensity Graph*, a *Property Node* will return the position of X and position of Y (from 0 to number of  $\tau-1$ ). Let us say the total range of a sample is 30.000 and there are 100 intervals on the sample, deltaX and deltaY will be  $30000/100=3000$  which means every interval is 3000.

To get the position on x-axis and y-axis the following formula is used:  $x = (\text{Cursor X} + 0.5) * \text{deltaX}$  and  $y = -(\text{Cursor Y} * \text{deltaY})$ .

After the positions are received it will be converted to string with *Number to Decimal String VI* and then added with other string constants to build up a command string. This is done with *Concatenate Strings VI*.



5) After scanning it is important to save the data so it can be analysed later. This part of the program saves the result as a binary file. The figure below shows how it was implemented.



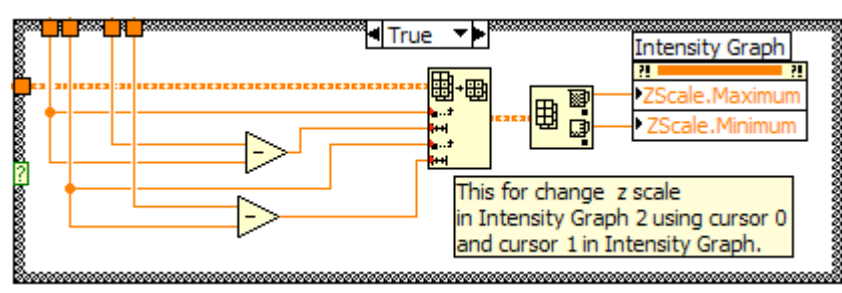
This part was implemented as subVI.

The case loop is needed to make sure the program saves the data after scanning and not during the scanning. This part is quite standard procedure and can be found in Labview examples.

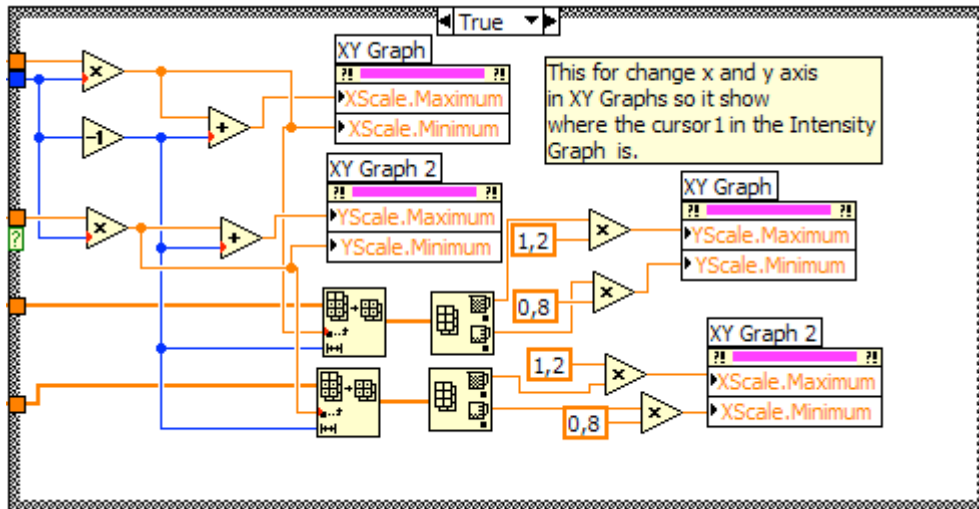
First use a *File Dialog VI* to select location and name of file to save and then use *Open/Creative/Replace File VI* to open new file and set it for write. Use *Write to Binary File* to save the data and then close file.

6) This part of the program is for changing the intensity with respect to a specific region and see how the intensity changes along the x-axis and y-axis. There are two cursors used to frame the specific area and with one click on a button the *Intensity Graph* will change the intensity scale with respect to this region.

Next figure shows how to change the intensity, it takes x-y position from cursor 1 and subtract with x-y position from cursor 0 which gives an area that both cursors have frame in. Then take the maximum and minimum value in this area and use a property node of the Intensity Graph to rescale it.

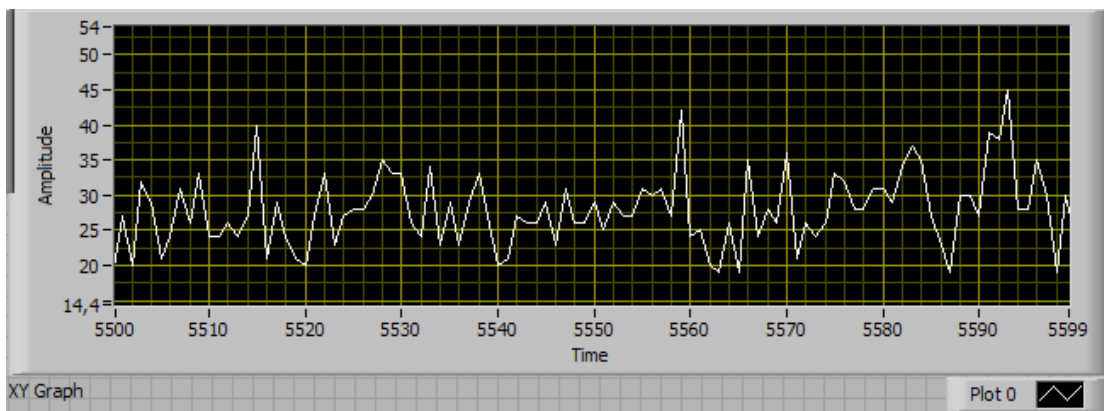


The last part of *Buffered Counter* program is to let the user to move one of the cursors and mark a spot and see how the intensity changes on the x-axis and y-axis which cross this point. This means the graphs will show the values of 1D-array with number of  $\tau$  elements, for example number of  $\tau$  can be 100, 200 or 300 and so on. Figure below shows how it was implemented.



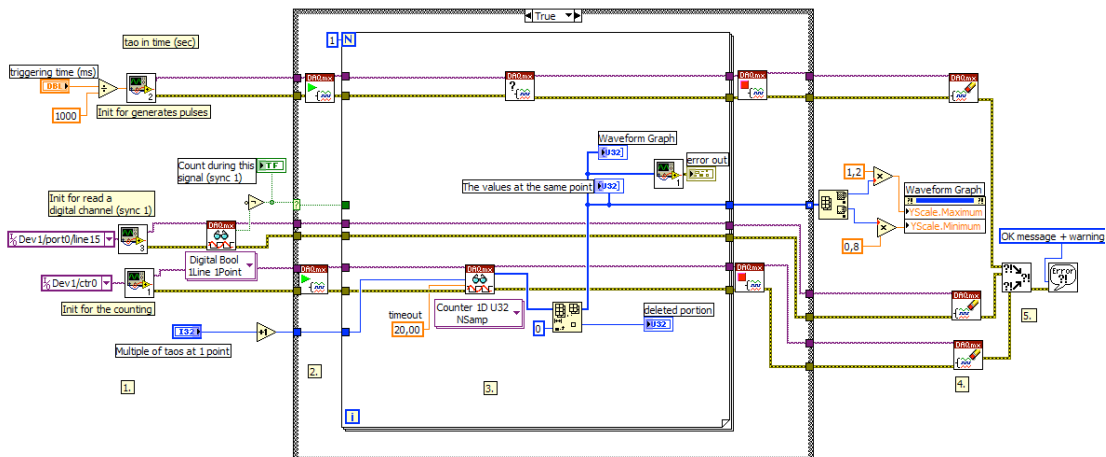
The idea is first to convert the matrix to a 1D-array which contains all the values, for example  $100 \times 100 = 10.000$  and this array has index from 0 to 9999, if it is  $200 \times 200 = 40.000$  then the array has index from 0 to 39.999. Depending where the cursor is, this part of the program will take only the values which the cursor lays on the *Intensity Graph* currently and cut of all other values.

The intensity usually has a large dynamic range so it is necessary to make it visible on the graph, the solution is to let the y-axis be 20% larger than the largest value and 20% smaller than smallest value.



### 5.3 Buffered Counter at 1 spot

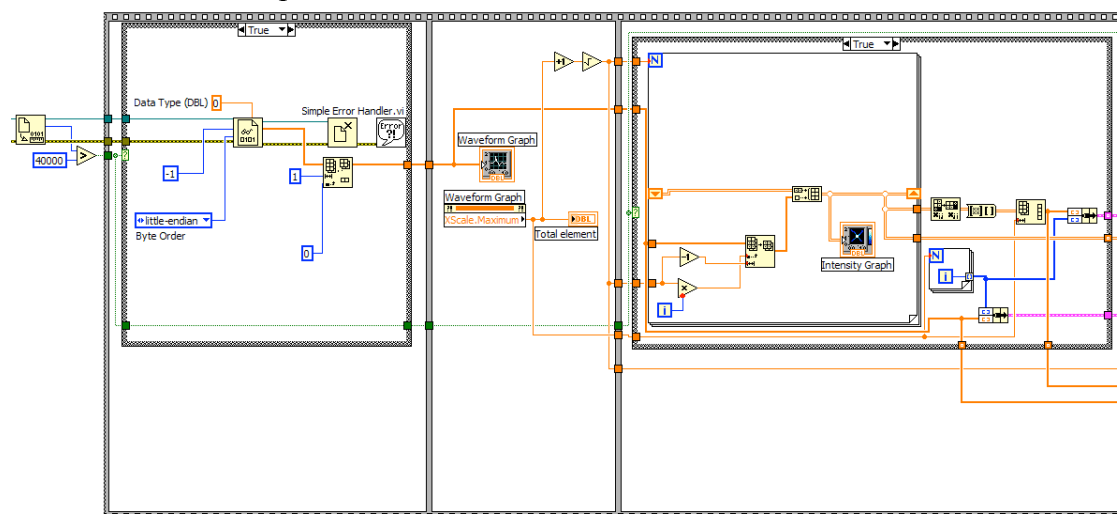
When the user examines the image there ought to be one spot which the user wants to examine closer. After the laser has moved to this point with *DeltaSlewXY* command the user can run this program to see how the intensity is changed during a time interval. This will be intensity over time  $I(t)$ . Figure below shows how the program is implemented.



This program is very similar to *Buffered Counter* with small differences, it needs only to count the number of pulses during a time interval and plot the result on a Waveform Graph. Note that the result is a 1D-array.

### 5.4 Read Binary File

All the counter programs save the results as a binary file so it can be analysed later with another software. However, it is better to be able at least to see saved images with LabView. This program is for that purpose, the front panel is almost identical with the *Buffered Counter* program. For example when this program runs to analyse a file which contains a matrix, all the graphs should look like the *Buffered Counters* graphs right after this matrix is received from a scanning. The front panel looks like the front panel of *Buffered Counter* but the implementation is different, figure below shows how it was implemented.



This picture is not complete, this show only the most important part of this program. The other part is the same as 6 in *Buffered Counter*.

When a file is saved in binary format Labview does not contain any information if it is a matrix or not, the values are considered as a 1D-array. One way to get around this problem is to use a *Get File Size VI* to determine which type it is. If there are 100 values in 1D-array then the size is  $100 \times 8 = 800$  bytes, if the matrix is  $100 \times 100$  then the size is  $10.000 \times 8 = 80.000$  bytes. The conclusion is the matrix size is much larger than 1D-array. In case that the file is a matrix and is represented as 1D-array then a *For loop*, *Shift Register* and *Build Array* will be used to convert this array to matrix and then plot it on *Intensity Graph* as it did *Buffered Counter*.  
If the file is a 1D-array then it is easy to plot on a *XY-graph*.

## 6. Results and Discussion

The result was functioning programs, which meets all the requirements, the simulations gave expected behaviours and values. The hardest thing was synchronizing the scanning and the counting because the number of time intervals on a line could exceed the maximum time allowed in SC2000, but fortunately this problem could be solved in software. After running *Buffered Counter* and a matrix was saved, then the *Read Binary File* program was opened to read that matrix, the pictures from the *Intensity Graph* and *XY-Graph* from both programs look identical which means the programs work as they should.

Since the program for controlling the mirrors was implemented in assembler code and the rest in LabView, the use of programs for entire operation from scanning to analysing data was a somewhat lengthy procedure because these programs are used separately.

After this project I have learned much more about Labview since many problems came up when I implemented these programs. I could either troubleshoot by myself or ask Tomas for assistance but it is a good idea to ask application engineers from National Instruments on their discussion forum, they are very professional and could answer most questions.

Concerning improvement of the implementation, much has already been done. But I think there are room for improvement because *The Buffered Counter* and the *Read Binary File* have block diagrams which are larger than the screen and it is maybe not desirable, parts of program can be implemented as subVI to save space. One more thing is if someone can figure out how to implement the scanning program in LabView then it would make it easier for the user to use the programs since they are more automated and better synchronized with each other.

## 7. References

SC 2000 User Manual

<http://en.wikipedia.org/wiki/Fluorescence>

[http://mesh.kib.ki.se/swemesh/show.swemeshtree.cfm?Mesh\\_No=H01.671.606.552.665&tool=karolinska](http://mesh.kib.ki.se/swemesh/show.swemeshtree.cfm?Mesh_No=H01.671.606.552.665&tool=karolinska)

<http://en.wikipedia.org/wiki/Labview>

<http://zone.ni.com/devzone/cda/tut/p/id/3536>

From NI discussion forum:

<http://forums.ni.com/ni/board/message?board.id=170&message.id=314546>

<http://forums.ni.com/ni/board/message?board.id=170&thread.id=312369>

<http://forums.ni.com/ni/board/message?board.id=250&message.id=34548>