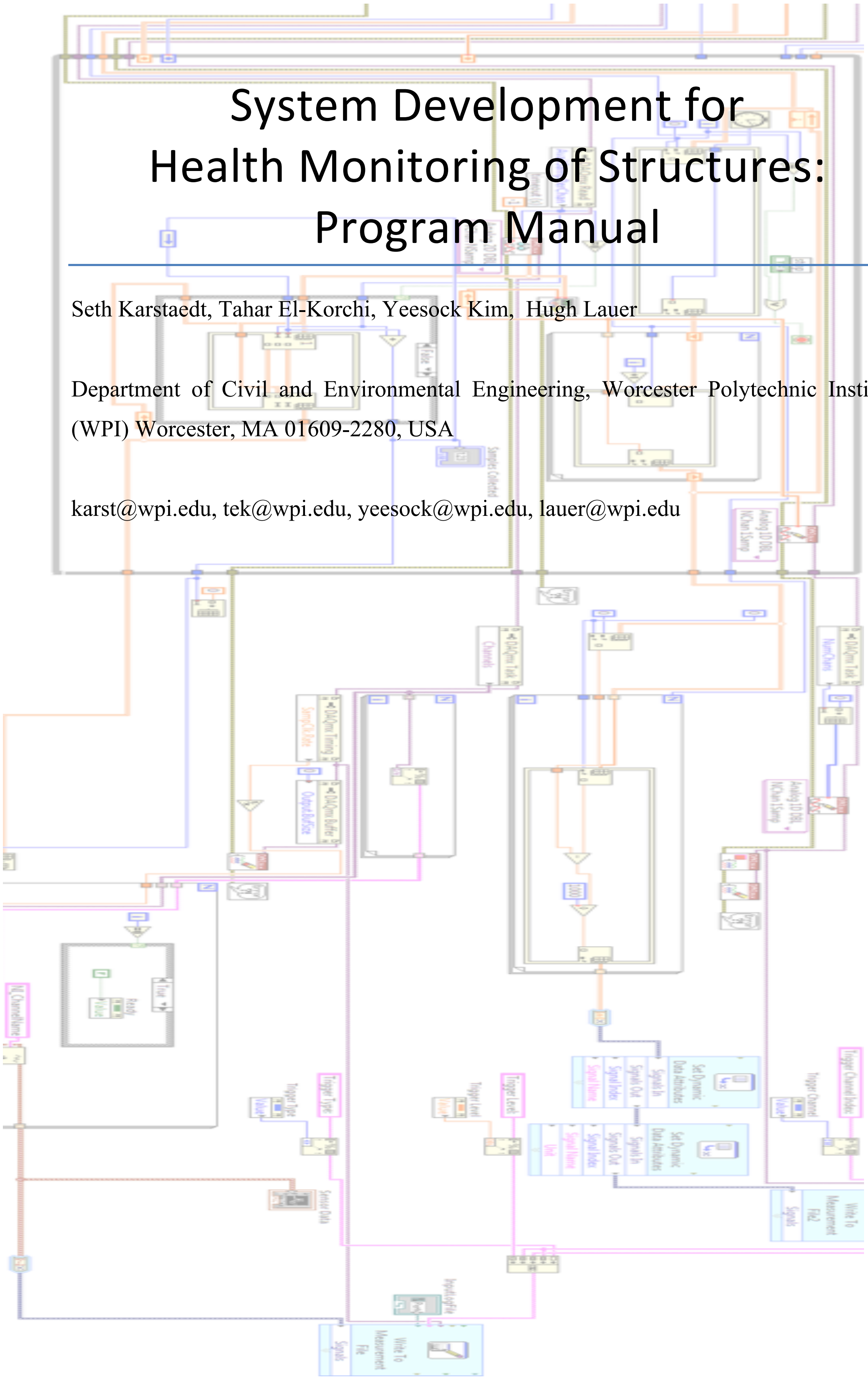


System Development for Health Monitoring of Structures: Program Manual

Seth Karstaedt, Tahar El-Korchi, Yeesock Kim, Hugh Lauer

Department of Civil and Environmental Engineering, Worcester Polytechnic Institute (WPI) Worcester, MA 01609-2280, USA

karst@wpi.edu, tek@wpi.edu, yeesock@wpi.edu, lauer@wpi.edu

Instructions for Operating the Program:

This program was designed for use with National Instruments (NI) DAQ modules. NI LabView, DaqMX, and NI Measurement and Automation Explorer (MAX) are required for using this program.
Sensors and Output are set up through the NI-MAX interface, for additional info on using this info please refer to:
DAQ Getting Started Guide: http://www.ni.com/pdf/manuals/373737f.pdf
DAQmx Support Overview: http://sine.ni.com/psp/app/doc/p/id/psp-268
Complete Listing of DAQmx for Windows Documentation:
http://digital.ni.com/public.nsf/allkb/CCDFC93878BD8781862570FC00559980

This program can be used for collecting data through NI-DAQmx sensors and controlling output to NI output modules in real time.

Using This Program:

The Testing Workflow:

Every test follows this basic workflow.

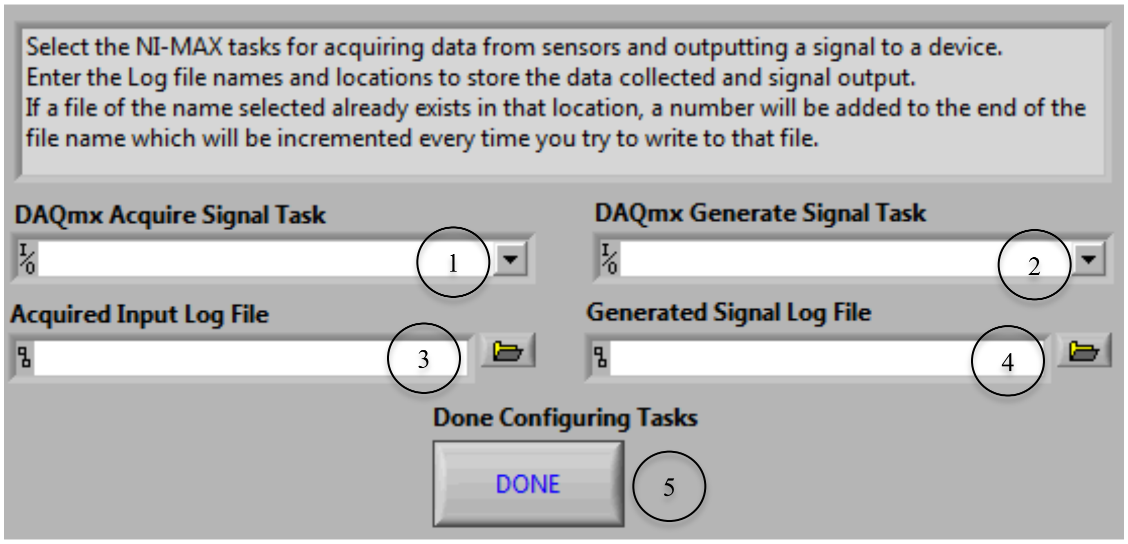
1. Select or create the tasks that contain the channel information for your sensors and output. (Tasks are NI-MAX tasks; for help creating tasks see the NI-MAX help file and resources)
2. Enter the location and names for the input and output log files to be saved to.
3. Click “Done Configuring”
4. You can then set the trigger information.
5. Click “Next” to begin waiting for the trigger and then collecting data. (The graph will not be updated in real time.)
6. Click “Stop” to stop collecting data and preview the data that has been collected.
7. The data collected will be output to the files specified in step 2.
8. Click “Next” to begin from step 5 or end the program.

Starting:

To begin, double click on the file called “Data Collection Main.vi”. This will open a LabView screen; click on the run arrow at the top left to begin running the program.

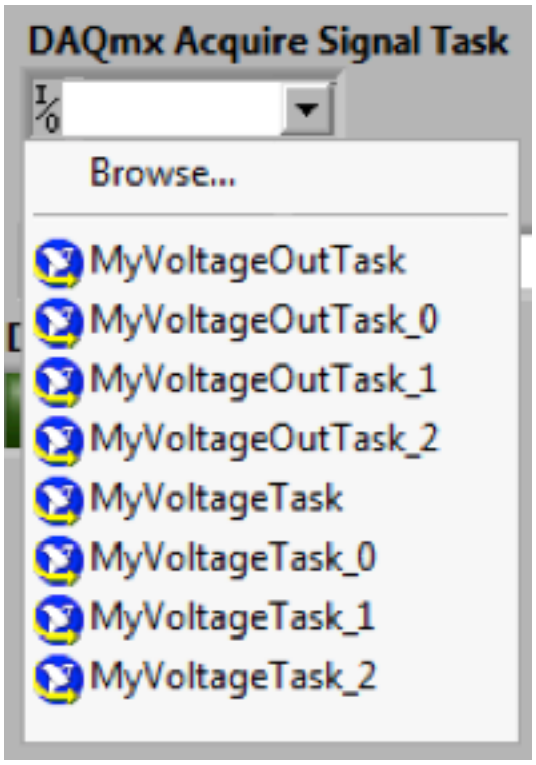
The Task and File Configuration Screen:

The first screen that comes up will look like this:

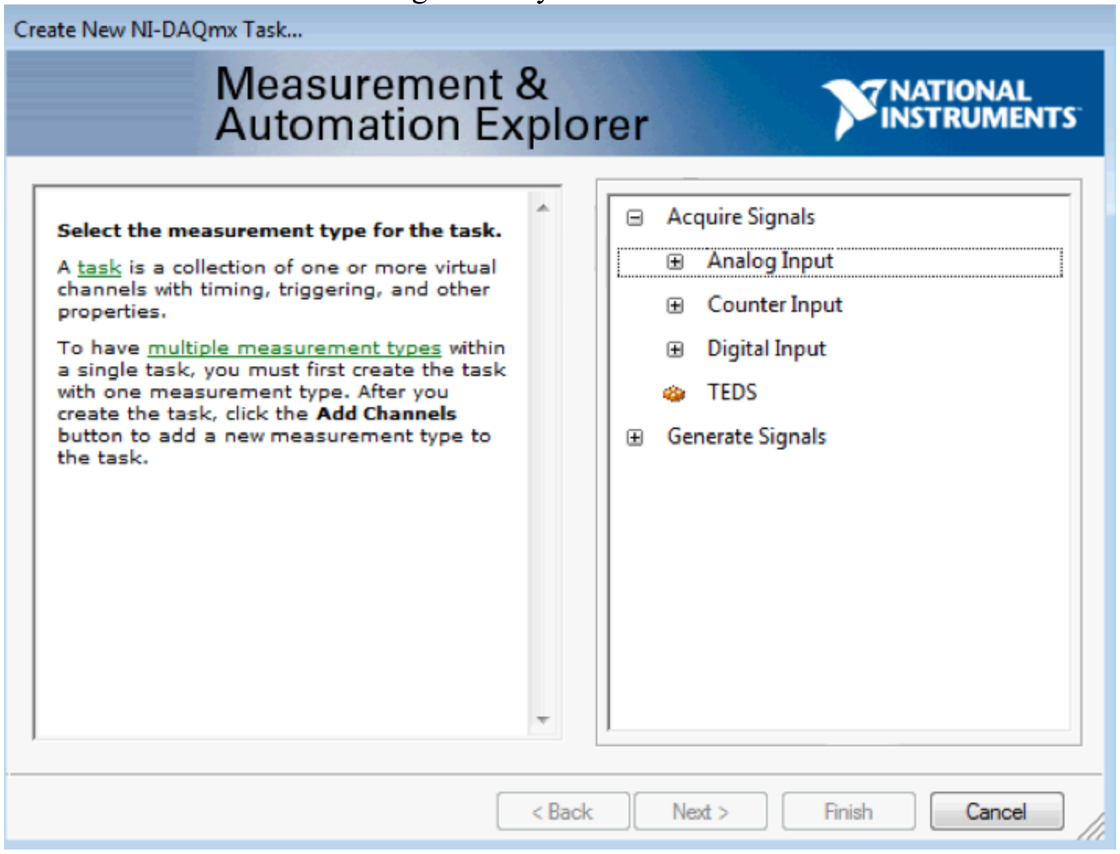

Select the NI-MAX tasks for acquiring data from sensors and outputting a signal to a device.
Enter the Log file names and locations to store the data collected and signal output.
If a file of the name selected already exists in that location, a number will be added to the end of the file name which will be incremented every time you try to write to that file.

DAQmx Acquire Signal Task
1
DAQmx Generate Signal Task
2
Acquired Input Log File
3
Generated Signal Log File
4
Done Configuring Tasks
DONE
5

The selector on the left (1) is for the task that contains information on what devices and channels your sensor data will be coming from. The selector on the right (2) is for the task that will have the devices and channels for any output devices you may have. Click on either selector to see a list of the available tasks. You can select a previously made task, or click “browse” , then “create new”, and then“MAX Task” to create a new task. To run tests without sending a control signal leave the Generate Signal Task selector (2) blank.



When a new task is being created you will first see a screen like this:



Select the measurement type for the task.

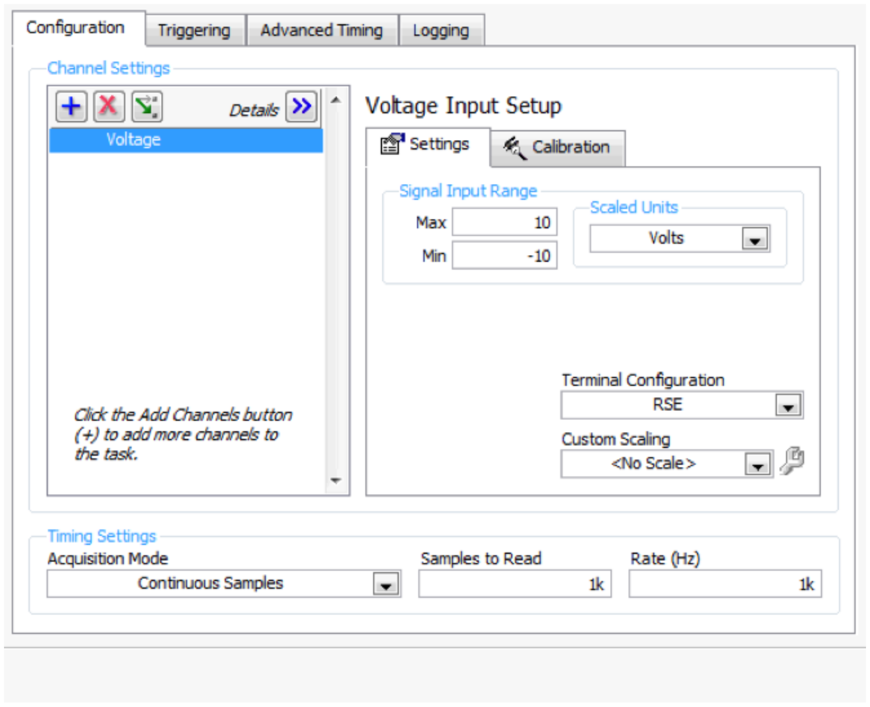
A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

This is to select what type of measurement to use for the task, i.e. if you are reading from an accelerometer use Analog Input Accelerometer. If you are creating a task to read data from sensors you must use an Acquire Signals task, if you are creating a task to send data to an output device you use a generate signals task.

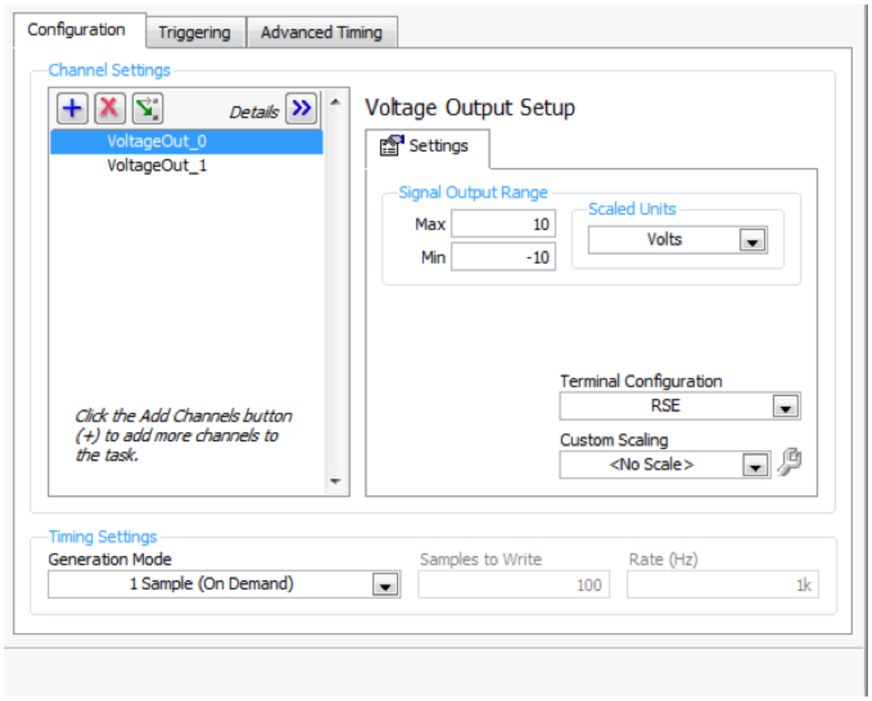
You will then be prompted to choose the device and channel to read from. The devices and channels available are imported from the NI-MAX application. Name your new task and click "Finish" to continue.

When creating a new task, the data acquisition task must have the Acquisition Mode set to "Continuous Samples" in the NI-DAQmx screen (see below), the rate you provide will set the number of samples retrieved per second per channel.



Max 10
Min -10
Scaled Units: Volts
Terminal Configuration: RSE
Custom Scaling: <No Scale>
Click the Add Channels button (+) to add more channels to the task.
Acquisition Mode: Continuous Samples
Samples to Read: 1k
Rate (Hz): 1k

When creating a new task, the signal generation task must have the generation mode set to "1 Sample (On Demand)".



Channel Settings

VoltageOut_0
VoltageOut_1

Click the Add Channels button (+) to add more channels to the task.

Voltage Output Setup

Signal Output Range

Max 10

Min -10

Scaled Units: Volts

Terminal Configuration: RSE

Custom Scaling: <No Scale>

Timing Settings

Generation Mode: 1 Sample (On Demand)

Samples to Write: 100

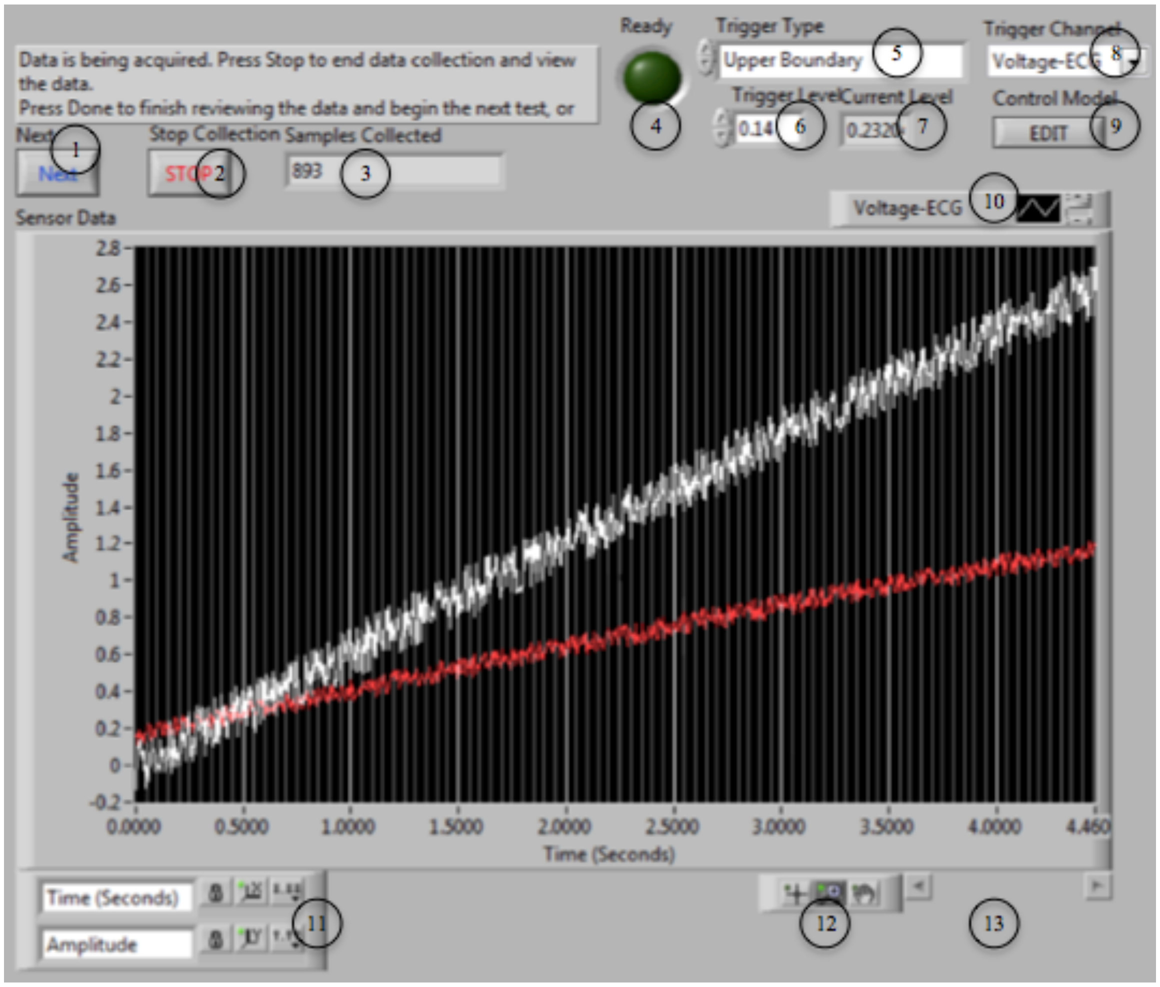
Rate (Hz): 1k

Any new tasks you make will automatically be saved in NI-MAX and can be accessed later or edited by using the NI-MAX tool.

Below (3) and (4), enter the file names for the log files to be written to. If the name is taken the program will add automatically incremented numbers to the end of it. The left selector (3) is the file to store the data recorded from the sensors, and the right selector (4) records the control signal being sent out.

When you have configured and selected the tasks you want and finished entering the file paths click the blue “Done” button (5) beneath “Done Configuring Tasks”. The Data Acquisition Screen will then come up automatically.

The Data Acquisition Screen:



Data is being acquired. Press Stop to end data collection and view the data.
Press Done to finish reviewing the data and begin the next test, or

Trigger Type: Upper Boundary
Trigger Channel: Voltage-ECG
Trigger Level: 0.14
Current Level: 0.2320
Control Mode: EDIT
Samples Collected: 893

When the screen comes up the program will take a few moments to initialize. You can spend this time setting the trigger information, when you are ready click "Next" the program will then begin waiting for the trigger condition to occur before acquiring data.

Triggering is a way to have your program not store any data until it detects a certain kind of reading from one of the sensors. The Trigger Channel (8) is the channel being monitored for the trigger conditions. The name corresponds to the name given in the NI-Max task. The Trigger Type selector (5) is used to choose a kind of trigger: “No Trigger”, “Upper Boundary”, “Lower Boundary”, or “Increase in Standard Deviation”. “No Trigger” simply starts data acquisition immediately. “Upper Boundary” waits until it sees a value above the “Trigger Level” (6) to start data acquisition. “Lower Boundary” waits until it sees a value below the “Trigger Level”. “Increase in Standard Deviation” waits until the difference in the current samples standard deviation and the previous samples standard deviation is greater than the “Trigger Level”. The “Current Level” shows the value being compared to the “Trigger Level”, it can be used to adjust the “Trigger Level” for better performance. The trigger parameters can be changed at any time. For more on triggers see the "Triggers" section farther down. When the program is waiting to detect a trigger the "Ready light" (4) turns bright green.

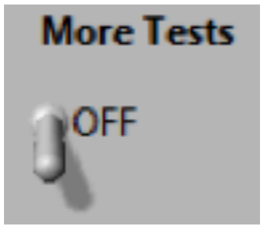
While data is being collected the "Samples Collected" indicator (3) will show how many samples have been collected so far for the current test.

When you are done collecting data press the "Stop Collection" button (2) or the "Page Up" key to stop data acquisition. The program will then graph the data for the most recent test and output the collected data to the files you specified earlier. The graph can be manipulated by zooming in or out using the zoom tools (11), scrolling using the horizontal scroll bar (12), or editing the axes using the axes control tools (10). The legend (9) can be scrolled through to see which line corresponds to each sensor channel.

When you are done reviewing the data press the "Next" button (1) or the "Page Down" key to either end the program (if not in More Tests mode) or begin the next test (if in More Tests mode).

Click the EDIT button (9) to edit the control model being used in a pop up window. For more information of Control Models see "The Control Model" section.

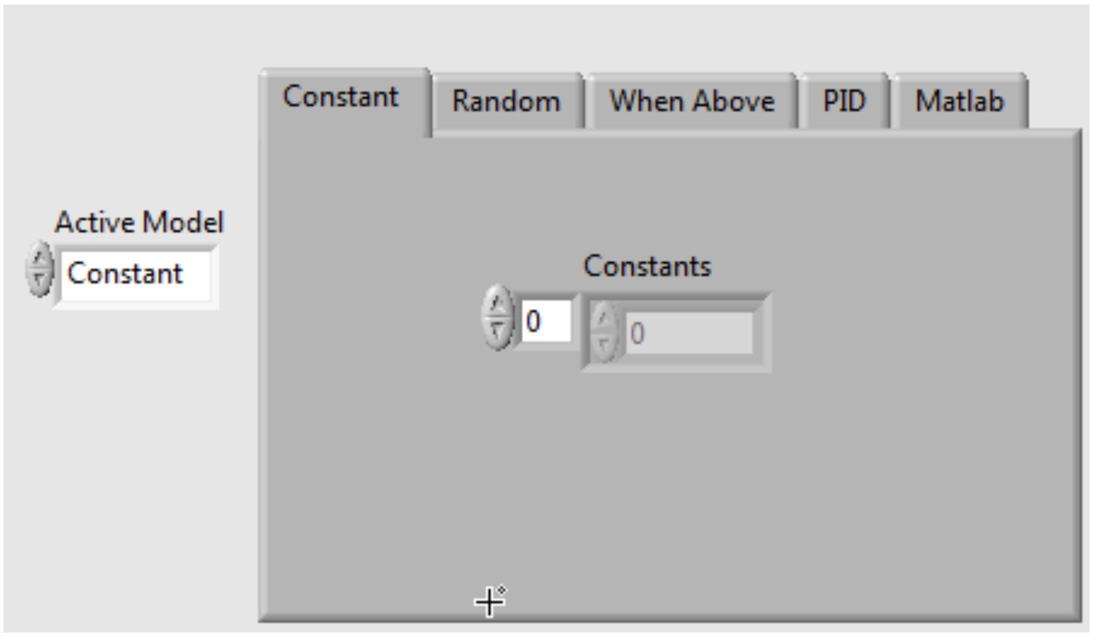
The top right corner of the screen will always contain a switch labeled "More Tests", while this switch is off More Tests Mode is disabled and when you are done reviewing data for the test the program will stop running. While the switch is on, More Tests Mode is enabled and when you are done reviewing data for the most recent test the program will clear the input array and begin collecting data for the next test automatically using the same tasks and files test (covered below). The switch can be clicked at any time to enable or disable More Tests Mode.



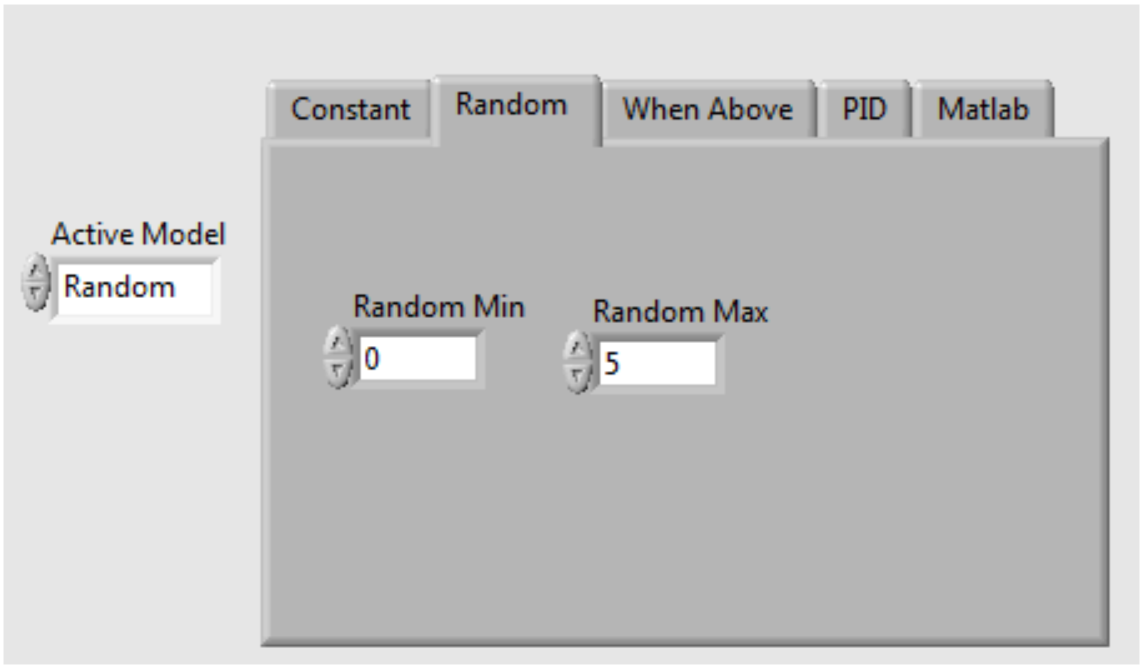
The Control Model:

To change the control model in use click on the edit button in the Data Acquisition screen while the program is running or open the Control Tool VI. This currently has five options, Constant, Random, When Above, PID, and Matlab. Change the name in the "Active Model" box to change which model will be used. If you are editing by opening the VI when you are done, go to >>Edit>>"Make Current Values

Default" and then save changes, the next time you run the Main program it will use the control model you specified in the "Active Model" field. If you are editing by clicking the edit button with Data Collection Main running then you only need to exit the Control Tool window, your changes will be kept until you stop running the program.



Constant allows you to create an array with one index for each output channel. The number on the left is the index and the number on the right is the value at that index. The value at each index in the array is constantly sent to the output channel regardless of what the input is. This will only begin after the control loop has been triggered and one batch of samples has been collected. Always on constant outputs are best set using the NI-DAQmx output task.

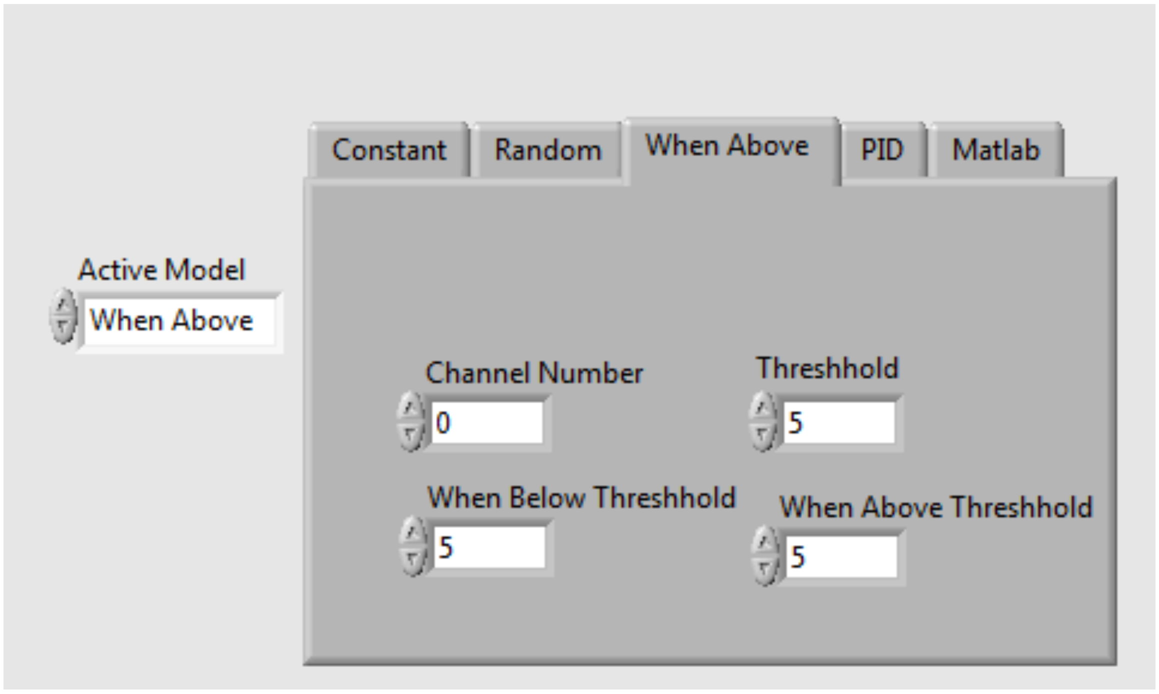


Active Model: Random

Random Min: 0

Random Max: 5

Random generates a random number between Random Min and Random Max.



Active Model: When Above

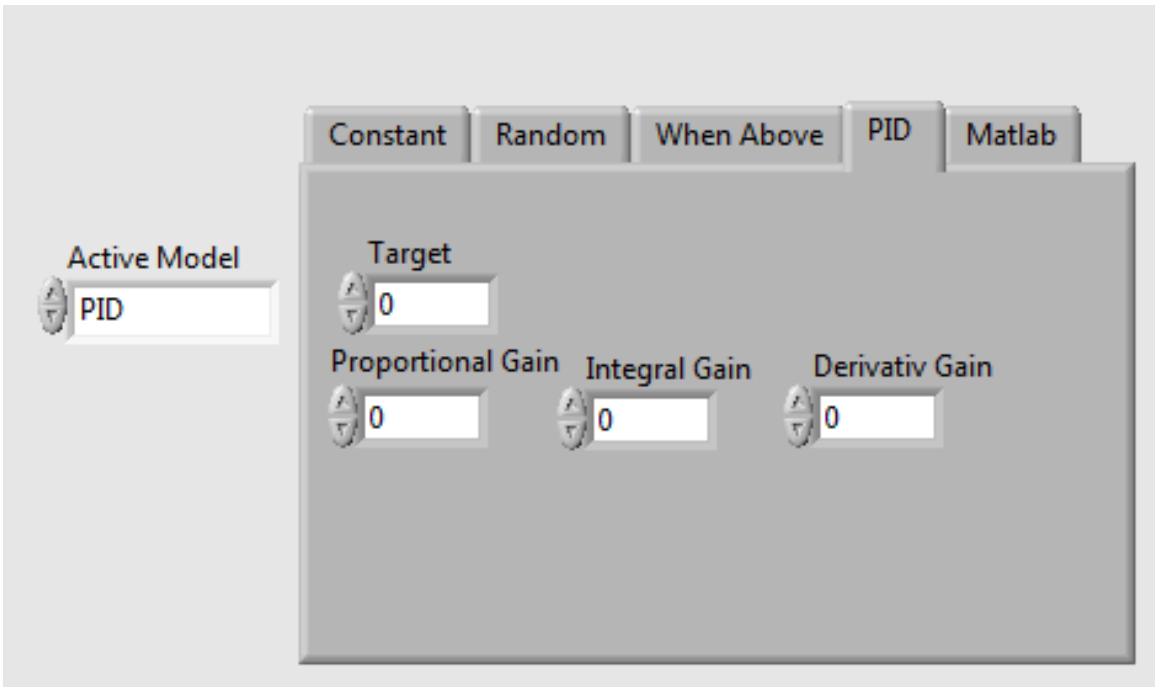
Channel Number: 0

Threshhold: 5

When Below Threshhold: 5

When Above Threshhold: 5

When Above generates a specific signal when all values in the most recent batch of samples is above a specified level.



Active Model
PID

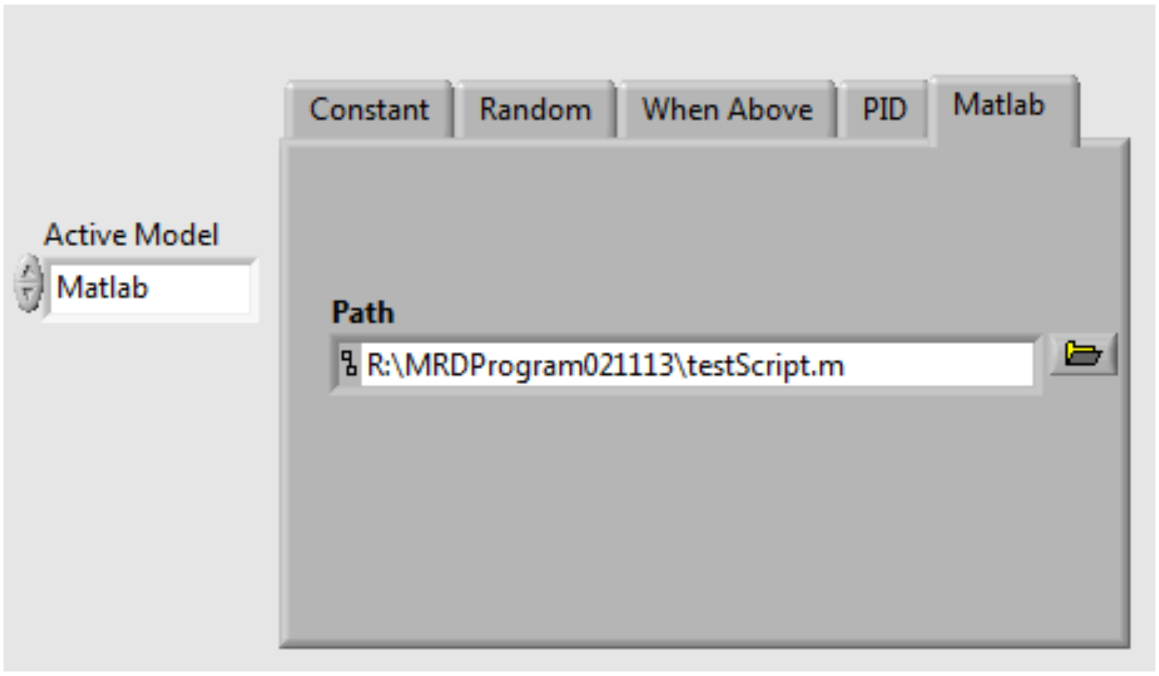
Target
0

Proportional Gain
0

Integral Gain
0

Derivativ Gain
0

PID uses the boxes to generate a signal based on the PID formula, calculating the error, its integral, and derivative and then the sum of those times their respective gains.



Active Model
Matlab

Path
R:\MRDProgram021113\testScript.m

Matlab allows you to design your own control model using the Matlab program. In order to use this option you need to have code, which makes use of two variables “numChannels” and “data”. “numChannels” is an array containing the output data from the previous run of the Control Tool and is passed to the “Control Tool” when the program is running. The other variable "data" is a two dimensional array of the data read from the most recent batch of samples, each row of the array is one of the sensor channels and each column is the value read from that channel at a specific instant in time. Your code must create a one dimensional array of length equal to the length of "numChannels" and each value being the value you want the output signal to be for the corresponding output channel.

Using Matlab can severely curtail program performance. This function uses Matlab and the Microsoft ActiveX architecture so both need to be installed on the computer you are running the program on.

To make use of this option when in the Control Model click Window from the toolbar at the top of the screen and then "View Block Diagram". When the block Diagram comes up there should be a box with the name for one of the control model choices in it. Click on that box and change the name to Matlab. Then copy and paste the script you want to use into the box labeled "Matlab Script". Or enter the file path of a Matlab script in the Matlab tab. Using a file path to specify a script will take additional time loading the script.

Creating a New Control Model:

To create a new control model that can be used with this program you will need to be able to do some programming using the LabVIEW language or Matlab. Because LabVIEW needs to open Matlab to run Matlab code and cannot do anything else while running the code, Matlab should only be used when the script is very simple and can be run quickly, in order to maintain high performance.

Using Matlab script can also result in an error when you attempt to change the "Control Tool" virtual instrument. You must save before running the control tool. If you run the "Control Tool.vi" you must exit and restart LabVIEW before making further changes because attempting to save will cause LabVIEW to quit without saving.

Creating a new Control Model in LabVIEW (Case Study):

To show how you can create a new control model consider the following example. This requires some working knowledge of programming in LabVIEW. To learn about programming in LabVIEW, try the going to the following webpage: http://www.ni.com/white-paper/7466/en

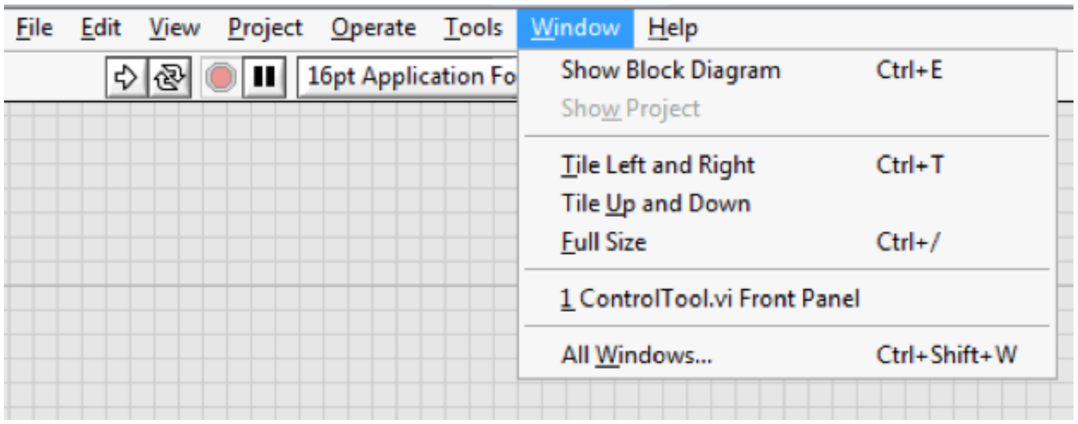
"When Above" Control Model: This model checks to make sure that every data point for a batch of samples from a specific sensor is greater than some threshold value. If it is above the threshold value then all sensors are set to some constant. Otherwise the threshold then all sensors are set to some other constant.

For example the following would be one instance of this model:
When channel 1 is above 5 units set all outputs to 1 unit otherwise set all outputs to 0 units.
This means that if the following is sent to the control model:
[2 , 6, 7, 8] then the output will be 0.
[2, 3, 4, 1] then the output will be 0.
[6 , 7, 8, 6] then the output will be 1.
[6, 7, 5, 6] then the output will be 0.

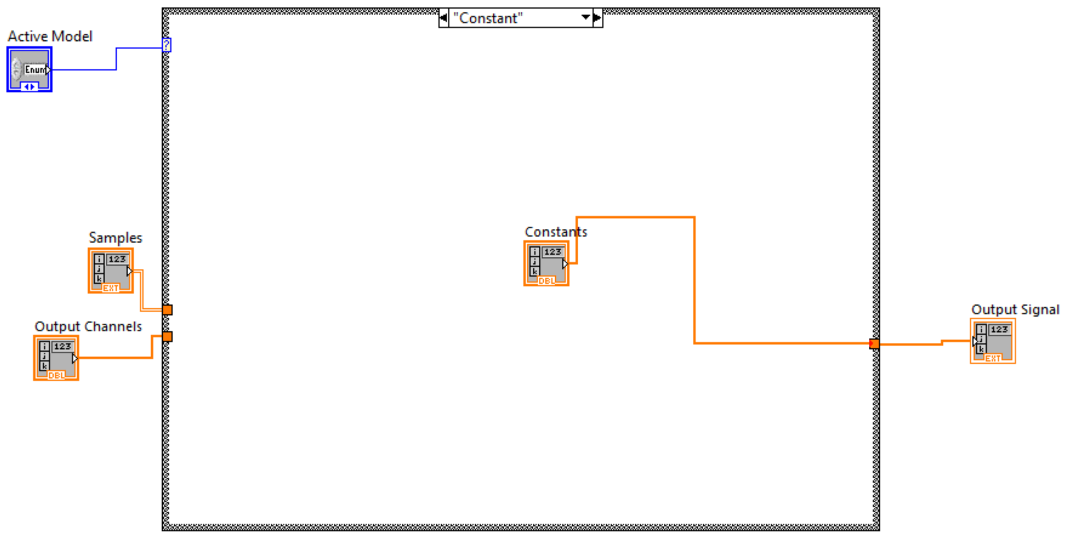
The Control Tool VI is given two variables from the main Data Acquisition program and is expected to output a single array where each index is an output channel. The input variables are called "Samples" and "Output Channels" and the output variable is called "Output Signal".

"Samples" is a 2-dimensional array containing the data for all channels for the most recent batch of samples gathered. It is arranged so that each row corresponds to a channel and each column is the instant of time the sample was gathered at. This means for two sensors where the first sensor reads a signal y=x and the second reads y = -x the subarray can be [[0 , 1, 2, 3][0, -1, -2, -3]] or [[7, 8, 9, 10] [-7, -8, -9, -10]] etc. "Output Channels" is an initialized array of the correct size to store the output signal. So if there are two output channels it is an array of length 2, if there are 3 output channels it is an array of length 4. The first time in a test it is sent to the Control Tool all of its values are 0, afterwards its values are the same as the previous "Output Signal" array.

1. Open the Control Tool.vi
2. Find "Show Block Diagram"

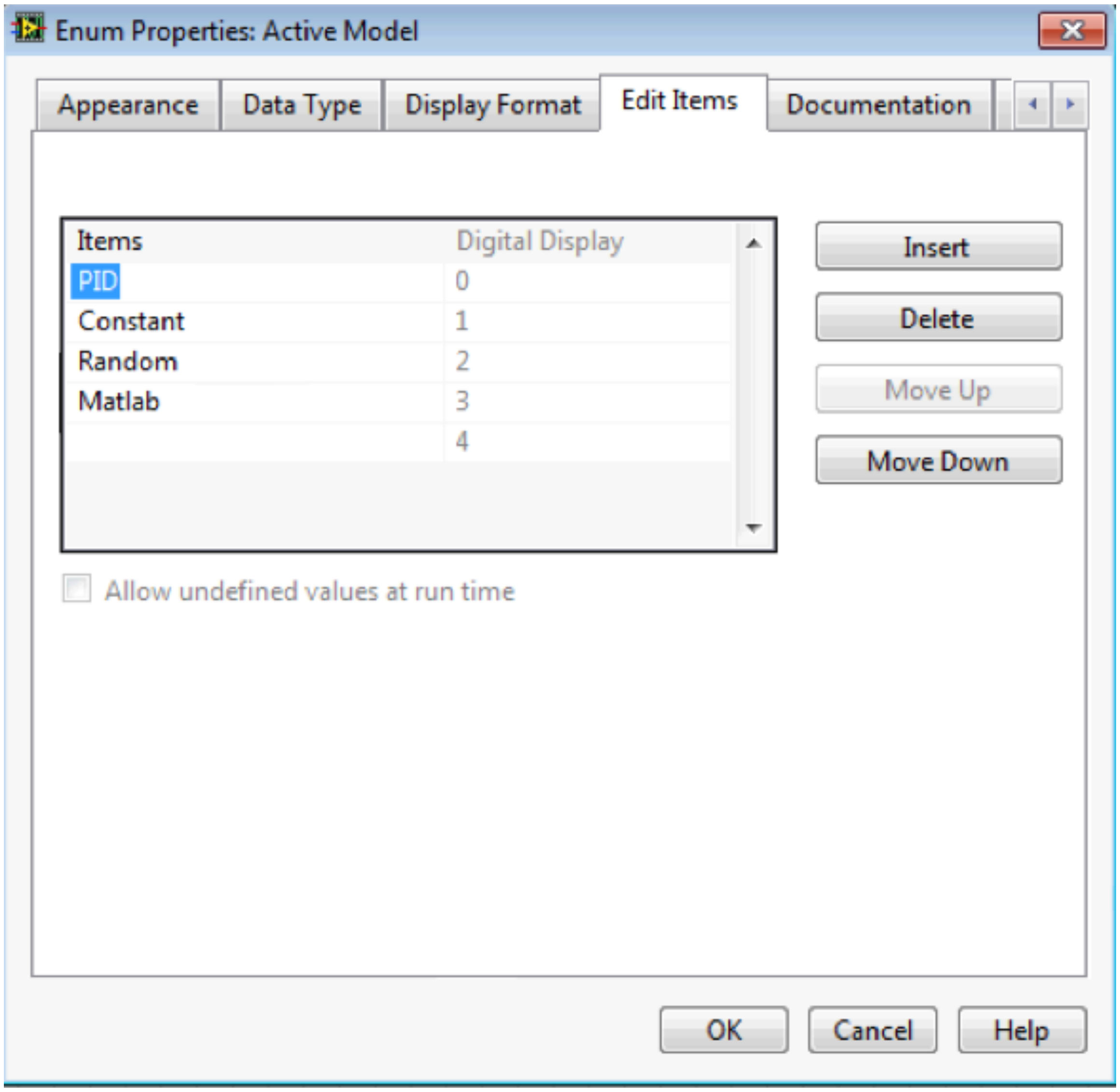

File
Edit
View
Project
Operate
Tools
Window
Help
16pt Application Fo
Show Block Diagram Ctrl+E
Show Project
Tile Left and Right Ctrl+T
Tile Up and Down
Full Size Ctrl+/
1 ControlTool.vi Front Panel
All Windows... Ctrl+Shift+W

3. In block Diagram right click on "Active Model"



4. Click properties.

5. Go to the “Edit Items” tab.


Enum Properties: Active Model

Appearance | Data Type | Display Format | Edit Items | Documentation

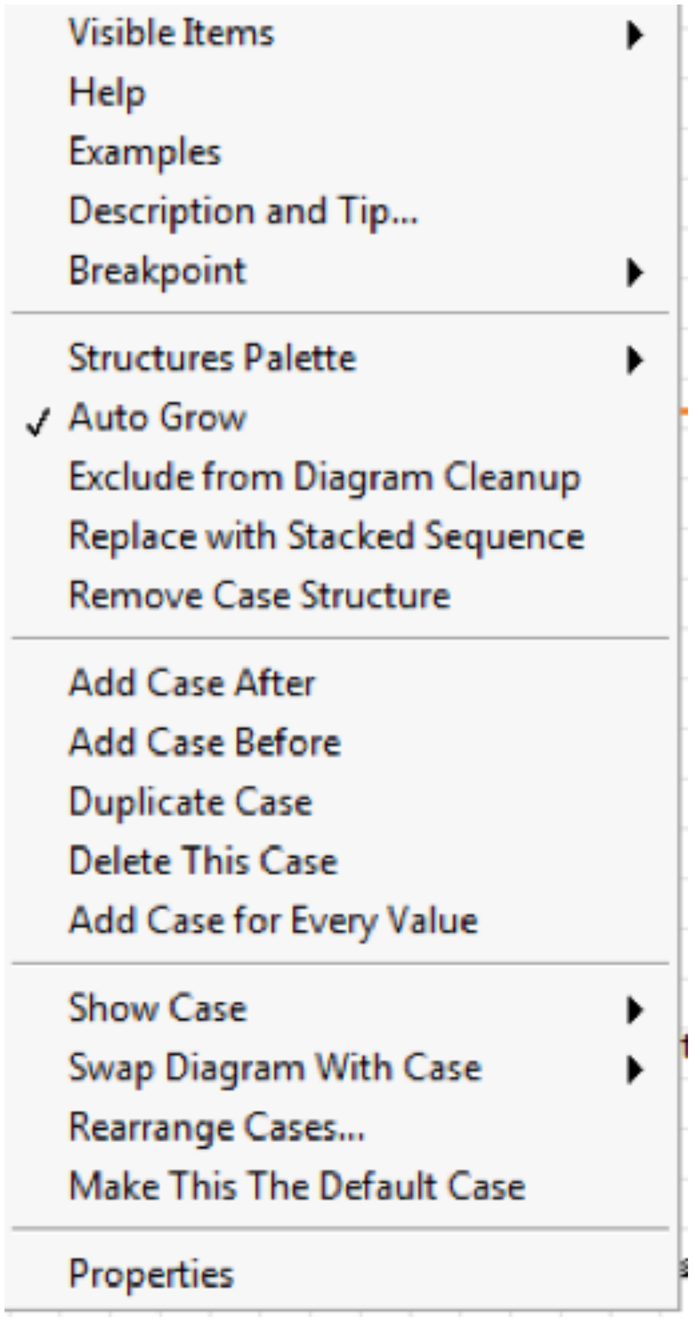
Items	Digital Display
PID	0
Constant	1
Random	2
Matlab	3
	4

6. Double click on the blank space at the bottom of the list.
7. Type new control model name.

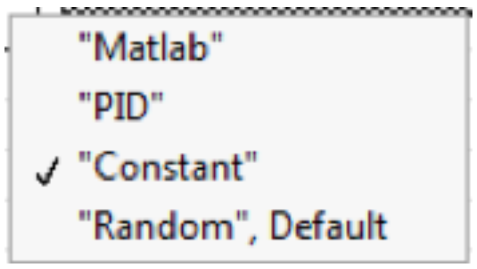
8. Hit enter and then click "OK"

9. Right Click on the case structure.

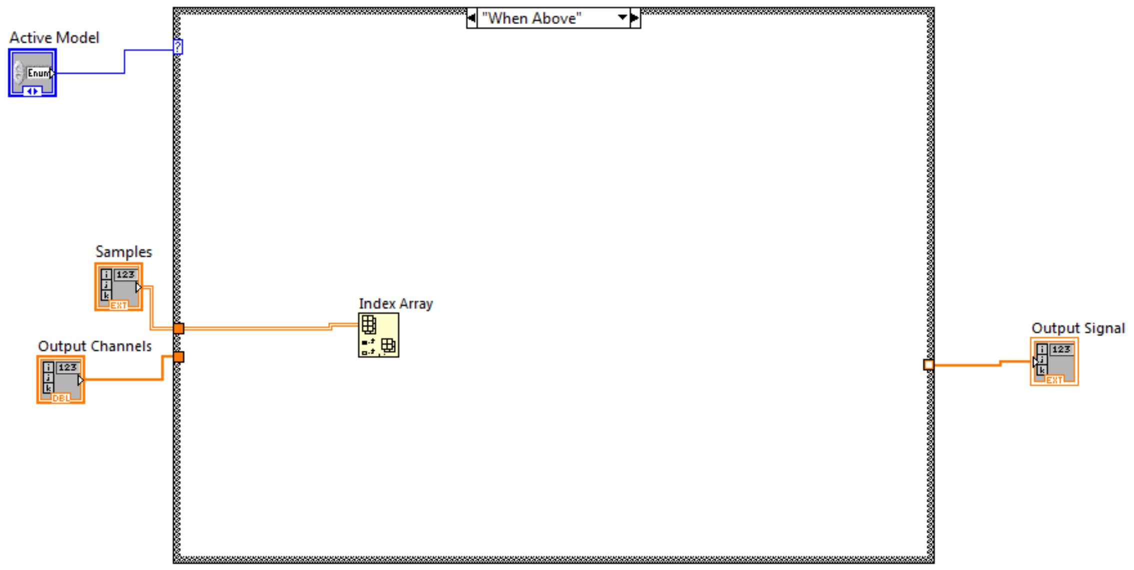
10. Select "Add Case for Every Value".



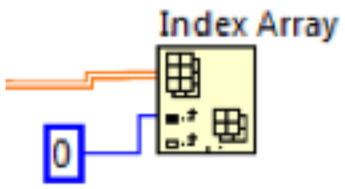
11. Select the new case.



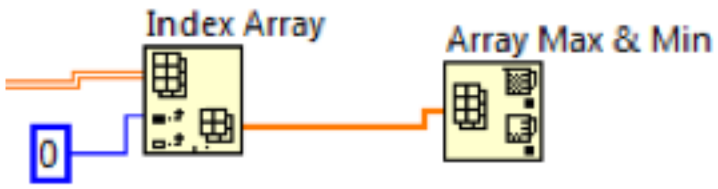
12. Add the blocks for the new case.
   a) Add index array and wire it to the data array.



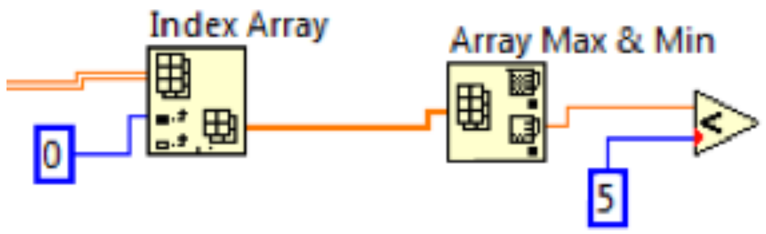
   b) Add constant and wire it to the Index Array



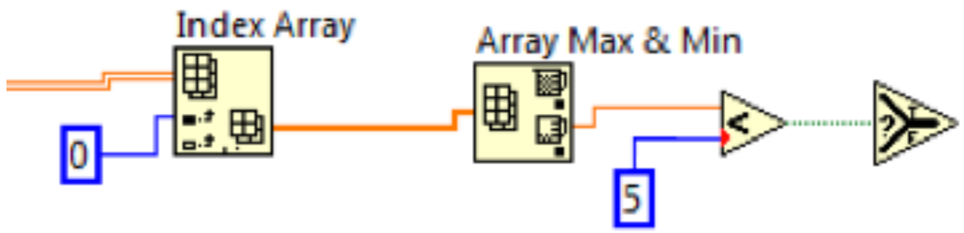
   c) Add "Array Max & Min" Block and wire it to the Index Array Block



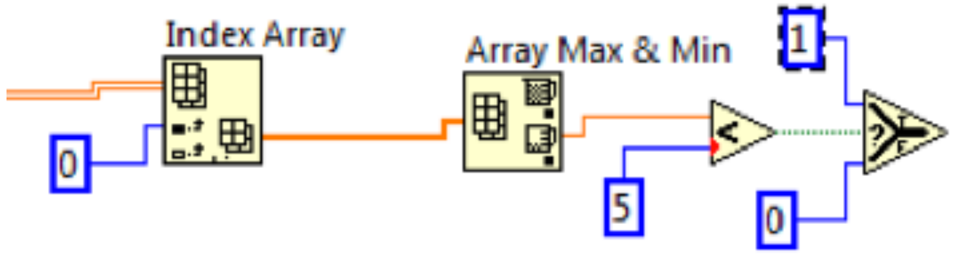
   d) Add Less than and wire it to the min terminal. Add threshold constant.



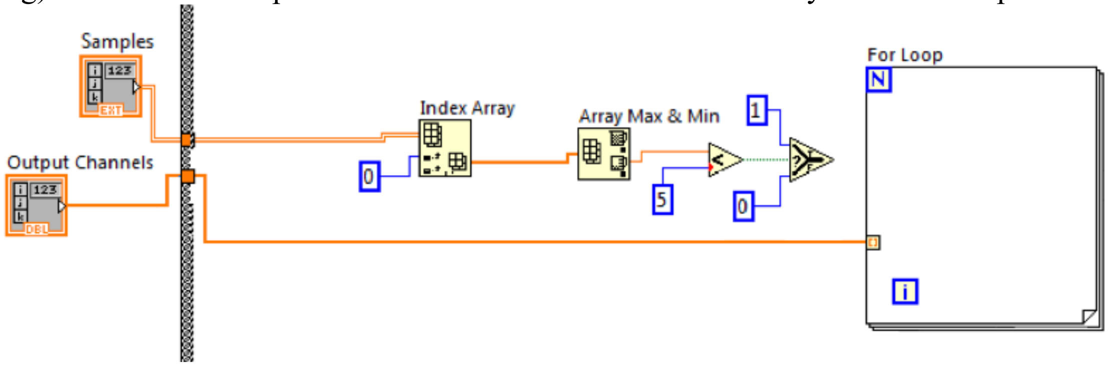
e) Add "Switch" Block and wire the ? terminal to less than.



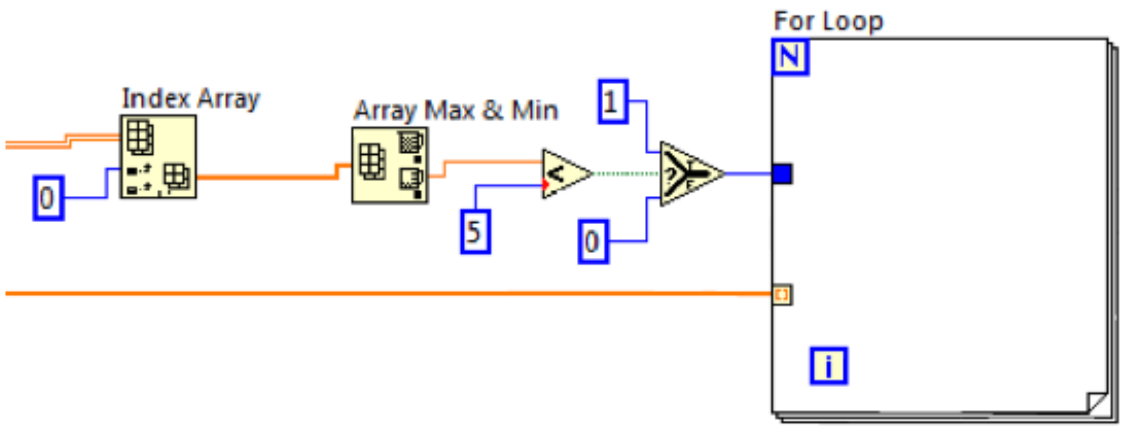
f) Add the switch constants.



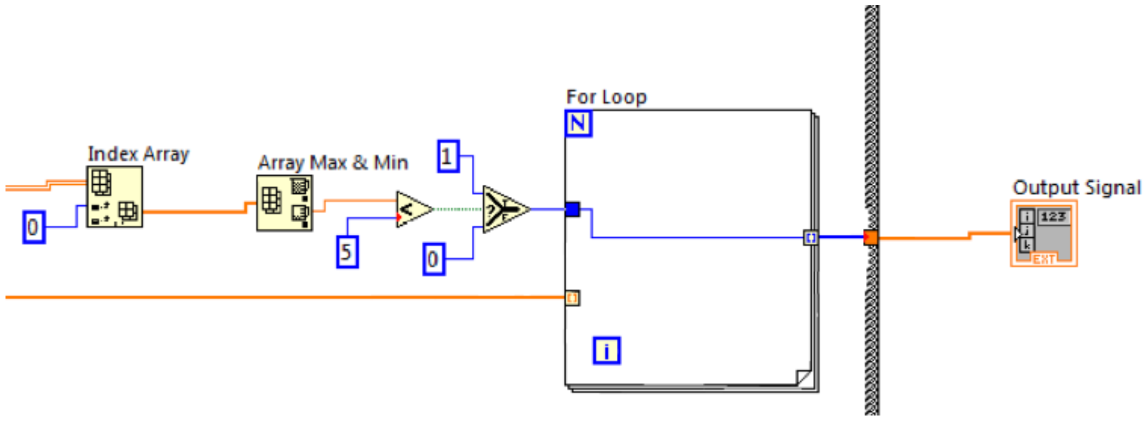
g) Add a "For Loop" structure. Wire the Num Channels array to the for loop.



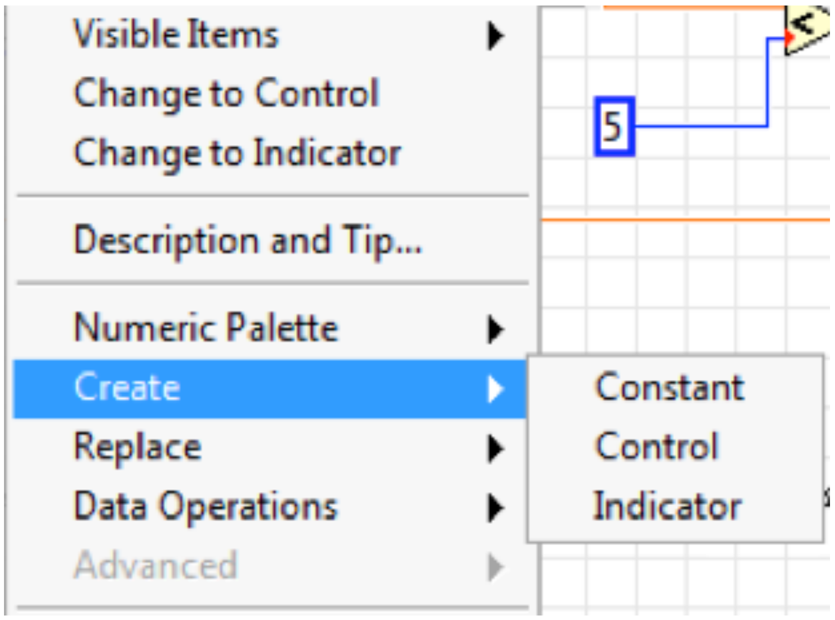
h) Wire the "Switch" block to the for loop.

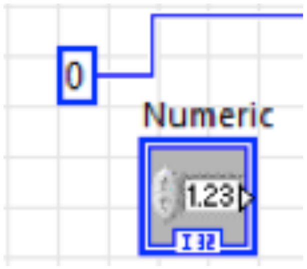


i) Wire the “Switch” block terminal to the “Output Signal” block. The for loop will automatically index the “Switch” block outputs to all indexs of the array.

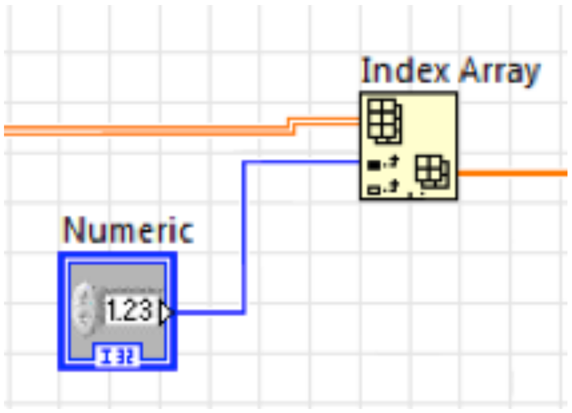


j) Right click on a constant, go to “Create “and click “Control”



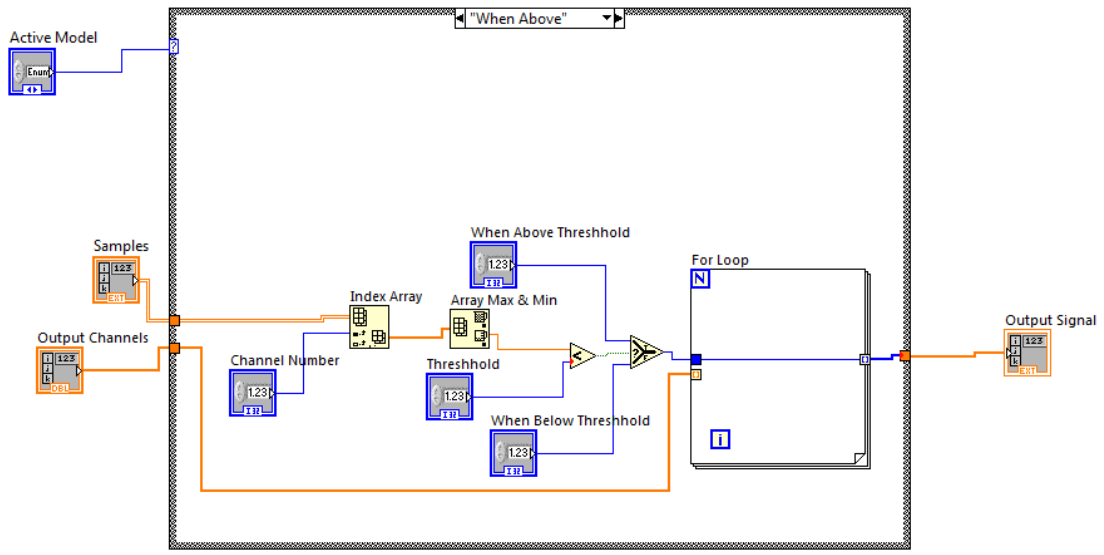


k) Hook control up in place of constant and then delete the constant

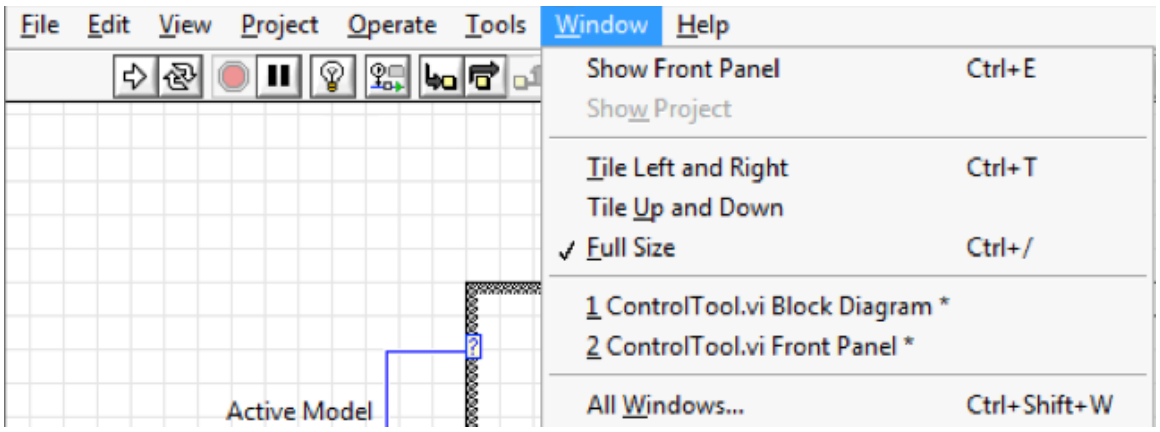


l) Click twice on the label to rename the control.

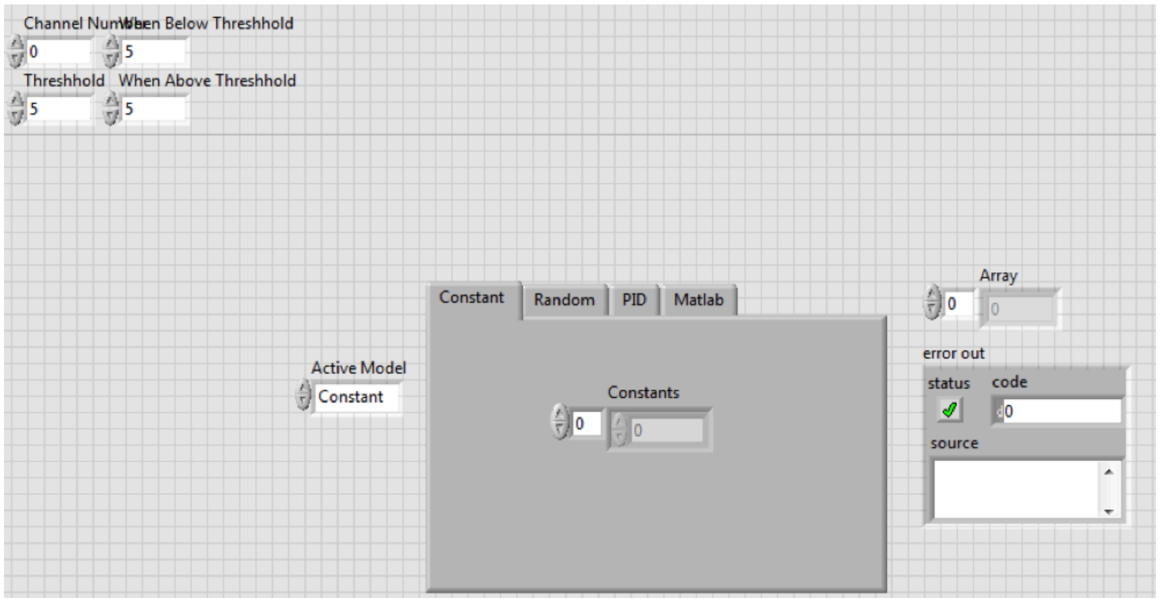
m) Repeat steps k-m for all constants that you want to be variable.



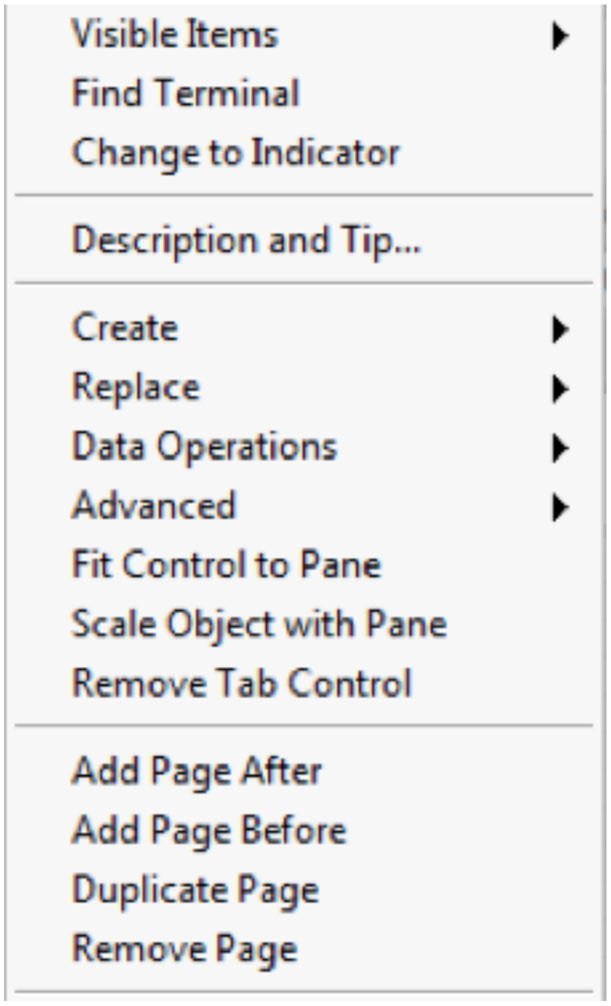
13. Go to Window>>Show Front Panel


File Edit View Project Operate Tools Window Help
Show Front Panel Ctrl+E
Show Project
Tile Left and Right Ctrl+T
Tile Up and Down
✓ Full Size Ctrl+/
1 ControlTool.vi Block Diagram *
2 ControlTool.vi Front Panel *
All Windows... Ctrl+Shift+W
Active Model

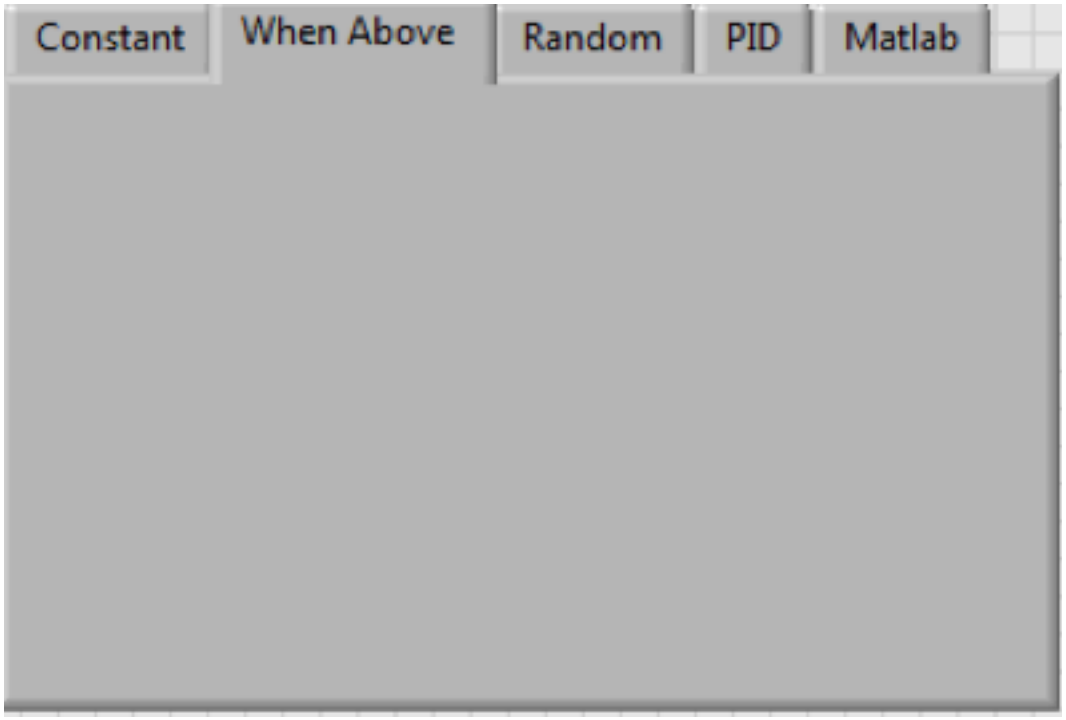
14. Locate the new controls.



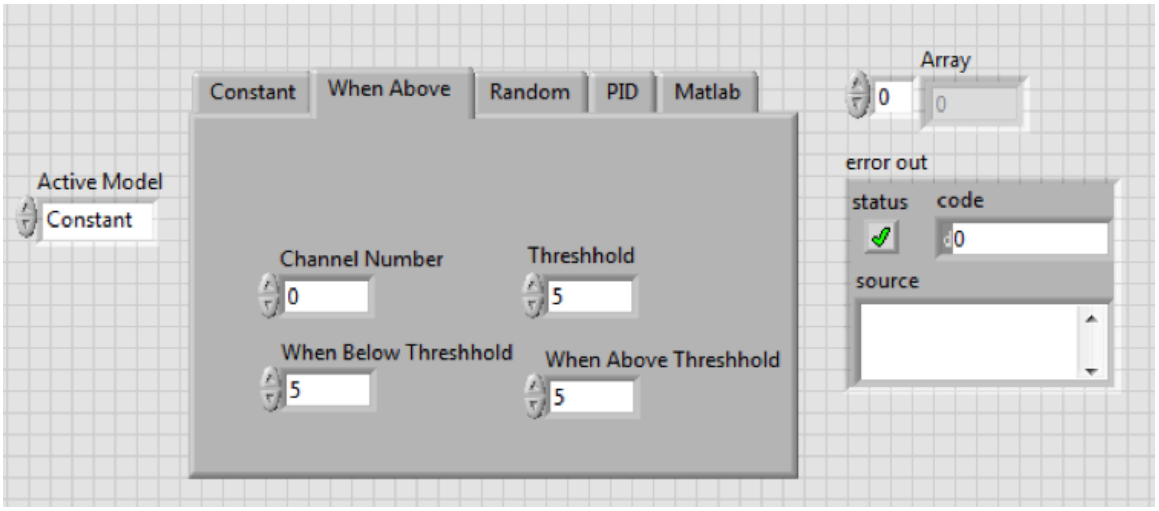
15. Right click on the tabbed pane and click add page after.



16. Rename the new tab. You can edit the name by double clicking on it.



17. Drag and drop the new controls into the new tab. And arrange them how you want them.



18. Click File>>Save

The new control model is ready to be used. Select its name in the “Active Control” menu to use it when running the main VI.

Triggers in Detail

Triggers are used to control when the program begins collecting data and sending control signals. All trigger modes save some data (20 samples minimum) that occurred before the actual trigger event, this is to ensure that the beginning of the data isn't accidentally cut off. The channel specified by the "Trigger Channel Index" box is the only one that can meet the trigger conditions and the program cannot have multiple triggers active at the same time or examine multiple channels at the same time.

There are four trigger modes:

1. No Trigger: In this mode the program immediately begins collecting data.

2. Upper Boundary: In this mode the program waits until it sees a value greater than the trigger level before collecting data.

3. Lower Boundary: In this mode the program waits until it sees a value less than the trigger level before collecting data.

4. Increase in Standard Deviation: In this mode the program waits until the standard deviation of the batch of samples has increased by the trigger level from the previous batch of samples. This will not go off on the first batch sampled unless the Standard Deviation is very high. The fundamental formula is
   $\Delta = \sigma_{current} - \sigma_{previous}$
   If ($\Delta >$ Trigger Level)
   Then, Begin Collection.

Consider the following examples of triggering by Increase in Standard Deviation:

The trigger level is set to 0.3.
The program collects the following batches of 20 samples each and calculates the standard deviation of each batch separately:
Batch A has a standard deviation of 1.2.
Batch B has a standard deviation of 0.9.
Batch C has a standard deviation of 1.21.
The trigger will begin data acquisition after reading C, the beginning of the output data will include all of B and C but none of A. The fact that the standard deviation of C is only 0.01 above A's isn't taken into account. The trigger only keeps track of the standard deviation for the single previous batch. If this is happening it means that the trigger level may not be large enough to ignore the variability in the noise data.

Assume for the next test the data read by the trigger looks like this:
A has a standard deviation of 0.9.
B has a standard deviation of 1.2.
C has a standard deviation of 1.21.
D has a standard deviation of 1.6.

Data acquisition will begin after reading D and the output will include C and D. Because the standard deviation of B is exactly 0.3 higher than the standard deviation of A the trigger will not begin data acquisition.

Now assume that the trigger level is set to -0.1 and the data looks like this:
A has a standard deviation of 0.9.
B has a standard deviation of 0.7.
C has a standard deviation of 0.68.

Data acquisition will begin after C and include B and C in the output.
0.7 - 0.9 = -0.2. -.02 is less than -0.1;
0.68 – (-0.7) = -0.02 which is greater than -0.1 and therefore the trigger begins.

Triggering by Increase in Standard Deviation is recommended because the trigger level is unlikely to change when testing parameters change so long as the same sensor is being used to check for the trigger. Also, standard deviation will always tend to increase during impact regardless of the direction of the change in the input read. This means that even if the orientation of the sensors is changed the trigger level will be likely to remain the same.

The trigger will never activate for a value equal to the trigger level.

Possible Issues and Troubleshooting:

If the data collected is inconsistent from test to test there may be aliasing. In order to fix this, try increasing the sampling rate.

If the program appears to freeze during tests it may be that the UI can't keep up with the data acquisition. It may be best to close any unnecessary programs. Or use the key commands to run the test stopping data acquisition some time after the impact, if the graph than updates after a few moments you can check the validity of data and keep running tests without using the UI while data is being collected.

If an error message says something about one of the input/output device being inaccessible or that the device was unable to do something try turning the device off and then on again and check to make sure that all of the modules are connected properly.

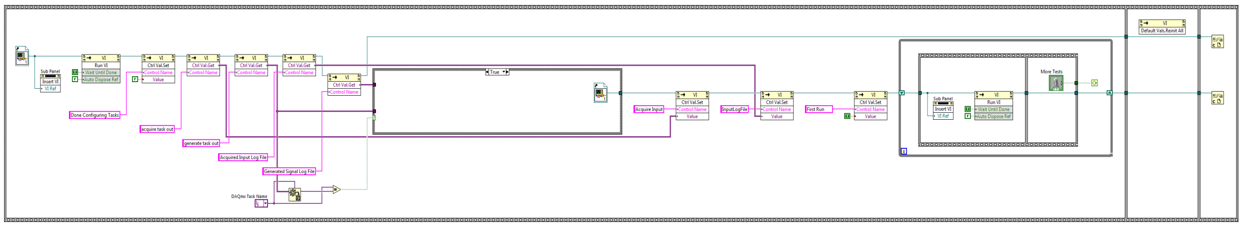
If attempting to edit or save the Control Tool VI results in an invalid operation error that closes the program without saving.

Program In-Depth Analysis:

Block Diagrams:

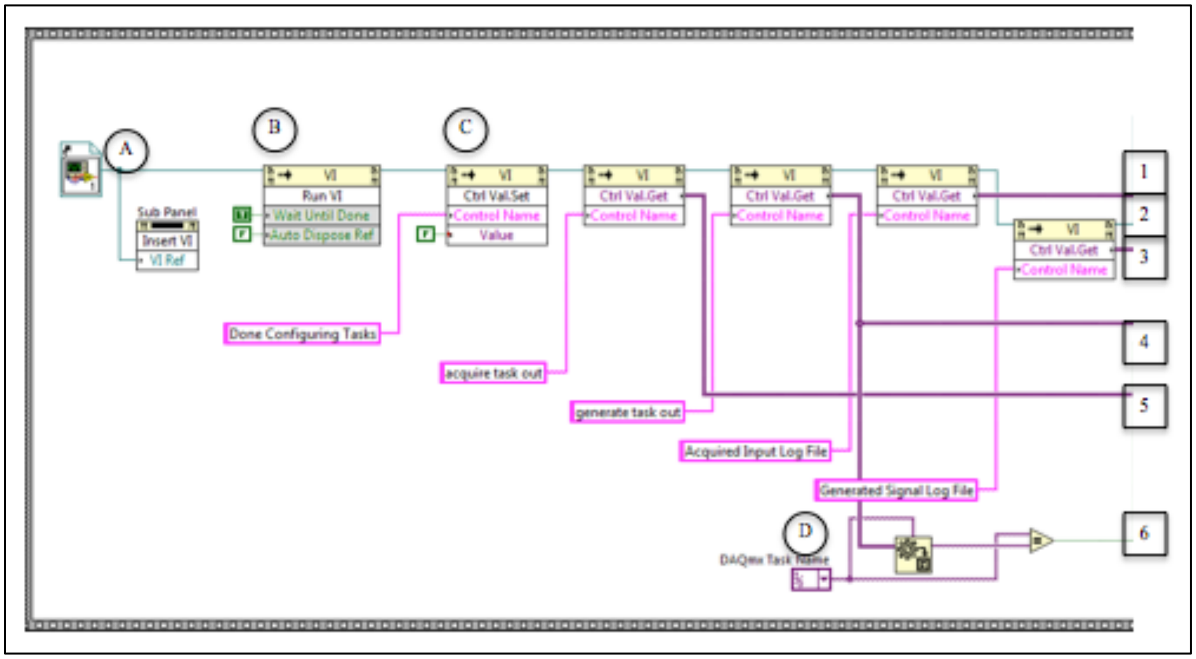
This section contains the block diagrams used in the program. Beneath each picture or set of pictures is a description of what the program is doing at that point in the process. Each circled letter represents a reference point to the blocks of code around it. Each numbered square is a reference to a wire that is broken by the way the block diagram is partitioned in this section. Each number refers to the same value within a virtual instrument (VI) and a complete list of the numbers and their values can be found at the end of each VI's section.

Data Collection Main



VI Process Outline:

1. Create a reference to "Task and File Config"
2. Run "Task and File Config" in a sub-pane on the Front Panel.
3. Collect the user inputs from "Task and File Config".
4. Create a reference to "DAQ and Display Loop" or "DAQ and Display Loop (input only)".
5. Load the input form "Task and File Config" into "DAQ and Display Loop"
6. Run "DAQ and Display Loop" while "More Tests" is on.
7. Close references and reinitialize everything to default.



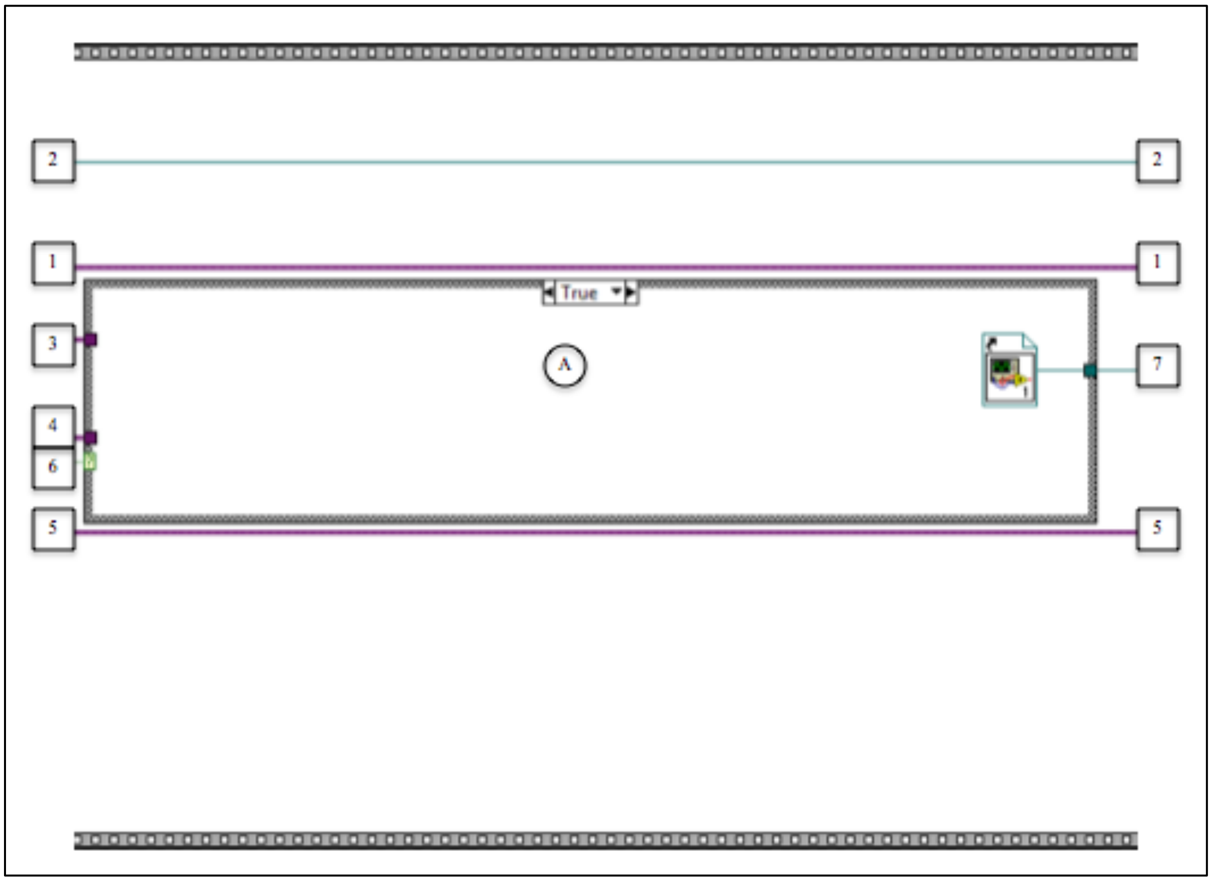
This code displays the “Task and File Config” screen to the user and then collects the data entered into that screen to be used later in the program.

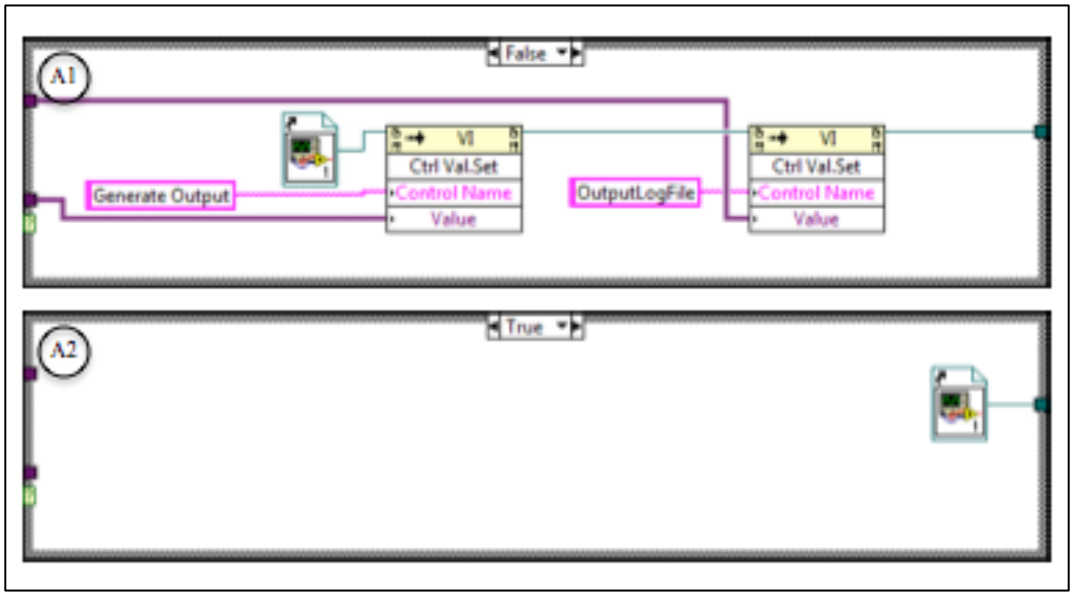
At A the program creates a reference to the “Task and File Config.vi” file. This reference is used to insert the “Task and File Config” VI into the subpanel on the front panel.

At B “Task and File Config” is run as a sub-VI from the reference. The Run VI block is given the parameters “Wait Until Done = True” and “Auto Dispose Ref = False”. This is to make sure that the program does not attempt to continue until the user has finished entering the information into “Task and File Config” and to make sure the reference is usable for retrieving the information the user input into that VI.

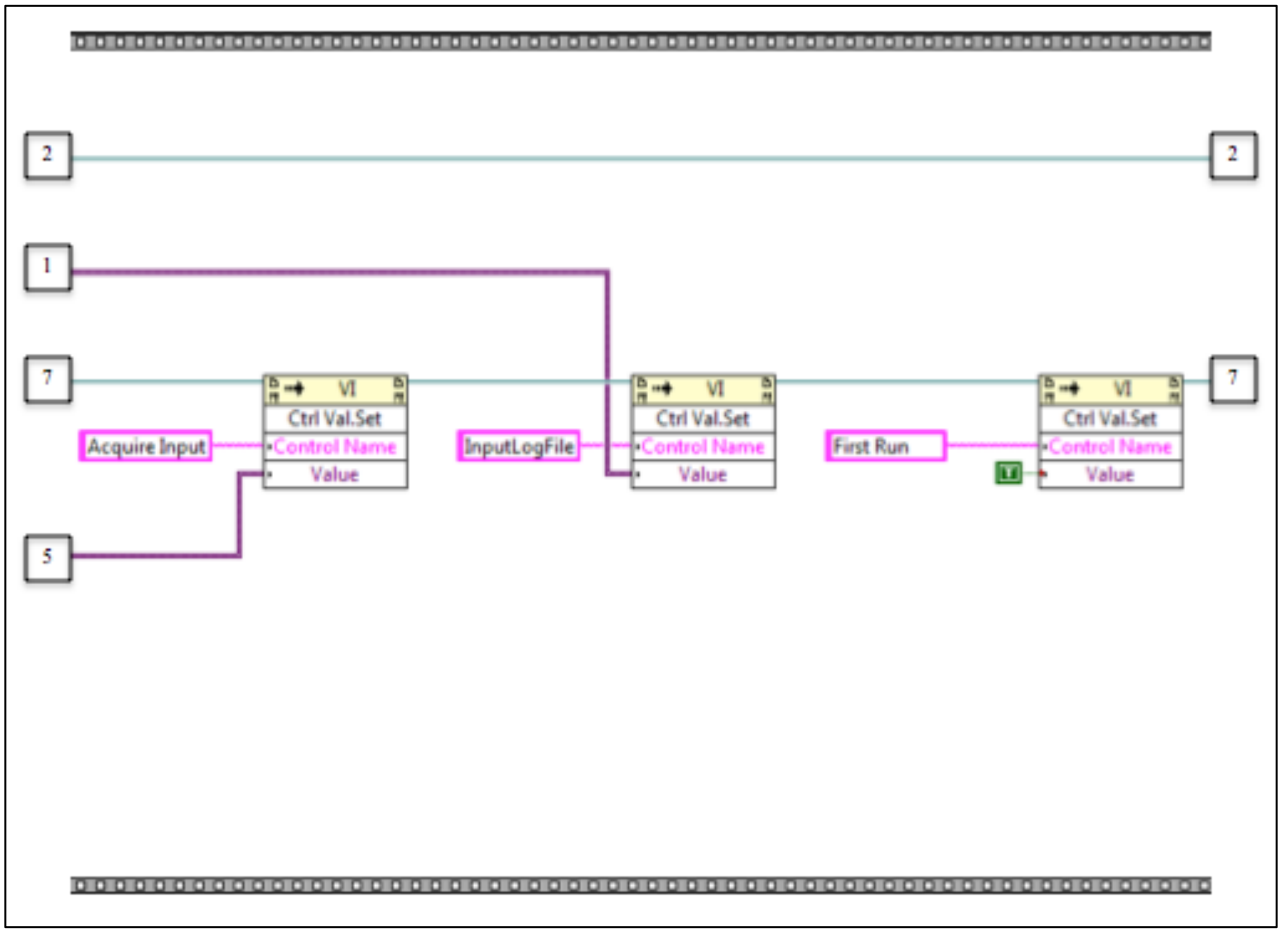
Starting at C the program uses VI Property nodes to retrieve data from the controls in “Task and File Config”. These controls are for the I/O tasks to be used and the paths for storing the two log files. The data is returned as a LabVIEW value data type.

At D the program converts the value for output signal generation to an equivalent NI_DAQmx task and then determines whether the task is empty.

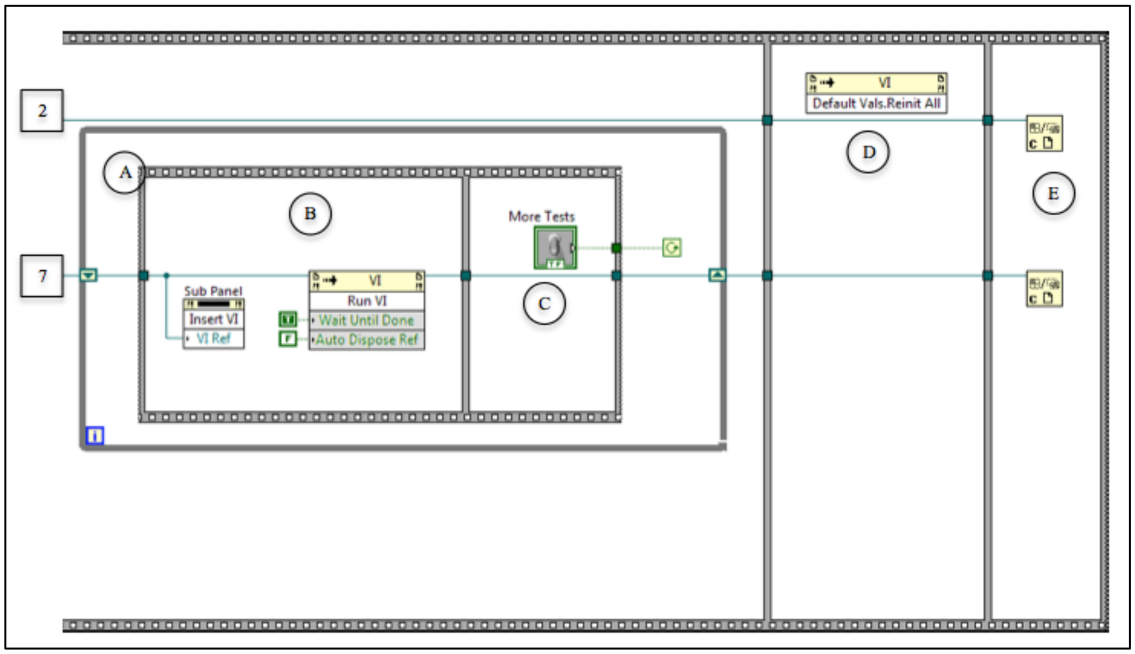




This section of code creates a reference to a VI based on whether the generate output task given is blank. If it is blank then A uses the true case (A2) and returns a reference to "DAQ and Display Loop (Input Only).vi". Otherwise it uses the false case (A1) and returns a reference to "DAQ and Display Loop.vi" and sets the generate signal task control and output log file path control based on the user input from “Task and File Config”.



This section of code sets the acquire input control and the input log file control of the VI given by reference 7 using the user input from "Task and File Config". It also sets the first run control of the VI at reference 7 to true. This indicates that it is the first test being run since the program was started. This is necessary to make sure that "DAQ and Display Loop" waits for the user to hit the next button before beginning to wait for a trigger.



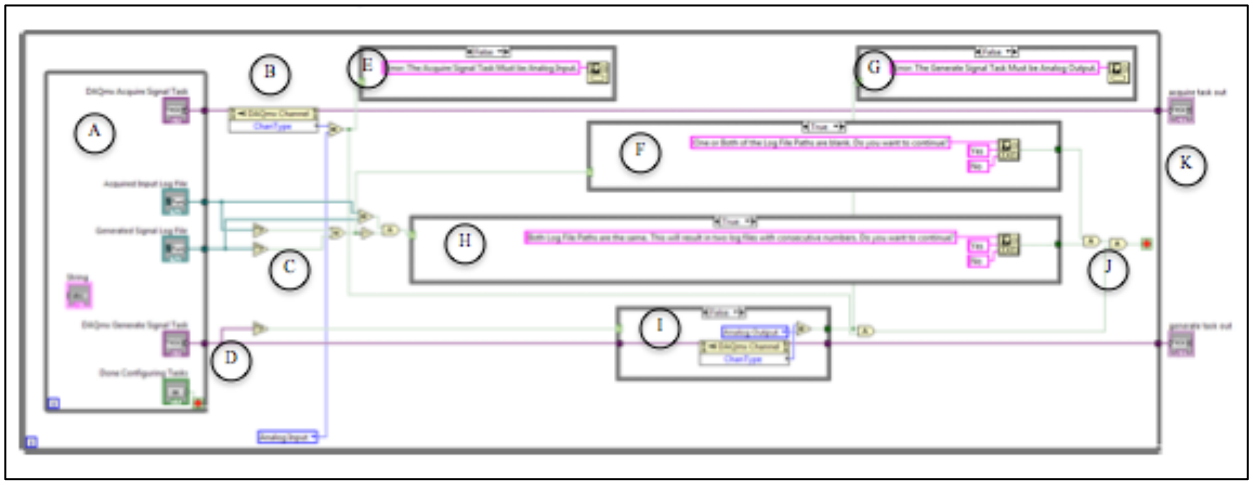
Loop A runs the "DAQ and Display Loop" multiple times each run counting as a single test. At B the program inserts "DAQ and Display Loop" into the sub-pane on the Front Panel and then runs it using the Run VI block. At C it checks if the "More Tests" switch is enabled if so loop A continues to run otherwise loop A stops.

After loop A stops the program uses the VI Reinitialize All to Default Values invoke property node with no parameters. This resets this VI and all sub-VI's to their default values. At E it closes the references to "Task and File Config" and "DAQ and Display Loop" before ending the program.

Connection Index:

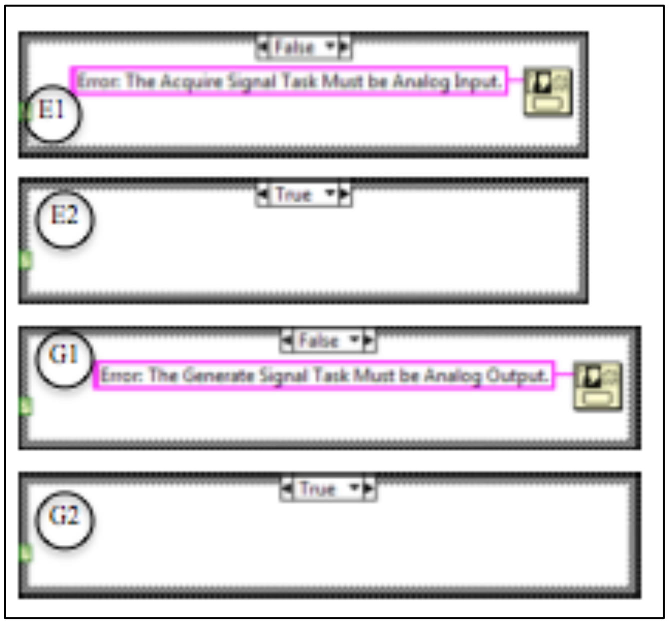
1. Value from the "Acquire Input Log File" control in "Task and File Config". Represents a files path for the record of input to be written to.
2. Reference to the "Task and File Config.vi" file.
3. Value from the "Generate Signal Log File" control in "Task and File Config". Represents a files path for the record of output to be written to.
4. Value from the "generate task out" indicator in "Task and File Config". Represents a task to be used for output signals.
5. Value from the "acquire task out" indicator in "Task and File Config". Represents a task to be used for acquiring data.
6. Boolean value indicating whether the task from 4 is null.
7. Reference to be used for the data acquisition and control. References the "DAQ and Display Loop.vi" file or the "DAQ and Display Loop (Input Only).vi" file.

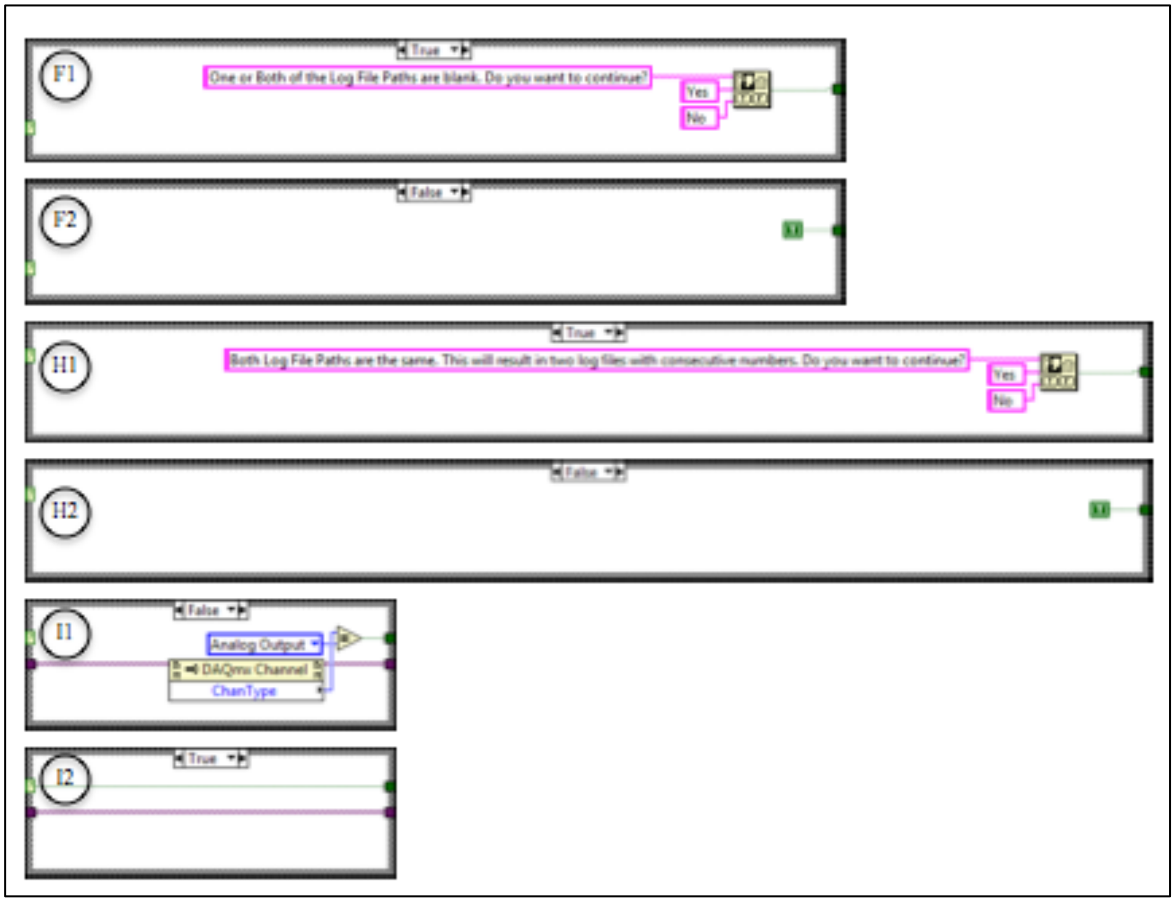
Task and File Config


A
B
C
D
E
F
G
H
I
J
K
Error: The Acquire Signal Task Must be Analog Input.
Error: The Generate Signal Task Must be Analog Output.
One or Both of the Log File Paths are blank. Do you want to continue?
Both Log File Paths are the same. This will result in two log files with consecutive numbers. Do you want to continue?
DAQmx Acquire Signal Task
Acquired Input Log File
Generated Signal Log File
DAQmx Generate Signal Task
Done Configuring Tasks
acquire task out
generate task out
Analog Input
Analog Output
DAQmx Channel
ChanType

VI Process Outline:

1. Wait until the user presses "Done"
2. Evaluate the inputs for different error and warning conditions.
3. If there are errors show the error dialogue to the user and return to 1.
4. If there are warnings display the warning dialogue and ask the user if they want to continue.
5. If the user doesn't want to continue go back to 1.
6. Otherwise end program.


False
E1
Error: The Acquire Signal Task Must be Analog Input.
True
E2
False
G1
Error: The Generate Signal Task Must be Analog Output.
True
G2


F1
True
One or Both of the Log File Paths are blank. Do you want to continue?
Yes
No
F2
False
H1
True
Both Log File Paths are the same. This will result in two log files with consecutive numbers. Do you want to continue?
Yes
No
H2
False
I1
False
Analog Output
DAQmx Channel
ChanType
I2
True

"Task and File Config" contains UI elements and verification for the user to input the tasks for acquiring and generating signals as well as the file paths for the log files. It begins with a loop to prevent the program from ending until the user has entered valid input. At Loop A the program maintains controls for specifying DAQmx tasks and file paths, one of each kind for input and one of each kind for output. The String indicator is used to provide general information on the VI to the user.

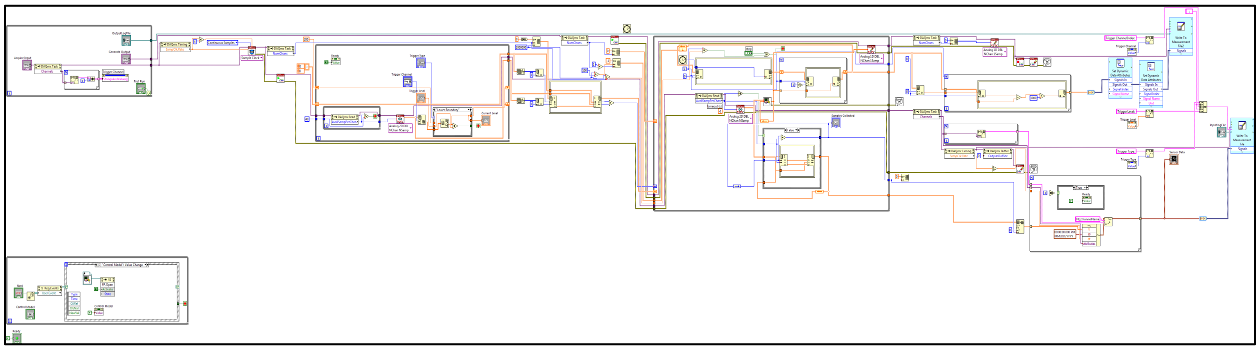
At B the program determines if the task entered in the “DAQmx Acquire Signal Task” control is an Analog Input Task and sends the Boolean value indicating this to the case diagram E. If given a true value (the task is an Analog Input task) E does nothing (E2), otherwise it creates and displays an error message (E1).

At C the program determines if the File Paths are not equal to each other or blank and sends the Boolean value to the case diagram H and F respectively. If H is given a true value (the File Paths are equal to each other) it displays a warning asking if the user wants to continue anyway (H1), otherwise it does nothing (H2). If F is given a true value (one or both of the file paths are blank) the program displays a warning asking if the user wants to continue anyway (F1), otherwise it does nothing (F2).

At D the program determines if the task in “DAQmx Generate Signal Task” is blank and sends the value to I. If the value given to I is false (the task is not null) it determines if it is an Analog Output task and sends the value to G (I1). Otherwise it sends true to G to prevent G from triggering an unnecessary error (I2). G throws an error if the task is not an Analog Output task (G1), otherwise it does nothing (G2).

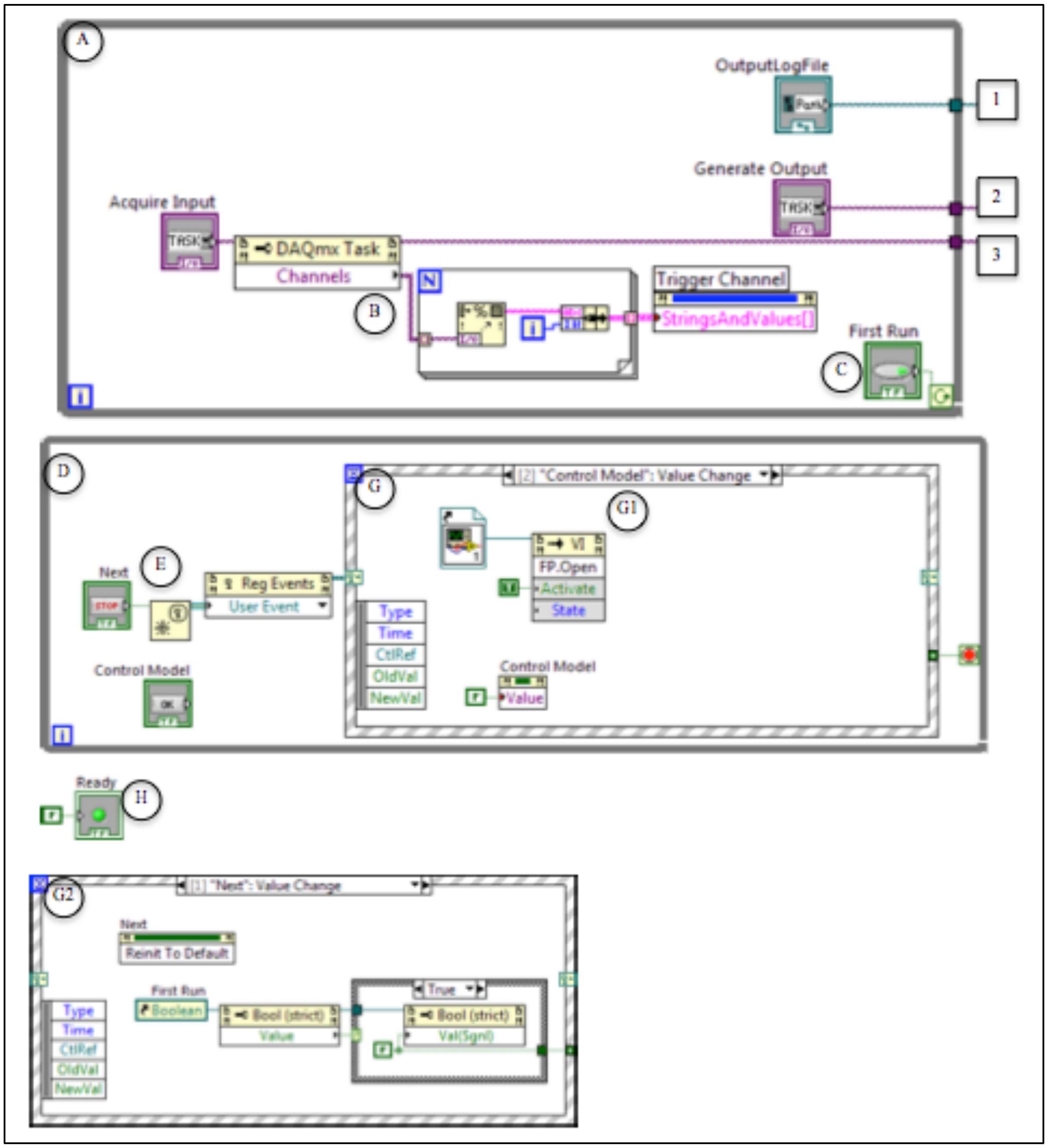
At J the program uses the Boolean values from B, F, H, and I to determine if the loop can end or if it should continue looping. If B or I return false indicating that one of the tasks is an invalid type the program always loops. If F or H return false indicating that the user chose not to continue after receiving a warning about the file paths the program loops. If B, F, H, and I all send true values to J then the loop ends. At K indicators are set with the values of the tasks from the “DAQmx Acquire Signal Task” and “DAQmx Generate Signal Task” controls to be used by the “Data Collection Main” VI when running “Task and File Config” as a sub-VI.

DAQ and Display Loop



VI Process Outline:

1. Controllers with the task information are set.
2. Wait until next is pressed.
3. Start the tasks. Perform initializations.
4. Read data and test for triggers until a trigger condition is met.
5. Add the data collected prior to the trigger to the final array for storing data.
6. Send the data to the main reading loop.
7. Read all available samples.
8. Send the read samples and the previous output to the Control Tool VI.
9. Write the output of the Control Tool to the output task.
10. Store the output task and click count.
11. Store the read data.
12. Repeat until stopped by the UI.
13. Convert the read data to waveforms.
14. Write the waveforms to a file and graph the waveforms.
15. Write the output data to a file.



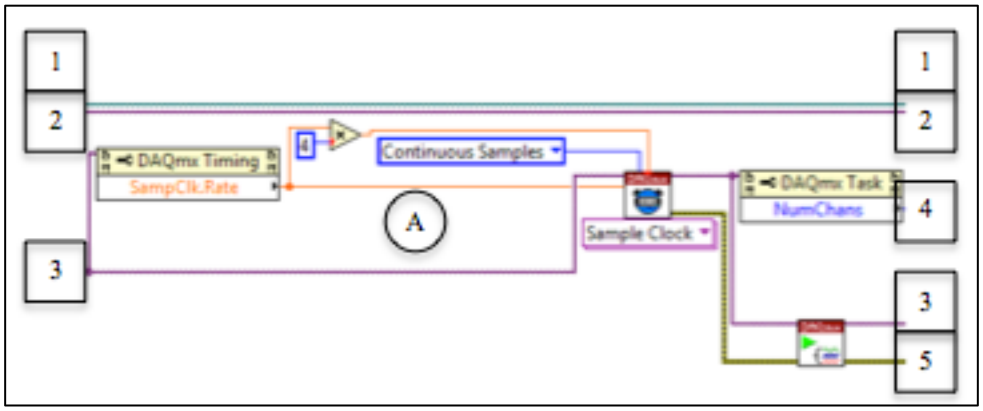
Loop A is the start of the "DAQ and Display Loop" VI. It only loops if First Run (C) is set to true which should only be the case if "Data Collection Main" is running "DAQ and Display Loop" for the first time since it was started. A contains the controls for the Input, Output tasks, and the Output log file path. These are hidden from the UI and are set by "Data Collection Main".

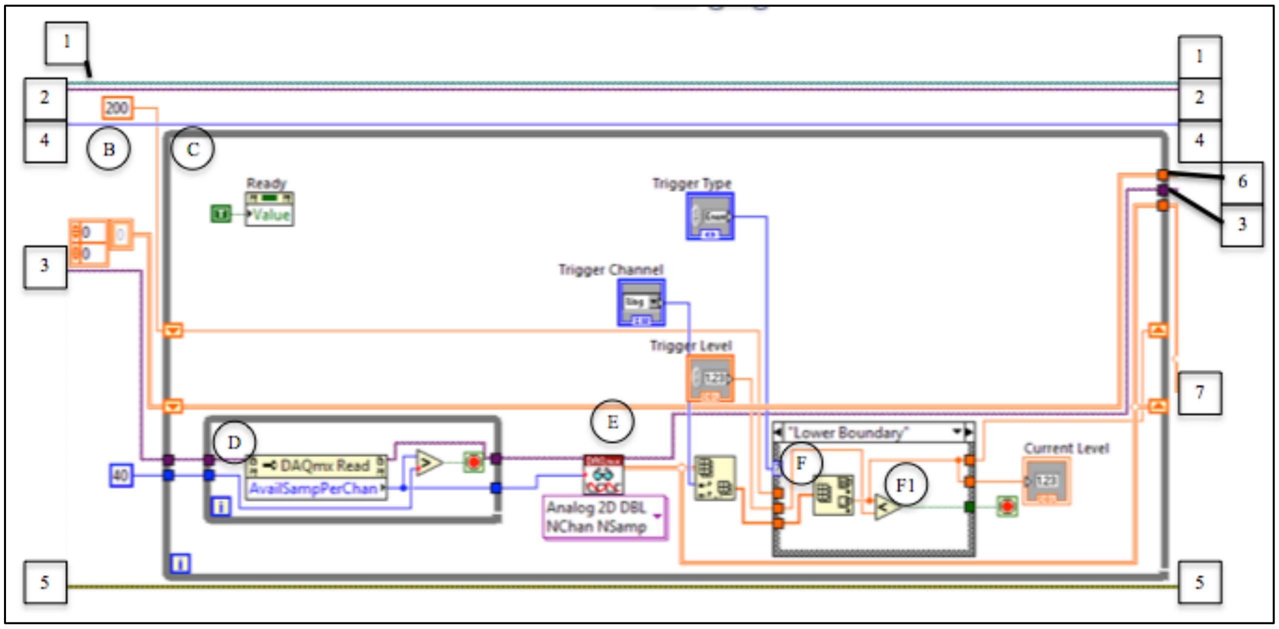
At B the program finds the names of the input task channels and sets them as the values of the Trigger Channel circle selector.

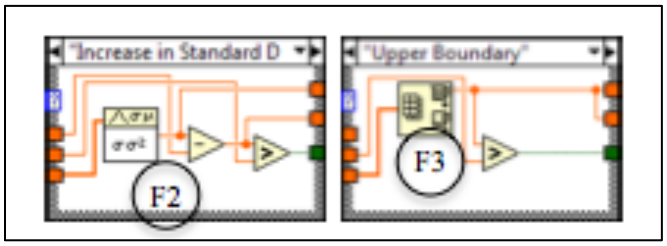
Loop D runs constantly during the program, waiting for UI events. It uses the "Next" button to dynamically register the event diagram at E.

The event diagram G has two cases, if a value change is signaled for the "Control Model" button it opens the "Control Tool" VI in a new window so that the control model in use can be changed and resets the "Control Model" button's value. If the "Next" button signals a value change, G resets the "Next" button's value. If "First Run" is true then the program changes first run to false using a signaling value invoke node and sends a false to the "stop if" in loop D. This ends loop A if it is running but does not end loop D. If "First Run" was already false, the program ends loop D by sending a true to the "stop if" in loop D.

At H the program turns off the "Ready" light each time "DAQ and Display Loop" is run.





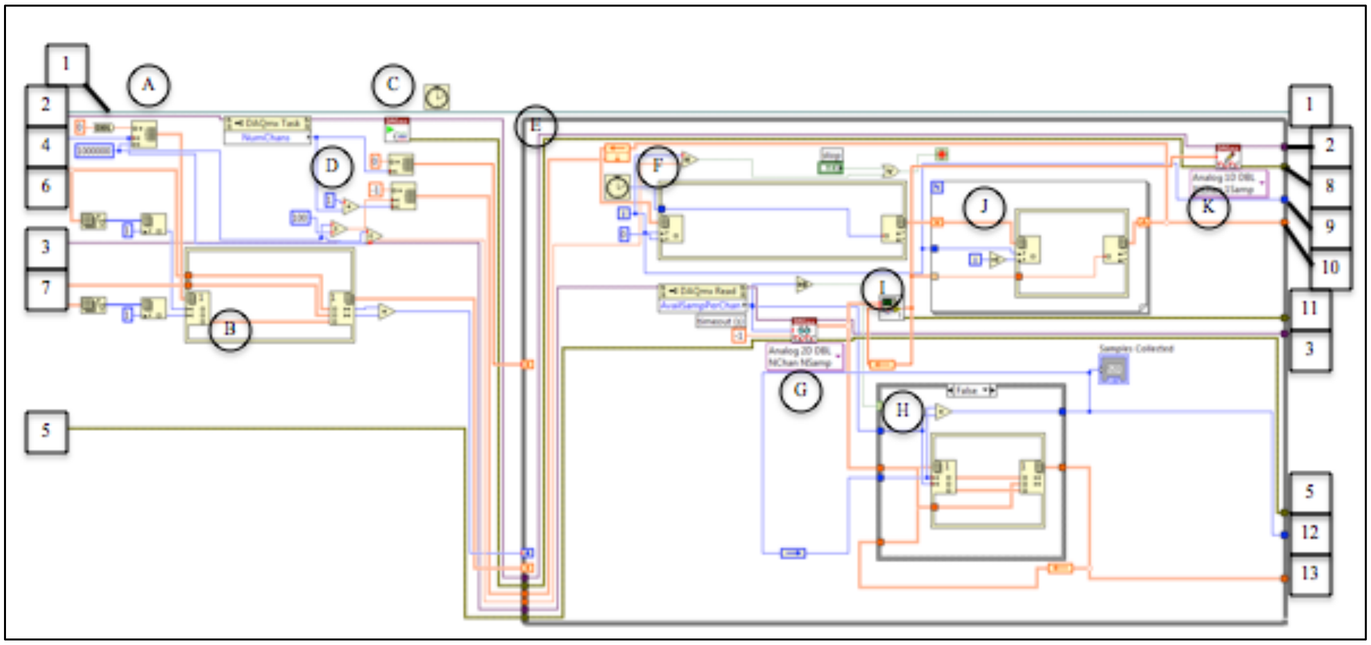


At A the program creates a sample clock for the input task with a buffer with length 4 times the sample click rate. After creating the clock it determines the number of channels to be used for instantiating arrays and calculations and then starts the input task.

At B the program initializes some constants that are needed for the trigger loop C. These constants are: a blank 2-dimensional array of doubles to be used to hold samples from the previous iteration of C, 200 the number to use for comparisons to prevent the "increase in standard deviation" trigger from always activating on the first iteration, and 40 which is the minimum number of samples per channel. 200 is a number chosen from the data used during testing of the program as it seemed unlikely that the noise in a sensor would be variable enough to have a standard deviation that high. 40 was also chosen based on testing as it guarantees a minimum of 40 data points prior to the first non-noise data point which was large enough for most sampling rates (tested up to 20kHz).

The trigger loop C loops until the sample data meets a trigger condition specified by the UI. At Loop D the program waits until a minimum number of samples (40) has been attained before reading and testing them against the trigger. At E the program reads all available samples from the input task. It passes the array of read samples into Case Diagram F.

Case Diagram F is responsible for analyzing data to determine if a trigger condition has been met. It has input for the previous value, the sub-array of samples for the channel specified in the UI, and the “Trigger Level” control value. It outputs the value for the “Current Level” indicator to show to the user, the value to compare the next iteration against, and whether the trigger condition has been met (true if it has been met). If F returns true, loop C stops. The three possible “Trigger Types” are F1, F2, and F3. To create new triggers a new case needs to be added to F and the “Trigger Type” enumeration control.



At A the array to store the data acquired is initialized. It is a 2 dimensional array (1,000,000 by the number of input channels) of doubles with the default value being 0.

At B the array created at A is loaded with the data collected while waiting for the trigger condition to be met. An in place sub-array replace structure is used which takes the empty array and the lengths of the two arrays generated by the trigger loop as inputs. This splits Array A into 3 sub-arrays. The first array starts at index 0 and has length equal to the length of the previous data array from the trigger loop. The second array starts after that array with length equal to the most recent data array from the trigger loop. These two arrays are replaced by the corresponding arrays from the trigger loop to preserve data prior to the trigger condition being met.

At C the generate signal task is started. And the Click Count block is used to initialize the first Click Count that is necessary to keep track of output timing. Each Click Count gets a number that is the amount of time that has past since some arbitrary time in milliseconds. The behavior of the reference time chosen isn't guaranteed but each click count in a program will use the same reference time. This means that for two Click Counts the greater number minus the lesser number will be the time in milliseconds between when each Click Count was called.

At D the number of channels in the generate signal task is used to initialize two new arrays. One array is a 1 dimensional array that stores the data output by the "Control Tool" sub-VI. The other array is a 2 dimensional array (10,000 by the number of output channels) and stores the history of the signal sent to the output task.

Loop E contains the main loop for data acquisition and control. Its basic process flow is: acquire all available samples, send them to "Control Tool", send the signal from "Control Tool" to the generate signal task, write the acquired and control data to their respective history arrays, repeat until user stops or the pre-allocated array is full (1,000,000/ number of input channels).

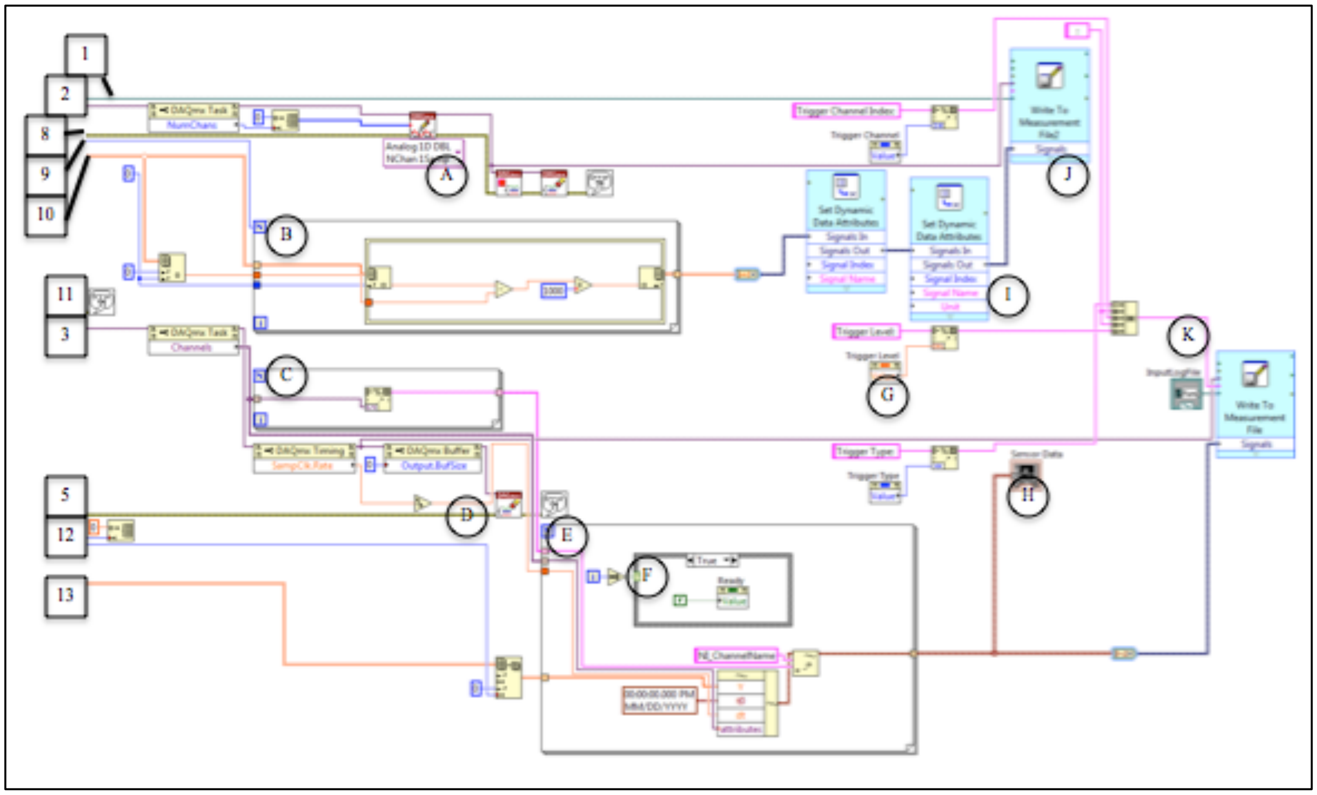
At F the program gets a Click Count for this iteration of the loop and adds it to the correct index of the control history array.

At G the program gets an array of all available samples. It sends it to Case Diagram H. When the number of available samples is greater than zero H adds the sample array to the acquired data history array and increments the number of samples collected counter by the correct amount. If the number of available samples is zero, H does passes the samples collected and array to the next iteration unchanged. The reason for this Case Diagram is that it prevents indexing errors and other miscellaneous errors caused by attempting to add empty arrays to the history array.

At I the "Control Tool" is called as a sub-VI. It is given the array it last returned and the array of the most recent samples collected. It outputs an array that contains the control data to send to the generate signal task.

At J the array for this iteration of the control is added to the control history array. This is a potential bottleneck as it uses a for loop to add the control data one at a time when an in place sub-array replace structure might be more time efficient. However, attempts to utilize such a structure while combining the data from the Click Count proved difficult.

At K the control array is sent to the generate signal task.



At A the program resets the output device to 0 by sending an array of length equal to the number of output channels with 0 at each index. The task is then stopped, cleared, and any errors reported in a pop-up box.

At for loop B the program converts the time values in the generated signal history array to millisecond values with the first time being zero.

The for loop C creates and populates an array containing all of the channel names for the input history array to be used for labeling the file output.

At D the input task is cleared and any errors are reported in a pop-up dialog.

Loop E creates an array of waveforms of the input data history. Each waveform corresponds to all of the data gathered for one channel. It labels the channels based on the array of channel names produced at C. The Case Diagram F is placed here to turn off the “Ready” light element only after data collection has been stopped. The array of waveforms is sent to the Waveform Chart on the front panel and is used for the log file.

G uses property nodes to determine what the trigger parameters were for this test and sends them to the Input Log File as a comment.

The “Set Dynamic Data Attributes” boxes at I are used to make sure that the data written to the Output Log File is correctly labeled.

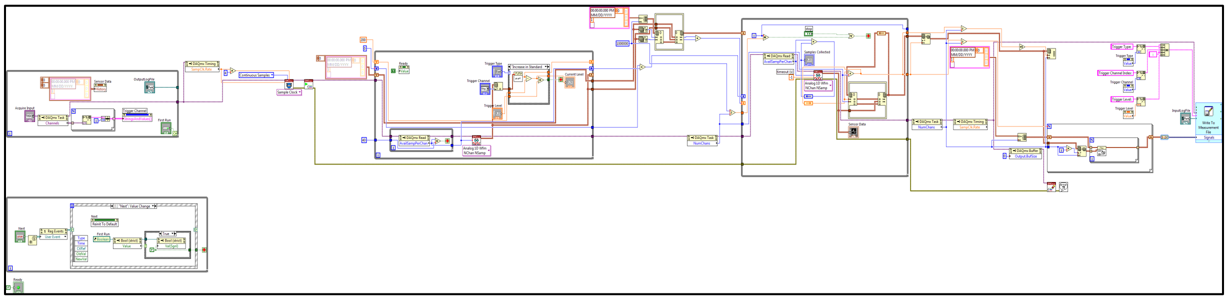
J and K write the data collected to the Output and Input Log Files respectively.

Once writing is finished the program waits for the user to press the Next button to end the “DAQ and Display” VI.

Connection Index:

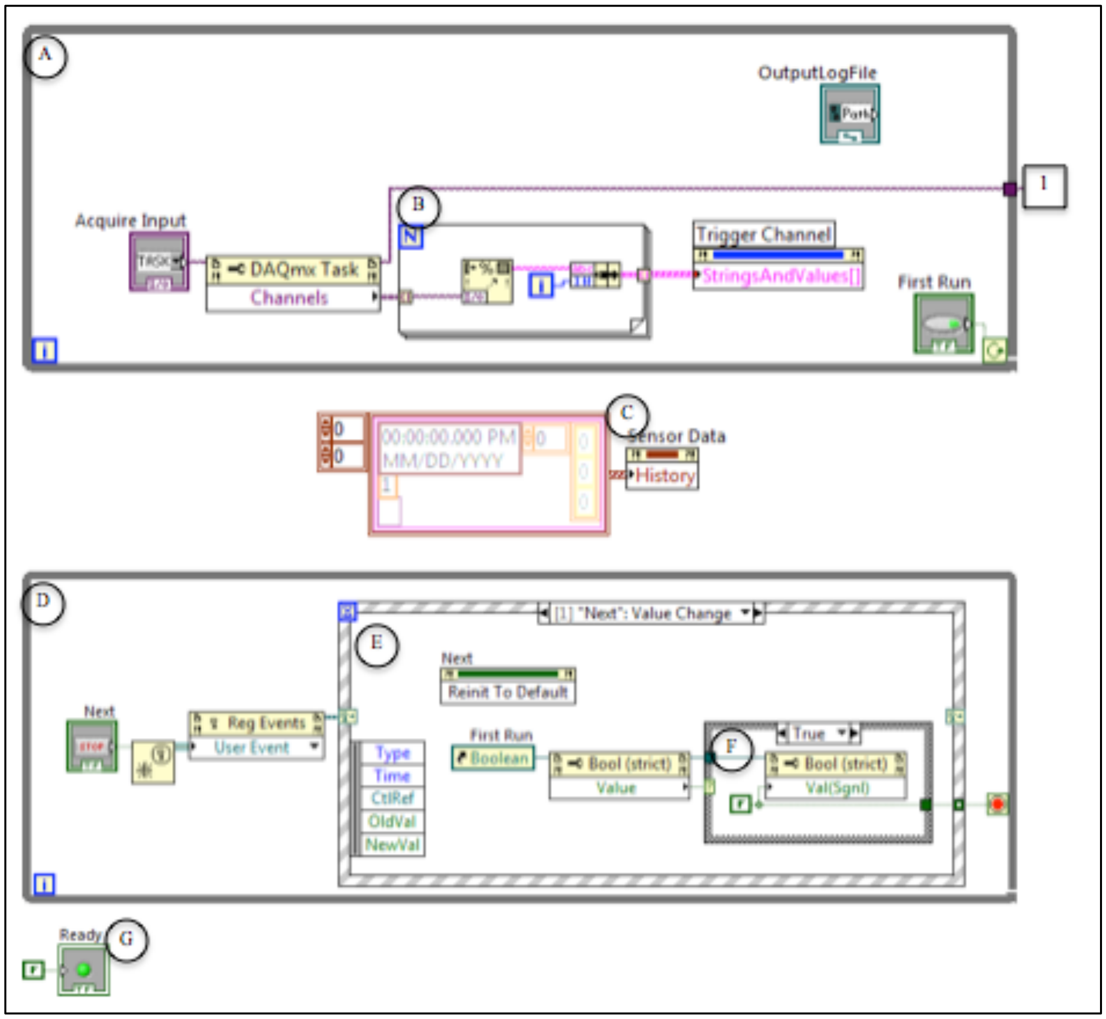
1. The file path for the Output Log File.
2. The task for sending the output signal.
3. The task for acquiring data.
4. The number of channels in the input task.
5. Errors caused by operations on the input task.
6. A 2-dimensional array of the samples collected in the previous iteration of the trigger loop. An empty array if the trigger activated on the first iteration.
7. The 2-dimensional array of samples read in the most recent iteration of the trigger loop.
8. Errors reported by sending a signal to the generate signal task.
9. Number of iterations of the Data acquisition loop.
10. The array storing the generated signal history.
11. Error reported by the “Control Tool” sub-VI.
12. Total samples read per channel from the input task.
13. The array storing all of the samples collected from the input task.

DAQ and Display Loop (Input Only)



VI Process Outline:

1. Controllers with the task information are set.
2. Wait until next is pressed.
3. Start the tasks. Perform initializations.
4. Read data and test for triggers until a trigger condition is met.
5. Add the data collected prior to the trigger to the final array for storing data.
6. Send the data to the main reading loop.
7. Read all available samples.
8. Store the read data.
9. Add the read data to the waveform graph on the UI.
10. Repeat 7 - 9 until stopped by the UI.
11. Concatenate the read waveforms into one waveform per channel.
12. Write the waveforms to a file.



Loop A continues to run until "First Run" is false.

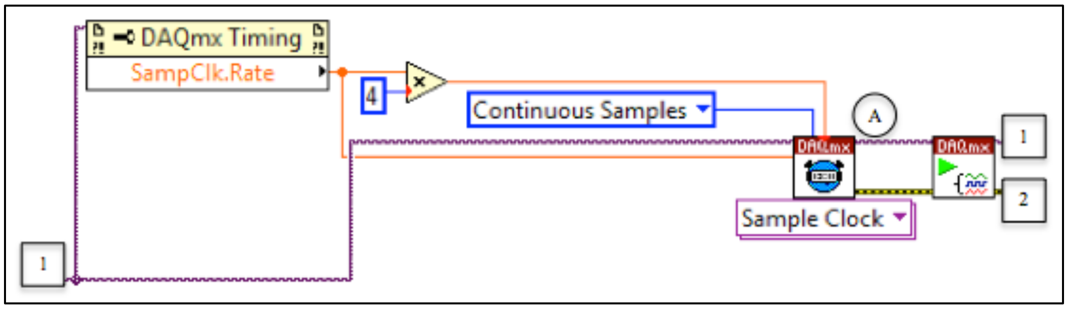
At B the names of the channels of the Input Task are used to set the Circle Selector used by the Trigger Loop.

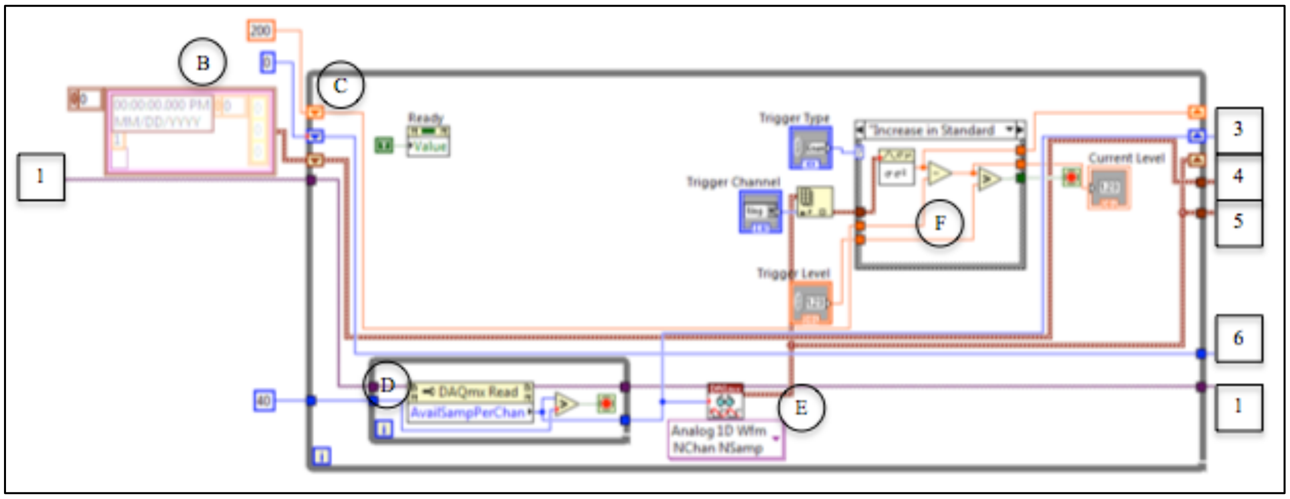
At C the program clears the waveform chart so that it doesn't become cluttered with data from previous tests.

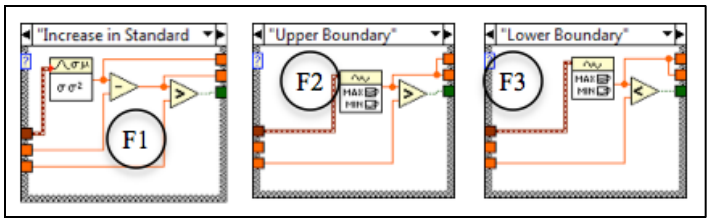
Loop D contains Event Structure E to handle changes when the user clicks the "Next" button.

Event Structure E has only one case, when the "Next" button is pressed it resets the "Next" button. After that it executes the Case Diagram F. If "First Run" is true then it sets "First Run" to false and outputs false causing Loop A to stop and Loop D to continue. If "First Run" is false then F outputs true which causes Loop D to stop. Loop A will never be running or will only run once more when "First Run" is false so the program doesn't need to try to stop Loop A if "First Run" is already false. The event structure E is used to make the “Next” button context sensitive, stop waiting to begin looking for a trigger if the VI is being run for the first time and end the VI otherwise.

G ensures that the “Ready” light is turned off at the beginning of the test.







At A the program sets the clock for the input task and starts the task. The clock is initialized with a buffer with length equal to 4 times the click rate.

At B the program initializes variables needed by Trigger Loop C. A constant array of empty waveforms is made to store the waveform data from the previous iteration of the loop. A 0 integer constant is created to store the number of samples per channel of that waveform data. The integer constant 40 determines the minimum number of samples per channel that the program waits for in each iteration of loop C. The integer constant 200 is an arbitrarily large value to prevent the "Increase in Standard Deviation" trigger condition from being met on the first iteration of loop C.

Loop C is the trigger loop. Like in "DAQ and Display" it collects samples and tests them against some trigger condition to determine when to start saving the acquired data. It outputs the task used for data acquisition, an array of waveforms for the most recent data set and an array of waveforms for the previous data set, as well as the number of samples per channel for both of those data sets. The number of samples is needed to get an accurate count of how many samples have been collected for the user and data processing.

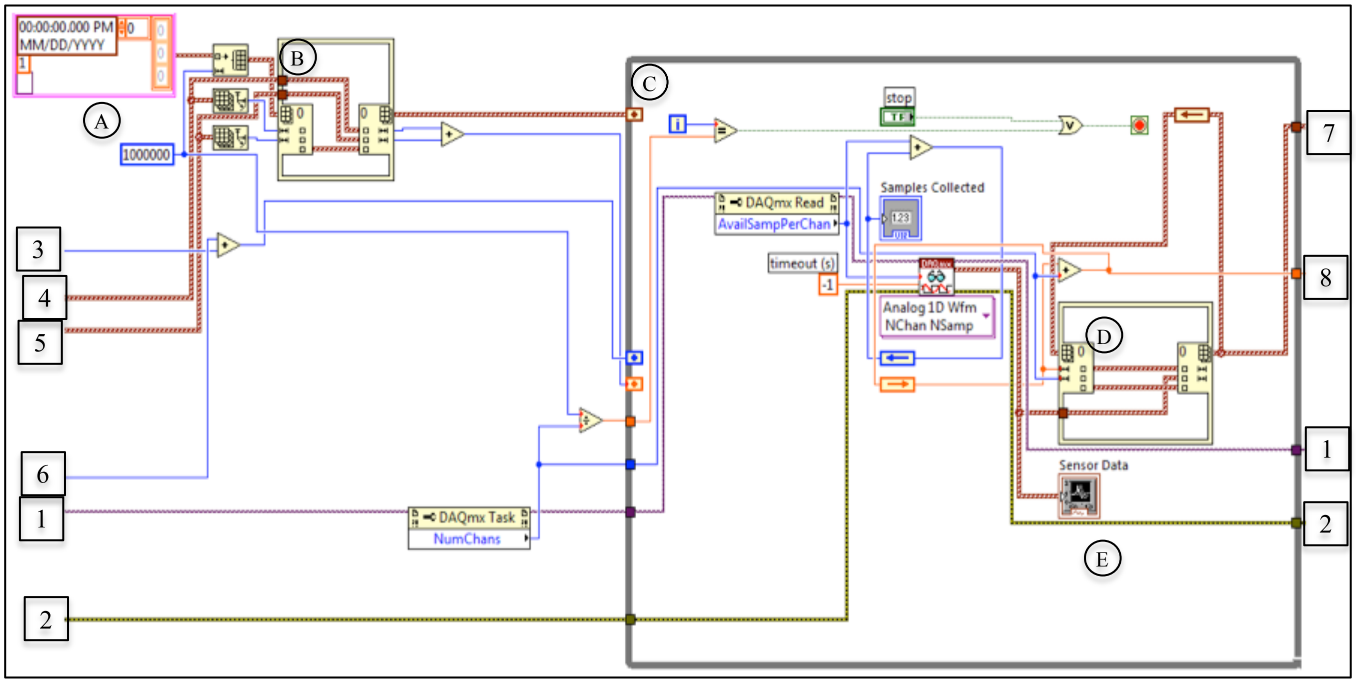
While Loop D waits until the number of samples available for the input task is equal to or greater than the specified integer constant, 40 samples in this case.

At E the program reads in all available samples and sends them to be tested for the trigger condition as well as to an output register and a shift register. The shift register is used to output the value of the input shift register. The input shift register is equal to the array of waveform data acquired in the previous iteration of C.

Case Diagram F contains the data for the different trigger conditions. To add another trigger a new case needs to be added to F and the corresponding name needs to be added to the “Trigger Type” enumeration control. F takes 4 inputs: “Trigger Type” determines which case to use, a waveform of the data from the channel determined by “Trigger Channel”, “Trigger Level”, and a double of the previous output of F. F has 3 outputs: a double to be given to the F in the next iteration, a double to be displayed to the user through the “Current Level” control, and a Boolean which is true if the trigger condition has been met. If the F outputs true then loop C stops other wise C continues.

F1 is an “Increase in Standard Deviation” trigger, it uses a Variance Statistics block to calculate the standard deviation of the waveform from the trigger channel and then subtracts the previous standard deviation and compares that to the “Trigger Level”. If it is greater than the “Trigger Level” it stops returns true.

F2 and F3 are the “Upper Boundary” and “Lower Boundary” triggers respectively. They both use the array max/min block to get the highest/lowest value in the waveform. If that is greater/less than “Trigger Level” then F returns true.

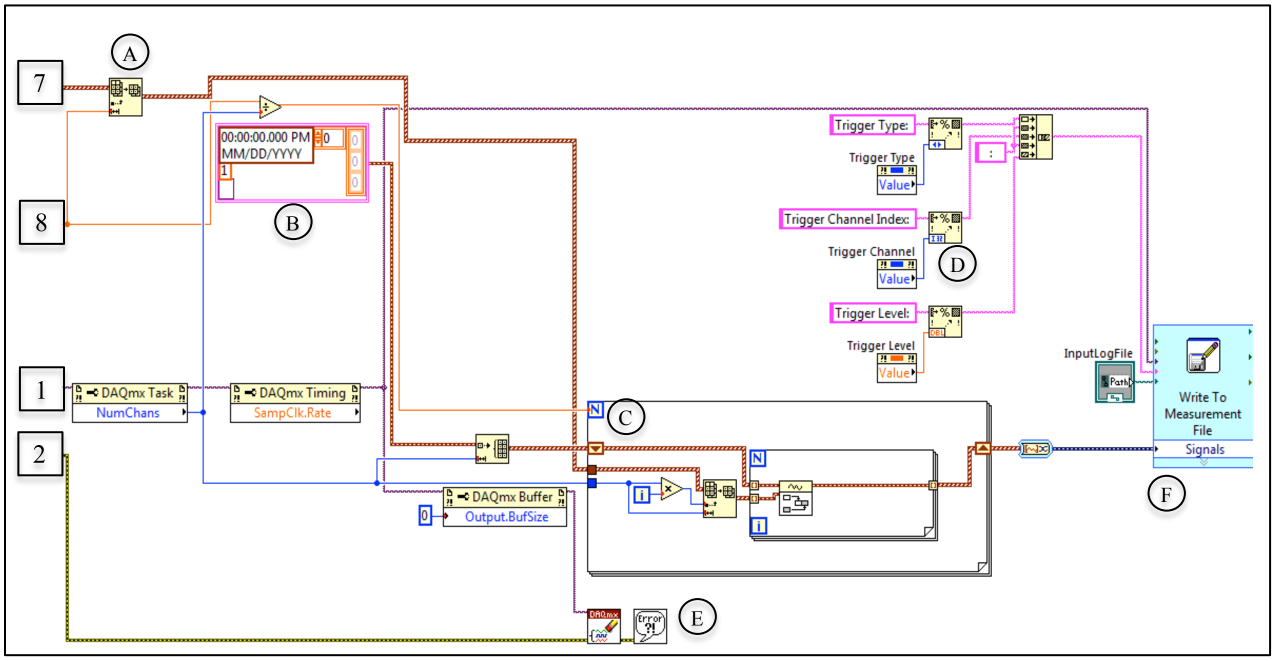


At A this code initializes the array to store all of the acquired data and preloads it with the data acquired from the trigger loop at B.

Loop C loops until the array is filled or the user presses the “Stop” button on the Front Panel. Unlike “DAQ and Display Loop” this displays the acquired data in real time using waveforms. This is to make display during collection easier and because, with no control function, the speed the program loops at is not a concern.

At D the waveform data collected is appended to the history array.

At E the data is displayed as it is collected, rather than after all collection is finished like in “DAQ and Display Loop”.



At A the program gets the filled portion of the input history array.

At B an empty waveform is initialized for storing the correctly formatted history data.

For Loop C iterates over the history array and converts the data into an array of individual waveforms containing the data gathered where each waveform is one input channel.

At D the data about which trigger was used is converted to a string to be added to the output file.

At E the data collection task is closed and any errors are reported.

At F the history waveforms are written to the input log file chosen in "Task and File Config". The trigger data is added as a comment at the first line of input recorded.

Connection Index:

1. The task for acquiring data as selected in "Task and File Config" and set by "Data Collection Main".
2. Errors returned by attempts to manipulate the input task.
3. The number of samples per channel collected most recently by the trigger loop.
4. The array of waveforms collected by the previous iteration of the trigger loop.
5. The array of waveforms collected by the most recent iteration of the trigger loop.
6. The number of samples per channel collected by the previous iteration of the trigger loop.
7. The total number of samples collected per channel.
8. The array of waveforms for the data acquisition history.