Aerodynamics Master Program Guide

For collecting and analyzing aerodynamic data.

By: Austin Scholp

Table of Contents

[Master Aero Program 2](#_Toc78201881)

[Purpose and use 2](#_Toc78201882)

[Block Diagram 2](#_Toc78201883)

[Incomplete Interruption Collection 4](#_Toc78201884)

[Purpose and USE 4](#_Toc78201885)

[THE BLOCK DIAGRAM 6](#_Toc78201886)

[Here is the EVENT handling loop 6](#_Toc78201887)

[Here is the Message Handling Loop 6](#_Toc78201888)

[Complete/Labial Intteruption Collection 10](#_Toc78201889)

[Purpose and Use 10](#_Toc78201890)

[The block diagram 11](#_Toc78201891)

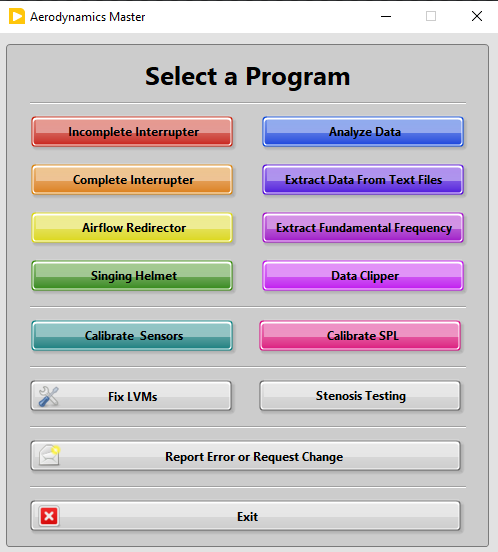
[Data collection and Saving loop 12](#_Toc78201892)

[State Machine/Trial control loop 13](#_Toc78201893)

[Balloon Control Loop 14](#_Toc78201894)

# Master Aero Program

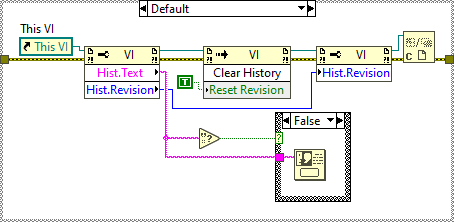
## Purpose and use

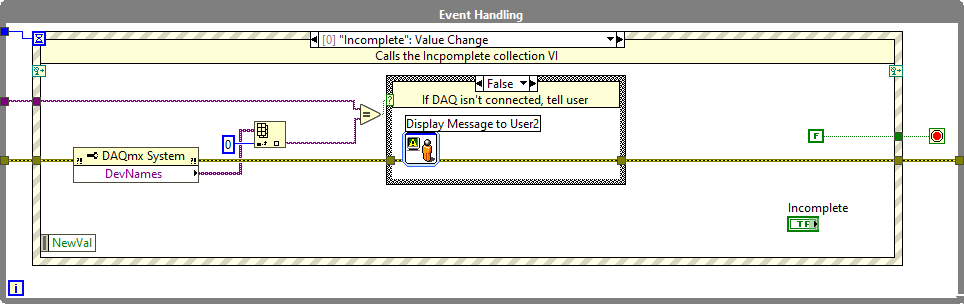
The Master Aero program contains all of the collection and analysis programs for our aerodynamic devices. Once opened, the user can select which program they would like to run.

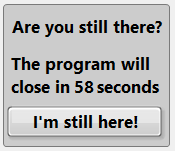
The Front Panel is simply an array of different buttons. Each one will open up its corresponding VI through the use of the event handling loop pictured below. Each button is handled by a different case that contains a VI. In the next couple of sections, we will go through each program individually.

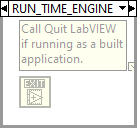
There is also an option to run the analysis program separately. To do this, run the VI or EXE called Analysis Master.

## Block Diagram

All LabVIEW programs run from left to right as seen on the block diagram. The first section of code in the VI checks to see what the most recent updates to the program and displays those to the user. This is contained in a Conditional Disable Structure. Everything inside this structure is only called when the program is from the LabVIEW project (i.e., when you are not using the executable or installed versions).

The rest of this VI is a simple event structure. It calls a SubVI for each of the buttons that are on the front panel. There are a couple other cases in the event handler that are not directly correlated with buttons on the front panel. Do not worry about these. They simply take care of different exit scenarios. *Do not edit these cases.*

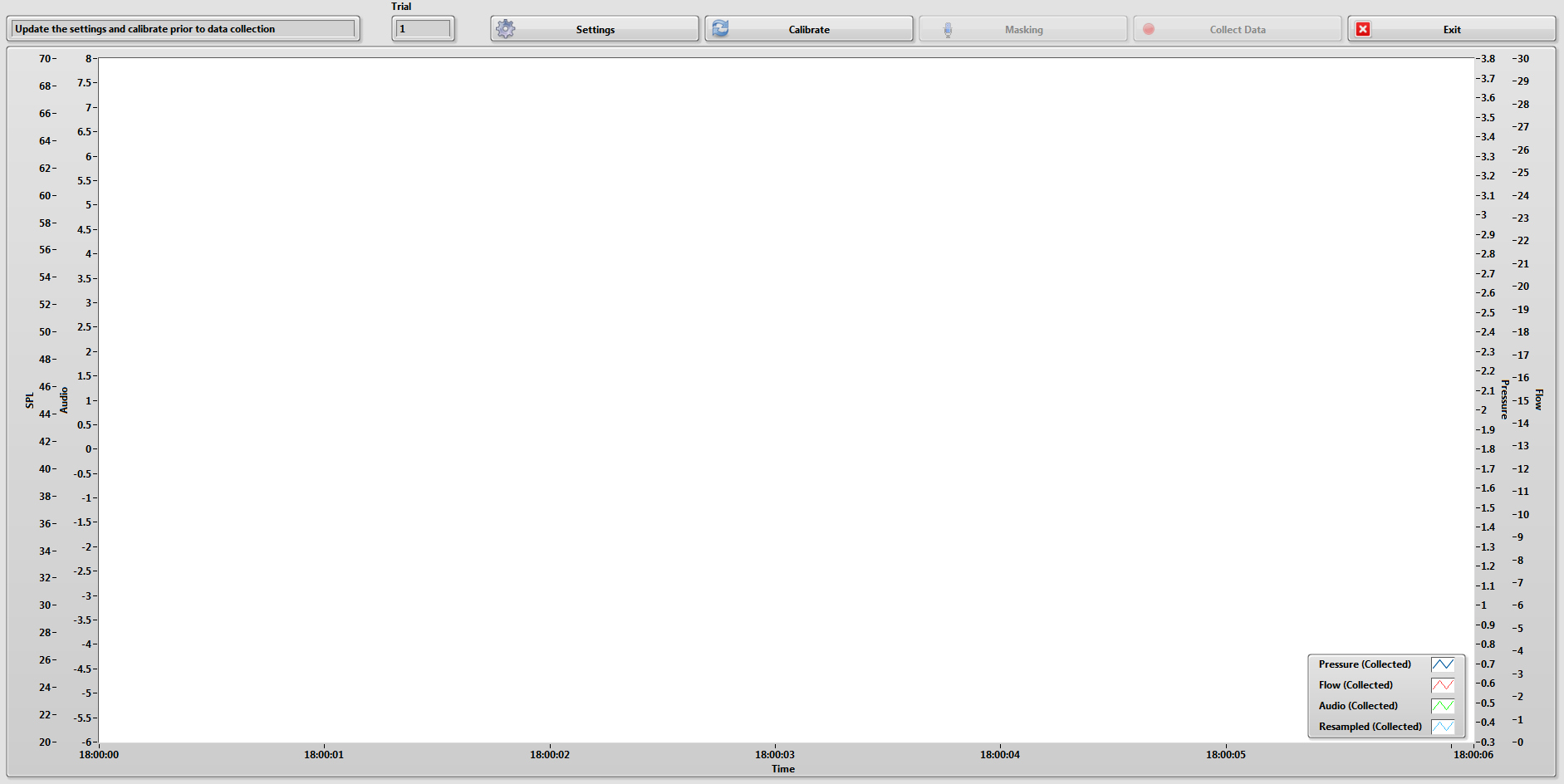
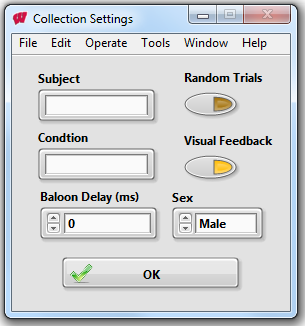
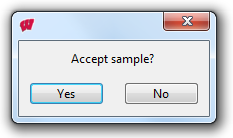
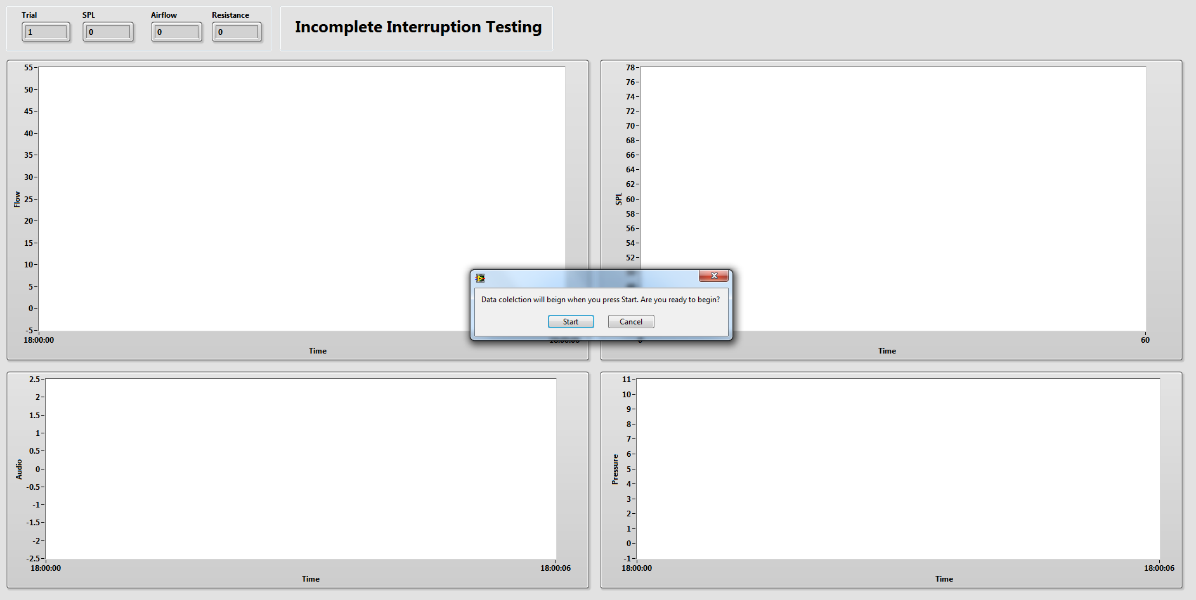
One of these cases is the Timeout case. It is called when the program does not detect any user input for two minutes (120,000 milliseconds). When triggered, the timeout SubVI will pop up with a countdown. If there is no input for another minute, the program will close.

 The final part of this VI is another Conditional Disable Structure. In this case, when the program is running as the executable or installed versions (RUN\_TIME\_ENGINE==True), it will close out of LabVIEW after the Event Handling loop stops. If this is not added to the end of a VI, LabVIEW will remain open even when you stop whatever program you are using.

# Incomplete Interruption Collection

## Purpose and USE

This program is designed to collect data through a DAQ board using the incomplete interruption device. If you do not know what that is, you should figure that out.

1. Open the program through the Master Aero program. You can also open it through LabVIEW if you would like, it is eaiser to use the installed program or the executable.
2. At first, the “Masking” and “Collect Data” buttons are disabled. This is because you must update your settings prior to creating an audio mask and collecting data.
3. Once you open settings, you can set:
   1. the subject number
   2. the condition to be tested
   3. how long the balloon will be inflated during the interruption
   4. whether or not you will be doing randominze resistences
   5. whether or not you want to have visual feedback
   6. and the sex of the subject
4. After you set your…uh…settings, you can now create an audio mask and collect data. While you do not have to make a mask, most trials will be using audio feedback. If no mask is made, there will be not feedback.
5. To create a mask, press the “Masking” and a window will pop up. Once you are ready to create the mask, press “Start” and have the subject say “Ah” into the microphone for about 3 seconds. You should hear a tone played back to you that should match the subject’s pitch.
6. If the playback pitch did not match, just hit “No” and the program will start listening for a new pitch. You can repeat this as many times as needed until you find a comfortable pitch for the subject (The first time usually is fine though).
7. Now you are ready to collect data…so click “Collect Data”
8. Assuming you left visual feedback on in the Settings window, the Data Collect window should look something like the image below. Once you click “Start”, the program will begin collecting flow, pressure, and acoustic data.
9. The program will record for a set number of seconds. At the time I am writing this, it is set to 6 seconds (this is a constant that we’ll get to later). This might have changed based on the needs of the project.
10. The program also automatically trigger the balloon valve to fill for a single interruption. This also might have changed to suit the needs of the project.
11. After it is done recording, the program will prompt you asking if the trial was acceptable, assuming nothing went wrong, you can click “yes” and more prompts will follow regarding saving the data. This part is pretty self-explanatory.
12. Once saved, you will be prompted to run another trial. If you select not to run again, the trial counter resets to 1. It is easier to run multiple trials under 1 condition.
13. Repeat ad nauseam.

## THE BLOCK DIAGRAM

If you need to change something on this program, please review this prior to making edits to the LabVIEW files.

This program is set up in a queued message handler (producer/consumer) structure. If you do not know what that means, please refer back to the section of this binder that covers program structures (advanced tab).

The program contains 2 parallel loops: the Event Handling Loop and the Message Handling Loop.

### Here is the EVENT handling loop

This loop contains the event structure. If this phrase confuses you, go review what an event structure is. This structure has 8 cases. Cases 0 through 3 and number 6 correspond to events where a button on the front panel is pressed. The rest handle application/panel closing. These are just different exit situations that you should not change.

### Here is the Message Handling Loop

This loop has a case structure with 13 cases for different messages sent in. You’ll find descriptions for each case on the following page.

Note: These cases are not in the order as they appear in the case structure, but in an order of how they would be called in a typical run of the program.

1. **Initialize** – Sets control references for each of the items on the front panel. These are then bundled into a cluster so that can be used later. Also, the next two cases are added to the message queue.
2. **Initialize Data** – Sets the default values for all of the data to be stored between cases.
3. **Initialize Panel** – This sets the front panel objects to their default states.
4. **Update Display** –Sends a new message to the UI to display to the user. It also updates the data display and trial number with the most recent info added to the data queue.
5. **Calibrate** – This opens up the sensor calibration SubVI. Calibration values are then added to the data array. The Update Display case is called again.
6. **Settings ­**­– The settings SubVI is opened and the user selected info is added to the data queue. Also, the Collect and Masking buttons are enabled. The Update Display case is called again.
7. **Masking** –Opens the audio mask setup SubVI and saves the frequency found to the data queue. The Update Display case is called again.
8. **Collect** – First, the exit button is disabled then the collection SubVI is opened. After the data is collected, it is taken from the collection SubVI and input into the title string creator and the save data SubVI. User input messages are also presented to control what to do after collection.
9. **Exit** – The message queue is released and the loop is stopped. All other loops are also stopped.
10. **Error** – If an error is thrown from the error detection SubVIs in the Event or Message handling loops, this case will be called. Right now, this just stops the loop.
11. **Confirm Quit** – If I wanted to double check if the user wanted to exit, I would do that here. Right now, there is no check, the loop is just stopped.

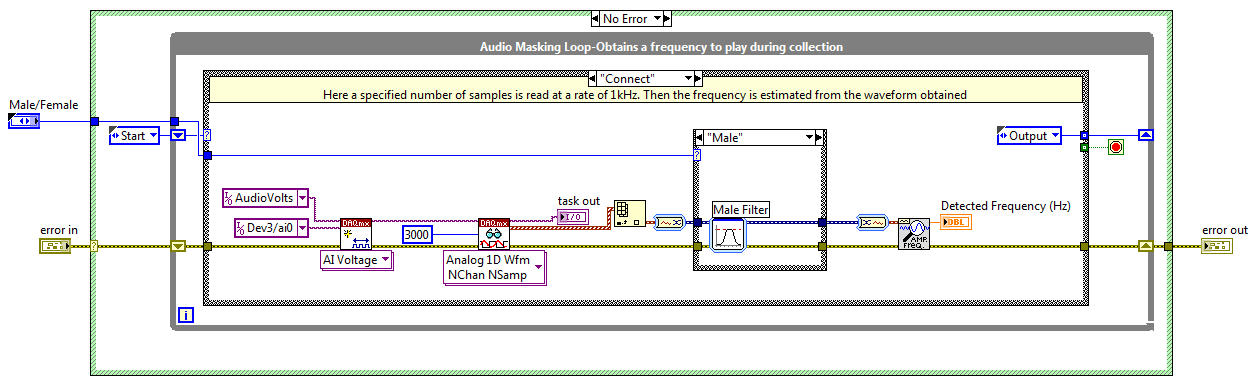
#### The Settings SubVI

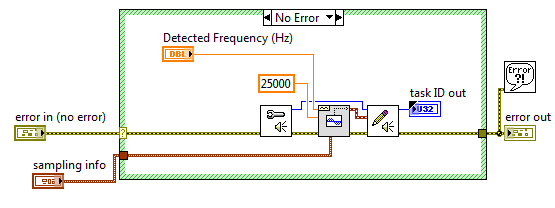
The settings VI is set up in a while loop that waits for the user to press the OK button. When that happens, the values the user inputs through the front panel are output from the SubVI to the controlling collection VI.

The random trials button currently controls the default condition string. This feature is not currently used.

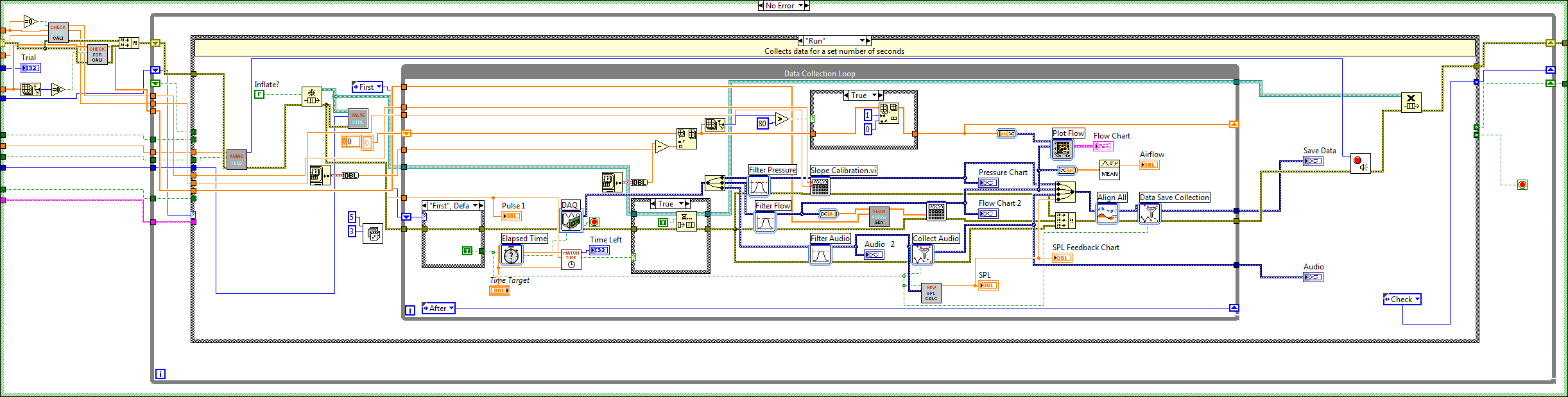
The sex selection is used during the masking SubVI to control filter settings. This is not saved anywhere else.

#### The masking Subvi

This SubVI is set up in a simple state machine. The cases are outlined below:

* **Start** – Contains a while loop that stops when the user hits Start through the One Button Dialog. The Connect case is called next through the shift register.
* **Connect** – 3000 samples at 1kHz are read in and the frequency of the signal is calculated. There are also 2 filter options (male and female) that make the calculated frequency more accurate for different subjects. The output case is called next.
* **Output** – The detected frequency is put into the Sound Out SubVI, which simply outputs a sound with the detected frequency to the speakers. The check case is called afterwards.
* **Check** – Prompts the user to accept the sample. If it is accepted, the Quit case is called, if not, it will return to Connect.
* **Quit** – Stops the loop.

#### Collection SUBVI

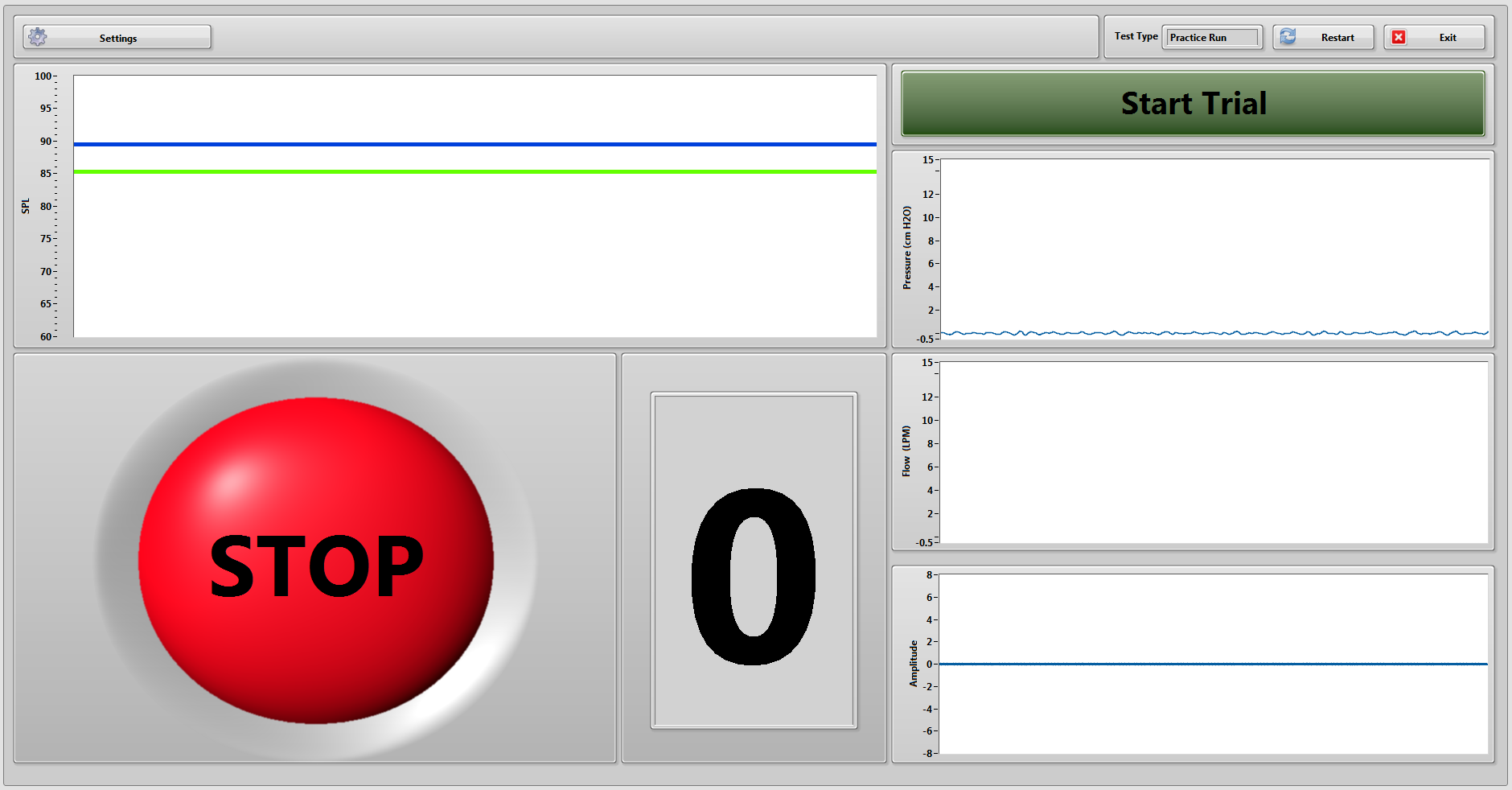
This SubVI is set up in a state machine structure. I made this one first, that is why is different from the complete interruption and redirector collection programs. I could remake it so it is cleaner and more similar to the others, but that takes time that I am not going to spend right now. Anyway, the cases are listed on the next page.

* **Start** – Initializes the UI objects to their default values. If visual feedback is turned off, that will hide the charts in this case. There is also a prompt for user input. The user can either start the trial or exit the program.
* **Run** – Collects data from the DAW board for a set number of seconds. There is only one interruption set for this program. Once the elapsed time matches the randomly generated number, the balloon will be triggered. Once the time target is met, the check case is called next.
* **Check** – The user is prompted to retry the collection trial, or accept and save it. If accepted, the Exit case will be called. If not, the start case is called.
* **Exit** – Exits the loop.

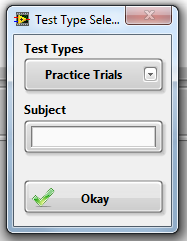
After the data is collected, the Collection SubVI is exited and data is output to a couple other SubVIs that create the file name and then save it to an LVM file. If you know how case structures work, the rest of this case is pretty self-explanatory. Also, the data collected is displayed on the main front panel.

# Complete/Labial Intteruption Collection

## Purpose and Use

This program is used to collect data with the complete mechanical interrupter or through labial interruptions. It is one of the four main collection programs in the Master Aero controller. There are four charts displaying live data. SPL is shown with an upper and lower cursor to be used as a target range. Pressure, flow, and acoustics are also displayed.

|  |  |  |
| --- | --- | --- |
| Test Type | Delay | Interruptions |
| Practice | 0 | 0 |
| Mech Mask | 250 | 5 |
| Mech Mouth | 250 | 5 |
| Labial Normal | 0 | 0 |
| Labial Quiet | 0 | 0 |

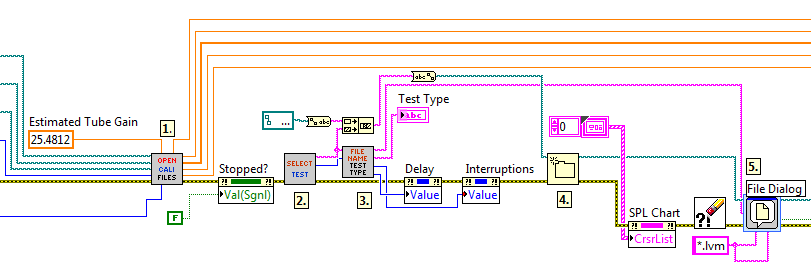
This program will be opened through the Master Controller VI. At first, the user will be prompted to select the test type and enter subject info. The different test types will control what the default configuration will be when data starts being collected. It also changes the default name and save path for the saved data. The settings themselves are hidden by default. They can be viewed by clicking the Settings button. Hover over each control to view a short description.

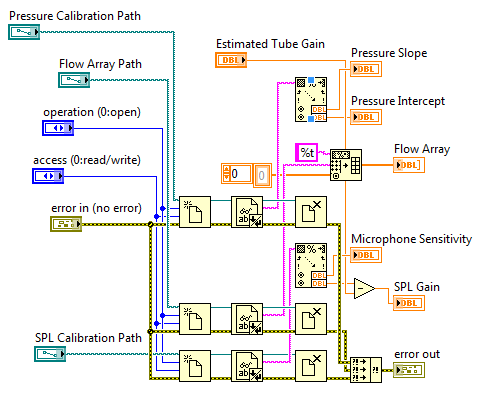
After the test type and subject info is entered, the program will start reading (but not recording) in data from the DAQ board. Flow, pressure, and acoustic data are read in directly from the DAQ board while the SPL is calculated from the acoustic data.

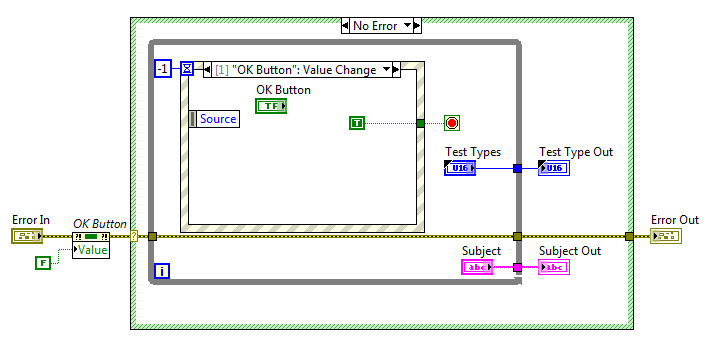
Data will only be saved to file during a trial, while the stoplight is green (GO). Once you click Start Trial, the button itself will turn red and now say stop trial. You can stop the recording before the trial is complete by pressing this button again. Assuming you want to run a complete trial, there will be a 3 second countdown before the stoplight turns green and starts writing collected data to a file.

Unless you click the stop trial button, the recording will stop automatically after however many seconds you set the trial length to be. If you wish to select a different trial type, you can click the Reset button and the test type select window will pop up again. If you are done collected, click exit.

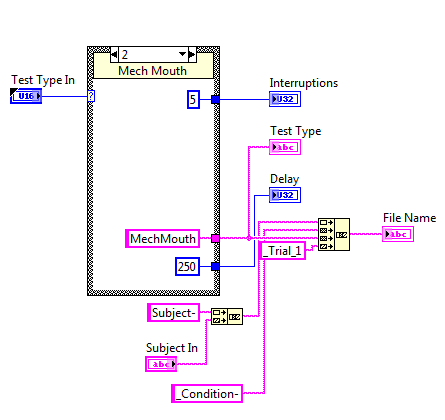
## The block diagram

This program is organized into three parallel loops within a master control loop. Before the parallel loops, there are a couple of SubVIs that get calibrations values, and set the default interruption values.

First, there is the Open Calibration Files VI. This takes in the paths for the pressure, flow array, and SPL calibration text files then opens them to extract the data as spreadsheet strings. These are then read to obtain their values as doubles (an array of doubles in the case of the flow array).

For the SPL gain, the microphone sensitivity is read along with the estimated gain from the … gain estimator. The estimated tube gain was calculated experimentally and is a constant that is then subtracted from the gain estimation to give us our actual gain. This value is then used in the SPL calculator later in the program.

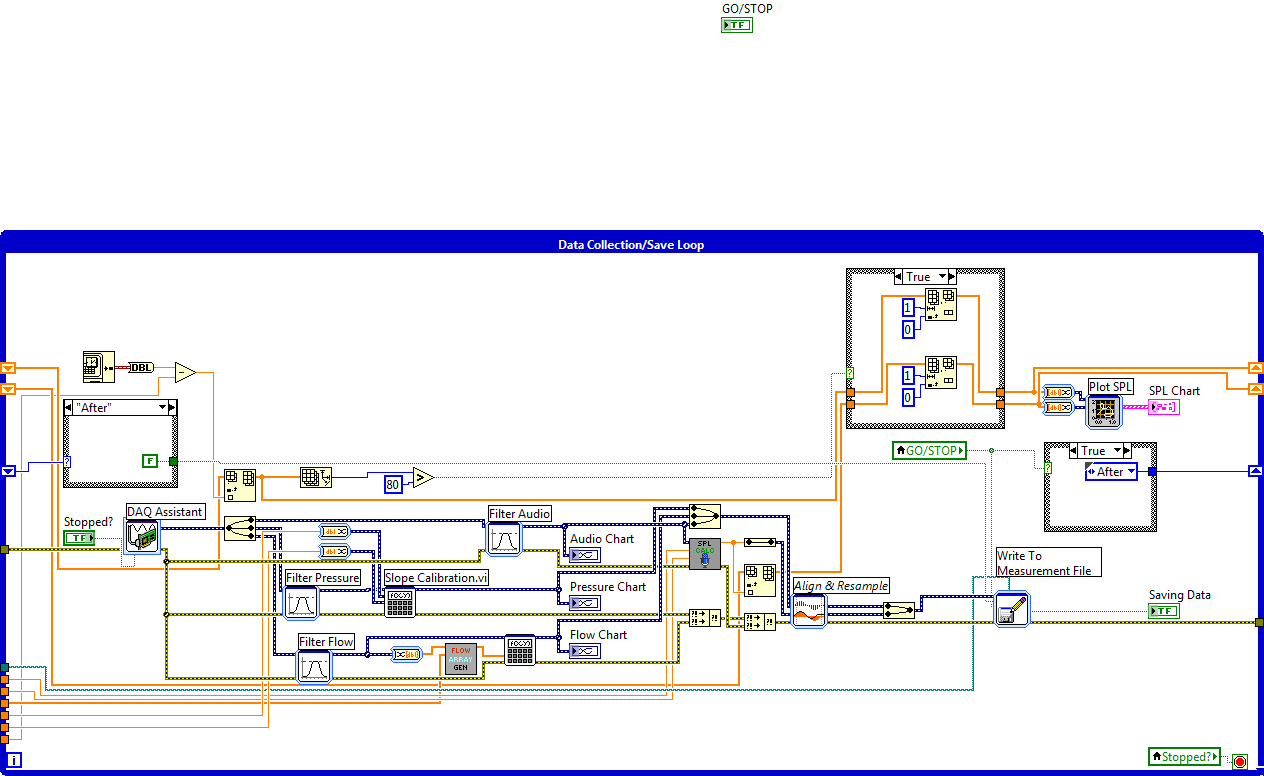
Second is the test type selector. It simply outputs the selected test type and subject info string when the user click the okay button or hits enter.

The third SubVI takes in the test type and subject info from the selector and creates a file time based on this information. It also outputs the default balloon delay and number of interruptions, depending on which test was selected. These two values are set through the two property nodes following this SubVI.

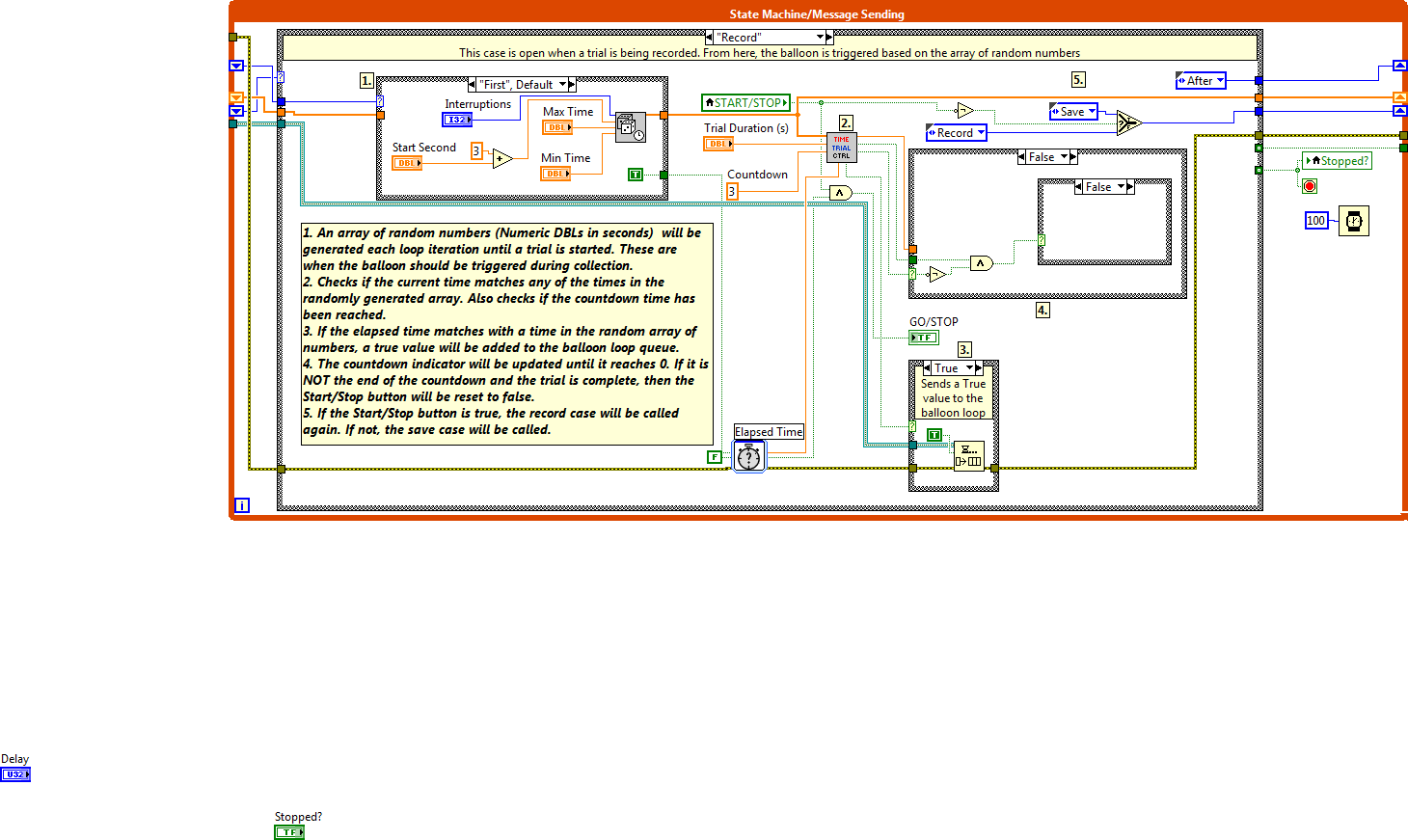
A folder is then created for the specific subject name/number. Currently, he base save path is to the peds\_data folder on my desktop. This will need to change once I am gone and my user account is no longer usable. The path itself is <C:\Users\scholp\Desktop\peds\_data\subject-> The new folder will simply be the same name/number that is input into the Test Selection SubVI. If the folder exists, an error is thrown by the create folder function. This error is then cleared by the Clear Errors VI that follows.

Through the file dialog express VI, the user is prompted to confirm the save location of the data that will be saved in this session. The user can change the location here if desired.

### Data collection and Saving loop

Here is the data collection/save loop. It reads from three channels on the DAQ board using the DAQ Assistant. It takes in acoustic, pressure, and flow data from their respective devices. Data is then filtered, and run through calibration curves to obtain actual values. Other things to note are the pair of “First-After” case structures. This this sets the reset value on the Write Measurement to file VI unless the GO/STOP light is on. Also, the x-axis of the SPL graph is controlled by building an array of values that is limited to 80 points.

### State Machine/Trial control loop

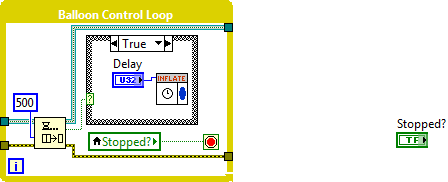
This loop is an event-controlled state machine. This controls the GO/STOP light, as well as when the balloon interruptions are triggered. A short description of each case is below.

* **Initialize** Sets the GO/STOP button to false (Red/Stop) and then calls the **wait** state.
* **Wait** Contains the event structure. The next state called depends on the event.
* **Record** Has the controls for the random array of numbers that will be used as interruptions. It also controls when to stop the trial and sends triggers to the Balloon Control Loop. Finally, it controls the stoplight and the START/STOP button.
* **Save** A previous version of the program utilized this state, but it is no longer called.
* **Exit** Stops the State Machine loop, which in turn stops the other two loops. This will also stop the master control loop.
* **Reset** Stops the State Machine loop, stopping the other two internal loops, but does not stop the master control loop, which then goes through another iteration of the loop, starting back with the calibration files SubVI.

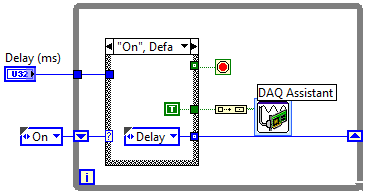
I mentioned an event structure. Here are the events:

* **START/STOP** Whenever this button is pressed, the record case is called. Whether or not data is recorded is controlled through the **Record** state.
* **Exit Button** calls the **Exit** case when the exit button is pressed.
* **Restart Button** calls the **Restart** case
* **Settings** Depending on if the settings button is true or false, the trial settings will become visible.

### Balloon Control Loop

Within the **Record** state of the trial control loop, if the current trial time matches one of the times in the randomly generated array of number, a true value will be added to the Boolean queue. When this happens, the case structure within the balloon control loop changes to true and the balloon is inflated for however many milliseconds the delay control is set to.

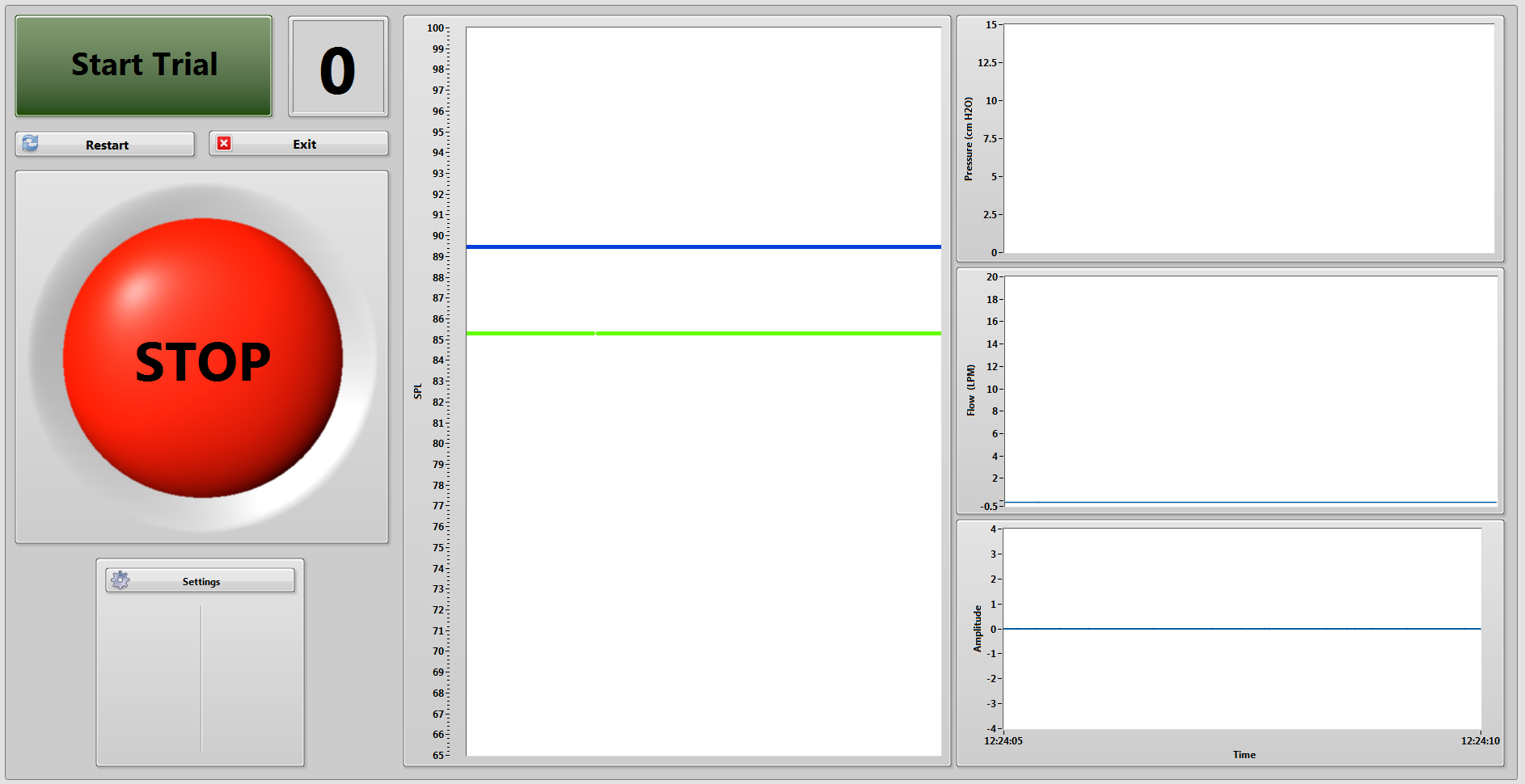
#### Inflate Subvi

The balloon interruption length is controlled by delaying the loop iteration by the input number of milliseconds.

First the “On” case is called which sets the DAQ digital output to True. In the “Delay” case, the loop waits for a given number of milliseconds before stopping the loop.

# Airflow Redirector Collection

## Purpose and Use

This program is used to collect data using the airflow redirection device. Programmatically, this works the same way as the complete interruption program. The main differences are in the data save locations, and the front panel layout.

I probably did not need to make this write-up page. Just look at the complete interruption section and you will be able to understand how this one works.

# Singing Helmet

