Cardiff School of Engineering

EN3100 Project - Cover Sheet – 2022/23

Module Details

Module Name: Project Module No: EN3100.

Project Title:

Design of a Novel Data Acquisition System
for Cardiff University's Gas Turbine Research Centre

Supervisor(s): Dr Dan Pugh

Personal Details

Name: Tim Atkinson Student No: C2071196

Personal Tutor: Dr Debajyoti Bhaduri Degree:- BEng Mechanical Engineering

Declaration

I hereby declare:

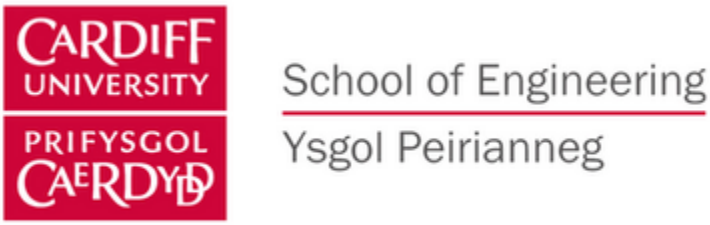
that except where reference has clearly been made to work by others, all the work presented in this report is my own work;
that it has not previously been submitted for assessment; and
that I have not knowingly allowed any of it to be copied by another student.

I understand that deceiving or attempting to deceive examiners by passing off the work of another as my own is plagiarism. I also understand that plagiarising the work of another or knowingly allowing another student to plagiarise from my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings against me.

Signed: Tim Atkinson Date: 31st March 2023

Total number of words = 9758

Total number of pages = 59



Design of a Novel Data Acquisition System for Cardiff University's Gas Turbine Research Centre

Tim Atkinson – C2071196

BEng Mechanical Engineering

Supervisor: Dr Dan Pugh

March 2023

Abstract

Cardiff University's Gas Turbine Research Centre uses an outdated data acquisition (DAQ) system for its primary experimental apparatus, the High-Pressure Optical Combustor (HPOC). Although reliable, the system's UI is limited to live display and has cumbersome data export capabilities. It was identified that a LabVIEW based DAQ could be used as a replacement.

The suitability of LabVIEW for creating a DAQ was explored through iterative development. The new program improves on existing functionality and implements new UI elements, such as real-time graphing and combustion calculations. The code makes use of state-machines and a producer-consumer design pattern. Testing showed that the program was capable of acquisition and export at higher sample rates than required, and the combustion calculations were validated.

The project concludes that a LabVIEW based data acquisition system could be suitable for the GTRC. Further investigation of hardware integration would be required to ensure full compatibility with the HPOC.

Contents

1 Introduction

1.1 Aims

The aim of this project is to explore the suitability of LabVIEW as a replacement for the data acquisition and display system in use at Cardiff University's Gas Turbine Research Centre (GTRC). To achieve this a new data acquisition system will be designed in LabVIEW, with the goal of demonstrating LabVIEW's capabilities. The scope of the project does not include hardware integration at GTRC.

A new DAQ system must be capable of capturing flow, pressure, and data and to display this data in real time and then save it to file. The new system must be more than a replacement, as there is plenty of room for improvement. LabVIEW's capabilities go beyond the functionality of the existing system. These additional capabilities should be explored.

1.2 Objectives

To achieve the aims of the project the following objectives were set:

- Research
    - Find out the functionality and design requirements.
        - Essential Functionality
        - Non-Essential Functionality
    - Understand LabVIEW as a programming environment.
- Preparation
    - Organise the project.
    - Setup LabVIEW for use on this project
    - Visit GTRC
    - LabVIEW Basics
        - Become familiar with the software and the resources available.
- Design
    - Code architecture
    - Front Panel
- Development
    - Further improve LabVIEW skills with test cases.
    - Build a functional prototype DAQ.
        - Following the feature list, starting with the essential functionality.

1.3 Background

GTRC is a multipurpose facility with a host of measurement techniques in use, carrying out research into a range of different Combustion and Energy Systems. First opened in 2007 it is used for the generation of original data for model validation or development (Bowen, et al., 2009). Some of the major projects currently undertaken at GTRC involve de-carbonisation research funded by FLEXIS (Cardiff University, n.d.) , Storage of Ammonia For Energy (SAFE) aims to use Ammonia in the combustor to reduce NOx emissions (Valera-Medina, 2019), and multiple collaborations and grants focused on aviation caused pollution and emissions. (H2020 Raptor (863969)) (Hochgreb, 2020) (RAPTOR (863969)). The full list of GTRC publications can be found on the centre's website https://www.cu-gtrc.co.uk/content/27.

Specific to this project is the High-Pressure Optical Combustor (HPOC). Experiments with the HPOC are referred to as test-campaigns.

Measurements are taken within the input lines, at the inlet itself, within the combustion chamber, and in the hot-end simulator. These are then fed to an existing DAQ system in the control room at GTRC, where they are monitored live throughout the test and recorded. The conditions being monitored by the DAQ System are temperature, mass flow rate, and pressure.

A range of other diagnostic techniques are also in use such as Emissions Sampling, Flame Chemiluminescence, etc but these facilities are not integrated directly into the existing DAQ system, and not used for direct control of the experiment. (Gas Turbine Research Centre, n.d.)

The existing system is a bespoke SCADA system built using Pro-Pack running on Windows 3.1. It's robust and simple. During a test, it displays a range of measurements within the control room at 1 Hz, whilst saving the data locally.

Figure 1 is a picture taken of the display screen in the control room at GTRC. Due to the age of the operating system this software is running on, a high-quality screenshot is not possible.

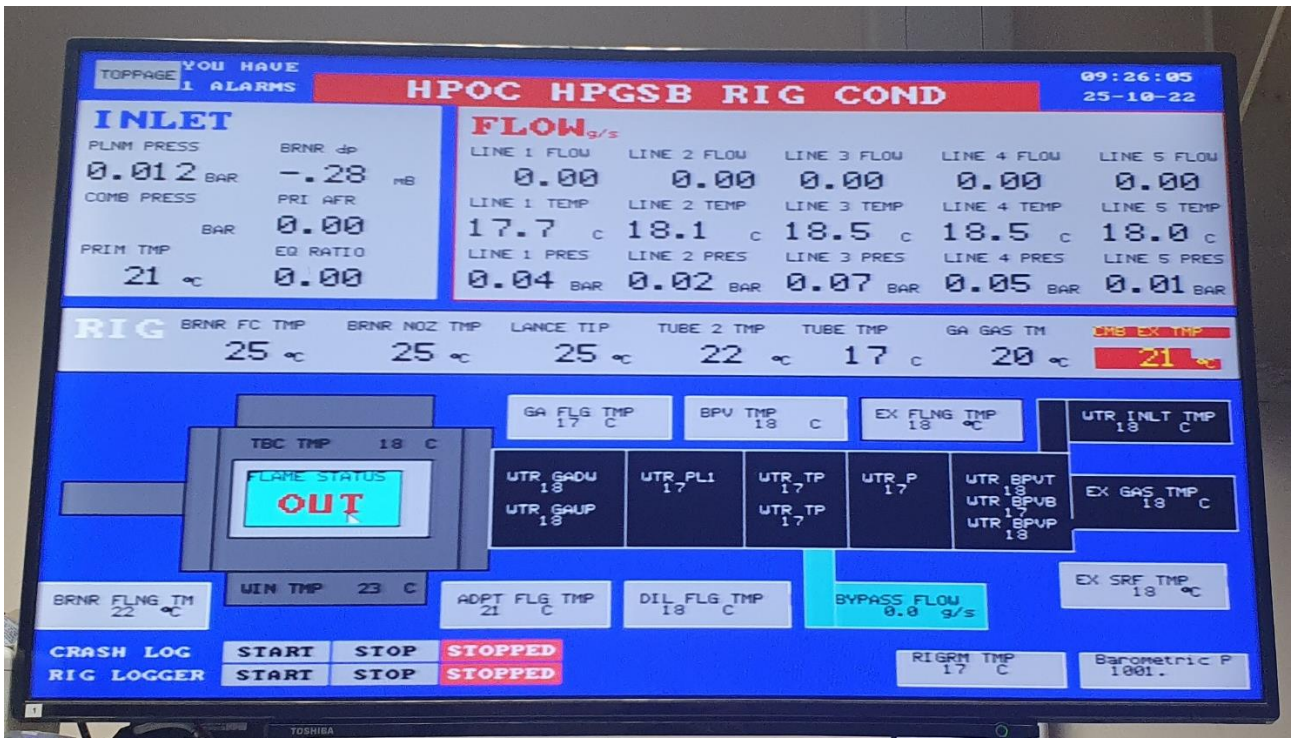


Figure 1 - Existing Systems UI

The system is outdated and limited in features. There is some concern about the age of the hardware running Pro-Pack, memory and storage are limited. A persistent issue is the live Air/Fuel Ratio and Equivalence Ratio Calculations are currently non-functional. This is not due to a technical fault, these required manual configuration within the code each time the test conditions change. In addition, a Thermal Power display does not exist.

Limitations of the existing system slows research carried out at GTRC. During a test campaign EQ Ratio and Thermal Power are used as driving variables. Without indicators for these, the values are estimated based on the Flow conditions. An improved system will increase the efficiency and speed of test-campaigns carried out with the HPOC.

A range of different hardware is used for each type of measurement, each with their own sample rates and uncertainty. K and R type Thermocouples, Quartz Pressure Transducers, and Coriolis mass flow meters are used to measure temperature, pressure, and flowrate respectively.

1.4 What is LabVIEW?

LabVIEW was identified as a suitable replacement for Pro-Pack by the team at GTRC. Developed by National Instruments (NI) LabVIEW is a graphical programming language used to build Virtual Instrument Interfaces (VIs) and Data Acquisition Systems (DAQs). Officially LabVIEW is the development environment, and the code produced is known as ‘G’ or ‘G Dataflow’, although code produced in LabVIEW is colloquially referred to as ‘LabVIEW Code’ or as ‘Block-Diagram’. (LabVIEW Wiki, 2019)

LabVIEW is a good choice for this project. It is flexible and supports a wide range of hardware, as one of the industry standards it’s compatible with many existing DAQ solutions. Hardware integration in LabVIEW is simple, with a wide range of built-in user interfaces and real-time data processing tools. Lastly there is significant support and an existing codebase for data acquisition, there is no shortage of materials or resources to learn LabVIEW.

The 2018 SP1 32bit version of LabVIEW was used for this project as it was the version available to students at the School of Engineering. Once completed the LabVIEW code was compiled into a standalone executable.

Alternative DAQ Software is discussed in LabVIEW for Data Acquisition Section 11.1 (Mihura, 2001), although given the age of this book the recommendations are likely to be out of date.

2 Literature Review

2.1 Self-teaching materials for LabVIEW

The LabVIEW programming language was unknown to the author at the onset of the project. All LabVIEW skills developed are entirely self-taught. Therefore, a review of some of the materials used has been included.

2.1.1 LabVIEW Core 1, 2, and 3

NI run a series of courses known as Core 1, 2, and 3 aimed at newcomers to LabVIEW. To be used alongside these courses are the participant guides/course manuals, containing the taught materials and exercises. For this project only the manuals themselves were available.

Core 1 (NI, 2014) focuses on the fundamentals, used as an introduction to the programming environment and the language. Briefly covering basic data acquisition and simple single loop programs. Good practice is described but not explained.

Core 2 (NI, 2012) serves to bridge the gap between the fundamentals taught in Core 1 and the high-level system design focused on in Core 3. Making use of asynchronous communication methods and the implementation of design patterns, finishing with basic file I/O tools. These concepts were applied directly during the development phase.

Core 3 (NI, 2013) aims to move beyond the practical coding skills taught previously and teaches the process of software development and bringing code to production with a team.

These manuals are meant to be secondary to the online classrooms used to deliver the course. In the absence of these interactive elements the manuals are lacking in regard to the understanding of how or why. This is compounded by the available manuals being out of date and across a range of different versions.

2.1.2 Available Textbooks

LabVIEW for Data Acquisition (Mihura, 2001) – Bruce Mihura

A dense textbook written by a former NI employee. This book is full of advice for programming in G and has in depth examples, Bruce Mihura demonstrates an exceptional understanding of LabVIEW's tools. It contains sections on DAQ hardware and transducers. If hardware integration had been pursued in this project this book would be invaluable.

Unfortunately, the book was published in 2001 and has not had a new edition since. Whilst the sections on fundamental LabVIEW skills hold up, there is no mention of design patterns, queues, parallel loops or TDMS. Many of the VIs in the book are out of date, LabVIEW having introduced improved functionality since 2001.

Hands-On Introduction to LabVIEW Third Edition (Essick, 2016) – John Essick

An excellent introduction to the programming environment, this book leans heavily on the use of images from LabVIEW itself. It builds code step by step explaining why the code has been employed. Often giving multiple solutions to common problems. The book is designed to have stand-alone sections covering specific aspects of LabVIEW programming. This book was an essential tool when implementing state-machines and the DAQmx palette.

2.1.3 National Instruments Product Documentation & Support Centre

The Support Centre contains detailed articles on how to code using the LabVIEW development environment. The Product Documentation Centre is an online version of the inbuilt help panel present in LabVIEW, it contains a detailed breakdown of the terminals and behaviour of all block diagram nodes.

The support centre typically provides the bigger picture on how to implement functionality, while the Product Documentation details specifically how to achieve this goal using LabVIEW's functions. Both are powerful and reliable resources, as such they been referenced throughout this report.

2.1.4 YouTube as a Learning Resource

The use of follow-along tutorials and demonstrations on YouTube was useful in understanding LabVIEW's programming environment. Observing real-time programming in LabVIEW accelerated the self-teaching process.

Many videos are published by NI themselves. Such as 'Programming Data Acquisition Applications with NI-DAQmx Functions' (NI Global, 2010) which lays out how the DAQmx palette can be used

to acquire data. Complemented by Zachary Neale (Neale, 2019) who demonstrates the use of the palette for multiple channels of data, but developing it further by implementing a timing mechanism, live graphing, and basic file output. Functionality included in this project's development.

A possible pitfall of learning programming with this method is whilst they provide guidance on how to code, the technical knowledge on why is often missed. Zachary Neale's demonstration of file output and a timing mechanism is functional but not best practice.

The expertise in these videos can sometimes be outstanding like Tom's LabVIEW Adventure (McQuillan, 2018) who is a certified instructor for NI, holding LabVIEW Architect Status (McQuillan, n.d.).

However often these resources are made by less experienced individuals, casting their efficacy into doubt, and without peer review it's unclear what can be relied upon.

These resources were most useful at the start of the project when the challenge was learning how to code within LabVIEW. As the project progressed these sources were dropped in favour of LabVIEW's official documentation and community forums.

2.1.5 LabVIEW's Community Forums

Solutions to bugs and problems encountered when programming can often be found on the community forums. Users can post requesting help with their code. NI awards titles to users based on their certifications and their contributions to the forums to help give an idea of the quality of advice given. Most difficulties when programming have been encountered before as such there is often a forum topic already devoted to this issue.

These boards are most useful when trying to perform a specific action within LabVIEW and can be useful when trying to understand the mistakes in code. A specific example of this was trying to understand the 'Write to Spreadsheet' VI, a user provides an excellent breakdown of common mistakes and tips for when using this VI. Solving the original problem and providing some advice in the process (nepomnyi & Bob_Schor, 2022). This post contributed when trying to build the file export structure detailed in Section 0 of this report.

This collaborative space allows users to discuss the topic and often compare different solutions. Whilst any one post can be unreliable as a source, seeing the topic discussed can give the reader a better understanding of LabVIEW code.

2.2 Related Journal Articles

Feel-soon Kang (Feel-soon, 2008) developed a monitoring system for a combined cycle gas turbine in LabVIEW. Like this project, data acquisition, display, and storage were implemented. The system was tested and found to be accurate and reliable. Though the contents of this article are pertinent and could be useful for this project, the official version of this document is a scan of a printout. The examples of block-diagram code presented are illegible, and the text of the article does not explain the code in detail. A higher quality publication could not be found.

LabVIEW has been used for high-speed acquisition of temperature, rotational speed, and vibrational signals from a Gas Turbine (Wei, et al., 2020). Making use of LabVIEW's signal processing capabilities for fault detection. An article from Trends in Analytical Chemistry (KrauB, et al., 1999) examines LabVIEW's potential use for building Virtual Instruments and advocates it's use for non-time critical programs. The block diagram code presented in the article is poorly designed, despite the extensive use of SubVIs it's difficult to follow, appearing to incorrectly use timing mechanisms.

3 Methodology

3.1 Organisation

All the work for this project was carried out on the author's personal laptop. To keep the project organised a Microsoft Teams was created. This served as a place to keep any documentation, but also a location to back-up LabVIEW code.

Scope creep can be an issue when developing software, to help combat this a Feature Tracker was created. This contained a list of requirements and features for the code, ranked by importance.

Once Main Branch Development began, change control was implemented. A notebook was used to record detailed information about each iteration. Changes, problems, potential solutions, and programming considerations were all noted down.

There are 39 separate iterations of the software, beginning with the experimentation with Producer-Consumer Loops. Ending with version 6.3 in Mid-March. These iterations do not include failed attempts at implementing code, as an iteration would continue to be worked on until it met the objectives for that iteration, at which point it would be saved and a new version created. The development process best fits the Spiral model detailed in Section 1-7 of LabVIEW's Core 3. (NI, 2013)

All LabVIEW code can be found on GitHub here:

https://github.com/atkinson-t/GTRC-DAQ-for-Dissertation

Early in the project a visit to GTRC was organised to gain a better understanding of the facility and the work done there. Pictures of the control room and the UI being used were taken.

3.2 Effective LabVIEW Programming

Building well-structured code in LabVIEW is difficult. The nature of the development environment can sometimes result in poor but functional code. Whilst most programming norms apply, there are good practices exclusive to LabVIEW and its visual dataflow language.

Good LabVIEW code is organised, modular, and easy to follow. It's also efficient, the code must execute fast enough to keep up with data acquisition.

LabVIEW has a style guide available (NI, 2023). This is a detailed list of rules to follow to ensure your front-panel and block-diagram are well organised and easy to understand. As the complexity of the software grew these guidelines were key to keeping it organised. Some examples are keeping diagrams flowing from left to right, arranging parallel loops from top to bottom, using clusters to keep data organised etc.

There is an add-on toolkit for LabVIEW called the VI Analyzer Toolkit. This can be used to perform static code analysis and automatically check it against the style guide mentioned above. Unfortunately, this toolkit is not readily available for any previous versions of LabVIEW (NI, 2021).

LabVIEW Core 2 Section 5 (NI, 2012), details how to refactor code in LabVIEW. It raises the idea of overly complicated logic, many of the design iterations detailed later in this section of the report demonstrate how when a new feature is implemented it is often done so with overly complex code.

Local and Global Variables should be avoided in LabVIEW for a variety of reasons, they're a deviation from dataflow (Mihura, 2001) potentially causing a race condition. They are also slower to execute and involve more overhead than wires (NI, 2023). Local Variables have been used sparingly in this project; state-machines have been used to avoid race conditions.

There are many potential pitfalls when programming within LabVIEW and good practice extend beyond what has been discussed in this section. Recommendations from LabVIEW's official documentation and NI's articles has been followed to try ensure the code is of high quality.

4 Development

The project is split into two phases, early development where the focus was on learning and understanding LabVIEW code, and later development where quality code was iteratively developed into a finished build.

As only the later development made its way into production, the early development has not been discussed in detail for the sake of brevity.

4.1.1 Mimic

A cosmetic recreation of the existing systems UI was made, referred to as the Mimic. Shown below in Figure 2. This featured very basic file output capabilities, and alarm thresholds. This served to demonstrate LabVIEW's cosmetic functionality, and as an early goal for learning how to use the LabVIEW programming environment. The block diagram code used to achieve this was very poor and could not be described as a DAQ system.

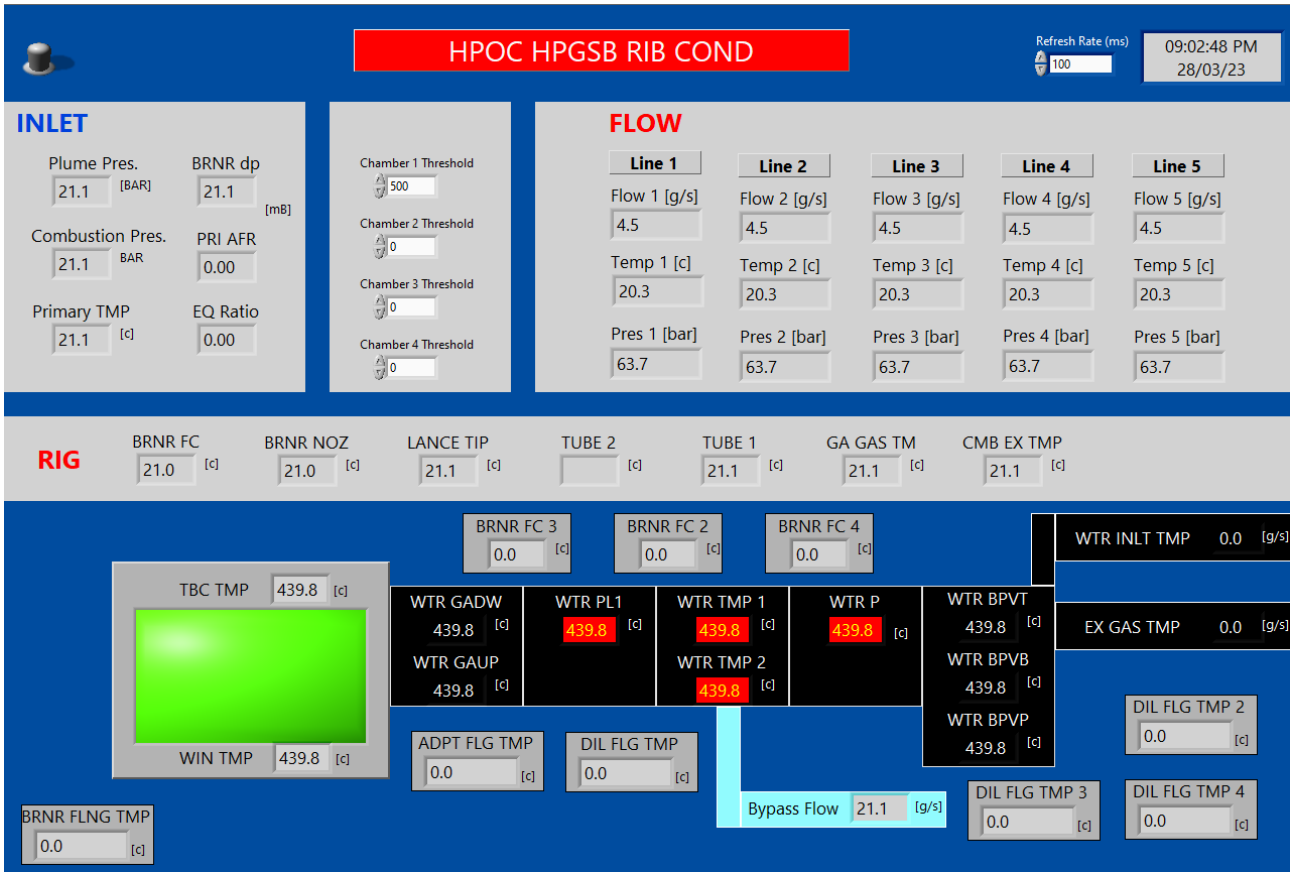


Figure 2 - The Mimic, a LabVIEW recreation of the existing user interface

4.1.2 Producer Consumer Tests

Producer-Consumer Loops are a commonly used design pattern. The producer loop is solely responsible for acquiring the data, it can asynchronously share the data with other loops running in parallel known as consumer loops.

This design pattern is commonly implemented in LabVIEW using the queue operations palette, this allows the VI to add data to a buffered list of elements. Which can then be accessed elsewhere in the block diagram. Queues operate chronologically using a first-in first-out principle. Allowing for lossless transfer of data between loops.

The Producer-Consumer design pattern has significant advantages for data acquisition. The producer loop can acquire the code at a fixed rate determined by the system, whilst the consumer loops can process the data at a variable speed. Decoupling data acquisition from data processing simplifies implementation of processes like export and live display.

To explore the Producer-Consumer design pattern a series of VI's were built. The first VI built for this purpose is pictured in Figure 3. This VI has a simple DAQmx setup to measure data from simulated signals within NI. This DAQmx read task is then fed into the queue. The queue is used to transfer the DAQmx task between the loops. When the producer loop is halted, the Dequeue element node outputs the error in the Consumer Loop causing it to stop.

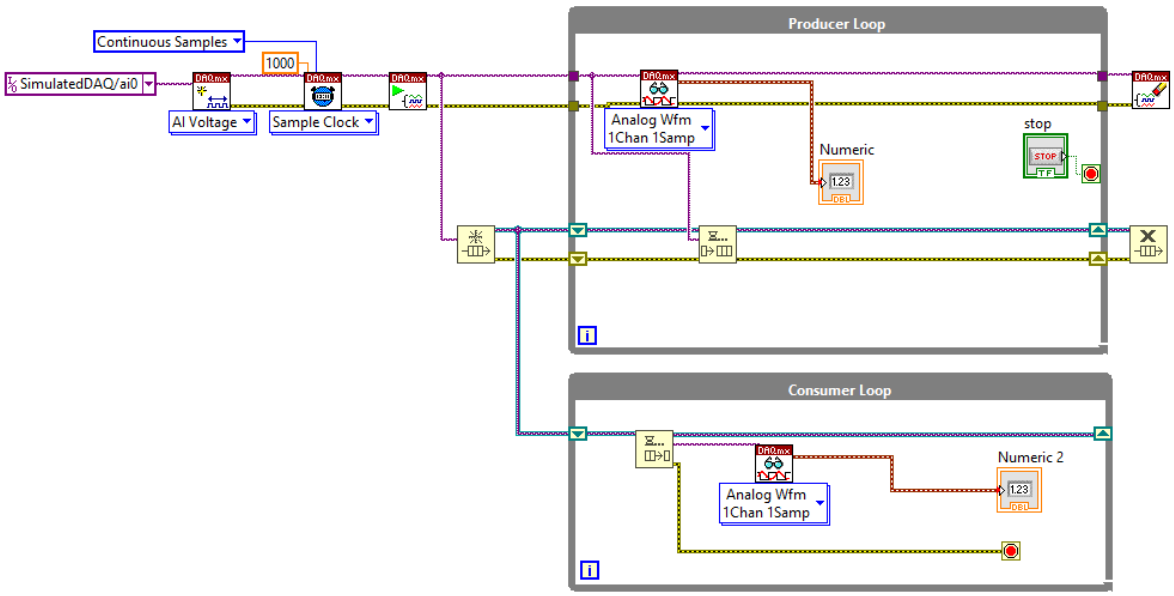


Figure 3- Producer Consumer Test 1

The VI demonstrates a misunderstanding of how the queue functionality should be used. In this first iteration both loops contain a DAQmx read task, although the numerical indicators match when

ran. It's running two separate acquisition tasks. The queue function should be used to transfer the data from the DAQmx read task, not the task itself.

'Producer Consumer Test 2' pictured in Figure 2 addressed these mistakes. Properly implementing a queue system for moving data from the Producer Loop to the Consumer Loop. In this iteration the queue is created outside the loop. Data acquired in the Producer Loop is fed into the Enqueue Element Function. This is then Dequeued in the Consumer Loop.

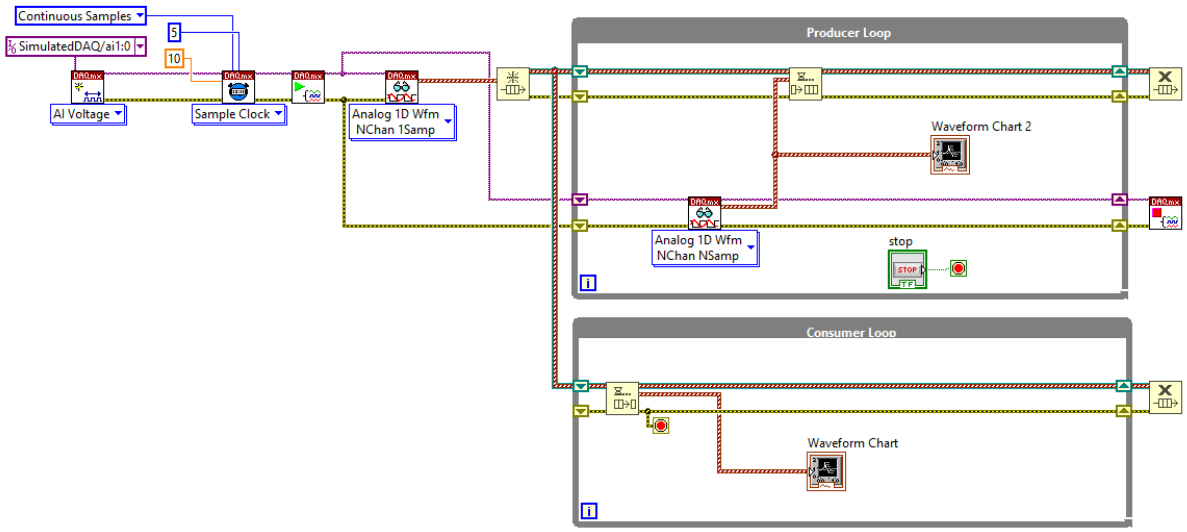


Figure 4- Producer Consumer Test 2

When the queue is created outside the loop, the element data type must be declared. Hence the use of a DAQmx read task outside the Producer Loop. This only executes once when the VI is ran. This VI includes a pair of waveform charts used to verify the synchronous display across the separate loops.

The next iteration of this branch replaced queues with Channel Wires. It also introduced a pair of nested case-structures used to take a rolling average of the signal. This is shown by Figure 5. Channel Wires were added in 2016 (AristosQueue-(NI), 2015) with the purpose of replacing the use of existing asynchronous data transfer tools, most notably queues.

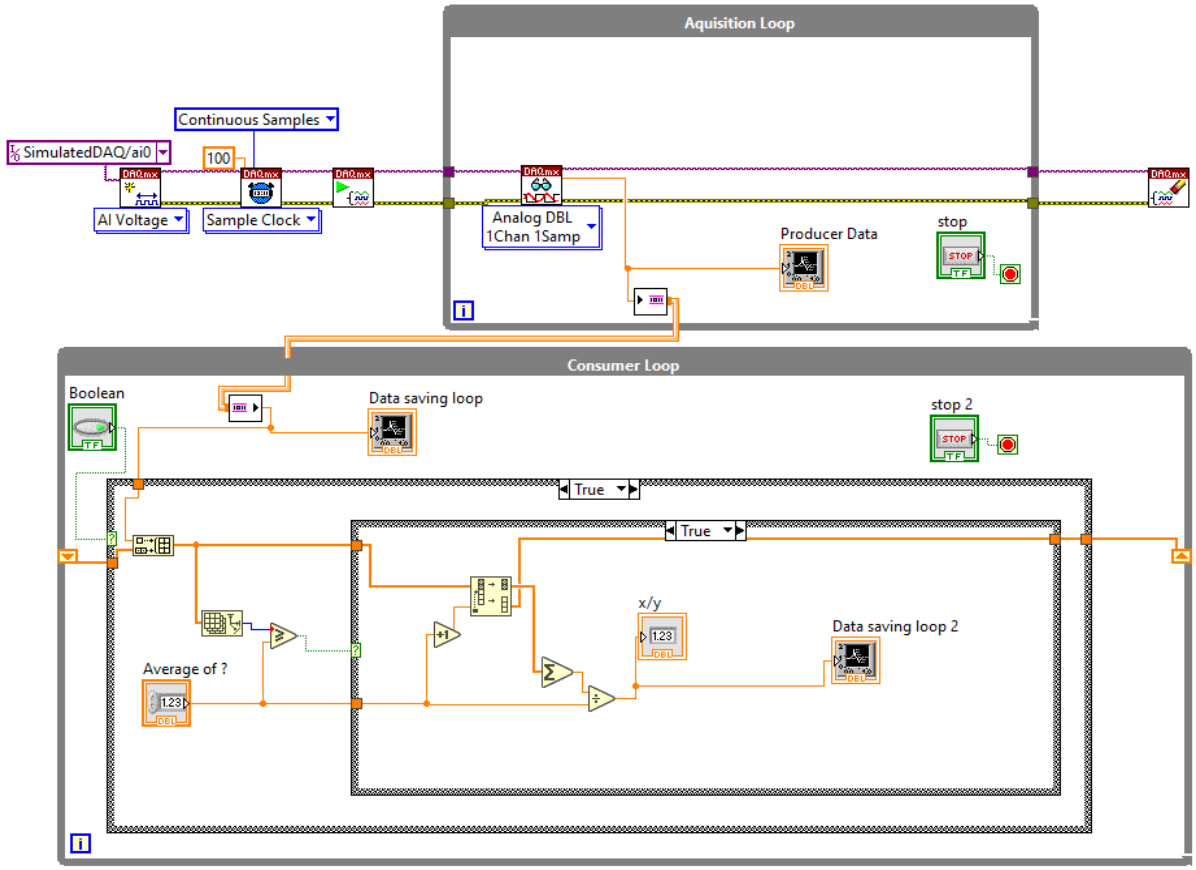


Figure 5 – Producer Consumer Test with Channel Wires

The way the average was being calculated uses an overly complex block diagram, and only works for a 1-dimensional array of data.

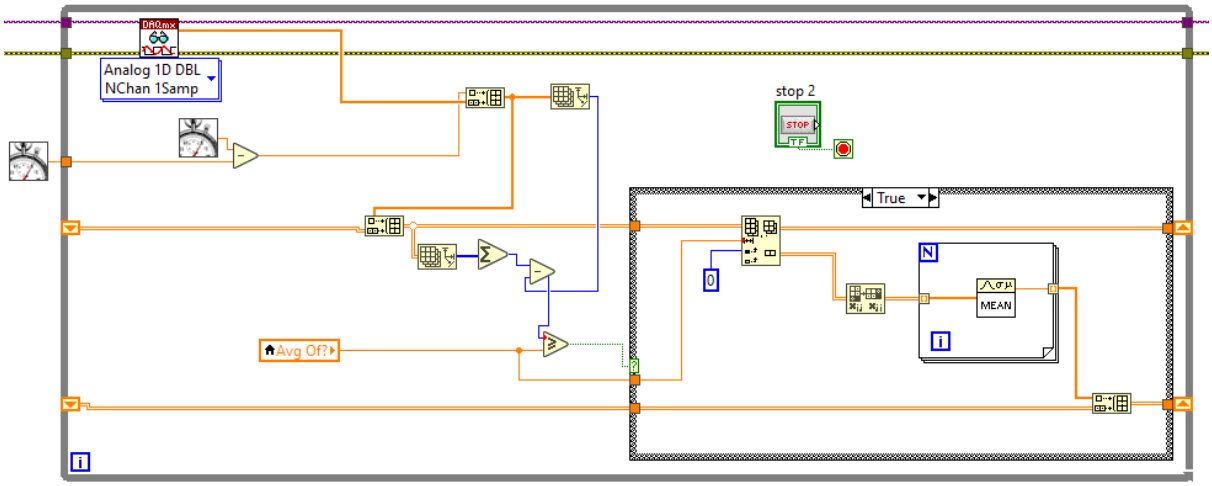


Figure 6 - A better system for taking the average.

Figure 6 demonstrates a better method for calculating the average. It also features the addition of an extra channel of data to the DAQmx task, and a timing mechanism making use of the 'High Resolution Relative Seconds' VI based on Zachary Neale's example (Neale, 2019). This SubVI

provided with LabVIEW returns the current time in seconds. This can be subtracted from a previous value to get elapsed time.

This is a commonly used timing method in LabVIEW and was eventually implemented in the final version of the software, however documentation states this timing method may drift over long periods (NI, 2023). A second absolute timing measurement was later implemented.

The last Producer-Consumer test build (Figure 7) further improved the method of averaging the signals from the producer loop. Making use of the auto-indexing function of For-Loops, by placing the Channel Reader within a For-Loop, it creates an array of values. This 2-D array is transposed and fed into another For-Loop containing LabVIEW's standard 'Mean' SubVI.

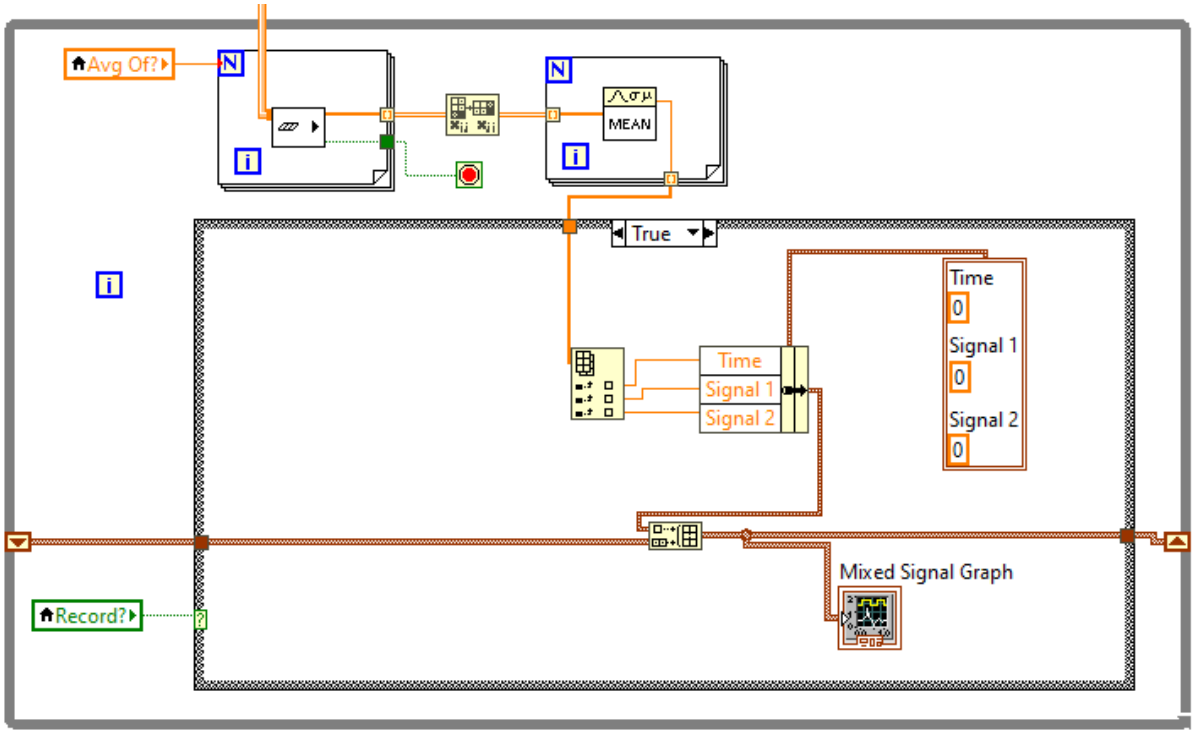


Figure 7 – Final Producer Consumer Test with Signal Averaging and Clusters.

4.2 Main Branch Development

After building simple Producer-Consumer design patterns, developing the early test projects, and creating the mimic, development began on the VI that would become the finished product. The version number began at 2.0 as this code carried on from the Producer/Consumer tests.

4.2.1 V2.0 - Implementing a state machine for the Data Acquisition Loop

The priority was building a robust method for Data Acquisition, John Essick's Hands on Introduction to LabVIEW for Scientists and Engineers (Essick, 2016) section 12.6 details how to build a data acquisition state machine using the DAQmx palette. To begin this was followed as a rough guide.

Using a state-machine comes with many advantages, it gives better control over the behaviour of the VI. Overhead operations such as setup and shut-down actions are only executed when required. The Profiler Performance and Memory window shows that these actions can take significant CPU time. It can also wait on user input or certain conditions to be met before progressing to the next state.

A state-machine helps to keep the block diagram organised and compact, making it easier to understand, as it's a common design pattern used in LabVIEW it can be easily recognised by an experienced LabVIEW user.

Version 2.0 implemented a state machine capable of data acquisition and display. The state machine is controlled by an Enumerated Type Control (Enum), this is a numbered list which can be fed to a case-structure to control its state. The case-structure is controlled by passing the desired Enum via shift register to the next iteration of the loop. An example of this can be seen in the top right of Figure 8.

Beginning with a 'Create Task' state, this created the DAQmx read task, once this has happened it progresses onto 'Change Settings'. During this state it idles waiting for the user to input the refresh rate and confirm by pressing the OK button.

Once the OK button has been pressed it moves onto 'Check Settings' this state is used to set the timing of the sample clock and acquire a datum start time using 'High Resolution Relative Seconds' VI. It also contains a Boolean check which can be used to compare the entered variables against

any desired conditions, if this Boolean check is passed it moves onto the ‘Read Data’ state, if not it returns to the ‘Change Settings’ state.

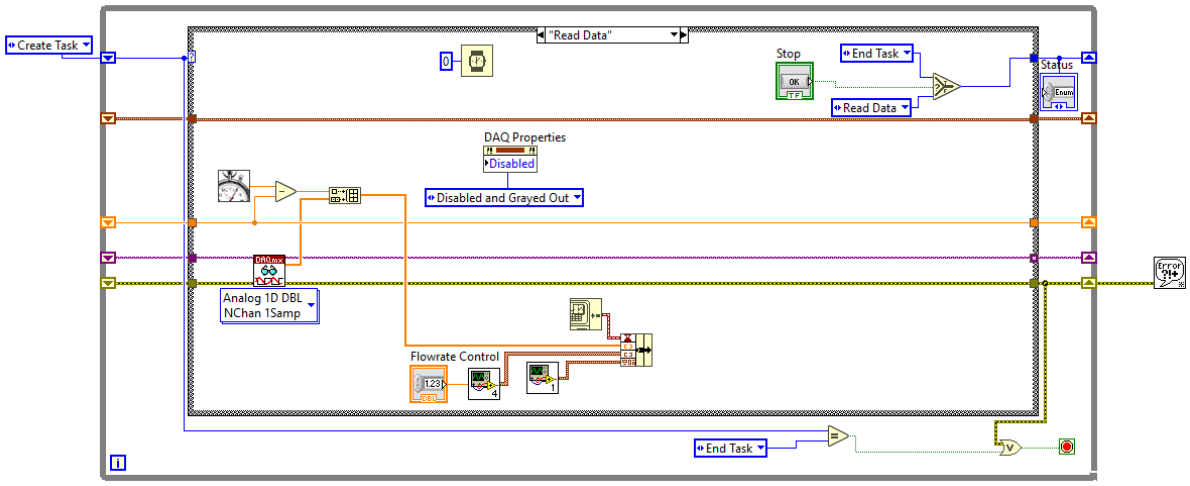


Figure 8 - v2.4 'Read Data' State

Pictured in Figure 8 is an example of the ‘Read Data’ state. A property node is used to disable and grey out the DAQ properties control. This is important as once this state is entered the sample rate is fixed, therefore being able to change it in the UI could be confusing. Once data acquisition is finished the user can press the stop button, moving onto the ‘End Task’ state, where the DAQmx task is cleared, and any reserved resources are released. The Boolean logic at the bottom of the loop then halts the loop.

This overall architecture went largely unchanged and was soon developed into the Producer Loop, although the specific behaviour and contents of each state was tweaked significantly through development.

4.2.2 V2.1/2.2/2.3/2.4 – Simulating Rig & Line Data

To further demonstrate LabVIEW's capabilities more signals would need to be simulated. This could be achieved with additional physical channels added to the NI MAX Simulated Device.

NI MAX simulated devices can only "create a noisy sine wave for all analogue input channels. Configuration of other simulated data is not available" (NI, 2021). Creation of more of these signals doesn't demonstrate any further LabVIEW skills or functionality, and all the signals would be identical sin waves.

Simulate Signal Express VI was used to generate additional data. By using this Express VI it's possible to change all aspects of the signal (shape, frequency, amplitude etc) using controls on the front panel. Signals were added to replicate temperature, pressure, flowrate for each of the five feed lines, and the Rig Temperatures. Later in the project this data is used to demonstrate combustion calculations and alarms. These were then bundled into a cluster to be used with a Channel Writer.

The block diagrams used to simulate this data was packed into SubVIs which are detailed in Section 4.3 of the report.

There is more data displayed on the existing UI than this, however they are only live display. The simple addition of a dozen live indicators serves no developmental purpose for this project.

4.2.3 V2.5/2.62 – Creation of a Consumer Loop for Data Export

The objective of v2.5 was to add a consumer loop responsible for exporting data. A lossless high-speed channel was used to move the data between the loops, and the ‘Write Delimited Spreadsheet’ VI was used to write the data. To do this the data was unbundled and organised into an array of strings. A SubVI was built to do this for the Flow Data as it would need to be repeated elsewhere in the code and took up considerable space on the block diagram.

Like the producer loop, this consumer loop uses the State-Machine design pattern. It begins with an idle ‘Waiting to Record’ state, allowing the user to begin recording from the front panel. From here it creates the spreadsheet and writes in the headers.

Versions 2.6 (Figure 9) saw improvements to the way the data was converted into an array of strings, more control over the behaviour of front panel objects, the spreadsheet was changed to be .csv, and the addition of a general error handler. Importantly the state machine only progresses past the ‘Recording’ once the Channel Reader has finished receiving data and the buffer is empty. This prevents data loss when recording is finished.

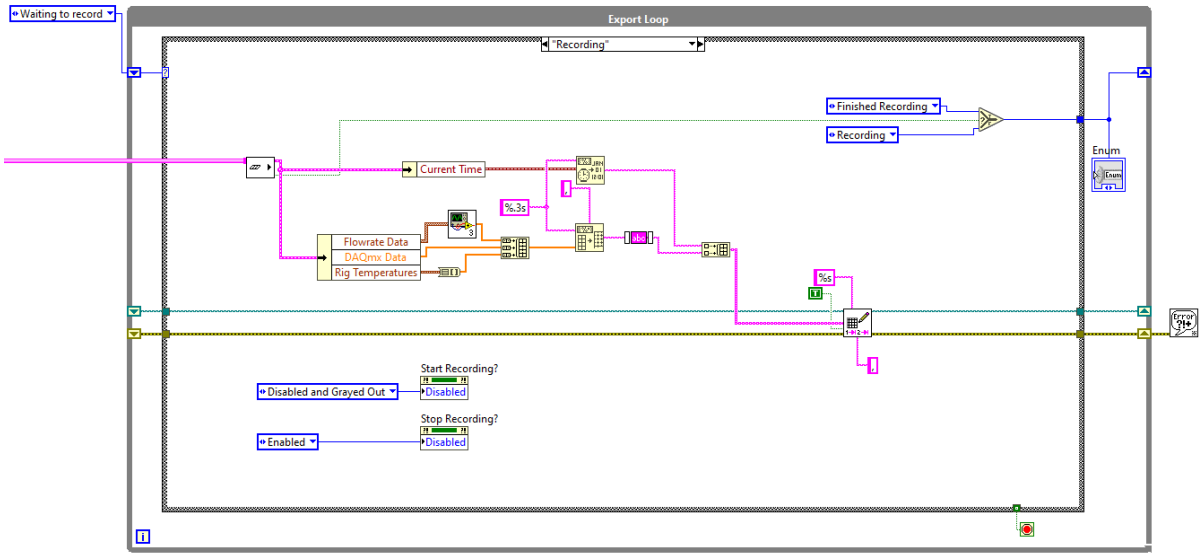


Figure 9 - v2.62 Export Loop, Read State

4.2.4 V3.0/3.1/3.2 – Live Display Consumer Loop, Improvements to Export Loop

Versions 3.0 – 3.2 saw the addition of a basic consumer loop for live display shown in Figure 10.

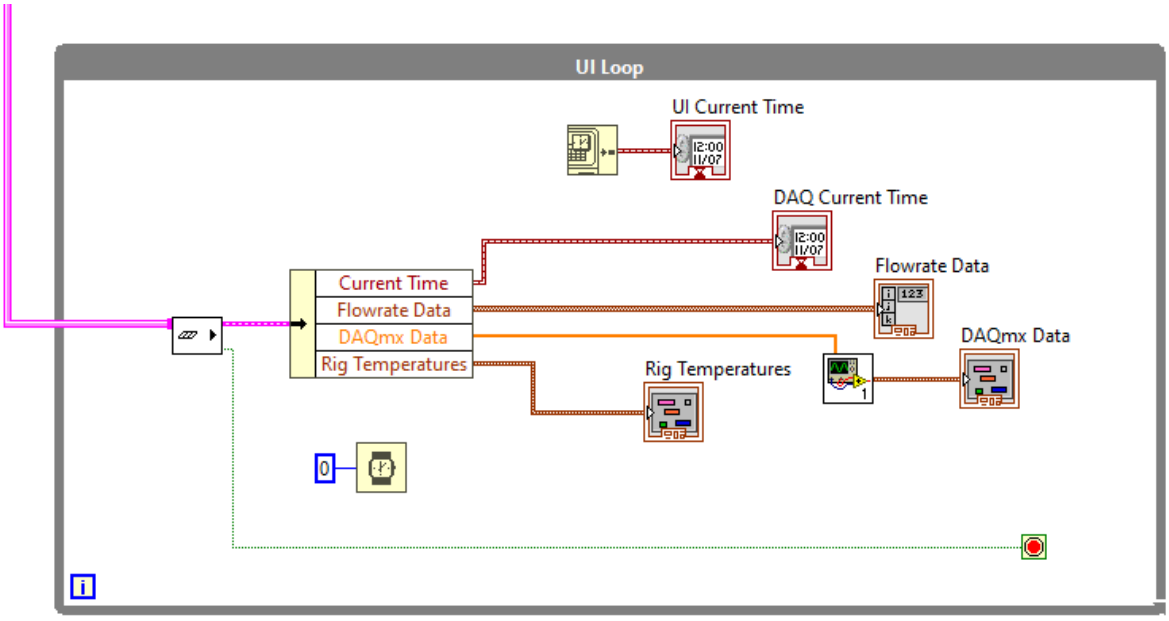


Figure 10 - v3.2 UI Consumer Loop

This loop contains displays for each of the clusters of data being captured, as well as the timestamp recorded in the Producer Loop, and a timestamp generated and displayed in the UI loop. The loop is halted using the 'done?' terminal on the channel reader which returns true once the channel has cleared its buffer and is receiving no more data. The addition of separate timestamp displays for each loop allows the user to identify if the live display has fallen out of sync with the acquisition.

As well as a basic live display, these iterations saw significant improvements to the export loop, fixing several problems:

If the data acquisition ran for too long whilst the export loop wasn't recording, the buffer on the channel would fill up, causing the VI to crash. This was because the producer loop would be writing data to the channel, which would then be trapped with nowhere to go as no Channel Reader had been created yet. Once the consumer loop moved to writing the data to file, it would first write all the saved data, emptying the buffer.

A potential fix for this, would be adding an additional state to the producer loop, where the data is acquired but is not sent to the channel writer. A simpler solution was used, by adding a channel reader to the 'Waiting to Record' state of the export loop which didn't output to anything, the channel would be read, and the buffer kept empty, without needing additional states or block diagram logic.

The next issue was that Microsoft Excel was truncating the exported timestamps when opening the CSV. When opening the file in Notepad the digits existed in plain text format, but when opened in Excel the timestamps lacked the seconds measurement. Adding code to format the date/time stamp as two separate strings corrected this error.

If the exporting filename/location was already open in windows when the export loop tried to create the .csv, no error would occur initially. Only flagging an error when the export loop exits, losing all recorded data.

Debugging showed that the error was generated within the ‘Creating Spreadsheet’ state but wasn’t addressed by the VI until the export loop exited. To remedy this an error handler was placed in the ‘Creating Spreadsheet’ state. If an error does occur the user is notified, and Boolean logic returns the loop returns to the ‘Waiting’ state. The loop only continues onto the ‘Recording’ state if there have been no errors with the creation of the spreadsheet. Adding an error handler to the ‘Recording’ state causes the UI to become unusable as it creates a new error dialogue with each loop iteration.

Another solution would be to concatenate the timestamp onto the end of the filename when the spreadsheet file is created resulting in a novel filename each time as described in LabVIEW for Data Acquisition Section 6.3.2 (Mihura, 2001). Attempting to implement this prevented the file path dialogue from automatically appearing if a file path hadn’t been entered already. This was a key safeguard preventing the user from recording without entering a file path, so the idea was scrapped.

Lastly the creation of the array of strings used to generate the spreadsheet headers was moved into its own SubVI.

4.2.5 V4.0/4.1 – Optimising the Export Loop

At sample rates greater than ~ 50Hz the data export loop was unable to keep up with the acquisition loop, causing the software to error and the export loop to terminate. This is a significant issue, if this occurred during a test campaign all data beyond that point would be lost. The export loop needed to execute faster.

Possible solutions were: pre-allocate the file size on the disk to increase write speed, the use of TDMS instead of the ‘Write to Spreadsheet’ VI, using multiple export loops to alternate in writing the data, batch the data and write in batches, hold all the data in memory and write when the loop terminates. This last option was not seriously considered.

The first step was to streamline the block diagram within the loop. The execution priority of this SubVI was increased slightly.

Flowrate Data, DAQmx Data, and Rig Temperatures were combined before being turned into an array of strings in one operation, rather than being done in three separate operations. Concatenating strings like this is advised by National Instruments (NI, 2023).

Batching the data is simply done by putting the channel reader inside a for-loop, this indexes the 1-D array of strings into a 2-D array. Which can be fed directly to ‘Write Delimited Spreadsheet’ VI. This is shown in Figure 11. The iterations on the for loop is set to match the sample rate of DAQmx, at 100Hz sample rate. For example the loop will collect an array with 100 rows of data, which will be written once per second. This sped up execution as file I/O calls each incur a significant overhead (NI, 2023), by batching the data the rate at which an I/O call occurs is significantly lower.

Batches could be made larger, potentially making a file call every few seconds. The bigger the buffer the more data must be held in memory. In Feel-Soon’s DAQ system the storage interval was set at 30 seconds, although given the age of the article this system would be running on older hardware than this project (Feel-soon, 2008). Changing the size of the buffer is simple should the user want to and could even be controlled from the front panel by a simple numerical control.

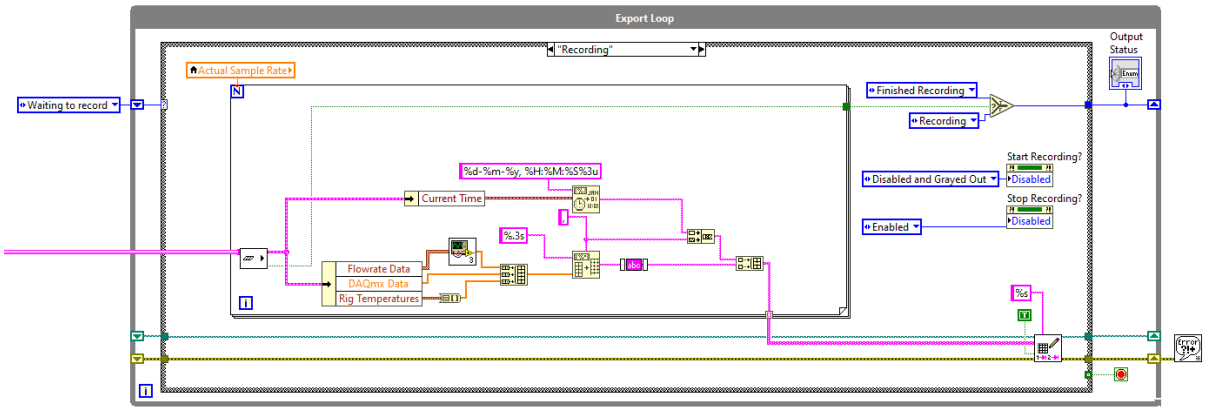


Figure 11 - v4.1 Data Export Loop, Recording State

With these changes the VI can now run at 250 Hz for an extended period, providing a significant safety factor given the existing system runs at 1 Hz. Sampling rates this high and beyond are void given the time constant on a typical thermal couple is slower than this (Omega, n.d.).

This version of the code was presented to Matt Allmark for a brief code review. The overall structure of the code was discussed. The use of Producer-Consumer and State-Machine design patterns were well received. Some key feedback was the consideration of loop priorities, timing mechanisms, and the unnecessary use of a lossless highspeed channel for the UI Loop.

The inclusion of a wait-milliseconds timer in the Export Loop was discussed as potentially useful, however later testing showed this to be unnecessary. The use of a for-loop batching the data forced the loop to iterate at a slower pace, as the Export Loop would wait for the for-loop to finish batching 1 second’s worth of data before writing to file and beginning its next iteration. The batching based on sample rate acts as a forced timing mechanism.

Lastly the lack of code comments was noted, up to this point the code did not contain any comments. As this is a solo design project it had not hampered development, however commenting is best practice. Comments were included in new code from version 5.0 onwards and retroactively added to old code in later versions.

4.2.6 V5.0/5.1/5.2/5.3 - Implementing EQ Ratio Calculations

The existing DAQ system at GTRC can calculate the Air-Fuel-Ratio and the Equivalence Ratio from the Flow Data. The EQ ratio calculation on the existing DAQ does not function, and a Thermal Power calculation is not present. These variables are often targeted as a given test point. Version 5 aimed to implement a simple system to handle these calculations and display them without the user needed to access the code.

To do this a look-up table was created containing some common fuels used at the GTRC, in this table was the AFR and Lower Heating Value for that substance. Controls were added to a tab on the front panel where the user can select the substance being fed in each line or enter custom properties if the substance isn't an available pre-set.

The look-up table is turned into an array of strings, a for-loop is used to index each of the lines. This loop reads in the substance for each line, searches the array containing the look-up table to get an index number for that substance, which is then used with a case-structure to obtain the line's variables. An early version of the block diagram code for this is shown below in Figure 12.

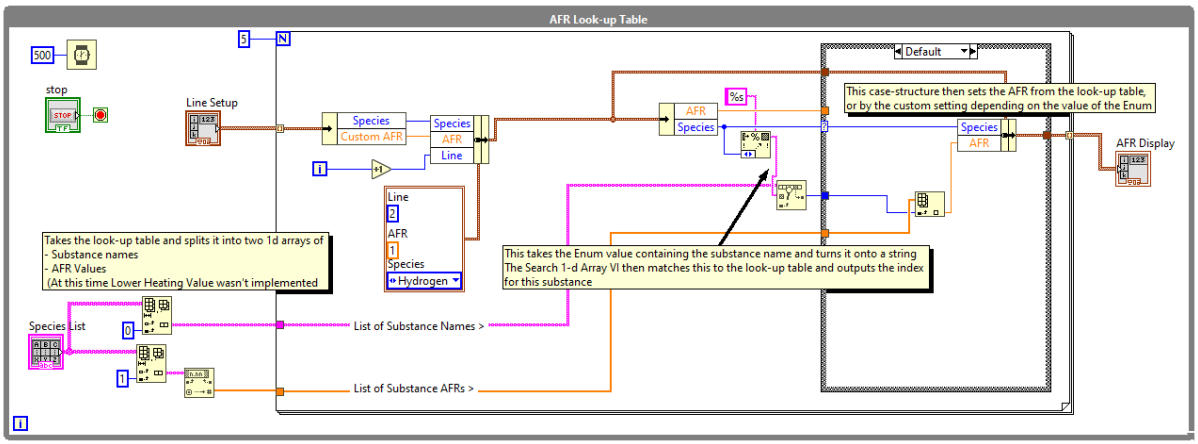


Figure 12 – v5.0 Block Diagram for AFR Calcs

Version 5.1 saw the addition of live graphing of temperature and pressure.

The addition of this look-up table for AFR created a bug where the Flow displays were disabled and greyed when the VI was run and the graphs didn't display properly, this would resolved once a substance was set and the AFR variable was initialised.

This was fixed in later versions with the addition of an 'Empty' Enum optiaon for the lines, and a Local Variable to set and initialise these parameters. Using local variables is considered bad practice in LabVIEW as they are slower than signal wires. It is acceptable in this case as the loop is not required to iterate at a high rate.

In Versions 5.2/5.3a DAQmx timing node was added to the acquisition loop, measuring the actual sample rate. Whilst this isn't required when simulating signals, during real acquisition the actual sample rate can vary from what's been set. Having a timing node to measure the real sample rate is considered best practice.

The high speed lossless channel to the live display loop was changed to a single element lossy channel. As a lossy channel it can still write at the speed of the acquisition loop, but can be read at any speed. If the buffer is full it overwrites the oldest element in the buffer. High speed live display is unnecessary and slows down any loop it is in. This change allows the control of the UI's refresh rate using the code shown in Figure 11.

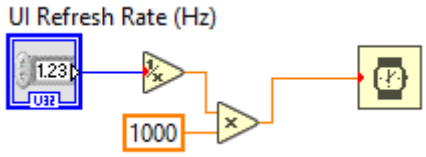


Figure 13 - UI Refresh Rate Block Diagram Code

Changes in refresh rate can cause display issues with the live charts as they have a fixed chart history length. This chart history length cannot be changed programmatically. (NI, 2020)

Another significant change in this version was combing the simulation SubVIs for Rig and Flow into one SubVI. The execution priority of this SubVI was set to 'Time Critical Data Acquisition', which is one before 'subroutine', this was because SubVIs containing simulated signals cannot have subroutine execution priority.

The acquisition loop for version 5.3 can be seen below in Figure 14

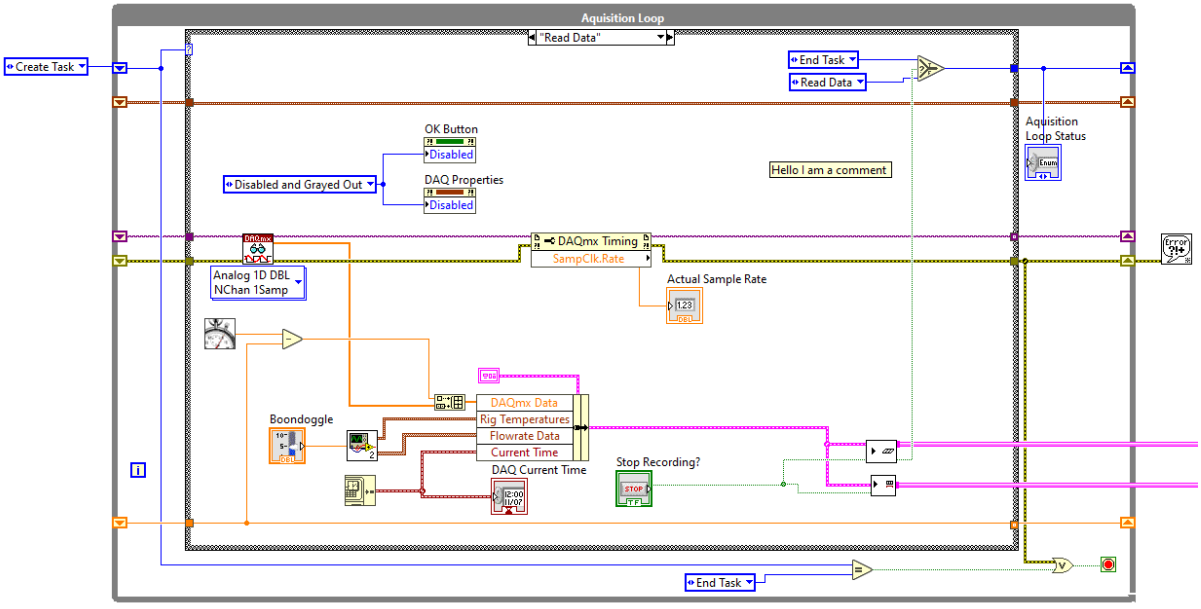


Figure 14 - V5.3 Acquisition Loop, Read State

4.2.7 Version 5.4/5.41 – Rig Temperature Alarms & Improving EQ Ratio Calculation

Safety is a concern when using high temperatures and pressures. The existing DAQ system has alarms built into the display. Flashing red when a temperature goes above a threshold value. This can be implemented in LabVIEW using property nodes. Versions 5.4 sought to demonstrate this capability using the Rig temperatures.

A pair of tabs were created on the front panel for the Rig temperature indicators and their threshold values. Allowing the users to change the alarm thresholds from the front panel.

Property nodes in LabVIEW can only apply to one front panel object at a time. The initial implementation of these alarm thresholds was to take the Rig Temperatures, and their thresholds as arrays. These arrays were fed into a for-loop to be compared which creates an array of Booleans. This Boolean array is then used to control each front panel indicator separately.

There was concern that if the thresholds tab was open on top of the indicators, the user would be unaware a threshold had been exceeded. To combat this the VI programmatically changed the tab back to the Rig Temperature Indicators tab.

This solution caused issues. When an alarm was activated, changing the threshold was very difficult as the VI would keep switching to the indicators tab. This solution was removed in favour of causing the threshold values in the thresholds tab to blink when they had been exceeded. Informing the user that the threshold had been exceeded even when looking at the wrong tab. Hence the existence of two columns of property nodes in the block diagram pictured below in Figure 15.

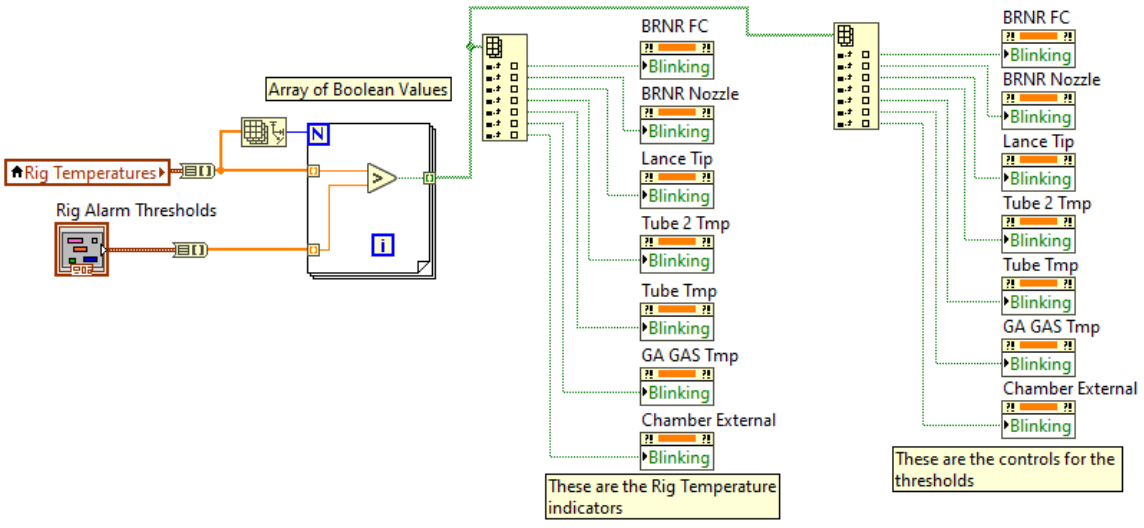


Figure 15 - Early Block Diagram Code for Alarm Thresholds

The LabVIEW discussion forums contained a method for extracting the names of indicators from a cluster (Dayley, 2008).

This was applied to the Rig Temperatures Cluster and Rig Alarm Thresholds to obtain arrays of their individual indicator's references. Using a for-loop these are compared as before but the property nodes are assigned by reference for each iteration of the loop. Allowing for individual control of each indicator, without individual property nodes. This code is shown in Figure 16.

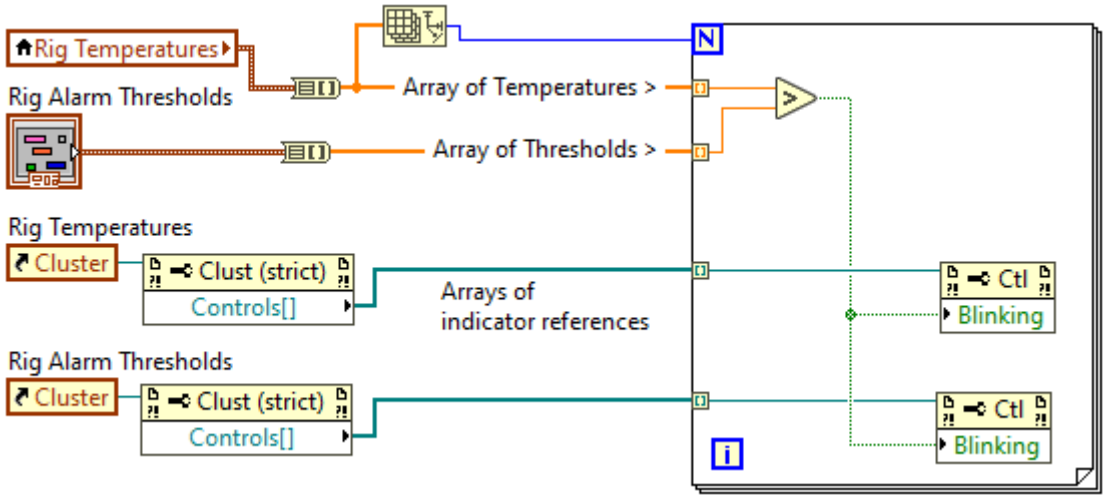


Figure 16 - Improved Block Diagram Code for Alarm Threshold

As seen, this is a more elegant solution, simplifying the block diagram. It would also scale automatically if more indicators were added to these clusters. One downside of this solution is the use of local variables as these take longer to transfer data than wires (NI, 2023), however this code is placed in the UI Loop, and is not required to iterate at high speed.

An alternative would be to install and use NI's Datalogging Supervisory & Control Module (DSC). This module provides advanced functionality for shared variables. This module can be used to manage alarms and events for a very large number of variables, it also provides tools for historical or real-time graphing of trends (NI, 2012) (NI, 2023).

4.2.8 V5.5/5.51 – Improving EQ Ratio Calculations and Implementing Thermal Power

5.5 saw the final implementation of EQ Ratio Calculations and Thermal Power. These two processes were bundled into a single for-loop and moved from their own while-loop to the Live Display while-loop. This reduced the total number of top-level loops to 3. The block diagram code used to do this is shown in Figure 17 on the next page.

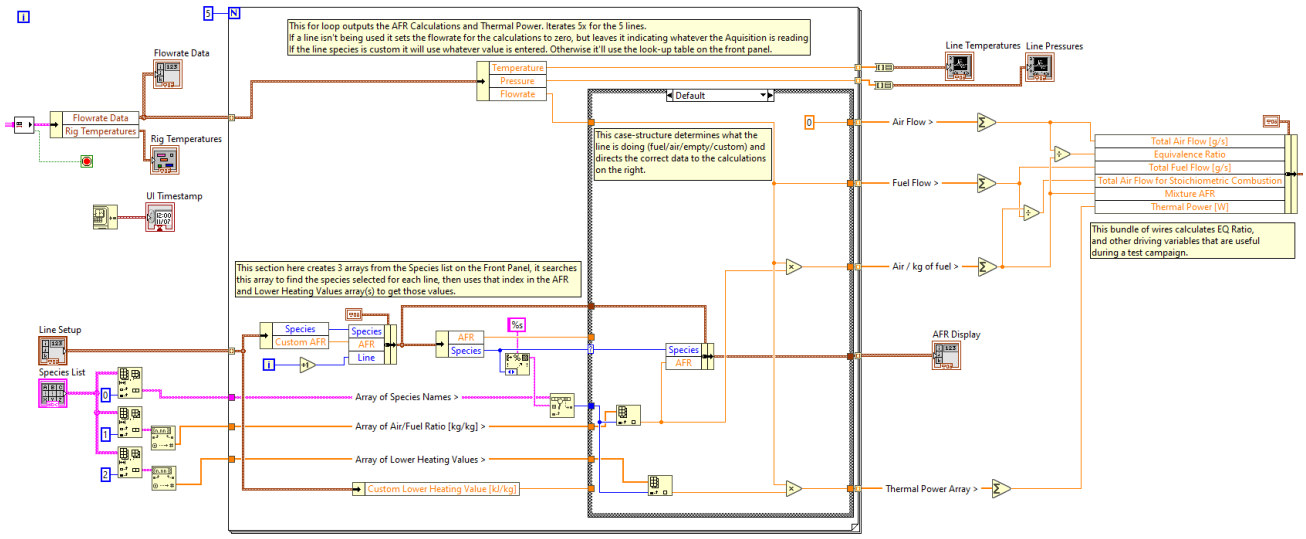


Figure 17 - Version 5.51 Block Diagram for Combustion Calculations

4.2.9 V6.0 – Improving simulation of Flow Data

This version was started to implement output control through the creation of additional higher Rig Temperature thresholds. The DAQ Assistant express vi was used, however in the absence of hardware integration to demonstrate this, the functionality was dropped due to time constraints.

During a Test Campaign the flowrate of the individual lines is controlled separately, and Thermal Power and Equivalence Ratio are used as driving variables or benchmarks. To help with demonstration and testing of the VI, individual controls for flowrate, temperature, and pressure, were added to the front panel. These are shown below in Figure 18. The SubVI used to simulate the data was changed to be able to take these values input as a cluster.

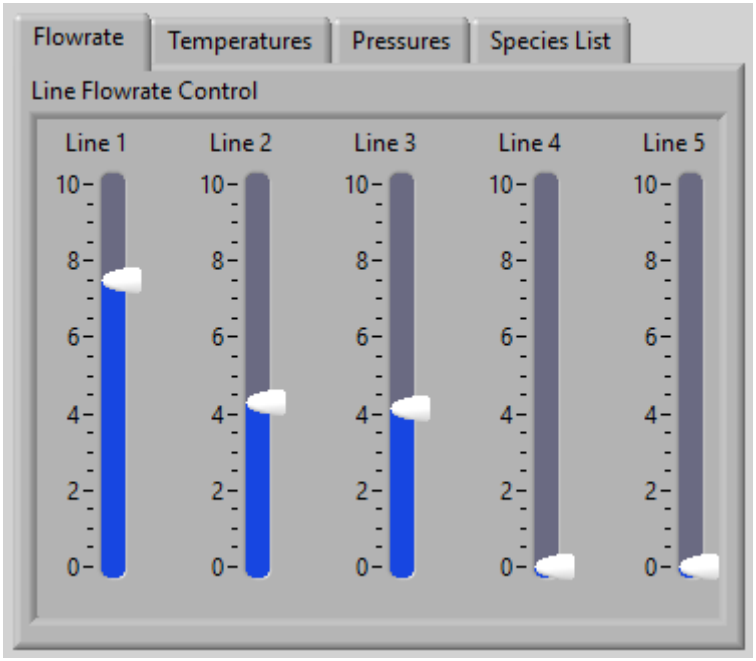


Figure 18 - Front Panel Controls for Flowrate, Temperature, Pressure

Whilst the exact values simulated for the flow data are not relevant, it's useful to be able to drive the VI and see how it responds to changing values. The last tab 'Species List' contains the look-up table for AFR and Thermal Power.

In this version the overall front panel was organised, and comments were added to the block diagram. The front panel for version 6.0/6.1 is shown in Figure 19.

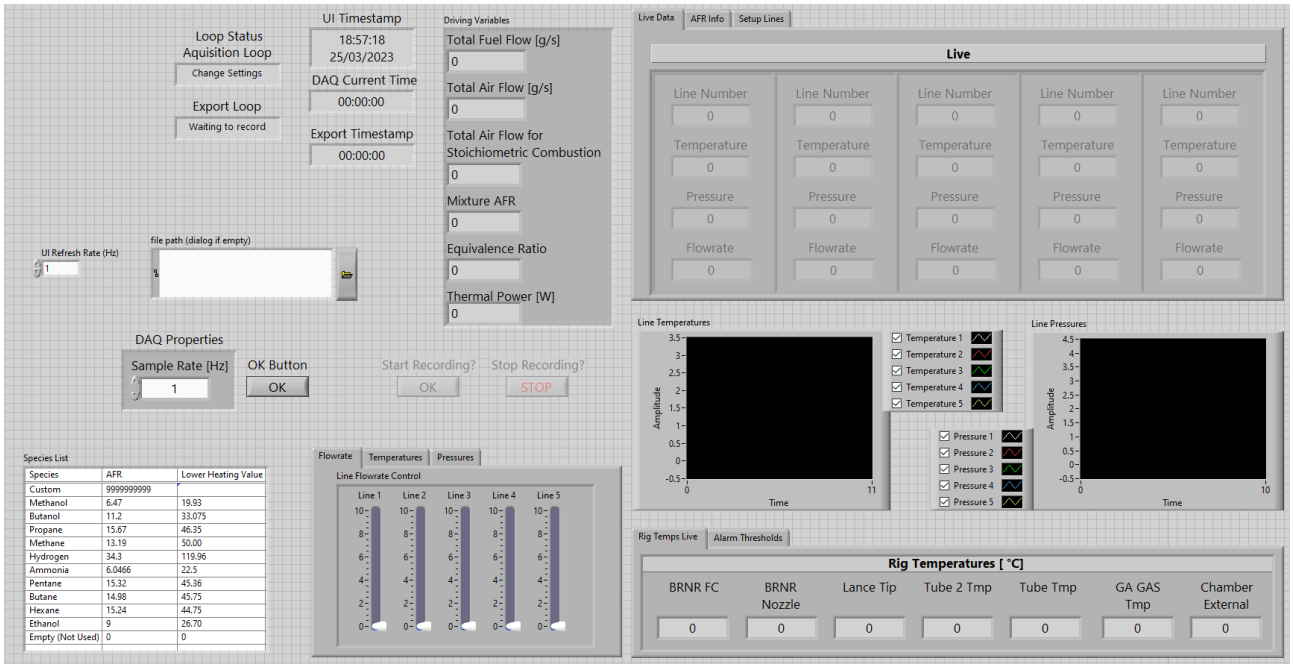


Figure 19 - Front Panel for Version 6.0/6.1

4.2.10 V6.2a/6.2b – Using a formula node.

LabVIEW's formula node is a structure on the block diagram which allows the user to enter traditional written code, commonly used for mathematical formulas. The syntax is based on C. (National Instruments, 2023).

There was a mistake in the code used to calculate EQ Ratio, in version 6.2a this was corrected using wires. This solution is untidy and unintuitive. As an alternative v6.2b uses a formula node to handle these calculations. Formula nodes use text-based programming. This section of the block diagram for versions 6.2a and 6.2b are shown in Figure 20 and Figure 21 respectively.

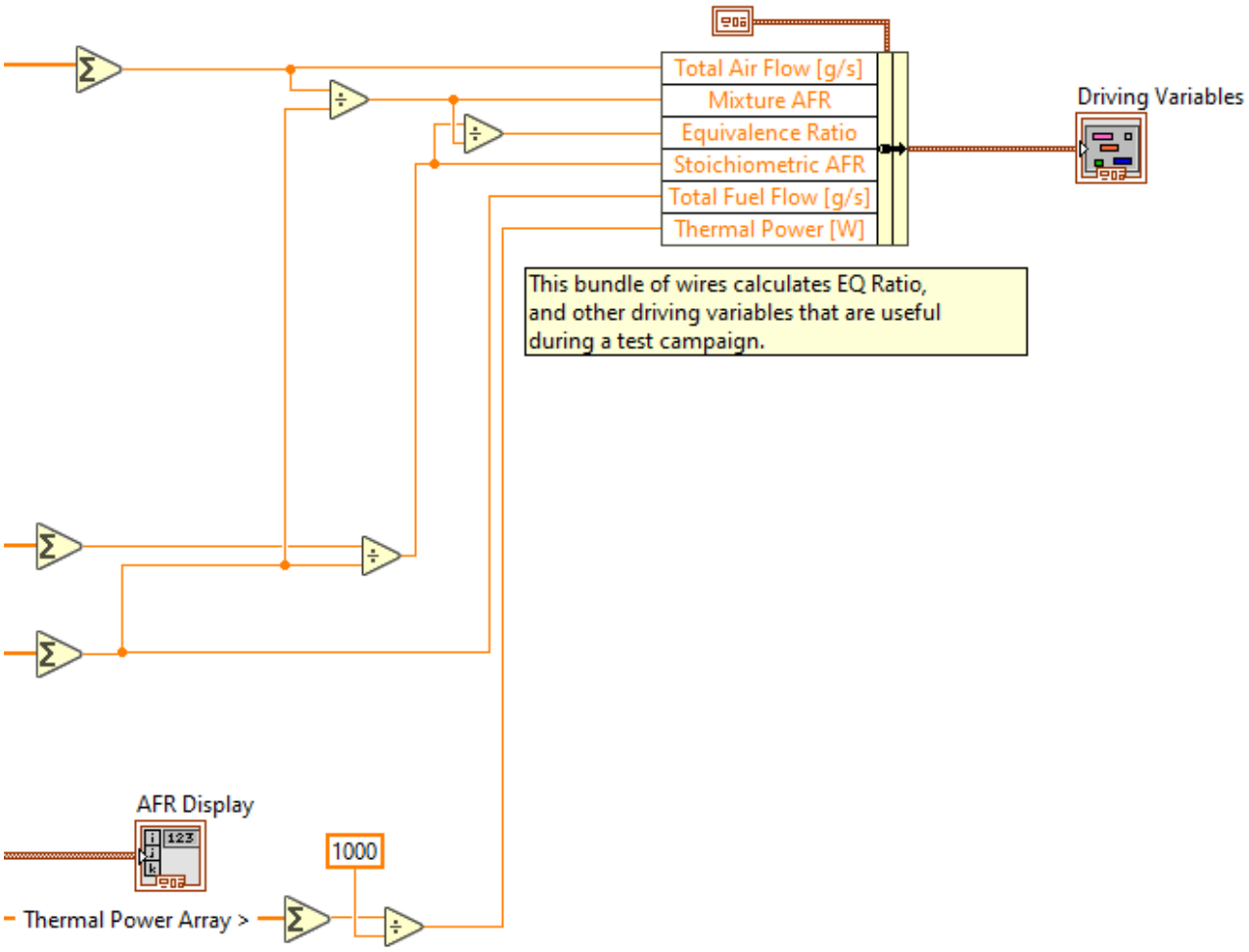


Figure 20 - Version 6.2a Combustion Calculations Block Diagram

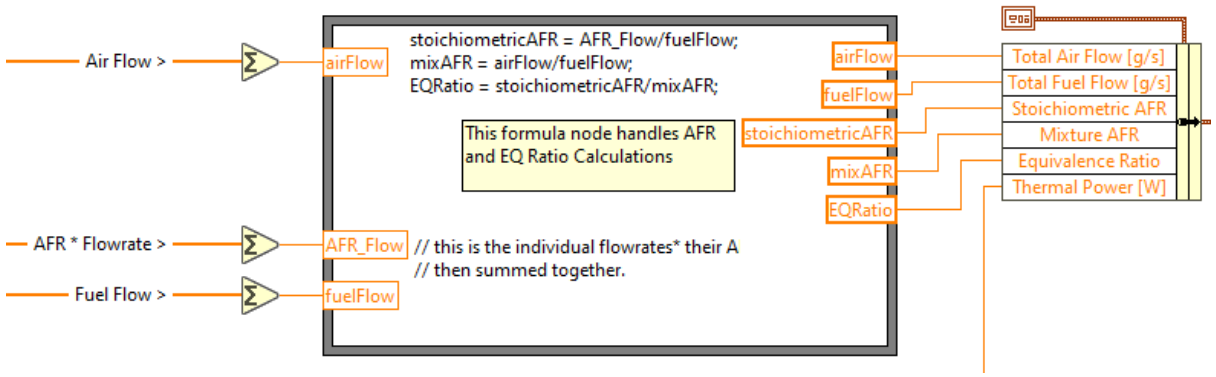


Figure 21 - Version 6.2b Combustion Calculations Block Diagram (using formula nodes)

Formula nodes can contain if statements and loops, in theory the whole section of code used to assign AFR values and calculate EQ Ratio could be put into formula node, however wires are the fastest most efficient data transfer in LabVIEW. (NI, 2023)

4.2.11 Version 6.3a – SubVI for Combustion Calculations & Finishing Touches

Version 6.2a was carried forward. To simplify the block diagram the combustion calculations were combined into a SubVI. Figure 22 shows the improvement in legibility. The contents of this SubVI are covered in section 4.3 of the report. To reduce overhead when calling this VI, it's set as an inline function. Its execution priority is unchanged as the UI display loop is the least time critical loop.

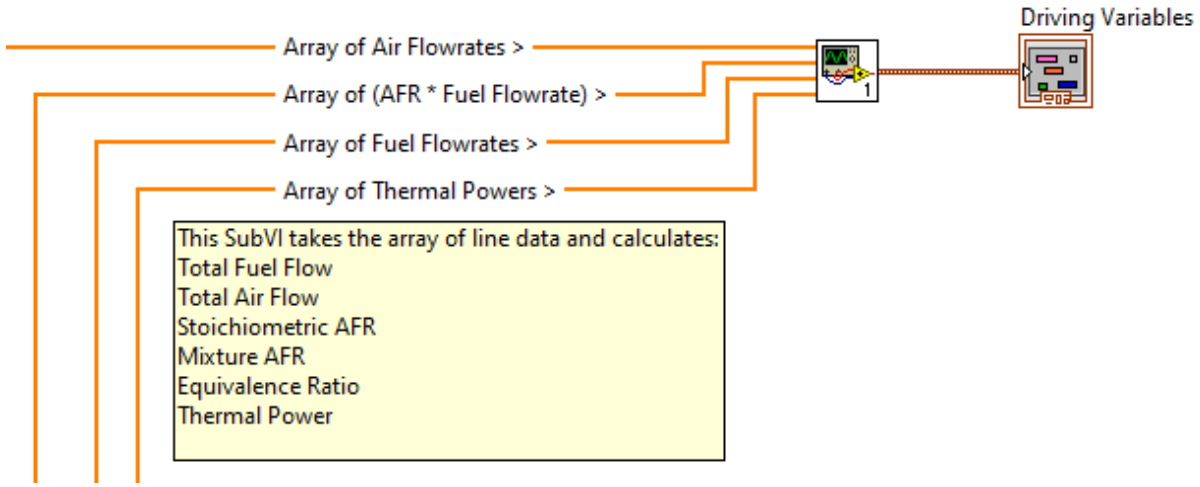


Figure 22 - Version 6.3a Block Diagram (Implemented SubVI)

In this final version minor improvements were made to the block diagram; code comments were added, and cosmetic improvements were made to the front panel. The full breakdown of the final code can be found in the Results section.

4.3 SubVIs

Modularity is considered good practice in programming, LabVIEW is no different. SubVIs are the LabVIEW equivalent of functions from traditional text-based programming languages. They allow the creation of modular self-contained code, which can be called by a larger VI to perform a task. Use of SubVIs is encouraged as it helps keep code manageable. (NI, 2022)

Four SubVIs were created for this project, section 1.10 of LabVIEW for Data Acquisition (Mihura, 2001) was followed initially. Custom logos were made for each to adhere to LabVIEW's style guide (NI, 2023).

4.3.1 Flow & Rig Data Simulation

The first SubVI created was used to simulate some of the data used by the existing DAQ system, this is called within the data acquisition loop. The final iteration simulated Flow and Rig temperatures, using the simulate signal express VI. The Flow Data is controlled by front panel objects. This SubVI is shown by Figure 23.

The auto indexing function of the for-loop is used to create an array of five clusters, one for each feed line at GTRC. Each cluster contains temperature, pressure, and flowrate. As this SubVI is called within the acquisition loop execution priority of this SubVI is set to 'time-critical' which is the second highest behind subroutine. The subroutine priority is incompatible with the simulate signal express VI. The preferred execution style is set to 'data acquisition'.

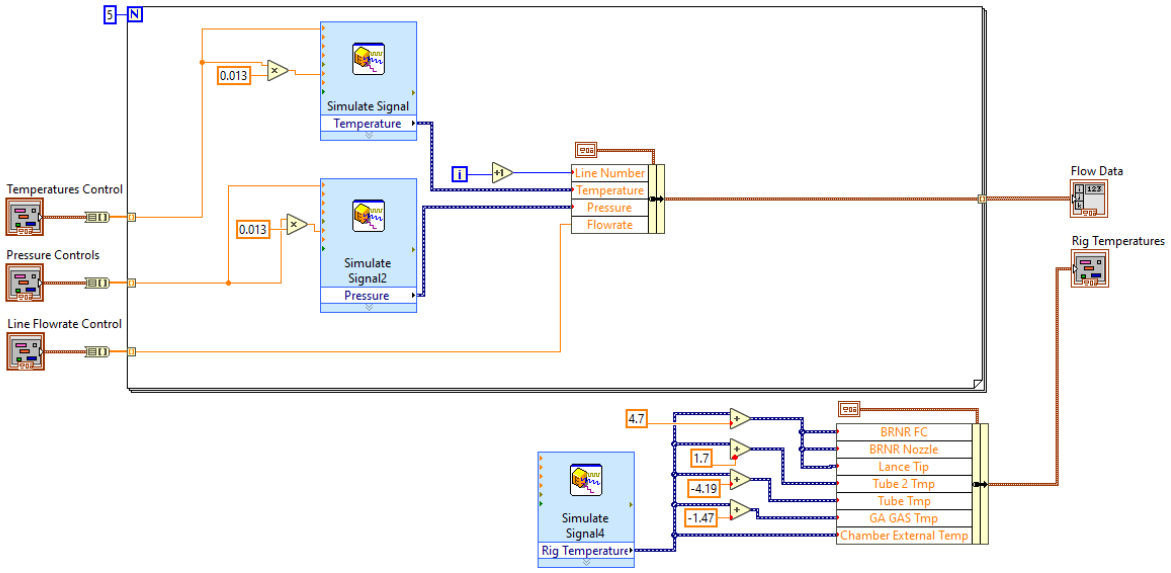


Figure 23 - Combined Simulation SubVI Final Version

4.3.2 Flow Data to 1D Array

Figure 24 is a SubVI used to unbundle the array of clusters containing the flow data into a 1D array. This is required by the export loop as the 'Write to Delimited Spreadsheet VI' cannot receive an array of clusters.

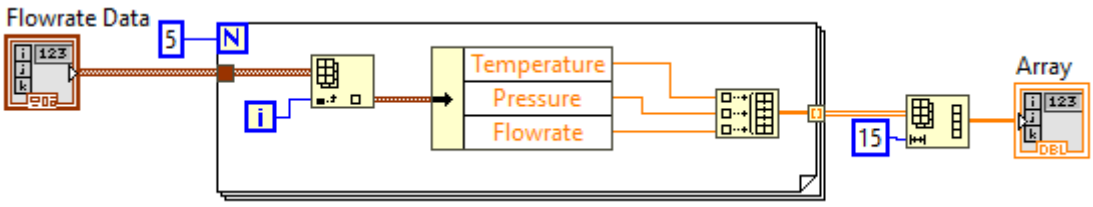


Figure 24 - Flow Data to 1D Array SubVI

This is set to be a high priority inline SubVI. Subroutine priority could be used here. But the code is called within the data export loop, it is sensible for it to have a lower execution priority than the Simulation SubVI called within the acquisition loop. Setting this as an inline function increases execution speed by compiling the SubVI code into the calling VI code, reducing overhead. (NI, 2023)

4.3.3 Create Spreadsheet Headers

The code used to generate the spreadsheet headers was moved into its own SubVI. This was done only to improve readability of the 'Create Spreadsheet' state of the Export Loop. As this VI only executes once, the priority is not important. The SubVI works by creating an array of strings

and then concatenating them into one long string. Which is then fed to the ‘Write Delimited Spreadsheet VI’. Due to it’s simplicity this VI is in Appendix B - Additional LabVIEW Code.

An improvement to this VI would be extracting the cluster names of the signals using a property node. This would be scalable, as currently if more signals are acquired additional spreadsheet headers would need to be added manually to the code.

4.3.4 Combustion Calculations SubVI for 6.3a

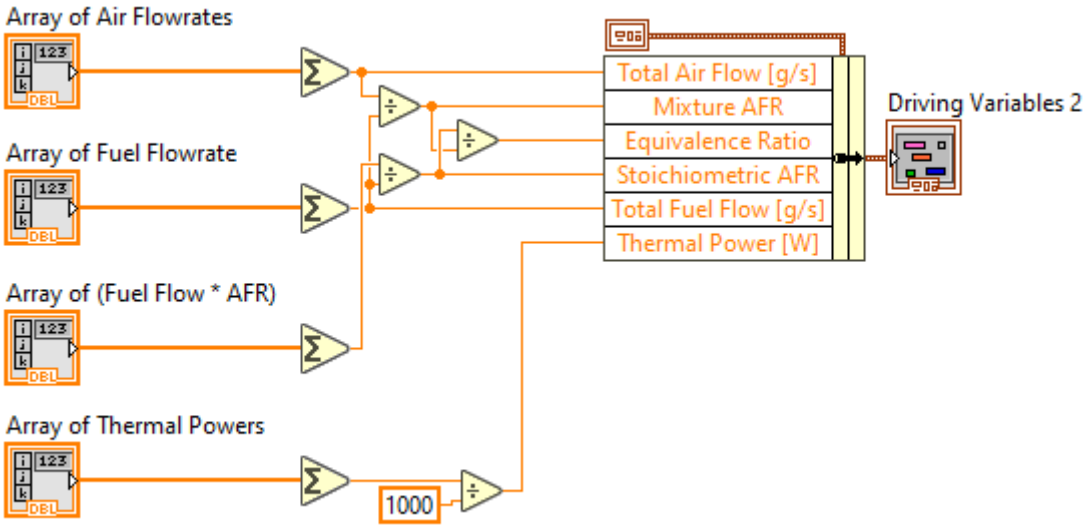


Figure 25 - Combustion Calculations SubVI

Figure 25 shows the contents of the SubVI used for the combustion calculations. The arrays on the left are the inputs, and the cluster on the right is the outputs.

5 Results

5.1 The Front Panel and Features

A LabVIEW project file (.lvproj) was created, within this a .exe was made for the VI to run as a standalone program. This project file and all LabVIEW code can be found here: https://github.com/atkinson-t/GTRC-DAQ-for-Dissertation.

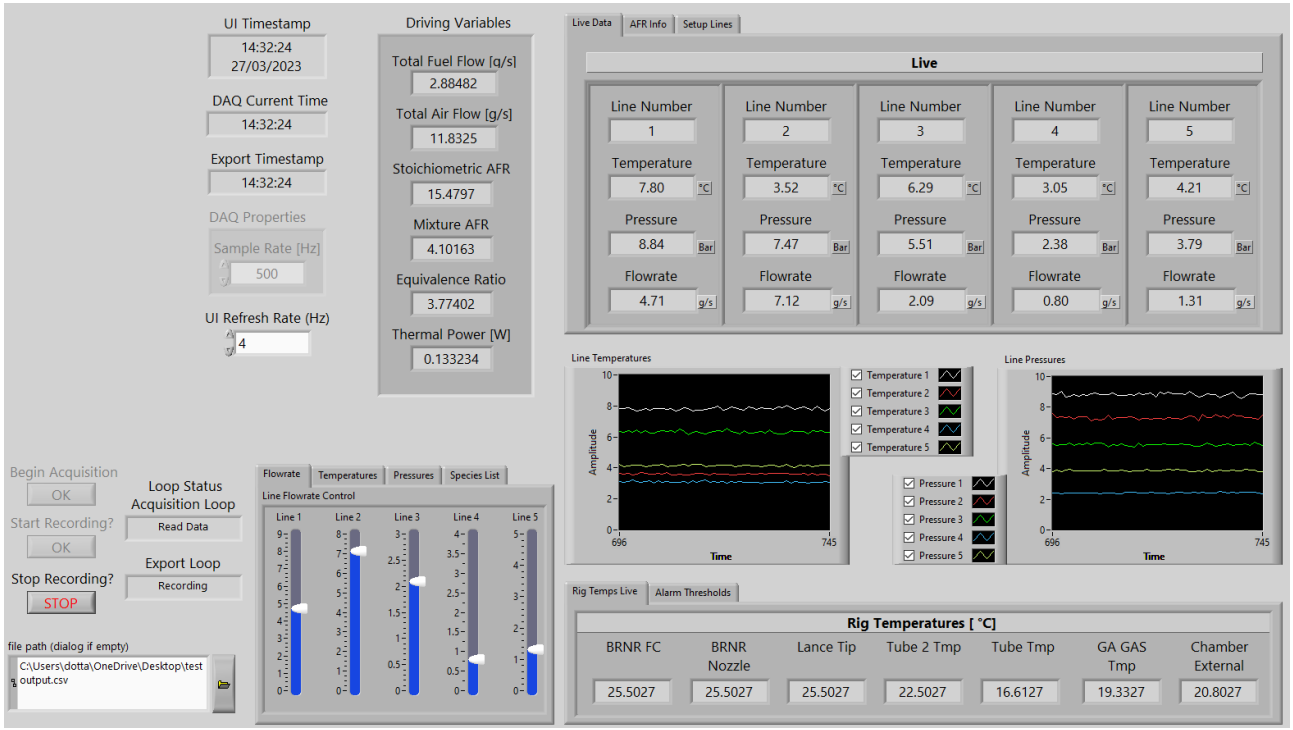


Figure 26 - Final Build Front Panel

Figure 26 shows the front panel design for the final build. The Driving Variables, Rig Temperatures, and Line Data are grouped into clusters. Tabs have been used contain controls for Alarm Thresholds, and AFR Information. Controls have been added for Flowrate, Temperature, and Pressure. Timestamp displays for the three parallel loops (DAQ, Export, UI) have been included as well as indicators for the status of the state machines. The default file path dialogue has been used. Lastly real-time graphs have been added.

The VI can run reliably at a sample rate of 250 Hz on an Intel Core i5-1035G1 running at a base processor frequency of 1.00 GHz. This sample rate is higher than required by a large margin as the transducers used to take measurements at GTRC often have a slower response time than 1 second.

Feature	Priorit	Status
Real Time Data Streaming/Display from all measurement devices	1	Done
Easy and simple data exporting to excel	1	Done
Easy and simple data exporting to another format? As a crash logger	1	Mitigated
In GUI setup and real time calculation of EQ Ratio	1	Done
Real Time Graphing of Line Temps	2	Done
Real Time Graphing of Line Pressures	2	Done
Sub Panel UI to set alarm thresholds	2	Done
Output Control - thresholds and shutoffs	2	
Ability to either show current or show rolling average	2	
Real Time Graphing of Line Flowrates	3	Done
Adjustable number of lines in use	3	Done
Adjustable sampling/polling rate	3	Done
Pull up a subpanel which shows a snapshot of the last x minutes of data?	4	
Colour gradient for the different chambers on the rig showing temp dist	4	
Toggle from mass-flowrate to volumetric-flowrate	4	
Integrated Camera Feed	5	

Figure 27 - Feature List as of Version 6.3a

Figure 27 shows the completion of the Feature List as of Version 6.3a. Over half the features set out at the start of the project have been implemented. The need for a crash-logger was mitigated by the batching process used to write the .csv file writes once per second. If the VI were to crash no data written before the crash would be lost.

Currently the live display loop simply shows the most recent recorded value, rather than an average since the previous value. Methods for calculating a rolling average were developed early in the process, but never implemented into the final build.

Output Control remains unexplored.

5.2 Block Diagram

This section contains the final block diagram code used to drive the front panel shown in Figure 26. A detailed breakdown of the can be found throughout the Development section of the report.

5.2.1 Acquisition Loop

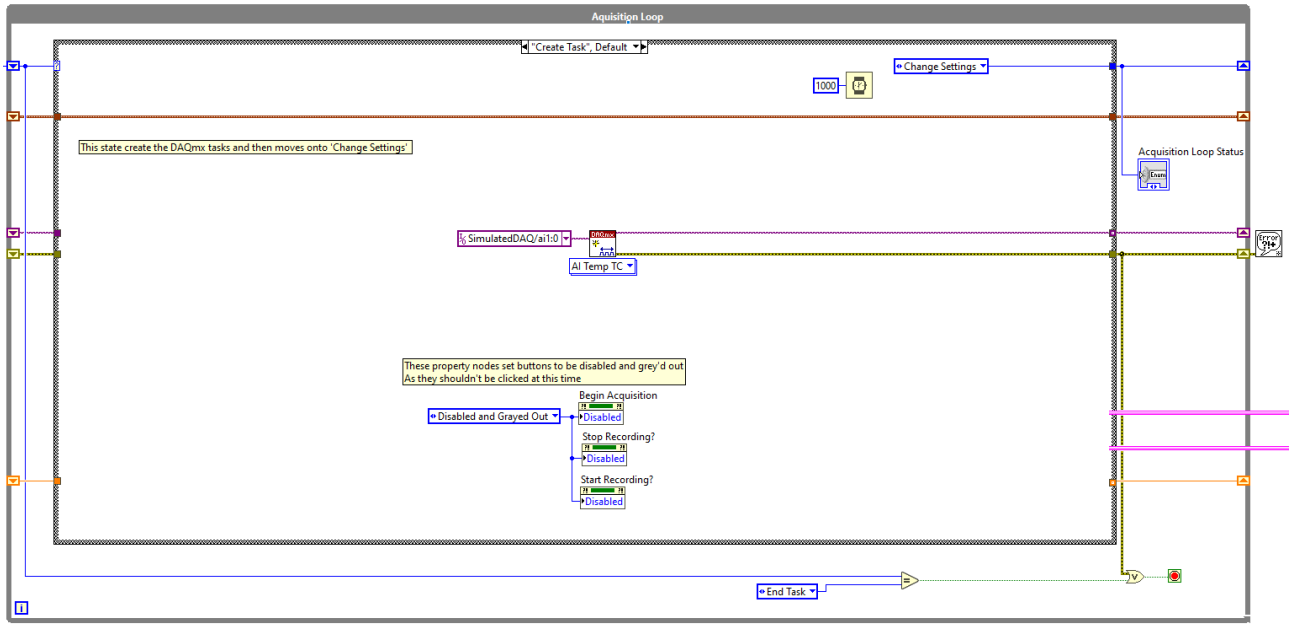


Figure 28 – 'Create Task' (Starting State) of the Acquisition Loop

Figure 28 shows the state used to create the DAQmx read task, this executes when the VI is first launched. It then moves onto the 'Change Settings' state shown in Figure 29. Where it waits for the user to input the DAQ settings and press the begin Acquisition button, changing the state to 'Check Settings' shown in Figure 30.

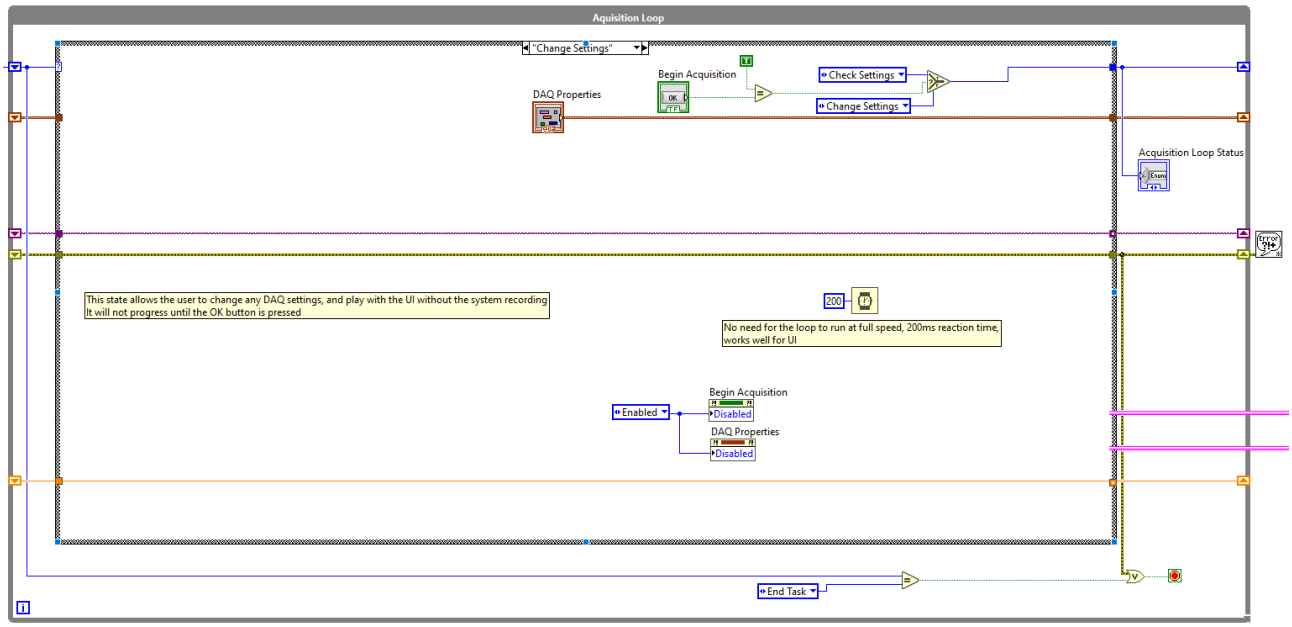


Figure 29 – 'Change Settings' State of the Acquisition Loop

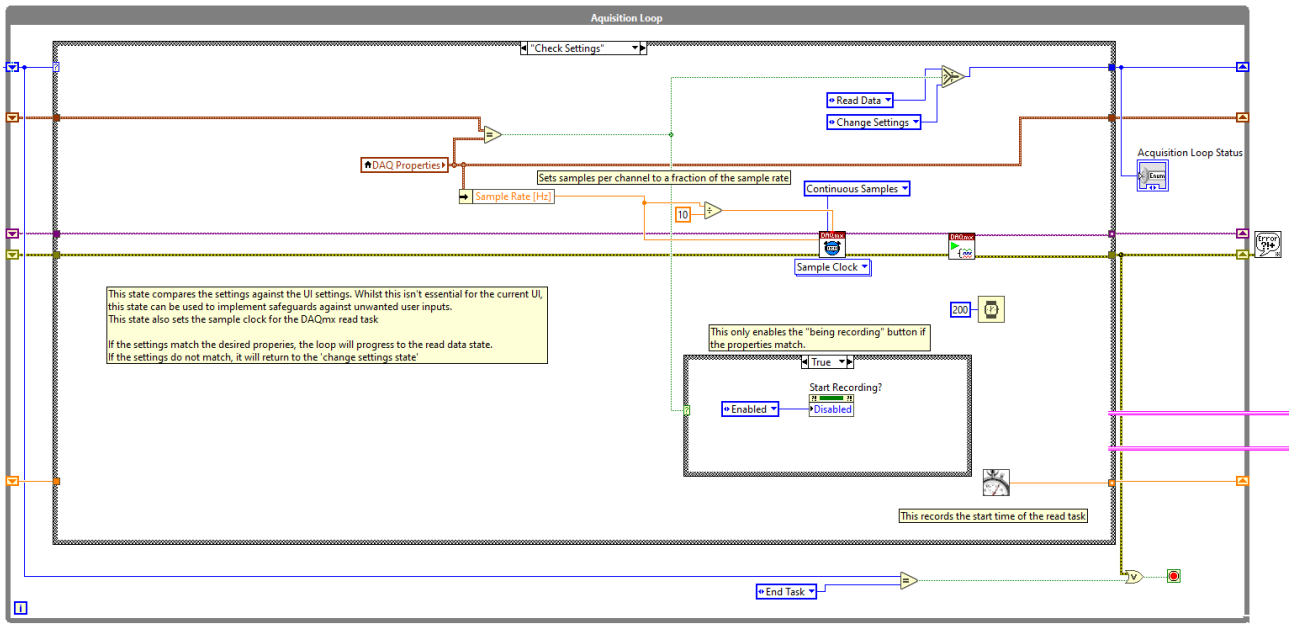


Figure 30 – 'Check Settings' State of the Acquisition Loop

This state (Figure 30) compares the DAQ settings, sets the sample rate of the DAQmx task, and acquires a datum start time. If the Boolean check is passed it enables the 'Start Recording' button used to being recording in the next phase and moves the state-machine to the state shown in Figure 31. This state is used to acquire the data using DAQmx and simulate the Line Data and Rig Temperatures. Channel writers are used to asynchronously communicate this data with the Export Loop and UI Display Loop. Once the 'Stop Recording' button is pressed the loop waits for the channel writers to finish being read and then moves the state to 'End Task'.

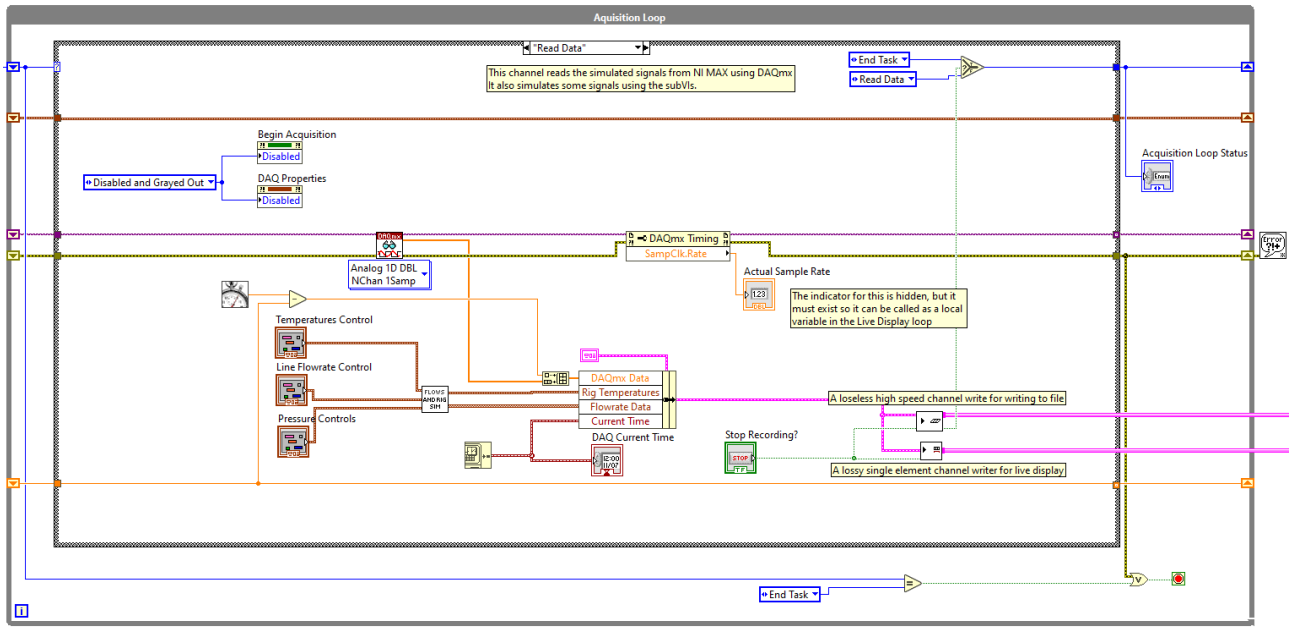


Figure 31 – 'Read Data' State for the Acquisition Loop

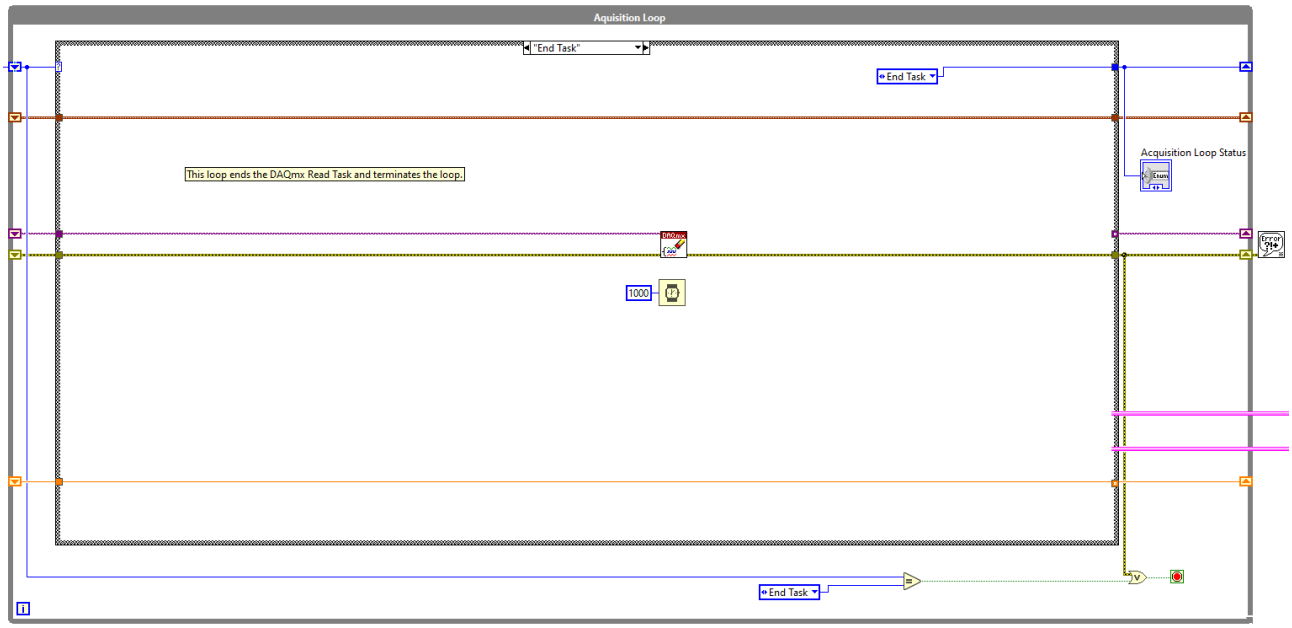


Figure 32 – 'End Task' State of the Acquisition Loop

This 'End Task' state simply stops and clears the DAQmx Read task then terminates the loop.

5.2.2 Export Loop

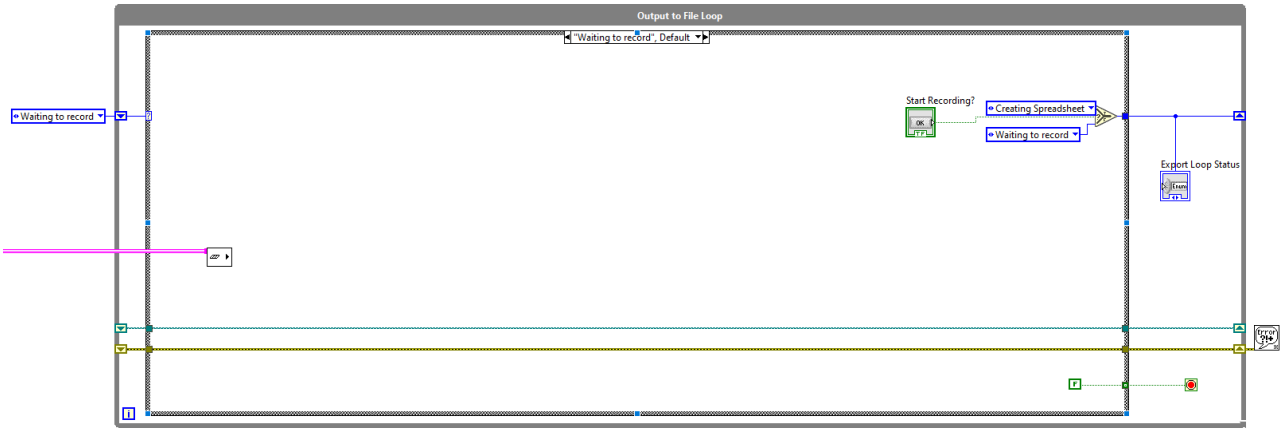


Figure 33 – 'Waiting to Record' State of the Export Loop

Figure 33 shows initial state for the Export Loop, a channel reader has been added for reasons discussed in Section 4.2.4 of the report. The loop progresses from this state onto 'Creating Spreadsheet' (Figure 34) when the 'Start Recording' button is pressed. To prevent this button from being pressed early it is only enabled once the Acquisition Loop is ready.

The code shown in Figure 34 creates the .csv for writing. It populates the headers using a SubVI. If the user hasn't entered a file path yet it will prompt the user for one. If this state encounters any errors, it will display the error in a pop-up and return the case-structure to the previous state. Otherwise, the case-structure progresses to the 'Recording' state.

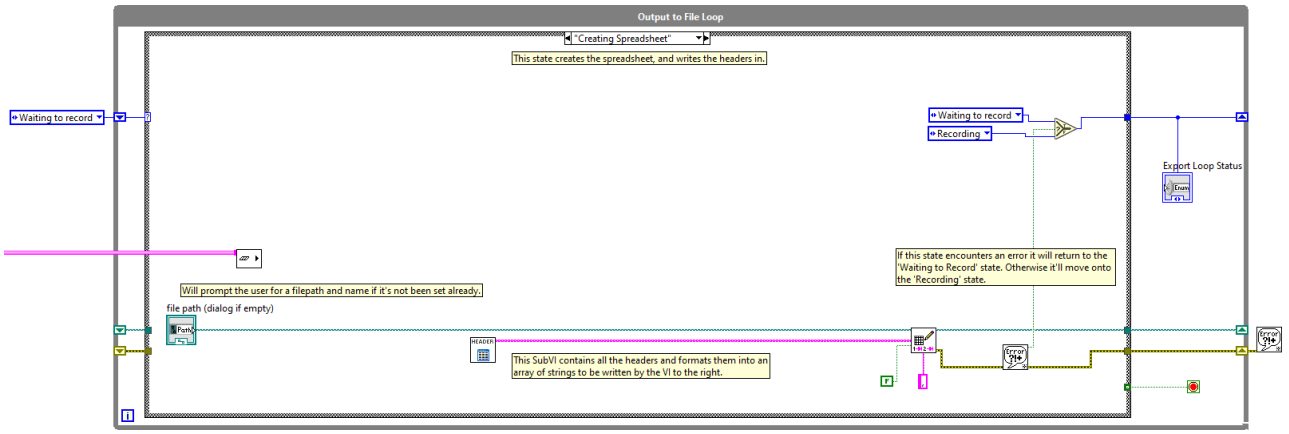


Figure 34 - 'Creating Spreadsheet' State of the Export Loop

Figure 35 below shows the ‘Recording’ state, this is used to record all data coming from the acquisition loop. A for-loop is used to batch the data coming from the channel reader, a SubVI is used to organise the line data, and all data recorded by the acquisition loop is combined into an array of strings for writing to file. Once the channel reader is no longer receiving data and is finished reading the ‘done?’ terminal is used to progress the loop to the ‘Finished Recording’ state. Shown in Figure 36 this state simply terminates the while-loop. No code is required to close the file as the ‘Write Delimited Spreadsheet’ VI used does this automatically. (NI, 2023)

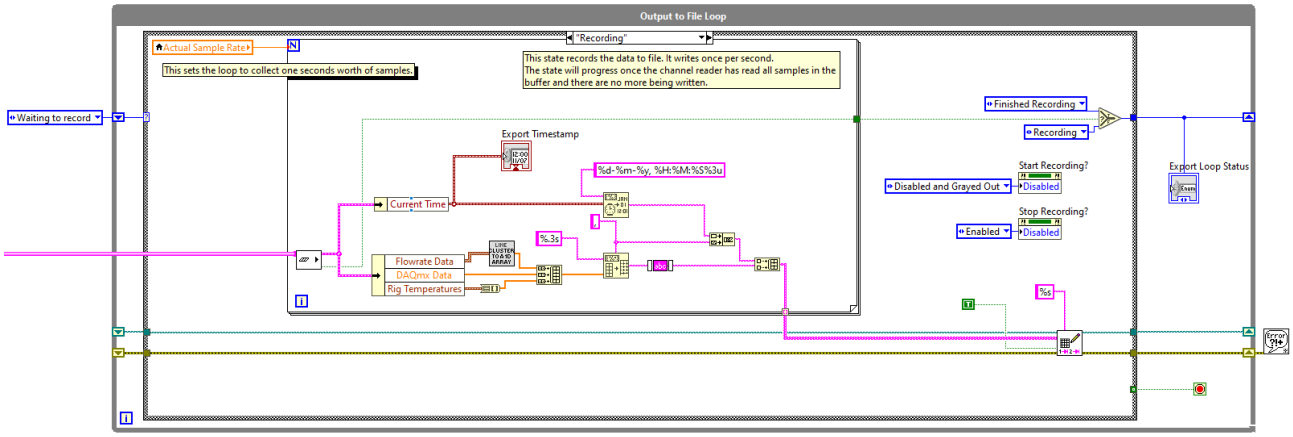


Figure 35 - 'Recording' State of the Export Loop

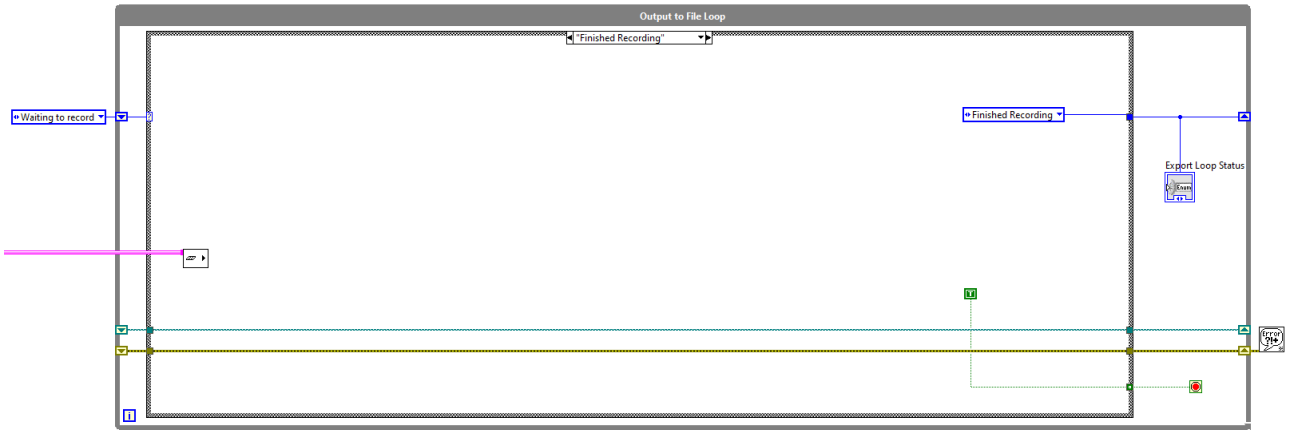


Figure 36 - 'Finished Recording' State of the Export Loop

5.2.3 Live Display (UI Loop)

This is the largest of the loops, to ensure legibility it's shown on the following page by Figure 38.

This loop is responsible for displaying the data acquired from the acquisition loop. Flowrate Data and Rig Temperatures are display using simple indicators. The for-loop dominating this loop is used to index and obtain the correct values for calculating the Driving Variables using the SubVI. The different states for the case-structure used here are shown below in Figure 37.

At the top of this for-loop are wires used to unbundle the Temperature and Pressure, so they can be indexed and re-clustered for display on the charts.

The box in the top-right allows the user to control the refresh rate of the UI (Figure 13, Section 4.2.5), in the bottom right is the code used to handle the Alarm Thresholds for the Rig Temperatures (Section 4.2.7).

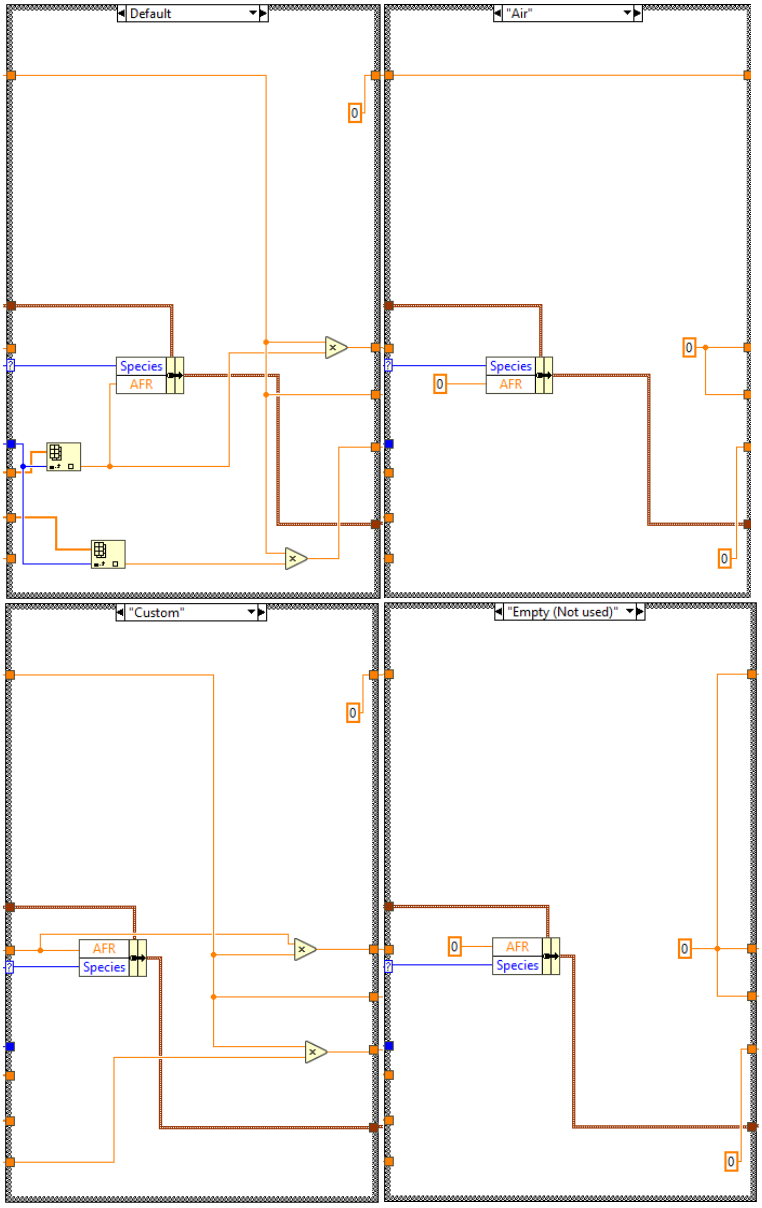


Figure 37 - The case-structures used in the UI Loop for the Combustion Calculations

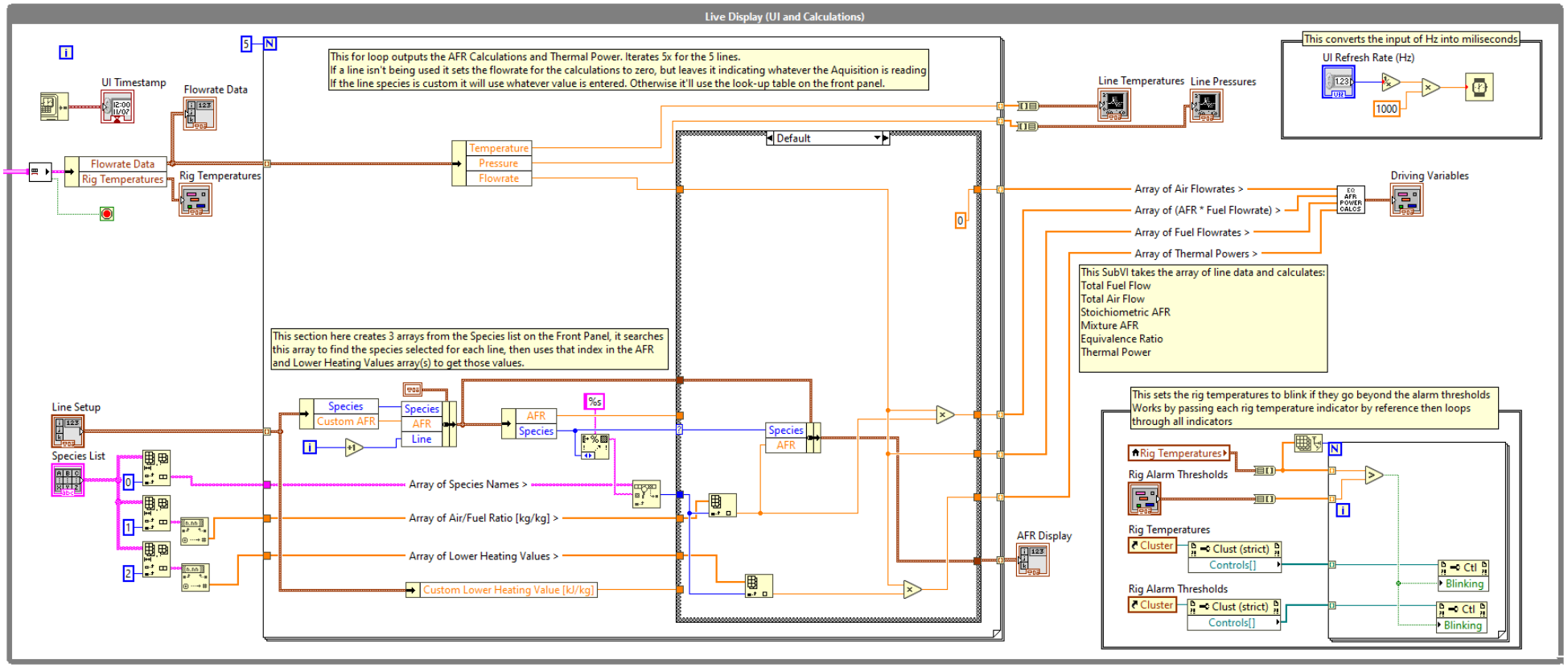


Figure 38 - The finished Live Display (UI) Loop

5.3 Verification of Thermodynamics Calculations

After introducing AFR, EQ Ratio, and Thermal Power displays into the VI testing was required to verify the code was correctly setup. Table 1 shows the input values used to setup the experiment.

Line	Substance	AFR	LHV [kJ/kg]	Flowrate (F) [g/s]
1	Air	0	0	9
2	Ethanol	9	26.7	8
3	Butane	14.98	45.75	3
4	Custom	7	5	4
5	Empty	0	0	0

Table 1 - Input Values for EQ Ratio, AFR, and Thermal Power Calculations

This setup was chosen as it uses each of the four available options for a line (Air, Fuel, Custom, Empty), and uses two different fuels to verify the look-up tables functionality. The VI calculates and displays: Total Fuel Flow, Total Air Flow, Stoichiometric AFR, Mixture AFR, EQ Ratio, and Thermal Power. Equations for these values are shown below.

$$Stoichiometric\ AFR = \frac{\sum\ (F_i * AFR_i)}{\sum F_i} \qquad \text{[Eq 1.]}$$

$$Mixture\ AFR = \frac{Total\ Air\ Flow}{Total\ Fuel\ Flow} \qquad \text{[Eq 2.]}$$

$$EQ\ Ratio = \frac{Stoichiometric\ AFR}{Mixture\ AFR} \qquad \text{[Eq 3.]}$$

$$Thermal\ Power = \sum\ (F_i * LHV_i) \qquad \text{[Eq 4.]}$$

Using the input values shown in Table 1 and the equations above theoretical values were calculated. Versions 6.2a which uses wires for the calculations and 6.2b which uses a formula node were tested. In both cases the values came out identically to four decimal places. Confirming the code used for the calculations is correct. The results of this test can be found in Section 9.3.1. It should be noted that it's possible some of the values for AFR and Thermal Power in the Look-up table are inaccurate as they're merely for demonstration purposes. This doesn't impact this testing as the values used by the software and in the theory kept consistent.

6 Discussion

The DAQ system designed here is believed to be an improvement on the existing system. The UI has improved functionality, with simpler configuration of EQ Ratio calculations and Alarms. This could be improved further with the use of Datalogging Supervisory Control.

Live graphing of historical data can give the individuals running the test campaign a better understanding of their flow behaviour. Line Temperature and Pressure have been used, but with LabVIEW it would be trivial to display other signals.

The file I/O method on the existing DAQ system is limited by the hardware it's running on. LabVIEW's file I/O tools are exceptional, only a very basic method has been employed here.

Whilst the code is specific to this application, the methods and architecture are transferrable. State-machines and Producer-Consumer loops are commonly used for Data Acquisition. The code produced provides a sound base for further development.

A limitation of this project is the lack of hardware integration, and therefore no avenue for output control. An effort was made in the project to use a MyRIO to capture data from a K-Type Thermocouple. This was unsuccessful, producing irrelevant data. This was dropped from the project in favour of using the time to further develop the software aspects. It's speculated that the thermocouple itself was broken as viewing the signals in NI MAX showed they were being acquired but did not respond to changes in temperature.

Whilst the scope was to design a new data acquisition system, the simulated signals only serve to demonstrate LabVIEW's capabilities. True data acquisition has not been demonstrated. Although the code developed would not differ significantly if real signals were being acquired through DAQmx, as NI MAX has been used to simulate analogue signals.

7 Conclusion and Further Work

The aims of the project have been met, and the objectives carried out. Through the Mimic a recreation of the existing UI was built. A new DAQ system was developed. Testing of the software validates the combustion calculations, and all functionality required has been implemented. This project also provided an opportunity to learn and mature LabVIEW coding skills.

Importantly the suitability of LabVIEW as a replacement for the existing DAQ has been demonstrated.

The code produced is the first step towards implementing a new system at the GTRC. There is plenty of room for expansion and further development. The inclusion of new functionality such as the real-time graphing, higher sample rate, an easily configurable UI, and a robust acquisition and export system could drastically improve the quality of life for researchers, and the efficiency of test-campaigns.

Significant work would be required to bring this code to production. A new computer would need to be installed at GTRC to run a contemporary version of LabVIEW. Windows 3.1 has significantly worse timing accuracy than Windows 10 and beyond. (NI, 2022)

Use of the Datalogging Supervisory Control Module is encouraged, static and dynamic code analysis using LabVIEW toolkits could be used to further improve the code. LabVIEW's signal processing tools have not been touched upon in this project and could be explored for real time analysis of experimental conditions.

Hardware integration would need to be investigated, the exact instrumentation in use at GTRC would need to be considered. The DAQmx palette implemented is a good choice for this application. New code would be required to organise the data produced by the increased number of channels in the DAQmx Read Task.

This project has laid the foundation for a new and improved DAQ system at the GTRC, and there is great potential for further development and implementation.

8 References

AristosQueue-(NI), 2015. *Getting Started With Channel Wires.* s.l.:s.n.

Bowen, P. et al., 2009. *GTRC First Year Contribution to Progress in Combustion and Energy Systems.* Boston, s.n.

Cardiff University, n.d. *Flexis.* [Online]
Available at: https://www.cu-gtrc.co.uk/flexis

Dayley, T., 2008. *Is there a "Get Cluster Names" VI?.* s.l.:s.n.

Essick, J., 2016. *Hands-On Introduction to LabVIEW for Scientist and Engineers.* 3rd ed. Oxford: Oxford University Press.

Essick, J., 2016. Hands-On Introduction to LabVIEW for Scientists and Engineers. In: *Hands-On Introduction to LabVIEW for Scientists and Engineers.* Third ed. Oxford: Oxford University Press, pp. 518-530.

Feel-soon, K., 2008. Gas Turbine Data Acquisition and Monitoring System for Combined Cycle Power Plant. *International Journal of KIMICS,* 6(4), pp. 405-410.

Gas Turbine Research Centre, n.d. *Diagnostic Techniques.* [Online]
Available at: https://www.cu-gtrc.co.uk/content/24
[Accessed 27 03 2023].

H2020 Raptor (863969), 2019. *Research of Aviation PM Technologies, Modelling and Regulation.* [Online]
Available at: https://cordis.europa.eu/project/id/863969
[Accessed 24 March 2023].

Hochgreb, S., 2020. *Details of Grant for Tracer-free, non-intrusive, time- and space-resolved temperature and scalar measurements.* [Online]
Available at: https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/T030801/1
[Accessed 2023 March 24].

KrauB, A., Weimar, U. & Göpel , W., 1999. LabVIEW for Sensor Data Acquistion. *Trends in Analytical Chemistry,* 18(5), pp. 312-318.

LabVIEW Courses Youtube Channel, 2022. *LabVIEW Data Acquisition Playlist.* [Online]
Available at: https://youtube.com/playlist?list=PLKRZJpHpKYx7o1MC9gAWIV6LBs3Wv6F8X
[Accessed 25 03 2023].

LabVIEW Wiki, 2019. *G.* [Online]
Available at: https://labviewwiki.org/wiki/G
[Accessed 26 03 2023].

McQuillan, T., 2018. *Tom's LabVIEW Adventure.* [Online]
Available at: https://www.youtube.com/@TomsLabVIEWAdventure/featured
[Accessed 28 03 2023].

McQuillan, T., n.d. *Tom McQuillan's Linkedin profile.* [Online]
Available at:
https://www.linkedin.com/in/thomasmcquillan/?original_referer=https%3A%2F%2Fwww.youtube.com%2F
[Accessed 28 03 2023].

Mihura, B., 2001. *LabVIEW for Data Acquisition.* s.l.:Prentice Hall PTR.

National Instruments, 2023. *Formula Node Syntax.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview/page/lvhowto/formula_node_syntax.html

Neale, Z., 2019. *NI-DAQmx multi-channel data acquistion LabVIEW program.* [Online]
Available at: https://youtu.be/fIy6XT3CdPQ
[Accessed 25 03 2023].

nepomnyi & Bob_Schor, 2022. *How to save data from multiple arrays into one file using "Write Delimited Spreadsheet.vi"?.* [Online]
Available at: https://forums.ni.com/t5/LabVIEW/How-to-save-data-from-multiple-arrays-into-one-file-using-quot/td-p/4264301
[Accessed 2 03 2023].

NI Global, 2010. *Programming Data Acquistion Applications with NI-DAQmx Functions.* [Online]
Available at: https://youtu.be/alHwllqoLj4
[Accessed 28 03 2023].

NI, 2012. *Getting Started with LabVIEW Datalogging and Supervisory Control Module.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview-datalogging-and-supervisory-control-module-getting-started/resource/372946d.pdf

NI, 2012. Improving an Existing VI. In: *LabVIEW Core 2 Course Manual.* Austin: National Instruments Corporation, pp. 5.1-5.10.

NI, 2012. *LabVIEW Core 2 Course Manual.* August 2012 ed. Austin: National Instruments.

NI, 2013. *LabVIEW Core 3 Course Manual.* August 2013 ed. Austin: National Instruments Corporation.

NI, 2014. *LabVIEW Core 1 Participant Guide.* November 2014 ed. Austin: National Instruments.

NI, 2020. *Knowledge.NI.* [Online]
Available at: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YI6UCAW&l=en-GB
[Accessed 21 March 2023].

NI, 2021. *Considerations With NI-DAQmx Simulated Devices.* [Online]
Available at: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019Nw0SAE&l=en-GB
[Accessed 19 March 19].

NI, 2021. *LabVIEW VI Analyzer Toolkit Download.* [Online]
Available at: https://www.ni.com/en-gb/support/downloads/software-products/download.labview-vi-analyzer-toolkit.html#411412
[Accessed 19 March 2023].

NI, 2022. *Accuracy of Software-Timed Applications in LabVIEW.* [Online]
Available at: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P9QiSAK&l=en-GB
[Accessed 29 March 2023].

NI, 2022. *Create and Configure a LabVIEW SubVI.* [Online]
Available at: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YK4VCAW&l=en-GB

NI, 2022. *Design Considerations in LabVIEW.* [Online]
Available at: https://www.ni.com/en-gb/support/documentation/supplemental/22/design-considerations-in-labview-.html
[Accessed 28 March 2023].

NI, 2023. *Datalogging and Supervisory Control Module.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview-datalogging-and-supervisory-control-module/page/lvdscconcepts/dsc_module_defa.html

NI, 2023. *High Resolution Relative Seconds VI.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview/page/glang/high_res_rel_sec.html
[Accessed 15 03 20].

NI, 2023. *LabVIEW Style Checklist.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview/page/lvdevconcepts/checklist.html
[Accessed 19 03 2023].

NI, 2023. *Minimizing SubVI Overhead.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview-nxg-creating-first-application/page/minimize-subvi-overhead.html

NI, 2023. *Strategies for Improving VI Execution Speed.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/dagexpress/page/strategies-for-improving-vi-execution-speed.html
[Accessed 02 March 2023].

NI, 2023. *VI Execution Speed.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/vi_execution_speed.html

NI, 2023. *Write Delimited Spreadsheet VI.* [Online]
Available at: https://www.ni.com/docs/en-US/bundle/labview/page/glang/write_delimited_spreadsheet.html
[Accessed 29 03 2023].

Omega, n.d. *Thermocouple Response Time.* [Online]
Available at: https://www.omega.co.uk/temperature/z/thermocoupleresponsetime.html
[Accessed 3 March 2023].

RAPTOR (863969), 2019. *Details of Grant for Assessing aViation emission Impact on local Air quality at airports: TOwards Regulation.* [Online]
Available at: https://cordis.europa.eu/project/id/814801
[Accessed 24 03 2023].

Valera-Medina, A., 2019. *Storage of Ammonia For Energy (SAFE) - AGT Pilot Grant.* [Online]
Available at: https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/T009314/1
[Accessed 24 March 2023].

Wei, Z., Liu, P., Wang, F. & Wang, T., 2020. *Design of Gas Turbine State Data Acquisition.* Singapore, Springer Singapore, pp. 48-55.

9 Appendices

9.1 Appendix A - Nomenclature

Table 2 - Nomenclature

Term	Meaning
.csv	.csv is the file extension for the “comma separated variables” format. Easily read by Excel, this is a plaintext file where the variables are separated by commas
.exe	.exe is the filename extension given to an executable file
Air/Fuel Ratio	The ratio of mass flowrates of Air and Fuel
Block Diagram	A term used to describe the code created in LabVIEW
Bug/Bugs	A software term used to describe a problem or mistake within code
Case-Structure	LabVIEW’s visual programming equivalent of an if-statement
DAQ	Data Acquisition
DAQmx (palette)	A LabVIEW palette used for data acquisition.
Design Pattern	A software design term for a common repeatable solution to a common problem
Driving Variables	A cluster of EQ Ratio, Thermal Power, and AFR. Used as benchmarks or targets during a test-campaign
Enum	A fixed list of strings corresponding to an integer
Equivalence Ratio	The ratio of Stoichiometric AFR to the Actual AFR of the mixture
Flow Data/Line Data	The array of clusters representing Temperature, Flowrate, and Pressure for each of the 5 inlet lines.
For-Loop	A loop in code which runs for a set number ‘for’ a set number of iterations
Front Panel	The visible user interface created by LabVIEW code
G [Code]	The official name of the code produced within LabVIEW
GTRC	[the] Gas Turbine Research Centre
HPOC	High-Pressure Optical Combustor
Local Variables	A LabVIEW node used to access any variables on the block diagram without using wires
NI MAX	National Instruments Measurement and Automation Explorer
Nodes	The name for individual icons on the block diagram
Palette	The name given in LabVIEW to a collection of similar or related nodes
Property Nodes	Property Nodes can read or change specific properties of an object on the front panel.
Rig Temperatures/Rig Data	The cluster of temperature measurements taken of the HPOC rig itself.
SCADA	“Supervisory Control And Data Acquisition” the current DAQ system in use at GTRC
SubVI	LabVIEW’s equivalent of a function
TDMS	A more advanced file I/O system - https://www.ni.com/en-gb/support/documentation/supplemental/06/the-ni-tdms-file-format.html
Test Campaign	An experiment carried out using the HPOC at GTRC
Thermal Power [W]	The Lower Heating Value of a substance being combusted multiplied by its flowrate.
VI	Virtual Instrument – A virtual version of an instrument’s control/display panel
While-Loop	A loop in code which runs ‘while’ a certain condition is true.

9.2 Appendix B - Additional LabVIEW Code

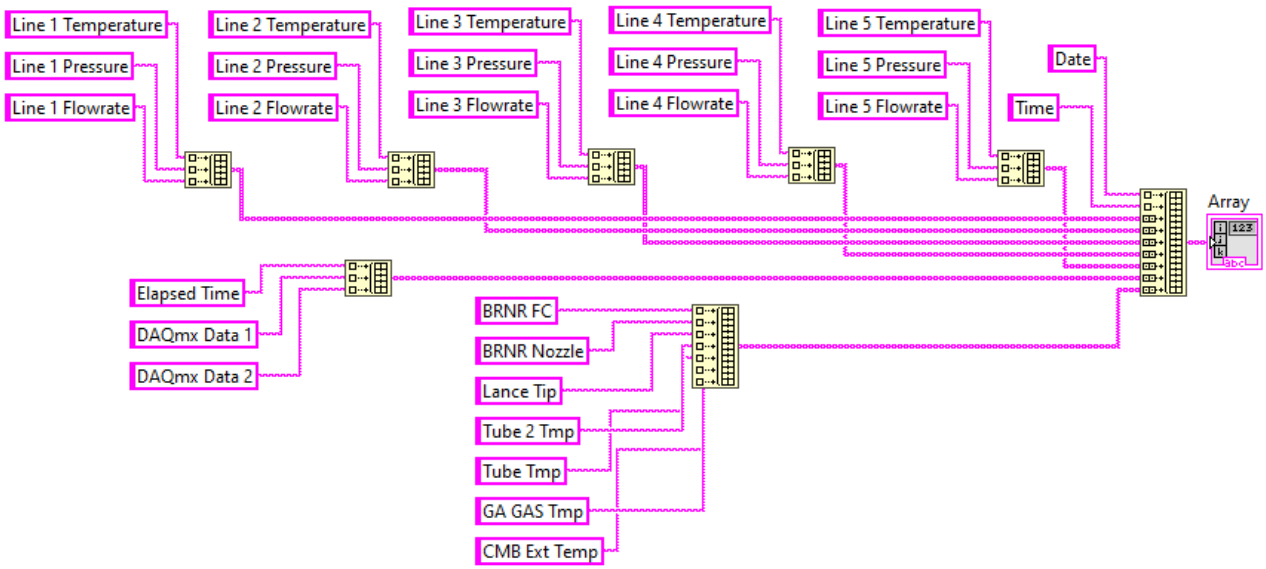


Figure 39 – The 'Create Spreadsheet Headers' VI called by the Export Loop to add the spreadsheet headers when creating the output file.

9.3 Appendix C: Raw Data

9.3.1 Testing of Combustion Calculations

Table 3 - Input Data used to Setup the testing of combustion calculations.

Line	Substance	AFR	LHV [kJ/kg]	Flowrate (F) [g/s]
1	Air	0	0	9
2	Ethanol	9	26.7	8
3	Butane	14.98	45.75	3
4	Custom	7	5	4
5	Empty	0	0	0

Table 4 – Results for the testing of combustion calculations

Results	Total Fuel [g/s]	Total Air [g/s]	Stoich AFR	Mixture AFR	EQ Ratio	Thermal Power [W]
Theory	15.0000	9.0000	9.6627	0.6000	16.1044	0.3709
6.2a (Wires)	15.0000	9.0000	9.6627	0.6000	16.1044	0.3709
6.2b (Formula Node)	15.0000	9.0000	9.6627	0.6000	16.1044	0.3709

9.4 Appendix D: Additional Functionality, Bug Fixes, Improvements

Appendix D is a collection of changes or new functions to implement that could improve the final build.

- Dynamic precision for time values based on the sample rate.
- DAQmx Data should be bundled by name into a cluster.
- Generate the spreadsheet headers dynamically by passing a reference into a property node, rather than manually creating a sting from string constants.
- Disable and Grey out the Custom AFR and Custom Thermal Power Control by default, only to be enabled when a ‘Custom’ is selected on the species control
- Adiabatic Flame Temperature Calculations
- Currently the look-up table is not persistent and unless manually set to be the new default value by the user, it will not save any changes.
- Load default parameters via an INI file
- Output all variable and system settings to a text file upon recording being stopped. Either by adding it to the end of the .csv or a new file. This would allow users to know for sure how the system was configured throughout a test campaign.
- Make use of Datalogging Supervisory Control mentioned in Section X.
- Rebuild Export Loop using TDMS for file writing, rather than the ‘Write Delimited Spreadsheet’ VI. This would further optimise the loop and reduce overhead.
- Implement ‘High’ alarm thresholds linked to output control.
- Setting the file path and amending it with a .csv extension automatically.
- Replace the Line Number Display with a display for the current substance in the line.
- Further optimise the UI Look by moving the local variables used to obtain the references for Rig Temps and Alarm Thresholds outside the loop. (At the cost of code readability)
- Improve the overall design of the front panel.
- Create an installer for the project to deliver the .exe.
- Integration of the live camera feed
- A networked DAQ System