2019-01-01

Development and Implementation of the tRIAC Data Acquisition and Control System

Raymundo Mendivil Rojo
University of Texas at El Paso, raymundo.rojo46@gmail.com

DEVELOPMENT AND IMPLEMENTATION OF THE TRIAC DATA ACQUISITION AND CONTROL SYSTEM

RAYMUNDO MENDIVIL ROJO III

Master's Program in Mechanical Engineering

APPROVED:

Jack Chessa, Ph.D., Chair

Joel Quintana, Ph.D.

Virgilio Gonzalez, Ph.D.

Stephen Crites, Ph.D.
Dean of the Graduate School

Dedication

I would like to dedicate this thesis to my family whom have supported me every step of the way and the friends I have made during my time at UTEP. The experiences have all felt too short.

DEVELOPMENT AND IMPLEMENTATION OF THE TRIAC DATA ACQUISITION AND CONTROL SYSTEM

by

RAYMUNDO MENDIVIL ROJO III, BSPHYS, MSAS

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Mechanical Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

May 2019

Acknowledgements

One of the first people I met at UTEP was Luz Bugarin. I knew I wanted to be a part of the cSETR and after a semester of bugging her out of passion and persistence, she hired me on the team. Since then she has been one of the kindest and most supportive people during my time at UTEP, and I am forever grateful for the opportunity she provided and for the entire team and various projects that Dr. Choudhuri has made possible.

All my cSETR colleagues and friends have all been a delight to work with and learn from. I wholeheartedly thank Jason Addams for his guidance, wealth of knowledge, and D&D shenanigans, Manny Herrera for being an attentive team lead who lead by example, Corey Hansen for his incredible work ethic and punny sense of humor, Adrian Welch for being a supportive friend and who has risen to the challenge to take my place, Mariana Chaidez for helping me with modeling with NX and getting acquainted with the team and our project, Marissa Garcia for being a listening ear and positive influence, Debbie Ortega for her diligence in work, in learning and in giving sass, Linda Hernandez who is a perpetual ball of sunshine that rarely lets anything get in the way of her responsibilities, Steven Torres for helping me prepare the MICIT and providing me electrical wiring challenges to figure out and learn from, and Justin Vanhoose for offering a great balance of productivity and having fun both at work and outside. My time with UTEP felt too short.

I also thank my committee, Dr. Jack Chessa, Dr. Joel Quintana, and Dr. Virgilio Gonzalez, for taking the time to both read this thesis and give feedback during my defense. Additionally, I also thank them for their patience as I shown this to them late into the semester.

Abstract

The Technology Research and Innovation Acceleration Park (tRIAC) located at Fabens, Texas, was established for the ongoing development and testing of the miro center for Space Exploration and Technology Research's (cSETR's) liquid methane and liquid oxygen rocket engines. These engines range from a 5 lbf Reaction Control Engine (RCE) to the 500 lbf and 2000 lbf Centennial Restartable Oxygen Methane Engines, called CROME and CROME-X respectively. Alongside their development, a comprehensive control system was built in the Modular Instrumentation and Control Interface Trailer (MICIT) and is equipped with National Instrument's LabVIEW software and hardware. The LabVIEW code is based off a provided project template already set up with a robust groundwork with state machine logic and data enqueueing. This groundwork is then customized to allow for modularity on the software and to incorporate a Field Programmable Gate Array (FPGA) for system reliability. The reliable monitoring and acquisition of the variety of installed pressure transducers, thermocouples, and flowmeters, and any other compatible instrumentation. With a modular software design that eases adaptation to current and future experimental setups. This document goes into further detail of the development of the software to meet the necessary design criteria.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

A countdown begins. The words "T minus ten…nine…" is blared through the radio or television as a rocket prepares to launch. Since man's first successful mission to orbit, there have been thousands of successful missions beyond our atmosphere to follow. Several rocket engines, utilizing all manner of fuels have been designed, built, tested, and utilized across lengthy development processes. Yet, despite the wealth of knowledge and lessons this history provides, igniting one of these propulsion systems in any context is a momentous and tense occasion every time.

Figure 1.1 Saturn V lifting off, courtesy of NASA

Issac Assimov humorously described the mentality behind those working the in the space industry with a quote in his book titled "Ignition!": *"….anyone working with rocket fuels is*

outstandingly mad. I don't mean garden-variety crazy or a merely raving lunatic. I mean a record-shattering exponent of far-out insanity." For readers, it was a good bit of comic relief. However, for members of the Center for Space Exploration and Technology Research (cSETR) over at the University of Texas at El Paso (UTEP), Assimov's quote takes up a new meaning.

Figure 1.2 Early rendering of the tRIAC facility.

The cSETR, in a partnership with NASA and the El Paso county, has established The Technology Research and Innovation Acceleration Park (tRIAC) located at Fabens, Texas. This is to be cSETR's facility that will house its ongoing research and development of rocket propulsion systems. At the time of this writing the Fabens facility houses two, one-hundred-gallon stainless steel tanks designed to hold and deliver liquid methane and liquid oxygen to a prototype engine. We are the very "outstandingly mad" researchers Assimov described. It is less of a jab and more of a real and shocking reminder of the dangers we are exposed to and the training and respect these

substances demand. Rightfully so, a great amount of care was taken in every facet of the design. Starting from the storage tanks I just described, the plumbing in the propellant delivery system, the testing procedures, and in the case of this thesis, the data acquisition hardware and software that will monitor and control the system which this thesis will explore.

Figure 1.3 The MICIT trailer

With all of this said, there is always improvements to be made, therefore it is important to critically view all the features of the design to come up with fresh ideas that may be more readily implemented as design and manufacturing capabilities expand throughout the years.

The data acquisition and control system will be housed in the Modular Instrumentation and Control Interface Trailer (MICIT). (Chaparro, 2017), detailed the groundwork for the MICIT and its initial design. Some of his work will be reiterated here, but the focus will shift more on how to utilize the trailer.

The hardware capabilities will be discussed first, as it is what the MICIT can do that will then dictate how to develop the associating LabVIEW software.

Chapter 2: Design Overview and Requirements

The tRIAC is expected to see ongoing development for the many years to come. During this development, the MICIT system was built to be a shared resource. It was over-engineered to accommodate not just the currently existing projects, which can continue to see slight modifications during their own development, but also any new projects that will be incorporated in the tRIAC. The intended purpose of the MICIT then leads to its three fundamental design requirements:

1. Overall system safety and reliability
2. System modularity
3. Streamlined design

As it is going to monitor and manage rocket propulsion systems, the first and most critical design requirement of the MICIT system is that it is safe and reliable. The control software provides another layer of defense to protect the researchers, facility, and the test article. At the same time, the system also must dependably acquire data from a large assortment of instrumentation, as there is a limited amount of fuel available on any given testing day.

The core component of the MICIT that enables the high degree of reliability is a Field Programmable Gate Array (FPGA), featured in the compact Remote Input/Output (cRIO) chassis by National Instruments. In short, an FPGA is an integrated circuit that can adjust its internal logic on a hardware level to accomplish a programmed task. This means that, after an initial set-up, it can function without the need of a software program or operating system, reducing points of failure and overhead. Further detail and the development of the code and logic that drives the FPGA will be extensively elaborated in its dedicated section in this thesis.

The second design requirement of the MICIT is modularity at both the hardware and software level so that switching between different testing platforms is quick and efficient. On the hardware level, the components and overall design are already mainly set in place with room for slight modifications or upgrades. The software on the other hand, is always subject for adjustments as testing plans or plumbing designs change and evolve.

Finally, as a final requirement, all this complexity and robustness must come in a package with a manageable learning curve to work with and adjust. This can include simple additions such as clearly labeled connections, to a software design structure a new programmer can trace and follow. Graduate researchers come and go, and a difficult system loses its long-term reliability for the higher risk of human error it would invite.

Chapter 3: MICIT Capability and Hardware

Upon setting foot in the MICIT, two server racks up are found against the trailer wall. One rack, the power distribution rack, contains all the hardware, power supplies, relays, and wiring to distribute power throughout the entirety of the system. The other rack contains the LabVIEW hardware for control and data acquisition, along with the elaborate wiring to the breadth of sensors to accommodate any test platform, with room for expansion. A closer look of the process behind the wiring can be found in (Hansen, 2019).

For clarity, each individual connection on the front of the MICIT server racks has an address that follows a pattern shown in Figure :

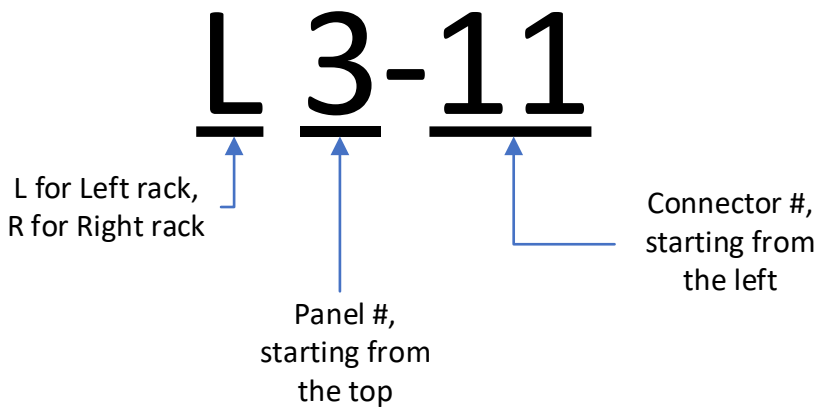


Figure 3.1 Numbering scheme for MICIT's front panel connections

Table 3.1 Outline of MICIT capabilities at time of writing

MICIT Power		
Component	**Capability**	**Notes/Comments**
ACP SMT1500RM2U **Uninterruptable Power supply**	1440 VA OR 1000 Watts	**The wattage sets the power budget for the MICIT in the event of emergency.**
Acopian A8MT500 **8 Volt DC Power supply**	4 Amps max output	(None currently)
TDK-Lambda, FPS100012 **12 Volt DC Power supply**	72 Amps max output	Powers solenoid valves, indicator lights

TDK-Lambda, FPS100024/S **24 Volt DC Power supply**	40 Amps max output	Power for cRIO, cDAQ, pressure transducers, load cell and turbine flowmeter signal conditioners
Data Acquisition		
Component	**Capability**	**Notes/Comments**
cRIO-9066	Built in FPGA, 667 MHz clock, dual core	Built in processor for autonomous operation
cDAQ-9189	Three internal base clocks at 80 MHz, 20 MHz, and 100 kHz	
4x NI-9214 **Thermocouple Modules**	68 Hz sample rate for high speed 0.96 Hz sample rate for high resolution (24-bit resolution)	32 K-type (16 shared with cRIO) 32 E-type
2x NI-9205 **Analog Voltage Input Module**	250 kHz Max sample rate 16-bit resolution (0.152 mV at 10 volts)	64 single ended signals OR 32 double ended signals
NI 9361 **Digital Frequency counter**	Can acquire up to 1 MHz signal	8 Channels. 5 volt differential, or 24 volt RSE input range
12x Omega DP25B-E-A **Programmable Signal Amplifiers**	Programmable with via linear transform of input signal (15 bit resolution)	Built-in excitation DC voltage supply for passive transducers
Front Panel Connections		
Solenoid valve interface board	64 Relay-controlled outputs to deliver 120 vac, 24vdc, or 12 vdc (reconfigurable)	5 amps max per plug
Motor Power and signal	24 vdc power and control signal for EPOS2 motor controller	
Ignitor power and signal	12 vdc power and control signal for coil-on plug spark ignitor	Signal is 100 Hz, 50% duty cycle square wave

3.1 Power Distribution Rack and Hardware

Figure 3.2 Front (left) and rear (right) of the power distribution rack

The Power distribution rack is the taller of the two racks and is directly up against the trailer wall. When walking up to the trailer from the ramp, it will on the left. This rack will house the components used to power the entire MICIT system, which includes solenoid valves and the passive instrumentation devices. Figure provides a high-level overview of how the power is distributed throughout both racks.

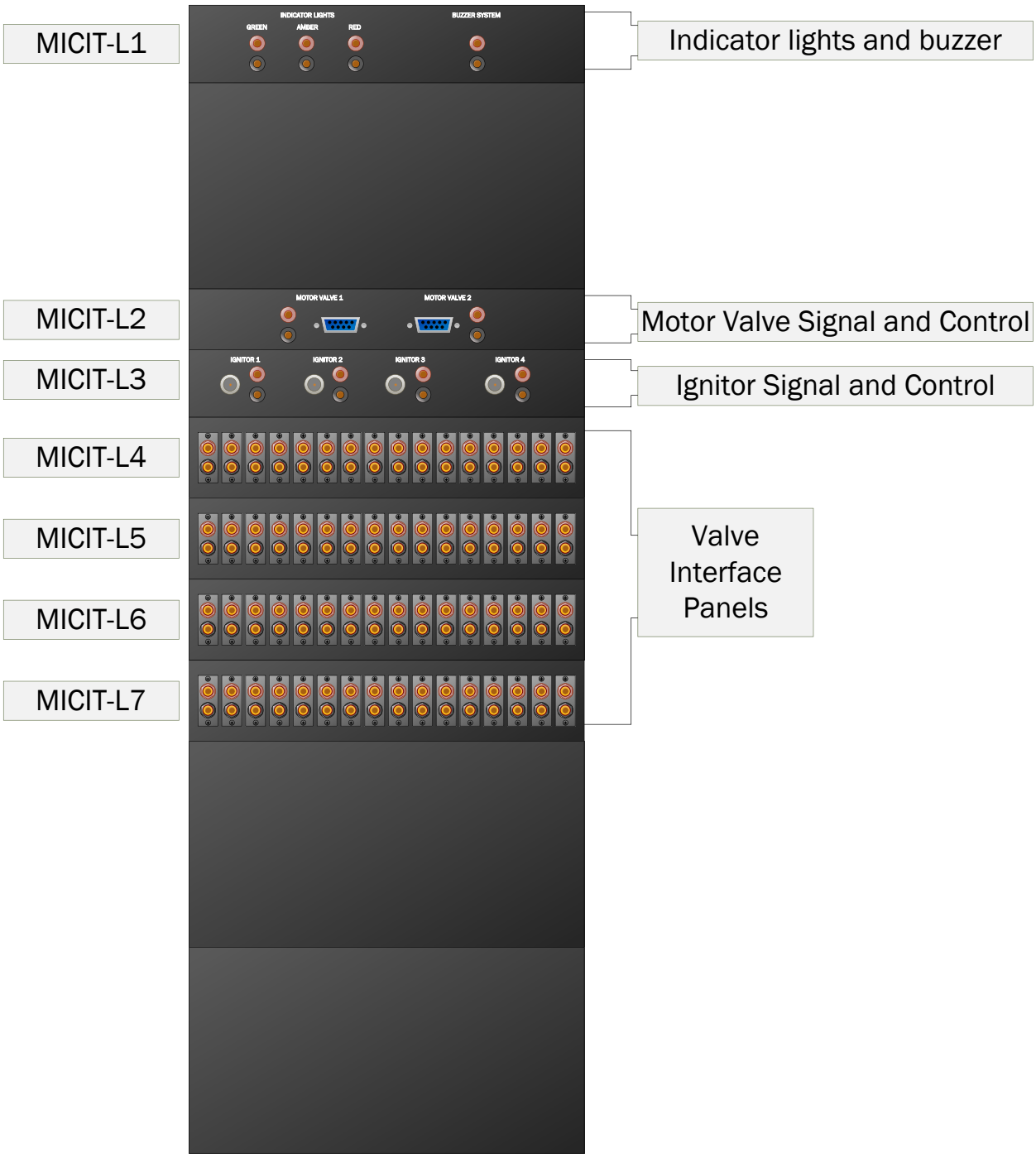


Figure 3.3 Outline of the power distribution rack's front panel

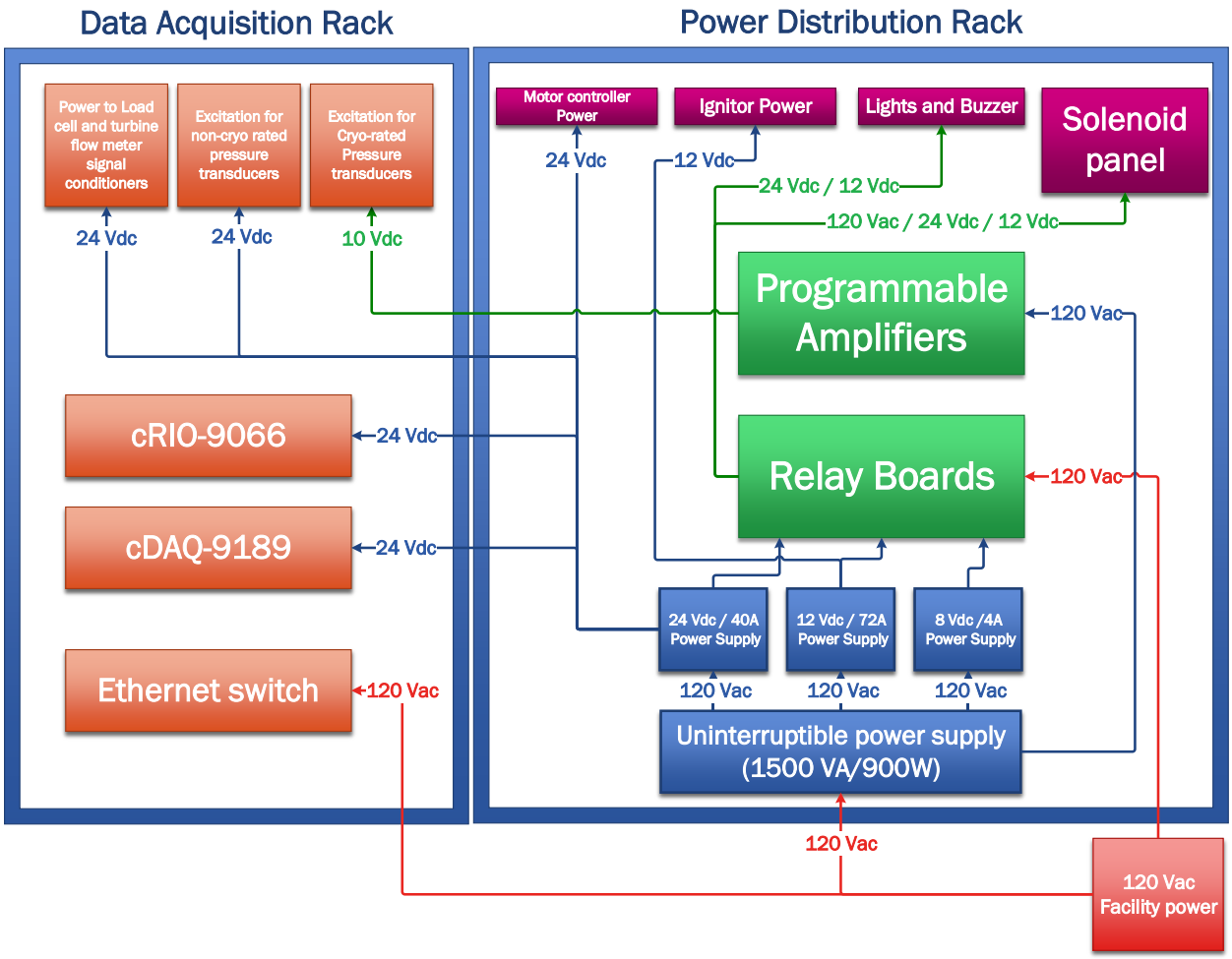


Figure 3.4 High-level diagram that shows how the different components in the MICIT are powered

Each one of these components will have a dedicated section further down.

3.1.1 Power Supplies

The MICIT receives power from a standard 120-volt AC outlet in the tRIAC facility. Some systems receive that AC power directly, while others are powered through an uninterruptable power supply (UPS) that delivers its own 120-volt AC power. The UPS is connected in-line between the MICIT components and the facility power, and its built-in battery allows it to act as a buffer in the event of a sudden outage. The UPS chosen for the MICIT is an ACP SMT1500RM2U That has a maximum output of 1440 volt-amps (VA) or 1000 watts, despite having 1500 VA in its advertising. Neither of these capacities can be exceeded by the system, otherwise the UPS can

abruptly shut off. The rating in VA describes the apparent power the system can deliver, while the wattage refers to the real power that is consumed by the hardware. In this use case, the wattage rating will be the determining factor as to how many systems the UPS can support during a power outage.

The UPS delivers power only to the three power supplies right above it. These power supplies convert the 120 volt AC source to 24 volts, 12 volts, and 8 volts, all in DC. For systems that operate using direct current, their volt-amp and wattage draws are identical. At their maximum output, the combination of these three power supplies can far exceed the maximum wattage the UPS can provide in an emergency. Most of the equipment in the MICIT do not require much power to function, except for any DC solenoid valves that will rely on these power supplies.

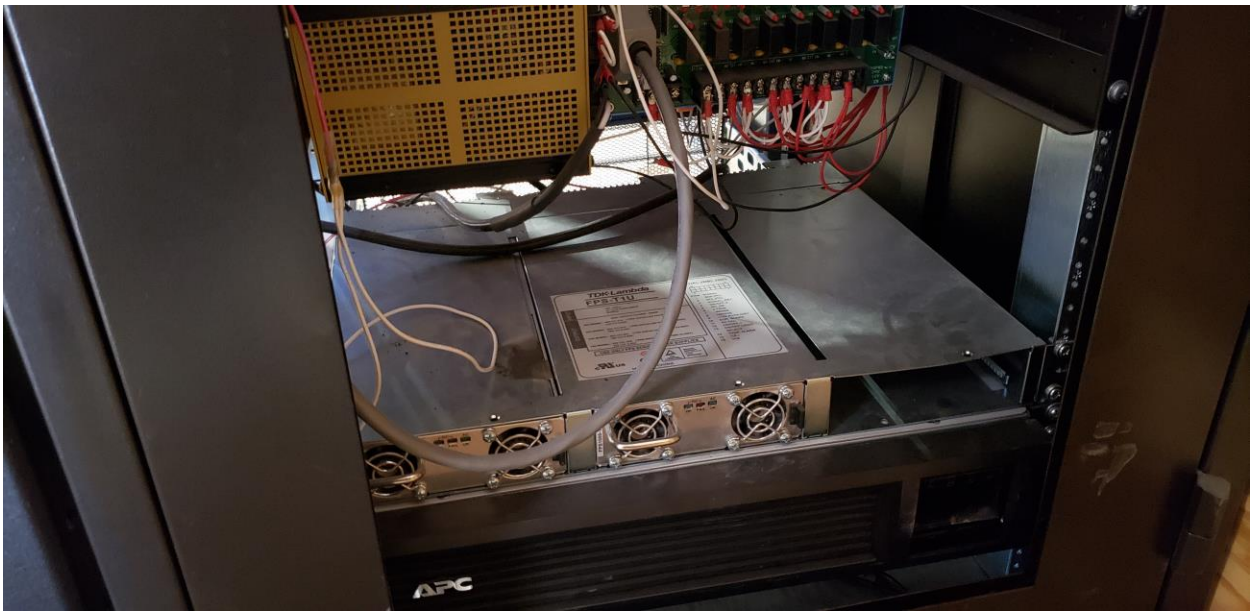


Figure 3.5 Power supplies and UPS at the bottom of the power distribution rack

Table 3.2 Outline of available power to MICIT

Rear of the Power Distribution Rack		
Part	**Capability**	**Purpose**
ACP SMT1500RM2U	1440 VA / 1000 Watts Uninterruptable Power supply	Emergency power to system in event of power outage
Acopian A8MT500	8 Volt DC Power supply (4 Amps max)	(None currently)
TDK-Lambda, FPS100012	12 Volt DC Power supply (72 Amps max)	Solenoid valves, indicator lights
TDK-Lambda, FPS100024/S	24 Volt DC Power supply (40 Amps max)	cRIO, cDAQ, pressure transducers, load cell and turbine flowmeter signal conditioners

Following the power supplies is a collection of terminal blocks that establish a power grid the various systems in the MICIT can tap into. They serve allow for easy distribution of each of the different voltage sources, as the prevent the need to have numerous wires emanate from a power supply.

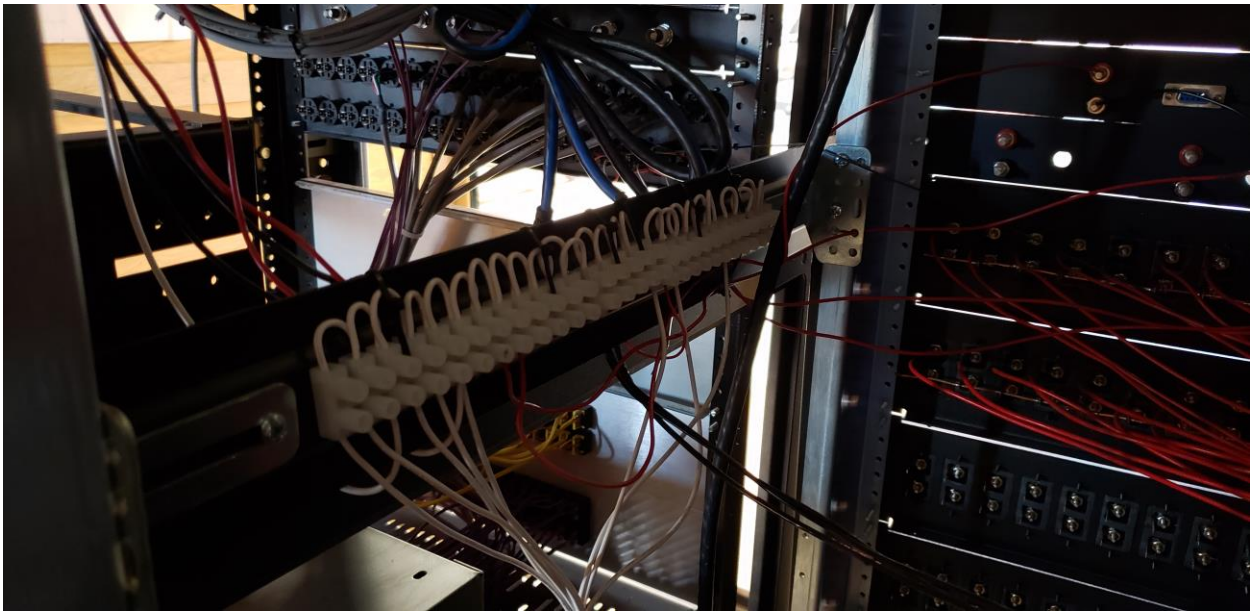


Figure 3.6 Terminal blocks for the hot wires of the power supplies.

3.1.2 Programmable Signal Amplifiers

Some of the instrumentation found in the test stand produce signals in the millivolt range. Such small voltages are difficult to resolve in any data acquisition system. For this, signal conditioners or amplifiers are necessary. Twelve *Omega DP25B-E-A* programmable amplifiers are installed in the rear of the power distribution rack. They were selected to be paired with the cryogenic pressure transducers found in both the RCE and CROME test setups.

An advantage these amplifiers bring is their programmability, allowing them to be used to amplify the signals of any transducer and apply a unique scaling before delivering the output to a measurement device, or in this case, the cDAQ. Upon proper calibration to a given transducers, their built-in displays return the processed values that can easily be seen at a glance. This feature assists with troubleshooting as it removes the need for a DAQ system to first interpret the voltages to a measurement of pressure or pounds-force, depending on the transducer.

Figure 3.7 Omega Signal amplifiers

The amplifiers are numbed starting from top to bottom, *then* left to right.

3.1.3 Indicator Lights and Buzzers

The lights and buzzer on the trailer are used to communicate the status of an experiment to any nearby personnel. The lights themselves are powered using 12 Volts DC at 3.8 amps each, and the buzzer operates at 24 volts and 0.8 amps. They are connected at the very top of the power distribution rack and are appropriately labeled. The relay boards below control which of these components are receiving power at any time.

Figure 3.8 Lights above the MICIT. The Buzzer is on the bottom right

Figure 3.9 Dedicated power outlets for the lights and buzzer near the top of the power rack

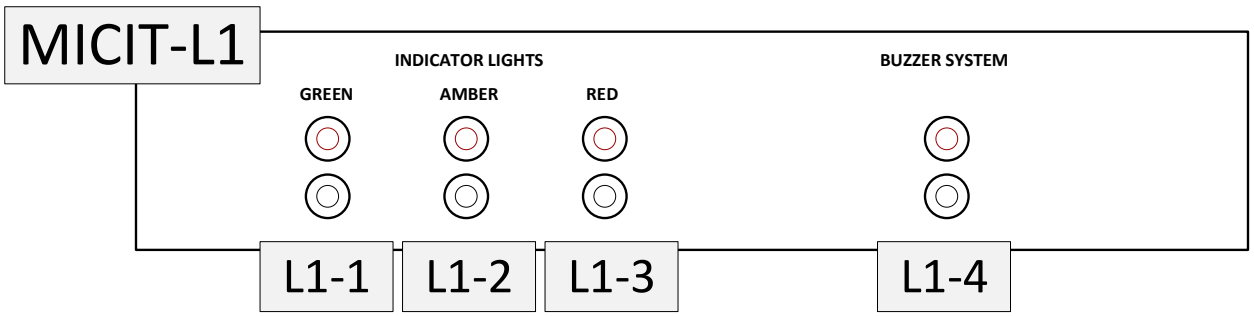


Figure 3.10 Lights and buzzer connector address on the front panel

Table 3.3 Outline of indicator lights and purpose

Item	Part	Power draw	Conveyed Message
Green Light	McMaster-5849T57	12 V / 3.8 A	Personnel are free to move about the facility
Amber Light	McMaster-5849T57	12 V / 3.8 A	Caution: only trained personnel can move about the facility
Red Light	McMaster-5849T57	12 V / 3.8 A	Danger: Engine test imminent. Maintain a safe distance.
Buzzer	McMaster-3038T18	24 V / 0.8 A	Danger: Engine test imminent. Maintain a safe distance.

Further detail on the wiring of the relays can be found in their dedicated section.

3.1.4 Motor Valve Power and Control Signal.

The motors used are stepper motors from Maxon. The motors actuate their respective valves through a titanium thermal isolator and are controlled and monitored with their own proprietary EPOS2 controllers. The controller is powered using 24 Volts DC, which then manipulates the stepper motors in accordance to their calibration. More detail on how their calibration is done can be found on the software portion of this manual.

Figure 3.11 Motor valve and power on front panel

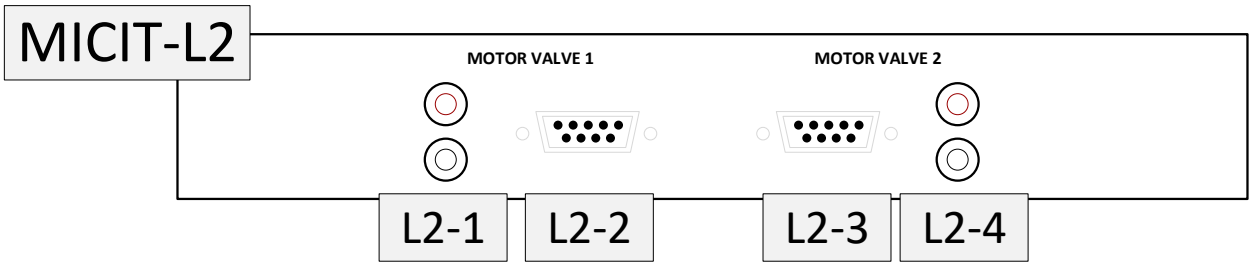


Figure 3.12 Motor valve connector addresses on the front panel

3.1.5 Ignitor Power and Control Signal

The method of ignition for the oxygen/methane mixture in RCE, CROME and CROME-X engines is a coil-on plug sparker. These are the same kind used in the internal combustion engines found in cars and have been slightly modified so they can be implemented into their respective systems. The following is the electrical schematic for the RCE's ignition coil, that was detailed in (Ott, 2018) but the CROME and CROME-X ignitors should still function with the same principal.

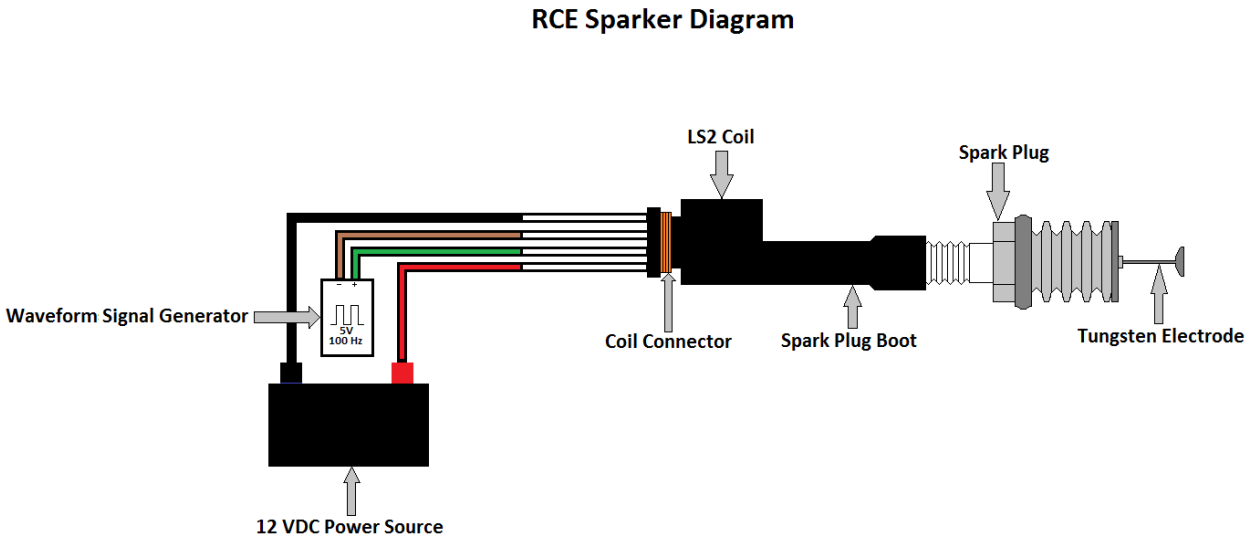


Figure 3.13 Sparker diagram for the RCE

A coil on plug system works by using a primary and secondary coil near one another, each with a different number of turns in their windings. A 0 to 5 volt, 100Hz square wave trigger signal enables or restricts the flow of current to the primary coil. When the signal is high, current flows to the primary coil. When the signal is low, the current is restricted, and the coil quickly discharges to the negative terminal of the power supply. Recall that, due to *Lenz's law*, coils, or solenoids, try resist the change in the current flowing through them with an opposing voltage or electromotive force (emf) also referred to as a back emf. As current begins to flow through the individual windings in a coil, the looping current produce a magnetic field, which in turn produces a magnetic flux through the other turns in a coil, and any coils in close proximity.

When a coil starts discharging, its magnetic field collapses and the back emf spikes in response. As all of this occurs in the primary coil, the secondary coil nearby is also subjected to the same changes in the magnetic field. The current in the secondary coil also observes a very sharp voltage spike, where the peak emf voltage induced in the secondary coil is proportional to the ratio of the number turns between the coils.

This phenomenon causes the tungsten electrode of the spark system to reach voltages that range into the thousands or tens of thousands of volts in relation to a grounded conductor, or the inner walls of the chamber. This is more than enough to break down the air and create the spark.

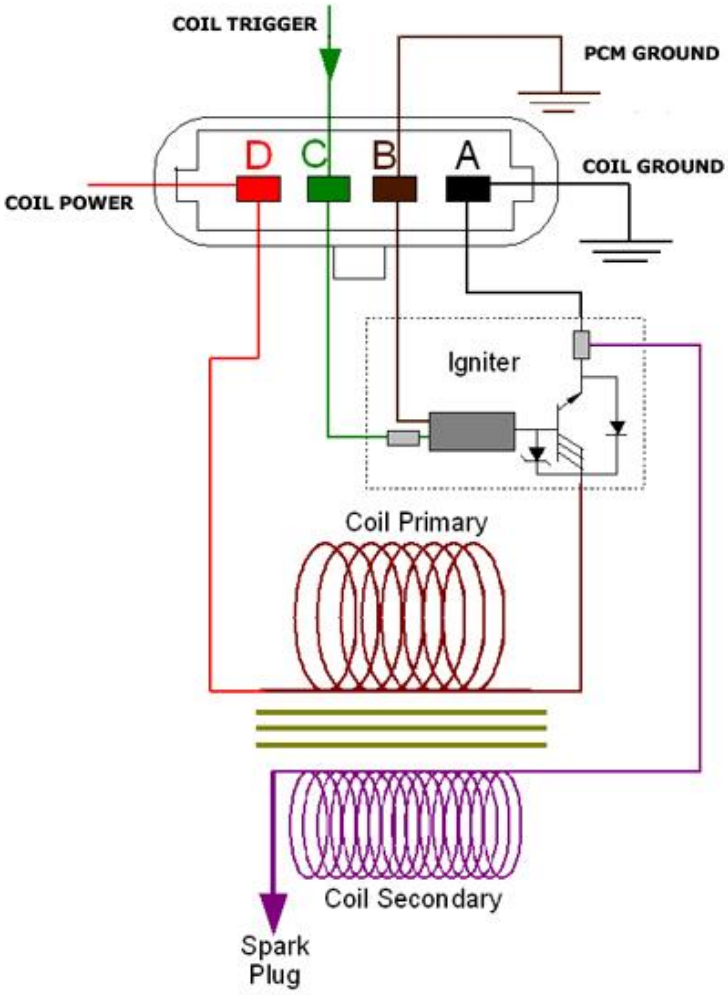


Figure 3.14 Example wiring schematic for a coil-on plug sparker

The front panel on the MICIT has room to power and control up to four separate ignitors.

Figure 3.15 Front panel connections for ignitors

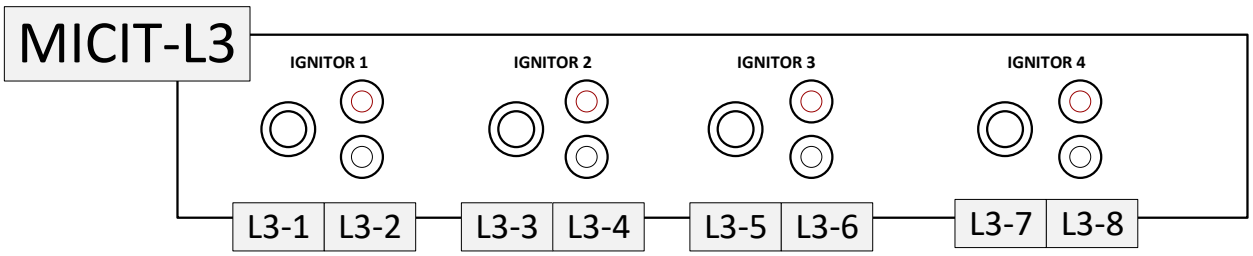


Figure 3.16 Ignitor connectors front panel addresses

3.1.6 Solenoid Valve Power and Control

The solenoid valve interface panels are likely the first thing you will notice when looking at the front of the power distribution rack. It is conspicuous with how much space it takes up.

The valve panels consist of 64 positive-negative terminal pairs of female banana plugs in total that are used to deliver the power to the solenoid valves. They are split into four rows of 16 pairs with each row labeled L4,L5,L6, and L7 from top to bottom and each connection is numbered in ascending order from left to right as shown in Figure 3.19 below. This way, each connector can be referred to with two numbers for row and its position in the row.

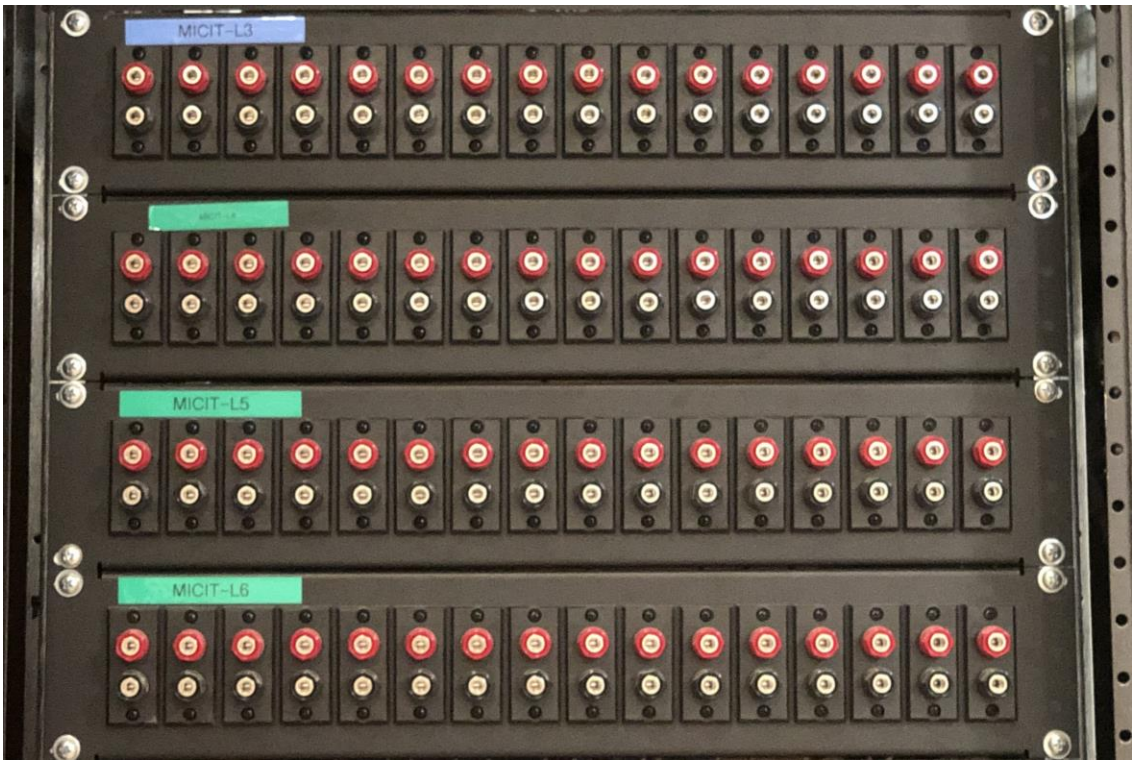


Figure 3.17 Valve interface panels on the power distribution rack. Panels have been updated to start from L4 instead of L3 that the picture shows.

By default, none of these connections are delivering any power as their circuits are interrupted with a *nominally open* relay. A nominally open relay switch will only close and complete the circuit to deliver power when they receive an electrical signal to do so.

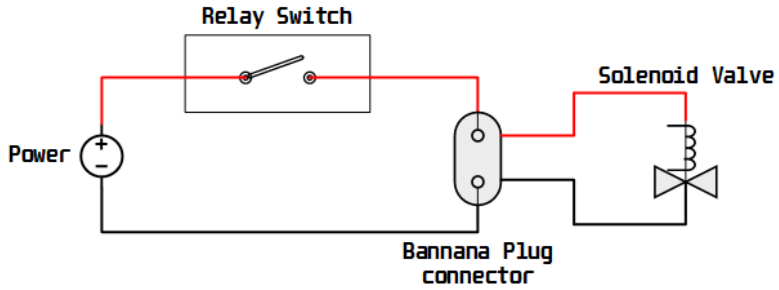


Figure 3.18 Example of a circuit with a single relay

Lastly, even though these panels were made with solenoid valves in mind, they can just as easily be used to power anything else that needs the on/off control that the relays provide. For

instrumentation that requires a constant stream of power, such as passive instrumentation like pressure transducers, they are directly connected to their source of power.

Figure 3.19 Solenoid front panel connection addresses and their current voltage output by row (outputs can be adjusted)

3.1.7 Relays

It would be problematic if the solenoid valves immediately received power the moment they are plugged into the front panel of the power distribution rack. To answer this, the MICIT needs a way to control which valves receive power and open the pathways in the piping. This is where the relay boards come in.

To turn on the lights to illuminate a dark room, the mechanical action of flipping the light switch by the entrance completes a circuit to deliver current the bulbs. A relay is simply a different type of switch that is controlled with an electronic mechanism dependent on a voltage potential. Relays come in different forms and the different teams in the cSETR employ different kinds of relays. Relays can be either electromechanical or solid state. Solid state relays can be designed for

use with alternating current or direct current, whereas electromechanical relays can function with either. Always carefully check if a particular relay is suitable for its intended application.

On the back of the power distribution rack are three relay boards found towards the bottom and below the programmable amplifiers. They will be referred to simply as “Relay 1” starting at the bottom, “Relay 2” in the middle, and “Relay 3” at the top.

The electronic signals for these relays originate from the cRIO installed in the separate rack. How the software manages to communicate with the relays will be described in its own section.

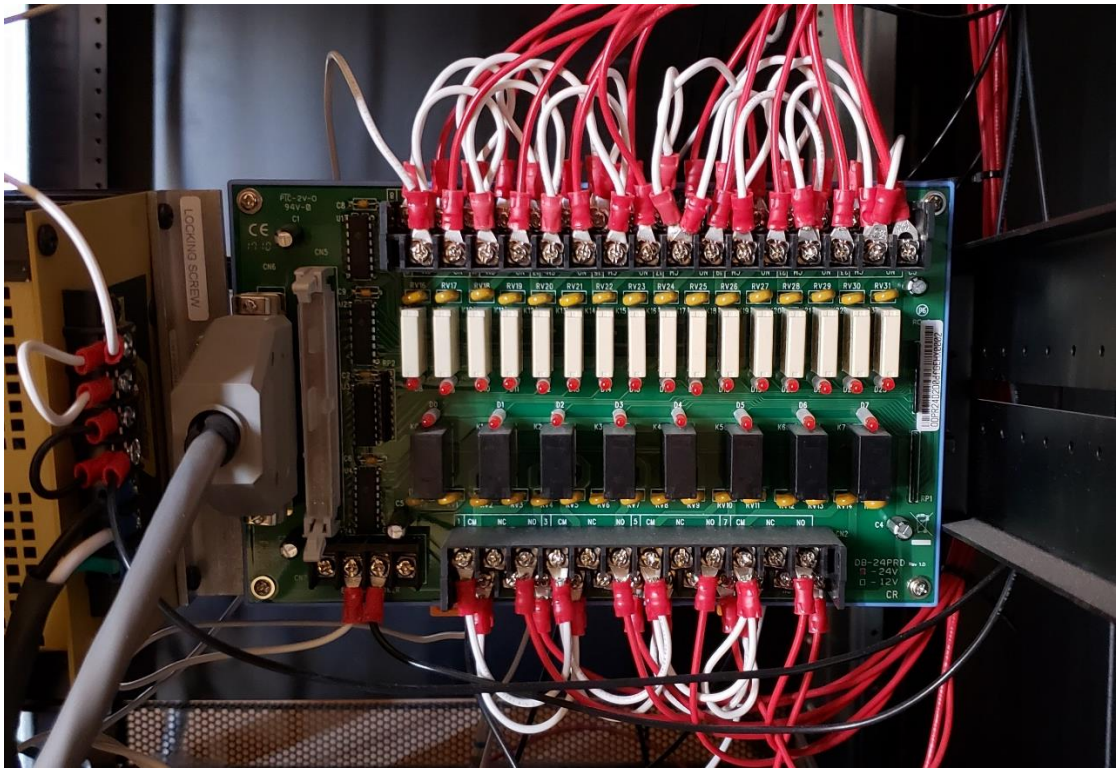


Figure 3.20 Picture of the Omega relay board, with a 8 volt DC power supply to the left

The relay boards installed are the Omega OME-DB-24PR/DB-24PRD. These boards have 24 *electromechanical* relay channels each for a total of 72 across the three. An individual relay can handle up to 5 amps of current before overheating and failing. The relay boards themselves are powered from the 24-volt power supply near the very bottom of the rack and just on top the

uninterruptable power supply. On the far left of a relay is a 37-pin connector that connects to a NI-9403 Module installed in the cRIO, although only 25 of those 37 pins will actually be utilized. These NI-9403 modules, through the FPGA software, send a 5-volt Transducer to Transducer logic (TTL) signal through a pin. This signal then instructs a board to use the 24 volts it is supplied with to power a solenoid and close a switch and to complete the specified circuit, thus delivering power to either the solenoid valves, motor, lights, or buzzer. A red indicator light turns on right beside the matching relay to indicate that the signal got through and a circuit is complete. There should also be a distinct 'click' sound as the internal electromagnet moves the contact in place.

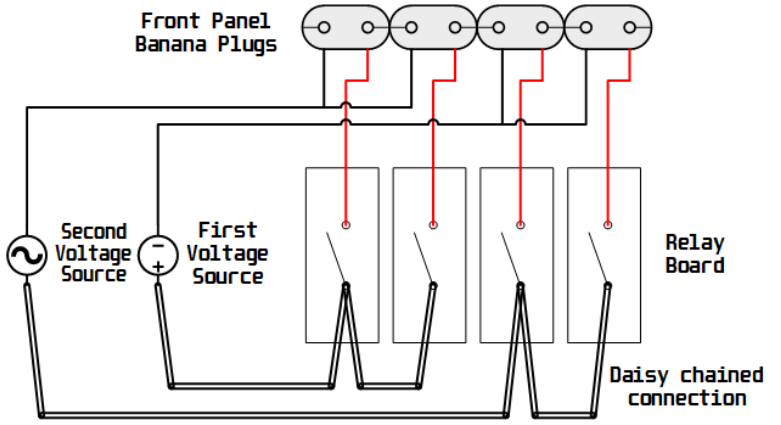


Figure 3.21 Example of parallel circuit arrangement with a 4-channel relay board

Not every instrument in the front panel requires the same power source. For instance, the solenoid control valves at the facility are powered with 120 Volts AC, 24 Volts DC, or 12 Volts DC. The motorized valves and the ignitor both require 24 Volts and 12 Volts DC respectively. Figure 3.2 shows a simplified example of how the relay board is wired to utilize different voltage sources.

Before the positive terminal is interrupted, all the channels utilizing the same voltage source are connected in parallel to the relay board. To make this parallel connection, all the individual relay switches are jumped together as shown in Figure 3.21. The red cables connect to

the red banana plug terminals on the front panel, and the white cables are the positive end of the voltage source.

The full electrical schematic can be found in the same folder as this document in both a PDF and CAD format.

A detailed explanation as to how the relays are connected to compactRIO in the neighboring rack will be elaborated in the next section.

3.2 DATA ACQUISITION RACK

The next server rack is where is where the data acquisition and control systems are housed. On the control side of things is a cRIO (cRIO-9066) equipped with a field programmable gate array (FPGA) and a 40 MHz system clock speed. On the data acquisition end there is a compact DAQ (cDAQ-9189).

They are both powered by the 24 Volt DC power supply and communicate to one another via an ethernet switch installed in the middle of the rack. A personal laptop can connect into this ethernet switch to interface with both systems. Alternatively a long fiber optic cable is used to connect the MICIT ethernet switch to a another separate ethernet switch right beside the tRIAC's Control Computer.

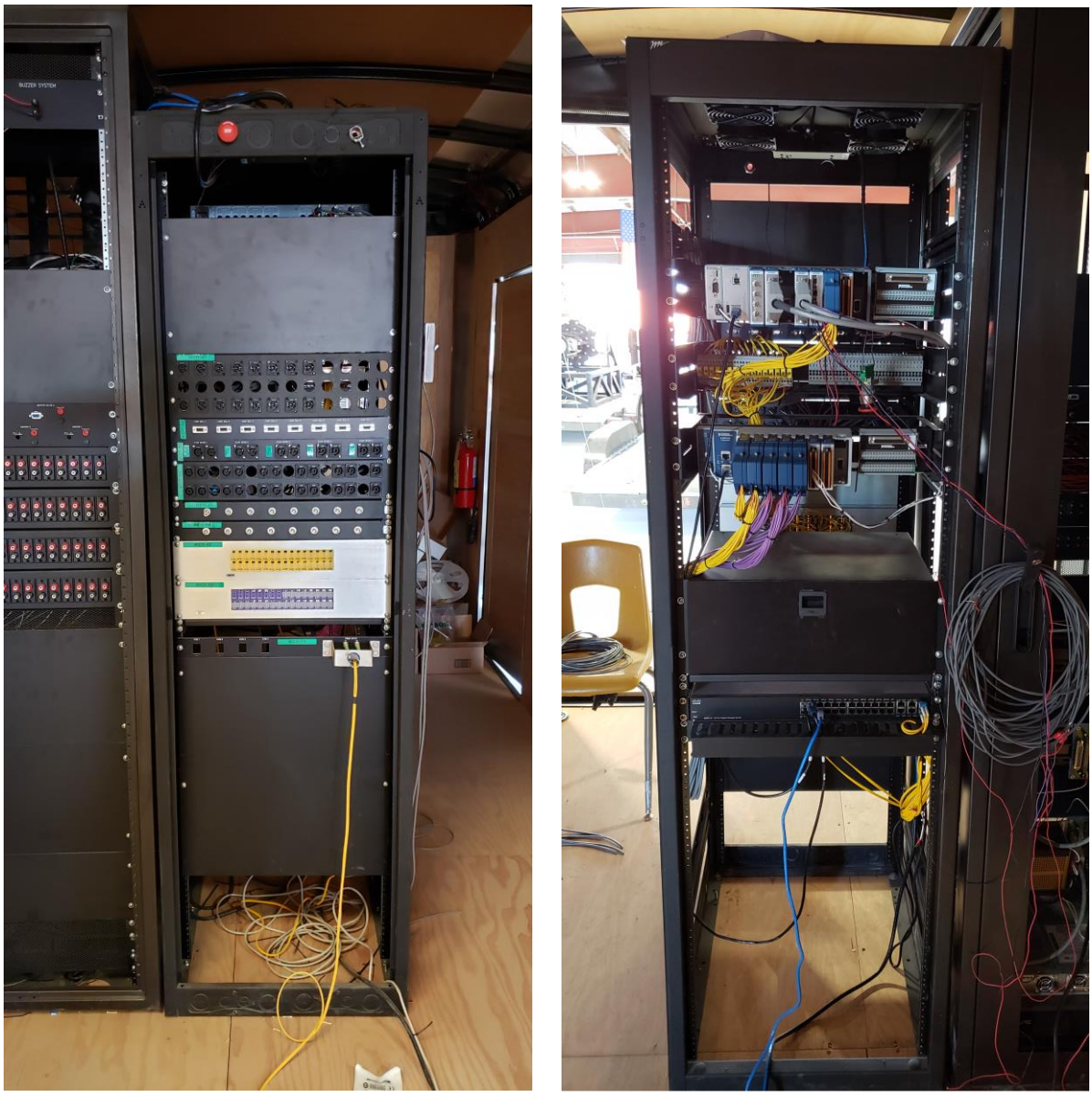


Figure 3.22 Front of the data acquisition server rack (Left) and the rear (right)

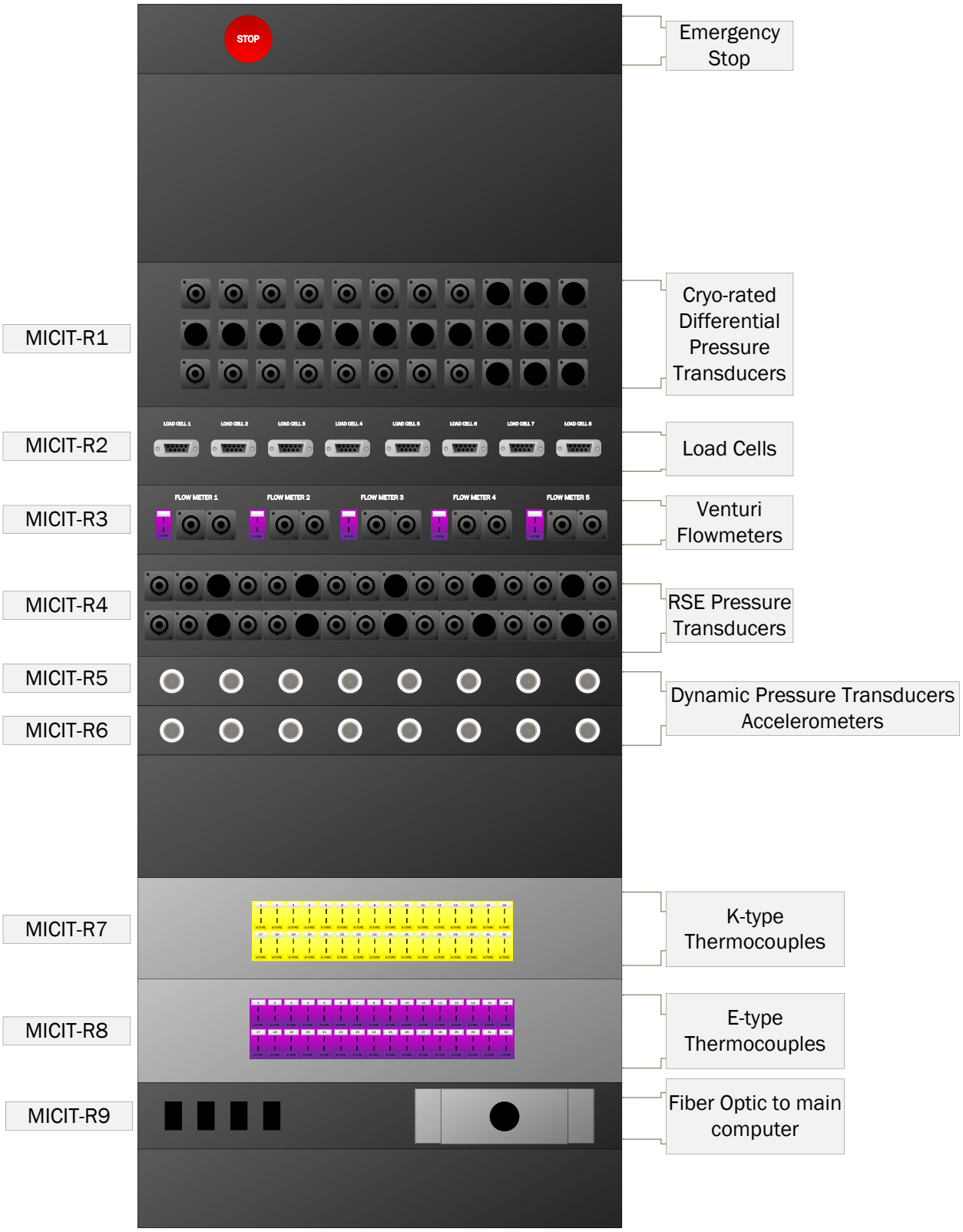


Figure 3.23 Outline of the data acquisition rack's front panel

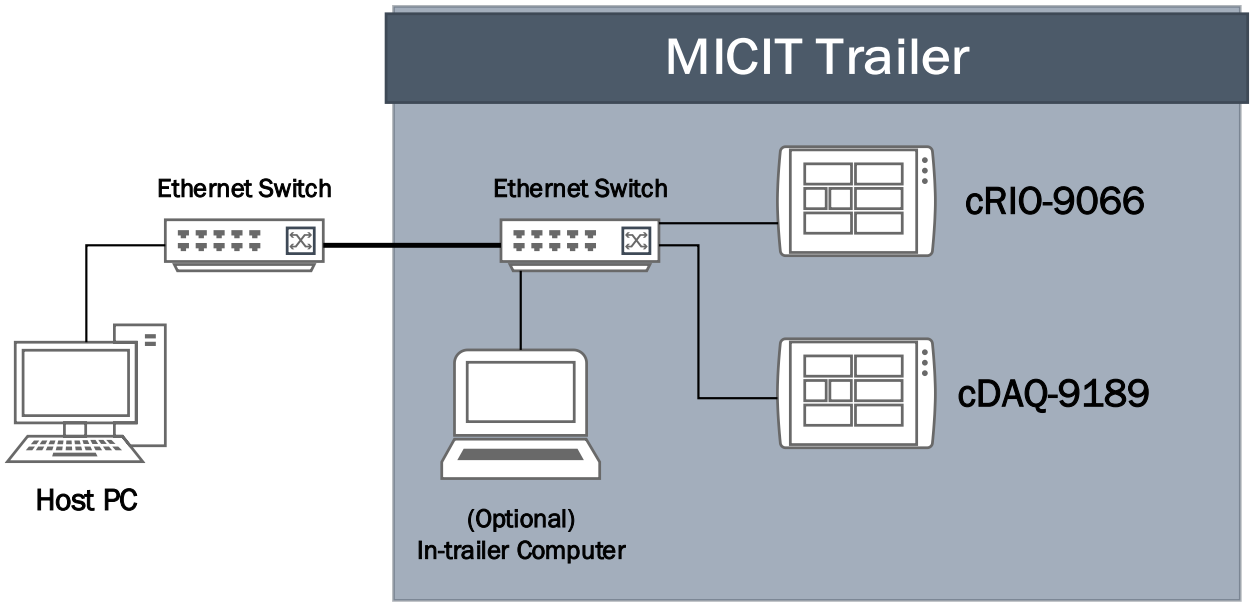


Figure 3.24 Connectivity between systems in the MICIT

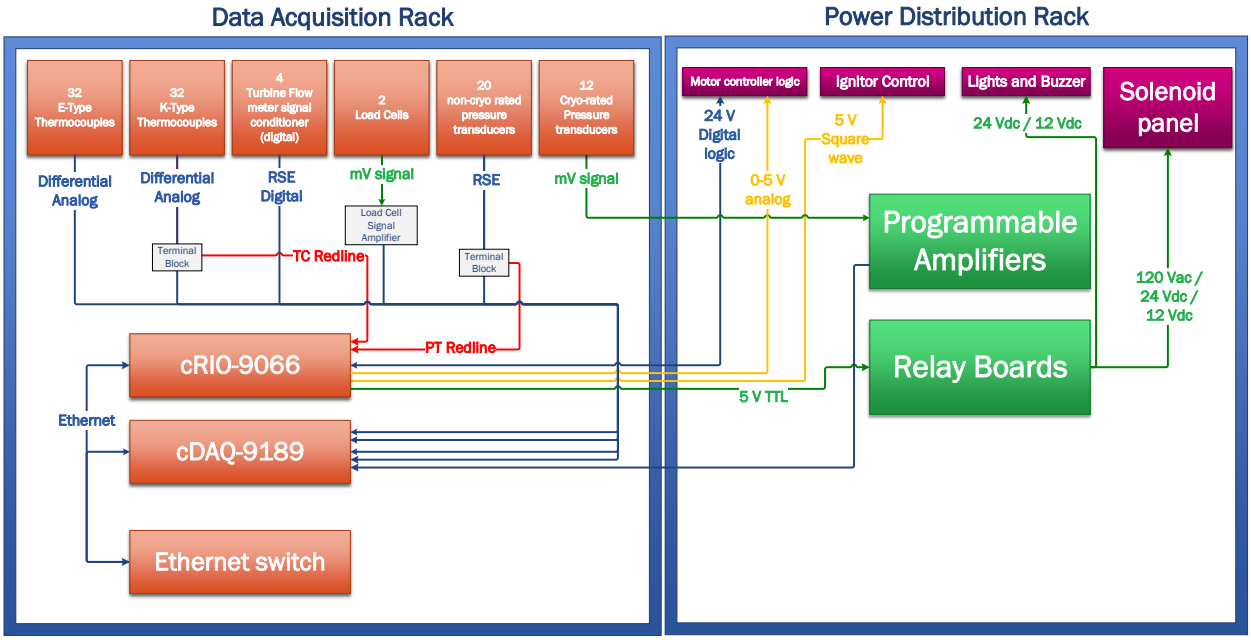


Figure 3.25 High-level diagram that shows how the data flows and hardware interact with each other

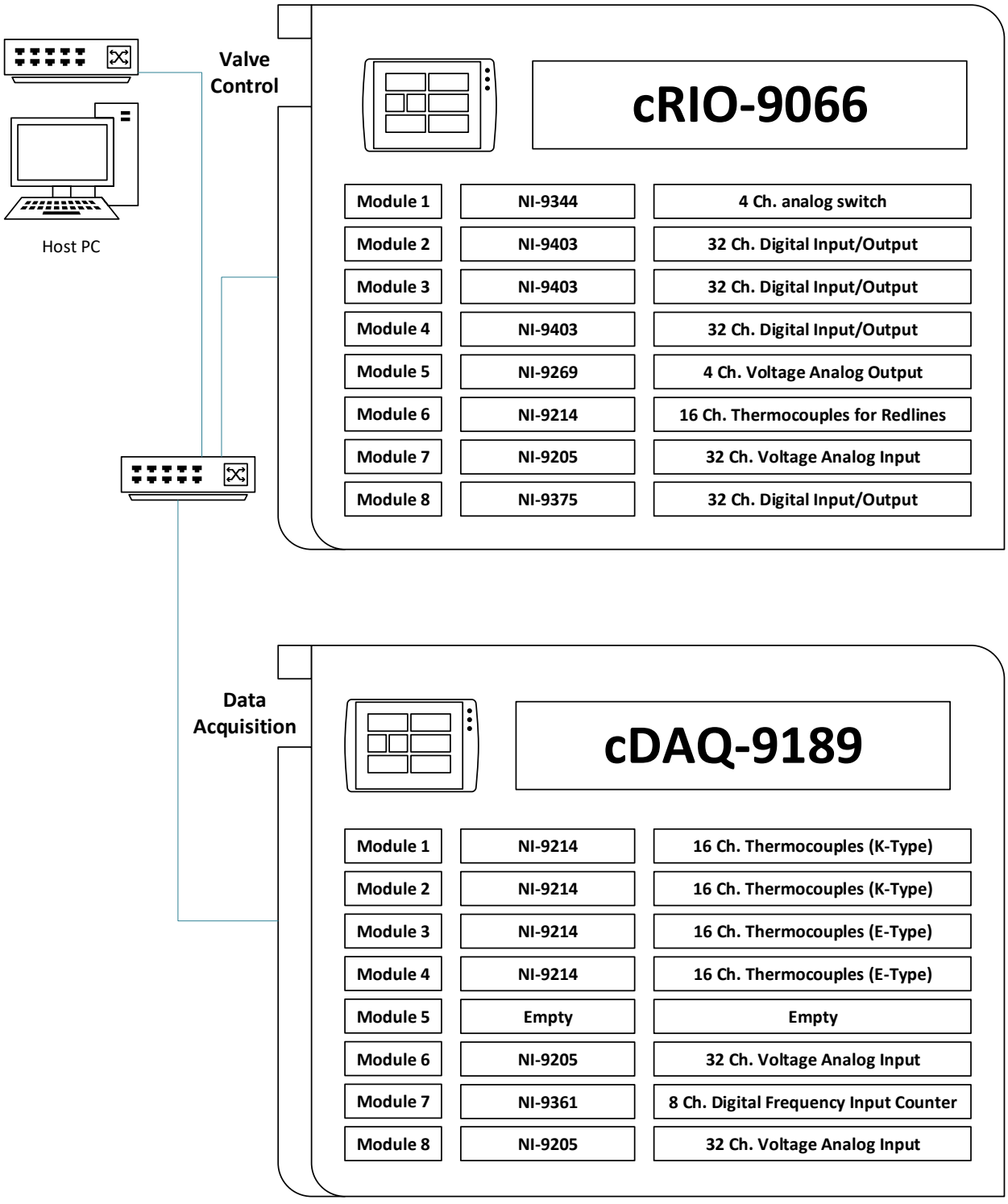


Figure 3.26 Layout of the cRIO and cDAQ Chassis in the rack

3.2.1 The emergency button

The top of the data acquisition server rack houses a wired hardware emergency stop button. It is wired into one of the NI-9403 modules in the cRIO-9066 chassis to signal to the system to initiate an emergency shutdown manually if the software fails to trigger an emergency response or if there is an unforeseen circumstance that was not programmed for.

Figure 3.27 Emergency stop button on the top of the data acquisition rack

3.2.2 The cryogenic pressure transducers

Following the emergency stop button is the connections for the cryo-rated static pressure transducers. These transducers output a double ended signal while receiving 10 volts of excitation from the programmable amplifiers. The outputs are wired back to the amplifiers, in which they are then wired to the cDAQ NI-9025 voltage input modules for acquisition.

Figure 3.28 Cryogenic pressure transducer inputs

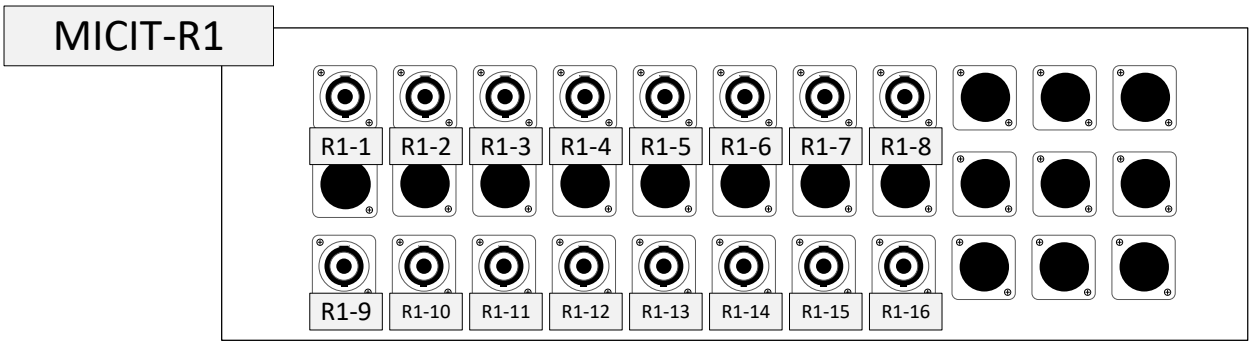


Figure 3.30 Cryogenic pressure transducers connectors front panel addresses

3.2.3 Load cells

The load cells, like the cryogenic pressure transducers, need to have their signals amplified. PCB provided such an amplifier that operates using 24 volts DC. Upon amplification, the outputs of the load cells can be wired in a referenced single ended (RCE) input on a NI-9205 module.

Figure 3.29 Load cell inputs

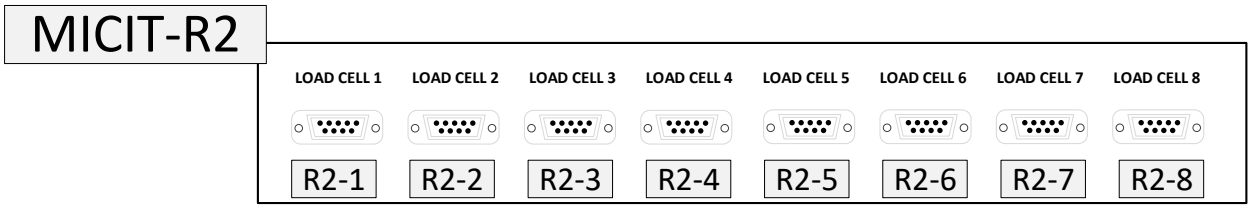


Figure 3.31 Load cell connectors front panel addresses

3.2.4 Venturi Flow meter

The venturi flowmeter does not directly measure the flowrate of a fluid by itself, but rather a combination of a thermocouple, differential pressure transducers, and geometry of the venturi itself provide the values to determine the volumetric flow rate of the fluid. These quantities are processed in software and multiplied by a density value provided by RefPROP. *density value would only be relevant in the case of a single phase flow. Temperature and pressure by themselves are not enough to give the condition of a two-phase flow.*

Figure 3.32 Venturi flow meter panel. Includes connectors for E-type thermocouples and static pressure transducers

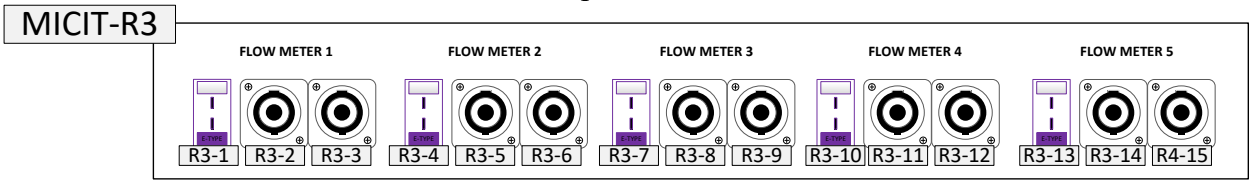


Figure 3.33 Venturi Flow meter connectors front panel addresses

3.2.5 Static Pressure Transducers

Next are the connections for the static pressure transducers. They operate with a 24 volt excitation, and output a 0-5 volt or 0-10 volt signal that is either sent directly to the cDAQ's NI-9205 analog voltage input module, or are split at the terminal blocks so that both the cDAQ and cRIO can simultaneously receive the pressure transducer readings. The holes in this panel are a result of the connectors being too thick and the inputs too close together for three transducers to be plugged in side by side.

Figure 3.34 Static pressure transducer input

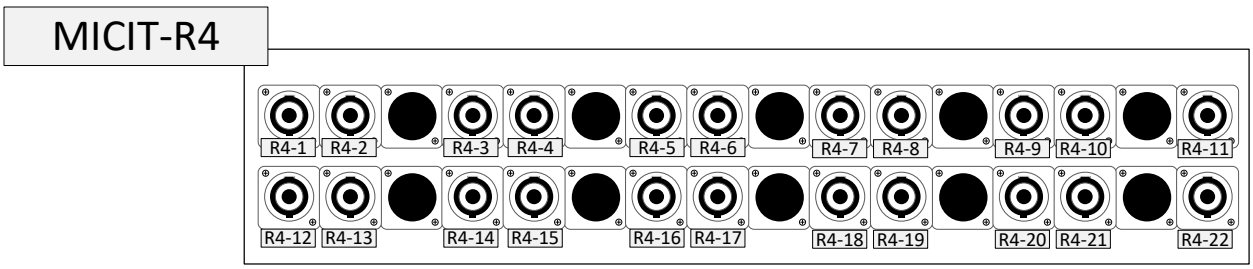


Figure 3.35 Static pressure transducer connectors front panel addresses

3.2.6 accelerometers and dynamic pressure transducers

Following the inputs of the static pressure transducers are the accelerometers and dynamic pressure transducers that have a much faster response time.

Figure 3.36 Dynamic pressure transducers (Top) and Accelerometers input (bottom)

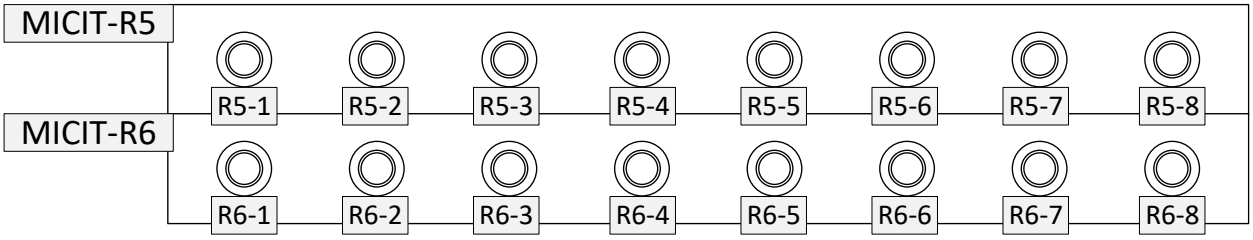


Figure 3.37 Dynamic pressure and accelerometer connectors front panel addresses

3.2.7 Thermocouples

Lastly are the thermocouple inputs for both the K-type and E-type thermocouples. K-types are traditionally yellow, while the E-types are traditionally purple so that they can be distinguished from one another. All of these are wired to the NI-9214 thermocouple modules on the cDAQ chassis, with the exception of the first 16 K-type connections, which are split with a terminal block so both the cDAQ and cRIO can observe their measurements.

Figure 3.38 K-type and E-type thermocouple inputs

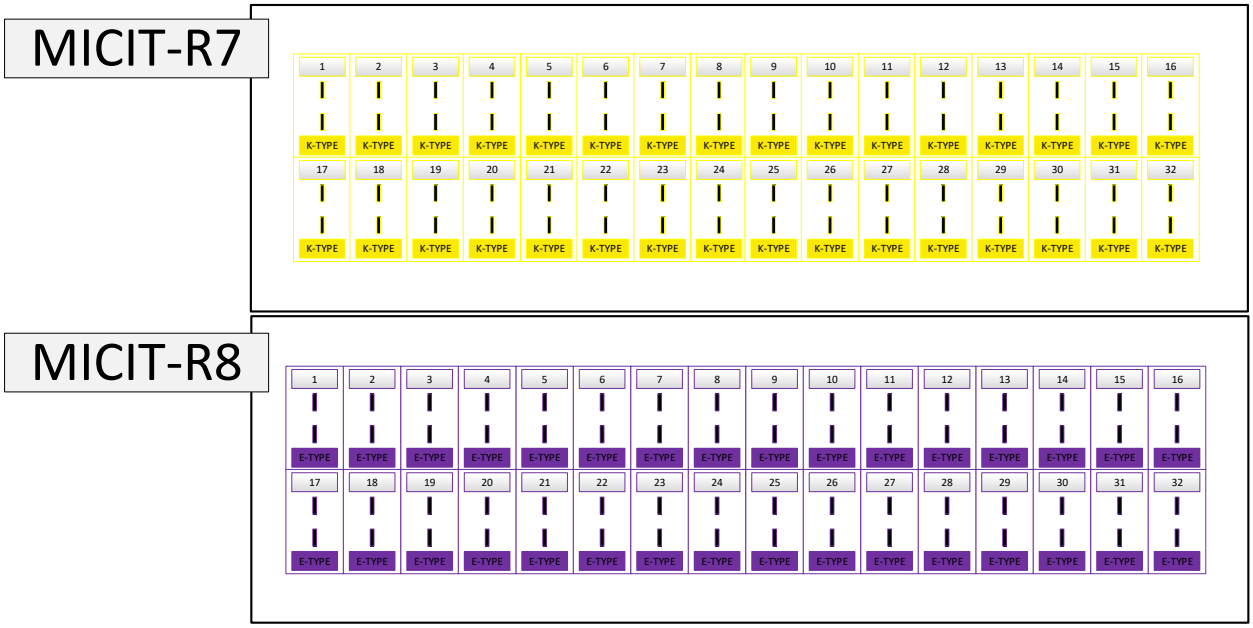


Figure 3.39 Thermocouple connectors front panel addresses

3.2.8 The cRIO cDAQ

Figure 3.40 Wired Compact RIO chassis

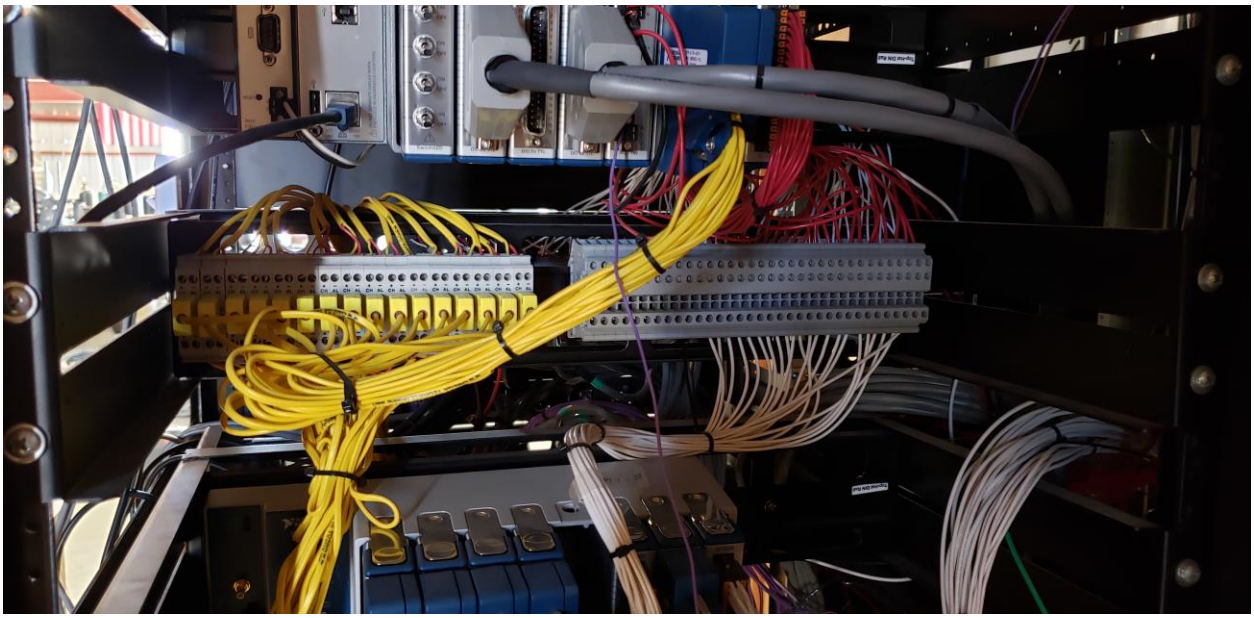


Figure 3.41 Terminal blocks splitting K-type thermocouples and static pressure transducer signals

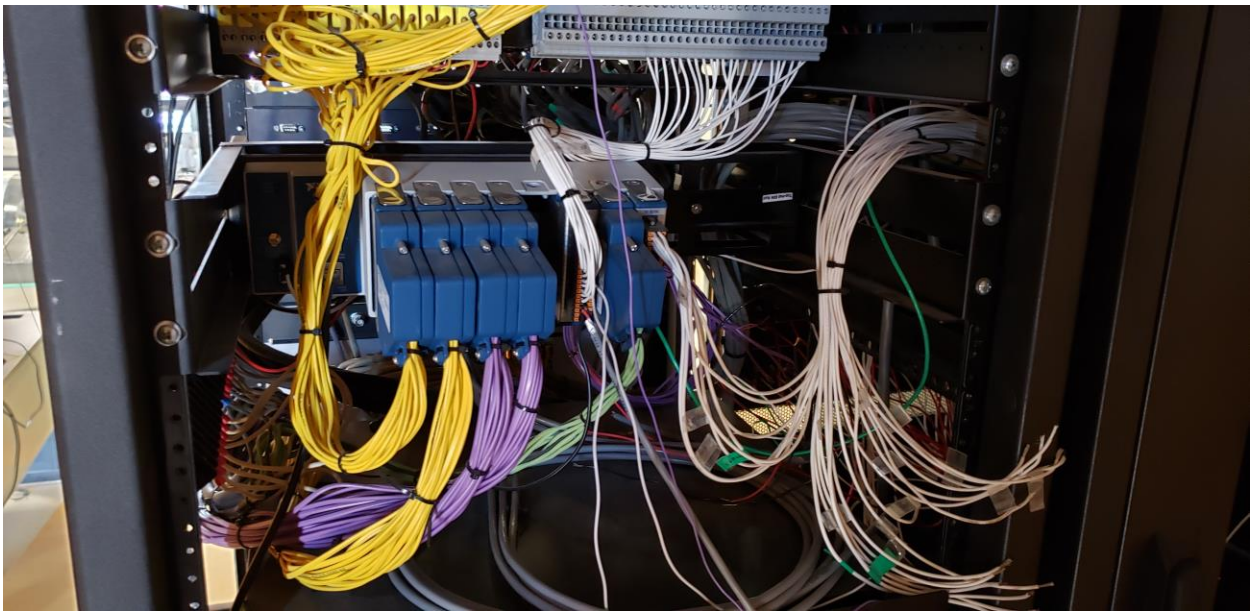


Figure 3.42 Wired Compact DAQ chassis

Figure 3.43 Ethernet switch connecting NI chassis with a host computer

Chapter 4: MICIT Software

4.1 Installing LabVIEW and Other Necessary Drivers and Add-ons

The MICIT system is updated to the latest version of LabVIEW according to the date of this thesis. It is LabVIEW 2018 **version 18.0f2**. With LabVIEW, you also need the latest version of NI-MAX along with latest cDAQ and cRIO drivers. Lastly, there are a few necessary add ons:

1. Real-Time Module
    a. Allows for the creation and deployment of real time applications for monitoring and control
2. FPGA Target
    a. Allows for programming and interfacing with an FPGA system
3. Xilinx Compilation tool for Vivado
    a. This allows the compilation of a bitfile that allows for modification of the FPGA

4.2 LabVIEW overview and block diagram

Once the latest version of LabVIEW 2018 installed with the appropriate drivers and add-ons, and have the simulation up and running if, the MICIT software can be worked on. This section will first go over the process that went behind building the code, then delve into the specifics and features that were added in such as the FPGA and any other sub-VIs.

To fulfill the software design requirements of the cSETR's experimental campaign and to make the most of the breadth of hardware we have installed in the MICIT trailer, it needs a robust and elaborate LabVIEW code. Fortunately, when dealing with comprehensive programs in any application it is often not necessary to start from a blank slate. It is common practice to build off a solid template and then customize it from there.

At the time of this writing, the LabVIEW code found in the MICIT trailer, High Heat Flux (HHF) experiment, High Pressure Combustion (HPC), and CubeSat Teams are all built using the

exact same template. They are of course, customized for their specific needs, but once you know and understand how they are fundamentally set up you will be navigating their web of wire diagrams and sub-VIs like a pro and your team will think you are some kind of magician. It is pretty cool. The features discussed in this thesis will be for the MICIT in particular, but it should still help with any of the research teams I have listed and for entirely new experimental set ups.

To access the template used, open LabVIEW and click on "Create Project". From there, navigate to "Sample Projects" and select "Continuous Measurement and Logging (NI-DAQmx)" before finally clicking "Next". It will prompt for a project name and icon. Name it in a way that suits your set up and customize the icon if you are feeling artistic. You can always play around with the icon design later if you want to adjust it. Once that is done, hit "Finish" and LabVIEW will take a moment to get the template set up. You will then have an entire project file ready to adjust.

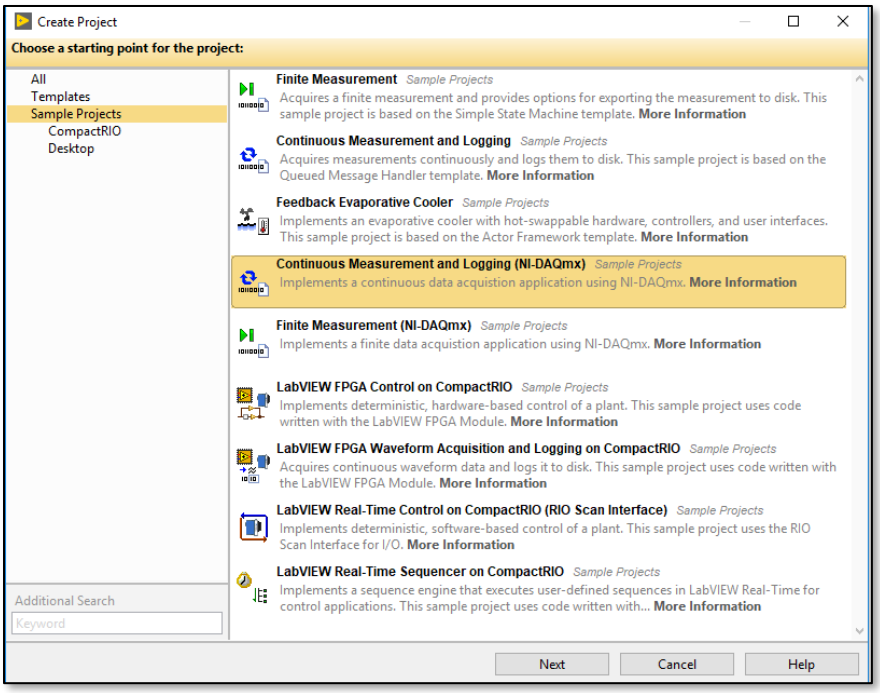


Figure 4.1 "Create project" dialogue box

There will be a large assortment of VIs in the project, but there are only a handful you will need to adjust. The suggested order in which to go through these will be illustrated in the following flow chart. If you are going through the MICIT Project file, I left behind detailed comments on the

code as well. Make it a habit to comment your code so it is as clear as possible! Not only will others thank you immensely, it will also help you in case you have to adjust a sub-VI you may not have touched for an extended period of time.

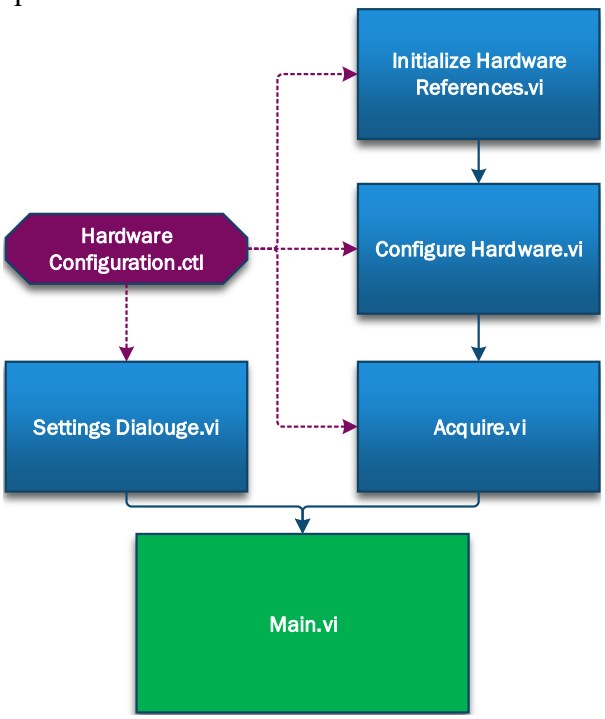


Figure 4.2 Relevant LabVIEW .vi files to adjust in project

4.3 NAVIGATING THE LABVIEW CODE

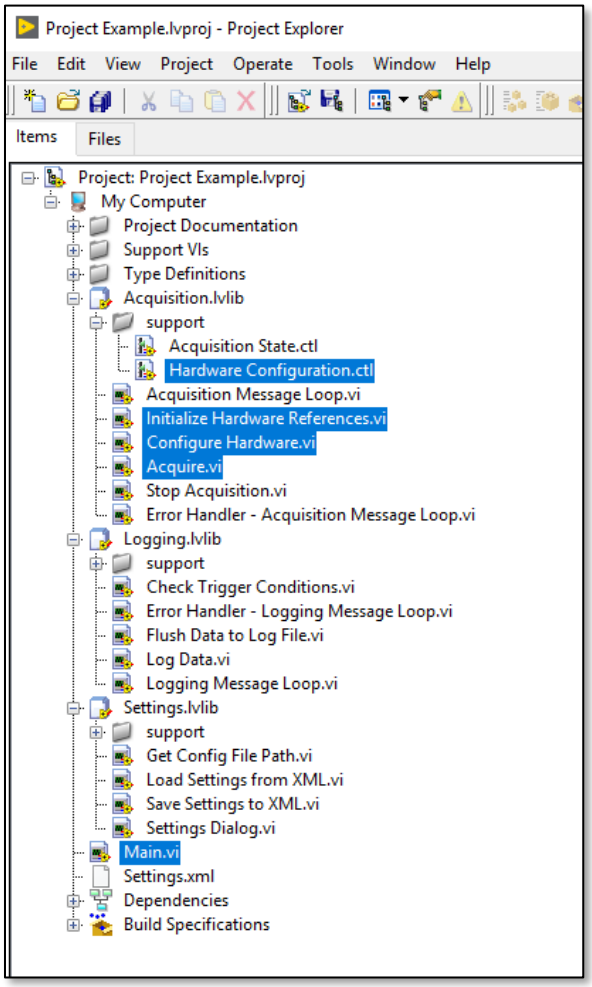


Figure 4.3 View of the project template with relevant .vi files highlighted

4.3.1 Hardware Configuration.ctl Type-Definition

The first thing to adjust is not actually a VI file, but rather a type-definition. This type definition is a cluster that will be shared across the key VIs to adjust. The file name of this type-definition is “Hardware Configuration.ctl” and it can be found under the “Acquisition>Support” folder.

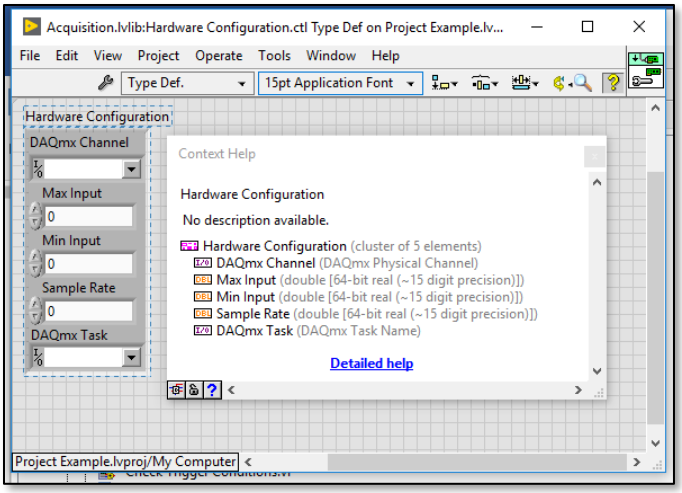


Figure 4.4 Template "hardware configuration.ctl"

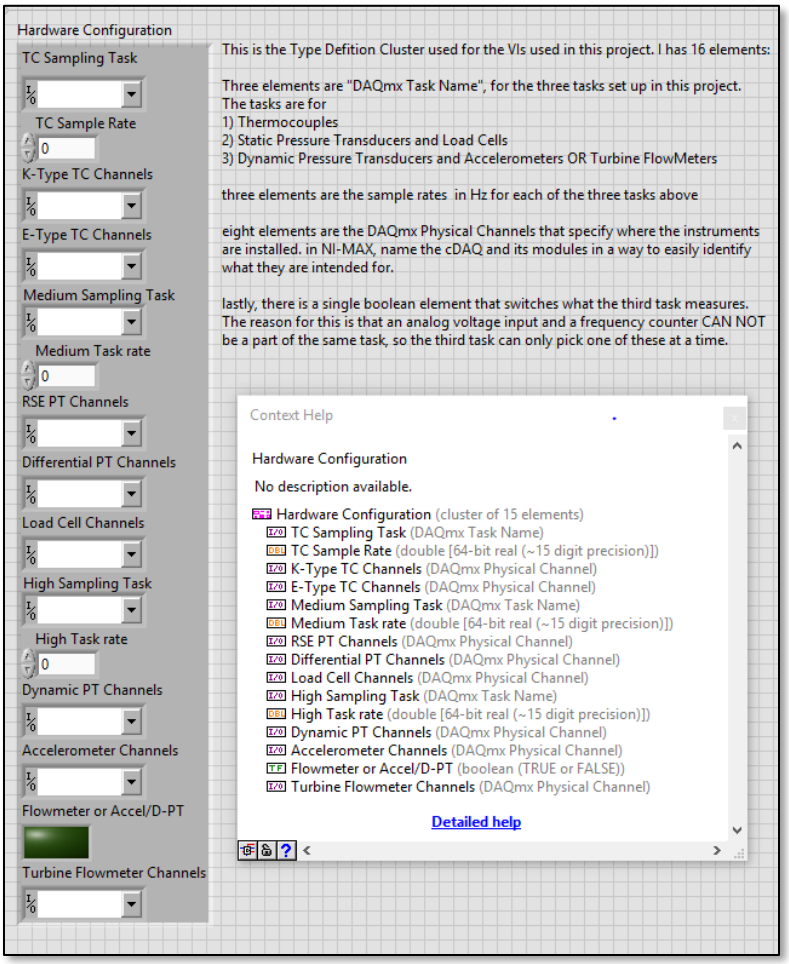


Figure 4.5 MICIT "Hardware configuration.ctl"

A detailed comment is provided in the Type definition itself that will be reiterated here. For a quick refresher on what a type-definition is, it is a control file for a variable or a cluster of variables. It can be easily identified in a wire diagram by a black triangle on the top left corner of said variable. They are suitable in situations where a set of variables or clusters are used several times in a file or project. That way, when you adjust the type-definition file, everything else that uses it will also automatically adjust to reflect the changes. Otherwise you would have to go through each individual file it is referred to and make the changes.

An important thing to keep in mind is that, a type definition does not keep track of specific values, only the variable types. The file the type definition is used in is what determines the value of those variables.

With all of this in mind, what is the logic behind the set up of the Type Definition file in our case? The pictures above can give a clue. At first, there needs to be variables for the DAQmx Task Names. The MICIT Project will have three tasks, so there will be three variables for them called:

- TC Sampling Task (I/O)
- Medium Sampling Task (I/O)
- High Sampling Task (I/O)

They are named as such to take advantage of the different internal clocks present in the cDAQ-9189. Not all of the installed instrumentation have the same response rate, and so it is wasteful to sample slow-responding transducers at a high frequency. Three additional double point precision variables are added in the cluster to specify the sample rate for each of the three tasks

- TC Sampling Task (I/O)
    - TC Sample Rate (DBL)
- Medium Sampling Task (I/O)
    - Medium Task Rate (DBL)
- High Sampling Task (I/O)
    - High Task Rate (DBL)

Next is a set of variables that point LabVIEW to where the transducers are physically connected on the cDAQ. There will be a “DAQmx Physical channel” for each kind of measurement device. In the MICIT there are eight:

- TC Sampling Task (I/O)
    - TC Sample Rate (DBL)
        - K-Type Thermocouple Channels (I/O)
        - E-Type Thermocouple Channels (I/O)
- Medium Sampling Task (I/O)
    - Medium Task Rate (DBL)
        - RSE Pressure Transducer Channels (I/O)
        - Differential Pressure Transducer Channels (I/O)
        - Load Cell Channels (I/O)
- High Sampling Task (I/O)
    - High Task Rate (DBL)
        - Dynamic Pressure Transducer Channels (I/O)
        - Accelerometer Channels (I/O)
        - Turbine Flowmeter Channels (I/O)

Finally, a boolean was added for the purpose to select between the measurement devices used in the “High Sampling Task”. This is needed because the Dynamic Pressure Transducers and Accelerometers are analog input signals, whereas the Turbine Flowmeters are digital signals. A single task cannot measure both, so our third and final task switches between analog or digital based on the boolean.

- TC Sampling Task (I/O)
    - TC Sample Rate (DBL)
        - K-Type Thermocouple Channels (I/O)
        - E-Type Thermocouple Channels (I/O)

- Medium Sampling Task (I/O)
    - Medium Task Rate (DBL)
        - RSE Pressure Transducer Channels (I/O)
        - Differential Pressure Transducer Channels (I/O)
        - Load Cell Channels (I/O)
- High Sampling Task (I/O)
    - High Task Rate (DBL)
        - Dynamic Pressure Transducer Channels (I/O)
        - Accelerometer Channels (I/O)
        - Turbine Flowmeter Channels (I/O)
        - Flowmeter or Accel/D-PT? (Boolean)

4.3.2 Initialize Hardware References.vi setup

With the “Hardware Configuration.ctl” file set up, next is the “Initialize Hardware References.VI” Under “Acquisition” in the project file. This will be a very small VI that simply imitates the task(s) used in the project. As mentioned prior, the MICIT system will need three separate tasks to establish three different sampling rates. To help with troubleshooting, each of the tasks are also given names.

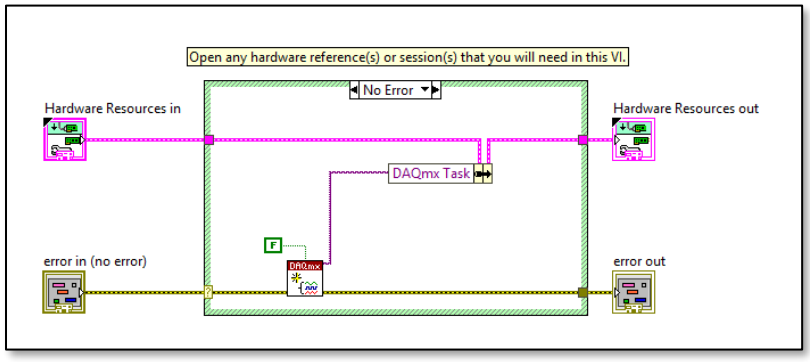


Figure 4.6 Template "Initialize Hardware References.vi"

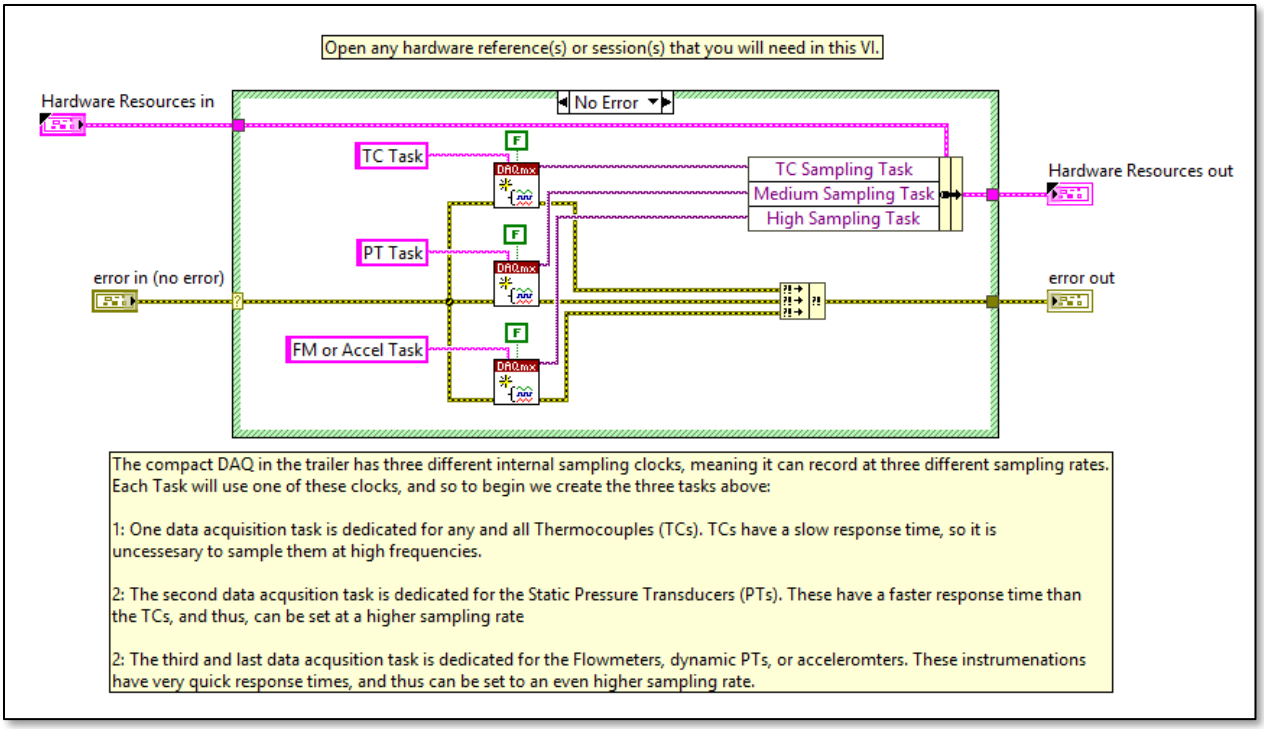


Figure 4.7 MICIT "Initialize Hardware References.vi"

4.3.3 Configure Hardware.vi setup

After “Initialize Hardware References.VI”, is “Configure Hardware.VI” under “Acquisition”, or the same folder. Here is where things begin to get involved. The wire diagram used for the MICIT is quite large, so a flowchart and several screen captures will be used to break it down. The “Configure Hardware.VI” in the MICIT project file will have detailed comments to describe the setup to help follow along.

The purpose of the “Configure Hardware.VI” is to then give each of the tasks that were just created what signals they will be measuring and how those signals are processed. The blank template uses a single analog input as an example, but the MICIT has eight separate kinds of instrumentations, each of which have set signal ranges and signal types.

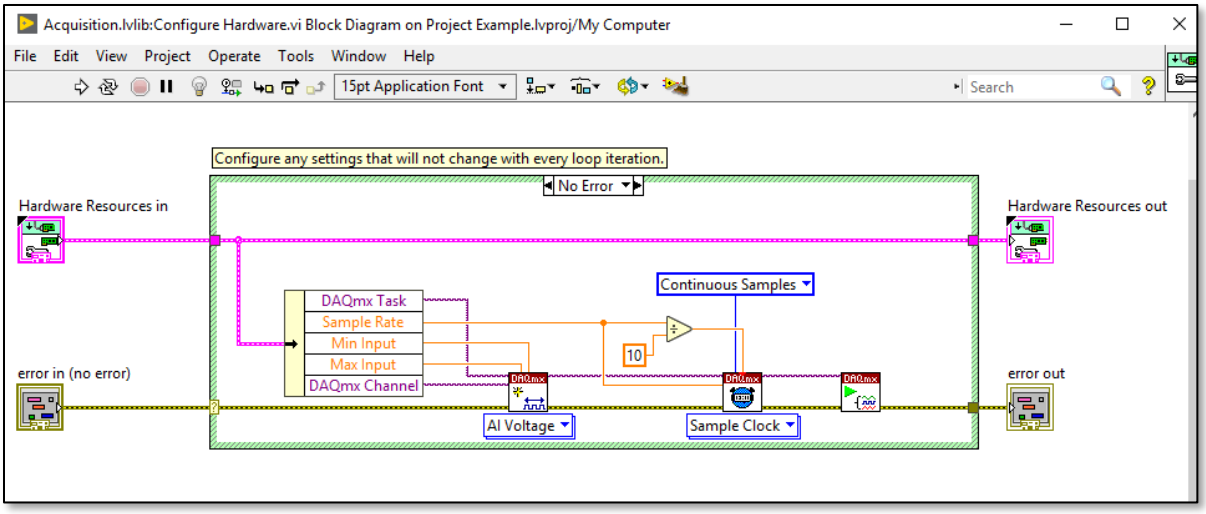


Figure 4.8 Template "Configure Hardware.vi"

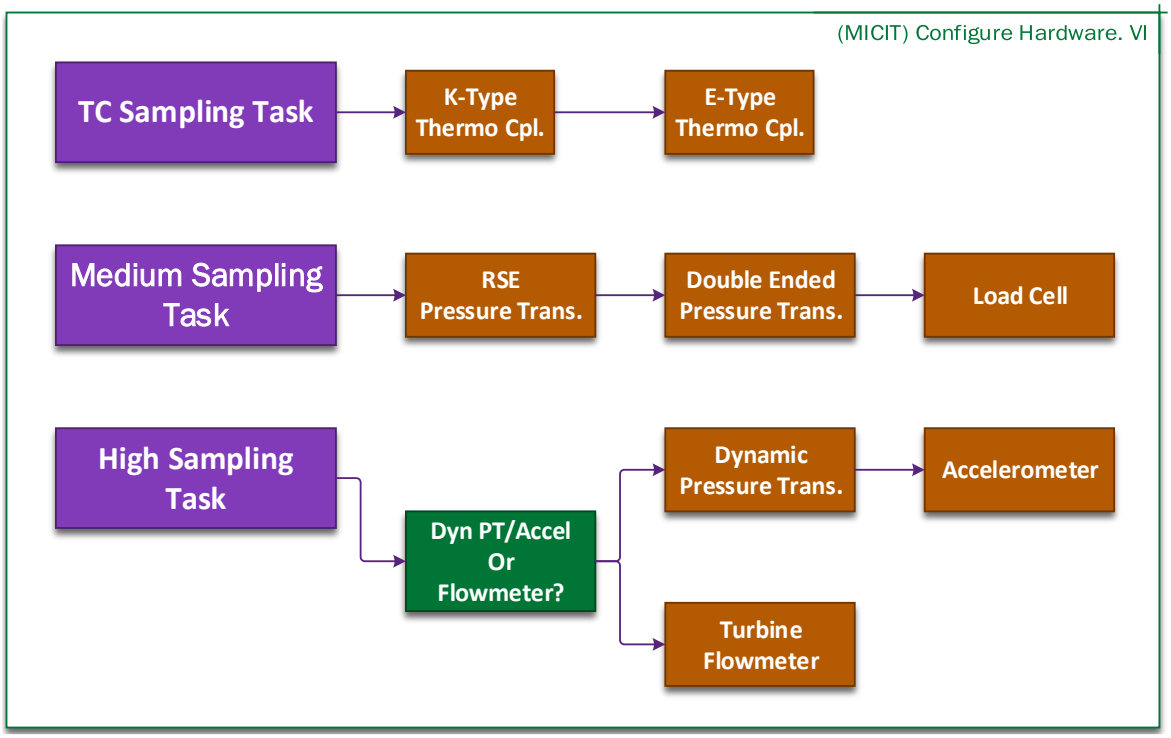


Figure 4.9 Overview of MICIT "Configure Hardware.vi"

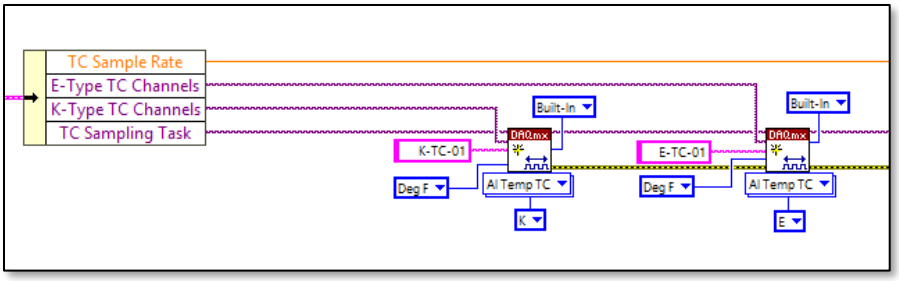


Figure 4.10 Thermocouple task

The first portion in the "Configure Hardware.VI" creates the DAQmx channels for the thermocouples, starting with the K-type and followed by the E-type. They are given a generic name like "K-TC-01" and LabVIEW automatically names the rest in a given channel in numeric order (i.e. "K-TC-02, K-TC-02…). Each "Create DAQmx channel" has the following settings:

- Set for an analog input (AI) for thermocouples.
- Given a generic yet descriptive name
- Temperature scale set to degrees Fahrenheit.
    - This processes the signal voltage into a temperature measurement
- Uses the "Built in" setting for the input terminal configuration.

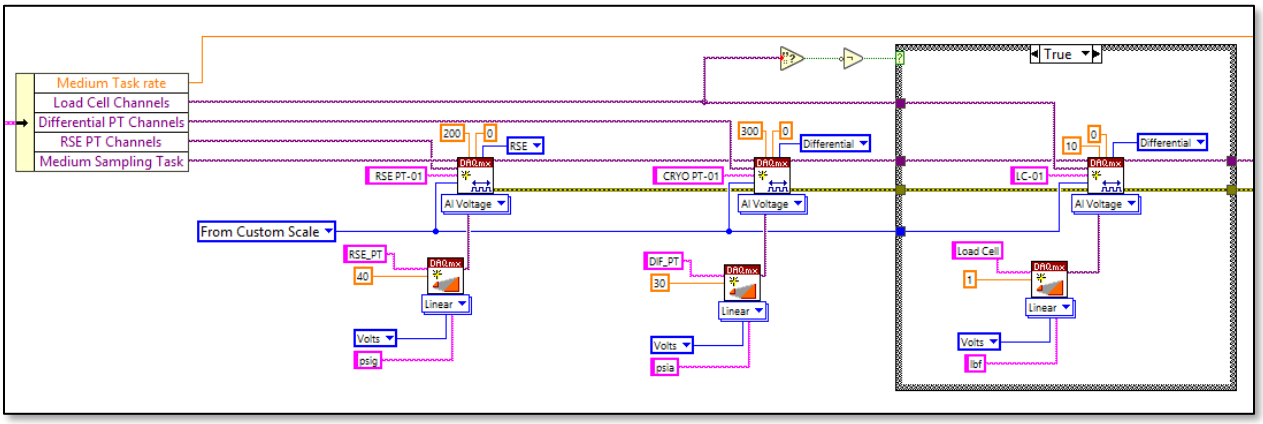


Figure 4.11 Static pressure transducers and load cell task

After the thermocouples come the static pressure transducers and, if applicable, the load cells. Since not every set up is equipped with load cells to measure, a chase structure was

introduced to check if there are any load cell channels specified at all. If not, the channels will not be created. Otherwise, LabVIEW will return an error that it is attempting to create channels from nothing. To the cDAQ, All of these measurements are voltage inputs and have the following settings:

- Set for an analog input (AI) for voltage.
    - Referenced single ended (RSE) pressure transducers are defined first
    - Differential signal pressure transducers are defined next
        - These coincidentally are the cryo-rated pressure transducers that have their signals amplified.
- Load cells are defined last and also function using a differential signal.
    - Given a generic yet descriptive name
    - Have lower and upper bounds defined
    - Uses the “Built in” setting for the input terminal configuration.
- Each use a custom linear scale to process the raw voltage to a pressure or pound-force measurement.
    - This is a blanket scale applied to ALL of the instrumentation defined in a “Create DAQmx channel” sub.vi. Not every pressure transducer has the exact same behavior as indicated in their calibration sheets. In the case of the amplified pressure transducers, the custom scales are applied on the amplifier side.

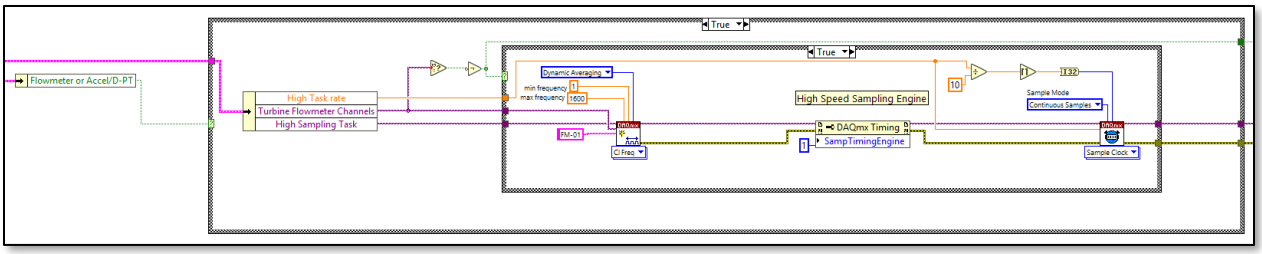


Figure 4.12 Turbine flowmeter task

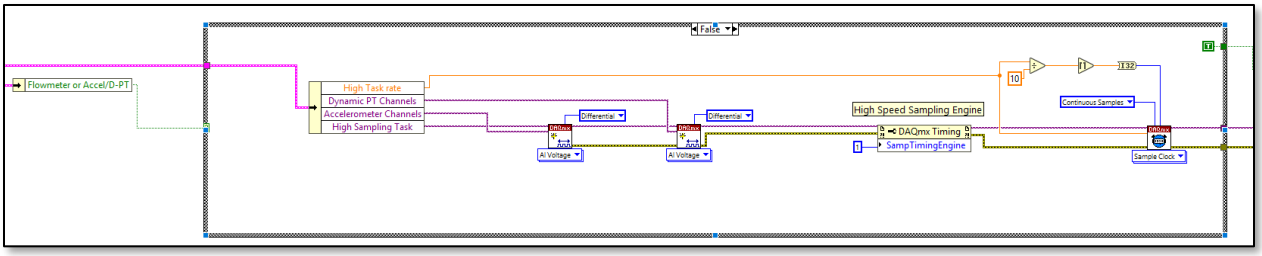


Figure 4.13 dynamic pressure transducer and accelerometer task

Finally, is a chase structure that switches between creating and defining the channels for either accelerometers and dynamic pressure transducers, or for turbine flowmeters that function using a digital frequency counter.

4.3.4 Acquire.vi setup

With the proper hardware configurations established in “Configure Hardware.VI”, next is “Acquire.VI”, still under the same “Acquisition” folder as the previous VIs. The purpose of “Acquire.VI” is to organize how all of the acquired signals will be displayed and what will be logged in the data file each experiment will produce.

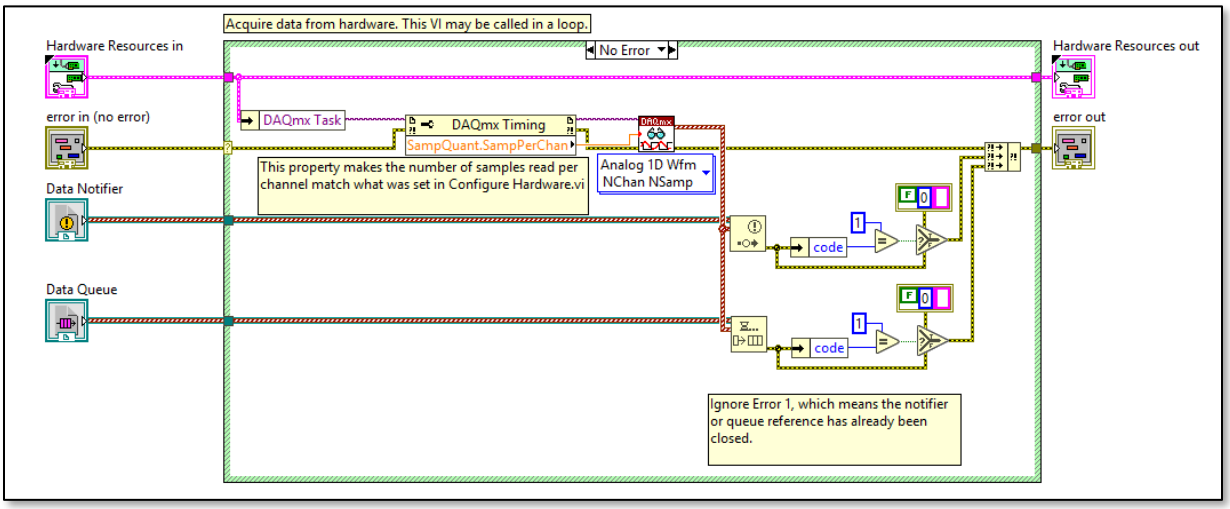


Figure 4.14 Template "Acquire.vi"

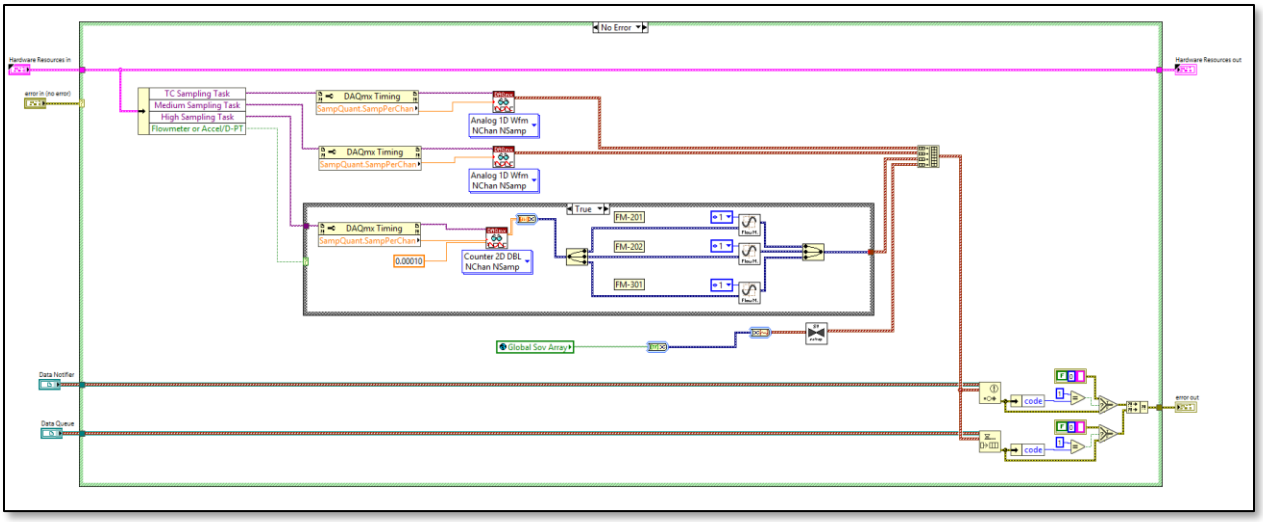


Figure 4.15 MICIT "Acquire.vi"

As it is, it is a simple.VI, but an additional feature was added for the system to also record the state of the solenoid valves. this feature uses a global variable that passes through a custom-made sub vi that attaches a channel name for the solenoid valves as they are recorded.

4.3.5 Settings Dialouge.vi setup

After the “acquire.VI” comes the “Settings Dialouge.VI” which can be found under the “Settings” folder separate from all of the previous ones. This is a menu that appears whenever there is a need to specify or adjust how everything is plugged into the system (more on that in the very next section). This vi gets very intricate because of the MICIT’s capabilities and. The default vi that comes with the template is overly simplistic for our use case, and a lot of functionally and features were added and removed along with a descriptive explanation.

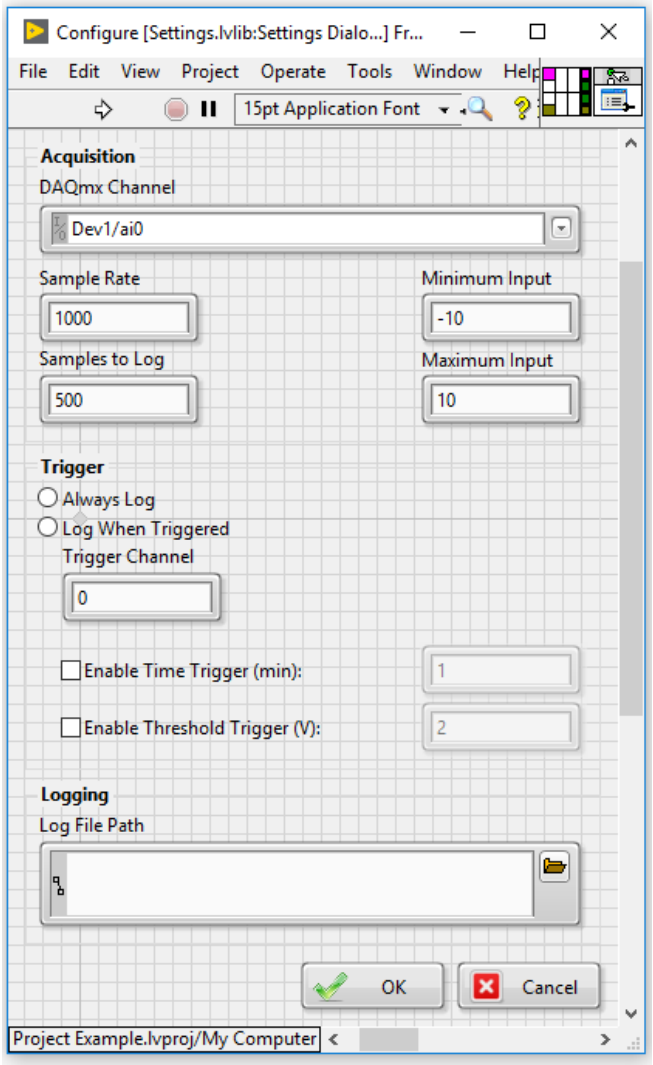


Figure 4.16 Template "Settings Dialogue" front panel

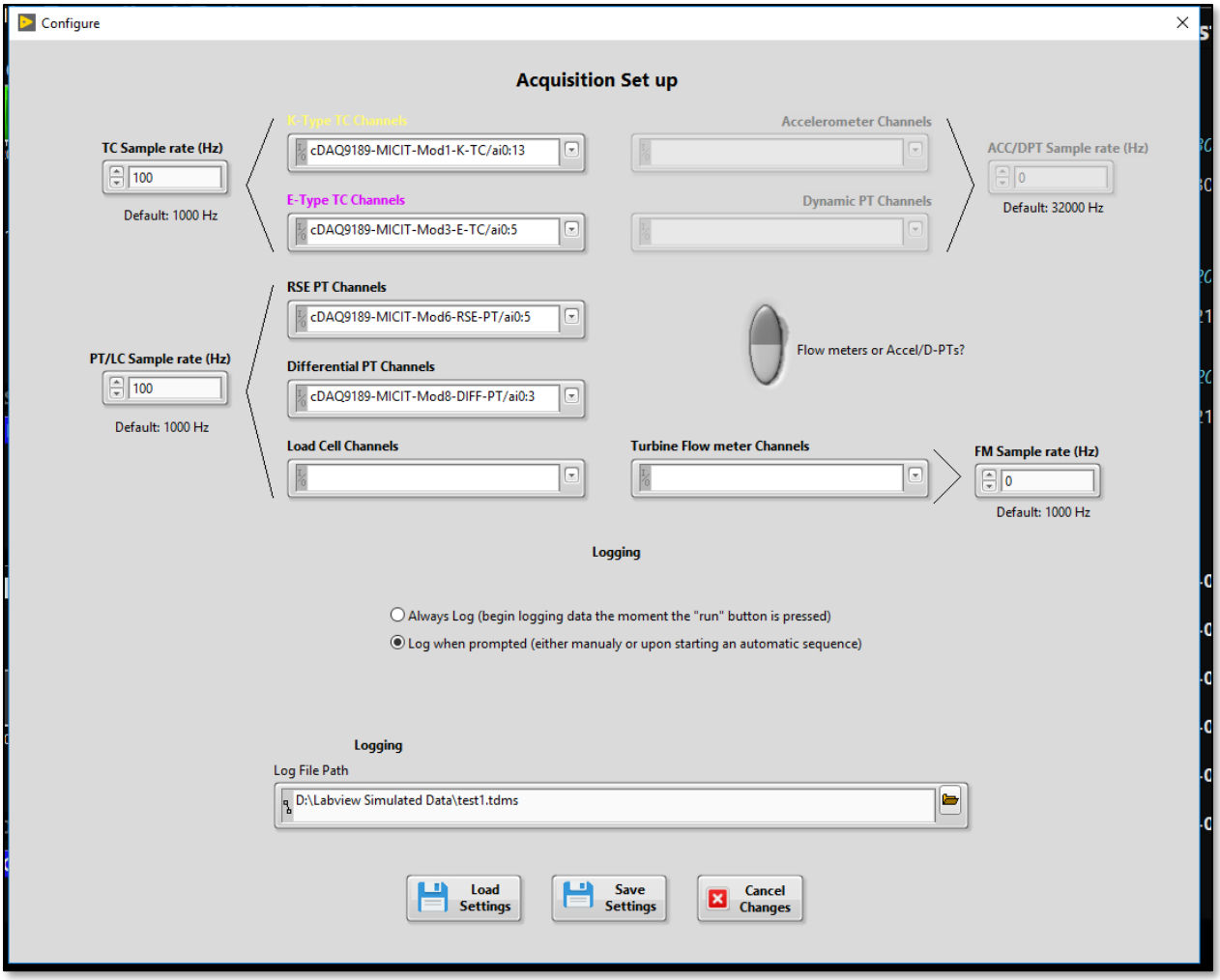


Figure 4.17 MICIT "Settings Dialogue" front panel

1. The ability to save and load from a specific settings file (.xml extension)
2. Ability to adjust the sample rates of the three separate clocks
3. Ability to specify the location of the transducers for each type of instrumentation
4. Ability for the third task to switch between sampling accelerometers and load cells, or turbine flow meters.
5. Simplified the logging menu.
6. Error messages when an improper setting is inputted.
    a. This won't catch every possible error.

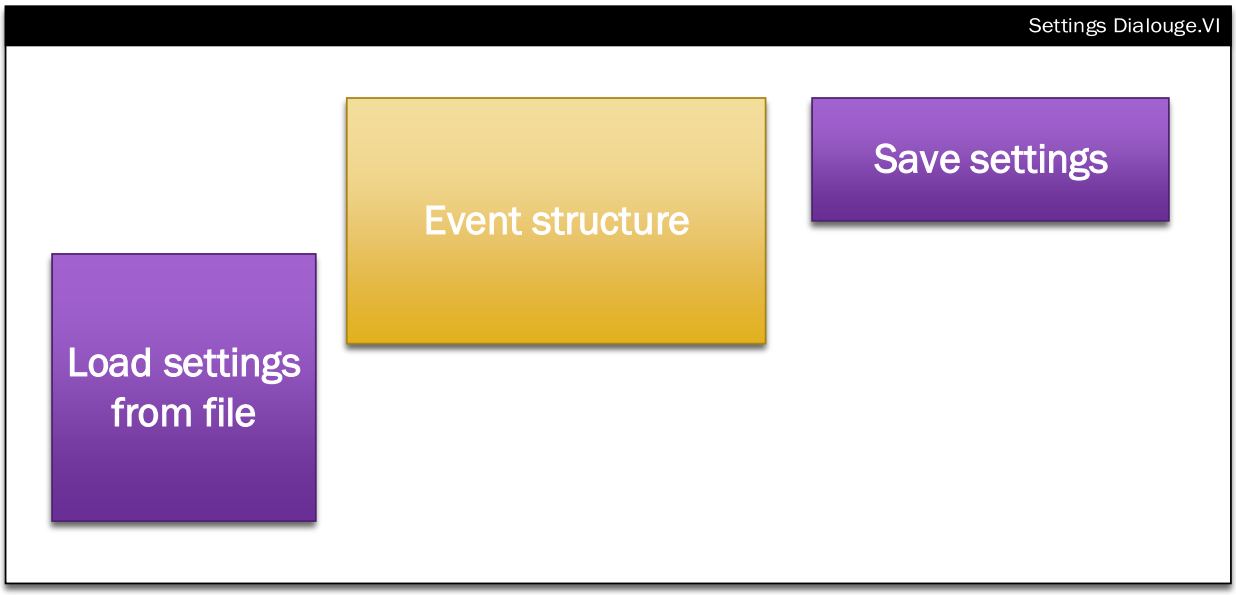


Figure 4.18 Overview of "Settings Dialouge.vi" wire diagram for both Template and MICIT versions

4.4 FPGA Overview and Block Diagram

This section will go through the FPGA code that is set up in the cRIO. The VI can be found under the cRIO target in the LabVIEW project explorer.

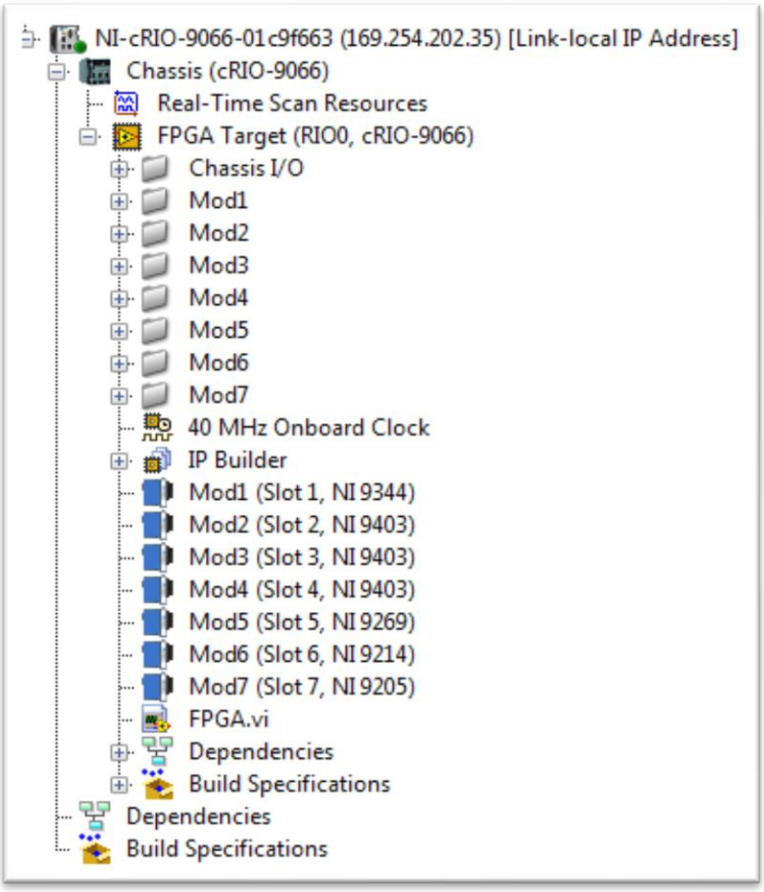


Figure 4.19 cRIO tree in LabVIEW project file

If for some reason, any of the seven modules listed are missing (which is a rare bug that happens) the FPGA.vi has to be backed up. Then cRIO is removed entirely from the project before it is re-added to the project file. The modules will then appear in the order that they are installed. The FPGA code will not work if all the modules are not listed, since the VI will have to read to or manipulate them to control the valves.

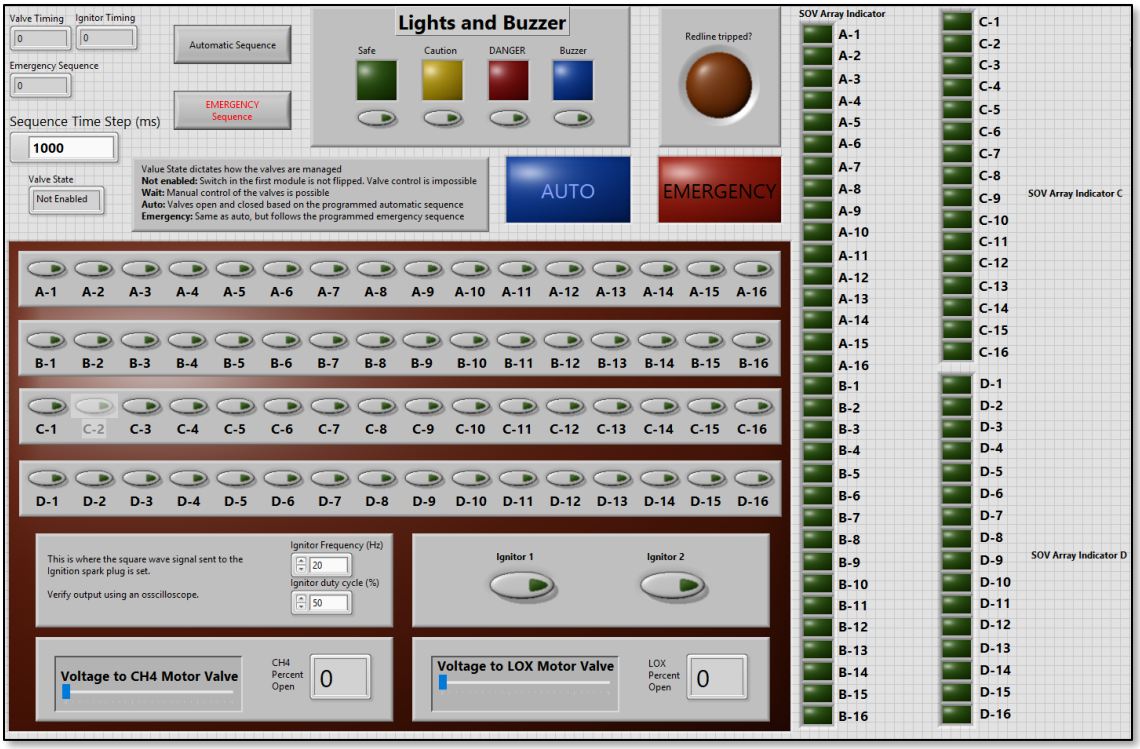


Figure 4.20 Front panel of FPGA.vi

Above is the front panel for the FPGA code. You would only need to be here to test and troubleshoot the functionality of the FPGA. The Main.VI discussed in the previous section interacts with this code directly. The following is sort of a road map of the FPGA code and where to find each section. Further details of each section are provided in the order that they are numbered in the map below. The code itself has several comments and explanations that reiterate what is written in this manual.

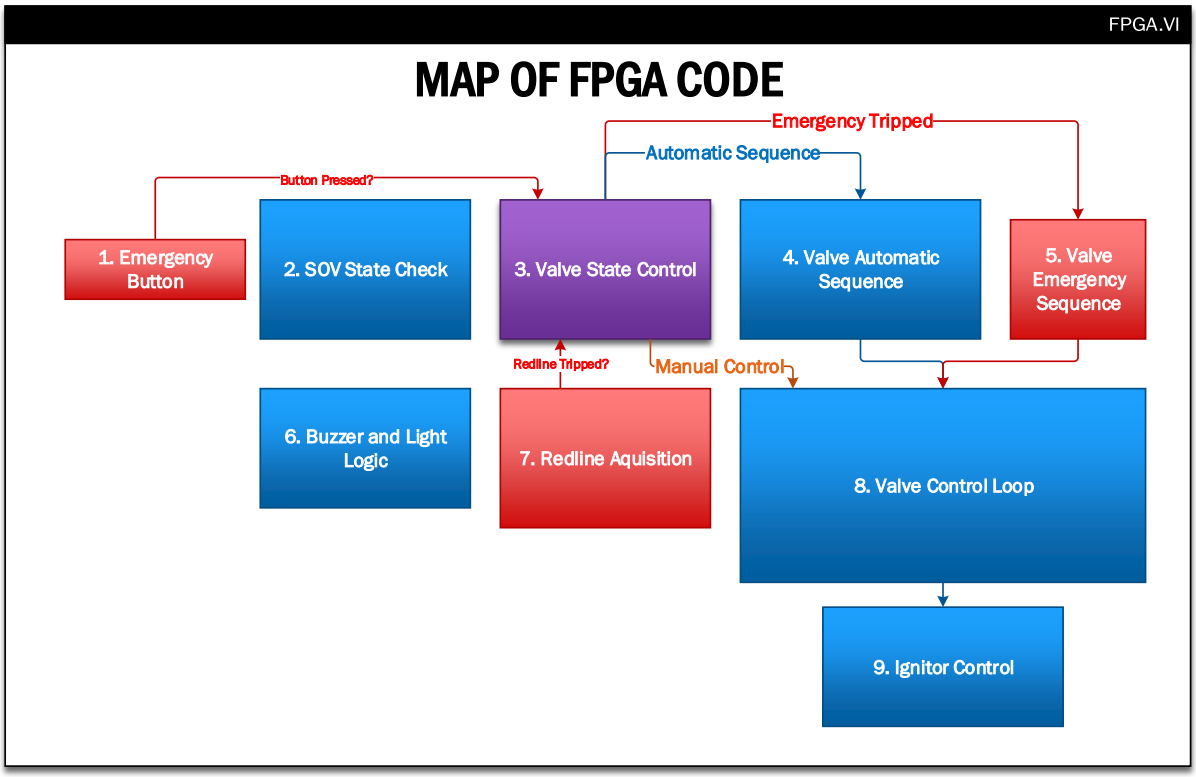


Figure 4.21 Overview of FPGA.vi block diagram

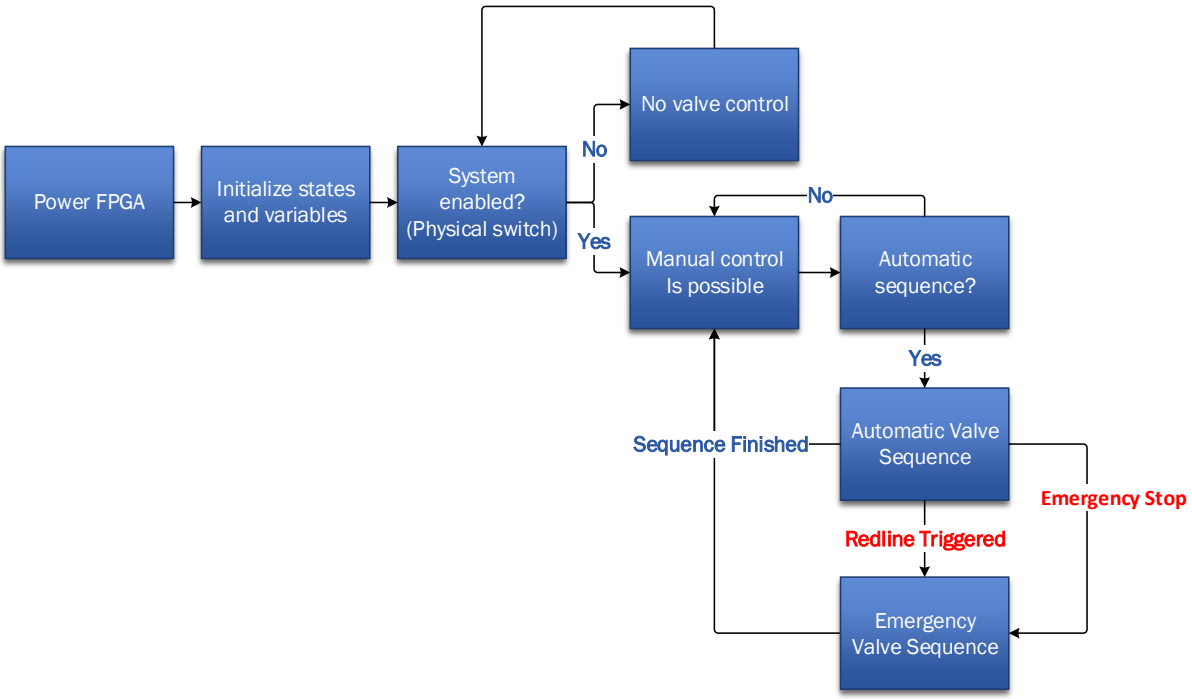


Figure 4.22 Flow chart of FPGA.vi functionality

4.4.1 Emergency Button

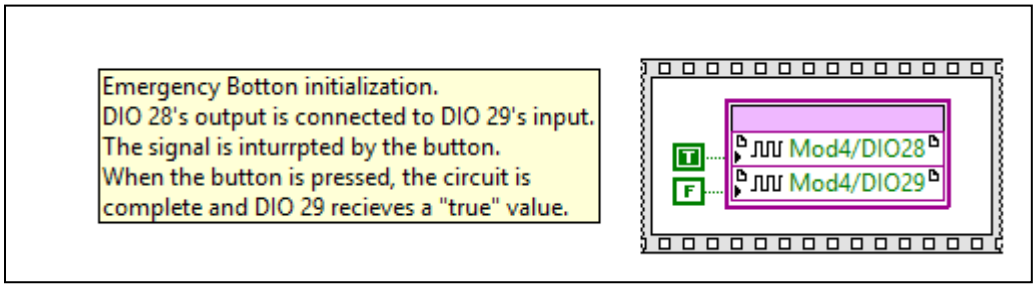


Figure 4.23 Emergency Button logic.

The output from DIO28 in the 4th module (an NI-9403) is connected to itself in DIO29. The emergency button interrupts this connection, and the circuit is only completed when the button is pressed down. When pressed, a TRUE signal from DIO28 reaches DIO29. When DIO29 receives the TRUE signal, it initiates the emergency valve sequence.

A mechanism keeps the emergency button held down after it has been pressed. To release it, twist the button clockwise

4.4.2 SOV State Check

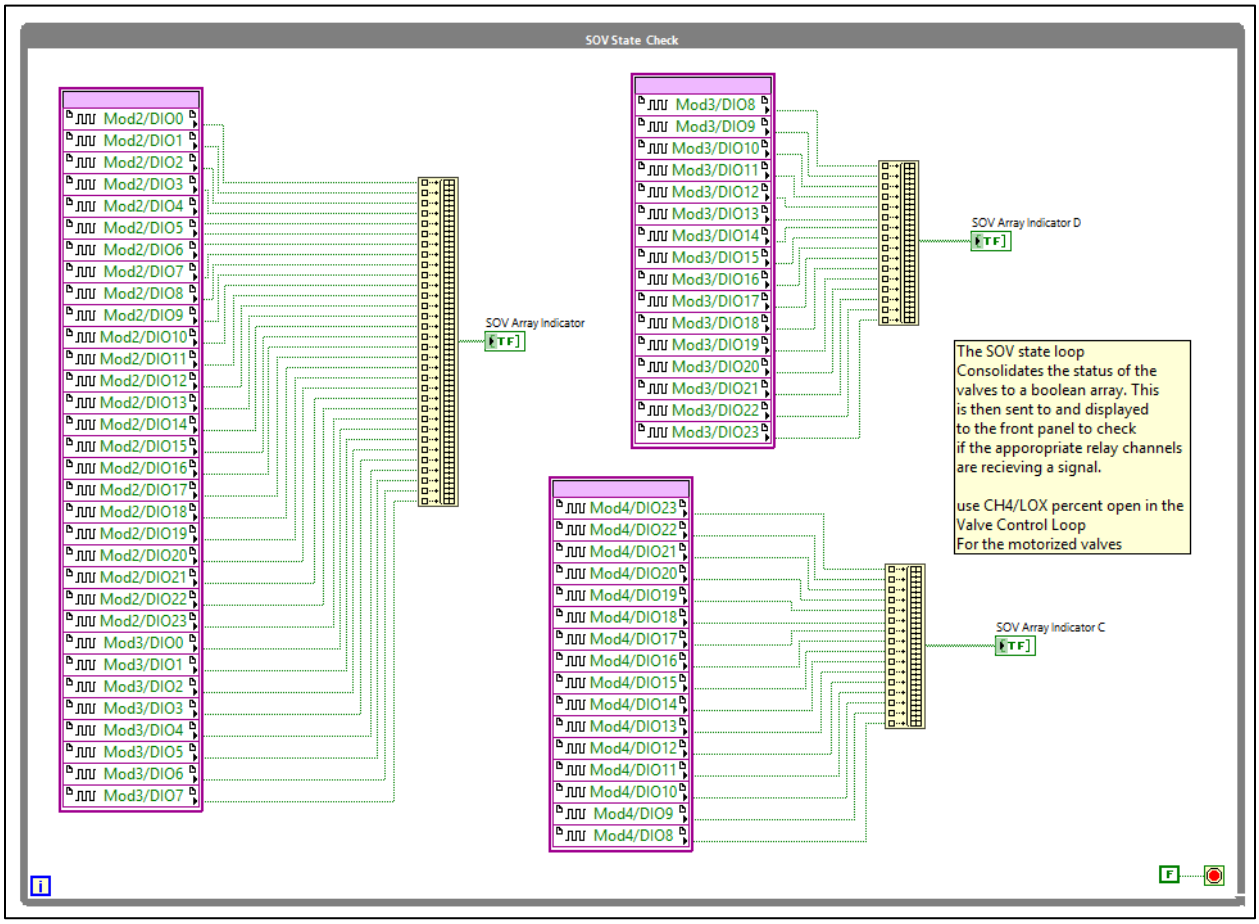


Figure 4.24 Solenoid valve state check while loop

All that the SOV (Solenoid Valve) State check loop does is consolidate the states of the NI-9403 Digital Input/output modules into a Boolean array. This Boolean array can then be called to verify if the appropriate valves are receiving power through the relay boards and are open during an automatic or emergency sequence.

However, it is important to keep in mind that this array only checks to see if the modules are outputting a digital signal to the relay boards. There is no way to directly verify in the software if the valves are truly receiving power and are responding. Carefully watch the sensors downstream of a valve for any expected changes. If a valve is unresponsive but the VI states that it should be open, check the wiring and ensure that it is not frozen the next time it is safe to approach the test bed.

4.4.3 Valve State Control

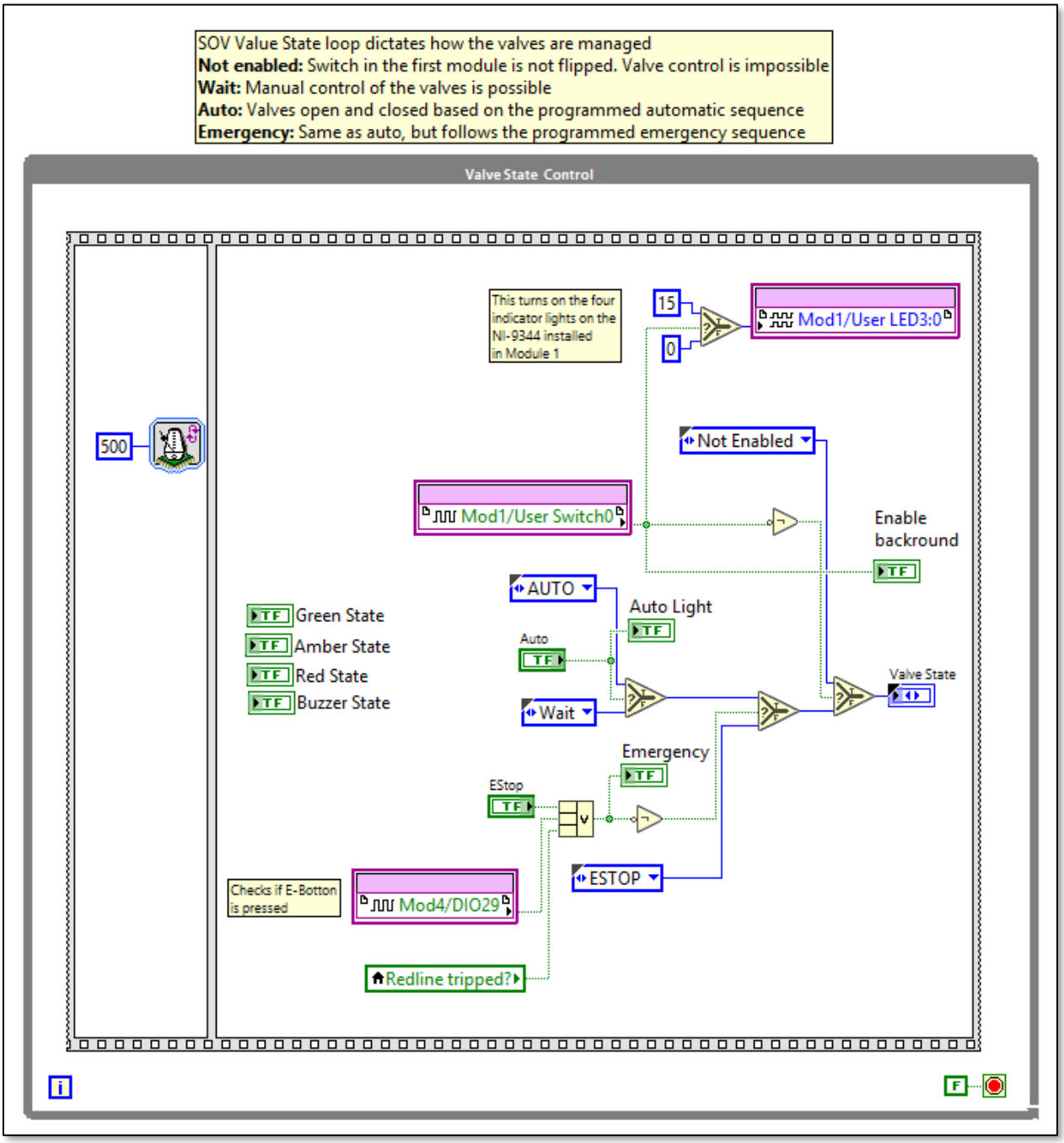


Figure 4.25 Valve state control logic and while loop

This entire loop updates once every half millisecond and determines if the system should operate in one of four states:

1st State "Not Enabled": As the name implies, this indicates that the system is not enabled. This is based on the top-most physical switch on 'Module 1' of the cRIO. If this switch is off, then there is no way for you to manipulate any of the valves whatsoever.

2nd State "Wait" (Manual valve control): If the switch on 'Module 1' is flipped on AND the automatic button is NOT pressed AND none of the instrumentations are above their redline values, then the valves are free to be manipulated at will. At the time of this writing, the control for the ignitors has not been programmed in.

3rd State "AUTO" (Automatic Valve control): If the switch on 'Module 1' is flipped on AND the automatic button IS pressed AND none of the instrumentations are above their redline values, then the valves will open and closed based on the programmed automatic sequence (See very next section). The time step of the sequence can be freely adjusted under the Unsigned-32 bit integer control "Sequence Time Step (ms)", though right now the default setting is 1000 ms or 1 second.

4th State "ESTOP" (Emergency Valve control): If the switch on 'Module 1' is flipped on AND at least ONE of the instrumentations are above their redline values, then the valves will open and closed based on the programmed emergency sequence (See section 4). This state takes priority over both the manual and automatic states. The time step of the sequence is currently set at 1000 ms, or 1 second.

4.4.4 Valve Automatic Sequence

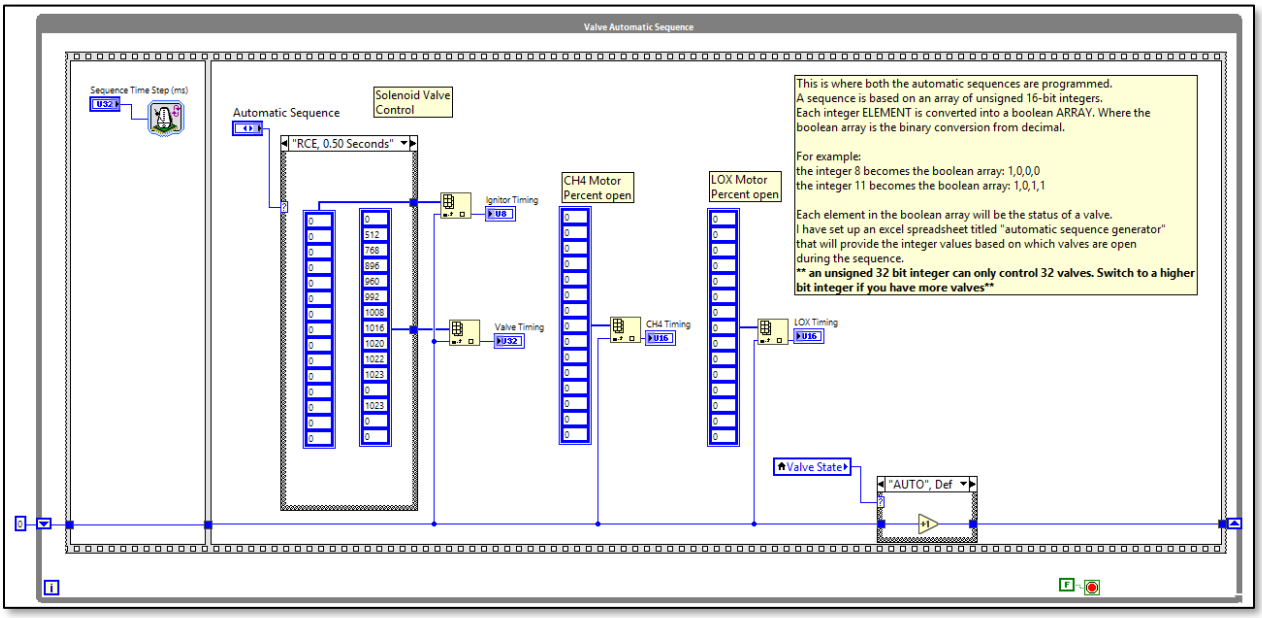


Figure 4.26 Automatic valve sequence while loop

This loop is where you program the both the solenoid valve and motor-controlled valve sequence. The way the sequence for the solenoid valves works is that it takes an array of unsigned 16bit integers. Each element in the array of integers are converted from decimal into binary. Since the integers are 16 bit, each individual bit becomes an element in a binary array. Each of those binary elements are then used to dictate the status of a valve.

The loop goes through each iteration the sequence after a pre-determined time step. Right now the time step is a default value of 1000 ms, or 1 second. I have set up an excel spreadsheet titled “Automatic sequence generator” that provides the integer value based on which valves are open at any given time step, and should be found in the same folder as this manual.

The motor valves are straight forward and are just based on the percentage of how open they are requested to be. This is subject to change however, but only slightly.

4.4.5 Valve Emergency Sequence

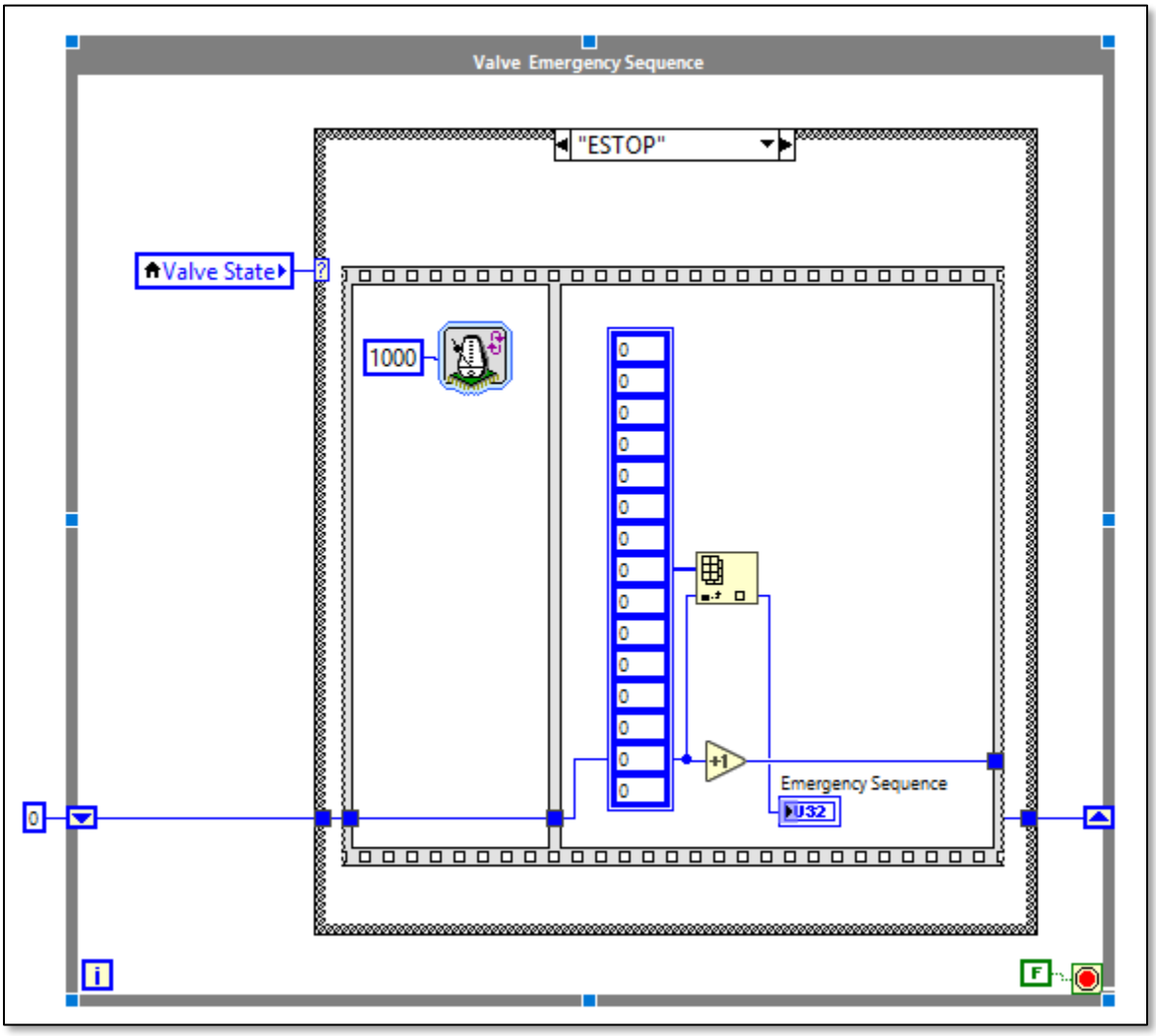


Figure 4.27 Emergency valve sequence while loop

This loop works identically as in the automatic sequence. The only difference is that there are not any arrays for the motor-controlled valves. They are unnecessary as in an emergency case, they are already programmed to immediately shut. The only valves that need to be controlled are the ones that manage the purging of the system to remove the fuel and oxidizer from the system.

4.4.6 Buzzer and Light Logic

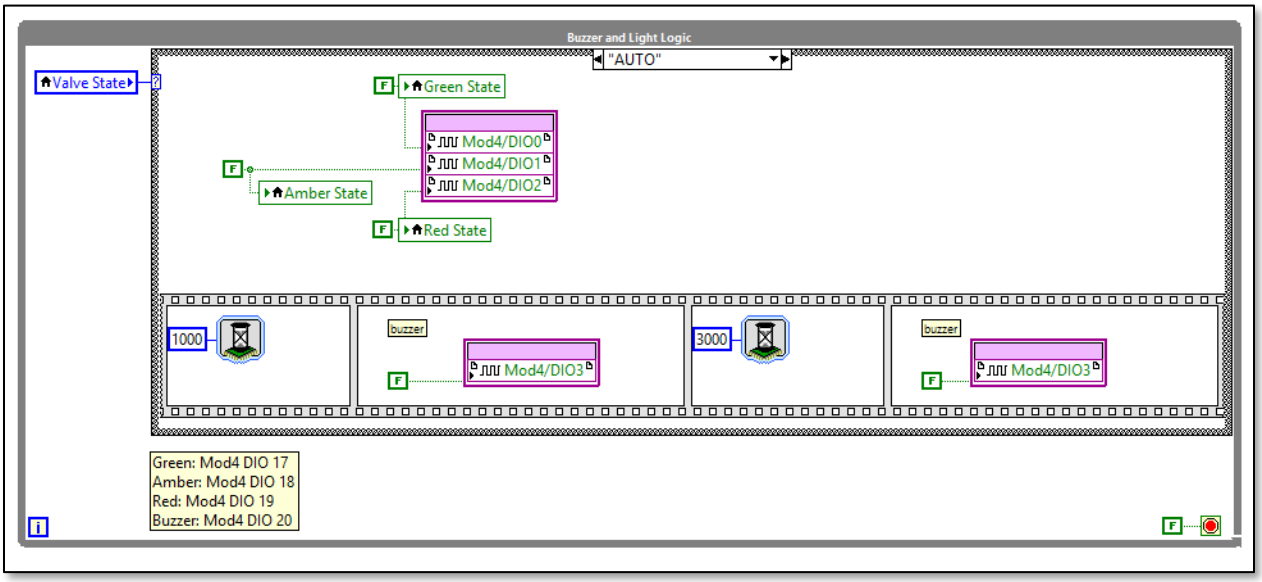


Figure 4.28 Buzzer and indicator lights logic and while loop

As the name implies, this loop controls how the lights and buzzer are controlled. When the system is not enabled, the lights and buzzer can be freely manipulated. It only when the system is enabled that the lights and buzzer are set to their programmed logic.

4.4.7 Redline Acquisition

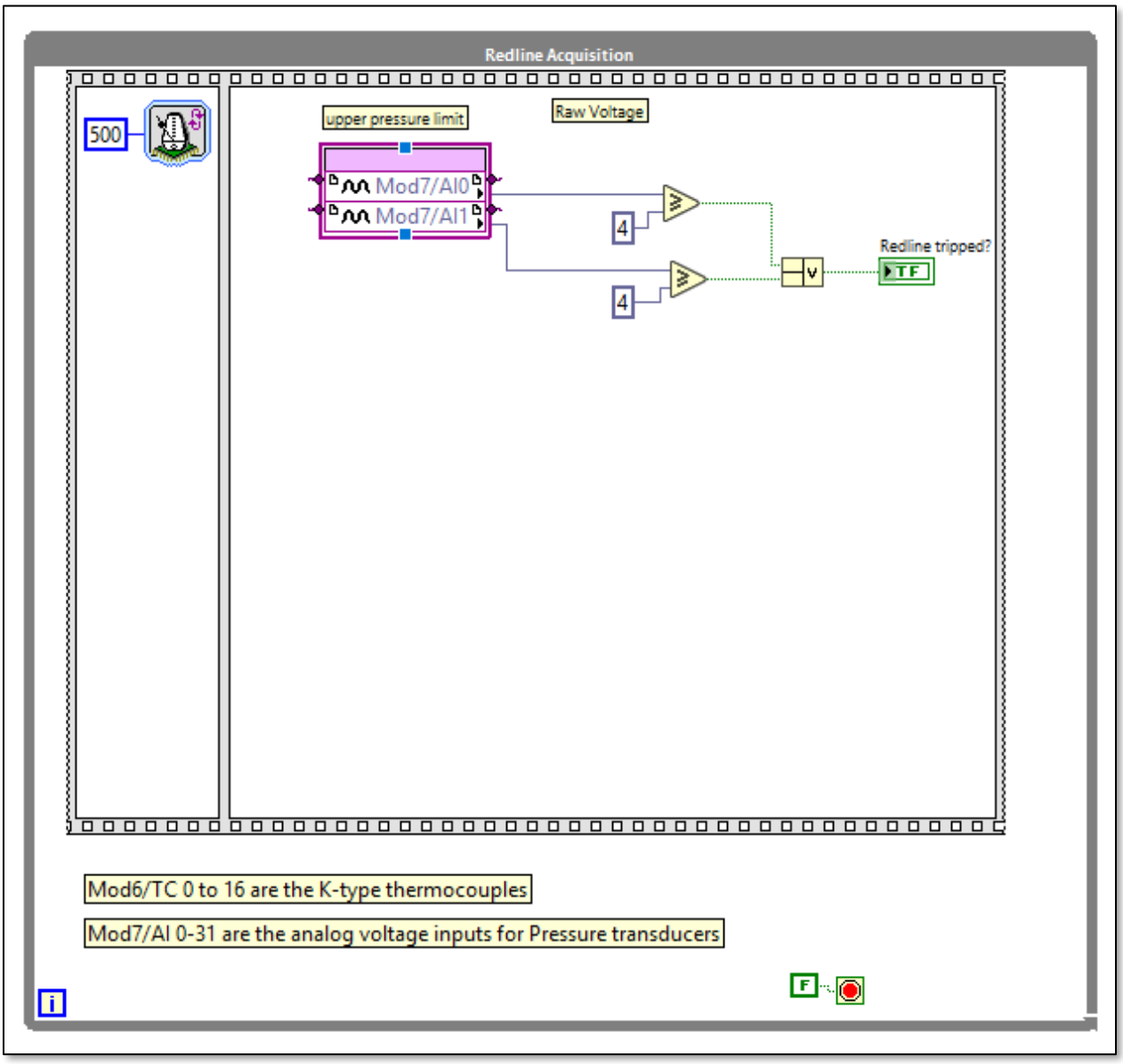


Figure 4.29 Redline acquisition (unfinished)

The Redline Acquisition loop is where the conditions for redline are programmed in. They take the readings from the thermocouple and analog voltage input modules (Modules 6 and 7 respectively) to make this determination. Since the voltage signals are not processed, you must use their raw values to determine if they are above a critical value. For instance, if 4.0 volts from a pressure transducer means 400 psig, and that constitutes a redline, the “Redline Tripped?” Boolean variable switches to true. That switch will then the valve state will switch to “ESTOP”

and begins the emergency sequence. There is not any redundancy. If ANY single one of the readings reach a critical value, then the emergency state will be set.

4.4.8 Valve Control Loop

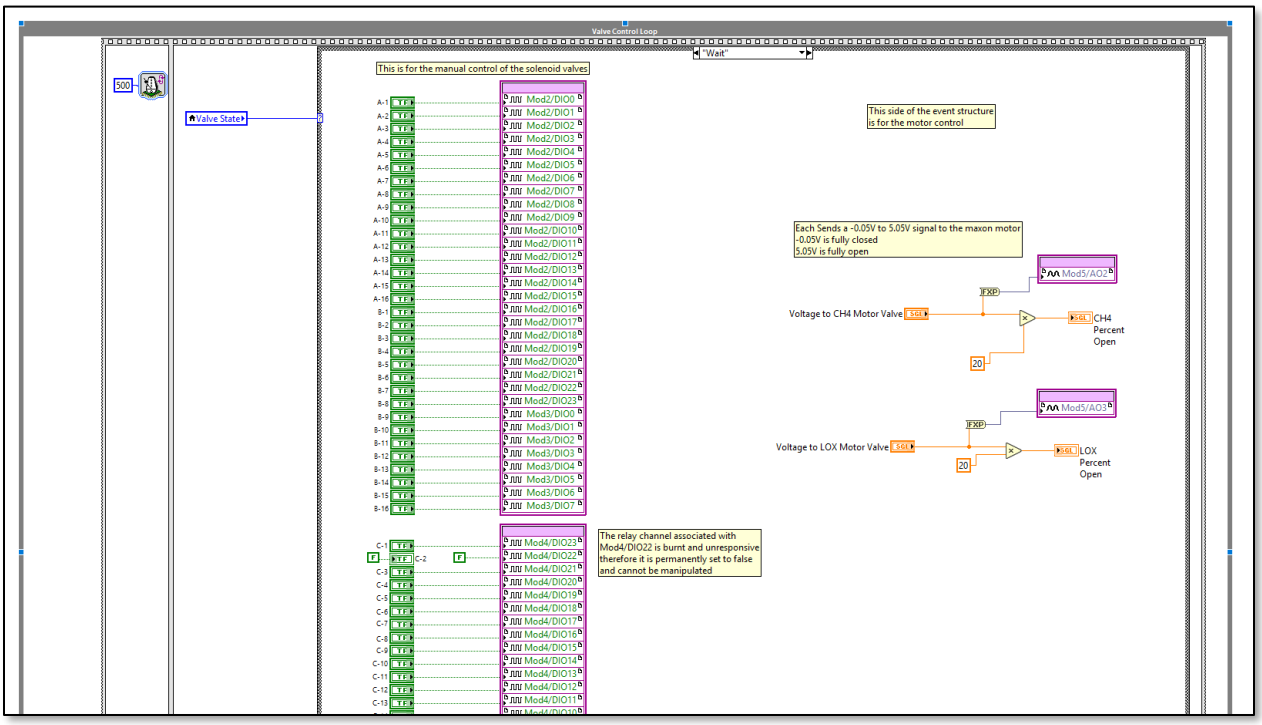


Figure 4.30 Valve control while loop

The Valve Control Loop is, depending on your point of view, where the magic happens. This is where the signal to the relay boards is managed and its function varies depending on the valve state established from the valve state control loop. Visually, this is the largest part of the code since it has controls for up to 64 solenoid valves and two motorized valves. It is so large, I could fit the entire thing into a single screenshot.

1st State "Not Enabled": Everything is set to "False" and the only thing that can be manipulated on the lights and buzzer. Valves will not respond to any commands.

2nd State "Wait" (Manual valve control): The Boolean controls to the relays and the signals to the motor valves and ignitor can be freely manipulated.

Only manipulate the valves and ignitors through the FPGA.vi to test their functionality

3rd State "AUTO" (Automatic Valve control): Takes the current integer value from the "Valve Automatic Sequence" loop and converts it into a Boolean array after the binary conversion from decimal. The motor valves themselves convert the integer into a voltage output sent to the motors where -0.05 Volts is fully closed, and 5.05 Volts is fully open.

If you look at the motor calibration, 0 Volts is fully closed, and 5 Volts is fully open. I set the solftware to range from -0.05 and 5.05 Volts because their controllers are sensitive enough to respond to noise in the millivolt range

4th State "ESTOP" (Emergency Valve control): Just as in the AUTO state, only that the motor valves are immediately shut and the emergency valve sequence is used to control the state of the solenoid valves

4.4.9 Ignitor Control Loop

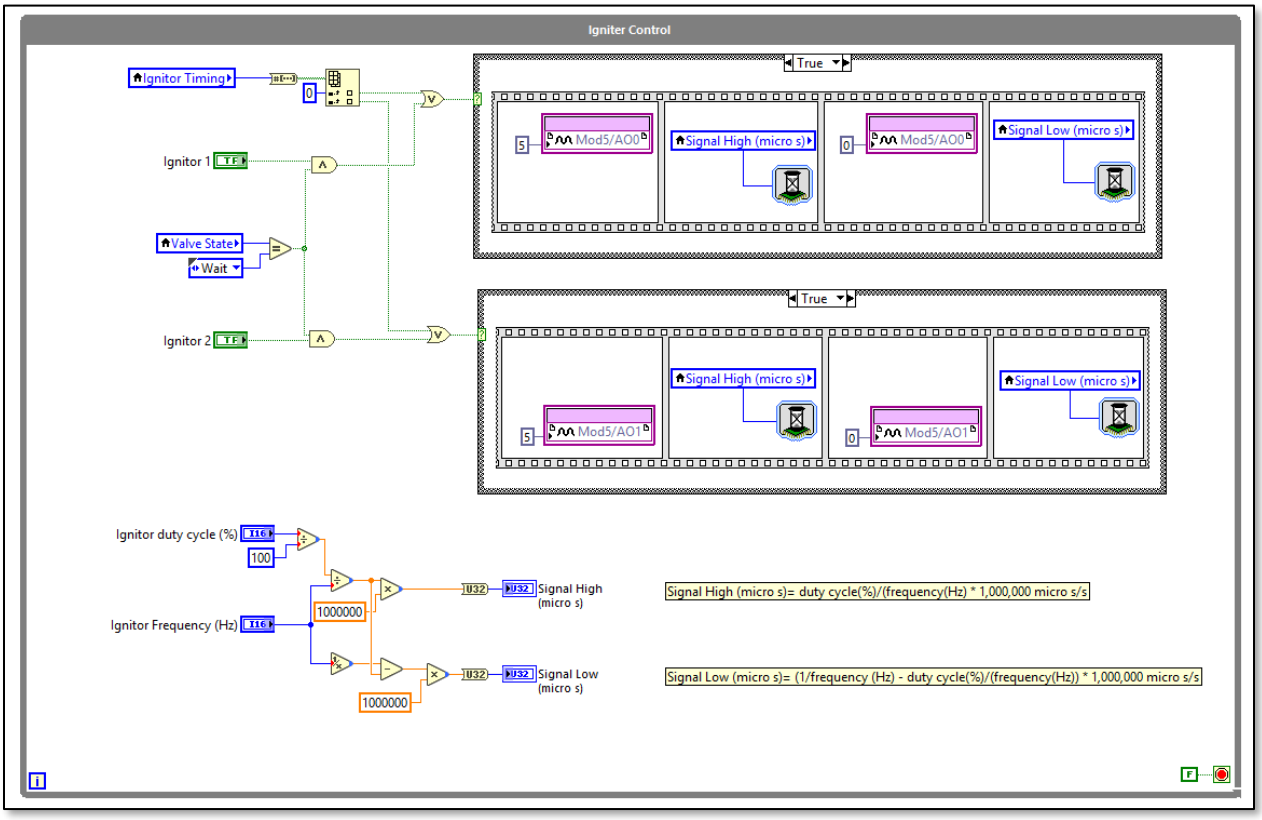


Figure 4.31 Ignitor control while loop

The Ignitor Control Loop instructs the Analog Output module to send a 5 to 0 volt square wave signal to the spark coil installed in either the RCE, CROME, or CROME-X chambers. It is programmed in a way to easily and readily adjust the frequency and duty cycle of the outgoing signal.

- When the signal is high, or at 5 Volts, the ignition coil begins to charge and store energy.
    - **The ignition coil should never be receiving a constant "high" signal.**
- When the signal drops to 0 Volts from 5, the ignition coil discharges its stored energy as a spark.
- The frequency of the square wave signal dictates how frequent the spark emanates from the electrode.
    - Too fast of a frequency will not give the coil enough time to charge and discharge.
    - Double check the rating of the ignition coil. A good default value is 100 Hz
- The duty cycle of the square wave can influence the spark energy
    - A longer duty cycle can mean more energy stored in the coil before discharging (but up to a limit). Too high of a duty cycle and the coil will not have enough time to discharge
    - Double check the rating of the ignition coil. A good default value is 50% at 100 Hz

4.5 MAIN.VI

Lastly is the Main.vi. This will be the vi that the test operator will see and that will provide all of the indicators and readings from any installed sensors. At the same time, this vi will be able to communicate to the FPGA code and manipulate the state of the valves while initiating either an automatic or emergency sequence.

Since this is the visual portion, the main.vi's graphical display can be simple or incredibly intricate depending on the developer. Creating an intuitive and user-friendly front-panel is an art form, and LabVIEW does provide the means to extensively customize every aspect of the panel. These customizations allow for unique Boolean buttons and indicators. For the main.vis associated with RCE and CHROME, Microsoft Visio and Adobe Photoshop were employed. The sky really is the limit when it comes to the visual aesthetics of the main.vi.

Aesthetics aside however, the main.vi still has the crucial purpose of allowing the user to interface with the system. In a bare-bones state, it is not too complex to develop and work with, which is fortunate if the luxury of time is not available to experiment and iterate on different front panel designs.

The Main.vi is where the state machine is found. It initializes the data acquisition, and transitions between different operating states depending on the user's input. What is found on the template does not have to be modified for the most part, except for two *while* loops on the bottom of the block diagram. Any additional loops would serve to enhance the functionality of the main.vi,

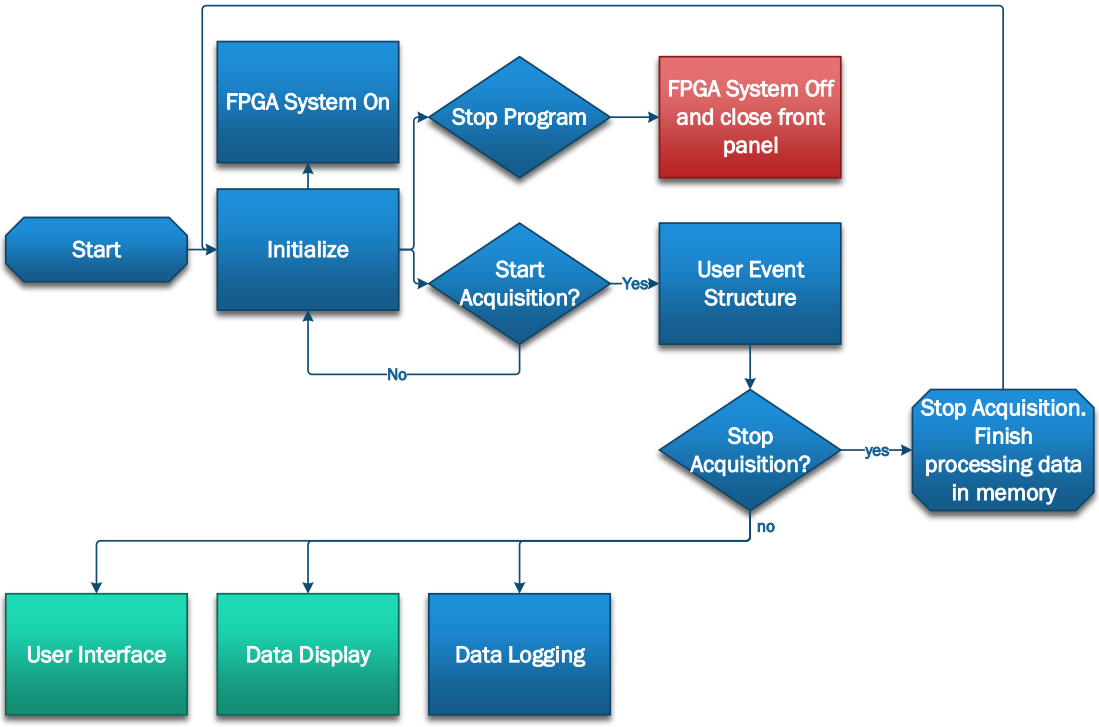


Figure 4.32 Flow chart of Main.vi operation

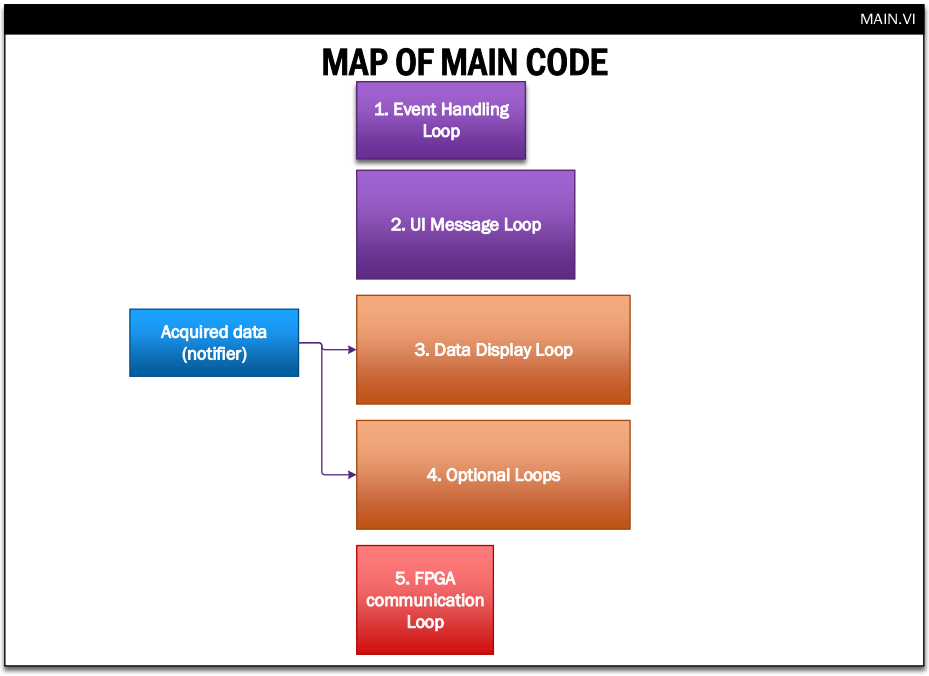


Figure 4.33 Main.vi Block Diagram overview

4.5.1 Event Handling Loop

The Event handling loop is on the very top of the block diagram and contains an event structure that ONLY executes if the user interacts with the buttons on the front panel. This event structure is what dictates the .vi to transition between key states, such as starting, acquiring, and stopping.

From the template, this portion of the code is left mainly untouched or only expanded on if the programmer opts to add more buttons to include additional states if necessary.

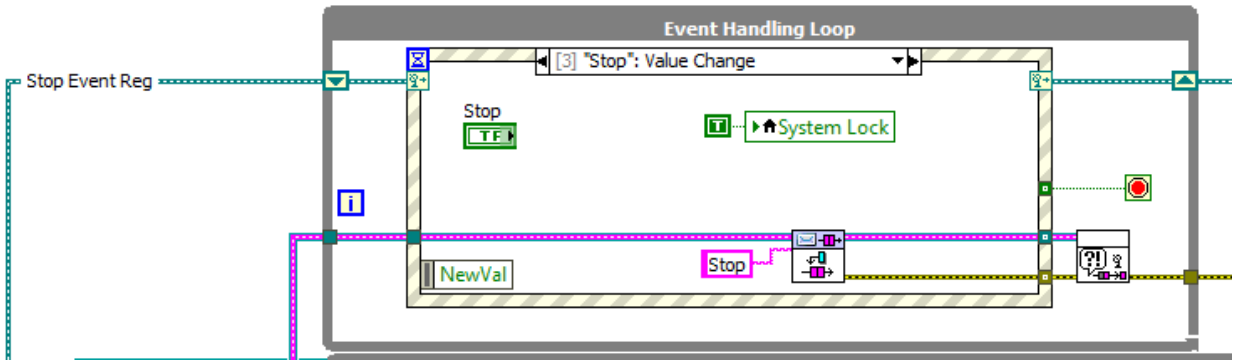


Figure 4.34 Event Handling Loop as seen in RCE_Main.vi

4.5.2 UI Message Loop

This portion of the Main.vi code changes depending on the current state of the state machine and communicates this state in a dialogue box. Here is how the system is initialized, set to acquire, stop and shutdown, etc. It is relatively untouched from what is provided from the template, except for the addition of a feature that disables the valve command buttons on the front panel until the “start” button is pressed.

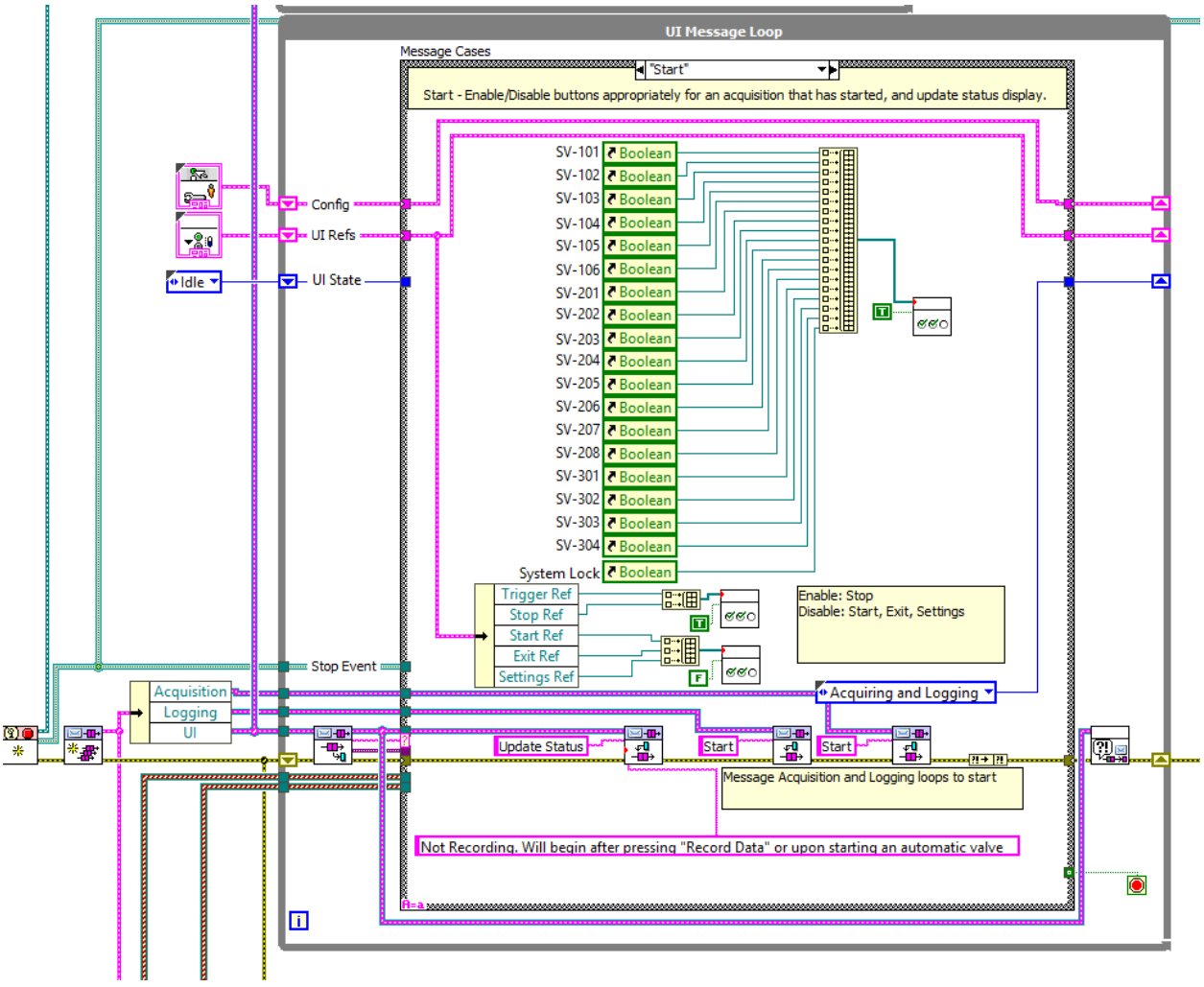


Figure 4.35 UI Message Loop as seen in RCE_Main.vi

4.5.3 Data Display Loop

The Data Display Loop is where all the indicators on the front panel for all the sensors are placed. This loop updates as fast as the notifier provides the data, which can be the acquisition rate of the fastest transducer or the processing ability of the computer.

The data comes in the form of a single large array that was built in the "Acquire.vi" from section 4.3.4. From the way the "Acquire.vi" is set up, the first indexes of the array carry the thermocouple data, followed by the static pressure transducer and load cell data, before finishing with either the turbine flowmeters or dynamic pressure transducers and accelerometers.

To conserve memory and avoid creating global variables to share between vi files, a sub.vi dedicated for displaying additional graphical interfaces can be placed inside this loop to update automatically and in tandem with the notifier.

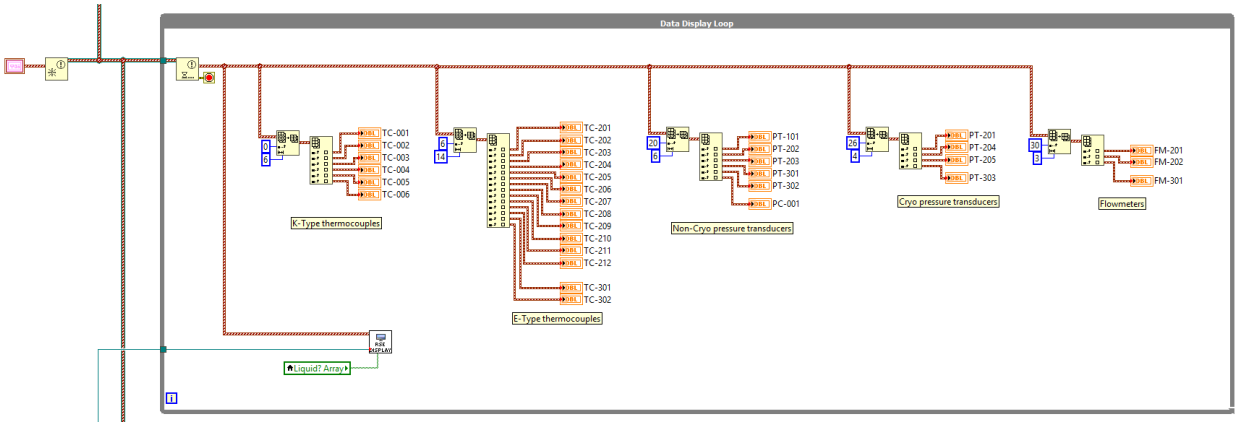


Figure 4.36 Data Display Loop seen in RCE_Main.vi

4.5.4 Optional Loop

An optional loop or loops below the Data Display Loop can be created to process any additional values or add more features to the code. In the example shown in the next figure, a separate while loop was established to determine the phase state of the fluid flow. It takes a thermocouple and pressure transducer pair and combines them with a RefPROP sub.vi to

determine the phase. There is no limit to the additional loops that can be created, provided the host computer can handle all of the additional processing.

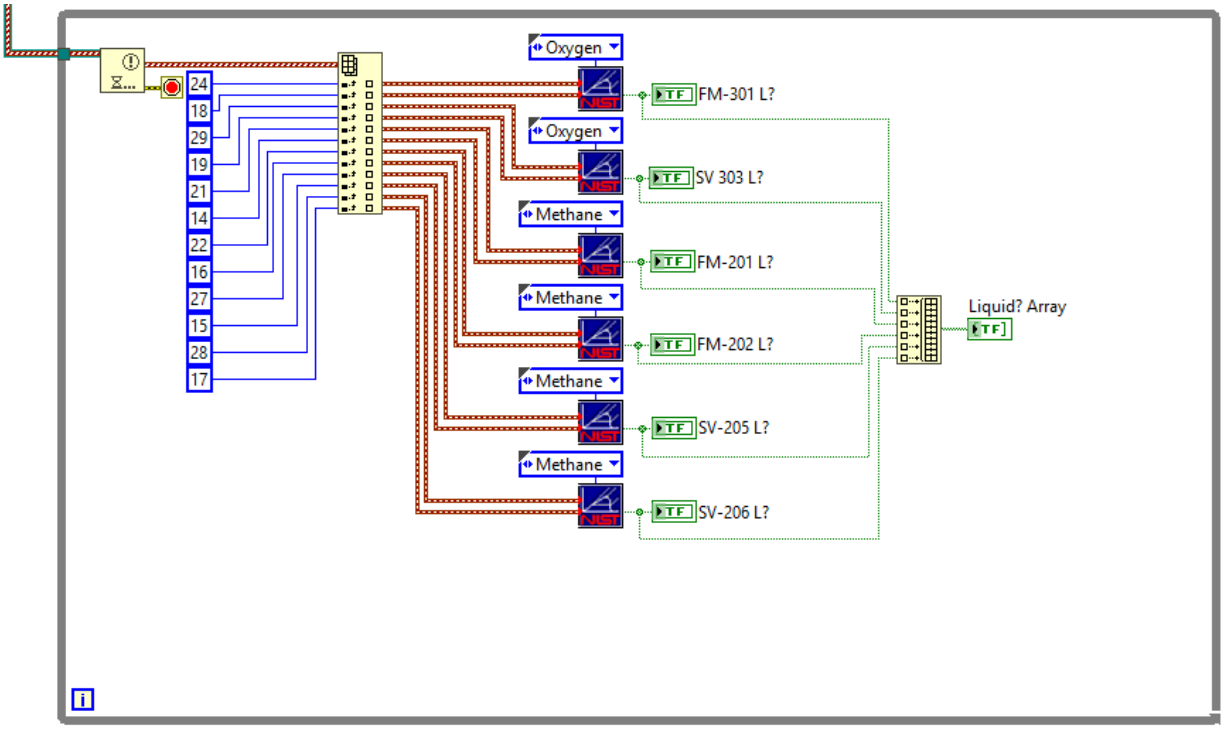


Figure 4.37 Optional Loop for indication of fluid phase as seen in RCE_Main.vi

4.5.5 FPGA communication loop

Lastly is the while loop dedicated to the host computer interacting with the FPGA in the cRIO chassis. Buttons on the front panel are tied to the buttons found in the FPGA.vi code. In other words, the state of the valves is not *directly* controlled with the Main.vi. Instructions are merely sent to the FPGA and then it performs the operation *if* it is in a state that allows for manual control to begin with. In section 4.4 where the FPGA programming is discussed, should the FPGA be in a state where manual valve control is not enabled, the buttons on the front panel will not do anything. However, because of this, it is recommended to add a condition in the main.vi to disable the buttons associated with valve control to mimic the FPGA's logic so that the user is aware.

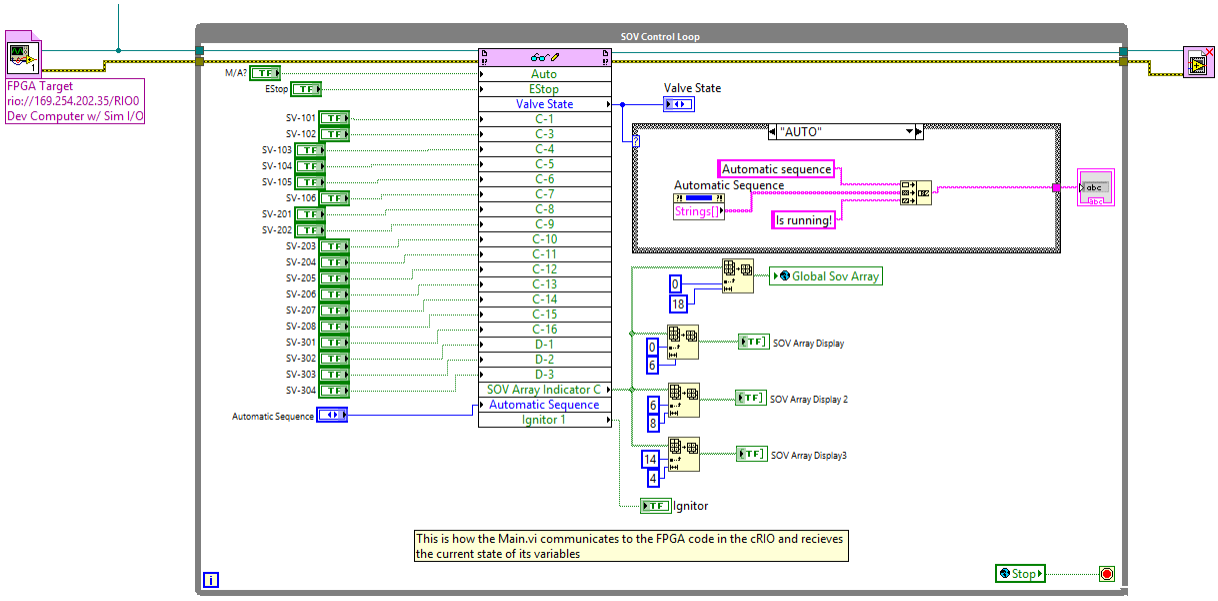


Figure 4.38 FPGA Communication Loop as seen in RCE_Main.vi

4.5.6 Main.VI Graphical User Interface

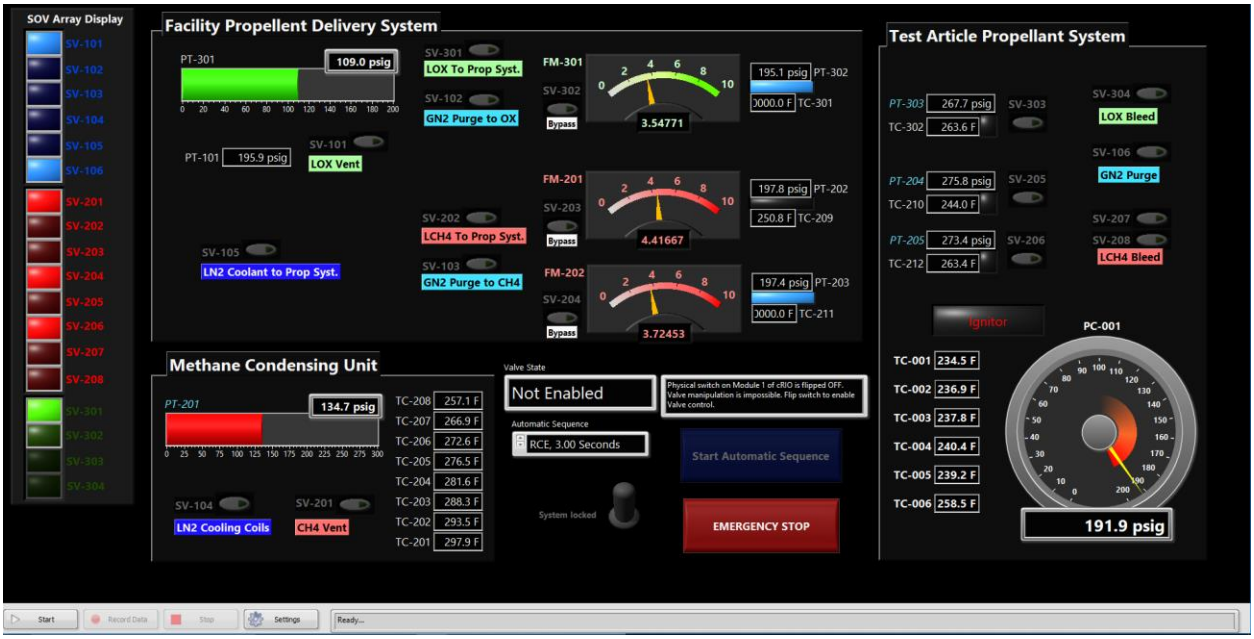


Figure 4.39 Main.vi front panel for RCE

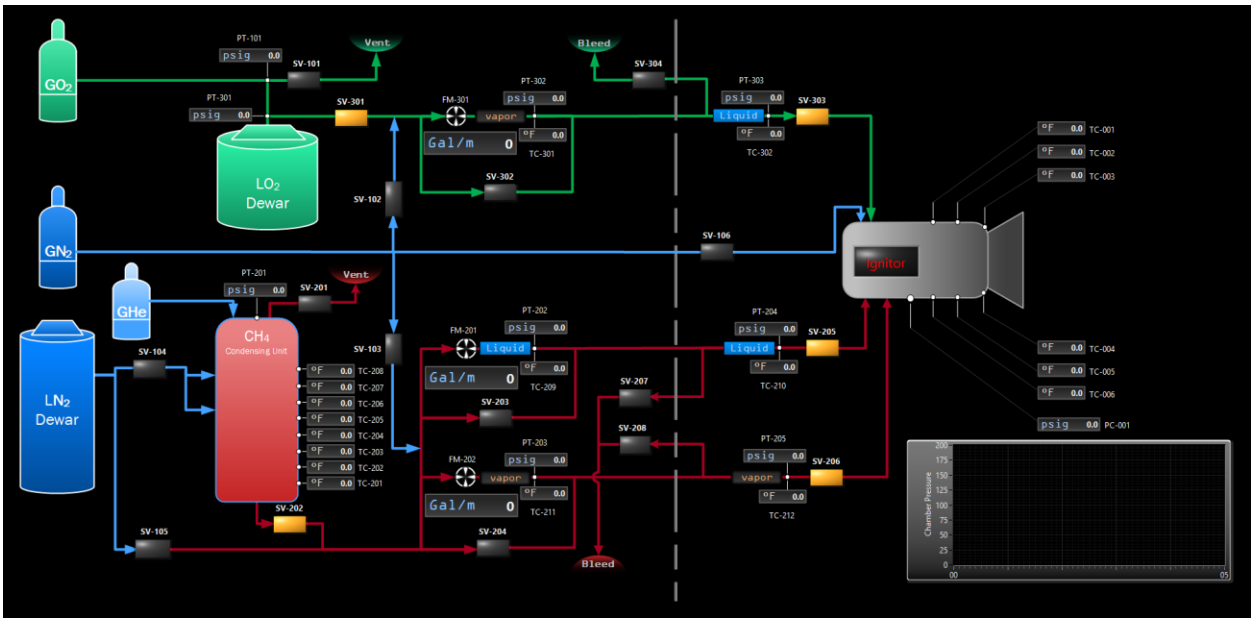


Figure 4.40 Plumbing display for RCE

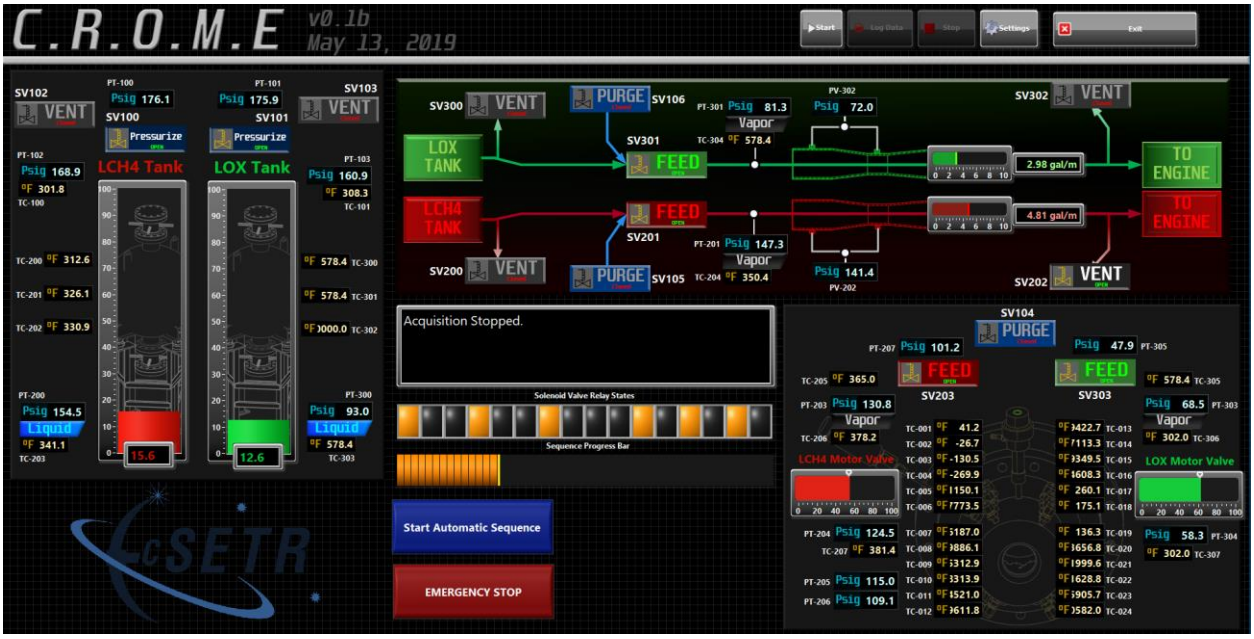


Figure 4.41 Main.vi front panel for CROME

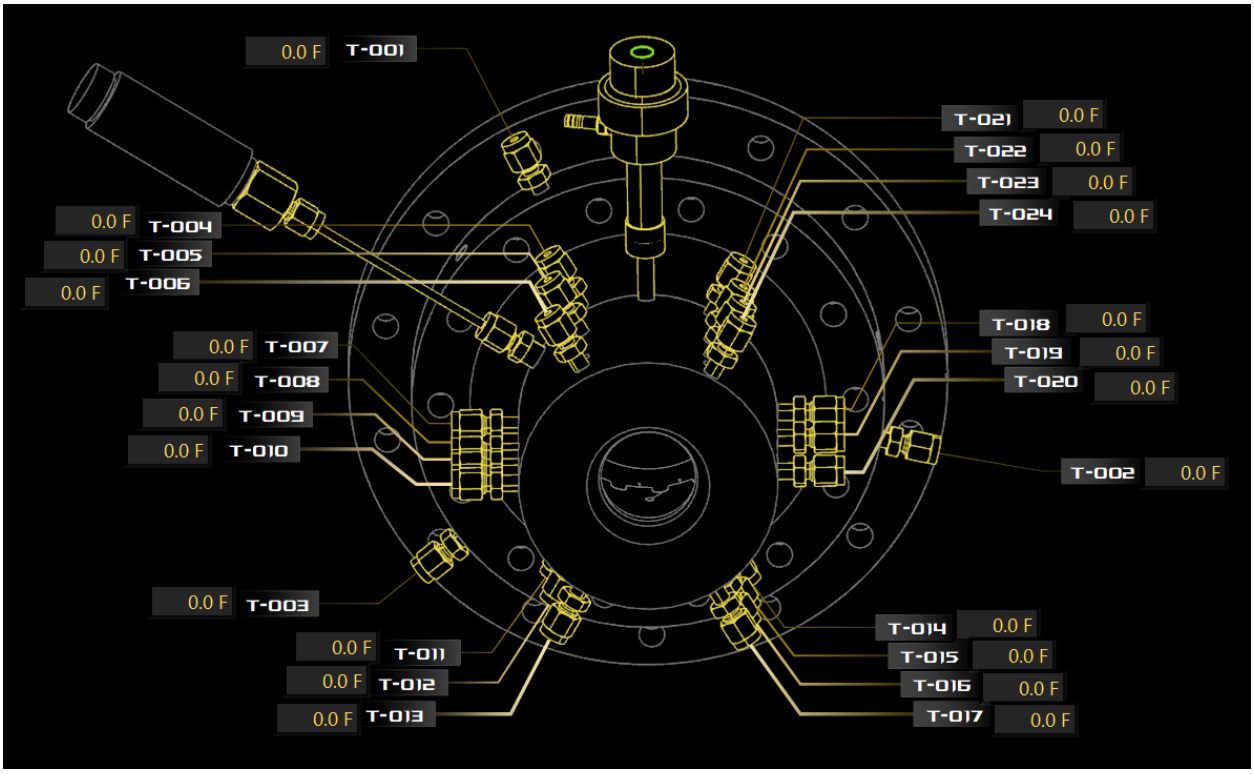


Figure 4.42 Engine display for CROME *TC numbering has changed since image*

Chapter 5: Future Work

Software development is an ongoing process, and with the constantly evolving state of both the tRIAC facility and the MICIT trailer, the code will regularly need to be updated. There are always ways to streamline the code here or hammer out some bugs there.

The CROME-X VI has not been written, but the existing VI for CROME should serve as an adequate template.

References

[1] Ott, D. (2018). THE DESIGN OF A ROCKET PROPULSION LABVIEW DATA ACQUISITION AND CONTROL SYSTEM. Master's. University of Texas at Austin.

[2] Chaparro, J. (2017). THE DEVELOPMENT OF A DATA ACQUISITION AND CONTROL SYSTEM FOR ROCKET PROPULSION RESEARCH. Master's. University of Texas at El Paso.

[3] Hansen, C. (2019). Initialization and Troubleshooting of the MICIT System for LO2/LCH4 Engine Testing.

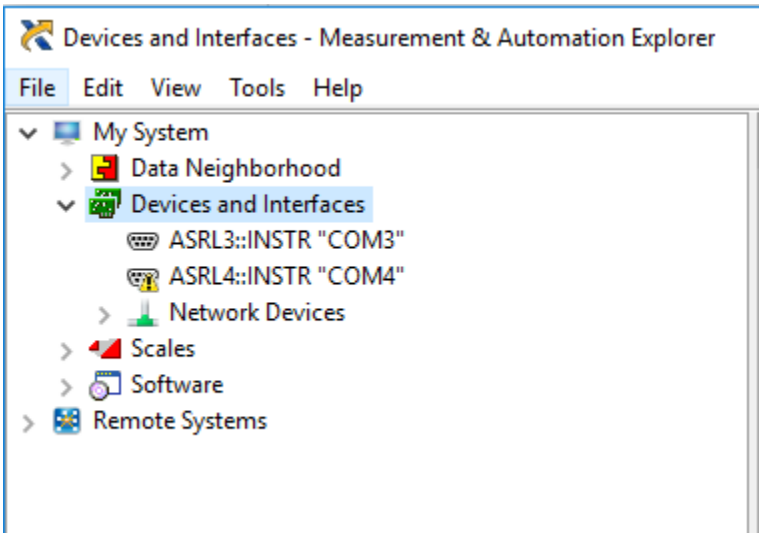
[4] National Instruments. 2018. "LabVIEW 2018."

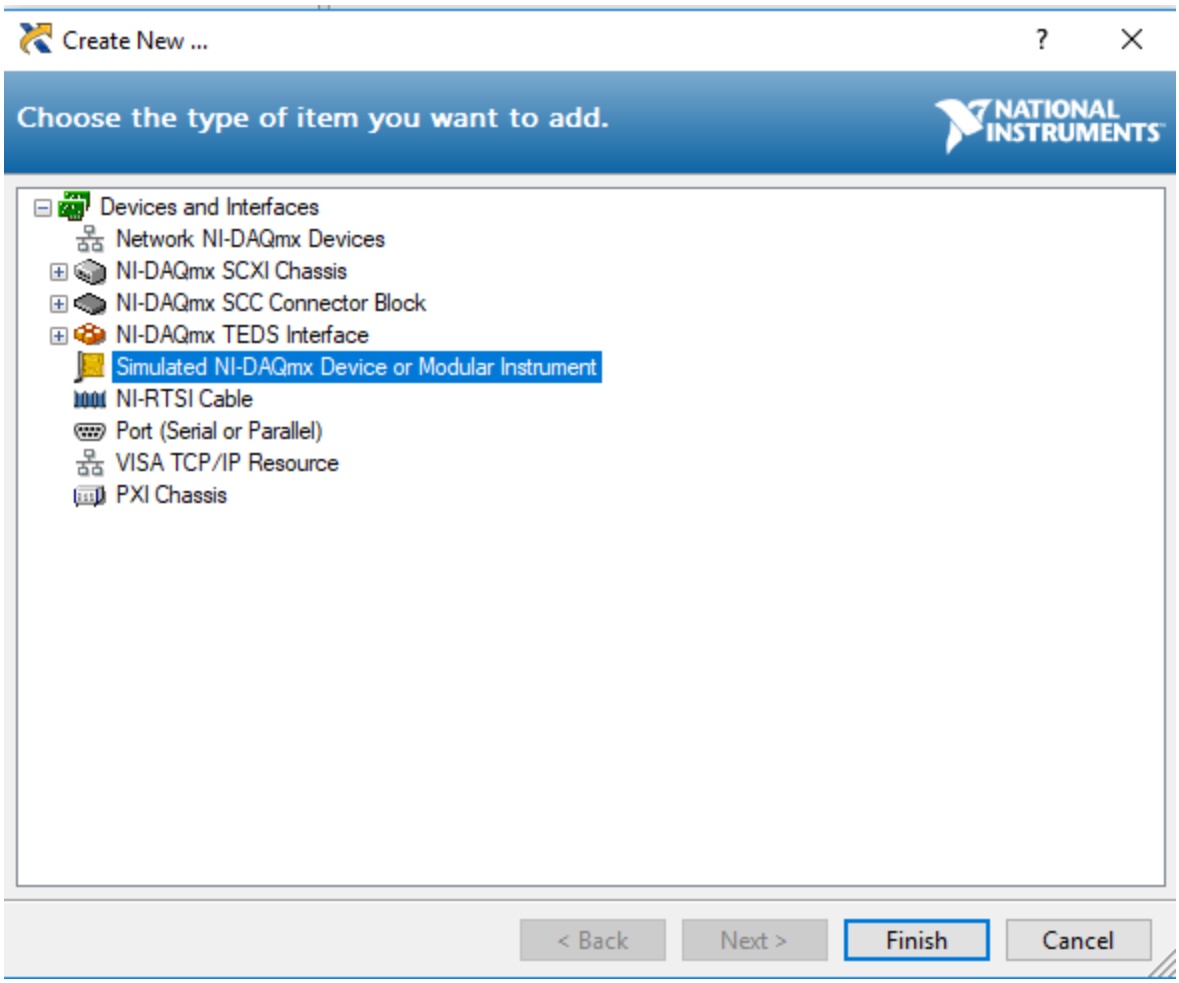
Appendix

A.1 Simulating the MICIT on a personal Computer

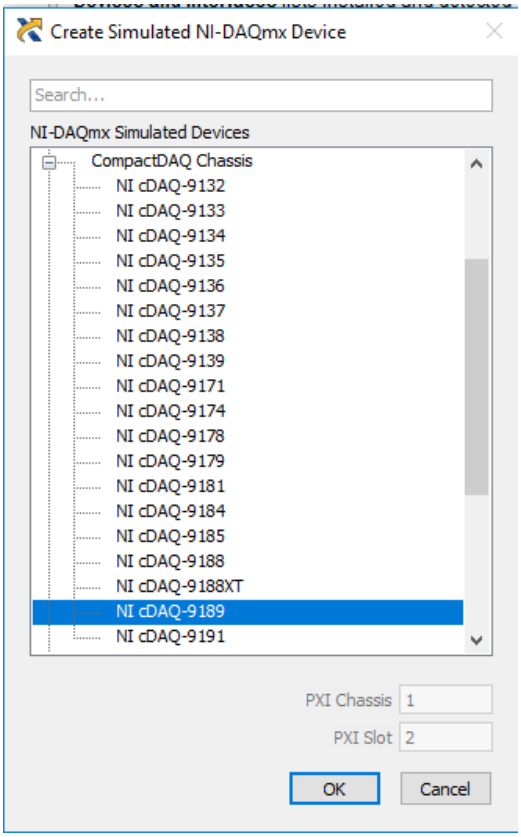
National Instruments has developed tools to enable anyone working with a DAQ or cRIO system to simulate the hardware. By doing so, you can program for an experiment while away from the tRIAC and before all of the necessary NI modules and sensors are available. One important thing to note is that simulating both the cDAQ and cRIO and all their modules is computationally intensive. This may result in your computer becoming noticeably more sluggish if you have an older system.

After installing all of the components in the section immediately before, open NI-MAX (Measurement and Automation Explorer). Left click on “Devices and Interfaces” and select “Create New…” from the menu.



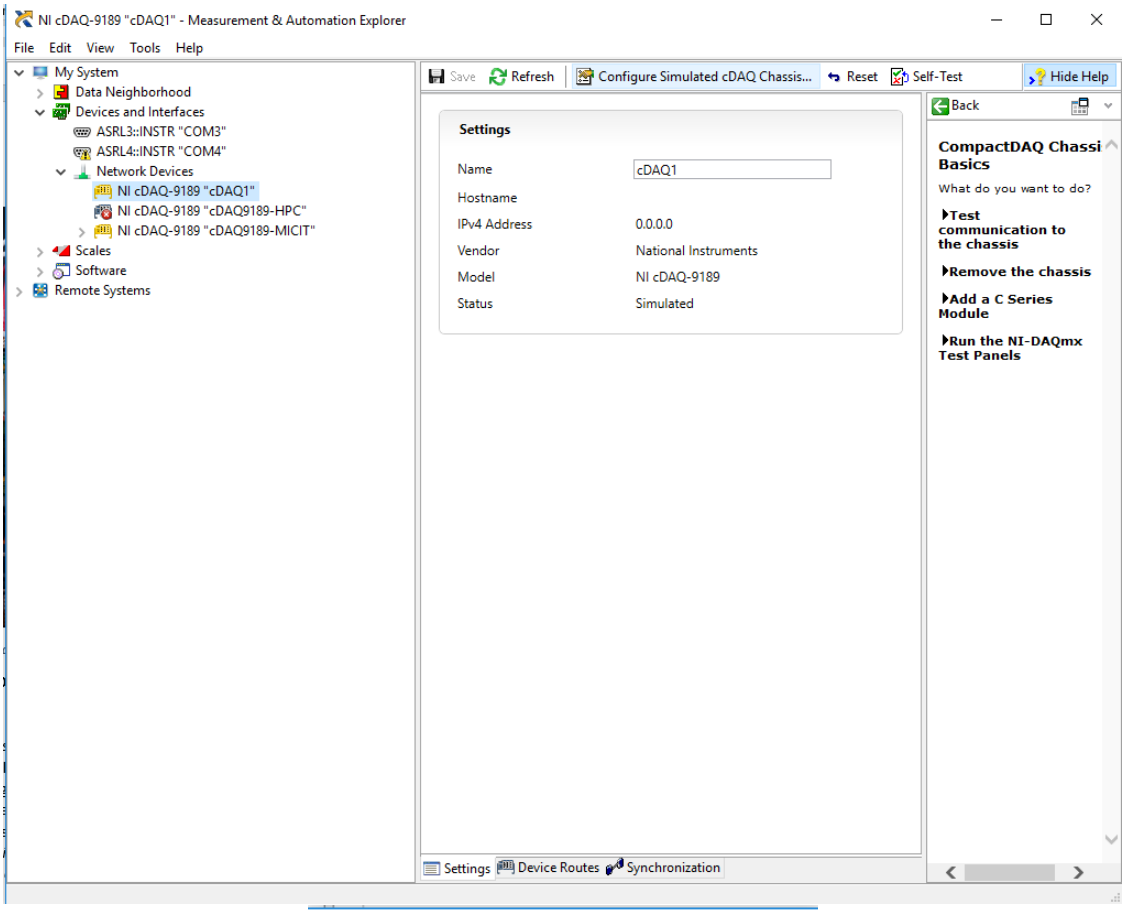


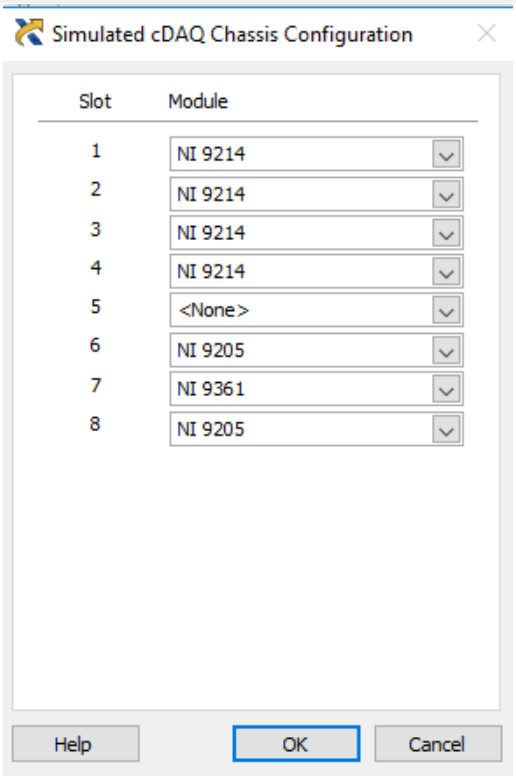
A menu will open asking for what new system to add. In this case, we are going to add a simulated NI-DAQmx device. The cDAQ in the MICIT is an NI-9189. It can be reached under the “CompactDAQ Chassis” subtree, or from typing in “9189” in the search bar above.

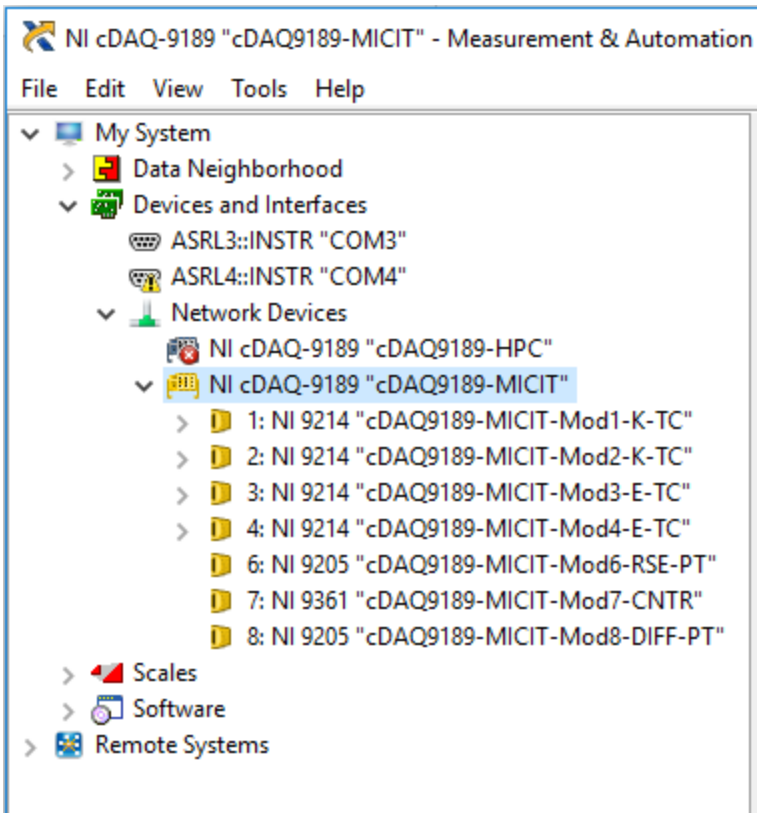


After hitting the "OK" button, a simulated DAQ device will appear under "Network devices" of the Devices and Interfaces menu in NI-MAX. Select it, and on the top middle of the NI-MAX window will be the option to "Configure Simulated cDAQ Chassis". From that menu, the modules installed in the cDAQ can be specified to match what the MICIT has.

Lastly, after setting up the simulated hardware, rename the cDAQ and its individual modules to match identically to the real ones installed in the MICIT as indicated in the following figures. While it is not necessary for the names between the real and simulated to perfectly match, it makes it a touch more convenient when copying the code from a personal computer to the computer at Fabens.

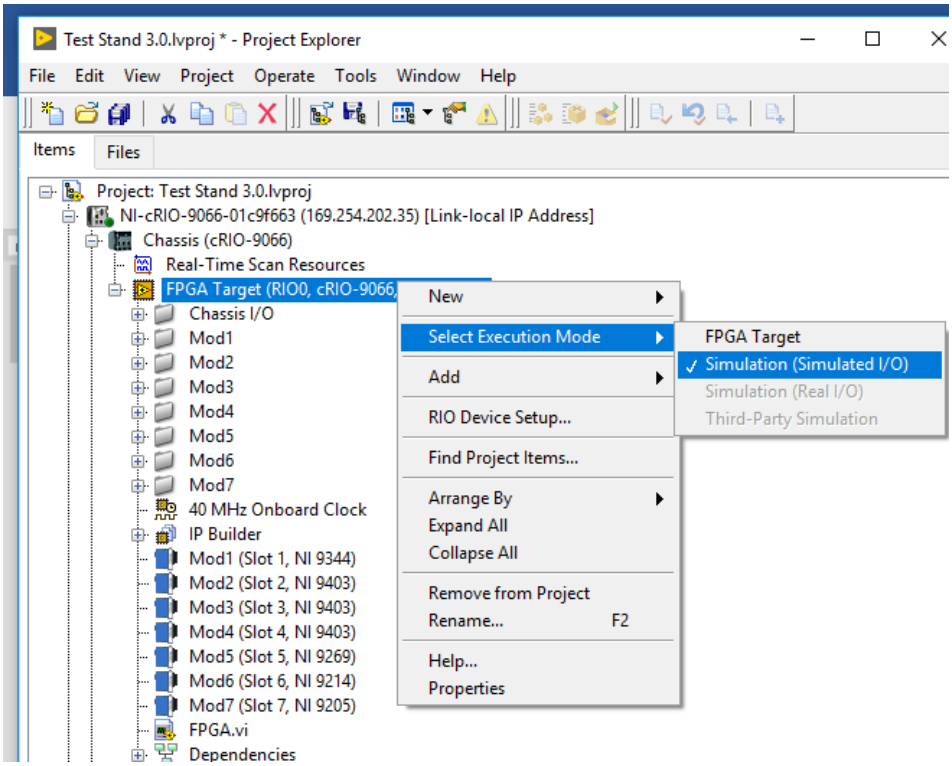

NI cDAQ-9189 "cDAQ1" - Measurement & Automation Explorer
File Edit View Tools Help
My System
Data Neighborhood
Devices and Interfaces
ASRL3::INSTR "COM3"
ASRL4::INSTR "COM4"
Network Devices
NI cDAQ-9189 "cDAQ1"
NI cDAQ-9189 "cDAQ9189-HPC"
NI cDAQ-9189 "cDAQ9189-MICIT"
Scales
Software
Remote Systems
Save
Refresh
Configure Simulated cDAQ Chassis...
Reset
Self-Test
Hide Help
Settings
Name
cDAQ1
Hostname
IPv4 Address
0.0.0.0
Vendor
National Instruments
Model
NI cDAQ-9189
Status
Simulated
Settings
Device Routes
Synchronization
Back
CompactDAQ Chassi Basics
What do you want to do?
Test communication to the chassis
Remove the chassis
Add a C Series Module
Run the NI-DAQmx Test Panels


Simulated cDAQ Chassis Configuration
Slot
Module
1
NI 9214
2
NI 9214
3
NI 9214
4
NI 9214
5
<None>
6
NI 9205
7
NI 9361
8
NI 9205
Help
OK
Cancel



This concludes how to simulate the cDAQ. Simulating the cRIO is done a little bit differently however as it is set up through the project file for the MICIT.

With the LabVIEW project file open, find the FPGA Target under the cRIO, and Chassis subtrees. Right click “FPGA” target and in the “Select Execution mode” menu item, select “Simulation (Simulated (I/O)”. Now, the cRIO will be simulated as well whenever any of the .vi files in the project communicates with the cRIO.



When it comes to simulating the cDAQ and all its modules, it simply produces a sine wave that moves up and down between its maximum and minimum expected readings. The frequency counter used in module 7 will not produce any kind of simulated signal however.

The cRIO mainly consists of Boolean signals for all of the valve states, and thus, they are simulated by individually turning off and on very quickly. To put it more simply, a simulated cRIO will look as if it is having a seizure as the Boolean states change rapidly and randomly.

Vita

Raymundo Rojo earned a Bachelors of Science in Physics on May 2013 followed by Master's of Science in Aerospace on December 2015, both from the University of Texas at Austin. His graduate work centered around rocket acoustics, but at the end it was not a field he was interested in pursuing. Unsure of where his passions lied, he enrolled in the University of Texas at El Paso to earn a second master's, this time in Mechanical Engineering and joined the cSETR with initial hopes to get into rocket design. This goal shifted as his work would center on the electronics and system control. Before graduating, he completed an internship at Johnson Space Center.

Along with his studies, Raymond privately tutored others in their engineering classes from introductory physics to thermodynamics. He applied this experience to also help his cSETR colleagues with LabVIEW and their own data acquisition systems.

Before graduation, Raymond accepted a job offer with Lockheed Martin as a Quality Assurance Engineer. However after exposure to NASA's rich history at the Johnson Space Center, Raymond aspires to return to the space industry.