INDUSTRIAL TRAINING REPORT

ON

PROJECT "LabView Programming"

AT

INTERNSHALA

Submitted in partial fulfillment of the requirements
for the award of degree of

Bachelor of technology

In

Electronics and Communication Engineering

Submitted By: - SHIRSHENDU CHATTERJEE

Institute of Radiophysics and Electronics

Kolkata, West Bengal-700009

Affiliated To

UNIVERSITY OF CALCUTTA

DECLARATION

I, **Shirshendu Chatterjee,** Student of Btech (ECE) declare that the project titled "**LabView Programming** " which is submitted by me to Institute of Radiophysics and Electronics, University of Calcutta.

Date: 18TH AUG 2021
SHIRSHENDU CHATTERJEE

ACKNOWLEDGEMENT

The successful completion of this project mark the beginning of an ever - going learning experience of converting ideas and concepts into real life, practical system. This project was a quite a learning experience for me at each and every step. At the same time it has given me confidence to work in professional setup. I feel the experience gained during the project will lead me to gain the bright prospect in the future. First of all I would like to give thanks to **Internshala** for giving me the opportunity to learn in this esteemed organization, which not only has increased our awareness about latest fields but also taught me the importance of team building. I will also like to my teachers at **IRPE** for taking keen interest in my project and giving valuable suggestions and helping me directly or indirectly to complete this project.

SHIRSHENDU CHATTERJEE

BTech(hons.)

T91/ECE/184068

ABSTRACT

National Instruments **LabVIEW** is a graphical programming language that has its roots in automation control and data acquisition. Its graphical representation, similar to a process flow diagram, was created to provide an intuitive programming environment for scientists and engineers. The language has matured over the last 20 years to become a general purpose programming environment. LabVIEW has several key features which make it a good choice in an automation environment. These include simple network communication, turnkey implementation of common communication protocols (RS232, GPIB, etc.), powerful toolsets for process control and data fitting, fast and easy user interface construction, and an efficient code execution environment. We discuss the merits of the language and provide an example application suite written in-house which is used in integrating and controlling automation platforms. In this report, consists of fundamental of data types, code structures, hardware implementation, design patterns and building applications of the Finite Data Acquisition software.

CONTENTS

INTRODUCTION

ORGANISATION PROFILE:

Internshala is an internship and online training platform, based in Gurgaon, India. Founded by Sarvesh Agrawal, an IIT Madras alumnus, in 2011, the website helps students find internships with organizations in India.

The platform, which was founded in 2010, started out as a WordPress blog that aggregated internships across India and articles on education, technology and skill gap. Internshala launched its online trainings in 2014. As of 2018, the platform had 3.5 million students and 80,000 companies. Internshala initially started out as a WordPress blog that aggregated internships and articles on education, technology, and skill gap across India and later converted into the platform. Agarwal had humble beginnings for two years, during which he was the only full-time employee of the company, along with the support of virtual interns working from different cities across India.

This is what Internshala offers-

- Filter internships according to location preferences
- Select the field of internship
- Increase oner chances of getting hired via Online Winter Internshala Training
- Apply and get selected
- Gain the certificate of getting hired from Internshala.

The Intershala platform started out intending to provide quality internship opportunities to its users but has slowly transformed to become an all-rounder career platform for students. Internshala now has over 5 million students registered on its platform, with thousands of companies on a constant run to find the perfect intern for their company. The one-stop-internship platform has also extended its wings towards providing the students on its platform with a chance to get specialization in the field of their choice through **Internshala Trainings**.

INTRODUCTION OF TRAINING

1. INTRODUCTION

The main objective of the training was to start from the very basic of how to start with LabView and to access its different tools to build different projects as per customer requirement. LabVIEW, which stands for Laboratory Virtual Instrumentation Engineering Workbench is a graphical programming language first released in 1986 by National Instruments (Austin, TX). LabVIEW implements a dataflow paradigm in which the code is not written, but rather drawn or represented graphically similar to a flowchart diagram. Program execution follows connector wires linking processing nodes together. Each function or routine is stored as a virtual instrument (VI) having three main components: the front panel which is essentially a form containing inputs and controls and can be displayed at run time, a block diagram where the code is edited and represented graphically, and a connector pane which serves as an interface to the VI when it is imbedded as a sub-VI. Figure 1a shows the front panel of an example VI and Figure 1b the block diagram or the code for the VI. VIs such as Mean.vi, Median Filter.vi, and Write to Spreadsheet File.vi are being used as sub-VIs. Unlike most programming languages, LabVIEW compiles code as it is created thereby providing immediate syntactic and semantic feedback and reducing the time required for development and testing.2 Writing code is as simple as dragging and dropping functions or VIs from a functions palette onto the block diagram within process structures (such as For Loops, or Case Structures) and wiring terminals (passing input values, or references). Unit testing is simplified because each function is separately encapsulated; input values can be set directly on the front panel without having to test the containing module or create a separate test harness. Memory allocation/deallocation is automatically managed by LabVIEW.3,4 The functions that generate data take care of managing the storage for the data. Associated memory is automatically deallocated if the data are no longer being used. In Figure 1b, the For Loop generates an array of random numbers with no explicit memory allocation step. The array is passed as input to each of the sub-VIs that manage memory independently. When passing data to a sub-VI, LabVIEW uses pass-by-value semantics with pass-by-reference implementation. There are also functions in LabVIEW that allow developers to manage memory explicitly.

NI LabVIEW supports multithreaded application design and executes code in an inherently parallel rather than sequential manner; as soon as a function or sub-VI receives all of its required inputs, it can begin execution. In Figure 1b, all the sub-VIs receive the array input simultaneously as soon as the For Loop is complete, and thus they execute in parallel. This is unique from a typical text-based environment where the control flows line by line within a function. When sequential execution is required, control flow can be enforced by use of structures such as Sequences, Events, or by chaining sub-VIs where output data from one VI is passed to the input of the next VI.

Similar to most programming languages, LabVIEW supports all common data types such as integers, floats, strings, and clusters (structures) and can readily interface with external libraries, ActiveX components, and .NET framework. As shown in Figure 1b, each data type is graphically represented by wires of different colors and thickness. LabVIEW also supports common configuration management applications such as Visual SourceSafe making multideveloper projects reasonable to manage. Applications may be compiled as executables or as Dynamic Link Libraries (DLLs) that execute using a run-time engine similar to the Java Runtime Environment. The development environment provides a variety of debugging tools such as break-points, probes (trace), and single-step. Applications can be developed using a variety of design patterns such as Client-Server, Consumer-Producer, and State-Machine. There are also UML (Unified Modeling Language)5 modeling tools that allow automated generation of code from UML diagrams and state diagrams.6 The GOOP toolkit7 (Endevo, Stockholm, Sweden) enables full object oriented programming support for LabVIEW, including inheritance.

Over the years, LabVIEW has matured into a general purpose programming language with a wider user base. There are open source communities such as the OpenG.org that promote and share code development for LabVIEW and user forums such as LAVA (LabVIEW Advanced Virtual Architects)8 with local chapters that organize seminars and training sessions on LabVIEW. The LabVIEW developer zone on the NI website is also a useful online resource to interact with the LabVIEW community.

2. SOFTWARE TOOLS

2.1 LabView

LabVIEW programs are called virtual instruments, or VIs, because their appearance and operation imitate physical instruments, such as oscilloscopes and multimeters. LabVIEW contains a comprehensive set of tools for acquiring, analyzing, displaying, and storing data, as well as tools to help one troubleshoot code one write. In LabVIEW, one builds a user interface, or front panel, with controls and indicators. Controls are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays. After one builds the front panel, one adds code using VIs and structures to control the front panel objects. The block diagram contains this code.

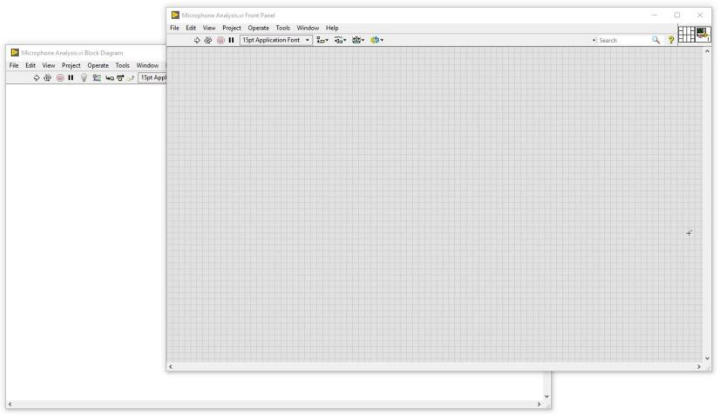
We first started with opening a new VI for a project of microphone analysis.

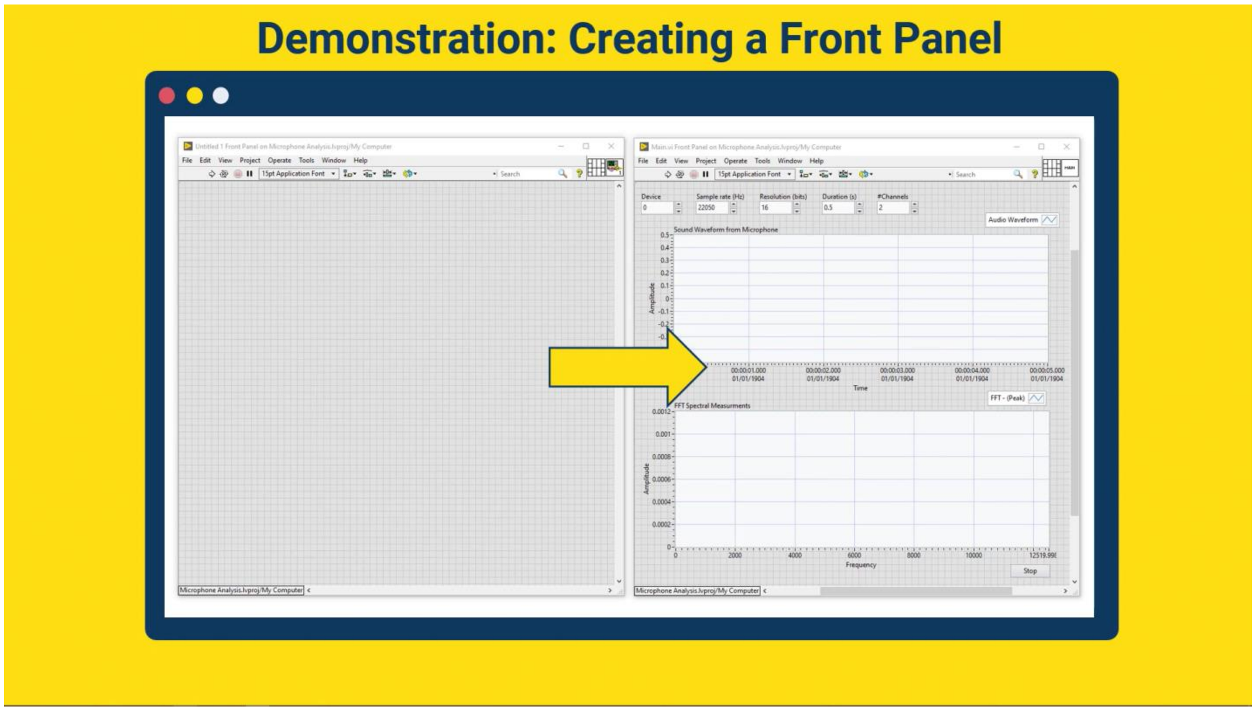
3.INDUSTRIAL TRAINING WORK UNDERTAKEN

3.1 Launching LabView

3.1.1Front Panel

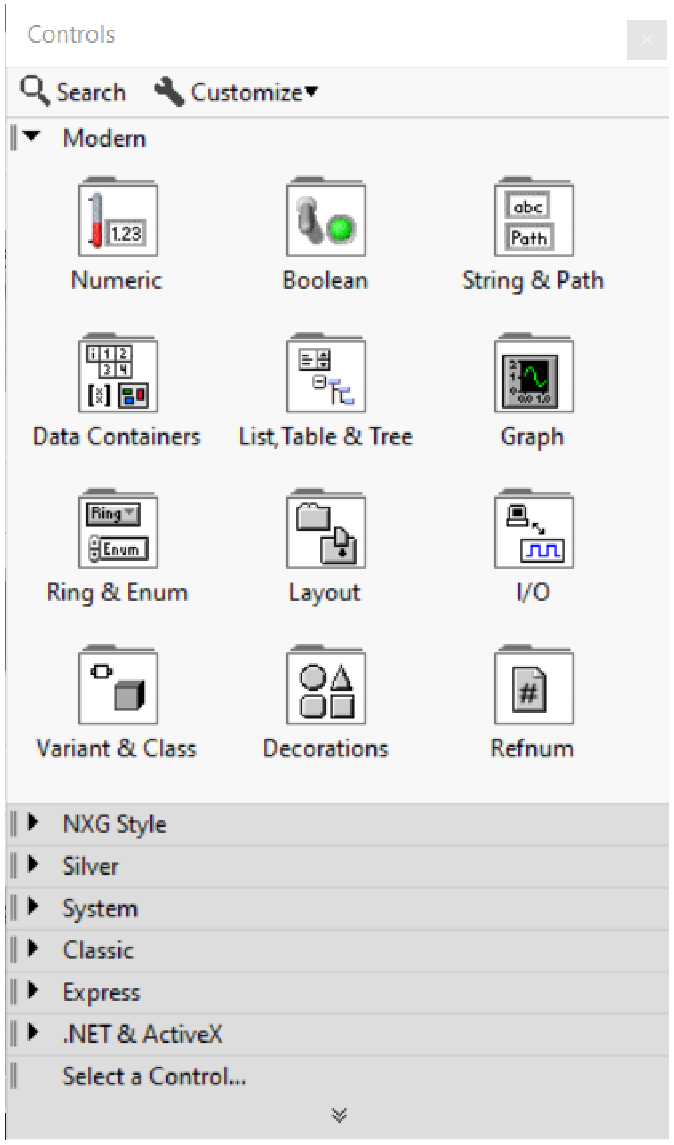
When one opens a new or existing VI, the front panel window of the VI appears. The front panel window is the user interface for the VI.


Microphone Analysis.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Microphone Analysis.vi Block Diagram
File Edit View Project Operate Tools Window



3.1.2 Controls Palette

The Controls palette contains the controls and indicators you use to create the front panel. You access the Controls palette from the front panel window by selecting View»Controls Palette or by right clicking on any empty space in the front panel window. The Controls palette is broken into various categories; you can expose some or all of these categories to suit your needs. **Figure** shows a Controls palette with all of the categories exposed and the Modern category expanded.



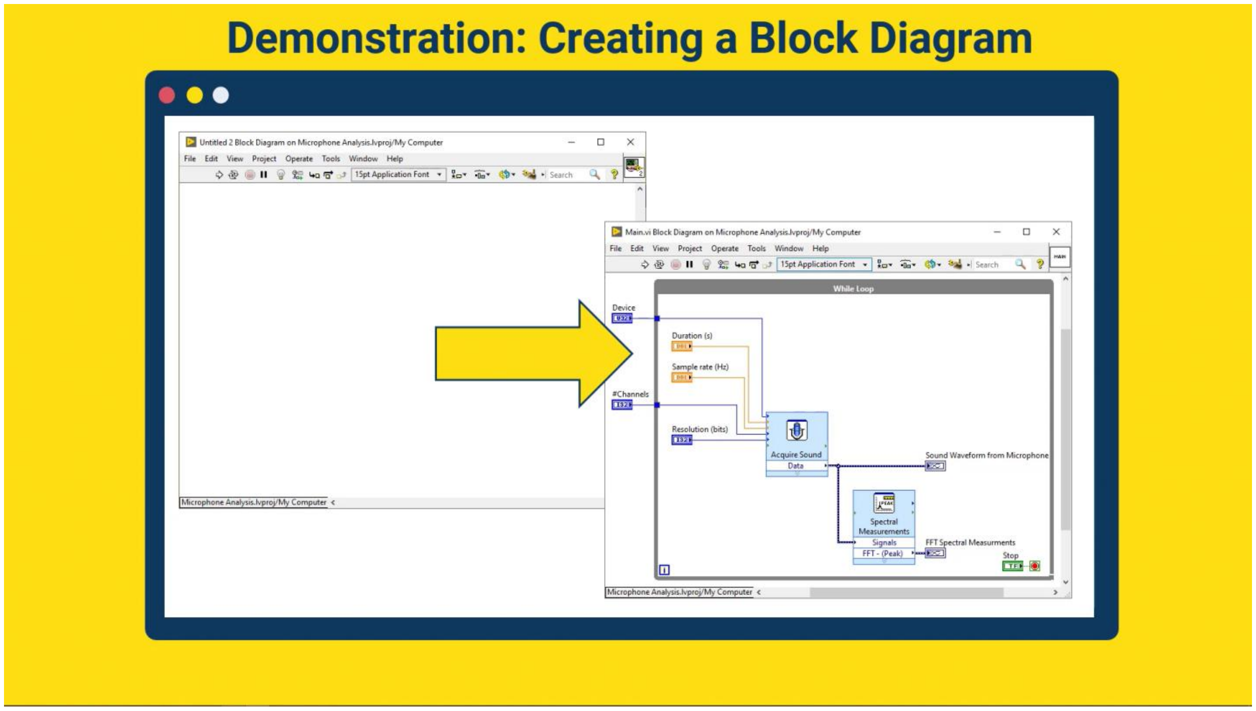
3.1.3 Controls and Indicators

Every VI has a front panel that you can design as a user interface. You also can use front panels as a way to pass inputs and receive outputs when you call the VI from another block diagram. You create the user interface of a VI by placing controls and indicators on the front panel of a VI. When you interact with a front panel as a user interface, you can modify controls to supply inputs and see the results in indicators. Controls define inputs, and indicators display outputs.
Controls are typically knobs, push buttons, dials, sliders, and strings. They simulate instrument input devices and supply data to the block diagram of the VI. Indicators are typically graphs,

charts, LEDs, and status strings. Indicators simulate instrument output devices and display data the block diagram acquires or generates.

3.2 Block Diagram

The block diagram contains the graphical source code, also known as G code or block diagram code, for how the VI runs. The block diagram code uses graphical representations of functions to control the front panel objects. Front panel objects appear as icon terminals on the block diagram. Wires connect control and indicator terminals to Express VIs, VIs, and functions. Data flows through the wires in the following ways: from controls to VIs and functions, from VIs and functions to indicators, and from VIs and functions to other VIs and functions. The movement of data through the nodes on the block diagram determines the execution order of the VIs and functions. This movement of data is known as dataflow programming


Demonstration: Creating a Block Diagram
Untitled 2 Block Diagram on Microphone Analysis.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Microphone Analysis.lvproj/My Computer
Main.vi Block Diagram on Microphone Analysis.lvproj/My Computer
While Loop
Device
Duration (s)
Sample rate (Hz)
#Channels
Resolution (bits)
Acquire Sound
Data
Sound Waveform from Microphone
Spectral Measurements
Signals
FFT - (Peak)
FFT Spectral Measurments
Stop

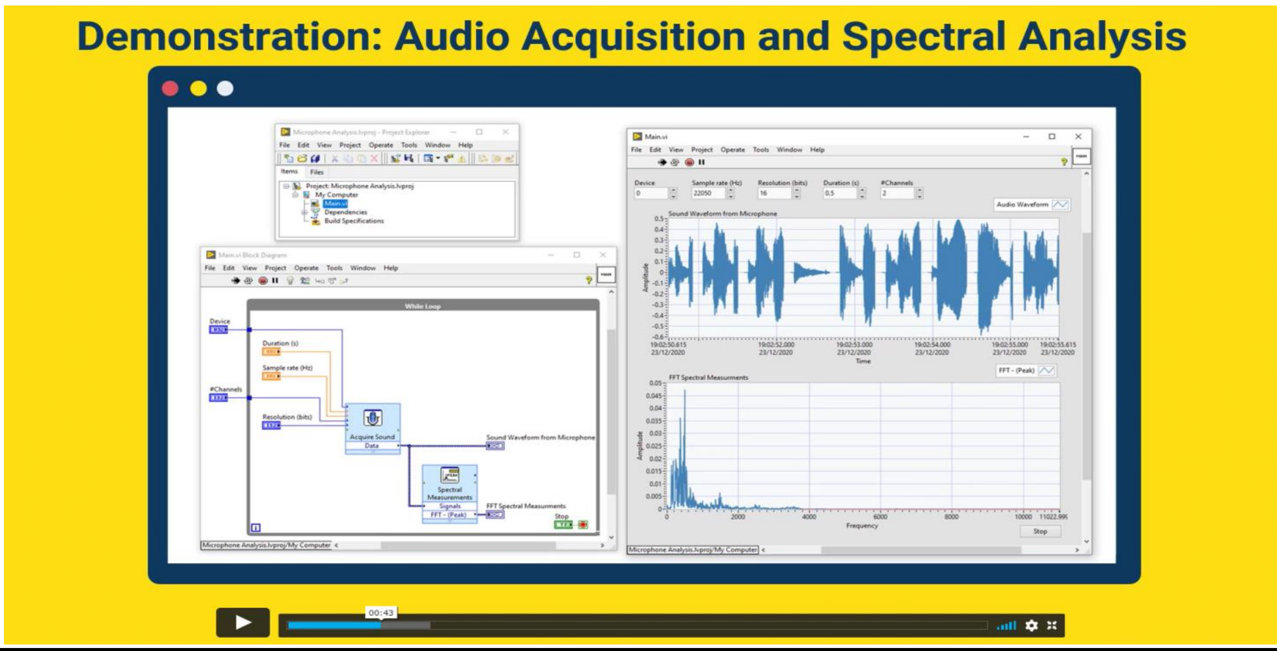
3.3 Front Panel and Block Diagram Tools:

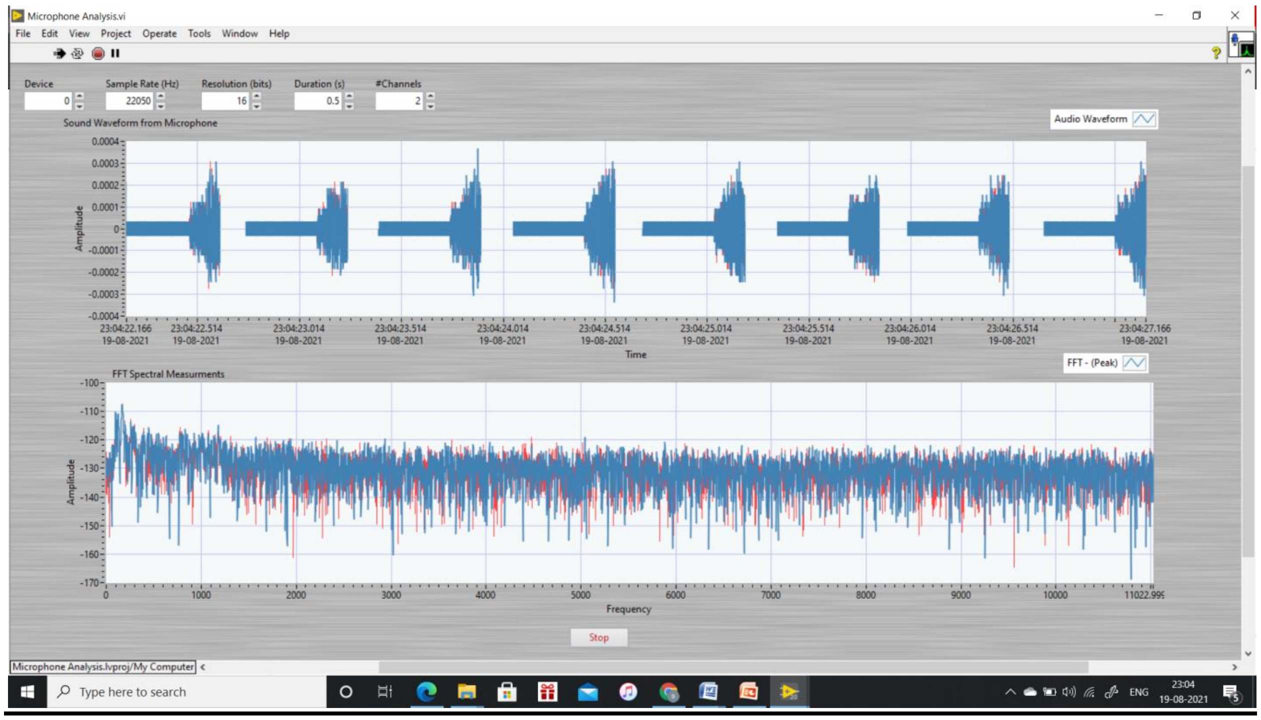
The Positioning tool appears when you move the cursor over an object in the front panel window or on the block diagram. The cursor becomes an arrow that you can use to select, position, and resize objects. The Wiring tool appears when you move the cursor over a terminal of a block diagram object. The cursor becomes a spool that you can use to connect objects on the block diagram through which you want data to flow.

Running and Stopping a VI:

Running a VI executes the solution of the VI. Click the Run button or press the keys to run a VI. The Run button changes to a darkened arrow to indicate the VI is running. You can stop a VI immediately by clicking the Abort Execution button. However, aborting a VI that uses © National Instruments | 1-19 Getting Started with LabVIEW external resources might leave the resources in an unknown state. Design the VIs you create with a stop button to avoid this problem. A stop button stops a VI after the VI completes its current iteration.

3.4 Microphone Analysis





Device 0 | Sample Rate (Hz) 22050 | Resolution (bits) 16 | Duration (s) 0.5 | #Channels 2

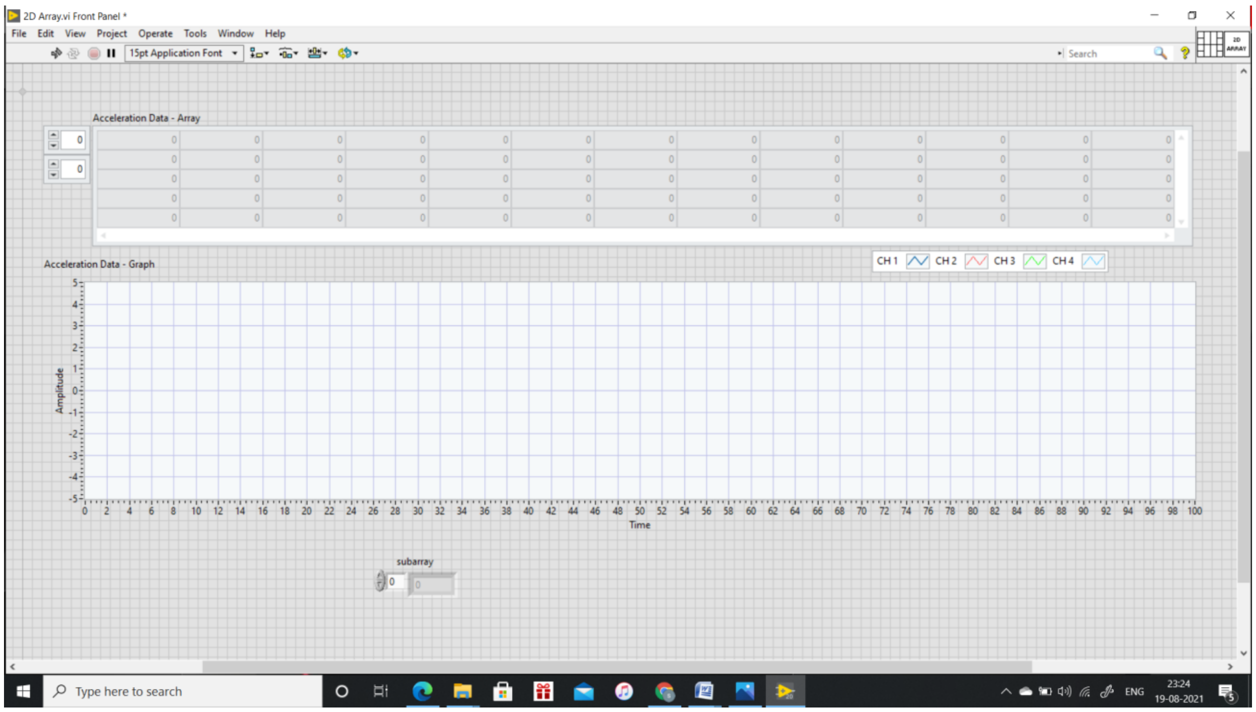
3.5 DATAFLOW AND DATATYPES

3.5.1 ARRAYS

Sometimes it is beneficial to group related data. Use arrays and clusters to group related data in LabView. Arrays combine data points of the same data type into one data structure, and clusters combine data points of multiple data types into one data structure.

An array consists of elements and dimensions. Elements are the data points that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as (2^{31})—1 elements per dimension, memory permitting.

You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types. Consider using arrays when you work with a collection of similar data points and when you perform repetitive computations. Arrays are ideal for storing data you collect from waveforms or data generated in loops, where each iteration of a loop produces one element of the array.

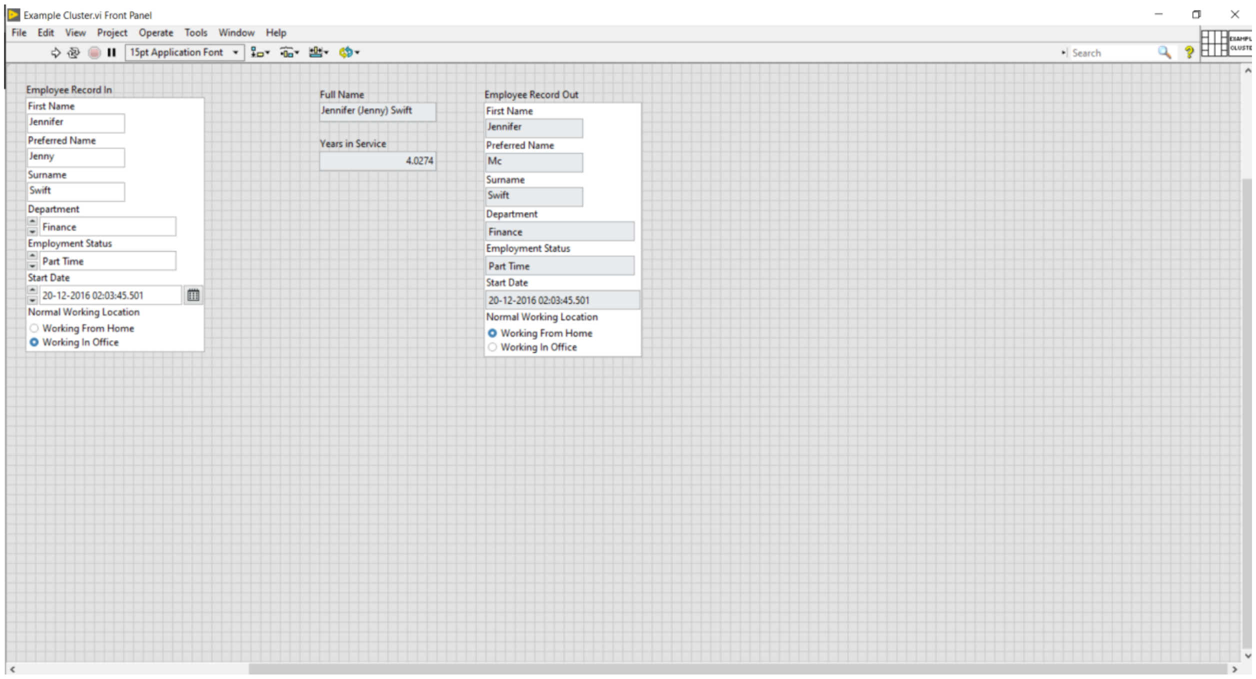

2D Array.vi Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
Acceleration Data - Array
Acceleration Data - Graph
CH 1
CH 2
CH 3
CH 4
Amplitude
Time
subarray
Type here to search

3.5.2 CLUSTERS

Clusters group data elements of mixed types. An example of a cluster is the LabVIEW error cluster, which combines a Boolean value, a numeric value, and a string. A cluster is similar to a record or a struct in text-based programming languages.

Bundling several data elements into clusters eliminates wire clutter on the block diagram and reduces the number of connector pane terminals that subVIs need. The connector pane has, at most, 28 terminals. If your front panel contains more than 28 controls and indicators that you want to pass to another VI, group some of them into a cluster and assign the cluster to a terminal on the connector pane.

Most clusters on the block diagram have a pink wire pattern and data type terminal. Error clusters have a dark yellow wire pattern and data type terminal. Clusters of numeric values, sometimes referred to as points, have a brown wire pattern and data type terminal. You can wire brown numeric clusters to Numeric functions, such as Add or Square Root, to perform the same operation simultaneously on all elements of the cluster.


Example Cluster.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Search

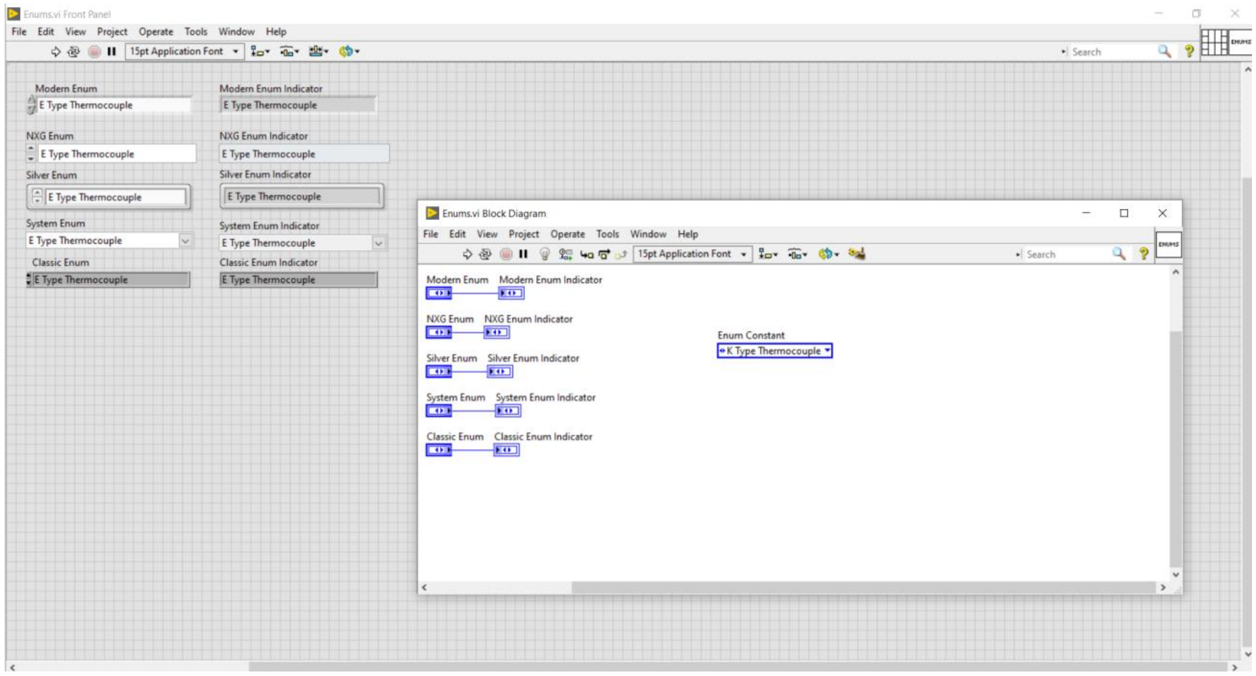
Employee Record In
First Name: Jennifer
Preferred Name: Jenny
Surname: Swift
Department: Finance
Employment Status: Part Time
Start Date: 20-12-2016 02:03:45.501
Normal Working Location: Working From Home / [✓ Working In Office]

Full Name: Jennifer (Jenny) Swift
Years in Service: 4.0274

Employee Record Out
First Name: Jennifer
Preferred Name: Mc
Surname: Swift
Department: Finance
Employment Status: Part Time
Start Date: 20-12-2016 02:03:45.501
Normal Working Location: [✓ Working From Home] / Working In Office

3.5.3 ENUMS

An enum (enumerated control, constant or indicator) is a combination of data types. An enum represents a pair of values, a string and a numeric, where the enum can be one of a list of values. For example, if you created an enum type called Month, the possible value pairs for a Month variable are January-0, February-1, and so on through December-11.



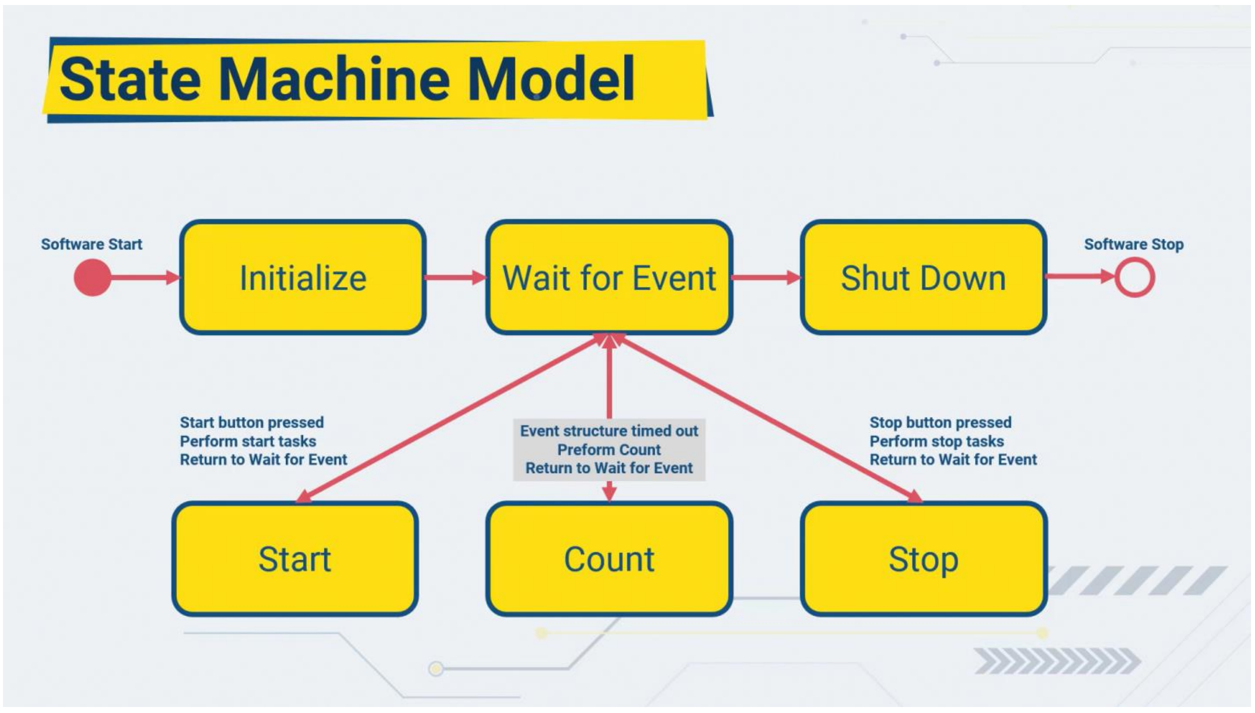
Modern Enum: E Type Thermocouple	Modern Enum Indicator: E Type Thermocouple
NXG Enum: E Type Thermocouple	NXG Enum Indicator: E Type Thermocouple
Silver Enum: E Type Thermocouple	Silver Enum Indicator: E Type Thermocouple
System Enum: E Type Thermocouple	System Enum Indicator: E Type Thermocouple
Classic Enum: E Type Thermocouple	Classic Enum Indicator: E Type Thermocouple

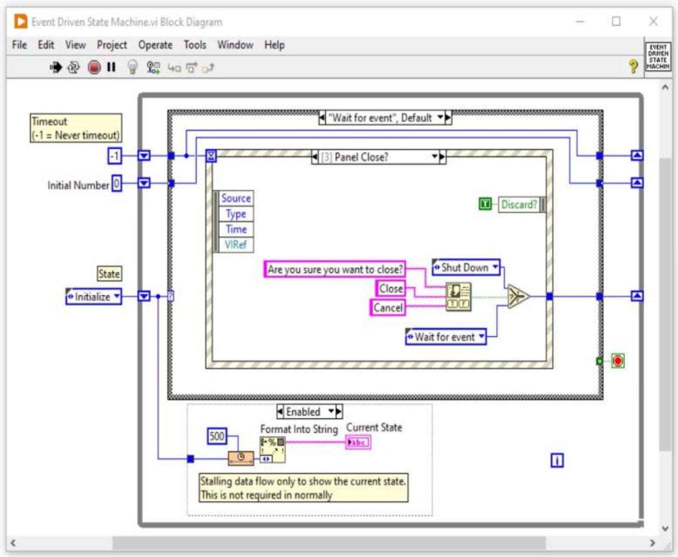
Enum Constant: K Type Thermocouple

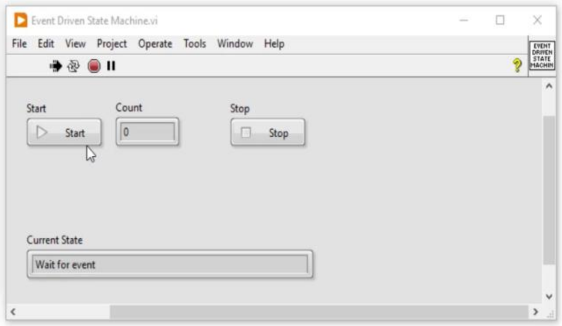
4. PROJECT WORK

4.1 EVENT DRIVEN STATE MACHINE

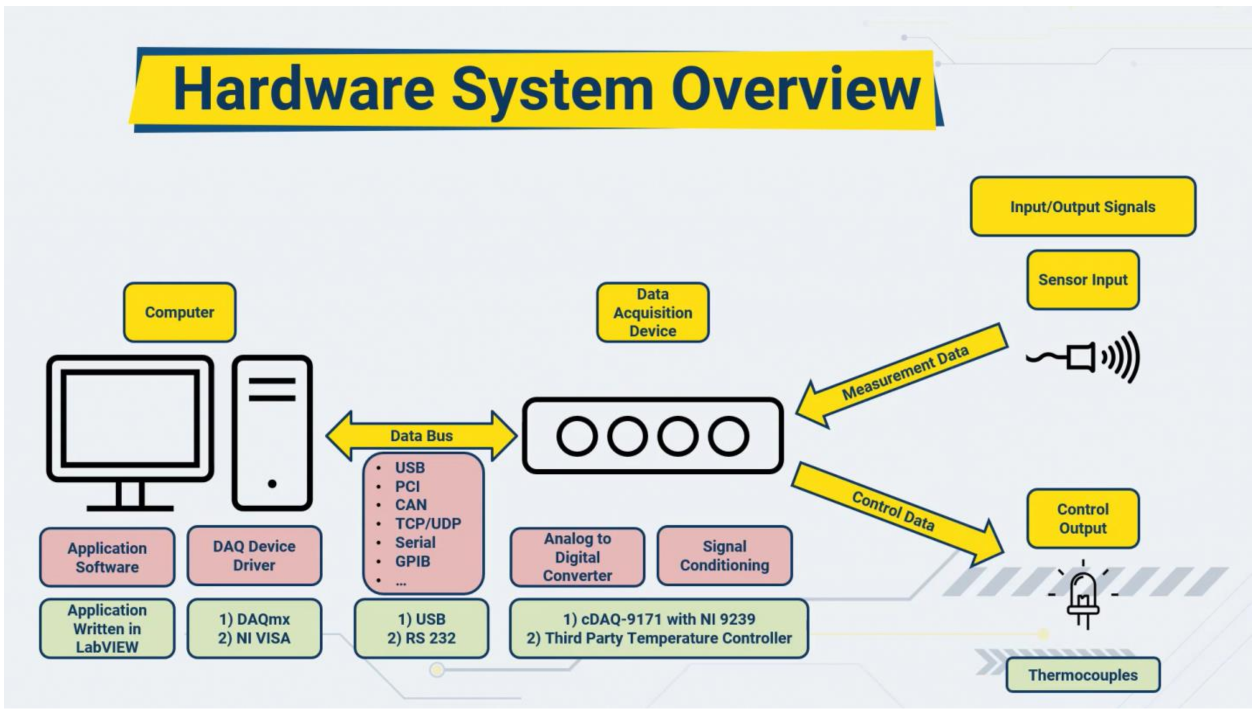
State machines revolve around 3 concepts: the state, the event, and the action. The state is the position or status that the program is at when it is working through the problem. For example, a state could be waiting for the user to do something, or running a calculation. States work to help break up the big picture and help to make everything run smoother. Developing these wisely will help to make the state machine run more efficiently. Events are occurrences that have specific meaning to the program itself. The example that we will be building is a vending machine that dispenses an item after the user has inserted the correct amount of money. An event for this program could be the money being inserted or the person hitting the start button. The action is how the program will react to the particular event that has occurred.




Event Driven State Machine.vi Block Diagram
File Edit View Project Operate Tools Window Help
Timeout (-1 = Never timeout)
-1
Initial Number
State
Initialize
"Wait for event", Default
[3] Panel Close?
Source
Type
Time
VIRef
Discard?
Are you sure you want to close?
Close
Cancel
Shut Down
Wait for event
Enabled
500
Format Into String
Current State
Stalling data flow only to show the current state. This is not required in normally


Event Driven State Machine.vi
File Edit View Project Operate Tools Window Help
Start
Count
Stop
Start
0
Stop
Current State
Wait for event

4.1.2 INTRODUCTION TO DATA ACQUISITION



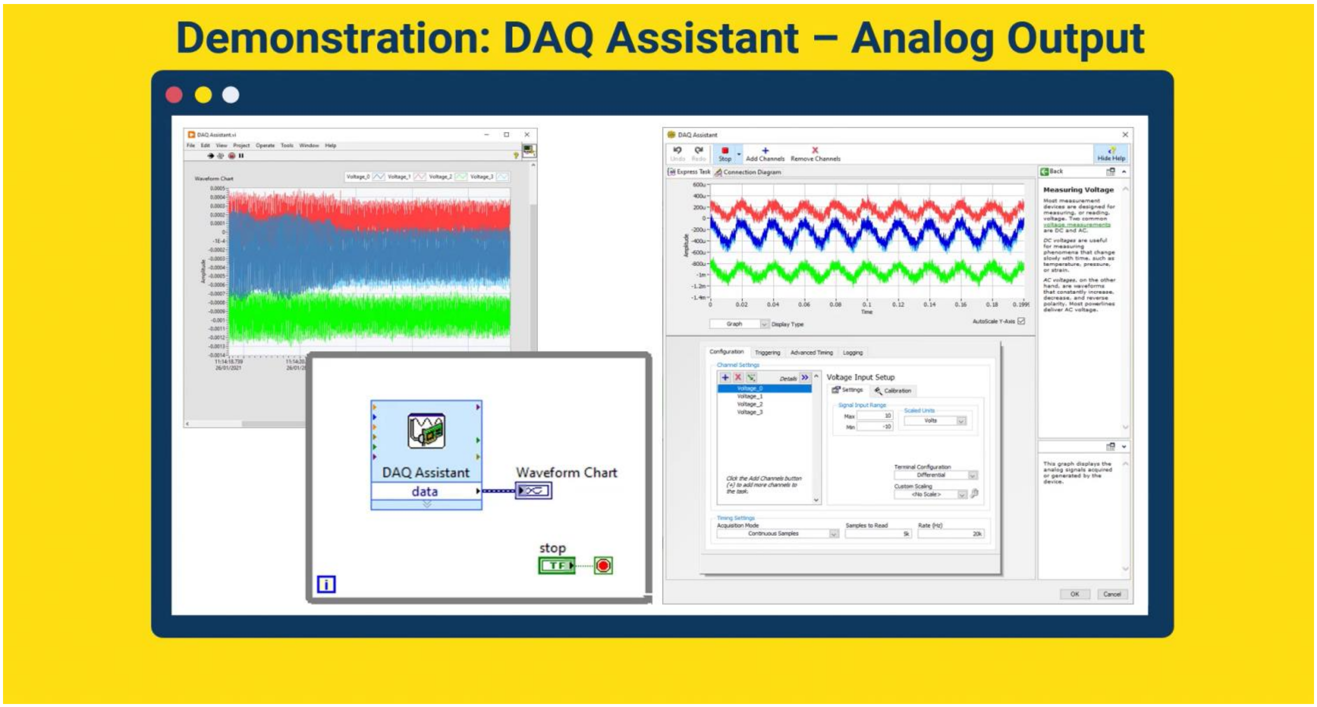
4.1.3 NI MAX

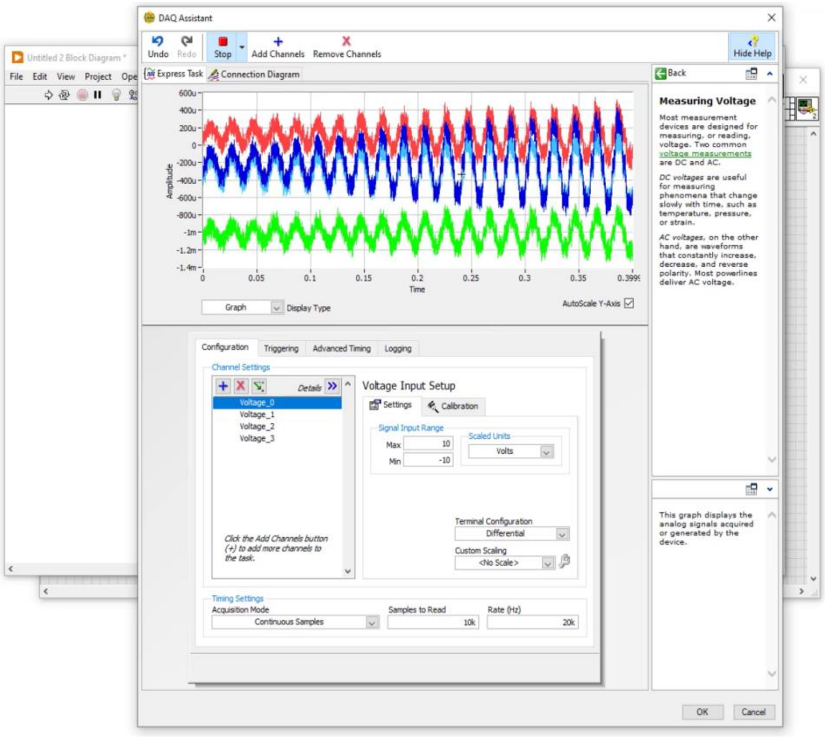
NI Measurement & Automation Explorer (MAX) provides access to your NI hardware. It is a free piece of software that cannot be downloaded by itself but is included, and automatically installed, with all NI drivers (NI-VISA, NI-DAQmx etc.) and NI System Configuration. For more information, please refer to Download NI Measurement & Automation Explorer (MAX).
With MAX, you can:

- Configure your NI hardware and software
- Export/Import the System Configuration
- Create and edit channels, tasks, interfaces, scales, and virtual instruments (ex. Create Tasks for NI-DAQmx Devices in NI MAX)
- Create Simulated Devices (ex. Create Simulated NI-DAQmx Devices in NI MAX)
- Execute system diagnostics and run test panels (ex. Using Test Panels in Measurement & Automation Explorer for Devices Supported by NI-DAQmx)
- View devices/instruments connected to your system and software installed on your system

4.1.4 USING NI DAQ Devices in LabVIEW

The very first multifunction DAQ driver was simply called NI-DAQ. This driver continued to version 6.9.3. Due to architecture changes, it was decided that NI-DAQ was to be replaced with NI-DAQmx as the future driver for our multifunction DAQ boards. This is where a split occurred in our drivers. NI-DAQ was split into DAQmx and Traditional NI-DAQ. DAQmx was built to support our new multifunction boards and Traditional NI-DAQ was created to continue to support customers that had developed applications on NI-DAQ, See below for more details on Traditional NI-DAQ and DAQmx.


Demonstration: DAQ Assistant – Analog Output



Measuring Voltage

Most measurement devices are designed for measuring, or reading, voltage. Two common voltage measurements are DC and AC.

DC voltages are useful for measuring phenomena that change slowly with time, such as temperature, pressure, or strain.

AC voltages, on the other hand, are waveforms that constantly increase, decrease, and reverse polarity. Most powerlines deliver AC voltage.

This graph displays the analog signals acquired or generated by the device.

Channel Settings: Voltage_0, Voltage_1, Voltage_2, Voltage_3

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

Voltage Input Setup

Signal Input Range: Max 10, Min -10

Scaled Units: Volts

Terminal Configuration: Differential

Custom Scaling: <No Scale>

Timing Settings

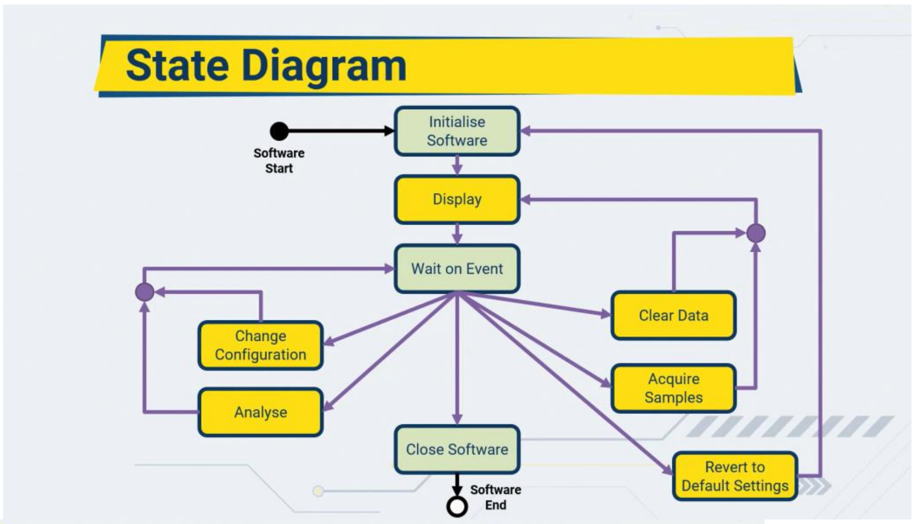
Acquisition Mode: Continuous Samples

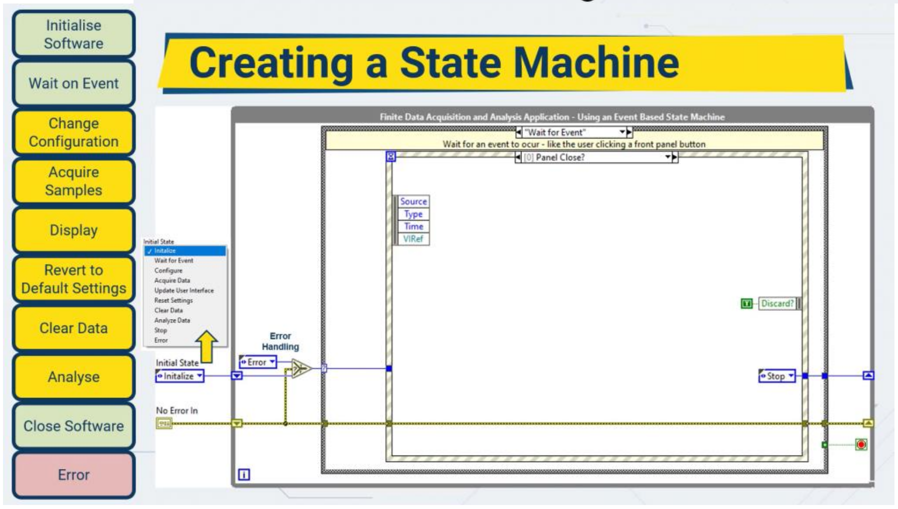
Samples to Read: 10k

Rate (Hz): 20k

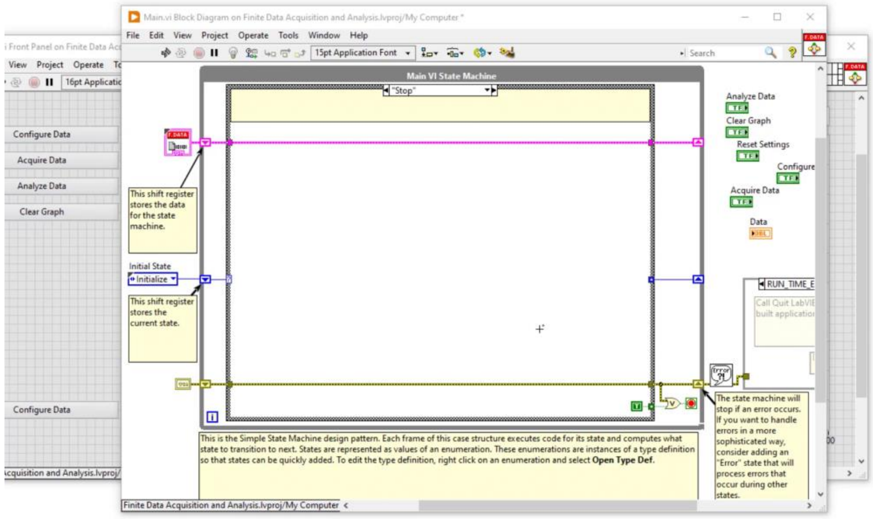
<u>4.2 EVENT BASED STATE MACHINE</u>

The implementation of event based state machine starts with the requirement of the customer and a state diagram is drawn to sequentially implement the requirements in labview. Then the whole state diagram is sequentially implemented. For eg. In the given problem the requirements are shown in the diagram.





The following figures shows the block diagram of the requirements in the problem of Finite Data Acquisition.

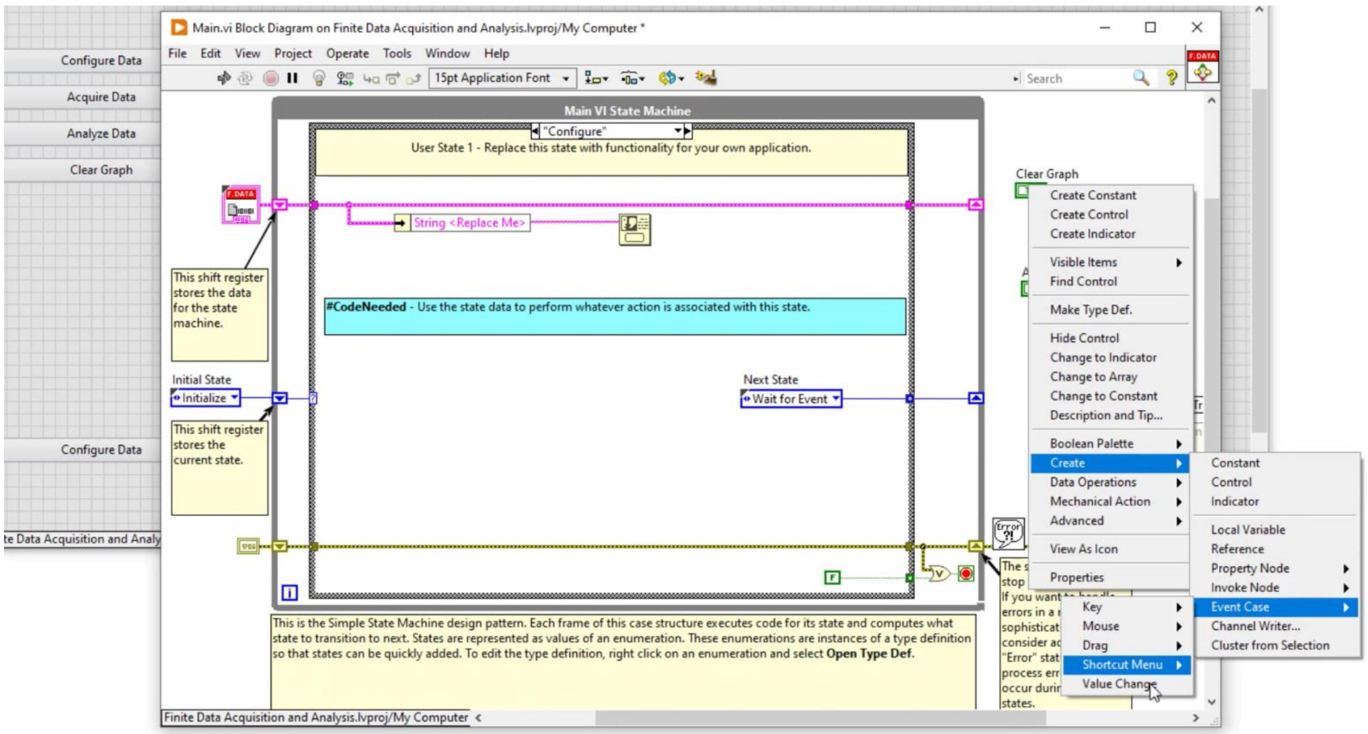


This shift register stores the data for the state machine.

This shift register stores the current state.

The state machine will stop if an error occurs. If you want to handle errors in a more sophisticated way, consider adding an "Error" state that will process errors that occur during other states.

This is the Simple State Machine design pattern. Each frame of this case structure executes code for its state and computes what state to transition to next. States are represented as values of an enumeration. These enumerations are instances of a type definition so that states can be quickly added. To edit the type definition, right click on an enumeration and select **Open Type Def.**



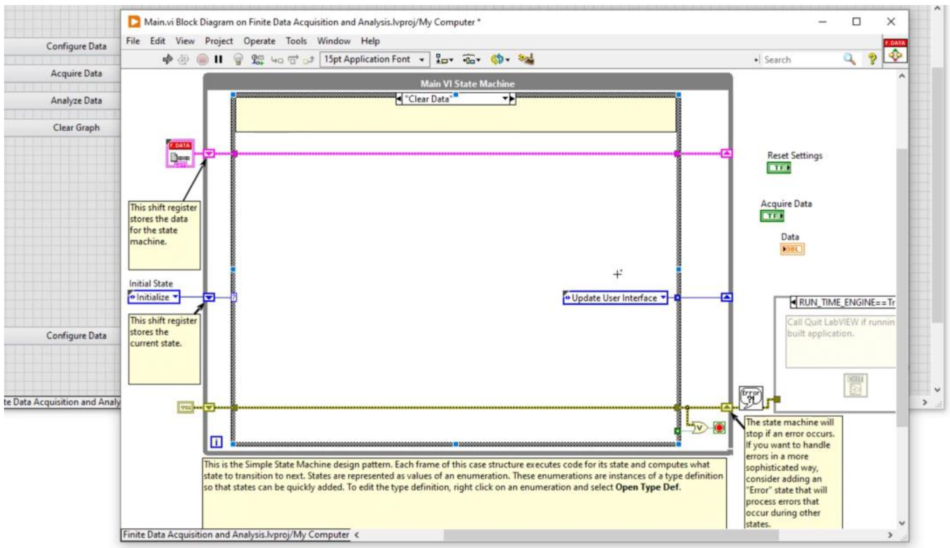
User State 1 - Replace this state with functionality for your own application.

#CodeNeeded - Use the state data to perform whatever action is associated with this state.

This shift register stores the data for the state machine.

This shift register stores the current state.

This is the Simple State Machine design pattern. Each frame of this case structure executes code for its state and computes what state to transition to next. States are represented as values of an enumeration. These enumerations are instances of a type definition so that states can be quickly added. To edit the type definition, right click on an enumeration and select **Open Type Def.**



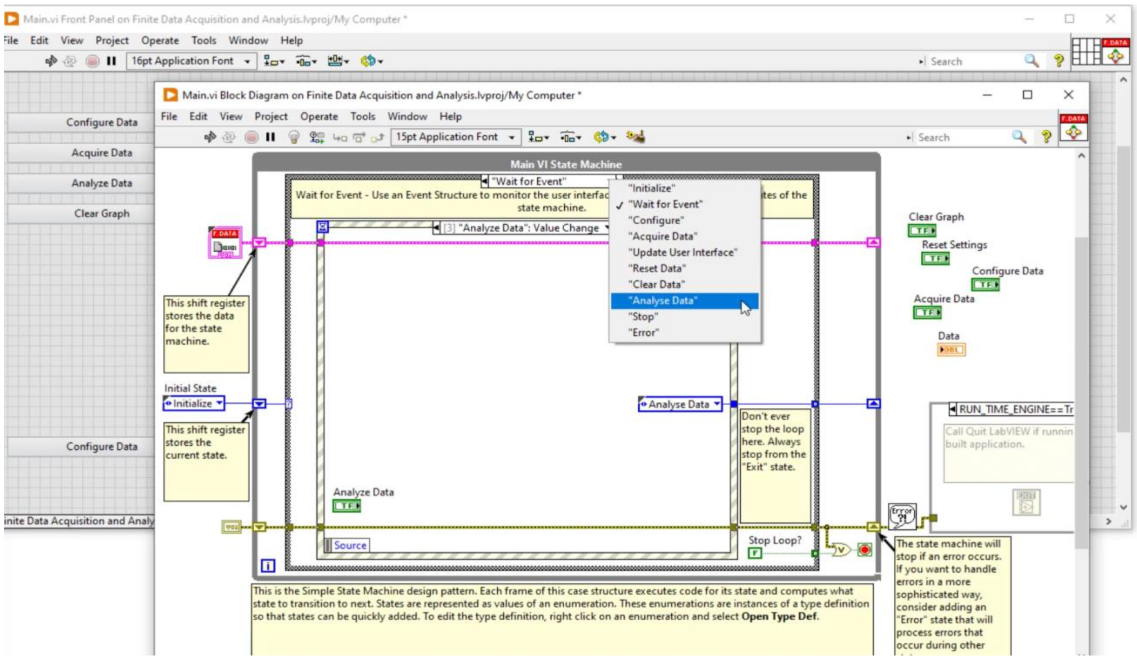
This shift register stores the data for the state machine.

This shift register stores the current state.

Call Quit LabVIEW if runnin built application.

The state machine will stop if an error occurs. If you want to handle errors in a more sophisticated way, consider adding an "Error" state that will process errors that occur during other states.

This is the Simple State Machine design pattern. Each frame of this case structure executes code for its state and computes what state to transition to next. States are represented as values of an enumeration. These enumerations are instances of a type definition so that states can be quickly added. To edit the type definition, right click on an enumeration and select **Open Type Def.**



Wait for Event - Use an Event Structure to monitor the user interfac tes of the state machine.

This shift register stores the data for the state machine.

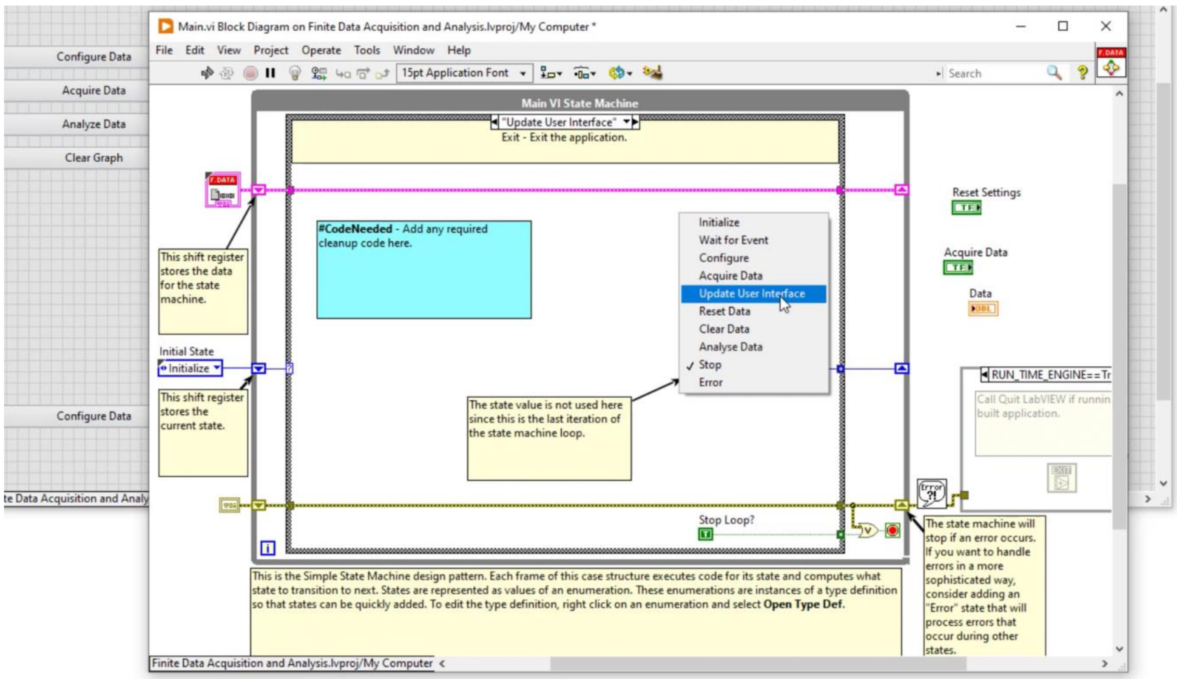
This shift register stores the current state.

Don't ever stop the loop here. Always stop from the "Exit" state.

Call Quit LabVIEW if runnin built application.

The state machine will stop if an error occurs. If you want to handle errors in a more sophisticated way, consider adding an "Error" state that will process errors that occur during other

This is the Simple State Machine design pattern. Each frame of this case structure executes code for its state and computes what state to transition to next. States are represented as values of an enumeration. These enumerations are instances of a type definition so that states can be quickly added. To edit the type definition, right click on an enumeration and select **Open Type Def.**



Exit - Exit the application.

This shift register stores the data for the state machine.

This shift register stores the current state.

#CodeNeeded - Add any required cleanup code here.

The state value is not used here since this is the last iteration of the state machine loop.

Call Quit LabVIEW if runnin built application.

This is the Simple State Machine design pattern. Each frame of this case structure executes code for its state and computes what state to transition to next. States are represented as values of an enumeration. These enumerations are instances of a type definition so that states can be quickly added. To edit the type definition, right click on an enumeration and select **Open Type Def.**

The state machine will stop if an error occurs. If you want to handle errors in a more sophisticated way, consider adding an "Error" state that will process errors that occur during other states.

4.3 VI Server Essentials.

4.3.1 Control Property Nodes-

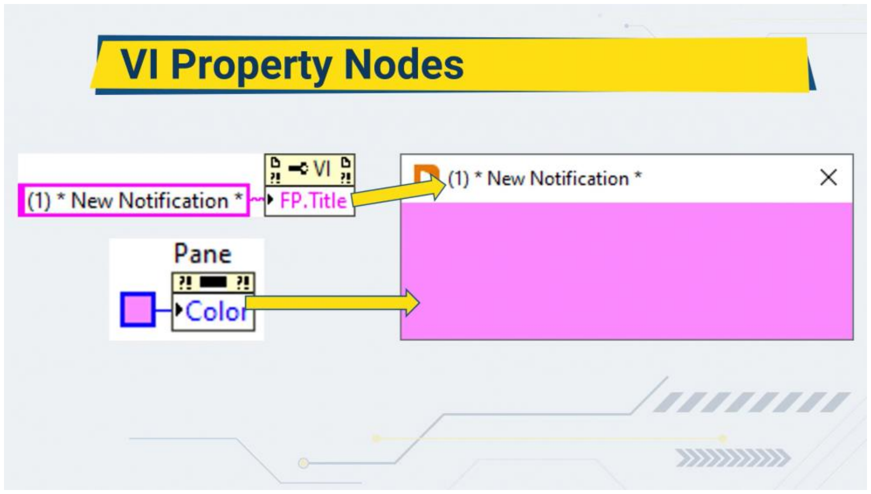
With property nodes, you can start making your LabVIEW program more powerful and a lot more fun. Property nodes allow you to programmatically control the properties of a front panel object: things such as color, visibility, position, numeric display format, and so on. The key word here, programmatically, is changing the properties of a front panel object according to an algorithm in your diagram. For example, you could change the color of a thermometer indicator from green to red as its numerical value increases. These types of changes can help create a more intuitive user experience.

To create a property node, pop up on either the front panel object or its terminal and select a property from the **Create>>Property Node** submenu. A terminal with the same name as the variable will appear on the diagram. To see what options you can set in a control's property node, click on the node with the Operating tool or pop up on the node and choose property. Now you have the choice of which property or properties you wish to select. Each object has a set of base properties (common to all types of controls), and usually, an additional set of properties specific to that type of control.

Just as with local variables, you can either read or write the property of an object (although a few properties are read-only). To change the mode of a property, pop up on it and select the Change to Write/Read option. The small arrow inside the property node's terminal tells you which mode it's in. A property node in write mode has the arrow on the left, indicating the data is flowing into the node, writing a new property. A property node in read mode has the arrow on the right, reading the current property and providing this data. The same analogy we used for locals, a control (read mode) and an indicator (write mode), holds for property nodes.

An interesting feature of property nodes is that you can use one terminal on the block diagrams for several properties (but always affecting the same control or indicator). To add an additional property, you can use the Positioning tool to resize the terminal and get the number of properties you need. Often you will want to use more than one option in an object's property node. Remember, instead of creating another property node, you can select several options at a time by enlarging the terminal with the Positioning tool (much like you enlarge cluster and array terminals). You will see each new option appear in sequence; you can later change these if you like by clicking on any item with the Operating tool.

Almost all controls or indicators have the base properties. Most of them have more, especially tables and graphs (which can have over 100 properties).



4.3.2 Invoke Nodes-

Invoke nodes are very similar to property nodes. Calling an invoke node runs a single method on the front panel object and sometimes requires inputs, (also known as "arguments"). In a way, calling an invoke node is similar to calling a subVI or calling a function or method in other languages.

The difference between a property node and an invoke node is that calling an invoke node, "executes something"it doesn't just change the value of a property. You can think of invoke nodes as functions that you execute; whereas a property node is a property, or state, that can be read from or written to.

To create an invoke node, pop up on either the front panel object or its terminal, and select one of the control's methods from the Create>>Invoke node>> short-cut menu. A terminal with the same name as the variable will appear on the diagram. The method name will appear as the first item in the invoke node, and all the arguments will appear as items beneath the method name. You can change the method being called by the invoke node with the Operating tool or pop up on the node and choose Methods>>. Now you have the choice of which method you wish to select. Each object has a set of base methods, and usually, an additional set of methods specific to that object. One base method that is available for all controls is Reinitialize to Default (see Figure 13.40). It doesn't have any arguments, and calling it will do just what it saysreinitialize the control to its default value.

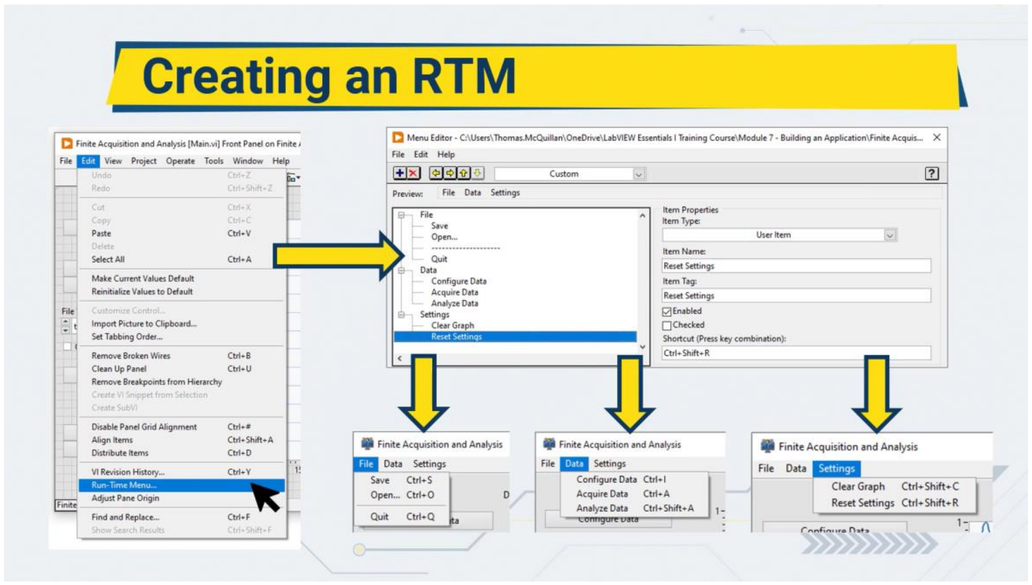
4.4 Building an Application

Objectives-

- To convert the software into a standalone application with installer.
- To make keyboard shortcuts for all functionality.
- To add menu items for all functionality.

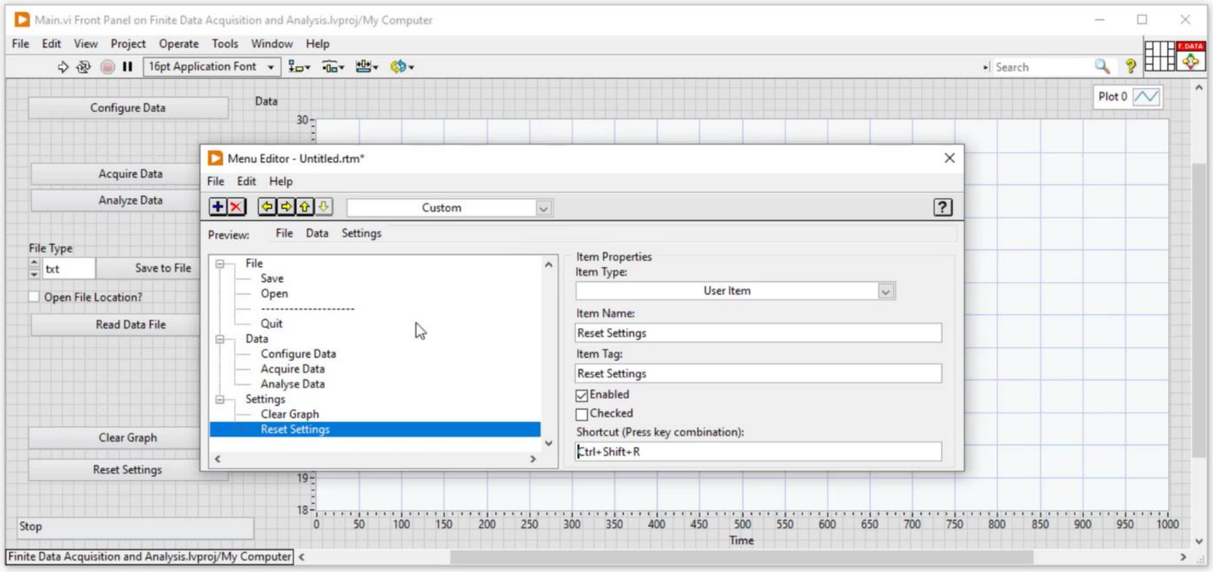
4.4.1 Custom Run Time Menu

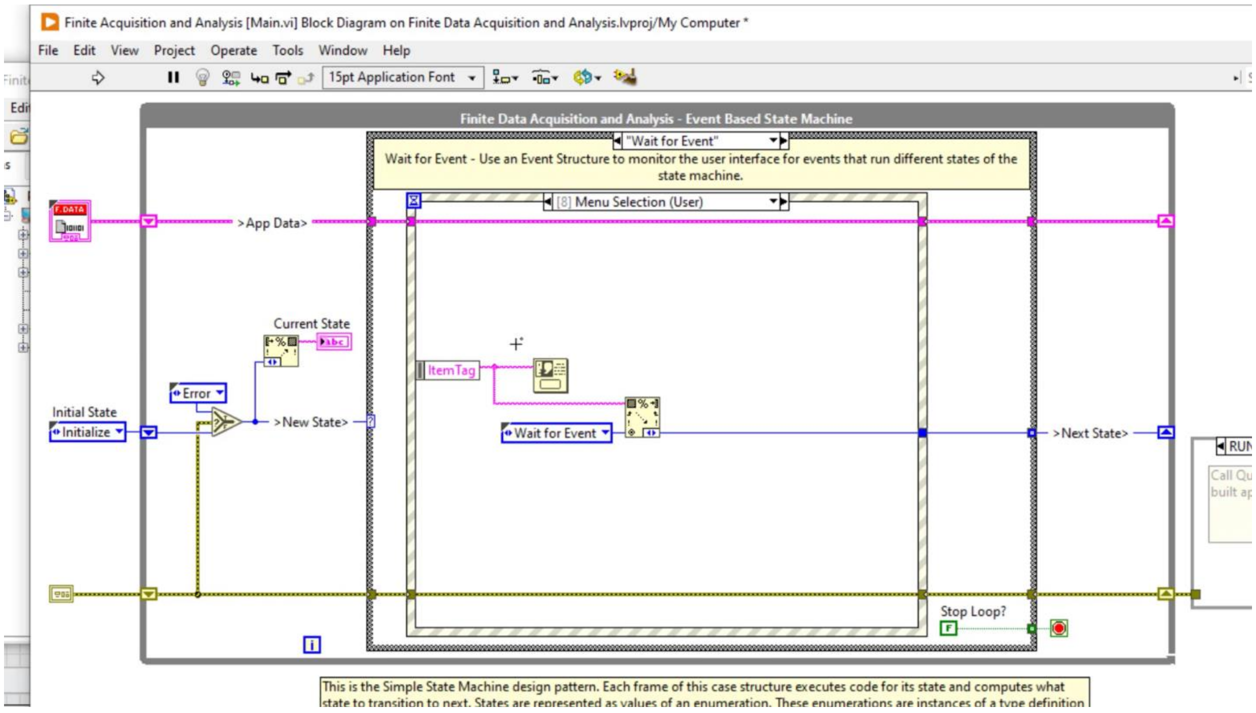
While LabVIEW provides hundreds of front panel controls for developing a professional user interface, there are often situations in which you need to customize the behavior of these controls. With LabVIEW 8 and later, you can create your own run-time shortcut menus for front panel objects. This gives users more control over the behavior of your application and creates a more professional interface for your users. In this tutorial, we will describe the process of creating a custom run-time shortcut menu for a native LabVIEW graph.


Creating an RTM
Finite Acquisition and Analysis [Main.vi] Front Panel on Finite
File Edit View Project Operate Tools Window Help
Undo Ctrl+Z
Redo Ctrl+Shift+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Select All Ctrl+A
Make Current Values Default
Reinitialize Values to Default
Customize Control...
Import Picture to Clipboard...
Set Tabbing Order...
Remove Broken Wires Ctrl+B
Clean Up Panel Ctrl+U
Remove Breakpoints from Hierarchy
Create VI Snippet from Selection
Create SubVI
Disable Panel Grid Alignment Ctrl+#
Align Items Ctrl+Shift+A
Distribute Items Ctrl+D
VI Revision History... Ctrl+Y
Run-Time Menu...
Adjust Pane Origin
Find and Replace... Ctrl+F
Show Search Results Ctrl+Shift+F
Menu Editor - C:\Users\Thomas.McQuillan\OneDrive\LabVIEW Essentials I Training Course\Module 7 - Building an Application\Finite Acquis...
File Edit Help
Custom
Preview: File Data Settings
File
Save
Open...
Quit
Data
Configure Data
Acquire Data
Analyze Data
Settings
Clear Graph
Reset Settings
Finite Acquisition and Analysis
File Data Settings
Save Ctrl+S
Open... Ctrl+O
Quit Ctrl+Q
Configure Data Ctrl+I
Acquire Data Ctrl+A
Analyze Data Ctrl+Shift+A
Clear Graph Ctrl+Shift+C
Reset Settings Ctrl+Shift+R

Item Properties
Item Type: User Item
Item Name: Reset Settings
Item Tag: Reset Settings
☑ Enabled
☐ Checked
Shortcut (Press key combination): Ctrl+Shift+R

Implementation-


Main.vi Front Panel on Finite Data Acquisition and Analysis.lvproj/My Computer
Menu Editor - Untitled.rtm*
Preview: File Data Settings
File
Save
Open
Quit
Data
Configure Data
Acquire Data
Analyse Data
Settings
Clear Graph
Reset Settings
Item Properties
Item Type: User Item
Item Name: Reset Settings
Item Tag: Reset Settings
Enabled
Checked
Shortcut (Press key combination): Ctrl+Shift+R


Finite Acquisition and Analysis [Main.vi] Block Diagram on Finite Data Acquisition and Analysis.lvproj/My Computer *
Finite Data Acquisition and Analysis - Event Based State Machine
"Wait for Event"
Wait for Event - Use an Event Structure to monitor the user interface for events that run different states of the state machine.
[8] Menu Selection (User)
>App Data>
Current State
Error
Initial State
Initialize
>New State>
ItemTag
Wait for Event
>Next State>
Stop Loop?
This is the Simple State Machine design pattern. Each frame of this case structure executes code for its state and computes what state to transition to next. States are represented as values of an enumeration. These enumerations are instances of a type definition

4.4.2 Creating Standalone Application-

Stand-alone applications are that can be distributed to users who have the LabVIEW Run-time Engine.

LabView Application Builder

The LabVIEW Application Builder leverages the organization provided by the LabVIEW Project, which organizes and manages the files associated with an application. These files include VIs, project libraries, documentation, data files, hardware configuration and more. The Application Builder creates applications, DLLs, and more from user-specified files in a LabVIEW project, and individual build settings are saved in Build Specification in the project.

With application builder one can find-

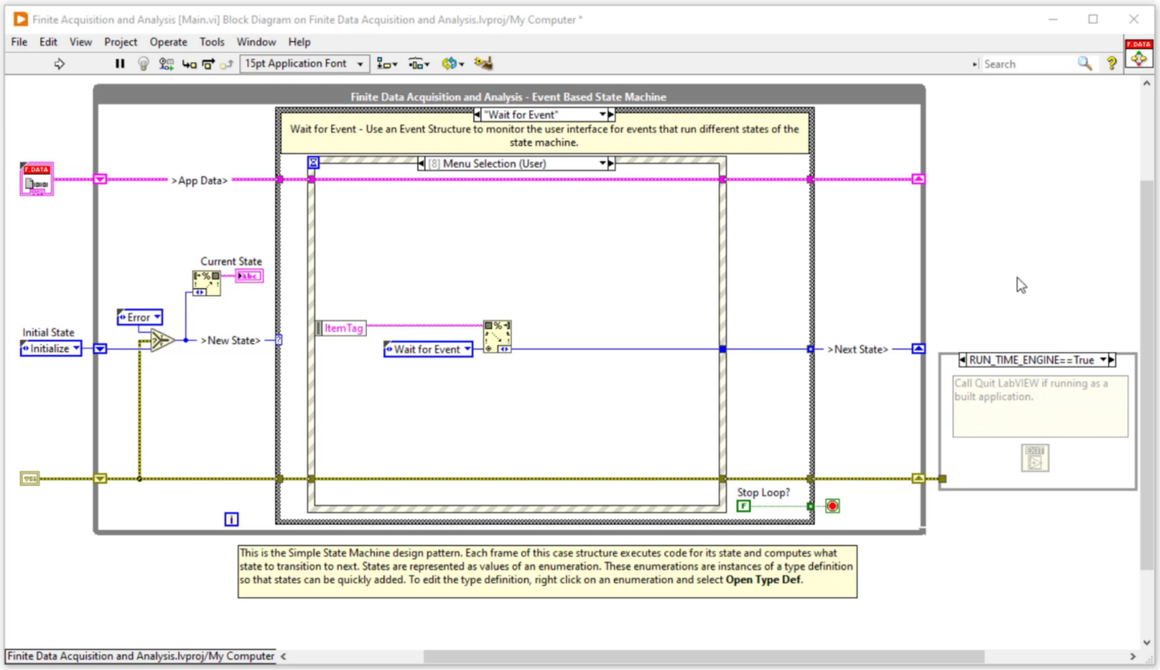
1. **Applications-** Compiled code that can run without the full LabVIEW development system. But they require LabVIEW runtime engine.
2. **Installers-** Will install applications into Program Files. One can include other software packages in the installer, such as the LabVIEW Runtime Engine, NI Max or DAQmx Runtime.

Conditional Disable Structure

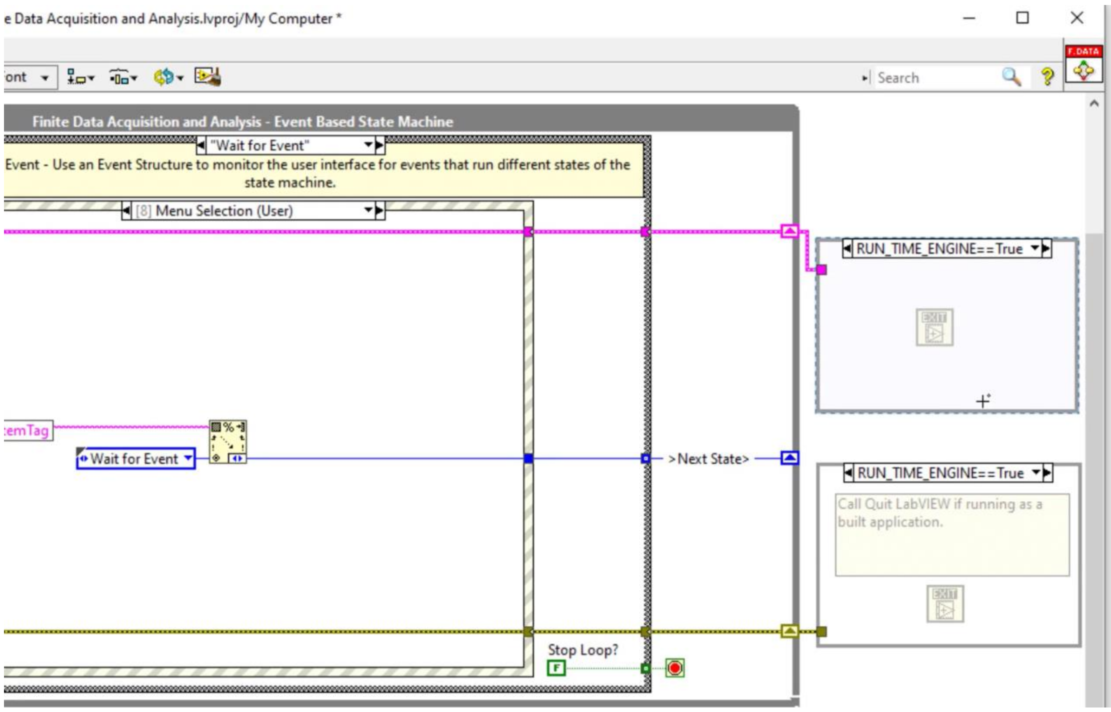
Stopping the Application

- If the code is running on the LabVIEW Environment the code should stop and return to the editor mode.
- If the code is running with the Runtime Engine, the code should stop and close completely.

Implementation-

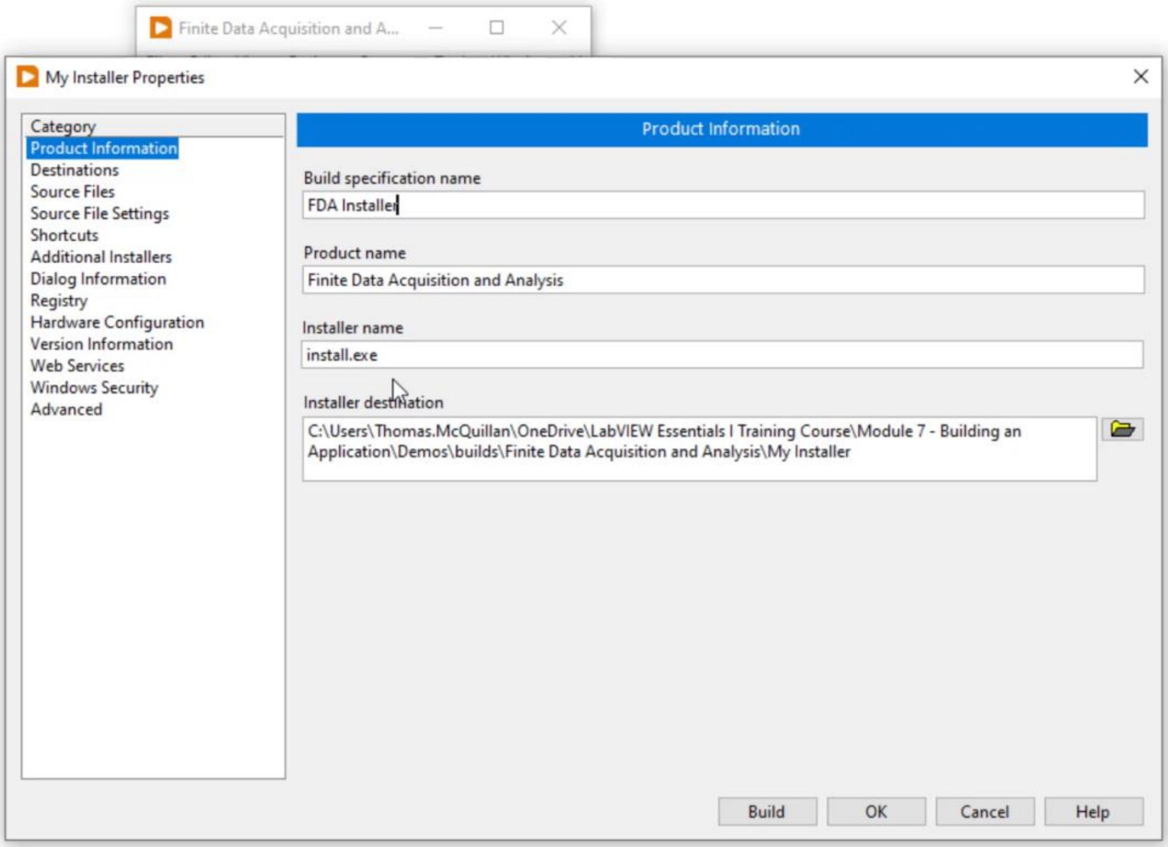

Finite Acquisition and Analysis [Main.vi] Block Diagram on Finite Data Acquisition and Analysis.lvproj/My Computer *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Finite Data Acquisition and Analysis - Event Based State Machine
"Wait for Event"
Wait for Event - Use an Event Structure to monitor the user interface for events that run different states of the state machine.
[8] Menu Selection (User)
>App Data>
Current State
Initial State
Initialize
Error
>New State>
ItemTag
Wait for Event
>Next State>
RUN_TIME_ENGINE==True
Call Quit LabVIEW if running as a built application.
Stop Loop?
Finite Data Acquisition and Analysis.lvproj/My Computer

This is the Simple State Machine design pattern. Each frame of this case structure executes code for its state and computes what state to transition to next. States are represented as values of an enumeration. These enumerations are instances of a type definition so that states can be quickly added. To edit the type definition, right click on an enumeration and select **Open Type Def.**


e Data Acquisition and Analysis.lvproj/My Computer *
Search
Finite Data Acquisition and Analysis - Event Based State Machine
"Wait for Event"
Event - Use an Event Structure to monitor the user interface for events that run different states of the state machine.
[8] Menu Selection (User)
RUN_TIME_ENGINE==True
temTag
Wait for Event
>Next State>
RUN_TIME_ENGINE==True
Call Quit LabVIEW if running as a built application.
Stop Loop?

- **Building an Installer-**

Implementation-

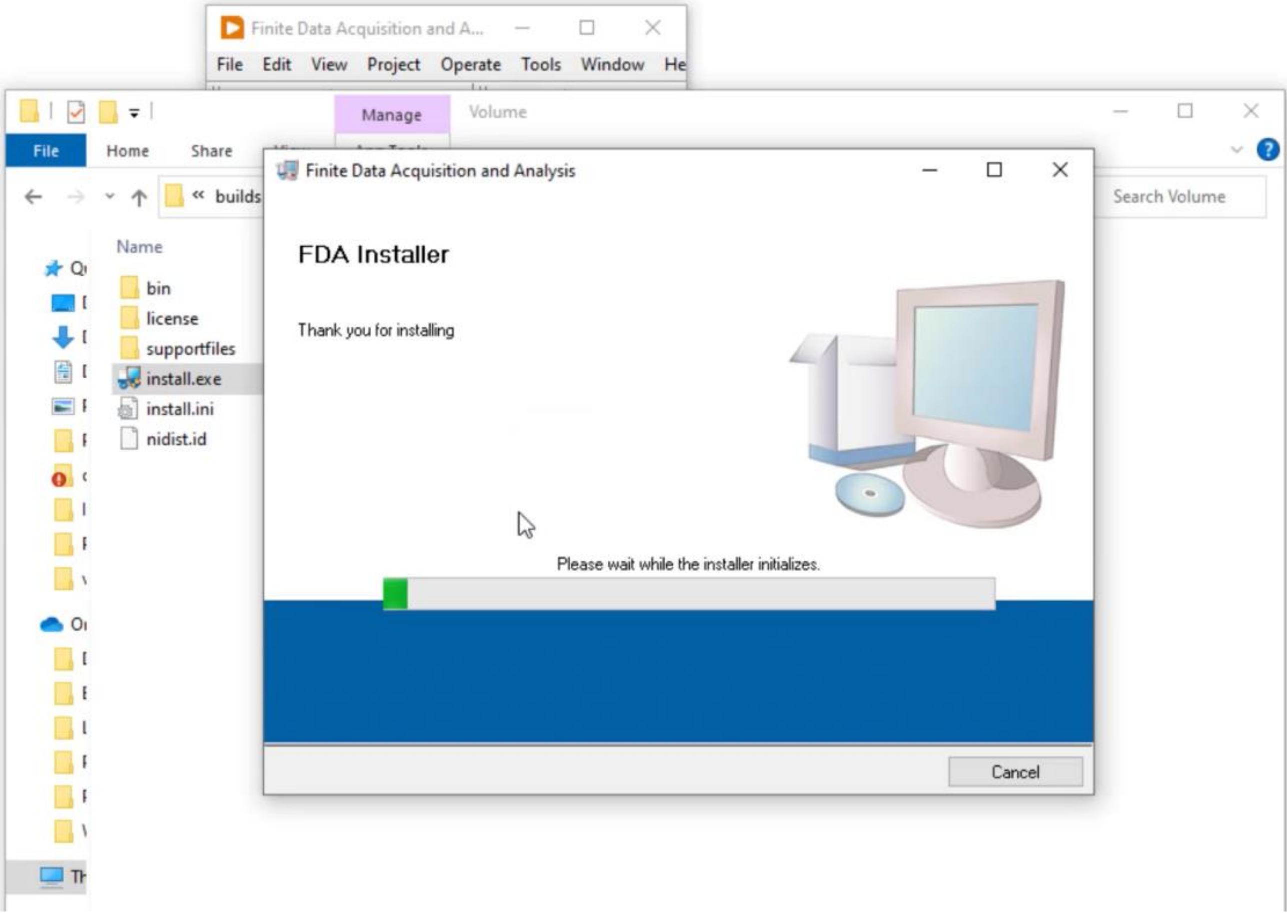


Build specification name
FDA Installer

Product name
Finite Data Acquisition and Analysis

Installer name
install.exe

Installer destination
C:\Users\Thomas.McQuillan\OneDrive\LabVIEW Essentials I Training Course\Module 7 - Building an Application\Demos\builds\Finite Data Acquisition and Analysis\My Installer


Finite Data Acquisition and A...
File Edit View Project Operate Tools Window He
Manage
Volume
File Home Share
builds
Search Volume
Name
bin
license
supportfiles
install.exe
install.ini
nidist.id
Finite Data Acquisition and Analysis
Cancel

FDA Installer

Thank you for installing

Please wait while the installer initializes.

5. CONCLUSION

This training on LabVIEW Programming started from the very basic of the starting of how to start LabView Program till the design and building of application of Finite Data Acquisition. The training comprised of fundamental of data types, code structures, Modular Code, VI server and file types, hardware implementation, design patterns and building applications.

This project is made as per customer requirement and can also be updated. LabVIEW has a tremendous value in industry and will help to connect and develop user defined works.

6. BIBLIOGRAPHY

1. *JALA: Journal of the Association for Laboratory Automation,* Volume: 12 issue: 1, page(s): 17-24

   Issue published: February 1, 2007
   Chance Elliott[1], Vipin Vijayakumar[1], Wesley Zink[1], Richard Hansen, Ph.D.*[1]
   [1] Cytokinetics, Inc., San Francisco, CA
2. Getting Started with LabVIEW: National Instruments