Faculty of Engineering & Technology Electrical & Computer Engineering Department

Real-Time system and interfacing Techniques laboratory - ENCS5140

LabVIEW Module

(EXP#4, EXP#5)

Student Name: Raghad Afaghani

Student Number: 1192423

Group Members:

Student Name: Hala Ziq

Student Number: 1191637

Student Name: Zakiya Abu Murra

Student Number: 1191636

Instructor: Dr.Hanna Bullata & Ahed Mafarjeh

Section: #1 **Date: 9/4/2024**

Abstract

This report provides a overview of LabVIEW and its integration with LINX for embedded system development, focusing on its graphical programming technique and practical applications. LabVIEW, a graphical programming environment, offers a powerful platform for developing various instrument-based applications, minimizing errors and enhancing data reliability. The report discusses examples how to use LabVIEW's front panel, which serves as the user interface, and its back panel, containing the graphical code responsible for processing inputs and generating outputs. Additionally, it explores the LabVIEW LINX Toolkit, facilitating seamless integration with embedded platforms like Arduino.

Table Of Contact

Table of Contents

Table of Figures

1.0 Theory

1.1. LabVIEW

Graphical Programming Technique is a method that uses visual blocks instead of text to create computer programs. LabVIEW, which stands for Laboratory Virtual Instrument Engineering Workbench, was the first graphical programming implementation and remains the most widely used. It provides a powerful and integrated environment for developing various applications involving instruments. A well-designed LabVIEW application minimizes unnecessary operations and reduces errors in data collection and processing. This leads to more reliable data, improving product quality control and enabling new discoveries. LabVIEW programs, also known as virtual instruments (VIs), resemble and function like physical instruments. They offer a wide range of tools for tasks such as acquiring, analyzing, displaying, and storing data, as well as troubleshooting code. LabVIEW also features built-in capabilities for connecting applications to the internet using the LabVIEW Web Server. It can handle large and professional applications and includes integrated project management tools, graphical debugging tools, and standardized source code control integration. LabVIEW provides the necessary tools for most applications and offers an open development environment that allows customization and flexibility. [1]

1.2. Front Panel

The front panel serves as the user interface for interacting with the VI (Virtual Instrument) in LabVIEW. It displays outputs and enables users to provide inputs to the program. The main objects found on the front panel are controls and indicators. Controls simulate input devices and supply data to the block diagram of the VI. They include knobs, pushbuttons, dials, and other similar input devices. On the other hand, indicators simulate output devices and display data acquired or generated by the block diagram. Examples of indicators include graphs, LEDs and other output devices. Together,

controls and indicators facilitate the exchange of information between the user and the VI, enhancing the usability and functionality of the program. [1]

1.3. Back Panel

The back panel, also known as the block diagram in LabVIEW, contains the code that processes inputs from the front panel and generates outputs. It is responsible for controlling the program through graphical code. The block diagram utilizes a graphical representation of functions to manipulate the front panel objects. Within the back panel, structures and functions perform operations on controls and provide data to indicators. LabVIEW offers three different palettes: the Tool Palette, available on both the front panel and block diagram, which provides various tools for creating, modifying, and debugging virtual instruments; the Controls Palette, exclusive to the front panel, which offers controls and indicators for building the user interface; and the Function Palette, specific to the block diagram, which provides different function categories like numeric, array, time, dialog, and waveform for constructing the code logic. These palettes enable users to efficiently design and implement their LabVIEW programs.[1]

1.4. LINX

The LabVIEW LINX Toolkit is a versatile add-on for the LabVIEW programming environment. It provides the capability to integrate and control embedded platforms such as Arduino, Raspberry Pi, and BeagleBone Black directly from LabVIEW. This toolkit simplifies the process of connecting to and communicating with these devices, allowing for a wide range of applications, including data logging and hardware control. By using LINX, developers can leverage the advanced graphical programming capabilities of LabVIEW to create applications without the necessity for costly data acquisition (DAQ) hardware. This makes embedded system projects more accessible and cost-effective, particularly for small-scale projects, education, and prototyping. [3]

1.4.1 Why LINX

LINX is great for working with Arduino in LabVIEW because: [3]

- It lets you control your project with a visual interface.
- The drag-and-drop programming makes designing easier.
- It comes with helpful tools to fix errors quickly.
- It has ready-to-use parts for both easy and complex jobs.
- You can change it however you want, so it fits exactly what you're doing.

1.5. LIFA

Previously, there existed a toolkit named LIFA (LabView Interface For Arduino). LIFA's creator chose to enhance the interface to include more devices (and make other improvements). So, LINX took over LIFA a few years ago ("LabVIEW INterface for X"; where X stands for the various devices that it supports, like Arduino, chipKIT, etc.). Furthermore, LINX eliminated numerous difficulties that existed in LIFA, resulting in considerable communication efficiency increases since the protocol was rebuilt, making most of the functions that connect with the device considerably more efficient. [3]

1.6. Advantages of LabVIEW

This technique offers several advantages over text-based programming. Firstly, graphical programming is highly interactive, providing a more intuitive and visual approach compared to text-based programming. Unlike text-based programming, where knowledge of syntax is necessary, graphical programming eliminates the need for syntax knowledge. Additionally, in text-based programming, extra coding is often required for front panel design, whereas graphical programming eliminates the need for such additional coding. Furthermore, graphical programming highlights errors as blocks are wired, making it easier to identify and correct them. In contrast, text-based programming typically requires compilation to check for errors.[1]

1.7. Disadvantages of LabVIEW

LabVIEW's single-source nature and lack of industry standardization may make it less appealing to certain companies. Before implementing LabVIEW, it is important to carefully evaluate the cost of ownership, even though it is comparable to similar industry products. Additionally, individuals accustomed to text programming may require some time to become familiar with LabVIEW's graphical programming approach. Considering these factors, such as single sourcing, cost implications, and the learning curve for text programmers, is crucial in determining the suitability of LabVIEW for a company's needs. [2]

2.0 Procedure Decision

2.1 Exp#4: Introduction to LabView

2.1.1 Design a Simple Alarm System

A control system was designed with the following specifications: the system had to have two inputs, namely pressure and temperature. An alarm was to be activated in two cases: when the temperature exceeded 100°C or the pressure exceeded 15KPa. The temperature and pressure transducers had transfer functions of 3.2mV/°C and 0.3V/KPa, respectively. The necessary components were added. The system's front panel and block diagram are shown in the figures below.

Figure 1 Front Panel for the Alarm System

At the current temperature reading of 50°C, the resulting value after multiplying by 0.0032 is 0.16. However, since this value does not exceed the threshold of 0.32, the alarm will not be triggered. This means that as long as the temperature remains below 100°C, the alarm will remain off. Similarly, for the pressure reading of 3KPa, the resulting value of 0.9 (after multiplying by 0.3) does not exceed the threshold of 4.5, thus the alarm will not be activated. Neither the temperature nor the pressure readings have exceeded the chosen threshold, indicating that the alarm has not been triggered.

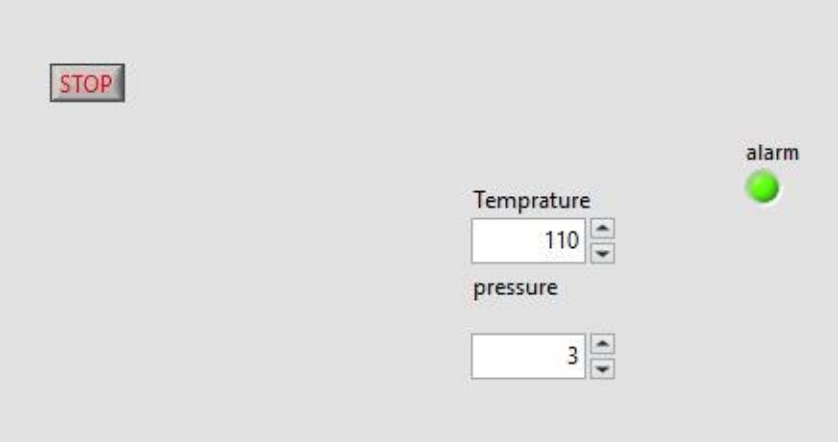

Figure 2 When Temperature exceeds 100C

- The alarm was activated when the temperature exceeded 100°C.

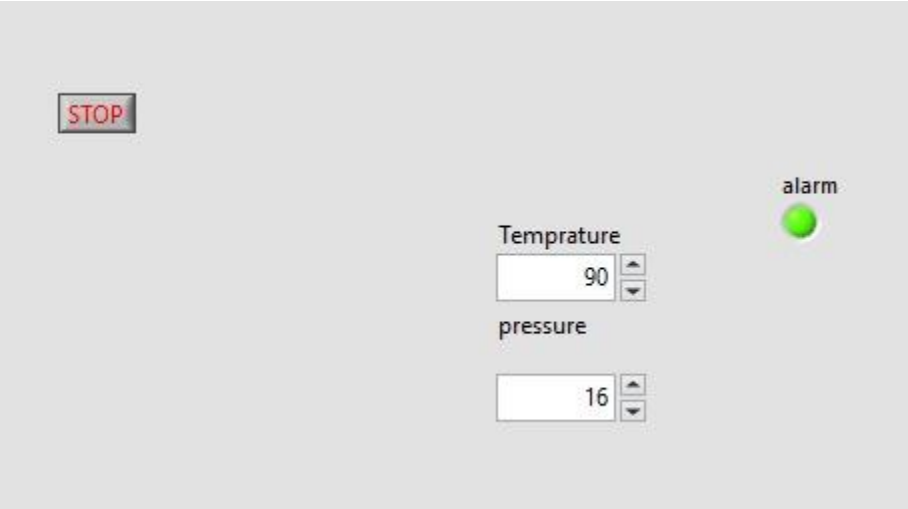

Figure 3 When Pressure exceeds 15Kpa

- The alarm was activated when the pressure exceeded 15KPa.

On the Block Diagram Panel, the logic was implemented by multiplying the temperature and pressure values with their respective transfer functions. This process yielded voltage values from each sensor. These voltages were then compared with the allowed voltage threshold. If either of the sensor voltages exceeded its limit, an alarm was triggered. This was achieved by using the OR logic operator to evaluate the voltages of the two sensors. If the OR condition was met, indicating that either sensor had exceeded its limit, the alarm was activated.

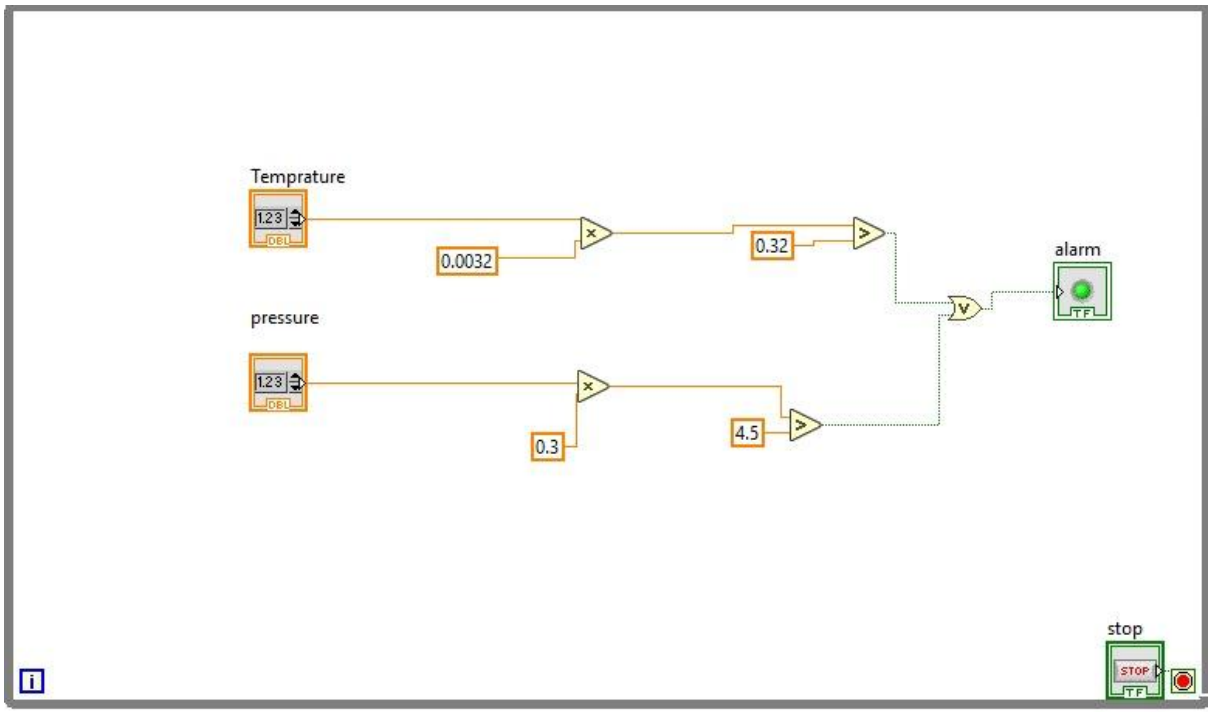

Figure 4 Block Diagram of an Alarm System

To locate any component in LABVIEW, a right-click was made and a search was performed. Similarly, to identify the components of this system, the following steps were followed. Firstly, on the block diagram, the Functions palette was accessed by right-clicking on any blank space. From there, the path was Functions palette > Express > Exec Control > while loop. The while loop block was expanded to the appropriate size, and it was noted that a stop button appeared on the front panel, allowing for the termination of the entire VI when pressed. All controls and indicators visible on the block diagram were

placed within the while loop. Secondly, on the front panel, the VI block diagram shown in Figure [4] was constructed. This involved placing an LED from the Control palette > Express > LEDs and renaming it as required. Additionally, two "Num Ctrl" VIs were placed and resized accordingly, also renaming them as needed. At this point, the front panel resembled Figure[1]. Moving back to the block diagram, all the blocks were dragged and dropped into the while loop. A "Multiply" function from Functions > Mathematics > Numeric VI was placed on the block diagram, and a constant value was wired by right-clicking on the component. Finally, a "Comparator" VI from Function > Express > Arithmetic & Comparison was placed on the block diagram, and the output of the multiply function from the previous step was wired into the comparator input. Again, a constant value was wired by right-clicking on the component.

2.1.2 Simple Liquid Store System

A simple liquid storage system was designed in LABVIEW with the following specifications: The system had four inputs (volume of the liquid, temperature of the liquid, and two enables) and two outputs (LED and screen). When the volume of the liquid exceeded 6 liters and the LED enable was on, the LED turned into a red color. Similarly, when the temperature of the liquid exceeded 60 °C and the temperature enable was on, the message "The temperature of the liquid is high" was displayed on the screen.

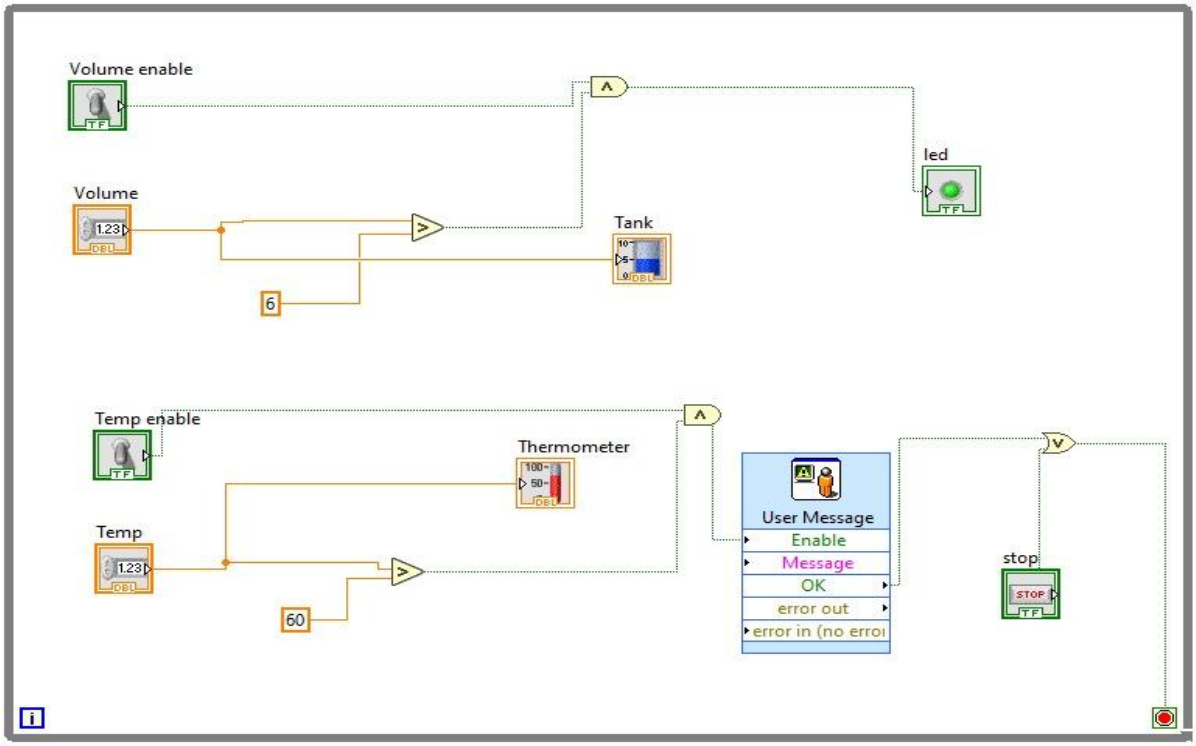

Figure 5 Block Diagram of Liquid Store System

The sensor readings are compared to their respective limits in the front panel. In the back panel, the comparison results and the enable signals of the sensors are used as inputs for an AND gate. The output of the AND gate determines the control of the LED and the display of a message.

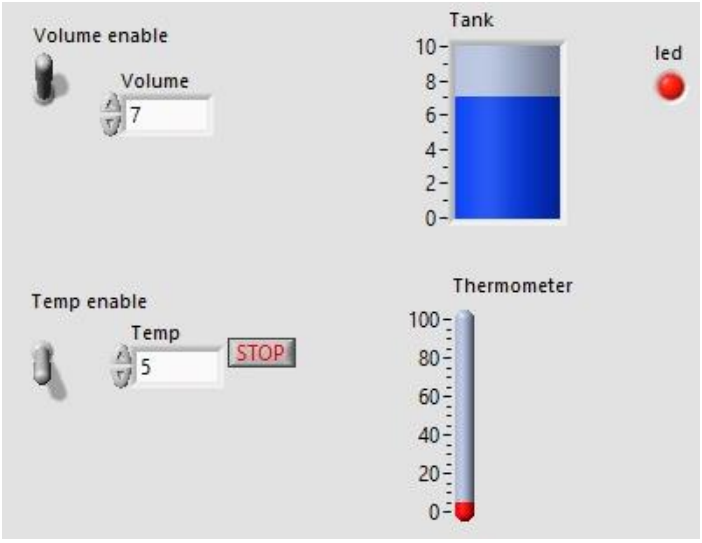

Figure 6 Liquid Store System Case-1

The volume enable is activated, allowing for monitoring of the liquid's volume. The current volume of the liquid is measured at 7 liters, exceeding the threshold of 6 liters. Consequently, the LED indicator is illuminated to indicate that the volume exceeds the specified limit. It is important to note that the liquid's volume is visually displayed in the Tank for easy observation and monitoring.

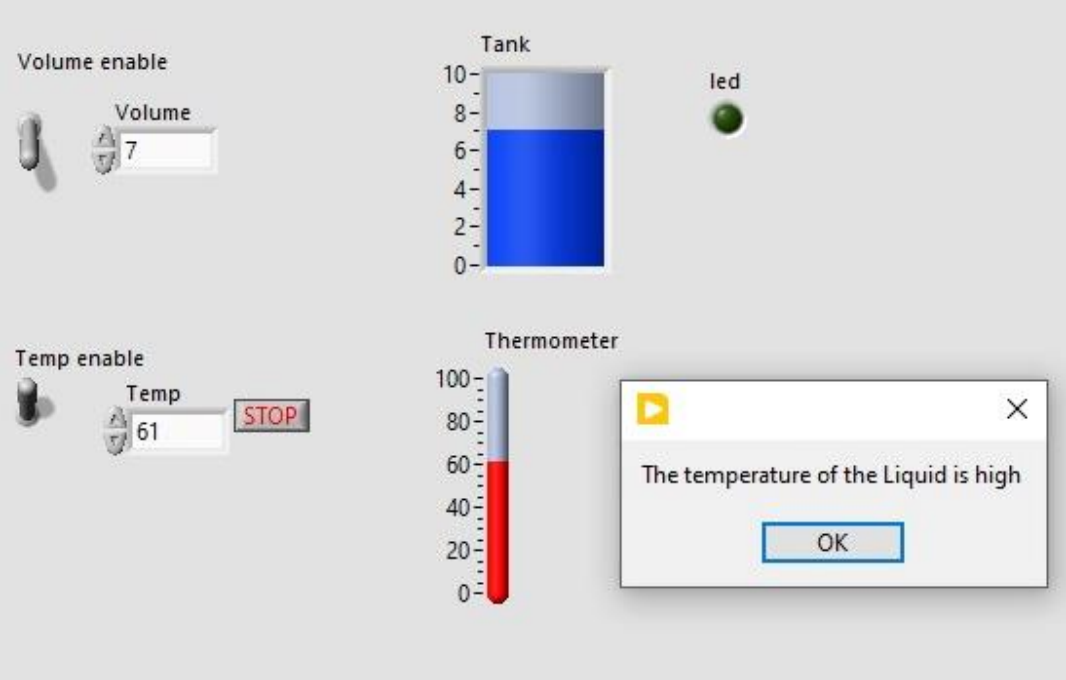

Figure 7 Liquid Store System Case-2

The LED is not illuminated because the volume enable is switched off, despite the liquid's volume being above 6 liters. However, the temperature enable is activated, and since the temperature reading is 61°C, which is higher than the set threshold of 60°C, a message is displayed.

Additionally, to design a system that turned the LED on for x seconds and then off for y seconds, a set of components were utilized. These included a while loop with a button, a case structure, a shift register, a delay function, a not function, a constant Boolean, and an LED. The front and block diagram for this system is shown in the below figures.

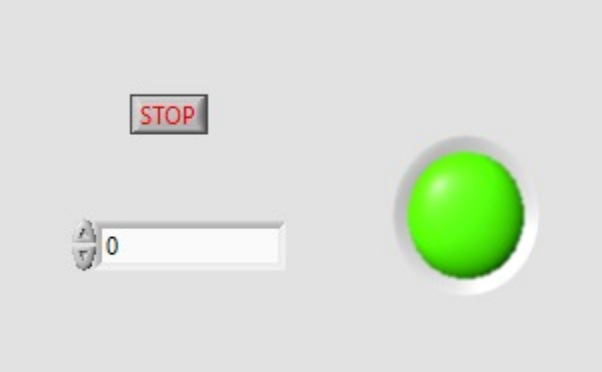

Figure 8 Front panel for case structure

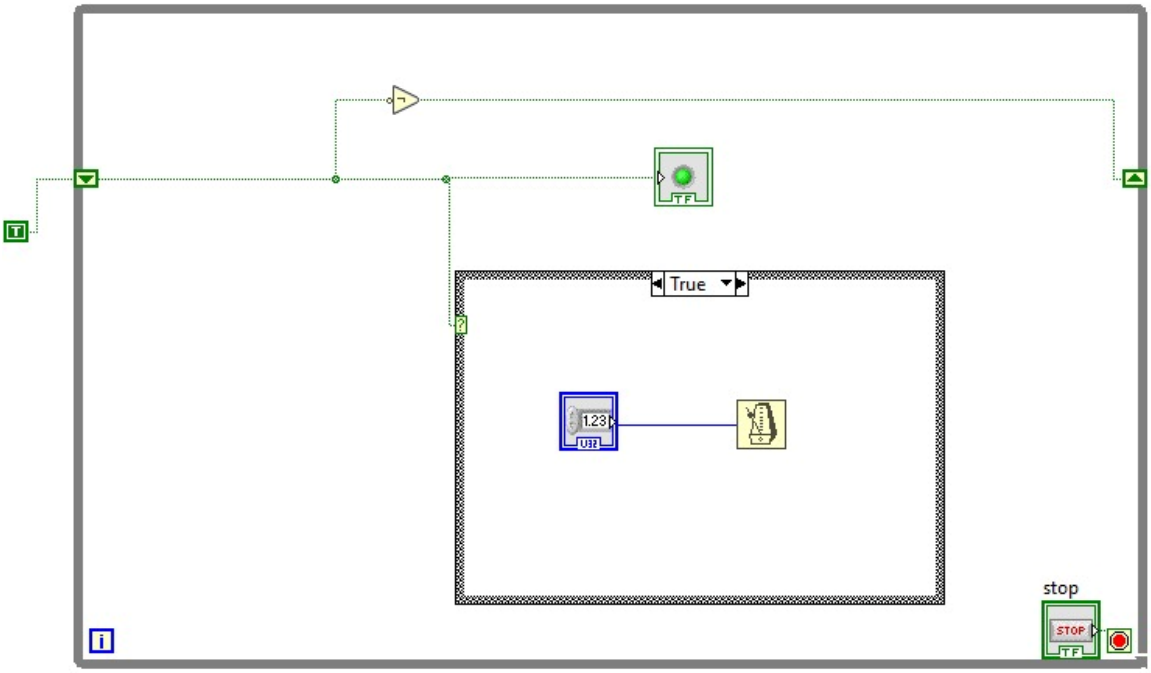

Figure 9 Block Diagram for case structure

2.1.3 Lowpass and High-pass filters in time domain

The system setup involved generating a sinusoidal sine signal with specific amplitude and frequency. This signal was then fed into both a low-pass filter and a high-pass filter, with the cutoff frequencies for each filter being user-defined. On the Front Panel, four sliders were added, allowing the user to adjust the input signal's frequency, amplitude, and cutoff frequencies for the low-pass and high-pass filters. Additionally, three signal monitors were included to display the original input signal, as well as how the signal behaves after passing through the low-pass and high-pass filters, respectively. In the Block diagram, the input

signal's amplitude and frequency were used as inputs for a signal simulation block, which generated the initial input signal displayed on the first monitor. This signal was then processed by the low-pass and high-pass filter blocks, using the user-defined cutoff frequencies. The filtered signals were subsequently displayed on the second and third monitors, respectively, showcasing the effects of the filtering process.

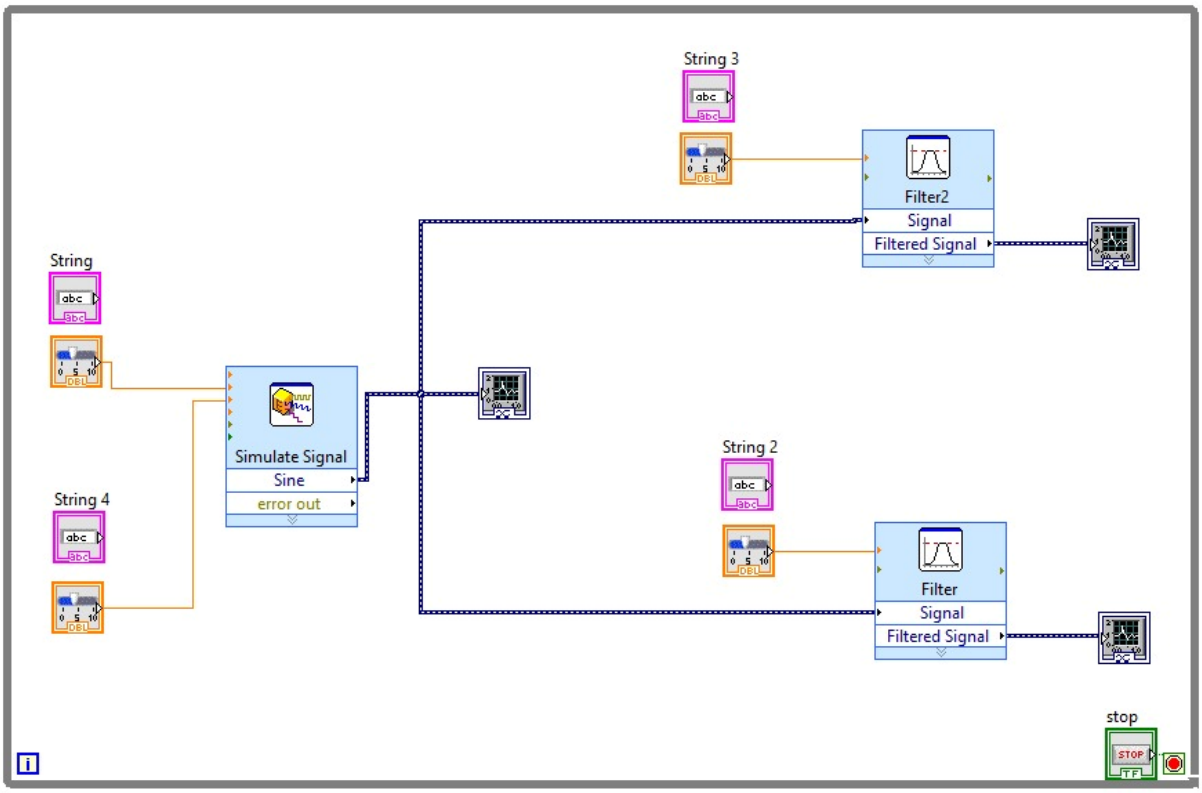

Figure 10 Block Diagram for Low-pass and High-pass Filters

A lowpass and high pass filters VI was constructed with a sampling frequency (fs) of 8000, which was twice the signal frequency (2fm). The cutoff frequency for the high pass filter was significantly lower than 8000, indicating that frequencies below the cutoff were attenuated.

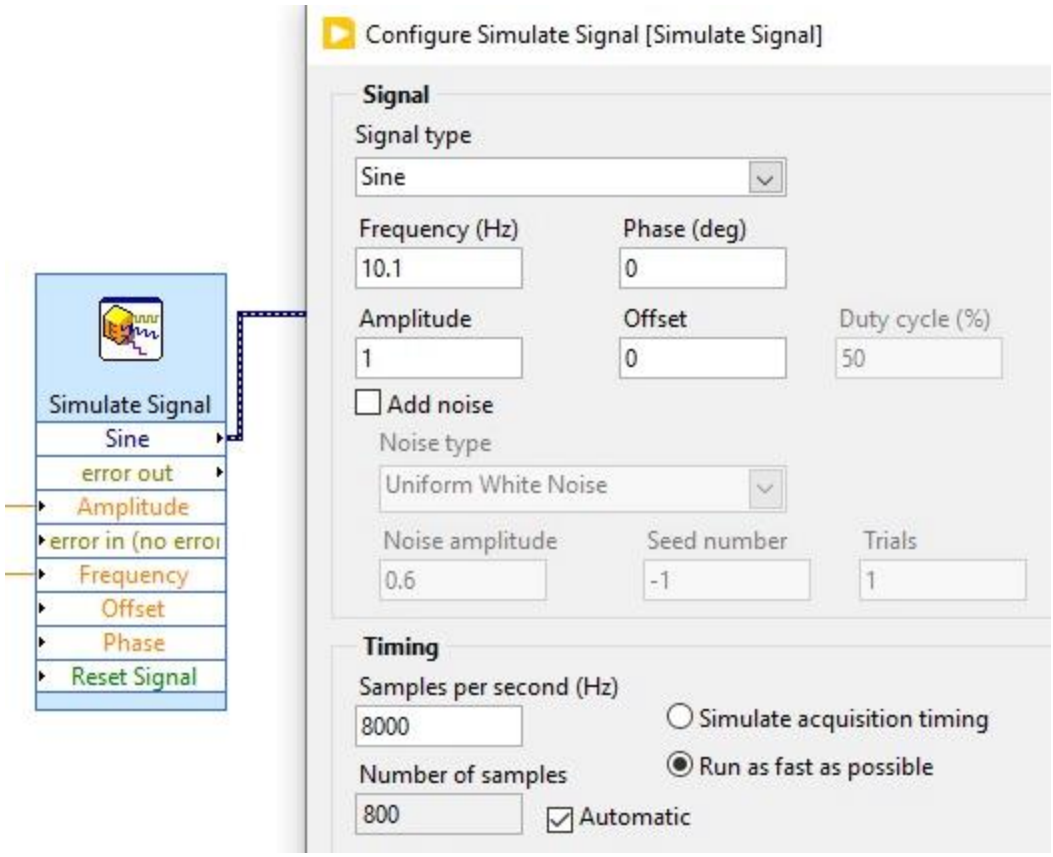

Figure 11 Changing the Sampling Frequency

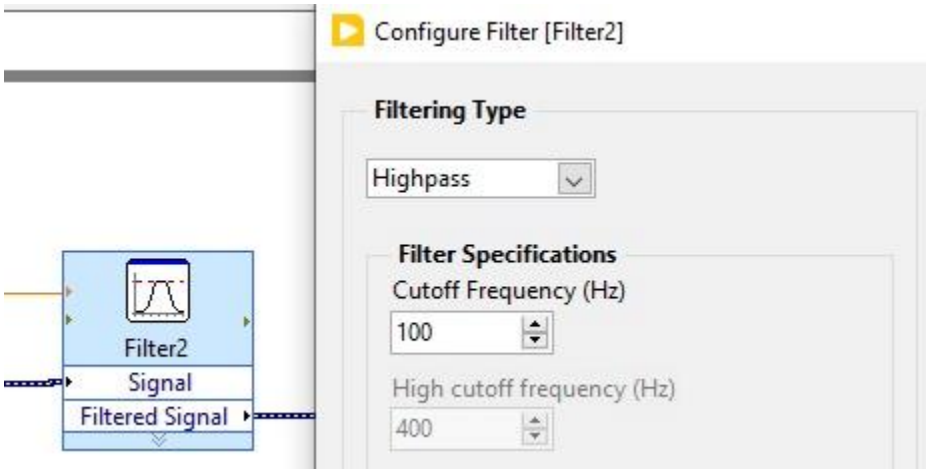

Figure 12 Changing the Cutoff Frequency

A lowpass and highpass filters VI was constructed with a sampling frequency (fs) of 8000, which was twice the signal frequency (2fm). The cutoff frequency for the high pass filter was significantly lower than 8000, indicating that frequencies below the cutoff were attenuated.

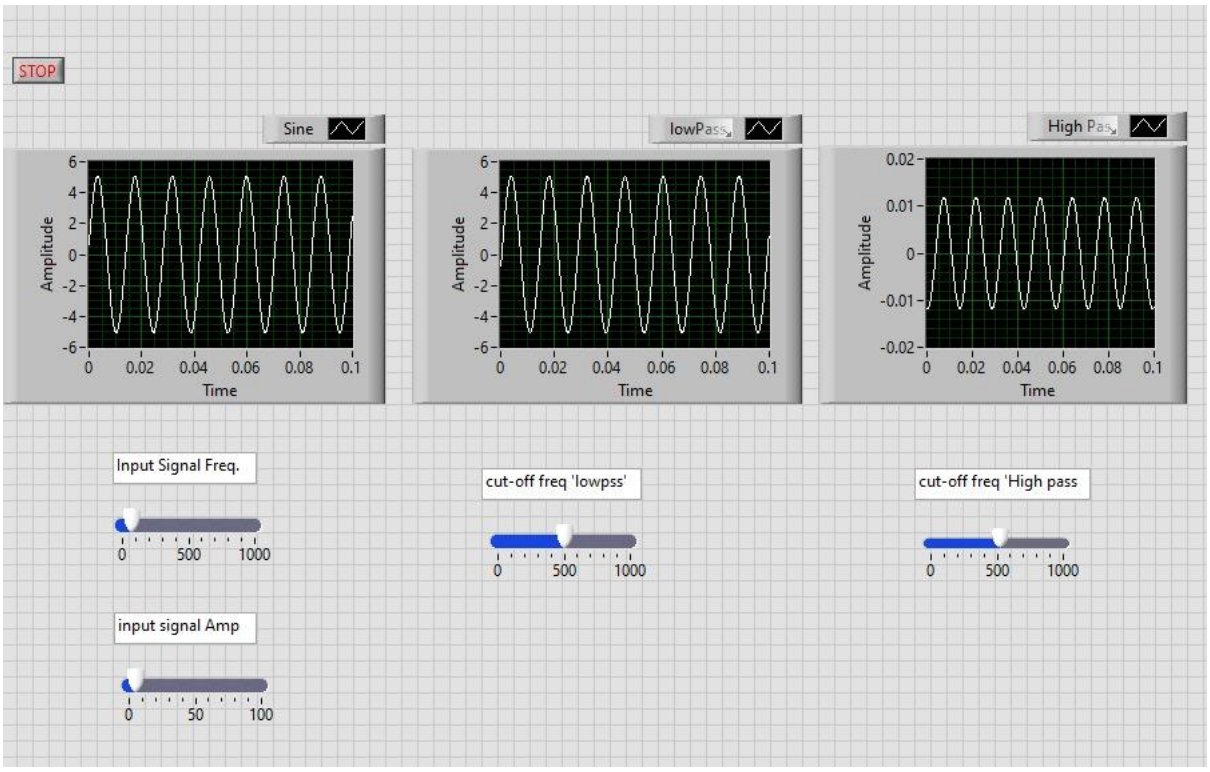

Figure 13 The Low-pass and High-pass Filters Case 1

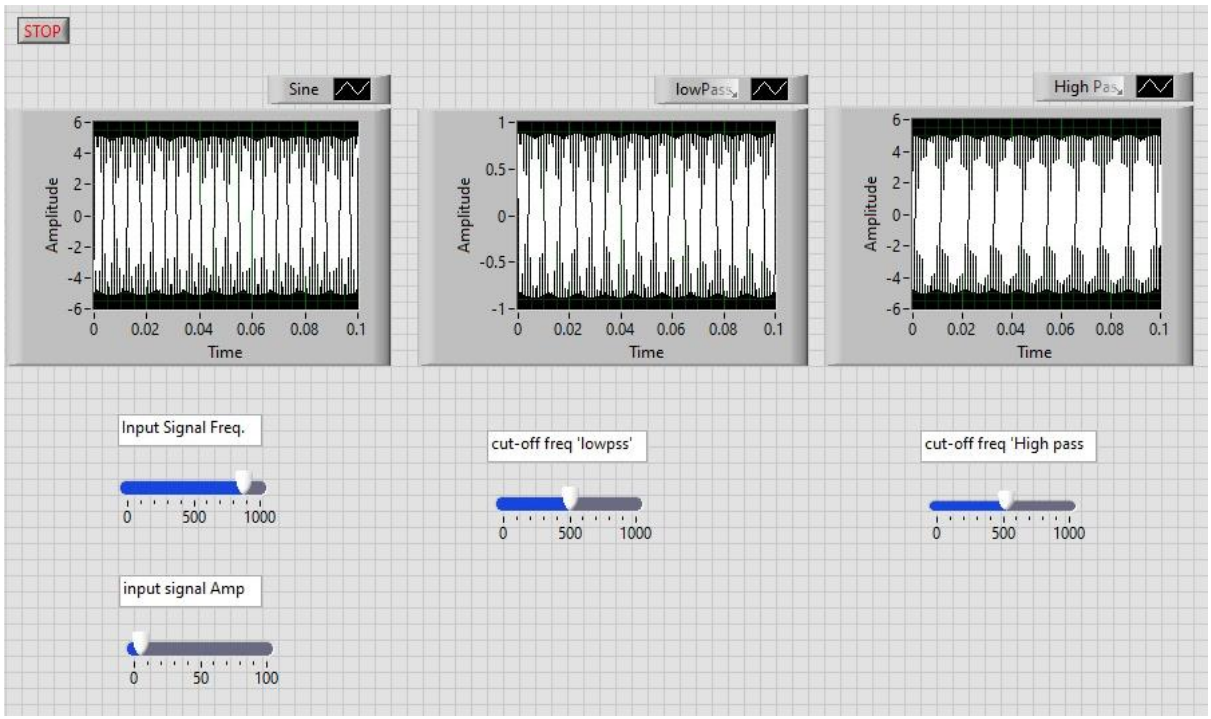

Figure 14 The Low-pass and High-pass Filters Case 2

The low-pass filter allows only the low-frequency components to pass through, while the high-pass filter allows only the high-frequency components to pass through. In the provided figure, when the cutoff frequency of the low-pass filter is set lower than the frequency of the input signal, it selectively captures and retains the low-frequency components that are usually the desired parts of the signal. On the other hand, when the cutoff frequency of the high-pass filter is set higher than the frequency of the input signal, it selectively captures and retains the high-frequency components that often represent the noise present in the original signal.

2.2 Exp#5: LabVIEW with Arduino using LINX

2.2.1 Control LED 13 in Arduino by LabView.

In this part of the experiment, an Arduino onboard LED (LED 13) was controlled by a switch made in the LabVIEW front panel. The process involved setting up a serial communication link between LabVIEW and the Arduino to enable control commands to be sent. In the LabVIEW environment, the front panel served as the user interface with a switch to toggle the LED and a stop button to terminate the program. The back panel was configured with LINX VIs to manage the serial port communication, specifying the channel for the digital signal and executing the control logic for the LED.

- Figure 15 and 16 are the configuration of the LED control on LabView and the Arduino connection.

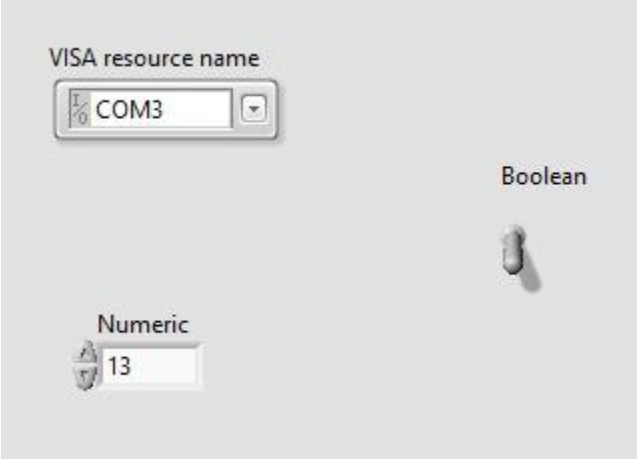

Figure 15 Front panel for Control LED 13 in Arduino by LabView.

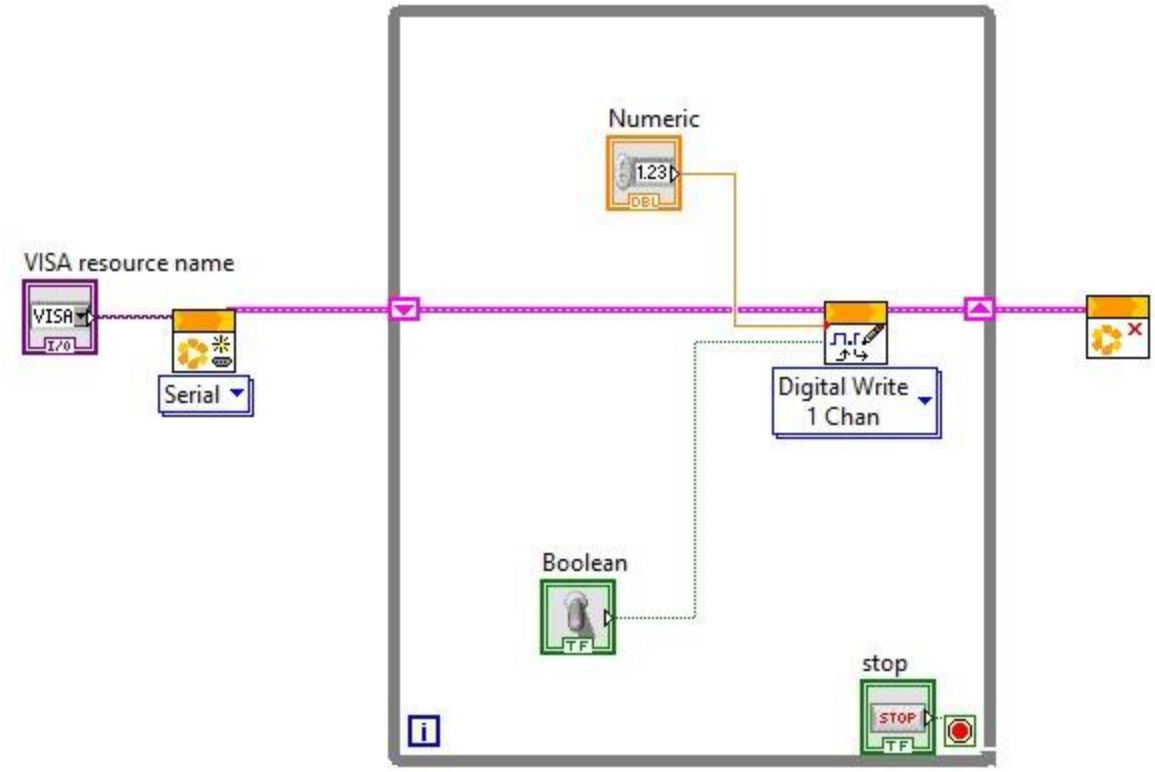

Figure 16 back panel for Control LED 13 in Arduino by LabView.

2.2.2 Array Example

In this section, the setup included two binary arrays: an indicator array and a control array. Seven LEDs were attached to the indicator array, and seven switches were linked to the control array. Both arrays were interconnected through a while loop on the back panel. Pressing a switch alters its state, which, in turn, activates the corresponding LED in the indicator array above, with each switch being designated to control a specific LED.

Figure 17 Led 13 in Arduino control

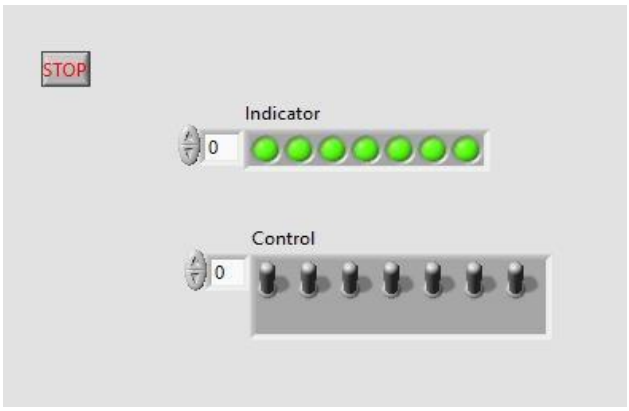

Figure 18 Front Panel for Array Example

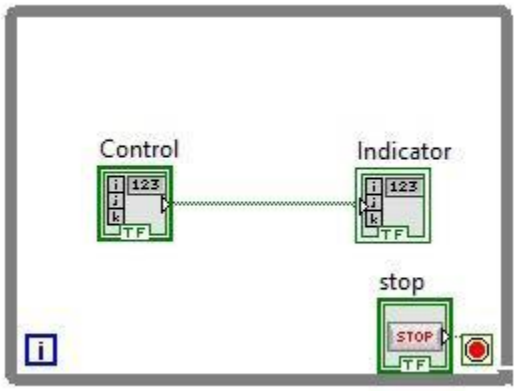

Figure 19 Back Panel for Array Example

The purpose of this setup was to show the basics of digital input and output control, a fundamental concept in embedded systems programming. By offering a real-time visual representation of the states of digital inputs and outputs. This highlights the effectiveness of LabVIEW in making intricate programming tasks more accessible and understandable.

2.2.3 Read analog value (LDR)

In this part, we focused on measuring light using a Light Dependent Resistor (LDR). We built a voltage divider circuit with the LDR to convert different light levels into measurable voltage levels. The circuit was connected to an Arduino board as shown in figure 19, which read the analog voltage value generated by the LDR, the LabVIEW program was then used to display these voltage readings on a gauge in the front panel. This configuration allowed us to monitor the changes in light detected by the LDR through changes in the voltage levels displayed on the gauge.

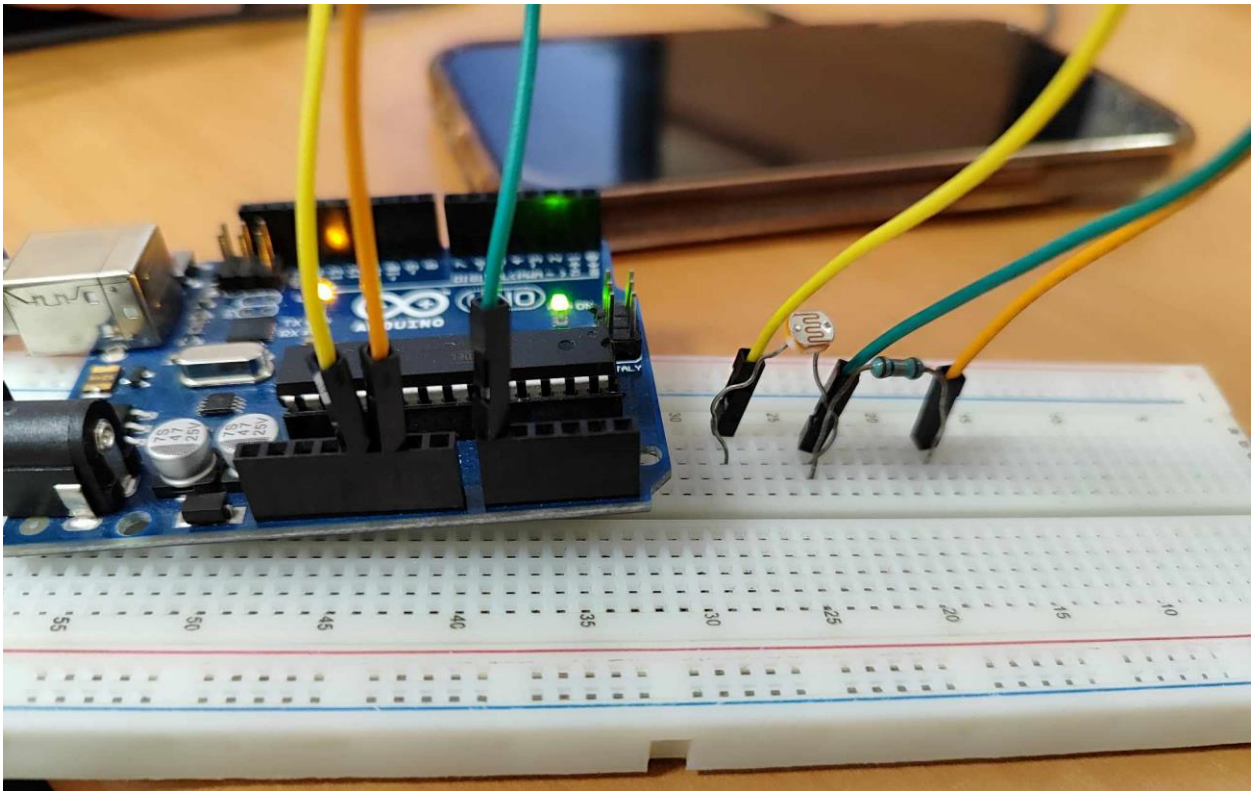

Figure 20 LDR and Arduino connection

- Figure 20, LabVIEW front panel showing the real-time voltage reading from the LDR on a gauge, with controls for stopping the program and selecting the VISA resource, and figure 21 shows the LabVIEW Block Diagram for Digital Write to Arduino using VISA communication.

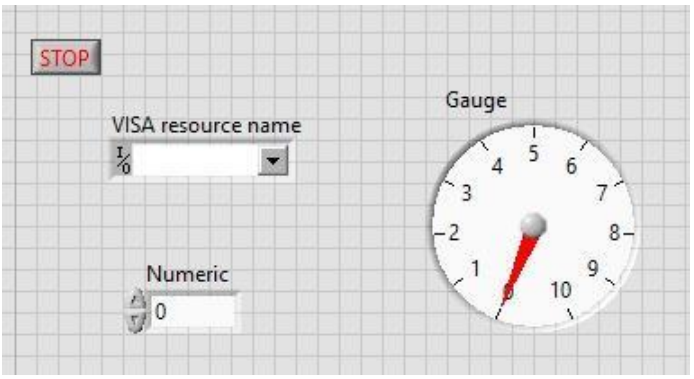

Figure 21 front panel for Read Analog Value configuration

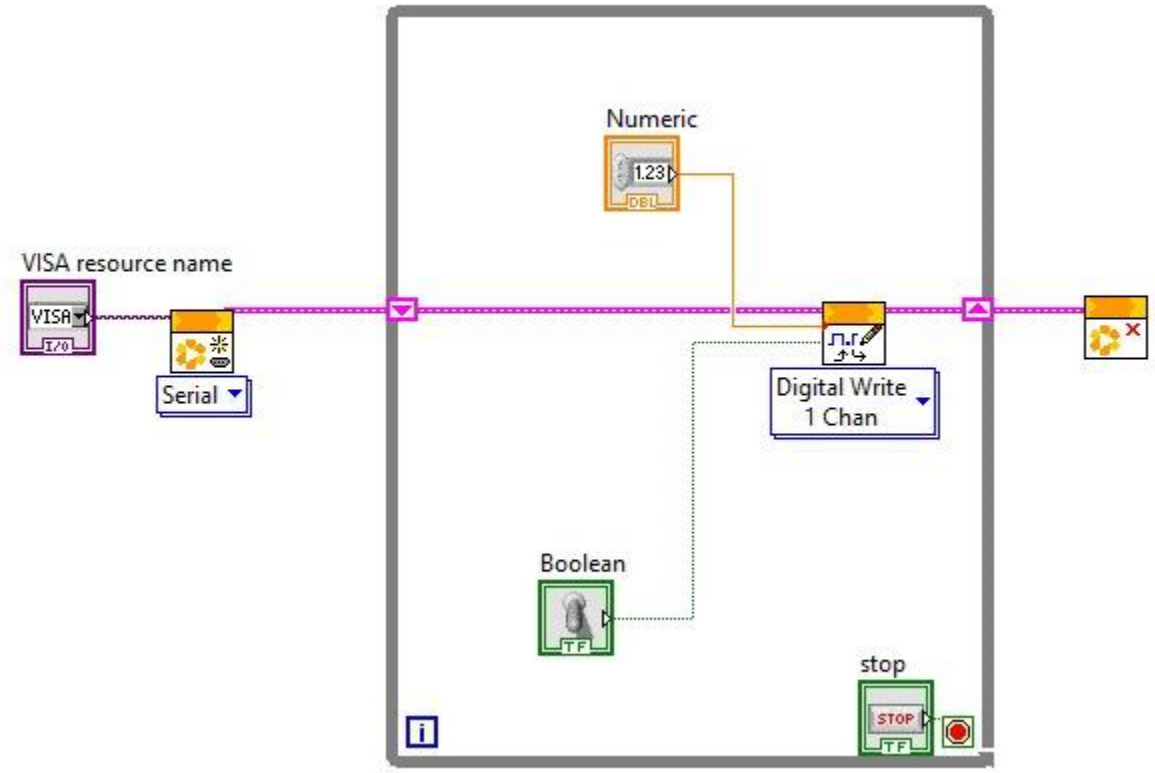

Figure 22 back panel for Read Analog Value configuration

It showed that when there's more light, the LDR's resistance goes down, which makes the voltage reading go lower too. This is a key feature of how LDRs work.

2.2.4 TO DO

The aim of this TODO is to show how to control the brightness of an LED on an Arduino board using a photocell sensor and LabVIEW. The goal is to adjust the LED brightness inversely proportional to the ambient light detected by the photocell.

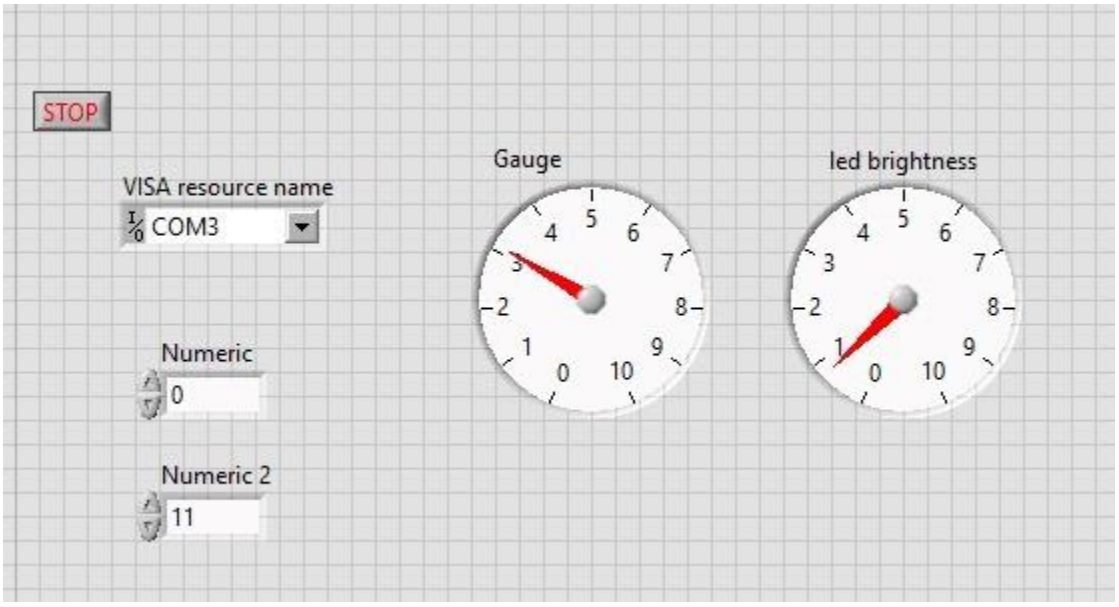

Figure 23 LabVIEW Front panel for the Ambient Light-Controlled LED Brightness.

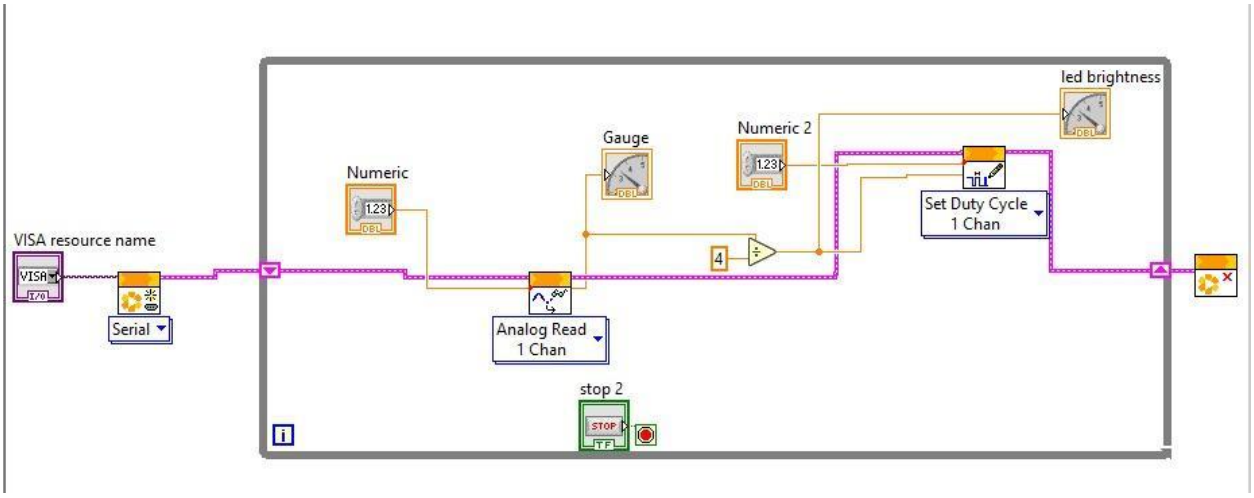

Figure 24 LabVIEW Block Diagram for Ambient Light-Controlled LED Brightness.

The system works by using a photocell sensor connected to an Arduino to detect ambient light levels. When light hits the photocell, it changes the resistance within the sensor, which in turn alters the voltage across it. The Arduino reads this voltage change as an analog input and then sends the information to a LabVIEW via a serial communication link, in the LabVIEW environment, we have a front panel and a block diagram as shown in figure 22 and 23. The front panel is the user interface, where inputs and outputs are displayed, including the LED brightness represented by a gauge. The block diagram is where the program's logic is built, which includes the LINX VIs for interacting with the Arduino, The LabVIEW takes the voltage reading from the photocell. The photocell's voltage reading is subtracted from a constant value , typically the system's maximum readable voltage, in this case, we assume to be 5 volts . This subtraction inverts the relationship, so a higher photocell voltage (indicating lighter) result in a smaller number.

The result of this subtraction is then divided by 4 [after testing and observing the gauge output, we determined that the correct maximum voltage should be 5 volts, but we put it 4 by mistake]. This division scales the value to a range between 0 and 1, which corresponds to a percentage (0% to 100%). This percentage represents the necessary duty cycle for the LED, if there is a lot of ambient light, the photocell's resistance decreases, leading to a higher voltage reading. The system is designed to then decrease the LED's duty cycle, dimming the LED, as less artificial light is needed. Conversely, in a darker environment, the photocell's resistance increases, resulting in a lower voltage reading, and the system increases the LED's duty cycle, making it brighter.

Conclusion

In conclusion, both Experiment #4 and Experiment #5 provided valuable insights into the integration of LabVIEW, LINX, and Arduino for circuit design and control tasks. Experiment #4 determine the effectiveness of LabVIEW in simplifying circuit design without extensive coding, by connecting Arduino and LINX, LabVIEW to enables data acquisition and enhances user experiences through graphical components, experiment #5 showed how we can make Arduino work together with LabVIEW using the LINX Toolkit for various tasks. We used it for different projects, like changing the brightness of an LED based on light around it and managing on/off switches in the program. We also took a specific task where we had to adjust the LED's brightness by measuring light with a sensor and then making the right calculations in the program. This showed us how using computer programs to control physical devices can be both efficient and practical, and how important it is to make precise adjustments to get things working just right.

References

[1] What is LabVIEW? - GeeksforGeeks

[2] Advantages and Disadvantages of LabVIEW - Viewpoint Systems (viewpointusa.com)

[3] Real Time &Interfacing Techniques Lab - Google Drive