I-V CURVE TRACER COMMUNICATION PROTOCOL

An Undergraduate Research Scholars Thesis

by

MARIA KHAN[1] AND SWATI SINGH[2]

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Robert S. Balog

May 2022

Major: Electrical & Computer Engineering[1,2]

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

We, Maria Khan[1] and Swati Singh[2], certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with our Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

I-V Curve Tracer Communication Protocol

Maria Khan[1] and Swati Singh[2]
Department of Electrical & Computer Engineering[1,2]
Texas A&M University

Research Faculty Advisor: Dr. Robert S. Balog
Department of Electrical & Computer Engineering
Texas A&M University

This thesis aims at developing a communication protocol for an I-V Curve Tracer. The IV Curve Tracer consists of a Raspberry Pi as the central unit to which a Source Measuring Unit (SMU) and a Relay Board is connected via USB and Ethernet, respectively. The SMU measures the I-V curve by generating a voltage sweep and measuring the resulting current. The relay board enables testing of multiple specimens. We include an Arduino Mega in this set up to accelerate the communication interface between the Relay modules and the Raspberry Pi. Therefore, our project aims to address the challenge of a scalable, low latency communication between the Pi-Arduino-Relay interface.

Each Arduino is connected to one or multiple relay modules that are used to configure the interconnection between a single measurement device and various combinations of multiple PV cells in a current-voltage curve tracer (IVCT) testbed. This is significant because the time taken to complete a battery of tests using previous methods is over 15 minutes. However, an initial timing analysis found that the majority of the testing interval was spent on the communication to setup the relay configuration, not on the actual measurement process and hence, caused the relay

boards to be slow to respond. This significantly limited the ability of the IVCT to scale-up in number of DUTs and complexity of interconnections tested.

Our project seeks to overcome limitations of the current technology by significantly reducing the packet size of the transmitted data and eliminating the bloated and slow IoT platform. The expected outcome is a lightweight communication platform that is flexible and scalable with fast messaging not just for the IVCT testbed but also suitable for other research laboratory automation and control.

ACKNOWLEDGEMENTS

Contributors

We would like to thank our faculty advisor, Dr. Robert S. Balog, for his guidance and support throughout the course of this research.

Thanks also go to our friends and colleagues and the department faculty and staff for making our time at Texas A&M University a great experience.

Finally, thanks to our family for their encouragement, patience and love.

The experimental data, equipment and setup used for the I-V Curve Tracer Communication Protocol were provided by the REAPER Lab overseen by Dr. Robert S. Balog at Texas A&M at Qatar. The analyses depicted in the I-V Curve Tracer Communication Protocol were conducted in part by Sawsan Shukri and Safin Bayes and were published in 2020.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Our research did not receive any funding.

1 INTRODUCTION

This section gives an overview of our project, the motivation behind it and our proposed solution to existing methods in use.

1.1 Motivation

Our project takes inspiration from the ALOHA protocol. The ALOHA protocol is a system proposed by Norman Abramson in the early 1970s to wirelessly connect various users on the Hawaiian Islands. This protocol is known for its random-access nature and its simplicity. Here, random access means that the needs each of the multiple users utilizing a channel are given equal importance and by simplicity, we are referring to the ability of this protocol to transmit information regardless of the activity of the other terminals. The outline of the communication proposed by the ALOHA protocol can be seen in Figure 1.1. The core principle of ALOHA protocol's bidirectional communication is ensuring complete data transfer with the help of positive and negative acknowledgements between terminals. If the data transmission is successful, the terminal responds with a positive acknowledgement over the channel. The communication is programmed with a maximum number of attempts of transmission and a delay (due to the random waiting time between a new transmission and collisions with other data packets from users) before the terminal responds with a negative acknowledgement, letting users know that the data was not received.

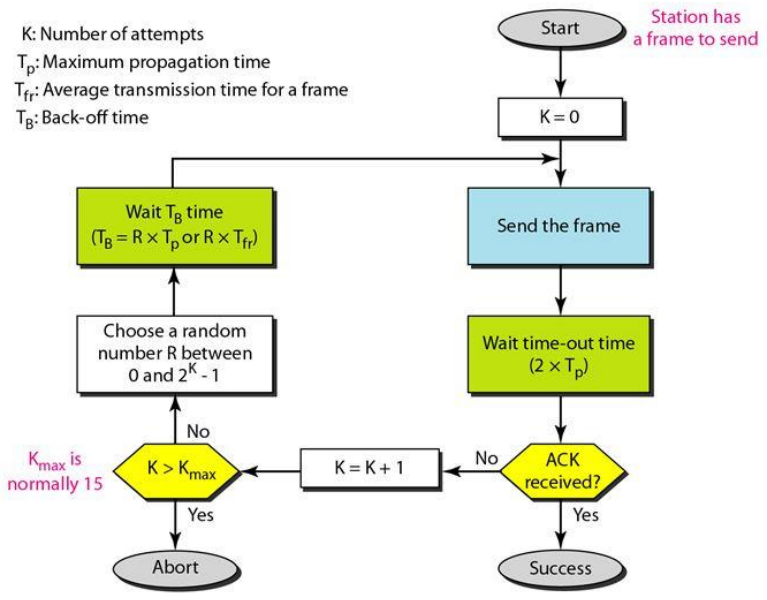

K: Number of attempts
T_p: Maximum propagation time
T_{fr}: Average transmission time for a frame
T_B: Back-off time
Start
Station has a frame to send
K = 0
Send the frame
Wait time-out time ($2 \times T_p$)
ACK received?
No
Yes
Success
K = K + 1
$K > K_{max}$
K_{max} is normally 15
Yes
Abort
No
Choose a random number R between 0 and $2^K - 1$
Wait T_B time ($T_B = R \times T_p$ or $R \times T_{fr}$)

Figure 1.1: Flow Diagram of Information in ALOHA Protocol to Receive Acknowledgement [1]

The positive-negative acknowledgement, simplicity and the random-access nature of the ALOHA protocol are all characteristics we use to implement a lightweight bidirectional communication protocol.

1.2 Equipment Required for Proposed Communication Protocol

Below, we list the major equipment we will be using in order to implement our proposed communication protocol.

1.2.1 Raspberry Pi

Raspberry Pi is mainly used to learn programming skills, build hardware projects, do home automation, and even in industrial applications. It is a low cost, credit-card sized computer that runs Linux. From Figure 1.2 we can see that the Raspberry Pi has multiple serial and IoT ports that are available to us for use. Additionally, It provides a set of GPIO (general purpose input/output) pins, that allow you to control electronic components for physical computing and explore the Internet of Things (IoT) environment.

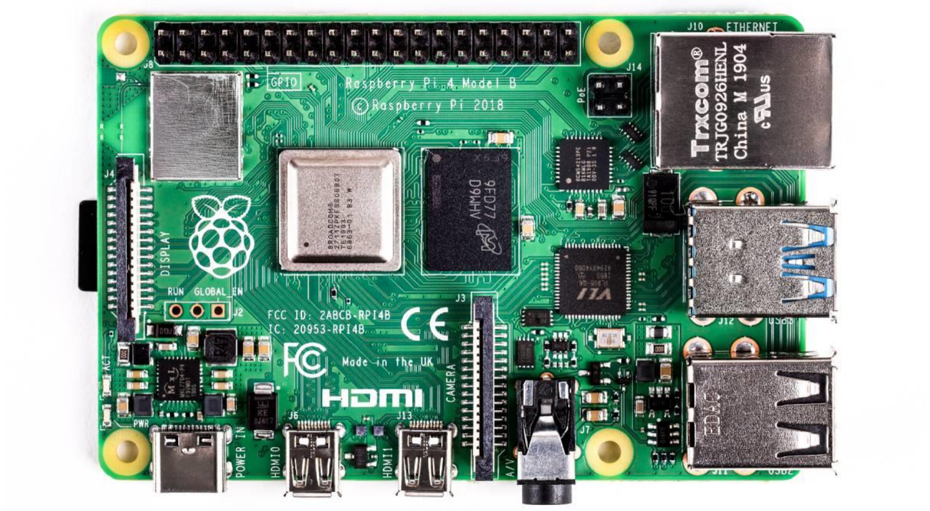


Figure 1.2: Raspberry Pi 4 Board [2]

1.2.2 Arduino Board

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards can read inputs like a light on a sensor, finger on a button, or a Twitter message - and turn it into an output i.e. - activating a motor, turning on an LED, or publishing something online. As we can see in Figure 1.3, the Arduino Boards is a small board that adds to its portability features alone with a wide variety of function packed onto a single chip. For this project, An Arduino Mega will be used to turn specified inputs into the required outputs of the project.

The key Arduino Mega features include 54 Digital Input Output Pins and 16 Analog Input Pins.

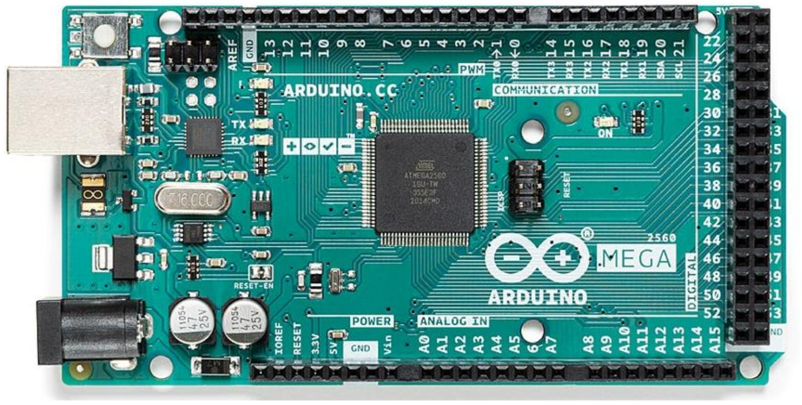


Figure 1.3: Arduino Mega Board [3]

1.2.2.1 Serial Communication

Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART). It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot simultaneously use pins 0 and 1 for digital input or output. The Arduino's built in serial monitor can be used to communicate with the board if there is no external device connected to it.

1.2.2.2 Ethernet Communication

Ethernet communication (between local and internet) is established via the Arduino Ethernet Shield found within the built in ethernet library in Arduino. This library provides both Client and server functionalities which will be used to switch to ethernet communication. The Arduino board communicates with the ethernet shield using the SPI bus. This is on digital pins 11, 12, and 13 on the Uno and pins 50, 51, and 52 on the Mega. On both boards, pin 10 is used as SS. On the Mega, the hardware SS pin, 53, is not used to select the Ethernet controller chip, but it must be kept as an output, or the SPI interface won't work. It is important to ensure that all these pins are free and not being used as regular digital I/O’s.

1.2.3 12-V 16 Channel Relay Board

A 16-channel relay board requires 12 V of input power to carry out its required functions. It can be used to control various appliances and equipment with large currents. For this particular project, they are used to control the PV modules. The Relay Board is controlled by a microcontroller such as an Arduino or a Raspberry Pi. For this project, the relay board is connected to the Arduino. Each pin on the Relay Board is individually controlled by the pin it is connected to on the Arduino. There are two types of relays boards:

1. Low-Level Triggered: A low level trigger type relay board will allow current to go through the power line when the voltage is below a certain level.
2. High-Level Triggered: A high level trigger type relay board will allow current to go through the power line only when the voltage is higher than a certain level.

For this project, a Low-Level trigger type relay board, as can be seen in Figure 1.4 will be used.

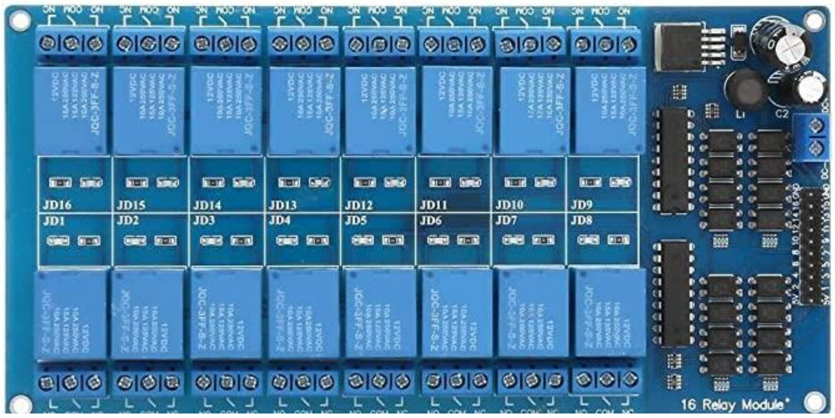


Figure 1.4: 12V 16 Channel Relay Board [4]

1.3 Project Setup

Our setup is centered around an I-V Curve Tracer, which is a testbed that collects data and plots the I-V curve of energy received by solar panels in order to remain in a useful MPPT (maximum power point tracking) range and checks if the solar panel is extracting the maximum voltage possible efficiently. The IVCT consists of a Raspberry Pi as the central unit to which a Source Measuring Unit (SMU) and a Relay Board is connected via USB and Ethernet, respectively. The SMU measures the I-V curve by generating a voltage sweep and measuring the resulting current. These data points that are collected are then plotted and optimized using the central hub to create the I-V graph we need to analyze the efficiency of the PV modules that are in use. These graphs help us identify the PV modules that are not within a useful MPPT range and therefore, allow the user to be able to recognize which modules can be turned off to preserve energy. Thus, once a user is able to get a data swatch of the sun's movements and identify which PV cells should remain on or off on average, the process of switching on and off the relays can be read from an Excel sheet (which can always be edited by the user in case of changes in weather, environment, new data sets, and so on) and automated by the central hub.

Our setup also includes multiple 16-channel relay boards. Each relay controls a PV module. Thus, the relay boards enable testing of multiple specimens. We include an Arduino Mega in this set up to accelerate the communication interface between the relay modules and the Raspberry Pi. The relays boards are controlled directly by the Arduino Megas which receive the relay configurations to be switched on from the Raspberry Pi in the form of an Excel sheet. Therefore, our project aims to address the challenge of a scalable, low latency communication between the Pi-Arduino-Relay interface. Each Arduino is connected to one or multiple relay modules that are used to configure the interconnection between a single measurement device and various combinations of multiple PV cells in a current-voltage curve tracer (IVCT) testbed. This is significant because the time taken to complete a battery of tests using previous methods is over 15 minutes. Since the sun moves 15 degrees across the sky per hour on average, this means that the environmental conditions change during the test, which makes it impossible to perform a quantitative comparative analysis of the various configurations under testing.

1.4 Previous Solutions

The previous research and application used an ethernet controller hosting a webpage interface that were provided by the manufacture as an internet of things (IoT) solution as can be seen in Figure 1.5. The webpage allows the user to test various configurations of PV cells by manually switching on the required relays.

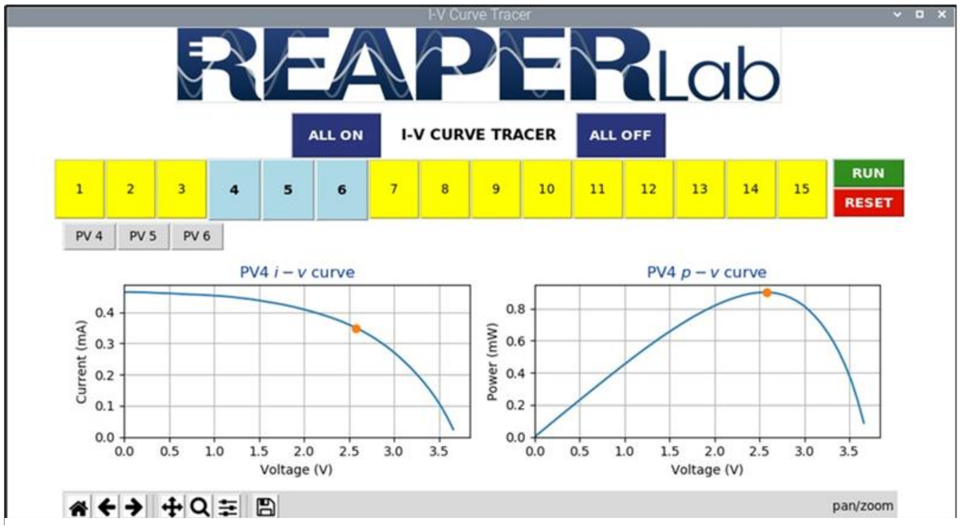


Figure 1.5:Layout for Website Used for Previous IVCT Setup to Manually Control PV Cell Condition

However, an initial timing analysis using the previous setup that is seen in Figure 1.6, found that the majority of the testing interval was spent on the communication to setup up the relay configuration, not on the actual measurement process. When switching from commands for one relay board to another, the system would require reading a new address character by character, which caused a large delay in the response of the relays and therefore made them slow to respond. This significantly limited the ability of the IVCT to scale-up in number of DUTs and complexity of interconnections tested. This is undesirable since the more the PV modules added, the more relay boards would be required and therefore, the more the delay between switching from one relay to the other, causing an unacceptable amount of time taken for one batch of tests to be conducted.

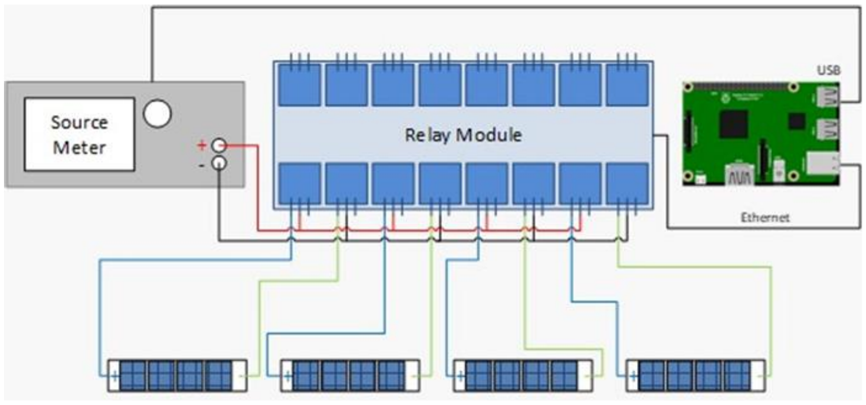


Figure 1.6: Overview of the Previous IVCT Setup

1.5 Proposed Solution

Our communication protocol establishes bidirectional communication via ethernet between the Raspberry Pi and the Arduino Mega. Ethernet is enabled on the Arduino with the help of an Ethernet shield that is compatible with the Arduino. The data is transmitted using a client-server communication model based on ethernet where the Raspberry Pi is our client, and the Arduino plays the role of the server. Each of the devices involved in the communication process are programmed to transmit data and receive an acknowledgement. If data is not received or if the packet received is incomplete, a negative acknowledgement is sent back. Once the payload is received by the Arduino it sends the Raspberry Pi. The ethernet aspect of this project allows us to have a stand-alone setup where the Arduino does not need to be serially connected to an external PC or the Raspberry Pi. Furthermore, it allows us to make a project more scalable by adding multiple Arduinos and Relay boards by modifying only the IP Addresses and including an ethernet adapter.

Our project seeks to overcome limitations of the current technology by significantly reducing the packet size of the transmitted data and eliminating the bloated and slow IoT platform. We accomplish this by using a microcontroller that utilizes digital I/O's and hence deals with binary (lightweight) data to control the relays. The expected outcome is a low latency communication platform that is flexible and scalable with fast messaging for the IVCT testbed. Further, it can be suitable for other research laboratory automation and control since the protocol is made as general use as possible and therefore, can be modified based on user specifications according to their needs. Some possible uses could be for home automation systems, security systems, etc.

2 METHODS

In the following section, we explain the implementation of our methodology.

2.1 Hardware Setup

This section covers all the hardware implementations for our project setup. As we can see in Figure 2.1, our hardware setup consists of a Raspberry Pi which is powered by a 5V power supply. The Pi is controlled using a keyboard and mouse connected to it via USB. The Raspberry Pi is then connected to an Arduino Mega and Ethernet shield stack, which is fitted with a printed circuit board (PCB). The PCB board is used to connect to 2 16-channel relay boards that will control our PV cells.

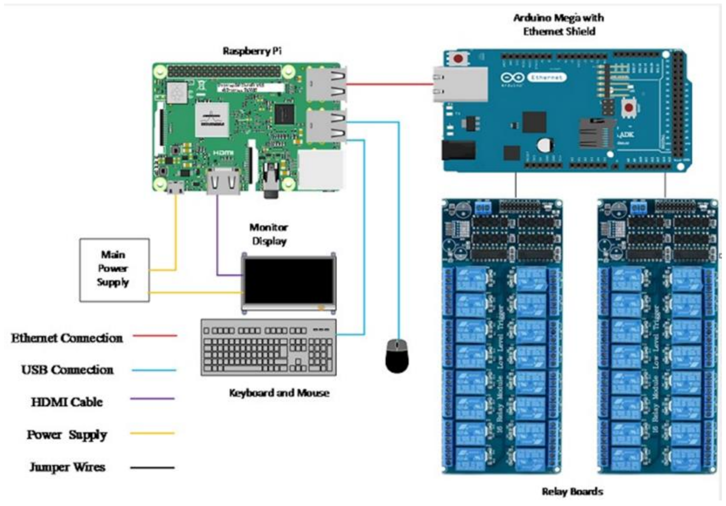


Figure 2.1: Hardware Setup Overview of Ethernet Communication Protocol

2.1.1 Raspberry Pi

To set up the Raspberry Pi, we fixed the Raspberry Pi board on the transparent case and inserted the SD card in the SD card port. Then, we plugged in the mouse and keyboard to the Raspberry Pi's USB ports and connected the display to the Pi using the micro-HDMI. Finally, we plugged in the power supply cable to the Raspberry Pi's supply port and turned the Raspberry Pi on.

2.1.2 Arduino Mega with Ethernet Shield

The hardware set-up for the Arduino Mega is much simpler than that of the Raspberry Pi. To set up the required system, we plug the Arduino Board into the Raspberry Pi via a USB cable (which also acts as its power source). An ethernet shield is mounted on top of the Arduino Mega. This shield is connected to the Raspberry Pi via an ethernet cable to the Ethernet RJ45 port.

It is also important to remember that for ethernet communication, digital input-output pins 50, 51, 52 and 53 will be in use for the SPI bus. Hence, these pins cannot be used as regular digital I/O's for the relays. Since the project requires at least 32 available pins to connect two 16 channel relay boards to, the Arduino Mega is capable of facilitating the required number of digital I/O's.

2.1.3 PCB Design

We designed a PCB Board to designate the pins we need for our project since there is a surplus of available DIO's. It also increases the convenience factor by avoiding any bends or twists in the ribbon cables which we are using to connect the relays to the Mega. The PCB Board includes a terminal of 20 header pins for each relay board (which includes 16 DIO's , 2 pins for 5V power, and 2 pins for GND).

For terminal 1 and terminal 2, we need to accommodate 20 pins each, the schematic of which is shown in Figure 2.2 and the pinouts of which are shown in Table 2.1 and Table 2.2:

- 16 relays
- 2 ground pins
- Two 5V power pins

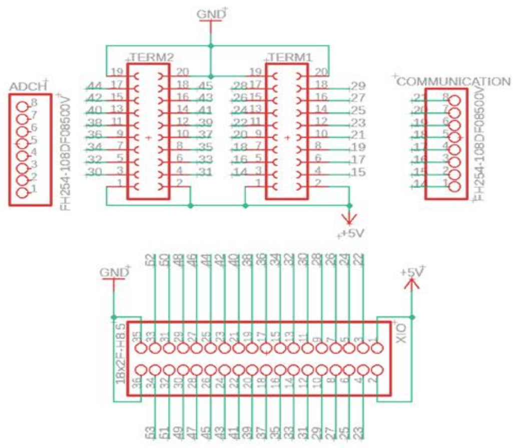


Figure 2.2: PCB Schematic

Table 2.1: Pins on Terminal 1 Corresponding to I/O's on the Mega for Relay Board 1

Pin on Terminal 1	DIO's on Arduino Mega
1,2	5V
3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18	14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
19,20	Ground

Table 2.2: Pins on Terminal 2 Corresponding to I/O's on the Mega for Relay Board 2

Pin on Terminal 2	DIO's on Arduino Mega
1,2	5V
3,4,5,6,7,8,9,10,11,12,13,14,15,16,17	30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45
19,20	Ground

2.1.4 Relay Boards

Ribbon cables are used to connect the 20 pin headers from the PCB to the relay boards via the pinouts shown in Table 2.1 and Table 2.2. The Relay Board is connected to a 12 V power supply via jumper cables. One cable is connected to the positive 12V supply and the second cable is connected to ground, to power up the relay board.

2.2 Software Setup

Below, we cover the software setup for the individual components that require it for our communication protocol.

2.2.1 Raspberry Pi

Once the hardware set up for the Raspberry Pi is complete, we then move on to software installations. A list of the downloadable OS's pop up upon powering up the Raspberry Pi. We selected and installed the Raspbian OS. The installation process took around 10 – 15 minutes. After installation, we were prompted to follow a wizard to set up location, time zone, password, and software update, which we followed to finish installation. Once the installation procedure is completed, the Raspberry Pi is ready to be programmed upon.

2.2.2 Arduino Mega

The software that we used for programming the Arduino was also installed on the Raspberry Pi, since the Arduino does not have a separate display of its own. In order to install the Arduino IDE onto the Pi, we first connected the Pi to the internet. After this, we went to the official Arduino website and downloaded the IDE that corresponded to version 1.8.13 and selected the Linux ARM 32 bits download option. We downloaded and extracted this IDE onto the Pi.

The Arduino IDE has two loops for programming the board. The first one is the set-up loop that contains all the code that only needs to be executed once, such as initializing variables and digital pins. The second is the main loop where the code that needs to be executed repeatedly is written i.e., it is an infinite loop.

2.2.3 Relay Boards

The states of the relays were programmed using the Arduino IDE. For our project, we used low-level triggered relay boards (which means HIGH turns the relay OFF and LOW turns the relay ON). The pinouts corresponding to the relays were initialized as outputs within the Arduino IDE.

2.3 Building the Communication Protocol

In the following section, we define the theory and implementation of how a client-server model is used to communicate between two devices over LAN.

2.3.1 Client-Server Communication Protocol

Client-server communication involves two components, namely a client and a server. The clients send requests to the server and the server responds to the client requests. The method of client server communication that is utilized is via sockets. Sockets facilitate communication between two processes on the same machine or different machines. They are used in a client-server framework and consist of the IP address and port number. Many application protocols use sockets for data connection and data transfer between a client and a server. Sockets only transfer an unstructured byte stream across processes [5]. The structure on the byte stream is imposed by the client and server applications [5]. Figure 2.3 summarizes the client-server relationship between two devices.

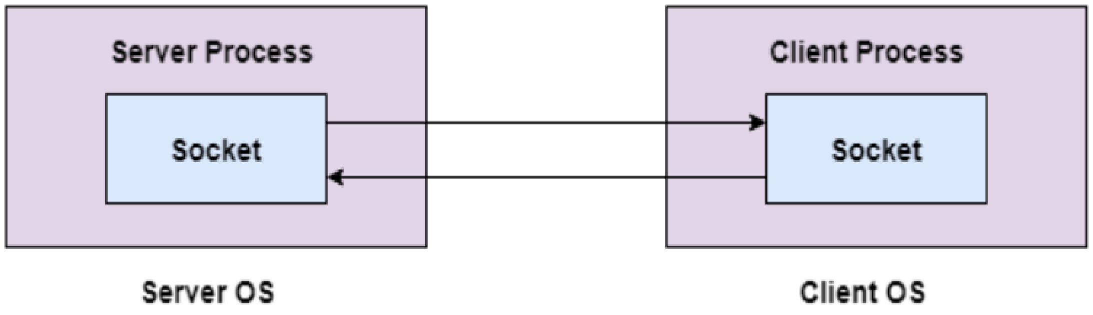


Figure 2.3: Raspberry Pi (Client) and Arduino Mega (Server) Connected via Ethernet [5]

2.3.2 *Building the Raspberry Pi as a Client*

To program the Raspberry Pi as the client, socket programming was used which is available in Python through the socket module. Once the required modules were imported, the client was built.

1. The socket library in Python, which is a way of connecting two nodes on a network to communicate with each other, was imported.
2. From the socket library, we imported the "select" module. The arguments to `select()` are three lists containing communication channels to monitor.
3. Next, we defined a static IP Address for the Raspberry Pi (which is going to be the same for the Arduino to establish communication) and specified the port for ethernet communication.
4. The `connect()` method of Python's socket module, connected a TCP (Transmission Control Protocol) based client socket (the Raspberry Pi) to a TCP based server socket (Arduino Mega). Once the connection was made, data could be sent and received.

2.3.3 Building the Arduino as a Server

The Arduino acted as the server by receiving requests from the client and then performing the actions required as a response to the request. The Arduino as a server is built using the Ethernet Libraries that allowed communication via LAN to be established.

The first module imported is the `<SPI.h>`. This library allows you to communicate with SPI devices, with the Arduino as the master device. Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

The second Ethernet module that is imported is the `<Ethernet.h>`. This library is designed to work with the Arduino shield. It allows an Arduino port to connect to the internet and serve as either a server accepting incoming connections or a client making outgoing ones.

The Arduino Mega communicates with the Shield using the SPI bus. We then defined the MAC Address for the Ethernet Shield and the Static IP Address and specified which port was to be used as a server.

Next, in the set-up loop of the Arduino IDE, we initialized serial communication and set our baud rate (which is the rate of data transfer). This is done to upload our communication protocol code onto the Arduino Mega where it will be saved on the board. Therefore, once the serial communication that uploaded code onto the Arduino’s chip memory is terminated, it can continue carrying out commands using Ethernet. Therefore, we started Ethernet communication and told the server to start listening for incoming connections. In the main loop, we read the incoming message.

2.3.4 Using Hexadecimal Payloads in Ethernet Communication

A communication protocol is built that will be used to convey relay numbers from the Pi to the Arduino and then performing the respective action by the Arduino on the relays. We use a hexadecimal payload to transmit the states of the relays to the Arduino to reduce packet size since each character in a hexadecimal payload will contain the states of 4 relays. This helps reduce latency of the data transfer protocol. Below, we have shown how we execute this type of payload on a single 16 channel relay board. For the scope of our project, where one Arduino Mega controls two 16-channel relay boards, we expanded the idea below to a 8-bit hexadecimal payload that will contain the states of 32 relays (instead of the 4-bit payload that represents 16 relay states as shown).

2.3.4.1 Structure of Hexadecimal Payload

A hexadecimal payload is used to convey the relay numbers that are being targeted, to the Arduino. The raspberry Pi will have a Python script which will generate a 5-digit hexadecimal payload that will then be communicated to the Arduino. This section will describe the details of this payload and how it is generated.

To understand the format of the payload that is being used, the range of the hexadecimal representation is important. Since there are 16 relays, consider each relay to be a binary digit (0 or 1). Therefore, a 16-digit binary representation will be converted to a 4-digit hexadecimal representation ranging from 0000-FFFF i.e., all 0’s (all relays off) and all 1’s (all relays on). These 4 hexadecimal digits will constitute the status of the relays.

We wrote a Python script that asked the user to input the relay number they want to turn on (the relays not mentioned will be assumed to be off). Once the user has entered the relays they want to target, the Python script will generate a binary representation that signifies the user input.

This binary representation is then converted to a 4-digit hexadecimal representation which was communicated to the Arduino. The following steps show the detailed procedure of how the payload was generated.

2.3.4.2 Communication of Hexadecimal Payload from Raspberry Pi to Arduino Mega

This hexadecimal payload was communicated to the Arduino in the form of a string. The Python script generates the payload as a type string. The initialization and reception of the payload is incorporated on both the Python script and the Arduino sketch. They will be used to turn on/off the relays. The method of communication is done via ethernet using UDP objects. The Raspberry Pi is our client. This is done in the following way:

1. We imported the necessary modules and defined the address to which we were sending data to. The parameters of this address are the IP address and the defined port matching the ones defined on the Arduino Mega.
2. We then set up the socket for socket programming.
3. Next, we generated the hexadecimal payload based on user input.
4. The communication of the hexadecimal payloads was done to the address using the command `client_socket.sendto()`. The parameters of this command are the data to be sent and the address we sent the data to.

2.3.5 Receiving Acknowledgment from Arduino (Reception of Payload)

Once the payload was transmitted, wait for an acknowledgement from the Arduino. Once the data has been received, we receive positive acknowledgement on the terminal of the Raspberry Pi. The acknowledgement was done in the form of a message sent back to the Pi with an echo of the data it received i.e., the Arduino recited the payload back to the Pi to ensure that the full payload was communicated and there was no error. This positive-negative acknowledgement on both sides of the communication protocol was inspired by the working of the ALOHA Protocol. Here, the Arduino was our server.

1. We imported the Ethernet libraries required and defined the static IP and MAC addresses of the Arduino.
2. We initialized a UDP object needed for data transfer.
3. We defined the output pins on the Arduino and set their initial states to “OFF”.
4. We began the Ethernet and UDP communications.
5. We checked the availability over the specified port in the address. If data is available (payload has been transmitted), we stored it into the character array and converted the packet buffer into a string.
6. This hexadecimal number was sent back to the client as a response and then printed.
7. When the message has been sent back to the client, the Arduino code converted the hexadecimal payload to its corresponding binary representation which was then used to switch on and off the relays (which we will see in the following section).
8. The corresponding binary representation found after conversion was also sent back to the client to be printed.

2.3.6 Conversion of Received Hexadecimal Payload to Binary on Arduino IDE

Once the Arduino received the payload, this payload needed to be analyzed and then changed back to its original binary representation. This representation was then used to turn on and off the relays. The steps to convert the payload to its corresponding binary representation are as follows:

1. Wrote a loop that allowed us to iterate through the entire payload and access each character individually. The loop ensured that the payload would be analyzed digit by digit and converted to its corresponding binary representation, with the help of embedded loops that resembled a switch case.
2. Once each digit had been converted to binary, we concatenated the binary strings together to form the final binary representation.
3. For consecutive payloads, the binary variable had the previous representation stored, which was solved with string slicing. This allowed us to always take the last 16 digits of the binary representation, so we were always considering the most updated user inputs.

2.3.7 Using Binary Representation to Switch On/Off Respective Relays

Once we have converted the hexadecimal representation to binary on the Arduino's side, we needed to switch on and off the respective relays. This was done by carrying out the following steps:

1. We iterated through the binary representation string and used an if-else statement within this loop to set a condition to check whether the relay corresponding to the character being accessed in the binary representation was 0 or 1.
2. If the relay was set to 0, we used the "`digitalWrite()`" command to set this relay to HIGH (which is OFF for a low-level triggered relay board)
3. If the relay was set to 1, we used the "`digitalWrite()`" command to set this relay to LOW (which is ON for a low-level triggered relay board)

An example using our Ethernet communication protocol is summarized below in Table 2.3.

Table 2.3:Using Hexadecimal Payloads to Control Relay States

User Input (to Pi)	Hexadecimal Payload	Binary Representation
Turn on relays 1,10,12,16	8A01	1000101000000001
Turn off all relays	0000	0000000000000000
Turn on all relays	FFFF	1111111111111111
Turn on relays 2,5,6,9,10,11,13,14,15,16	F732	1111011100110010

2.4 Scaling the Communication Protocol

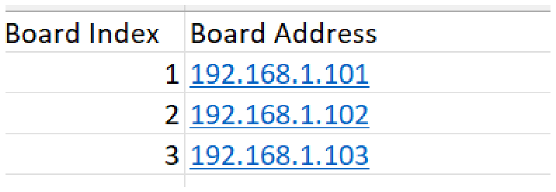
Up to now, we have defined a lightweight communication protocol. This is important for a scaled version of this project, since in reality, there may be hundreds if not thousands of PV cells to be controlled that need to be turned on or off on command as quickly and efficiently as possible. We decided to switch to our Raspberry Pi reading Excel files with all the information about relay states in it. The advantage of doing this is twofold: firstly, it allows for scaling in a much more simple way (since the Pi will be able to read as many new inputs as required appended in the Excel file created) and reduces the dependency on human input and thus, minimizes the chances of human error, especially when it would come to hundreds of relay states (since making small edits to an Excel file will be much quicker). In this section, we will define how we accomplish switching from user input to the Pi reading Excel files.

2.4.1 Using Companion Files

A companion file in Arduino can be created to pound define variables that will be used in the main block of code i.e., the communication protocol. For our program, we create a companion file to pound define the IP Addresses which the communication is taking place over. This allows the user to manually edit the IP Address in a separate text file (the companion file) rather than having to make changes to the original program. The advantage of making a separate companion is to make your original program as general and versatile as possible and any specifications required are put into the companion file.

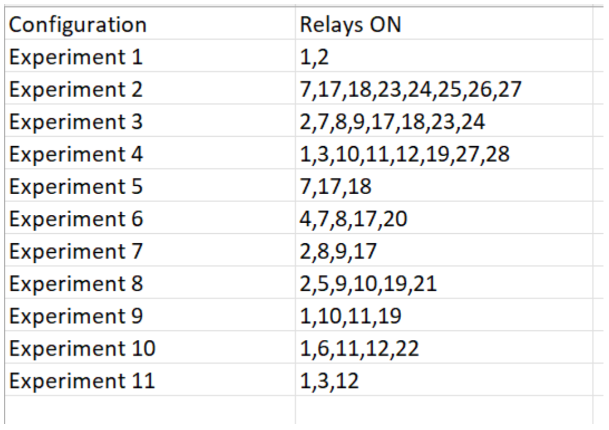
2.4.2 Reading Excel Files for Relay Configurations

In order to minimize modification in our main block of code, we use Excel files to feed the communication data and relay configurations to the program. The Excel file that we read has 2 sheets, one with the data for establishing communication (IP Addresses) as shown in Figure 2.4 and another sheet with the relay configurations as shown in Figure 2.5.



Board Index	Board Address
1	192.168.1.101
2	192.168.1.102
3	192.168.1.103

Figure 2.4: Sheet 1 of Excel File



Configuration	Relays ON
Experiment 1	1,2
Experiment 2	7,17,18,23,24,25,26,27
Experiment 3	2,7,8,9,17,18,23,24
Experiment 4	1,3,10,11,12,19,27,28
Experiment 5	7,17,18
Experiment 6	4,7,8,17,20
Experiment 7	2,8,9,17
Experiment 8	2,5,9,10,19,21
Experiment 9	1,10,11,19
Experiment 10	1,6,11,12,22
Experiment 11	1,3,12

Figure 2.5: Sheet 2 of Excel File

For the first sheet, we manually assigned an IP Address for each Arduino Mega Board (each board facilitates the working of 2 relay boards). This helps our objective of attaining scalability since we can add as many Arduino Mega's to the system as needed as long as we assign an IP Address to communicate over.

For the sheet with the relay configurations, we defined the experiments we want to run and the corresponding relays which need to be turned on. The relay numbers that are accepted by the program lie between 1 and 32 since each Mega controls 2 relays board i.e., 32 total relays. These experiments were read and run in the order they were defined.

In our code, we also had a mapping that allowed the program to choose the IP Address corresponding to the correct Arduino Mega. For example, if we have 2 Mega's connected to the system, we can control 64 relays. Relays 1-32 will use the IP Address defined for Board 1, relays 33-64 will use the IP Address defined for Board 2.

3 RESULTS

3.1 Using Hexadecimal Payloads to Control Relay States

Our communication implemented a 4-bit hexadecimal payload that was transmitted from the Raspberry Pi to the Arduino Mega, to control a single 16-channel relay board. This 4-bit payload was converted to its corresponding 16-bit binary representation and the corresponding relays were turned on and off. We accomplished this process of data transfer with reduced latency by first prompting the user to input the relays that were to be switched on, via a Python program executed on the Raspberry Pi. This Python program then generated a 16-bit binary representation that contained the states of these relays (1 signifies the relay is ON and 0 signifies that the relay is OFF). The binary representation was then converted to its corresponding 4-bit hexadecimal representation before being transmitted to the Arduino Mega, via the communication channel that is set up. Once the Arduino Mega received this payload it sent an acknowledgement back to the Raspberry Pi with an echo of the payload received to ensure no data had been lost or corrupted during the transmission. This payload was then analyzed by the Arduino and each bit of the hexadecimal payload was converted to its corresponding binary representation. The binary representation was then utilized to switch on or off the relays by setting the output of the pins as HIGH or LOW. The above explained methodology is demonstrated using the following table of commands and payloads:

Table 3.1: Using Hexadecimal Payloads to Control Relay States

User Input (to Raspberry Pi)	Hexadecimal Payload	Binary Representation
Turn on relays 1,10,12,16	8A01	1000101000000001
Turn off all relays	0000	0000000000000000
Turn on all relays	FFFF	1111111111111111
Turn on relays 2,5,6,9,10,1113,14,15,16	F732	1111011100110010

Table 3.1 holds the relay numbers that need to be turned on, and the corresponding hexadecimal and binary representations. The outputs for each of the above experiments are shown below in Figures 3.1-3.4:

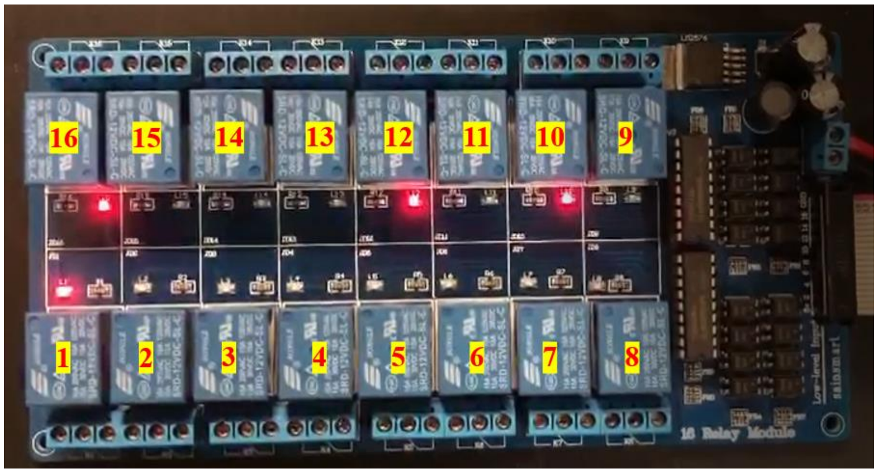


Figure 3.1: Relay Output for a Hexadecimal Payload of 8A01

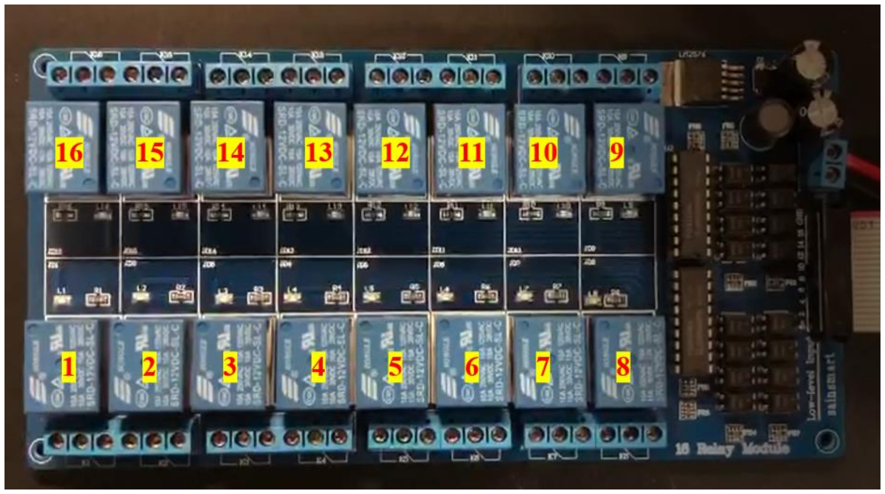


Figure 3.2: Relay Output for a Hexadecimal Payload of 0000

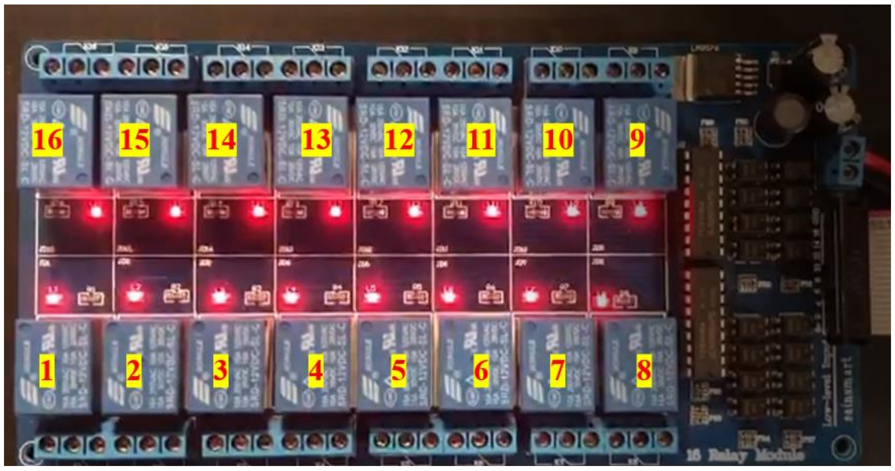


Figure 3.3: Relay Output for Hexadecimal Payload of FFFF

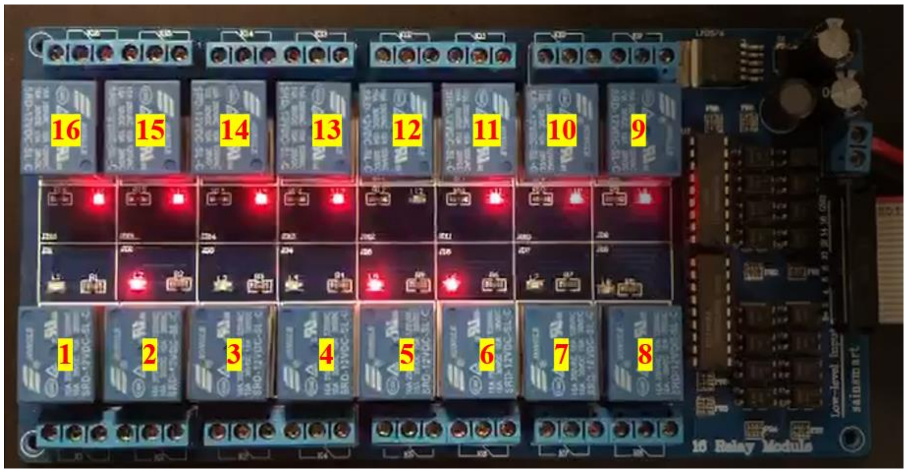
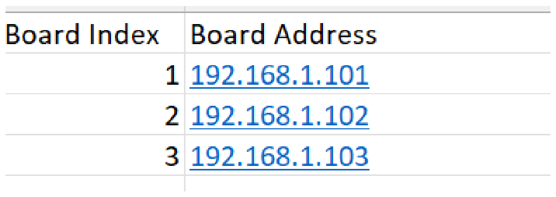


Figure 3.4: Relay Output for Hexadecimal Payload of F732

3.2 Using Excel File to Generate Hexadecimal Payload

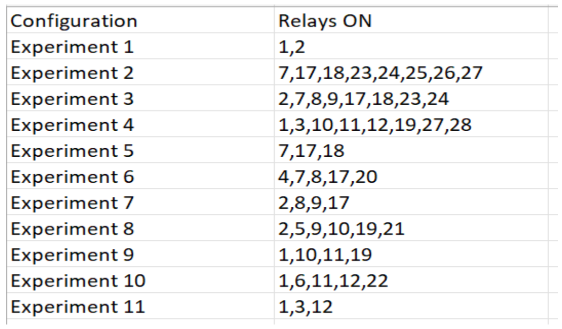
For the scope of our project, we expanded on our previous result of utilizing a 4-bit hexadecimal payload for a single 16-channel relay board by connecting two 16-channel relay boards to an Arduino Mega i.e., one Arduino Mega was able to control up to 32 relays. Every relay board that is added to the system adds 4 hexadecimal characters to the payload. 2 relays boards require an 8-character hexadecimal payload which will contain the states of 32 relays.

Furthermore, we modified our communication protocol to minimize human dependency, by writing a Python program that reads an excel file containing the states of the relays and the IP Addresses of the Arduino Mega's connected to the Raspberry Pi. This information can be modified directly in the Excel file which allows the user to make changes without having to directly interfere with the code. The Excel file consisted of two sheets. The first sheet had the IP Addresses of each of the Mega's that were connected to the Raspberry Pi. The second sheet consisted of the list of experiments to be run and the relays that were to be turned in the form of a comma separated list. The python program reads this Excel file and sequentially runs each experiment. For every experiment, the relays to be switched on were extracted from the file and the corresponding 32-bit binary representation was generated by the Python program. This binary representation was then converted to its corresponding 8-bit hexadecimal payload. Once the payload was generated, the IP addresses of the Mega's that the relays were mapped to (relays 1-32 correspond to board 1, relays 32-64 correspond to board 2 and so on) were used to determine which Arduino the payload needs to be transmitted to as can be seen in Figure 3.5. The payload was then transmitted to the respective Arduino Mega's, where it was converted back to its binary representation. The binary representation was then used to turn on and off the required relays. The following example was used to demonstrate how the Excel files were used to control the states of the relays:



Board Index	Board Address
1	192.168.1.101
2	192.168.1.102
3	192.168.1.103

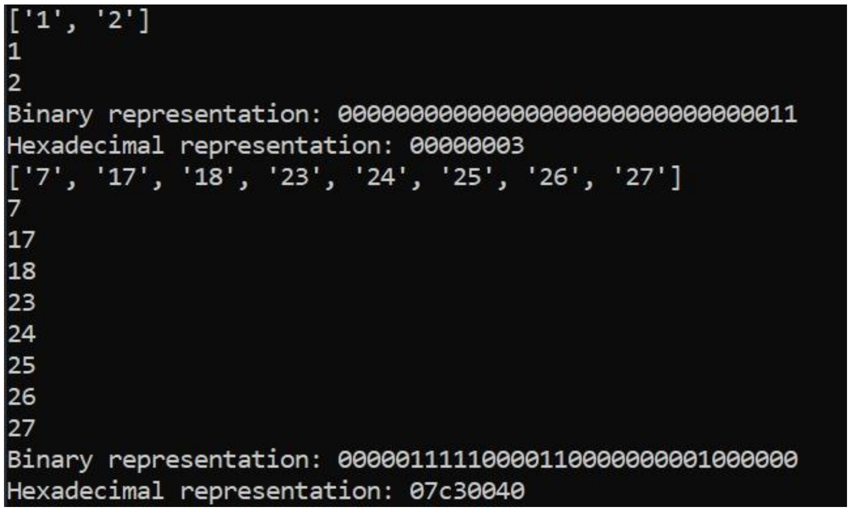
Figure 3.5: Sheet 1 of Excel File



Configuration	Relays ON
Experiment 1	1,2
Experiment 2	7,17,18,23,24,25,26,27
Experiment 3	2,7,8,9,17,18,23,24
Experiment 4	1,3,10,11,12,19,27,28
Experiment 5	7,17,18
Experiment 6	4,7,8,17,20
Experiment 7	2,8,9,17
Experiment 8	2,5,9,10,19,21
Experiment 9	1,10,11,19
Experiment 10	1,6,11,12,22
Experiment 11	1,3,12

Figure 3.6: Sheet 2 of Excel File

The outputs of experiment 1 and 2 from Figure 3.6 is shown below in Figure 3.7:



```
['1', '2']
1
2
Binary representation: 00000000000000000000000000000011
Hexadecimal representation: 00000003
['7', '17', '18', '23', '24', '25', '26', '27']
7
17
18
23
24
25
26
27
Binary representation: 00000111110000110000000001000000
Hexadecimal representation: 07c30040
```

Figure 3.7: Output of Python Program that Reads the Excel File

From the figure above, we can see that the experiments are run in order. First the relays to be turned on are extracted using which the corresponding 32-bit binary representation is generated. This binary representation is then converted to its corresponding 8-bit hexadecimal payload which was transmitted to the Arduino Mega's. Once the Arduino board received the payload, it converted it back to its binary representation using which the respective relays were turned on as can be seen in Figure 3.8 and Figure 3.9.

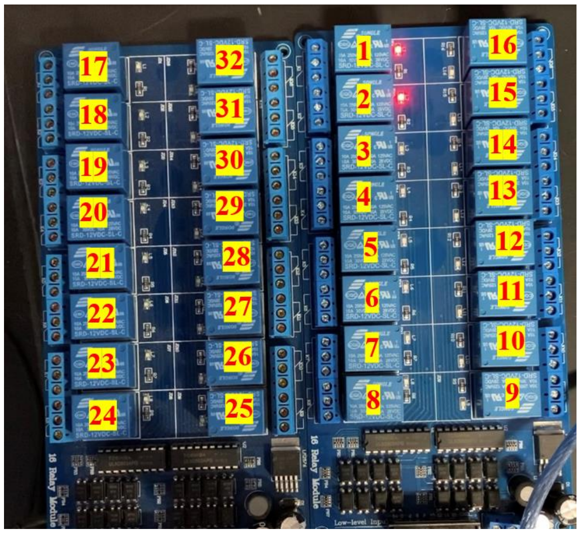


Figure 3.8: Output on Relays for Experiment 1

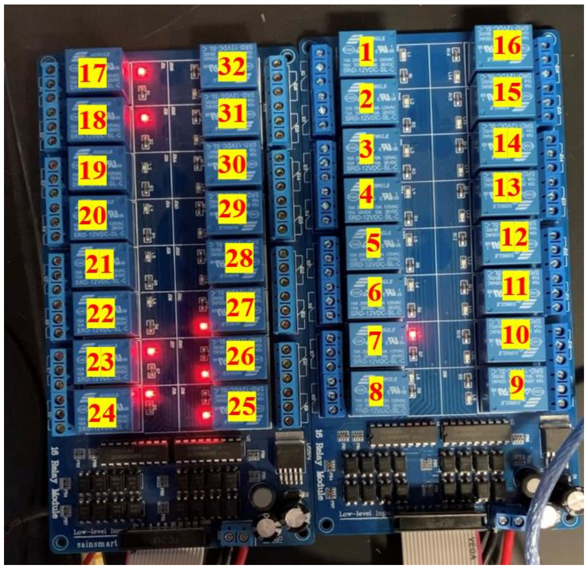


Figure 3.9: Output on Relays for experiment 2

4 CONCLUSION

In conclusion, the aim of our project was to create a low latency communication protocol to facilitate bidirectional communication between a Raspberry Pi and an Arduino Mega over Ethernet. This protocol is further significant since it can be used to reduce latency in Ethernet communication between any two devices. We accomplished reducing the latency by encoding the payload into smaller packets of data that is transmitted which is then decoded on the receiver side in order to carry out the instructions being assigned by the master device. To ensure that the data transferred is complete we use the concept of a positive and negative acknowledgement which is sent back to the transmitter as an echo. To summarize, our communication protocol targeted the core latency issue with the previously existing communication protocol for the curve tracer and found alternative that significantly cut down on the communication time so we can perform a sweep of tests within a shorter span of time, and hence achieve uniform testing conditions and results for the PV cells.

REFERENCES

[1] Myreadingroom.co.in. 2022. Aloha Protocols. [online] Available at: <http://www.myreadingroom.co.in/images/stories/docs/dcn/aloha%20Protocols_Pure%20aloha%20strategy.JPG> [Accessed 18 January 2022].

[2] NDTV Gadgets 360. n.d. Raspberry Pi 4 With 3x Faster CPU, Dual 4K Display Outputs Launched. [online] Available at: <https://gadgets.ndtv.com/laptops/news/raspberry-pi-4-model-b-price-in-india-launched-faster-cpu-2058315> [Accessed 22 January 2022].

[3] Arduino Online Shop. n.d. *Arduino Mega 2560 Rev3*. [online] Available at: <https://store-usa.arduino.cc/products/arduino-mega-2560-rev3> [Accessed 22 January 2022].

[4] Amazon.com. n.d. [online] Available at: <https://www.amazon.com/Interface-Optocoupler-Protection-microcontroller-expansion/dp/B085Y42J66> [Accessed 22 January 2022].

[5] Operating systems client/server communication. (n.d.). Retrieved May 10, 2021, from https://www.tutorialspoint.com/operating-systems-client-server-communication

APPENDIX A : SENDING A STRING FROM ARDUINO TO PI & PI TO ARDUINO

Arduino Code 1:

```
void setup() {
  // put your setup code here, to run once:
  // Set the baud rate
  Serial.begin(9600);
}
void loop(){
  //check whether data is available over the port.
 if(Serial.available() > 0) {
  //send message to the Raspberry Pi.
  Serial.print("Hello!My name is Arduino ");
 }
}
```

Raspberry Pi Code 1:

```
import serial # Module needed for serial communication
# Set the port name and the baud rate. This baud rate should match
baud rate set on the Arduino.
#initialize serial communication
ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
# Get rid of incomplete data
ser.flush()
# Infinite loop
while (1):
 # If there is data available
 if(ser.in_waiting > 0):
 # Read everything until the new line character
 # Convert the data from a byte into a string of type 'utf-8'
# rstrip() function removes trailing characters like the new line
character '\n'
 line = ser.readline().decode('utf-8').rstrip()
 # Print the data received from the Arduino
 print(line)
```

Arduino Code 2:

```
void setup() {
  // put your setup code here, to run once:
  // Set the baud rate
  Serial.begin(9600);
}
```

```
void loop(){
  //check whether data is available over the port.
 if(Serial.available() > 0) {
  //read the data until the new line charachter
 String data = Serial.readStringUntil('\n');
 //print the message for acknowledgement of successfull
 //data transfer
 Serial.print("Hi Raspberry Pi! You sent me: ");
 Serial.println(data);
 }
}
```

<u>Raspberry Pi Code 2:</u>

```
import time # Module needed to add delays in the code
# Set the port name and the baud rate. This baud rate should match
# baud rate set on the Arduino.
#initialize serial communication and port.
ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
# Get rid of incomplete data
ser.flush()
# Infinite loop
while (1):
 send_string = ("My name is Raspberry Pi\n")
 # Send the string. Make sure you encode it before you send it to
 ser.write(send_string.encode('utf-8'))
 # Receive data from the Arduino
 receive_string = ser.readline().decode('utf-8').rstrip()
 # Print the data received from Arduino to the terminal
 print(receive_string)
```

APPENDIX B: COMMUNICATION OF RELAYS FROM ARDUINO TO PI

Raspberry Pi Code 3:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
###############
#Program: To send the relay number we want to switch on.
#File:send_relays_toPi.py
#Description: This program sends the number of relay boards we
#want to switch on to the arduino. and waits for the arduino
#to reply back with a acknowledgement before it further recites
#the next relay number.
#Date: 6th february 2021
##################
import serial #needed for serial communciation
ArduinoSerial= serial.Serial('/dev/ttyACM0',9600)#specifies arduino
serial port.
import numpy as np #import the mathematical library
def turn_on(condition,n):##defining our own function that
     #switches on/off relays
        global bits #global variable that can be accessed
                 #within or otuside the function
           if condition == 1: #if condition input is 1
                                 bits[n-1] = 1#set bits (relay no. -1)-
bit to 1
                                  #index is from 0 to 15 but relays are
from 0 to 16
          elif condition == 0:#if condition input is 0
                                 bits[n-1] = 0#set bits (relay no. -1)-
bit to 0
              return bits[::-1]#return all the bits

def actualHex(n):#function to change the bits to hexadecimal
    hex_number = str(n)#n is the number to be converted to hex
    #convert it to a class string
    hex_len = len(hex_number)#find length of the hex number
    needed_zeros = 4 - hex_len#add zeroes to spaces we dont have
numbers
    return '0'*needed_zeros + hex_number#put together the needed
zeroes
    #and the hex number

bits  = [0]*16#16 0's
integer = int(input("Enter relay no. "))#ask for relay number input
condition = int(input("Enter condition of relay "))#asks for condition
of relay
```

```
while integer != -1:#set condition that to exit from loop we have to
enter -1
    if integer <=16 :#set condition that relay number is elss than 16
        x = turn_on(condition,integer)#switch on relay that the user
inputs
        x = [str(i) for i in x]
        binary = "".join(x)#join the binary digits indivudally to form
the binary representation
        print('binary', binary)#print binary rep
        hex_rep = hex(int(binary,2))#convert binary to hex
        hex_rep = hex_rep[2:]
        hex_rep_actual = actualHex(hex_rep)#use the added zeroes to
form the full 16 bit representation
        print('hex representation', hex_rep_actual)#print hex
repsentation

        integer = int(input("Enter relay no."))#user prompt again

        if integer == -1:#exit condition is -1
            break

        condition = int(input("Enter condition of relay "))
        if condition != 1 and condition !=0:#error check
                                    #ensure that relay number and relay
condition are in their ranges
            print("Enter correct state of relay switch")
            condition = int(input("Enter condition of relay "))
    elif integer > 16:#error chech for relay number
        print("Error, Enter correct relay number")
        integer = int(input("Enter switch no."))
        condition = int(input("Enter condition of relay"))

        if condition != 1 and condition != 0:
            print("Enter correct state of relay switch")
            condition = int(input("Enter condition of relay "))
print("the binary representation is ",binary)#print final binary rep
print("The payload is ", hex_rep_actual)#print final hex rep

    ArduinoSerial.write(payload.encode('utf-8'))#encodes the string of
relays to be sent to the arduino
    receive_string=ArduinoSerial.readline().decode('utf-
8','replace'.rstrip())#decode the received message from the arduino
    print(receive_string)#print response from arduino.
```

Arduino Code 3:

```
/* Program: Send string to Arduino
 * FIle:string_to_Pi.ino ; Date:27/03/2021
 * Description: Receiving payload from Rasberry Pi **/
```

```
const int controlPin[16] = {2,3,4,5,6,7,8,9,10,11,12,A0,A1,A2,A3,A4};
//defining output pins on Arduino
const int triggerType= LOW; //intializing relay type (LOW or HIGH)
int tmpStat=1; //1 is for on and 0 for off
int Stringlength,i,relayreplen,j;//declare integer variables
String hexnumber,bin, payload; //declare string variables
String relayrep = ""; //intializing empty string
void setup() {
  //only code that is executed once is put here
  //data communication from this channel

  //This for loop sets up the Arduino pins and their states.
  //LOW means relay is ON and HIGH means relay is OFF
  for(int i=0; i<16; i++)
  {
    pinMode(controlPin[i], OUTPUT);//set pin as output.
    if(triggerType==LOW){ //if relay type is LOW
       digitalWrite(controlPin[i],HIGH); // set initial state OFF for
low trigger relay
    }else{ //if relay type is HIGH
        digitalWrite(controlPin[i],LOW);//set the initial state OFF
for high tigger relay.
          }
        }
  Serial.begin(9600);//set baud rate (i.e., data communication rate)
as 9600 (standard)
}


void loop() {
  // put your main code here, to run repeatedly
  if(Serial.available()!=0) {//if data is available through the serial
port
   Serial.print("Data Received. \n"); //Confirmation that data is
available
   payload = Serial.readString();//reading the received payload as a
string
   Serial.print("Payload Received:");
   Serial.println(payload); //printing the payload

    //This for loop converts 4 bit hexadecimal to 16 bit binary value
    for(i=0;i<5;i++){ //iterates through the hexadecimal number (4 bit
value)
      if(hexnumber.charAt(i)=='0'){ //if the ith hexadecimal character
being read is 0
        bin="0000"; //binary translation of hex value 0 to binary
value 0
        relayrep=relayrep+bin; //we store binary value in initialized
string and keep concatenating to it
        }
      else if(hexnumber.charAt(i)=='1'){
        bin="0001";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='2'){
        bin="0010";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='3'){
        bin="0011";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='4'){
        bin="0100";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='5'){
        bin="0011";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='6'){
        bin="0110";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='7'){
        bin="0111";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='8'){
        bin="1000";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='9'){
        bin="1001";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='a'){
        bin="1010";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='b'){
        bin="1011";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='c'){
        bin="1100";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='d'){
        bin="1101";
        relayrep=relayrep+bin;
        }
      else if(hexnumber.charAt(i)=='e'){
        bin="1110";
```

```
         relayrep=relayrep+bin;
         }
       else if(hexnumber.charAt(i)=='f'){
         bin="1111";
         relayrep=relayrep+bin;
         }
     } //end of for loop that iterates through hexadecimal value
     } //end of if loop checking if we are dealing with 'a' command

     /*Here, we ensure that no matter how many new hexadecimal payload
values the user inputs,
      * we always pick the last 16 (i.e., most recent updated)
characters of converted binary values as our final binary value.
      */
       relayreplen = relayrep.length(); //relayreplen stores total
length of our concatenated binary value
       relayrep = relayrep.substring(relayreplen-16,relayreplen);
//ensures we are always picking most recent 16 characters of our
concatenated string
       Serial.print("Corresponding Binary Payload:");
       Serial.println(relayrep); //prints our 16 bit binary value from
most recent hexadecimal input
   }//end of if loop checking if user input is entered/available.
//Binary to Relay
//This for loop checks which positions of binary payload are at 1 and
switches those respective relays ON. Binary 0 corresponds to relay
OFF.
//Also, here we are reversing the binary value in order to switch on
the correct relay corresponding to it.
//Example: 1000000000000000 binary input should switch on the 16th
relay, NOT the first one.
  for (i=0;i<16;i++){//iterates through the binary string
    if(relayrep.charAt(i)=='1'){//if the character in the binary
string is 1:
      j=15-i; //reverses the binary index
      digitalWrite(controlPin[j],LOW);}//switch on the relay number
(15-i)
    else if(relayrep.charAt(i)=='0'){//if charachter in the binary
string is 0:
      j=15-i; //reverses binary index
      digitalWrite(controlPin[j],HIGH);}//switch off the relay number
(15-i)
} //end of Binary to Relay for loop

} //end of void loop
```

APPENDIX C: ETHERNET COMMUNICATION

<u>Arduino Code 4:</u>

```
#include <Ethernet.h>//import ethernet libraries and modules
#include <EthernetUdp.h>
#include <SPI.h>

byte mac[]= { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEE};//define the max
address
IPAddress ip(192,168,1,101);//define the IP address
unsigned int port =5000;//initialize the port of comm
//This data consists of the source and destination ports to
communicate on, the packet length and a checksum.
//Array that holds the information during data transmission.
char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
//Once we get the data we will define a string in which we will put
this data.
String dataReq;
int packetSize;
EthernetUDP Udp;//create a UDP packet used in the transfer of data

void setup() {
  Serial.begin(9600);//begin serial comm
  Ethernet.begin(mac,ip);//
  Udp.begin(port);//begin the udp object at the specified port
  delay(1500);

}
void loop() {
  packetSize= Udp.parsePacket();//if the packetsize is greater than 0
there is data available.
  //Looks at ethernet and checks if there is a request. If size is 0
no request.
  if(packetSize>0){
    Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE); //read the data
into the packetBuffer
    String dataReq(packetBuffer);//convert the array to a string
    if(dataReq="Red"){//if the string is "Red"
      Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());//begin the udp
comm again and send it back to the ip and mac address it came from
      Udp.print("You are asking for red");//Udp.print to print back
the message to the client.
      Udp.endPacket();//end communication
    }
    if(dataReq="Blue"){//repeat the steps for an array that holds the
message "Blue"
      Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());
      Udp.print("You are asking for blue");
      Udp.endPacket();
    }
  }
 memset(packetBuffer,0,UDP_TX_PACKET_MAX_SIZE);//reset the
packetbuffer to zero everytime
}
```

Raspberry Pi Code 4:

```
from socket import*#import sockt module
import time#import time module

address= ('192.168.1.101', 5000) # define who we are talking to
client_socket=socket(AF_INET,SOCK_DGRAM)# set up the ocket.
client_socket.settimeout(1) #wait 1 second- change accordingly

while(1):
     data= b"Red" #set data to Red
     client_socket.sendto(data,address)#send data to the Arduino over
the specified adddress
     try:
               #reading response from arduino.
     rec_data, addr =client_socket.recvfrom(2048)#receiving data
               print (rec_data) #printing data
     except:#
                         pass
     time.sleep(2)
     data= b"Blue" #set data to Blue
     client_socket.sendto(data,address)#send data to the Arduino
     try:
               #reading response from arduino.
     rec_data, addr =client_socket.recvfrom(2048)#receive the message
          print (rec_data) #3print the received data
     except:
                         pass
     time.sleep(2)
```

APPENDIX D: SENDING HEXADECIMAL PAYLOADS FROM PI TO ARDUINO OVER ETHERNET

<u>Arduino Code 5:</u>

```
#include <Ethernet.h>//import ethernet libraries and modules
#include <EthernetUdp.h>
#include <SPI.h>
//For ethernet communication

byte mac[]= { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEE};//define the max
address
IPAddress ip(192,168,1,101);//define the IP address
unsigned int port =5000;//initialize the port of comm
//This data consists of the source and destination ports to
communicate on, the packet length and a checksum.
//Array that holds the information during data transmission.
char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
//Once we get the data we will define a string in which we will put
this data.
String hexnumber;
int packetSize;
EthernetUDP Udp;//create a UDP packet used in the transfer of data

//for receiving paylod
const int controlPin[16] = {2,3,4,5,6,7,8,9,10,11,12,A0,A1,A2,A3,A4};
//defining output pins on Arduino
const int triggerType= LOW; //intializing relay type (LOW or HIGH)
int tmpStat=1; //1 is for on and 0 for off
int Stringlength,i,relayreplen,j;//declare integer variables
String bin, payload; //declare string variables
String relayrep = ""; //intializing empty string
void setup() {

  Serial.begin(9600);//begin serial comm
  Ethernet.begin(mac,ip);//begin ethernet communication from specified
address
  Udp.begin(port);//begin the udp object at the specified port
  //data communication from this channel
  //This for loop sets up the Arduino pins and their states.
  //LOW means relay is ON and HIGH means relay is OFF
  for(int i=0; i<16; i++)
  {
    pinMode(controlPin[i], OUTPUT);//set pin as output.
    if(triggerType==LOW){ //if relay type is LOW
       digitalWrite(controlPin[i],HIGH); // set initial state OFF for
low trigger relay
    }else{ //if relay type is HIGH
        digitalWrite(controlPin[i],LOW);//set the initial state OFF
for high tigger relay.
          }
        }
  Serial.begin(9600);//set baud rate (i.e., data communication rate)
as 9600 (standard)
  packetSize= Udp.parsePacket();//if the packetsize is greater than 0
there is data available.
    //Looks at ethernet and checks if there is a request. If size is 0
no request.
    if(packetSize>0){
      Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE); //read the data
into the packetBuffer
      Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());
      Udp.print("Data Received: ");
      String hexnumber(packetBuffer);//convert the array to a string
      Udp.print(hexnumber);
      Udp.endPacket();
     //This for loop converts 4 bit hexadecimal to 16 bit binary value
      for(i=0;i<5;i++){ //iterates through the hexadecimal number (4
bit value)
        if(hexnumber.charAt(i)=='0'){ //if the ith hexadecimal
character being read is 0
          bin="0000"; //binary translation of hex value 0 to binary
value 0
          relayrep=relayrep+bin; //we store binary value in
initialized string and keep concatenating to it
          }
        else if(hexnumber.charAt(i)=='1'){
          bin="0001";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='2'){
          bin="0010";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='3'){
          bin="0011";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='4'){
          bin="0100";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='5'){
          bin="0011";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='6'){
          bin="0110";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='7'){
          bin="0111";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='8'){
          bin="1000";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='9'){
          bin="1001";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='a'){
          bin="1010";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='b'){
          bin="1011";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='c'){
          bin="1100";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='d'){
          bin="1101";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='e'){
          bin="1110";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='f'){
          bin="1111";
          relayrep=relayrep+bin;
          }
      } //end of for loop that iterates through hexadecimal value
    }

      /*Here, we ensure that no matter how many new hexadecimal
payload values the user inputs,
       * we always pick the last 16 (i.e., most recent updated)
characters of converted binary values as our final binary value.
       */
        relayreplen = relayrep.length(); //relayreplen stores total
length of our concatenated binary value
        relayrep = relayrep.substring(relayreplen-16,relayreplen);
//ensures we are always picking most recent 16 characters of our
concatenated string
        Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());
        Udp.print("Corresponding Binary Payload:");
        Udp.print(relayrep); //prints our 16 bit binary value from
most recent hexadecimal input
        Udp.endPacket();
  //end of if loop checking if user input is entered/available.
  //Binary to Relay
  //This for loop checks which positions of binary payload are at 1
and switches those respective relays ON. Binary 0 corresponds to relay
OFF.
  //Also, here we are reversing the binary value in order to switch on
the correct relay corresponding to it.
  //Example: 1000000000000000 binary input should switch on the 16th
relay, NOT the first one.
     for (i=0;i<16;i++){//iterates through the binary string
      if(relayrep.charAt(i)=='1'){//if the character in the binary
string is 1:
        j=15-i; //reverses the binary index
        digitalWrite(controlPin[j],LOW);}//switch on the relay number
(15-i)
      else if(relayrep.charAt(i)=='0'){//if charachter in the binary
string is 0:
        j=15-i; //reverses binary index
        digitalWrite(controlPin[j],HIGH);}//switch off the relay
number (15-i)
  } //end of Binary to Relay for loop
  }
  void loop() {
  }//end of void loop
```

Raspberry Pi Code 5:

```
from socket import*#import sockt module
import time#import time module
import Serial
address= ('192.168.1.101', 5000) # define who we are talking to
client_socket=socket(AF_INET,SOCK_DGRAM)# set up the ocket.
client_socket.settimeout(1) #wait 1 second- change accordingly
import numpy as np #import the mathematical library
def turn_on(condition,n):##defining our own function that
     #switches on/off relays
        global bits #global variable that can be accessed
                 #within or otuside the function
           if condition == 1: #if condition input is 1
                                bits[n-1] = 1#set bits (relay no. -1)-
bit to 1
```

```
                                  #index is from 0 to 15 but relays are
from 0 to 16
          elif condition == 0:#if condition input is 0
                                bits[n-1] = 0#set bits (relay no. -1)-
bit to 0
              return bits[::-1]#return all the bits

def actualHex(n):#function to change the bits to hexadecimal
    hex_number = str(n)#n is the number to be converted to hex
    #convert it to a class string
    hex_len = len(hex_number)#find length of the hex number
    needed_zeros = 4 - hex_len#add zeroes to spaces we dont have
numbers
    return '0'*needed_zeros + hex_number#put together the needed
zeroes
    #and the hex number

bits  = [0]*16#16 0's
integer = int(input("Enter relay no. "))#ask for relay number input
condition = int(input("Enter condition of relay "))#asks for condition
of relay


while integer != -1:#set condition that to exit from loop we have to
enter -1
    if integer <=16 :#set condition that relay number is elss than 16
        x = turn_on(condition,integer)#switch on relay that the user
inputs
        x = [str(i) for i in x]
        binary = "".join(x)#join the binary digits indivudally to form
the binary representation
        print('binary', binary)#print binary rep
        hex_rep = hex(int(binary,2))#convert binary to hex
        hex_rep = hex_rep[2:]
        hex_rep_actual = actualHex(hex_rep)#use the added zeroes to
form the full 16 bit representation
        print('hex representation', hex_rep_actual)#print hex
repsentation

        integer = int(input("Enter relay no."))#user prompt again

        if integer == -1:#exit condition is -1
            break

        condition = int(input("Enter condition of relay "))
        if condition != 1 and condition !=0:#error check
     #ensure that relay number and relay condition are in their ranges
            print("Enter correct state of relay switch")
            condition = int(input("Enter condition of relay "))
    elif integer > 16:#error chech for relay number
        print("Error, Enter correct relay number")
        integer = int(input("Enter switch no."))
        condition = int(input("Enter condition of relay"))

        if condition != 1 and condition != 0:
            print("Enter correct state of relay switch")
            condition = int(input("Enter condition of relay "))
print("the binary representation is ",binary)#print final binary rep
print("The payload is ", hex_rep_actual)#print final hex rep
while(1):
     data=str.encode(hex_rep_actual)#set data to the hexadecimal
payload
     #byte type object is required hence we encode it
     client_socket.sendto(data,address)#send data to the Arduino
     try:
                #reading response from arduino.
     rec_data, addr =client_socket.recvfrom(2048)#receiving data
                print (rec_data) #printing data
     except:
                            pass
```

APPENDIX E: COMPANION FILE FOR ARDUINO CODE

```
//Date Created: 25/10/2021
//Companion file containing IP Address for ethernet communication
//between Arduino and Pi
//IP Address
#define IPAddress1    192,168,1,101    // your IP Address for
communication
```

APPENDIX F: FUNCTION DEFINITIONS FOR GENERALIZATION

<u>Arduino Code 6:</u>

```
/************************************************
 * Date created: 3rd October 2021
 * Project Name: IV Curve Tracer (IVCT)-Phase 2
 * Code written by: Swati Singh and Maria Khan
 * Mentor: Dr. Robert S. Balog
 ************************************************/
 /*####################################################*/
 /************************************************
  * Hardware used:
  * (1)Arduino Mega2560 R3 Board ATmega2560 ATMEGA16U2
  * (2)Ethernet Shield W5100
  * (3)12V 16 Channel relay module by SainSmart
  ************************************************/
  /*####################################################*/

//import ethernet libraries and modules
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <SPI.h>
#include "IPAddress.h"   //user definitions of static variables


/*####################################################*/

//For ethernet communication
//define the mac address
byte mac[]= { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEE};
IPAddress ip (IPAddress1);//define the IP address
unsigned int port =5000;//initialize the port of communication
//This data consists of the source and destination ports
//to communicate on, the packet length and a checksum.
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //Array that holds the
information during data transmission.
//Once we get the data we will define a string in which we will put
this data.
String hexnumber;
int packetSize,elapsedtime;
EthernetUDP Udp;//create a UDP packet used in the transfer of data
int repeat=0;
```

```
/*#####################################################*/

//Declaring variables
//const int
controlPin[32]={15,14,17,16,19,18,21,20,23,22,25,24,27,26,29,28,31,30,
33,32,35,34,37,36,39,38,41,40,43,42,45,44};//pin
const int
controlPin[32]={14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
32,33,34,35,36,37,38,39,40,41,42,43,44,45};
const int triggerType= LOW; //intializing relay type (LOW or HIGH)
int tmpStat=1; //1 means switch relay ON, 0 means switch relay OFF
int Stringlength,i,relayreplen,j;//declare integer variables
String bin; //declare string variables
String relayrep = ""; //intializing empty string
int timer;

/*#####################################################*/
/*#########################################*/
//Function Definitions: (Python Code remains the same)

String HextoBin(String hexnumber){
    //Hexademical to Binary
     //This for loop converts the 8 bit hexadecimal payload to its
corresponding 32 bit binary value.
     /*
      * Here, we read each character of the hexadecimal payload.
      * Each character corresponds to a 4 bit binary value.
      * We iterate through the hexadecimal characters and manually
assign each possible hexadecimal value with its corresponding binary
conversion as a string
      * We store each eight bit binary value into an empty string.
      * Concatenating all eight characters of the hexademical payloads
and each of their corresponding 4 bit binary values gives a final 32
bit binary value.
      * This final 32 bit representation corresponds to the state of
the relays depending on user input.
      */
   for(i=0;i<8;i++){ //beginning of for loop that iterates through
hexadecimal value

        if(hexnumber.charAt(i)=='0'){ //if the ith hexadecimal
character being read is 0
          bin="0000"; //binary translation of hex value 0 is binary
value 0000
          relayrep=relayrep+bin; //we store binary value in
initialized string and keep concatenating to the end of it
          }
        else if(hexnumber.charAt(i)=='1'){
          bin="0001";
          relayrep=relayrep+bin;
          }
        else if(hexnumber.charAt(i)=='2'){
```

```
  bin="0010";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='3'){
  bin="0011";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='4'){
  bin="0100";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='5'){
  bin="0101";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='6'){
  bin="0110";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='7'){
  bin="0111";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='8'){
  bin="1000";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='9'){
  bin="1001";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='a'){
  bin="1010";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='b'){
  bin="1011";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='c'){
  bin="1100";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='d'){
  bin="1101";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='e'){
  bin="1110";
  relayrep=relayrep+bin;
  }
else if(hexnumber.charAt(i)=='f'){
```

```
          bin="1111";
          relayrep=relayrep+bin;
          }
      }
      return relayrep;
}

void BinaryToRelay(String relayrep){
  //Binary to Relay
  //This for loop checks which positions of binary payload are at 1
  //and switches those respective relays ON. Binary 0 corresponds to
relay OFF.
  //Also, here we are reversing the binary value in order to switch on
  //the correct relay corresponding to it because binary values are
read from the rightmost value.
  //Example: 1000000000000000 binary input should switch on the 16th
relay, NOT the first one.
    for (i=0;i<32;i++){//beginning of for loop that iterates through
the binary string
          if(relayrep.charAt(i)=='1'){//if the character in the binary
string is 1:
            j=31-i; //reverses the binary index
            digitalWrite(controlPin[j],LOW);}//switch on the relay
number (15-i)
          else if(relayrep.charAt(i)=='0'){//if charachter in the
binary string is 0:
            j=31-i; //reverses binary index
            digitalWrite(controlPin[j],HIGH);}//switch off the relay
number (15-i)
      } //end of for loop that iterates through the binary string
}

void DataAvailability(int packetSize){
       //timer=millis();
       if(packetSize>0){ //beginning of if loop checking data
availability from Pi to Arduino Mega.
        //If the packetsize is greater than 0, it means there is data
available.
        Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE); //This reads
the data into the packetBuffer
        Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());
        Udp.print("Data Received: "); //This print command confirms to
us that data from Pi was received by Arduino Mega
        String hexnumber(packetBuffer);//This converts the data array
to a string.
        //We convert to a string in order to iterate through each
character in the hexadecimal payloadsent by the Pi to the Arduino
Mega.
        Udp.print(hexnumber); //This prints the hexadecimal payload.
        Udp.endPacket();
        HextoBin(hexnumber);//Calling function that converts our 8 bit
hex payload to 32 bit binary value.
```

```
        }//end of if loops checking 8 bits for data availability
        /*else if (packetSize<8){//if 8 bits not received
          Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());
          Udp.print("Data not Received");
          Udp.endPacket();
        }*/ //end of else statement
 }//end of function definition

void setup() {
  Serial.begin(9600);//begin serial communication needed to upload the
sketch to the arduino board.
  Ethernet.begin(mac,ip); //begin ethernet communication from
specified address
  Udp.begin(port);//begin the udp object at the specified port
  delay(1000);//to ensure that all the data has been communicated
completely.

  //LOW means relay is ON and HIGH means relay is OFF

    for(int i=0; i<32; i++)
  { //begins for loop to set up the Arduino pins and their states on
terminal
    pinMode(controlPin[i], OUTPUT);//set pin as output.
    if(triggerType==LOW){ //begin if loop
      //if relay type is LOW, we set initial relay state OFF for low
trigger relay
       digitalWrite(controlPin[i],HIGH);
    } //end if loop
    else{ //begin else loop
      //if relay type is HIGH, set the initial realy state OFF for
high tigger relay.
        digitalWrite(controlPin[i],LOW);
          } //end else loop
  }
} //end of void setup loop

/*####################################################*/

void loop() {

  packetSize= Udp.parsePacket(); //Here, we read packetsize of data
being transmitted.
    //Looks at ethernet and checks if there is a request. If size is
0, there is no request.

       DataAvailability(packetSize);


      /*Here, we ensure that no matter how many new hexadecimal
payload values the user inputs,
       * we always pick the last 16 (i.e., most recently updated)
characters of converted binary values as our final binary value.
```

```
       */

        relayreplen = relayrep.length(); //relayreplen stores total
length of our concatenated binary value
        relayrep = relayrep.substring(relayreplen-32,relayreplen);
//ensures we are always picking most recent 16 characters of our
concatenated string
        Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());//transmit
data back to the port from which it was received.
        Udp.print(" Corresponding Binary Payload:");
        Udp.print(relayrep); //prints our 32 bit binary value from
most recent hexadecimal input
        Udp.endPacket();

        BinaryToRelay(relayrep);

  memset(packetBuffer,0,UDP_TX_PACKET_MAX_SIZE);//reset the
packetbuffer to zero everytime

}//end of void loop
```

APPENDIX G: EXCEL INPUT FOR SCALABILITY

<u>Raspberry Pi Code 6:</u>

```
from socket import*#import sockt module
import time#import time module
import pandas as pd
import xlrd
address= ('192.168.1.101', 5000) #define who we are talking to
client_socket=socket(AF_INET,SOCK_DGRAM)# set up the ocket.
client_socket.settimeout(1) #wait 1 second- change accordingly
import numpy as np #import the mathematical library

#Read Excel file:
loc = (r"C:\Users\maria\Desktop\Fall 2021\ECEN
491\IVCT1.xls")#deifning location of excel file
wb = xlrd.open_workbook(loc)#opening excel file
sheet1 = wb.sheet_by_index(0)#specifying first sheet to read.
sheet2= wb.sheet_by_index(1)#specifying second sheet with IP
Addresses.
sheet1.cell_value(0, 0)#start reading from first row first column
for i in range(sheet1.nrows): #iterating through the rows
     print(sheet1.row_values(i))#printing aany filled out rows i.e.
printing all information in the firrst sheet

experimentno=int(input("Enter which experiment you want to run (Please
enter numerical value only): "))
relayson=sheet1.cell_value(experimentno,1)#reading column with relay
states
relayson = relayson.split(",")#splitting the comma seperated values
and storing in a list
print(relayson)#printing the relay numbers to be switched on

#Generation of Payload:

def turn_on(condition,n):##defining our own function that
     #switches on/off relays
        global bits #global variable that can be accessed
                  #within or otuside the function
            if condition == 1: #if condition input is 1
                                 bits[n-1] = 1#set bits (relay no. -1)-
bit to 1
                                  #index is from 0 to 15 but relays are
from 0 to 16
```

```
           elif condition == 0:#if condition input is 0
                               bits[n-1] = 0#set bits (relay no. -1)-
bit to 0
             return bits[::-1]#return all the bits

#To incorporate control for 2 relay board by 1 Arduino Mega we need to
icrease the length of the binary representation to 32 bits which gives
#us a 8 bit hexadecimal payload.
def actualHex(n):#function to change the bits to base 32 (line change)
    hex_number = str(n)#n is the number to be converted to base
    #convert it to a class string
    hex_len = len(hex_number)#find length of the hex number
    needed_zeros = 8 - hex_len#add zeroes to spaces we dont have
numbers (line change)
    return '0'*needed_zeros + hex_number#put together the needed
zeroes
    #and the hex number

bits  = [0]*32#32 0's (line change)
for j in range(0,len(relayson)):
     integer=int(relayson[j])
     print(integer)#error check
#while integer != -1:#set condition that to exit from loop we have to
enter -1
     if integer <=32 :#set condition that relay number is elss than 32
(line change)
     x = turn_on(1,integer)#switch on relay that the user inputs
                    x = [str(i) for i in x]
     binary = "".join(x)#join the binary digits indivudally to form
the binary representation
       hex_rep = hex(int(binary,2))#convert binary to hex
                     hex_rep = hex_rep[2:]
     hex_rep_actual = actualHex(hex_rep)#use the added zeroes to form
the full 16 bit representation
             print("Binary representation:",binary)
     print('Hexadecimal representation:', hex_rep_actual)#print hex
repsentation


while integer != -1:#set condition that to exit from loop we have to
enter -1
     experimentno=int(input("Enter which experiment you want to run
(Please enter numerical value only): "))
     if experimentno == -1:#exit condition is -1
           break

while(1):
     data=str.encode(hex_rep_actual)#set data to the hexadecimal
payload
     #byte type object is required hence we encode it
     client_socket.sendto(data,address)#send data to the Arduino
     try:
                 #reading response from arduino.
     rec_data, addr =client_socket.recvfrom(2048)#receiving data
                 print (rec_data) #printing data
     except:
                               pass
```