Word Count: 2449

Video, Working with UART, I2C, & SPI Protocols

In this video, we will learn about UART, I2C, and SPI Protocols, Then we will implement UART Communication between the Pi and PC, Later we will implement I2C for interfacing I2C 16x2 LCD Module, and finally, we will implement SPI for interfacing the RFID Sensor.

Communication Protocols are a set of rules and regulations that allows two electronic devices to connect for exchanging data with one another. Currently, UART, SPI, and I2C are the common hardware communication protocols people use in microcontroller development, and the Raspberry Pi 4 supports all three. This allows for interfacing the Pi 4 with a large array of sensors, actuators, and communication devices much easier.

UART also known as Universal Asynchronous Receiver/Transmitter is a serial communication protocol in which data is transferred serially i.e. bit by bit. UART serial communication protocol uses a defined frame structure for their data bytes. The frame structure basically consists of 3 things:

* · START bit: It is a bit with which indicates that serial communication has started, and it is always low.
* · Data bits packet: Data bits can be packets of 5 to 9 bits. Normally we use 8-bit data packet, which is always sent after the START bit.
* · STOP bit: This usually is one or two bits in length. It is sent after data bits packet to indicate the end of frame. Stop bit is always logic high.

The frame can also consist of 2 STOP bits instead of a single bit, and there can also be a PARITY bit after the STOP bit. The parity bit is used to check the integrity of the information in the data frame.

In the Raspberry Pi 4, we will show you the basics of UART Communication by transmitting data from the Pi to your computer through a USB to Serial Convertor. The first step is to configure the Pi 4 to enable UART. In the terminal type “sudo raspi-config”, then select the interfacing options and then select Serial in the configuration menu. Then it will ask for login shell to be accessible over Serial. Select No as shown here. At the end, it will ask for enabling Hardware Serial Port, Select Yes. Finally, our UART is enabled for Serial Communication on RX and TX pin of Raspberry Pi 4. Now, reboot the Pi.

The Pi 4 uses the UART port to interface its Bluetooth module to the CPU. Another Hardware UART port is provided across the GPIO14 and GPIO15. We will use this port to send the data to the Laptop. Before working with the UART port, we need to know how its mapped. For this enter the following command in the Pi’s terminal.

ls -l /dev

You should look for the UART Mapping for serial0, which represents the GPIO14, and GPIO15 UART Bus. Here you can see that its mapped to ttyS0. Please keep this in mind.

Now let’s interface the USB to Serial Convertor as shown here in the circuit diagram. It is really simple, but you have got to make sure that you connect the Ground pin of the convertor to one of Pi’s ground pins. Another thing to note is that the Rx pin of the Convertor should be connected to the Tx pin of the Pi and the Tx pin of the convertor to the Rx pin go the Pi. Now connect the Converter to the USB port of your computer. Download any serial port monitor tool of your wish on your PC. We are using RealTerm. Now on the Raspberry Pi open the UART.py code, and run it. On the PC, Open the RealTerm application, and select the correct port after checking in the device manager. Then select the baud rate as 9600 and open the port. Now you can see that indeed, the data transmission is happening from the Pi to the PC. Now let me explain the code on the Pi. First, we import the inbuilt module named serial. Then we import the class sleep form the time module.

Next, we need to create an instance of class serial and assign it to “ser”. The Serial class accepts 5 arguments as inputs to configure the UART Communication. The first is the port name, the second is the baud rate, which determines how many bits of information will be sent per second. The receiver should also be configured to the same rate or we can’t retrieve the information. Finally, we set it to have no parity bit, on stop bit, and the data bitesize to be 8 bits. The rest of the code is just a counter inside an infinite loop with a delay of 1 second that is writing the following string using the “write” method of the serial class.

Now let us move on to I2C protocol.

Transmitting and receiving information between two or more than two devices requires a communication path called a bus system. An I2C bus is a bidirectional two-wire serial bus that is used to transport the data between integrated circuits. The I2C stands for “Inter-Integrated Circuit”. It was first introduced by Philips semiconductors in 1982. The I2C bus protocol is most commonly used in master and slave communication wherein the master is called “microcontroller”, and the slave is one of the other devices such as ADC, EEPROM, DAC, and similar devices in the embedded system. The I2C bus can have multiple slaves as well as masters. Each device on the bus will have a unique address.

The following steps are used to communicate the master device to the slave:

Step 1: First, the master device issues a start condition to inform all the slave devices so that they listen on the serial data line.

Step 2: The master device sends the address of the target slave device which is compared with all the slave devices addresses as connected to the SCL and SDL lines. If anyone address matches, that device is selected, and all the remaining devices are disconnected from the SCL and SDL lines.

Step 3: The slave device with a matched address received from the master, responds with an acknowledgment to the master thereafter communication is established between both the master and slave devices on the data bus.

Step 4: Both the master and slave receive and transmit the data depending on whether the communication is to read or write.

Step 5: The slave lets the master know that the message has been received or transmitted depending on whether the communication is to write or read

Step 6: Finally, the master device issues a stop condition to inform all the slave devices in the bus

Now, as we are clear with the I2C protocol, let's have a look at the I2C 16x2 LCD Display before we start interfacing.

A liquid crystal display or LCD draws its definition from its name itself. It is a combination of two states of matter, the solid and the liquid. Liquid crystal display is composed of several layers which include two polarized panel filters and electrodes. Liquid crystal display screen works on the principle of blocking light rather than emitting light. So, LCD’s requires a backlight as they do not emit light by themselves. The liquid crystal can twist based on the current supplied through the electrodes. Initially, when no current is applied across the electrode the crystal will be in a twisted state. When a current is applied between the electrodes, the crystal starts untwisting which changes the polarization characteristics of the light and allows it to shine through. Each pixel will have 3 such individual subpixels, with red, green, and blue filters respectively. The mixing of different intensity primary colors, allows us to reproduce millions of colors. If you want to know more about the polarization of light, please check out the links in the resources section.

The type of LCD we are using here is known as a monochrome 16x2 LCD. This means that it can only produce light of one color. The 16 by 2 means that it can display 16 characters per line in 2 such lines. In this LCD, each character is displayed in a 5×8 pixel matrix.

If you look at the pinout diagram of a 16x2 LCD, you can see that it has 16 pins. The I2C 16x2 LCD we are using is made by using a parallel to I2C converter based on PCF8574A shift register-, that converts the 16 pins to just a 4 Pin connection. These types of modules are commonly referred to as LCD backpacks. The module has a potentiometer to control the contrast of the LCD. It also has a Jumper pin, which can be removed to disable the backlight also. Now shall we interface the display to the Raspberry Pi.

Connect the GPIO2 SDA pin on the Pi to the SDA pin on the LCD, and the GPIO3 SCL pin on the Pi to the SCL pin on the LCD. The ground and Vcc pins will also need to be connected. Most LCDs can operate with 3.3V, but they’re meant to be run on 5V, so connect it to the 5V pin of the Pi.

Now, let’s get on with the software side of things. Like UART, you need to enable the I2C interface from the configuration tool. Follow the same method as we did for UART, but this type select the I2C option and enable it. Please don’t forget to reboot after the change.

The next step is to find the I2C address of your LCD. This can be done by entering.

I2cdetect -y 1

In the terminal.

The address is shown here as 3F. For you, it might be different.

Next, we need to modify the I2C LCD Driver library and add this address, instead of the default address. Go to your Repository and open the I2C\_LCD\_driver.py code in thonny IDE. Now modify the address to your I2C address and save the file. Now to work with the LCD in any code, you need to include this library file in your current working directory where the main code will be residing. Now open the Hello World LCD.py code in the repository and run the code. You can see the text “Hello World” on the LCD. Please check out the resources section, where I have provided different sample codes to explore different function on the LCD further, like Positioning the text, Clearing the screen, Blinking the text, scrolling the text, Printing custom characters, Printing data from a sensor and so on.

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface protocol used for short-distance communication, primarily in embedded systems. Devices communicating via SPI are in a master-slave relationship. The master is the controlling device, usually a microcontroller, while the slave usually a sensor, display, or memory chip takes instruction from the master. The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave also.

SPI uses 4 separate connections to communicate with the target device. These connections are the serial clock (SCLK), Master Input Slave Output (MISO), Master Output Slave Input (MOSI), and Chip Select (CS).

Now we will look at the steps of SPI Data Transmission

1. The master outputs the clock signal

2. The master switches the SS/CS pin to a low voltage state, which activates the slave

3. The master sends the data one bit at a time to the slave along the MOSI line. The slave reads the bits as they are received.

4. If a response is needed, the slave returns data one bit at a time to the master along the MISO line. The master reads the bits as they are received

We will practically learn about SPI by interfacing an RFID Sensor called RC522.RFID is an acronym for “radio-frequency identification” and refers to a technology whereby digital data encoded in RFID tags or smart labels are captured by a reader via radio waves. RFID is similar to barcoding in that data from a tag or label are captured by a device that stores the data in a database.

An RFID tag consists of an integrated circuit and an antenna. The tag is also composed of a protective material that holds the pieces together and shields them from various environmental conditions. The protective material depends on the application. For example, employee ID badges containing RFID tags are typically made from durable plastic, and the tag is embedded between the layers of plastic.

Wiring your RFID RC522 to your Raspberry Pi is fairly simple, with it requiring you to connect just 7 of the GPIO Pins directly to the RFID reader. Connect the SCK Pin to GPIO11, SDA pin to GPIO8, MOSI to GPIO10, MISO to GPIO9, GND to any ground pin, RST to GPIO25, and 3.3 V to 3.3 V pin on the Pi.

The next step is to go the raspberry pi configuration tool and enable SPI inside the interfacing option. Please reboot before going further. Now make sure you install SPI Supporting libraries by entering this command.

sudo apt-get install python-spidev python3-spidev

Now lets download the required library and example code by typing.

git clone<https://github.com/mxgxw/MFRC522-python.git>

Now go to the new folder that is created after downloading the library and open the read.py file and run it. Now bring an RFID Tag near the module. When it finds a tag, it reads the UID and displays it on the screen. The script runs in a loop and will keep waiting and displaying any detected UIDs. Running this script allows you to determine the UID of the tag or card that was supplied with the reader and later create it to use the logic for your application. You can find a detailed explanation of the read.py and write.py code in the resources section.

Summary

In this video, we have covered the following

● What is UART, I2C, and SPI protocols

● Implement UART between a PC and the Pi with a USB to Serial Converter

● Implement I2C for interfacing I2C 16x2 LCD

● Implement SPI for interfacing MFRC522 RFID reader

Section Summary.

In this section, we have covered the following

● Getting started with Physical computing on the Raspberry Pi 4

● Introduction to Hardware Interfacing on the Raspberry Pi 4

● Working with LEDs & Buttons

● Working with LDR & PIR Sensors

● Working with Relays

● Working with Ultrasonic Sensor & DC Motors

● Working with UART, I2C & SPI Protocols

In the next section, we will cover Wireless Communication in the Raspberry Pi 4