Week1-Day1&2: Introduction to Embedded C, Simulation and mini-hands-on tasks

**Learning Objective: Understanding Embedded-C concepts, Simulation of Projects Embedded systems basics.**

We started the day by explaining the differences between Microcontroller, Arduino, NodeMCU and RaspberryPi, and we were taught the different specifications of these boards. Then, we began by explaining microcontrollers and microprocessors and their respective properties.

After that, the explanation of embedded C programming, its purposes, and the use of its syntaxes were given.

**Hands-On activity: LED blinking**

**Input:** Power supply, Code

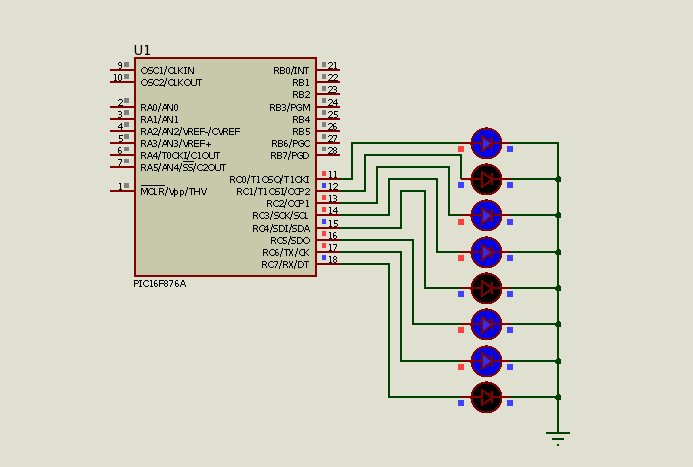
**Output:** consecutively Segmented LED blinking

**Tools:** Proteus, Keil uVision

**Logic:** The program is designed in such a way that consecutive adjacent LEDs were to blink for a given delay time while the adjacent LED remained off, this was interswitched between them using delay.

**Code (uploaded to Github)**

**Output**:



Week1-Day3: introduction to Button interfacing with 8051

**Learning Objective:** Taking input from a button and respectively controlling the IC output based on the input generated.

**Input:** Push Buttons, Power supply, Code

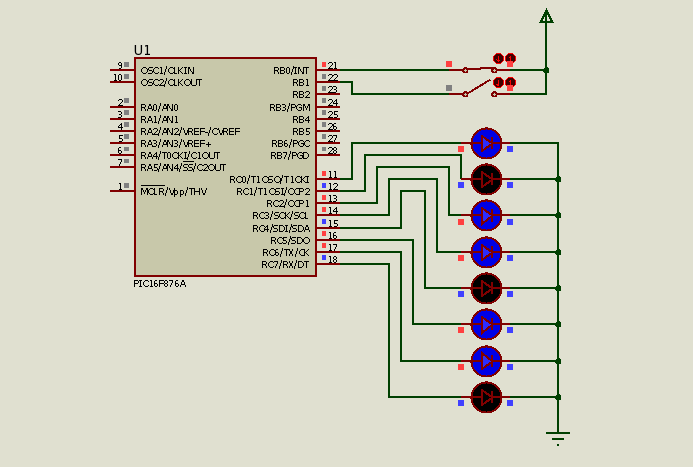
**Output:** LEDs

**Tools:** Proteus, Keil uVision

**Logic:**  Push buttons are connected to any respective defined input pin of the microcontroller, signal has to be read from these pins through the button states, and the LEDs are controlled prior to the button states using the program.

**Code (uploaded to Github)**

**Output:**

****

**Common issues: Proteus version compatibility issues, Connection errors.**

Week1-Day4: introduction to LCD interfacing with 8051

**Learning Objective:** basic knowledge of LCD, and interfacing it with 8051 microcontroller for informative display.

**Input:** Push Buttons, Power supply, Code

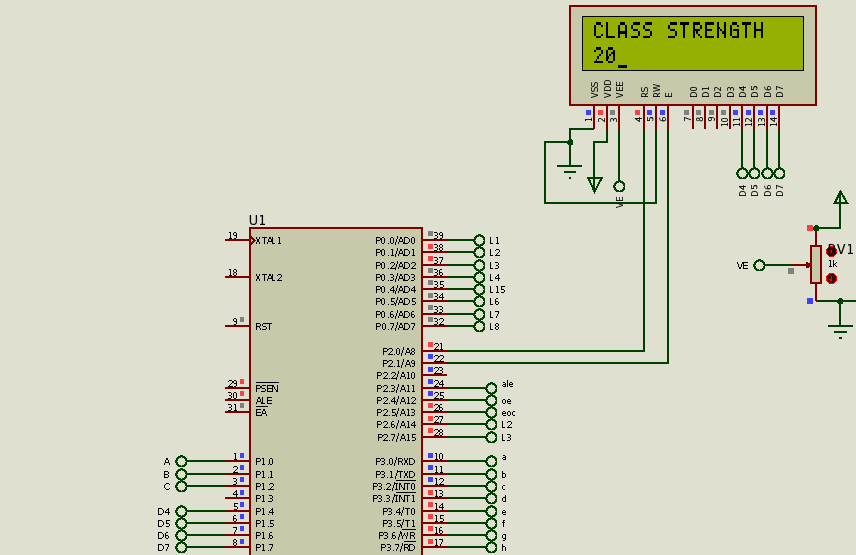
**Output:** LEDs, LCD

**Tools:** Proteus, Keil uVision

**Logic:**  LCD (16 columns x 2 rows) has 16 pin for connection, with the main 2 for power(Vcc and GND), data pins(D0-D7), control pins(en,rs,rw,a,k,ve), as per the LCD datasheet, these control and data pins has to configure in code for proper communication to the LCD to generate effective display, the code is designed to display the session of the day and the class strength.

**Code (uploaded to Github)**

**Output:**

****

Week1-Day5: introduction to ADC interfacing with 8051

**Learning Objective:** basic knowledge of ADC, and interfacing it with 8051 microcontroller for analog input interfacing and conversion to digital output generation.

**Input:** Push Buttons, ADC0808 IC, Power supply, Code

**Output:** LEDs

**Tools:** Proteus, Keil uVision

**Logic:**  ADCs convert **continuous analog signals** (like voltage variations from sensors) into **discrete digital signals** (represented by a series of 0s and 1s) that digital devices can understand.

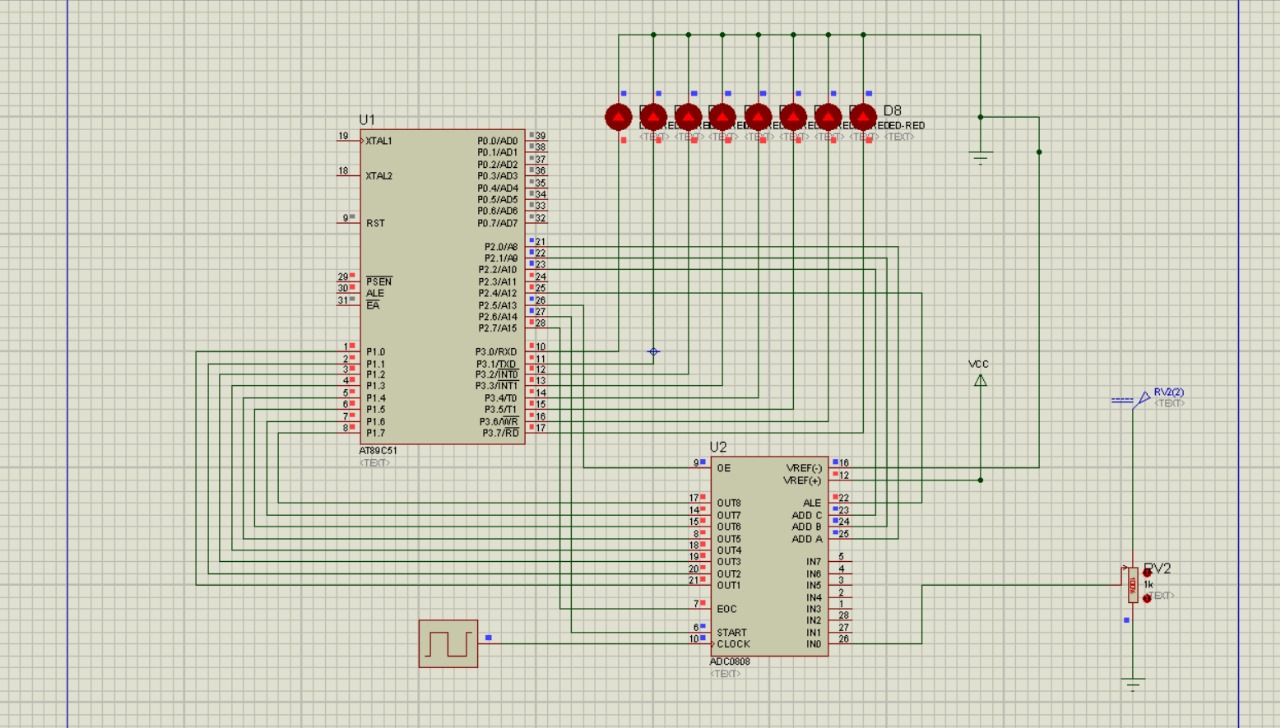
Imagine an analog dial with infinitely many positions between minimum and maximum. An ADC takes a snapshot of the dial's position at a specific moment and translates it into a specific digital value.

**Key Concepts:**

* **Resolution:** This refers to the number of bits used to represent the analog signal's value. A higher resolution ADC (e.g., 12-bit) provides more precise digital values with more steps between minimum and maximum. (Think of having more positions on the dial to capture)
* **Sampling Rate:** This defines how often the ADC takes a sample of the analog signal. A faster sampling rate is necessary for accurately capturing rapidly changing signals. (Imagine taking more frequent snapshots of the dial's movement)
* **Analog Reference Voltage:** This voltage sets the upper limit of the analog signal that the ADC can convert. The digital output will range from 0 (for 0V) to a maximum value (usually corresponding to the reference voltage).

**Code (uploaded to Github)**

**Output:**

****

Week2-Day1: introduction to RaspberryPi

**Learning Objectives: To gain Background knowledge and prerequisites of RaspberryPi**

The trainer explained the evolution of RaspberryPi, how the ideation came into existence, the stages of its development and models, it started from Pi model A, with few features and was upgraded to Pi model B with additional features compared to model A, it was then upgraded to Pi model 2, 3, 4, and presently in Model 5, this evolution evolves its advancement in features on the track, it’s a microprocessor that uses OS (operating system, previously Noobs, now PiOS) to operate modelled in Ubuntu/Linux structure/framework, presently it has a different version of board design from the Pi foundation and different educational Kit designed for hobbies and learning purposes.

Interfaces supported by the board were explained, and how they function compared to other microcontrollers or microprocessors, e.g I2C, SPI, M2HAT, etc., not neglecting its sophisticated features of visualization, connecting to external image visualizers as well as personal visuals, its onboard connectivity and ports functionalities were explained by the trainer, resources to broaden the basic knowledge and fundamentals of RaspberryPi were given and annotated.

Week2-Day2: Installation of RaspberryPi OS and Configuration

**Learning Objectives: Onboarding usage configuration of RaspberryPi board**

To start using the Pi board, as mentioned by the trainer and in the documentation on the process based, the OS(Ubuntu-based or Ubuntu core) has to be installed on the RaspberryPi board, which can be done in two ways:

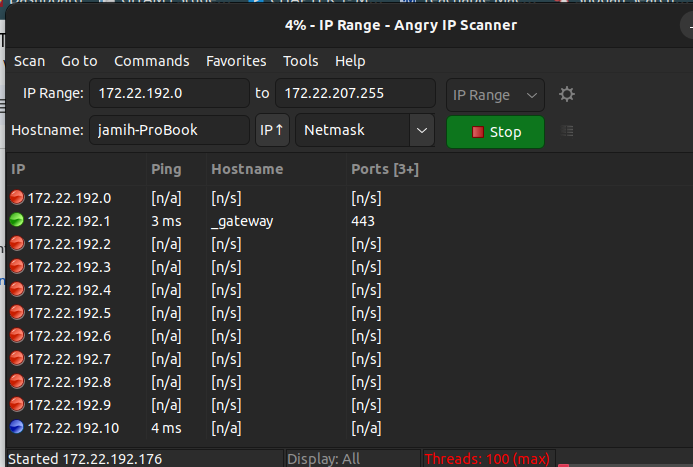
1. Downloading it and uploading it through the Imager
2. Using the imager to install it from the Pi foundation cloud to the mass storage with network connectivity

The steps for the installation and configuration are as follows:

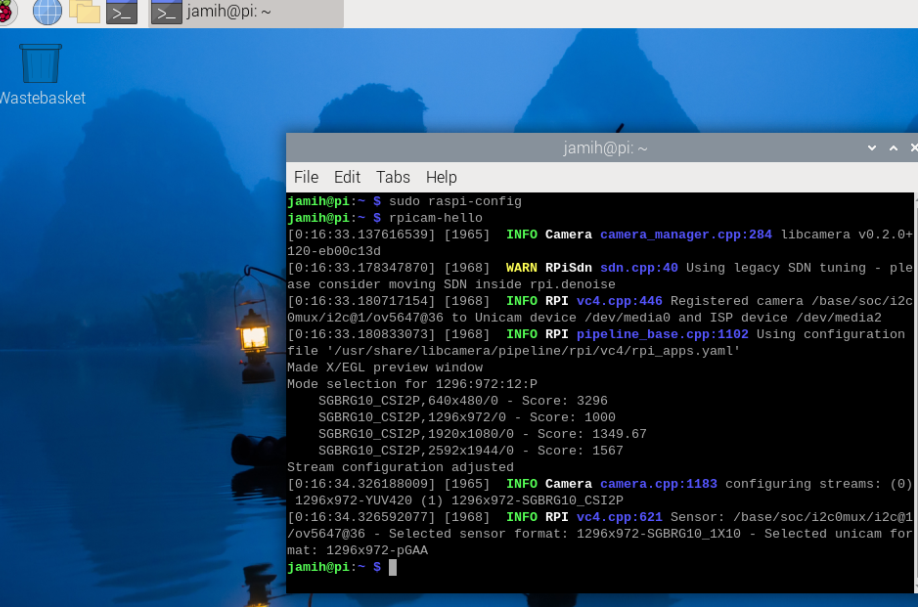
1. Inserting a memory card into a reader, which is to be inserted into the PC.
2. Opening the Imager and selecting the board type, OS version and the storage to load the OS to.
3. Performing the basic settings(device local name, device name, SSH auth, WiFi details, etc) before final approval of loading the OS to the external storage

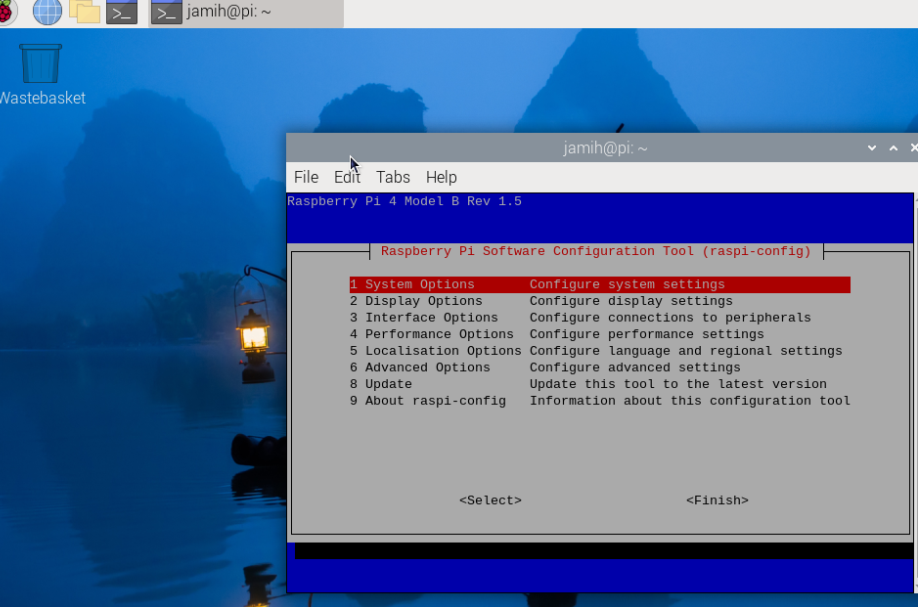


1. Approving the settings and awaiting the full writing and verification process by the imager to the external storage.
2. After the full installation, the memory card was removed from the reader and inserted into the Pi board, then the power was connected to the board and powered on while waiting for the board to bootload the OS in the card and connect to the network provided in the settings,
3. After connectivity confirmation, we then installed an IP address scanner to determine the IP address of the Pi board (this is needed as we ran a headless configuration of the board) called “AngryIPScanner”, provided both the PC and the Pi board are connected to the same network.

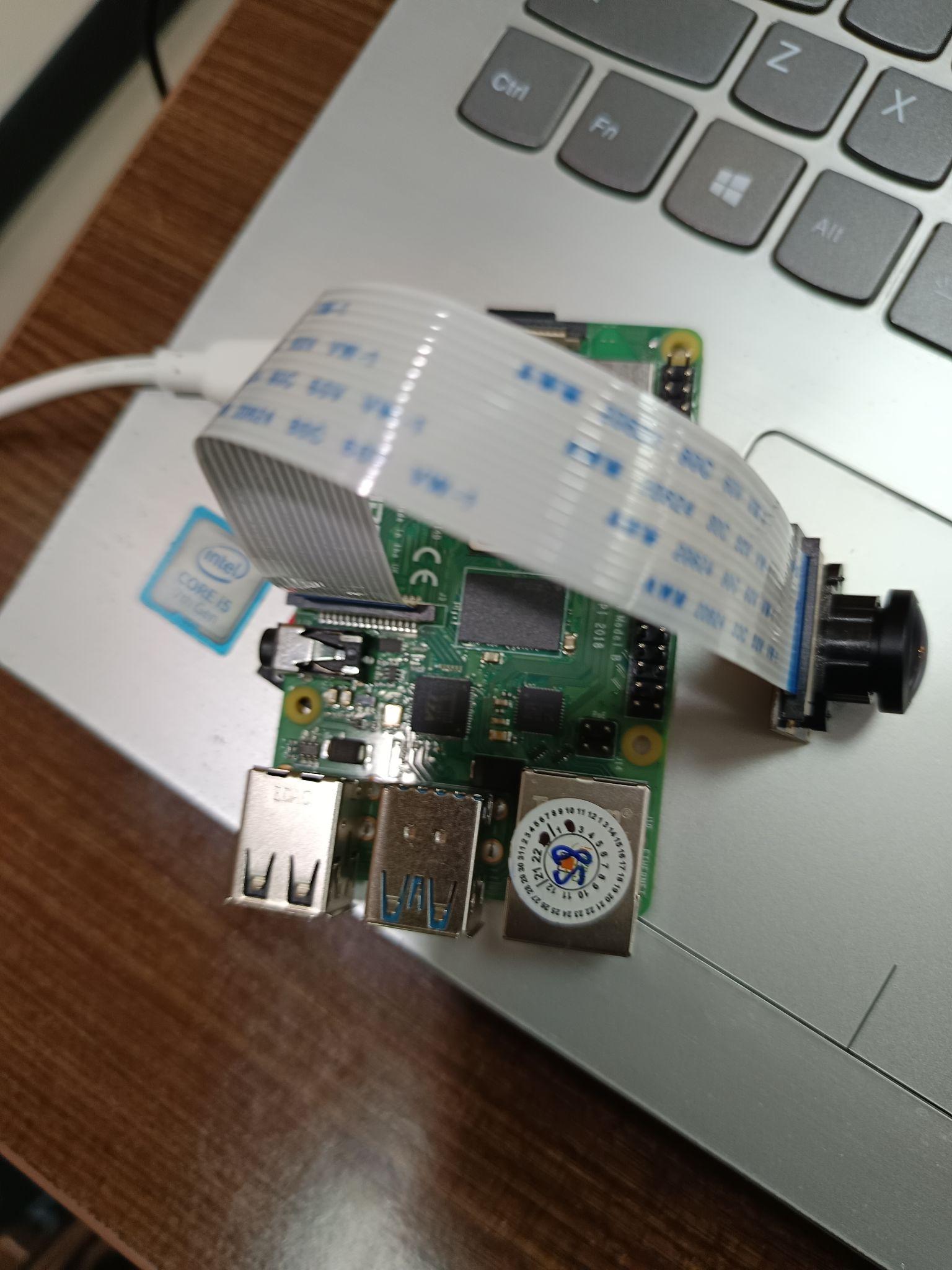


1. After the IP has been determined, we then connect to the board using either way (Termonal for Linux users “ssh localname@ipaddress”, installing Putty for Windows users) with the help of the IP address, authorization name and password was used to finally connect to the board.
2. Once the board is finally connected to, the terminal changes to the board’s interface terminal (for Linux users), or a terminal will pop up from Putty (windows users).
3. After this, “sudo raspi-config” command was used to access the settings interface of the Pi OS in order to enable the remote display interface(VNC) connectivity and other settings.





1. Installation of VNC viewer was done, and using the IP address generated initially, we were able to connect to the Pi and the OS interface was displayed on the VNC viewer.



Once the VNC viewer displays this, the board is ready for use on our PC.

Finally, for the day, we made a simple connection of the Pi-Cam module to the board and verified the simple initialization code for the camera bootup using the command **“rpicam-hello --timeout 0”.**

****

Above is the image taken by the Pi-cam module.