MODULE 3
IOT DESIGN METHODOLOGY

- What is an IoT Device-Basic?
- Building blocks of an IoT Device
- Exemplary Device:
  - Raspberry Pi, About the Board, Linux on Raspberry Pi, Raspberry Pi Interfaces – Serial, SPI, I2C, Programming, Raspberry Pi with Python-Controlling LED with Raspberry Pi ,
  - Interfacing an LED and Switch with Raspberry Pi,
  - Interfacing a Light Sensor (LDR) with Raspberry Pi ,
- Other IoT Devices- pcDuino, Beagle Bone Black, Cubieboard

What is an IoT Device?

- An **IoT device** is any object or "Thing" that:
  - Has a unique identifier.
  - Can send and/or receive data (including user data) over a network.
  - Is connected to the Internet and communicates with other systems.
- **Examples of IoT Devices:**
  - Smartphones, smart TVs, refrigerators, cars, wearables, computers.
- These devices share information about themselves or their environment using sensors or actuators.

Functions of IoT Devices:

- **Data Sensing**: Gather environmental or system data.
- **Communication**: Transmit and receive data via the internet/network.
- **Remote Actuation**: Trigger actions remotely (e.g., turning on lights).
- **Monitoring and Control**: Used for automation and user control.

Design Methodology in the Context of IoT Systems

Category	Example IoT Device	Functionality
Smart Home	Smart Thermostat (e.g., Nest)	Adjusts room temperature based on user habits and environmental data
	Smart Lock	Remote door access control and monitoring
	Smart Light Bulbs (e.g., Philips Hue)	Remote-controlled lighting with scheduling and color customization
Healthcare	Smart Glucose Monitor	Tracks and uploads blood sugar data to cloud for medical review
	Smart ECG Monitor	Sends real-time ECG data to hospitals
	Smart Inhalers	Tracks usage and connects to patient mobile apps
Industrial IoT (IIoT)	Vibration Sensors in Machinery	Detects abnormal conditions for predictive maintenance
	Smart Power Meters	Monitors industrial power usage and provides analytics
Agriculture	Soil Moisture Sensor	Measures soil moisture levels and sends data to central control
	Automated Irrigation System	Controls water valves based on weather and moisture data
Transportation	GPS Trackers on Vehicles	Real-time tracking and route optimization
	Smart Parking Sensors	Detects empty parking spaces and guides drivers
Retail	Smart Shelves	Monitor product quantity and alert when restocking is needed
	Connected Point-of-Sale Devices	Enable inventory tracking and analytics
Environment Monitoring	Air Quality Monitoring Devices (e.g., CO_2, PM2.5 sensors)	Track pollution levels and upload data to cloud for analysis
	Smart Water Quality Sensors	Detect contaminants in water pipelines

Basic Building Blocks of an IoT Device

An IoT device is composed of several **functional modules**, each responsible for a specific task. These modules work together to enable sensing, communication, decision-making, and actuation.

❖ **Sensing**

- Sensors collect information from the environment.
- Can be on-board (integrated into the device) or attached externally.
- Examples: temperature, humidity, light intensity, etc.
- Data is sent to other devices or to cloud-based servers/storage.

❖ **Actuation**

- Responsible for performing actions based on received commands.
- Actuators can turn physical devices on/off, move parts, trigger alerts, etc.
- Example: A relay switch connected to an IoT device can turn on/off a fan or a light.

Basic Building Blocks of an IoT Device

❖ Communication

- Enables data transmission between the IoT device and:
  - Other devices
  - Cloud storage/servers
- Supports receiving control commands from remote applications.
- May include Wi-Fi, Bluetooth, Zigbee, LTE, etc.

❖ Analysis & Processing

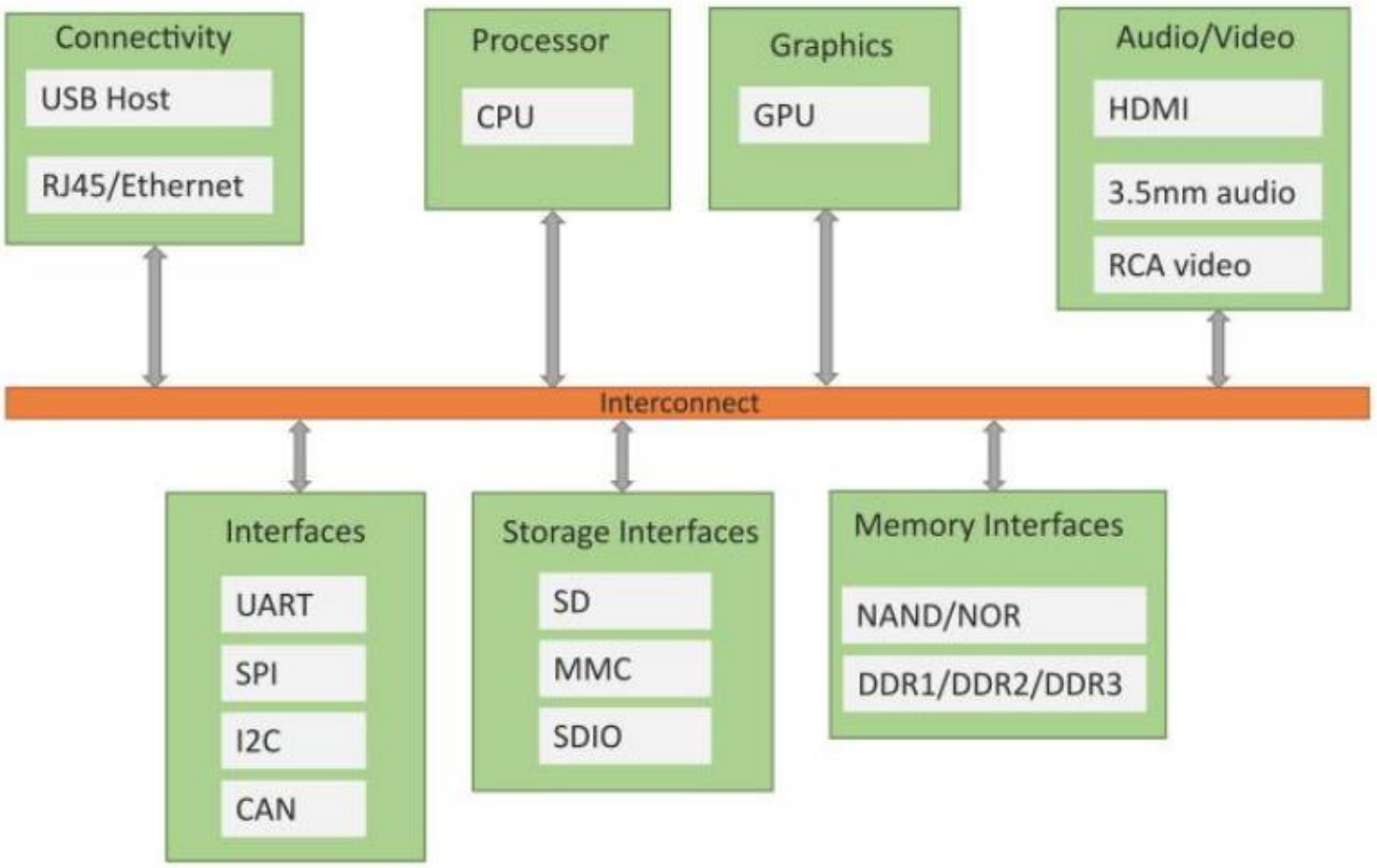
- Processes the collected data to extract useful insights.
- Involves filtering, aggregation, decision-making, etc.
- May be done locally (on the device) or remotely (in the cloud).

Basic Building Blocks of an IoT Device

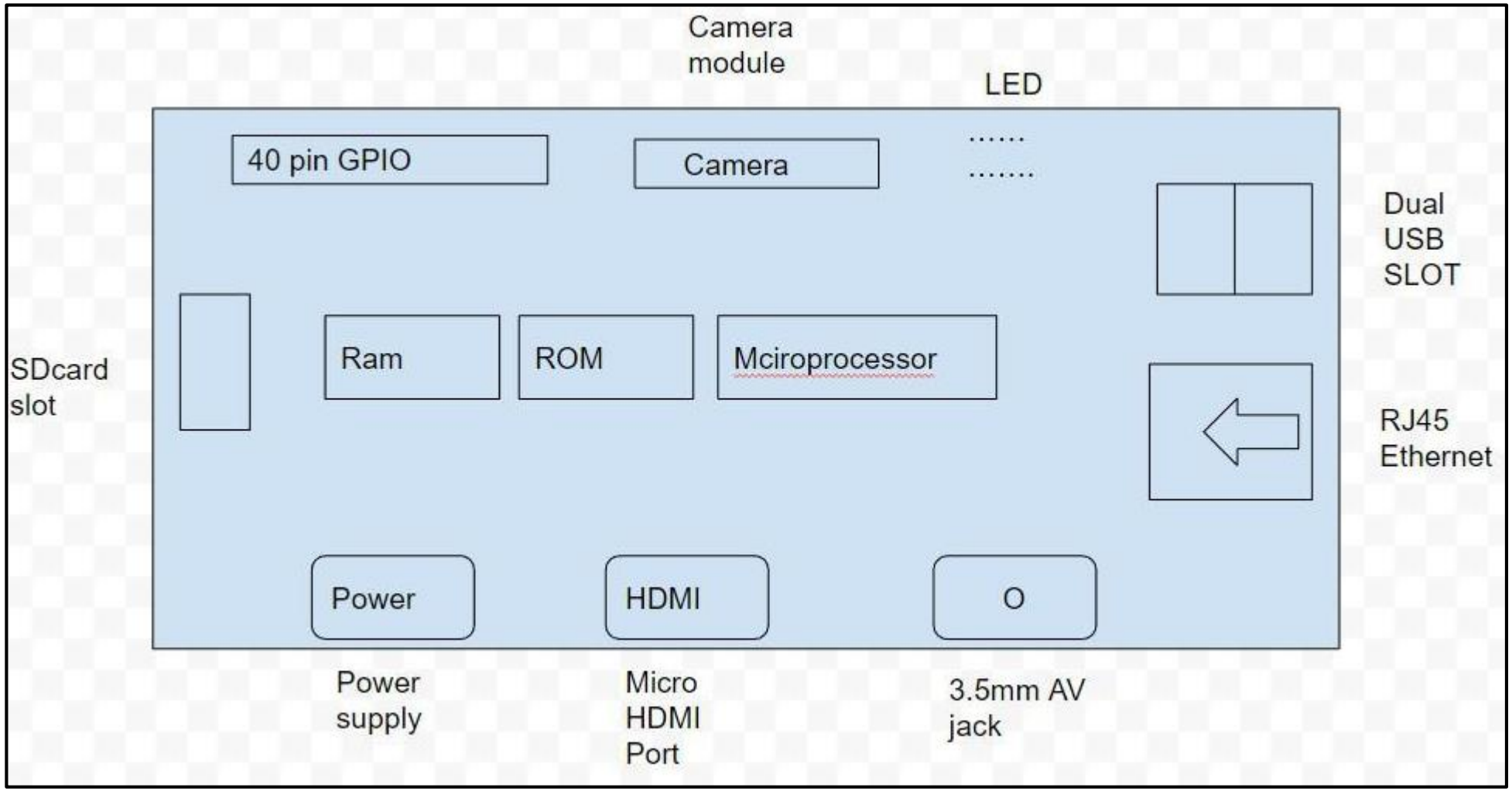
Example IoT Device Platform:

Raspberry Pi

- A widely used single-board computer (SBC).
- Features: CPU, GPU, RAM, storage, multiple interfaces.
- Chosen for its:
  - Low cost
  - Accessibility
  - Strong online community & documentation



The Raspberry Pi Board Block Diagram



The Raspberry Pi Board

- **Processor**
  - Raspberry Pi is built on an ARM processor.
  - Model B (Revision 2) includes a 700 MHz ARM1176JZF-S CPU and 512 MB SDRAM.
- **USB Ports**
  - Two USB 2.0 ports available.
  - Each port provides 100 mA.
  - Devices needing more power require an external powered USB hub.
- **Ethernet Port**
  - Standard RJ45 Ethernet connection.
  - Supports wired internet.
  - USB Wi-Fi adapter can also be used for wireless connectivity.

The Raspberry Pi Board

- **HDMI Output**
  - HDMI port provides audio + video output.
  - Can connect to TVs/monitors with HDMI.
  - For DVI monitors, use HDMI-to-DVI adapter.
- **Composite Video Output**
  - RCA jack provides PAL/NTSC composite video.
  - Suitable for older TVs without HDMI input.
- **Audio Output**
  - 3.5 mm audio jack for analog stereo audio.
  - Works alongside the RCA video output.
- **GPIO Pins**
  - Multiple General Purpose I/O pins.
  - Supports I2C, SPI, UART communication.
  - Used to interface sensors, actuators, and external devices.

The Raspberry Pi Board

- **Display Interface (DSI)**
  - DSI port connects LCD display panels.
- **Camera Interface (CSI)**
  - CSI port connects the Raspberry Pi Camera Module.
- **Status LEDs**
  - Raspberry Pi includes five status LEDs showing power, activity, and network status.
- **SD Card Slot**
  - Used for OS installation and storage.
  - Raspberry Pi requires SD card with Linux OS (NOOBS/Raspbian).
- **Power Input**
  - Powered via micro-USB connector.

Raspberry Pi Supported Operating Systems

Linux on Raspberry Pi

Raspberry Pi supports multiple Linux-based operating systems, each designed for different uses and performance requirements.

1. Raspbian

- A Debian Wheezy–based Linux distribution optimized specifically for Raspberry Pi.
- Official and recommended OS for Raspberry Pi.
- Lightweight and stable, suitable for beginners and general applications.

2. Arch

- A lightweight Arch Linux port for ARM devices.
- Designed for users who prefer a minimal, customizable Linux environment.

3. Pidora

- A Fedora Linux distribution customized for Raspberry Pi.
- Provides a user-friendly Fedora experience on ARM-based systems.

Raspberry Pi Supported Operating Systems

4. RaspBMC

- Linux-based XBMC media center distribution for Raspberry Pi.
- Designed for multimedia applications such as video playback and home media servers.

5. OpenELEC

- A fast and user-friendly XBMC/Kodi media center Linux distribution.
- Extremely lightweight and optimized for media playback.

6. RISC OS

- A very fast, compact, and efficient operating system.
- Unlike Linux-based OSes, RISC OS is designed with simplicity and performance in mind.

Raspberry Pi Interfaces

Raspberry Pi includes multiple hardware interfaces—Serial, SPI, and I^2C—which enable communication with sensors, modules, and peripheral devices.

1. Serial Interface (UART)

The Raspberry Pi has a serial communication interface commonly referred to as UART (Universal Asynchronous Receiver/Transmitter).

- It uses two main pins:
  - **Rx (Receive)** – receives serial data from external devices.
  - **Tx (Transmit)** – sends serial data to external devices.
- UART Used for:
  - Debugging and terminal communication.
  - Interfacing GPS modules, GSM modules, Bluetooth modules, etc.

Communication is asynchronous, meaning no clock wire is used.

Raspberry Pi Interfaces

2. SPI Interface (Serial Peripheral Interface)

SPI is a synchronous serial communication protocol used for high-speed data transfer between Raspberry Pi and one or more peripheral devices.

❖ **Key Features of SPI:**

- Works on master–slave architecture.
- Raspberry Pi functions as the Master.
- Peripherals (e.g., ADCs, sensors, displays) act as Slaves.
- Supports high-speed, full-duplex communication.

❖ **SPI Pins on Raspberry Pi:**

There are five important pins for SPI communication:

- **MISO (Master In Slave Out):** Used to send data from the slave to the master.
- **MOSI (Master Out Slave In):** Used to send data from the master to the slave.
- **SCK (Serial Clock):** Clock signal generated by the master. Synchronizes data transfer between master and slave.

Raspberry Pi Interfaces

- **CE0 (Chip Enable 0):** Also called CS0 (Chip Select 0). Active low signal used to enable/activate the first slave device.
- **CE1 (Chip Enable 1):** Also called CS1 (Chip Select 1). Used to enable/activate the second slave device.

❖ **How SPI Works:**

- Master selects the slave using CE0/CE1.
- Clock (SCK) is sent from master to slave.
- Data is exchanged:
  - Master → Slave using MOSI.
  - Slave → Master using MISO.

❖ SPI supports fast data transfer, making it suitable for displays (OLED, LCD), high-speed ADCs, temperature sensors, SD card modules, etc.

Raspberry Pi Interfaces

3. I^2C Interface (Inter-Integrated Circuit)

I^2C is a **two-wire**, synchronous serial communication protocol designed to connect multiple low-speed peripherals using minimal pin connections.

❖ **I^2C Pins on Raspberry Pi:**

- **SDA (Serial Data Line)** – used for transferring data.
- **SCL (Serial Clock Line)** – used to synchronize data transfer between devices.

❖ **Key Characteristics:**

- Supports multiple devices on the same bus (multi-slave communication).
- Only two wires are used regardless of number of connected devices.
- Devices have unique 7-bit or 10-bit addresses for identification.
- Suitable for:
  - Real-time clocks (RTC modules)
  - Digital sensors (temperature, pressure, environment sensors)

Raspberry Pi Interfaces

- I/O expanders
- Small EEPROMs
- Display modules

❖ **How I^2C Works:**

- Raspberry Pi acts as the Master.
- Communication is synchronized using SCL.
- Data is transferred through SDA, one bit at a time.
- Very useful for low-speed, short-distance communication.

Programming Raspberry Pi with Python

- Raspberry Pi runs on Linux and comes with Python installed by default, making it ideal for beginners and IoT developers. Using Python, you can control the Raspberry Pi's GPIO (General Purpose Input/Output) pins, which allow interaction with sensors, switches, and electronic components.

- Python programs on Raspberry Pi function just like on any regular computer, but the ability to use GPIO pins makes it uniquely suitable for embedded system applications and Internet of Things (IoT) projects.

- Using GPIO along with other interfaces such as SPI, I^2C, and Serial, Raspberry Pi can read sensor data, trigger devices, control motors, send information to servers, or perform automated actions like sending emails or activating a relay.

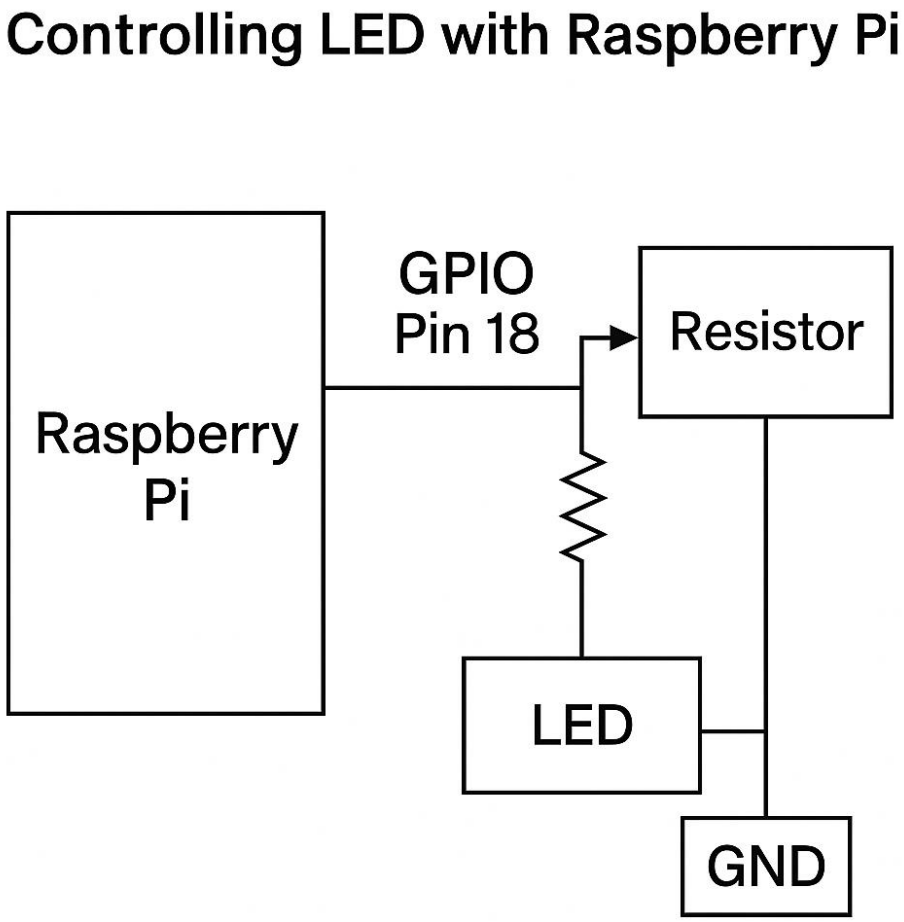
Programming Raspberry Pi with Python

1. Controlling an LED with Raspberry Pi

To understand GPIO usage, we start with a simple example—turning an LED ON and OFF using Python on Raspberry Pi.

Hardware Setup

- An LED is connected to **GPIO pin 18** on the Raspberry Pi.
- A resistor (typically 220Ω–330Ω) is used in series to protect the LED.
- The ground (GND) pin of Raspberry Pi is connected to the negative terminal of the LED.



Programming Raspberry Pi with Python

Controlling LED Using Raspberry Pi Console (Terminal)

You can switch the LED ON/OFF directly from the terminal using Linux file operations on GPIO.

- **Step 1: Export the GPIO pin**

  echo 18 > /sys/class/gpio/export

- **Step 2: Set the pin direction**

  echo out > /sys/class/gpio/gpio18/direction

- **Step 3: Turn LED ON**

  echo 1 > /sys/class/gpio/gpio18/value

- **Step 4: Turn LED OFF**

  echo 0 > /sys/class/gpio/gpio18/value

This method demonstrates how hardware pins can be controlled using simple Linux commands.

Programming Raspberry Pi with Python

Python Program for Blinking an LED

Below is a Python script that blinks the LED connected to **GPIO18** every 1 second:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)          # Use BCM pin numbering
GPIO.setup(18, GPIO.OUT)        # Set pin 18 as output

while True:
    GPIO.output(18, True)       # LED ON
    time.sleep(1)               # Wait 1 second
    GPIO.output(18, False)      # LED OFF
    time.sleep(1)               # Wait 1 second
```

The LED continues to blink until the program is stopped manually.

Programming Raspberry Pi with Python

2. Interfacing an LED and Switch with Raspberry Pi

In this section, we explore a practical example of using Raspberry Pi's GPIO pins to interface both an LED (output device) and a switch (input device). The objective is to toggle the LED ON/OFF when the switch is pressed.

This example demonstrates:

- How to read digital input from a GPIO pin.
- How to control an output device (LED) based on the input.
- How to write simple IoT-style Python programs.

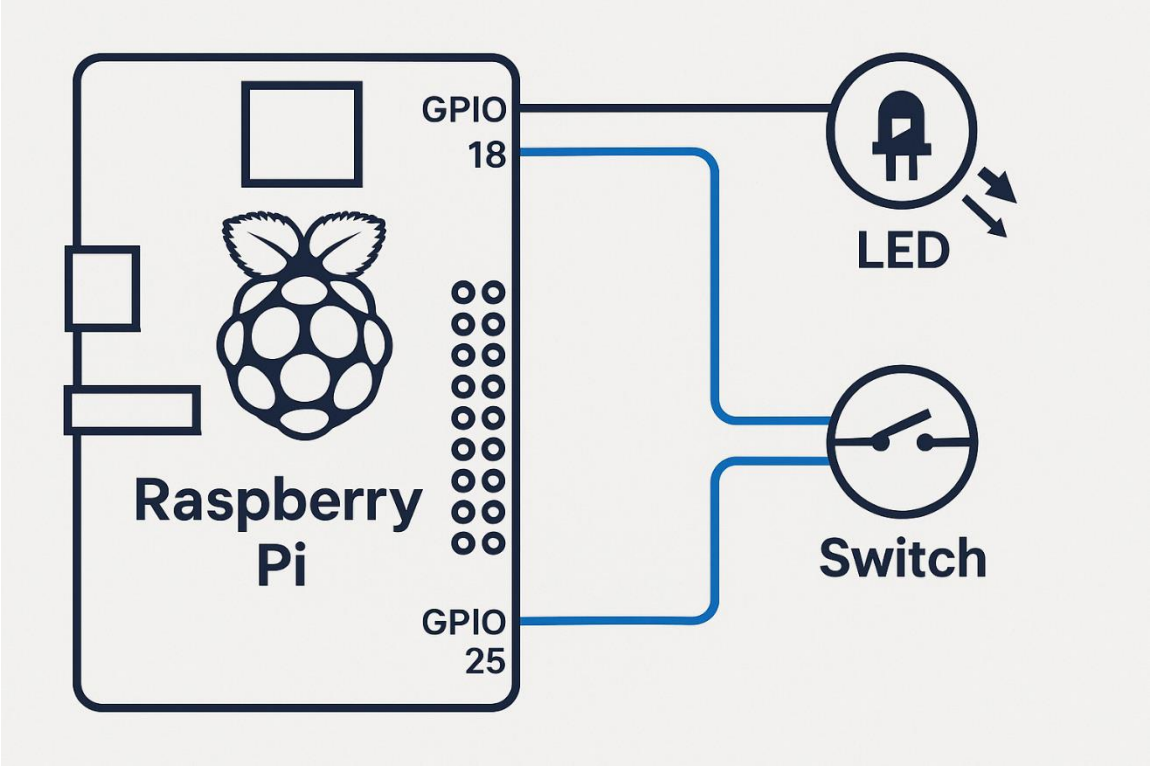
Programming Raspberry Pi with Python

Hardware Explanation

Connections

- LED is connected to GPIO pin 18 (configured as OUTPUT).
- Switch is connected to GPIO pin 25 (configured as INPUT).
- A resistor is used with the LED to prevent excessive current.

When the switch is pressed, Raspberry Pi detects a HIGH signal on GPIO pin 25. This triggers a function to toggle the LED state. Figure 7.10 (shown in your image) displays the wiring diagram on a breadboard and Raspberry Pi.



Programming Raspberry Pi with Python

Python Program: Controlling LED with a Switch

```
from time import sleep
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)     # Use BCM numbering

# Configure switch pin as Input
GPIO.setup(25, GPIO.IN)

# Configure LED pin as Output
GPIO.setup(18, GPIO.OUT)

state = False              # Initial LED state (OFF)
```

Step 1: Importing Libraries

- RPi.GPIO is used to control GPIO pins.
- sleep() adds small delays to improve stability.

Step 2: GPIO Mode Selection

- BCM mode uses the processor pin numbers (not physical board pin numbers).

Step 3: Setting Up Pins

- GPIO 25 is for reading input from the switch.
- GPIO 18 controls the LED.

Step 4: LED State Variable

- This variable keeps track of whether LED is ON or OFF.

Programming Raspberry Pi with Python

Python Program: Controlling LED with a Switch

```
# Function to toggle LED state
def toggleLED(pin):
    global state
    state = not state      # Reverse the current state (ON → OFF or OFF → ON)
    GPIO.output(pin, state)

# Infinite loop to monitor switch input
try:
    while True:
        if GPIO.input(25) == True:    # If switch is pressed
            toggleLED(18)             # Change LED state
            sleep(0.1)                # Small delay to avoid bouncing
except KeyboardInterrupt:
    GPIO.cleanup()                  # Clean exit on Ctrl+C
```

Step 5: Toggle Function

- Each time the function is called, the LED switches to the opposite state.

Step 6: Infinite Loop

- The program checks the input pin continuously.
- When the switch is pressed, LED toggles.
- Delay avoids rapid toggling due to noise or mechanical switch bounce.

Step 7: Graceful Exit

Ensures the program exits safely if the user presses **Ctrl + C**.

Programming Raspberry Pi with Python

3. Interfacing a Light Sensor (LDR) with Raspberry Pi

A Light Dependent Resistor (LDR) is a sensor whose resistance varies based on the amount of light falling on it.

- **More light → Low resistance**
- **Less light → High resistance**

Using an LDR with Raspberry Pi allows us to detect ambient light levels and automatically switch ON/OFF an LED or light.

Hardware Setup Explanation

❖ **Connections**

- One side of the LDR is connected to 3.3V of Raspberry Pi.
- The other side of the LDR is connected to:
  - One leg of a 1 µF capacitor, and
  - A GPIO input pin (GPIO 18).
- The other leg of the capacitor is connected to GND.
- An LED is connected to GPIO pin 25 and controlled based on the LDR reading.

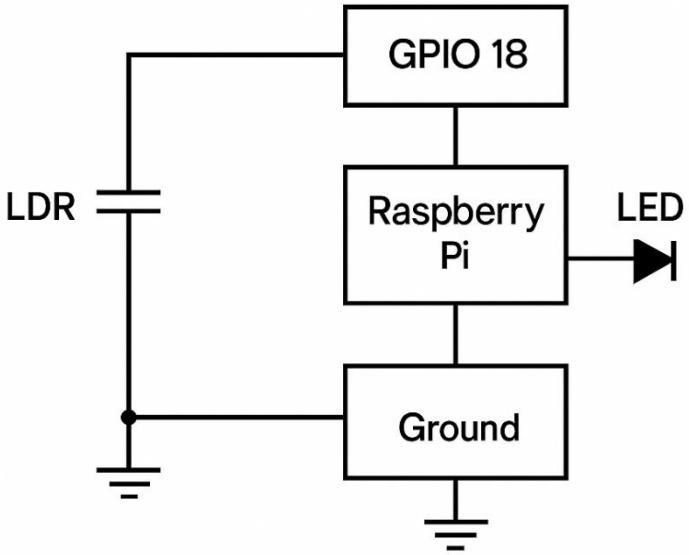
Programming Raspberry Pi with Python

- **Why capacitor is used?**
  - The Raspberry Pi cannot measure analog values directly.

    So, a resistor-capacitor (RC) timing method is used:
    - LDR + capacitor forms an RC circuit.
    - Time taken for the capacitor to charge is inversely proportional to light.
    - Raspberry Pi measures this charging time using digital input polling.

    Thus:
    - **Bright light → LDR resistance low → capacitor charges faster → small count**
    - **Low light → LDR resistance high → capacitor charges slowly → large count**

Interfacing a Light Sensor (LDR) with Raspberry Pi



Programming Raspberry Pi with Python

❖ How the Python Program Works

The Python program repeatedly:

- Discharges the capacitor.
- Measures time taken for the capacitor to charge to HIGH.
- Compares this time with a threshold.
- Turns the LED ON/OFF based on the reading.

```
import RPi.GPIO as GPIO
import time


GPIO.setmode(GPIO.BCM)
ldr_threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25
```

Step 1: Initialize Parameters

- ldr_threshold is the value above/below which light is considered low or high.
- LDR_PIN is used to read light value.
- LIGHT_PIN controls the LED.

Programming Raspberry Pi with Python

```
def readLDR(PIN):
    reading=0
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False)
    time.sleep(0.1)
    GPIO.setup(PIN, GPIO.IN)
```

```
    while(GPIO.input(PIN)==False):
        reading=reading+1
    return reading
```

Step 2: Discharge the capacitor
This forces the capacitor to 0V (discharged).

Step 3: Switch to input mode
Now Raspberry Pi waits for capacitor to charge.

Step 4: Count the time until capacitor charges
- The loop continues until the capacitor voltage reaches a HIGH level.
- More light → fewer counts
- Less light → more counts

Programming Raspberry Pi with Python

```
def switchOnlight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, True)

def switchOfflight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False)

while True:
    ldr_reading = readLDR(LDR_PIN)
    if ldr_reading < ldr_threshold:
        switchOnlight(LIGHT_PIN)
    else:
        switchOfflight(LIGHT_PIN)
    time.sleep(1)
```

Step 5: Functions to Switch Light ON/OFF

Step 6: Logic

- If **LDR reading < threshold** → environment is **bright** → LED ON.
- If **LDR reading > threshold** → environment is **dark** → LED OFF.

This makes the LED work like an **automatic night lamp**.

Other IoT Devices

Raspberry Pi is one of the most popular single-board mini-computers, but there are several alternatives used for IoT applications. These boards also provide GPIO, networking, multimedia interfaces, and support for various operating systems.

Examples of such devices:

- **pcDuino**,
- **BeagleBone Black**, and
- **Cubieboard**.

Other IoT Devices

1. pcDuino

- pcDuino is an Arduino-pin-compatible single-board mini-computer designed for high performance and cost-effectiveness.
- It is powered by a 1 GHz ARM Cortex-A8 processor, enabling it to run full PC-class operating systems such as:
  - **Ubuntu Linux**
  - **Android ICS**
- **Key Features**

- Has HDMI interface for video output, similar to Raspberry Pi.
- Supports multiple programming languages:
  - **C / C++** (using GNU toolchain)
  - **Java** (using Android SDK environment)
  - **Python**
- Provides Arduino-style headers, making it easy to interface with sensors and actuators.
- Suitable for applications where both Arduino compatibility and Linux-level processing are required.

Other IoT Devices

2. BeagleBone Black

BeagleBone Black is another popular single-board computer and is considered a more powerful alternative to Raspberry Pi.

- **Processor & Performance**
  - Powered by a 1 GHz ARM Cortex-A8 processor (similar to pcDuino)
  - Offers higher processing capabilities and more real-time IO features.
- **Operating System Support**
  - Supports both Linux and Android operating systems.
  - Provides easier real-time processing because of its PRU (Programmable Real-Time Units).
- **Key Features**
  - HDMI video/audio output
  - USB and Ethernet ports
  - Extensive GPIO capability
  - Ideal for robotics, automation, and industrial IoT applications

BeagleBone Black is chosen when low-level hardware control and high-speed, deterministic I/O are required.

Other IoT Devices

3. Cubieboard

Cubieboard is a more powerful and versatile IoT-oriented mini-computer compared to traditional boards.

- **Processor**
  - Powered by a dual-core ARM Cortex-A7 processor, providing greater performance.
- **Connectivity and I/O**
  - Cubieboard includes a wide variety of interfaces: USB ports, HDMI, IR (Infrared), Serial port, Ethernet, SATA (for external hard drives), 96-pin extended interface
- **Storage**
  - Provides SATA support, enabling high-speed storage (useful for servers and multimedia systems).
- **Operating Systems**
  - Capable of running: Linux distributions, Android
- **Use Cases**
  - Home media centers
  - Network storage systems
  - IoT gateways
  - Embedded computing projects requiring more storage and processing power