IOT SLIP SOLUTIONS

QUE1: BLINK LEDs

# Import the required modules

import RPi.GPIO as GPIO

import time

# Set up the GPIO channels

GPIO.setwarnings(False) # Disable warnings for GPIO setup

GPIO.setmode(GPIO.BCM) # Use Broadcom chip-specific pin numbers

# Set up pin 4 as an output pin

GPIO.setup(4, GPIO.OUT)

# This loop runs forever

while True:

GPIO.output(4, GPIO.HIGH) # Turn on the LED

time.sleep(3) # Wait for 3 seconds

GPIO.output(4, GPIO.LOW) # Turn off the LED

time.sleep(3) # Wait for 3 seconds

c. **Observations on Input and Output:**

* When the Python program runs, the LED connected to the GPIO pin on the Raspberry Pi board blinks at a frequency of 1 Hz.
* The IR sensor detects the presence of an object and generates a signal output when triggered by infrared radiation.
* The temperature sensor provides an analog or digital output corresponding to the ambient temperature.

d. **Result and Conclusion:**

The Python program successfully blinks the LED connected to the Raspberry Pi GPIO pin at a frequency of 1 Hz. This demonstrates basic GPIO interfacing and programming with Python on the Raspberry Pi platform. Observations on input and output highlight the behavior of the IR sensor and temperature sensor in response to external stimuli. Overall, this project underscores the versatility and ease of use of Raspberry Pi for various embedded systems and IoT applications.

Top of Form

QUE2: TURN ON/OFF BUZZER

import RPi.GPIO as GPIO

import time

# Set up GPIO pins

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM) # Use Broadcom SOC channel numbers

GPIO.setup(17, GPIO.OUT) # Set GPIO pin 17 as an output

try:

while True:

GPIO.output(17, GPIO.HIGH) # Turn buzzer ON

time.sleep(1) # Buzzer stays ON for 1 second

GPIO.output(17, GPIO.LOW) # Turn buzzer OFF

time.sleep(1) # Buzzer stays OFF for 1 second

except KeyboardInterrupt:

GPIO.cleanup() # Clean up GPIO settings if you stop the script manually

b. +---------------------+ +----------------------+

| Raspberry Pi | | IR Sensor |

| | | |

| GPIO Pins |<--->| Signal Out |

| Camera Module | +----------------------+

| Temperature Sensor | |

+---------------------+ |

|

+----------------------+

| Power Supply |

+----------------------+

c. **Observations on Input and Output:**

* When the Python program runs, the GPIO pin connected to the buzzer alternates between high and low states at regular intervals, causing the buzzer to turn on and off accordingly.
* The IR sensor detects the presence of an object and generates a signal output when triggered by infrared radiation.
* The temperature sensor provides an analog or digital output corresponding to the ambient temperature.
* The camera module captures images or video footage as per the programming instructions.

d. **Result and Conclusion:**

The Python program successfully controls the buzzer connected to the Raspberry Pi GPIO pin, turning it on and off at regular intervals. This project demonstrates the basic functionality of interfacing GPIO pins with external hardware components and showcases the versatility of Raspberry Pi for various applications. Observations on input and output highlight the behavior of the IR sensor, temperature sensor, and camera module in response to external stimuli. Overall, the project emphasizes the importance of understanding GPIO interfacing and programming for creating interactive and responsive projects with Raspberry Pi.

QUE3: TOGGLE 2 LEDs

import RPi.GPIO as GPIO

import time

# Set up GPIO pins

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM) # Use Broadcom SOC channel numbering

GPIO.setup(17, GPIO.OUT) # Set GPIO pin 17 as an output for LED 1

GPIO.setup(27, GPIO.OUT) # Set GPIO pin 27 as an output for LED 2

# Initial state for LEDs

GPIO.output(17, GPIO.LOW)

GPIO.output(27, GPIO.LOW)

# Main loop

while True:

GPIO.output(17, GPIO.HIGH) # Turn LED 1 ON

GPIO.output(27, GPIO.LOW) # Turn LED 2 OFF

time.sleep(1) # Wait for 1 second

GPIO.output(17, GPIO.LOW) # Turn LED 1 OFF

GPIO.output(27, GPIO.HIGH) # Turn LED 2 ON

time.sleep(1) # Wait for 1 second

b. +------------------------+ +----------------------+

| Raspberry Pi | | IR Sensor |

| | | |

| GPIO Pins |<--->| Signal Out |

| Camera Module | +----------------------+

| Temperature Sensor |

+------------------------+ |

|

+----------------------+

| Power Supply |

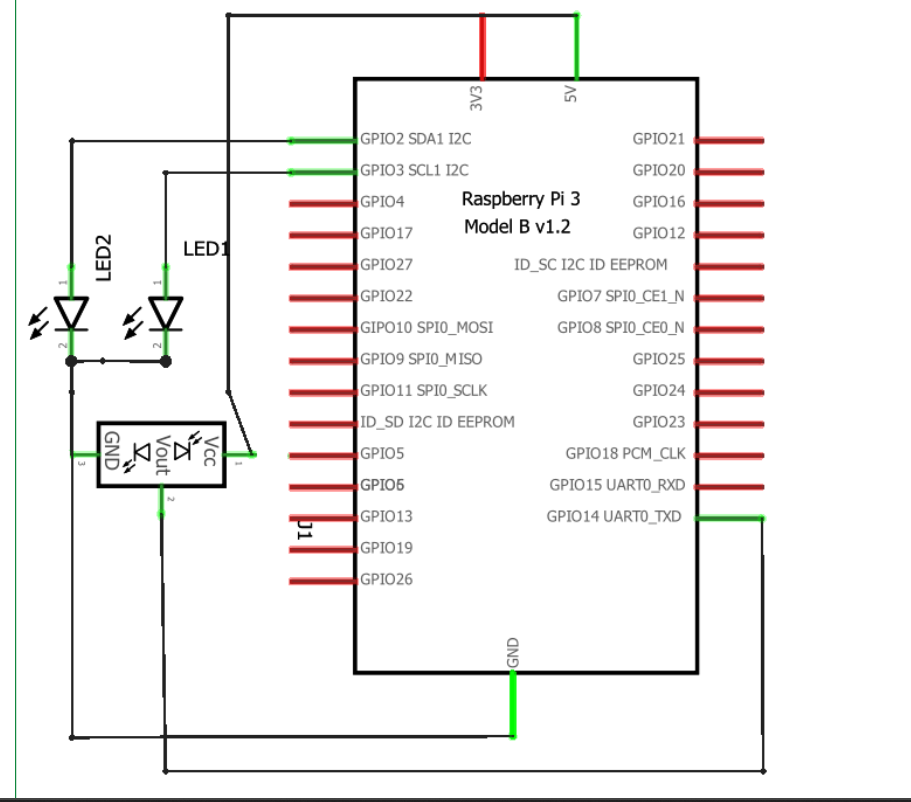
+----------------------+

c. **Observations on Input and Output:**

For the input part, you'll observe the state changes of the sensors (IR sensor detecting motion, temperature sensor providing temperature readings, camera capturing images). For the output part, you'll observe the LEDs toggling on and off based on your program logic.

d. **Result and Conclusion:**

The result would be successful interfacing of sensors/camera with Raspberry Pi and toggling of LEDs using Python/C++. This project demonstrates the basics of hardware interfacing and programming with Raspberry Pi, providing a foundation for building more complex projects in the future.

****

**block diagram of Raspberry-Pi board interfacing with IR sensor**

**Programs given by natasha:**

**Program1: Python program to blink LED**

import RP1. GPIO as GPIO import time

GPIO. setwarnings (False)

GPIO. setmode (GPIO. BCM) #assign numbering for the GPIO using BCM #GPIO. setmode (GPIO. BOARD) #assingn number for the GPIO using Board

# Import Raspberry Pi GPIO library

# Import time module

# Ignore warning for now

cnt = 0

Blink Time = 1

RED LED = 14

GPIO. setup (RED\_LED, GPIO. OUT) while True:

if cnt ==

GPIO.output (RED\_LED, False)

else:

GPIO. output (RED\_LED, True)

ent = 0

time. sleep (Blink\_Time)

GPIO. cleanup ()

Program2: Python program to toggle two LED's.

import RPi.GPIO

as GPIO # Import Raspberry Pi GPIO library

from time import sleep

# Import the sleep

function from the time module

GPIO. setwarnings (False)

# Ignore warning for now

GPIO. setmode (GPIO. BOARD) # Use physical pin numbering

RED LED = 14

GREEN LED = 15

GPIO. setup (RED\_LED, GPIO.OUT, initial=GPIO. LOW) GPIO. setup (GREEN\_LED, GPIO.OUT, initial=GPIO.LOW)

while

True:

\* Run forever

GPIO. output (RED\_LED, True)

# Turn ON

GPIO.output (GREEN\_LED, False) # Turn OFF

sleep (1) # Sleep for 1 second

GPIO.output (RED\_LED, False) # Turn OFF GPIO. output (GREEN\_LED, True) # Turn ON

sleep (1) # Sleep for 1 second

Program3: Python program to turn ON/OFF buzzer.

import Pi,GPIO as GPIO # Import Raspberry Pi GPIO library from time import sleep

# Import the sleep function from the time module

GPIO. setwarnings (False)

# Ignore warning for now

GPIO. setmode (GPIO. BOARD) # Use physical pin numbering

Buzzer = 18

GPIO. setup (Buzzer, GPIO.OUT, initial=GPIO. LOW)

while True:

\* Run forever

GPIO.output (Buzzer, True) # Turn ON sleep (1) # Sleep for 1 second GPIO.output (Buzzer,

False) # Turn OFF

Steps to create new project in proteus:

STEP 1 :- Open Proteus Software STEP 2 :- Select “ New Project “ Option , and click next option till this slide…STEP 3 :-Select “Create Firmware Project”

STEP 4 :-Select “Raspberri pi” in Family Section and click next . STEP 5 :- You Will See this Interface“Schematic Capture” for Circuit Design and “Source Code” for coding.STEP 6: After opening Schemaic Capture and Source code window ,You will see one circuit in Schematic Capture ,this is IC of Raspberry pi in software . Write the python code for LED Interfacing With Raspberry pi in Source code section and for initialize the LED in it …do following Steps. Right Click on RPI3 and Select Add PeripheralsSelect Breakout Peripherals in CategorySelect LED of any colour in Breakout Peripherals

Go viva

Go language : Go was designed at google in 2007 . The main objective of go language is to improve programming productivity in era of multi core. Develop by Robert Griesemer, rob pike, ken Thompson.

Constant : they are like variable with const keyword.

Variables : the var statement declares a list of variables with the type declared last.

Boolean : it have value either true or false, and defined as bool when declaring it as data type.

Pointer and address : pointer is a variable which stores memory address of another variable.

Pointer to pointer : user can form chain of pointers to pointer concept.

Strings : a string sequence of characters with a definite length used to represent text.

New function: another way of accessing pointers in go built in new function.

Switch : a switch statement allow a variable to be tested for equality against a list of values.

If statement :

Syntax : if(boolean\_ expression)

{

/ statement

}

If elseif if else syntax :

If( boolean \_ expression)

{ / execute

} elseif ()

{ /execute

} else ()

{

/ execute

}

For loop syntax : for [ condition |( init ; condition ;increment) |range]

{

Statement

}

If condition is available then for loop executes as long as condition is true

.if range is available then the loop executes for each item in the range.

Function: are the building block of program. Func is a keyword used to create a function.

Call by value : it passing argument to a function copies the actual value of an argument into the formal parameter of the function.

Call by reference : it passing arguments rona function copies the address of an argument into formal parameter.

Recursion : is the process in which function callls itself directly or indirectly.

Arrays : one of the most popular data structure. Used to simple and easy to understand and very flexible can stores many kinds of data.

Slices : it is a flexible and extensible data structure to implement and manage collection of data.

Methods : are similar to go function with one different ie the method contains a receiver argument.

Interface: interface type defines the behavior of the other type by specifying a set of methods that need to be implemented.

Type assertion : a type assertion is the x(T) notation where x is of interface type and T is a type.

Go routines: each concurrently executing method or function is called a go routines.

Go channel : a channel is a medium through which a go routines communicates with another go routine and communication is lock free.

Packages : are the most powerful part. Is to design and maintain a large number of programs by grouping related features together into single unit.

Xml file : the notes. Xml is read with the ioutil. Readfile () function and a byte slice is returned.

Iot

Raspberry Pi : is a series of small single board computers develop in a United Kingdom by the foundation to promote the teaching basic computer science in school.

Arduino : is a collection of three things. Hardware prototype platform, arduino language and ide, boards and microcontroller.

LED: a light emitting diode is a two lead semiconductor light source.

Buzzer : is an arduino signalling device, which may be mechanical or electro mechanical.

Raspberry Pi: Raspberry Pi is a small, affordable, single-board computer developed by the Raspberry Pi Foundation. It is widely used for various applications, including education, DIY projects, and IoT, due to its versatility and GPIO pins for interfacing with external hardware.

Arduino Uno board: Arduino Uno is a popular microcontroller board based on the ATmega328P chip. It is widely used for prototyping and DIY projects due to its simplicity, ease of use, and large community support. Arduino boards are programmable using the Arduino IDE and can interface with various sensors and actuators.

IR Sensor: IR (Infrared) Sensor is a device that detects infrared radiation emitted by objects. It consists of an IR transmitter and receiver pair. When an object comes within the detection range of the sensor, it reflects infrared radiation back to the receiver, triggering a signal.

Temperature sensor: A temperature sensor is a device that measures ambient temperature and provides an output signal proportional to the temperature. Common types of temperature sensors include thermistors, thermocouples, and digital temperature sensors such as DS18B20.

LED: LED (Light Emitting Diode) is a semiconductor device that emits light when an electric current passes through it. LEDs are widely used as indicators, displays, and lighting sources in various electronic devices and IoT applications.

Buzzer sensor: A buzzer sensor is an electronic component that generates sound when an electric current is applied to it. It is commonly used for audible alerts, notifications, and alarms in IoT applications.

Methods of IoT: Methods of IoT include sensor data collection, data processing and analysis, communication protocols for data transmission, cloud computing for storage and processing, and application development for IoT solutions.

Layers of IoT protocols used in IoT: The layers of IoT protocols include physical layer (e.g., wireless communication protocols), data link layer (e.g., Ethernet, Wi-Fi), network layer (e.g., IP, Zigbee), transport layer (e.g., TCP, UDP), and application layer (e.g., MQTT, CoAP).

Embedded system: An embedded system is a computer system designed to perform specific functions within a larger system. It typically consists of hardware components, such as microcontrollers or microprocessors, and software tailored to the application’s requirements.

Trends in IoT: Trends in IoT include the proliferation of connected devices (IoT devices), adoption of edge computing for real-time data processing, integration of AI and machine learning for data analytics, and focus on security and privacy in IoT deployments.

M2M (Machine-to-Machine): M2M refers to communication between devices or machines without human intervention. It enables devices to exchange data and control signals to perform tasks autonomously.

WSN (Wireless Sensor Network): WSN is a network of spatially distributed autonomous sensors that monitor physical or environmental conditions and communicate wirelessly to a central location for data processing and analysis.

SCADA (Supervisory Control and Data Acquisition): SCADA is a system for remote monitoring and control of industrial processes, infrastructure, and facilities. It consists of sensors, controllers, and a central host computer for data collection and visualization.

RFID (Radio Frequency Identification): RFID is a technology that uses radio waves to identify and track objects or people. It consists of RFID tags attached to objects and RFID readers to read the tags and collect data.

Features of IoT: Features of IoT include connectivity (wireless and wired), sensor data collection, real-time monitoring and control, data analytics and insights, remote management, and interoperability between devices and systems.

Security challenges in IoT: Security challenges in IoT include data privacy, authentication and authorization, secure communication, firmware and software vulnerabilities, device management, and protection against cyber-attacks and intrusions.

Sensors and actuators in IoT: Sensors collect data from the physical environment, such as temperature, humidity, and motion, while actuators control physical processes or devices based on input from sensors, such as turning on/off lights or adjusting temperature.

Basic building blocks of IoT (components): Basic building blocks of IoT include sensors, actuators, microcontrollers or microprocessors, communication modules (e.g., Wi-Fi, Bluetooth), power sources (e.g., batteries, solar panels), and software platforms for data processing and application development.

Modbus protocol: Modbus is a communication protocol commonly used in industrial automation systems for transmitting data between electronic devices over serial communication lines.

Zigbee architecture: Zigbee is a wireless communication protocol based on IEEE 802.15.4 standard for low-power, low-data-rate, and short-range communication between devices in IoT and home automation applications.

IP protocols in IoT: IP (Internet Protocol) protocols used in IoT include IPv4 and IPv6 for addressing and routing data packets over the Internet, as well as application layer protocols such as HTTP, MQTT, CoAP, and AMQP for data communication between IoT devices and servers.HTTP (Hypertext Transfer Protocol): HTTP is a protocol used for communication between web browsers and servers. It is primarily used for transmitting hypertext documents over the Internet. In IoT, HTTP is often used for communication between IoT devices and web servers to transfer data and commands.

MQTT (Message Queuing Telemetry Transport): MQTT is a lightweight messaging protocol designed for low-bandwidth, high-latency, or unreliable networks. It is commonly used in IoT applications for communication between devices and servers, particularly in scenarios where network connectivity may be intermittent or bandwidth is limited.

CoAP (Constrained Application Protocol): CoAP is a specialized web transfer protocol designed for constrained devices and constrained networks, such as IoT devices with limited processing power and memory. It is optimized for resource-constrained environments and supports efficient communication and data transfer.

AMQP (Advanced Message Queuing Protocol): AMQP is an open-standard messaging protocol designed for reliable message queuing and exchange between applications or devices. It is commonly used in IoT applications for interoperable and asynchronous communication between components of the IoT ecosystem.

TCP (Transmission Control Protocol): TCP is a connection-oriented protocol that provides reliable, ordered, and error-checked delivery of data packets over a network. It establishes a virtual connection between devices and ensures that data is delivered without loss or corruption.

UDP (User Datagram Protocol): UDP is a connectionless protocol that provides unreliable, unordered, and lightweight data transmission over a network. It is commonly used in applications where real-time data transfer is critical, such as streaming media, online gaming, and VoIP.

IPv6 and IPv4 difference: IPv6 (Internet Protocol version 6) and IPv4 (Internet Protocol version 4) are both protocols used for identifying and addressing devices on a network. The main difference between IPv6 and IPv4 is their address format. IPv6 uses 128-bit addresses, allowing for a significantly larger number of unique addresses compared to IPv4, which uses 32-bit addresses. IPv6 also includes built-in features for security and quality of service, whereas IPv4 requires additional protocols for these functionalities.

Layers in IoT with their function: The layers in IoT typically include:

Physical layer: Handles physical connectivity between devices, such as wireless or wired connections.

Data link layer: Manages data transfer between adjacent devices over a physical medium, ensuring reliable transmission and error detection.

Network layer: Routes data packets between devices in a network, handling addressing, routing, and packet forwarding.

Transport layer: Provides end-to-end communication between devices, ensuring reliable and ordered delivery of data packets.

Application layer: Enables communication between IoT devices and applications, supporting protocols for data exchange, device management, and application integration.